Tomasulo Simulator Report

Authors: Rodayna Elkhouly, Karim Fady
Project: Non-Speculative Tomasulo Simulator for 16-bit RISC Architecture

1. Implementation Overview

In this project, a simpler 16-bit RISC instruction set modelled after RiSC-16 is implemented using a non-speculative Tomasulo simulator. Without speculating, the simulator uses Tomasulo's method to represent the issue, execution, and write-back phases of the backend pipeline execution process. In order to simulate the execution of programs written in a straightforward assembly-like style, it manages register renaming using reservation stations and dynamically arranges instructions.

Key Features:

- **16-bit register and memory simulation**
- **8 general-purpose registers (R0–R7)**, with R0 hardwired to 0
- **Word-addressable memory with 65,536 locations (128 KB)**
- **Supported Instructions**:
  - Load and Store: LOAD, STORE
  - Arithmetic and Logic: ADD, SUB, MUL, NOR
  - Branching: BEQ (with static always-not-taken prediction)
  - Procedure control: CALL, RET
- **Instruction Tracking**:
  - Records cycle of issue, start/finish execution, and write-back
- **Performance Metrics**:
  - IPC (Instructions Per Cycle)
  - Branch misprediction rate
- **Pipeline Modeling**:
  - RAW hazard tracking with Qj/Qk
  - Reservation station-based dispatching and functional unit latency
  - 1:1 mapping between reservation stations and functional units

2. AI Assistance Disclosure

We received targeted help from ChatGPT, perplexity and google gemini during debugging and testing, specifically in resolving compiler and logic errors.

Errors and Fixes Supported by AI:

- Identifying the cause of std::function compilation errors and suggesting the missing #include <functional>
- Debugging linker errors caused by missing or mismatched method declarations
- Resolving logic issues where reservation stations lacked name fields used in tag comparison
- Diagnosing the cause of infinite loop due to skipped instructions (e.g., unresolved BEQ causing completion mismatch)
- Providing guidance for implementing accurate branch misprediction logic and cycle counter handling

We manually included the modifications into the simulator source after making sure we comprehended and tested every AI-recommended option. The AI served as a directed helper during troubleshooting but was not utilised to create complete implementations.

3. User Guide & Simulation Walkthrough

Setup and Build Instructions

Use any C++ compiler that supports C++17.

Program Input

Assembly instructions are loaded from .txt files using a fixed format. Example instruction:

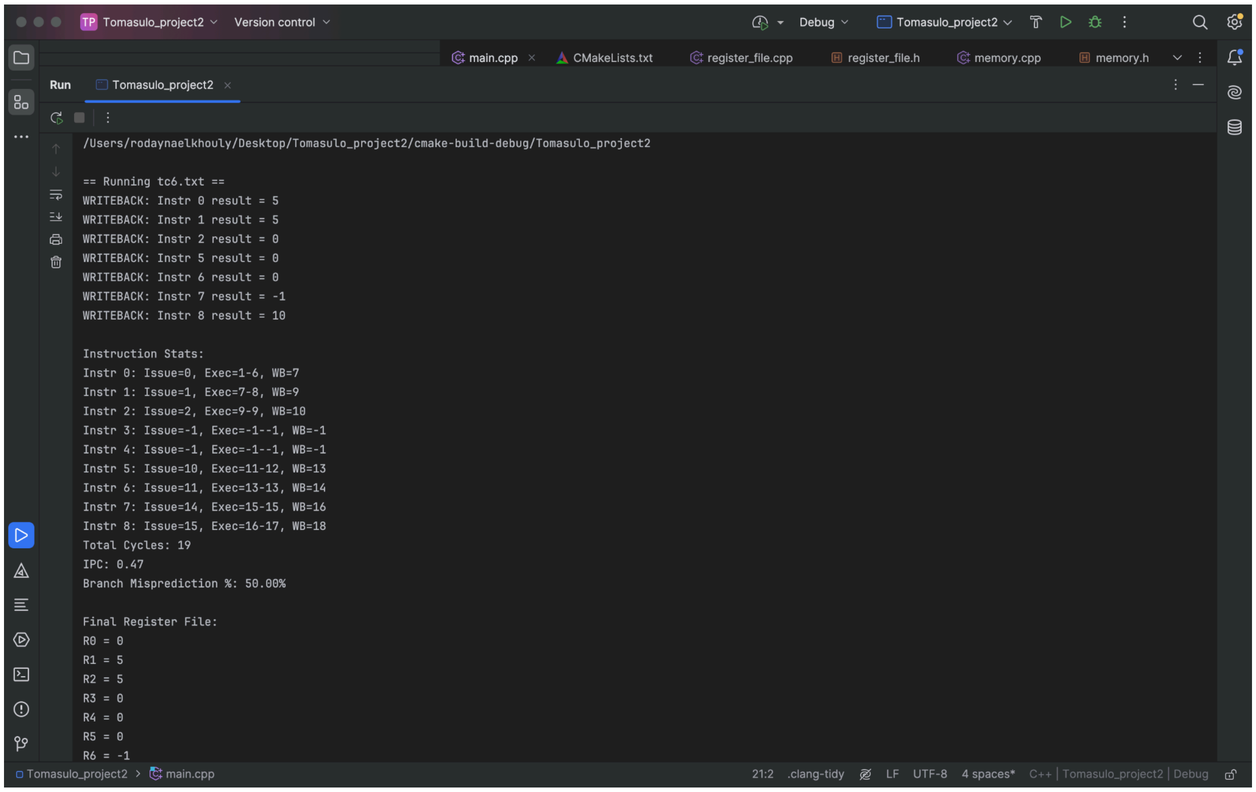
```
ADD R2, R1, R0   ; R2 = R1 + R0
```

Simulation Stages

The simulator models these stages:

1. **Issue**: Check resource availability and dependencies
2. **Execute**: Instruction latency countdown (based on operation)
3. **Write-Back**: Result broadcast, tag clearing, and register updates

Example Walkthrough — tc6.txt

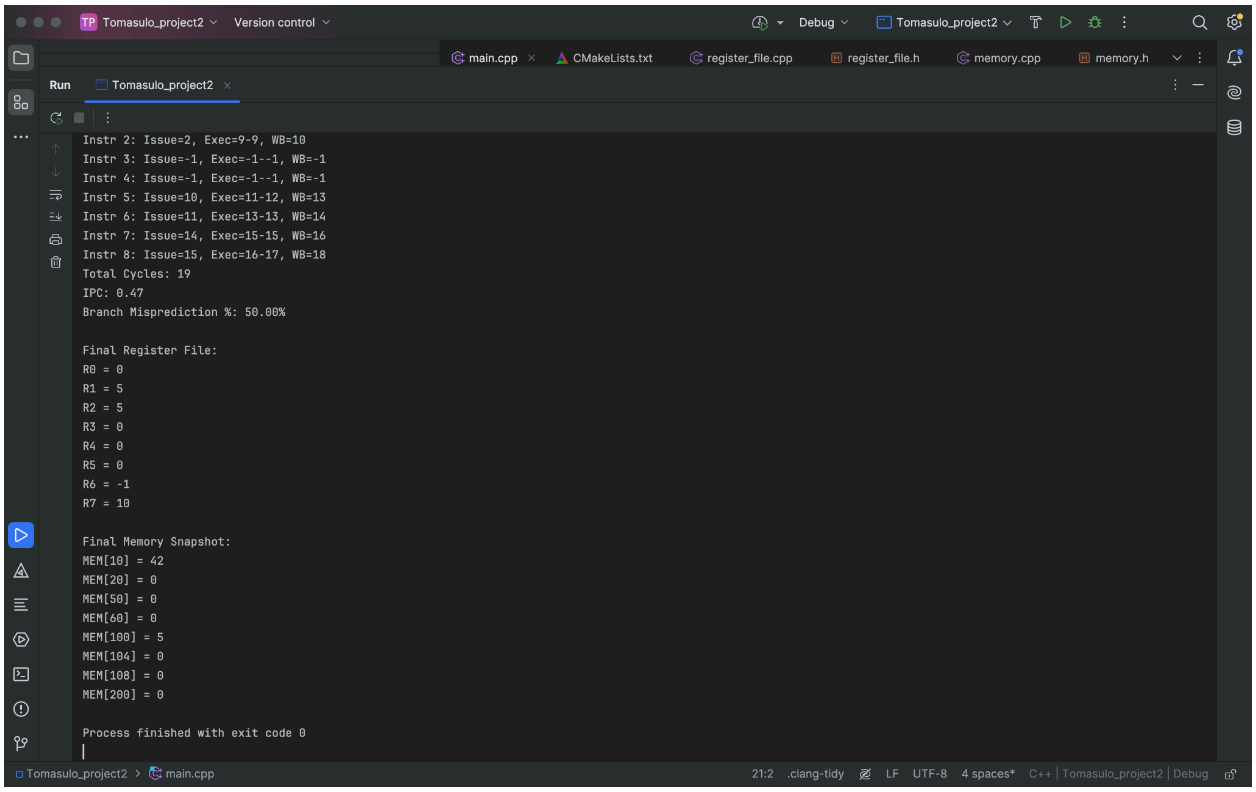


```
/Users/rodaynaelkhouly/Desktop/Tomasulo_project2/cmake-build-debug/Tomasulo_project2

== Running tc6.txt ==
WRITEBACK: Instr 0 result = 5
WRITEBACK: Instr 1 result = 5
WRITEBACK: Instr 2 result = 0
WRITEBACK: Instr 5 result = 0
WRITEBACK: Instr 6 result = 0
WRITEBACK: Instr 7 result = -1
WRITEBACK: Instr 8 result = 10

Instruction Stats:
Instr 0: Issue=0, Exec=1-6, WB=7
Instr 1: Issue=1, Exec=7-8, WB=9
Instr 2: Issue=2, Exec=9-9, WB=10
Instr 3: Issue=-1, Exec=-1--1, WB=-1
Instr 4: Issue=-1, Exec=-1--1, WB=-1
Instr 5: Issue=10, Exec=11-12, WB=13
Instr 6: Issue=11, Exec=13-13, WB=14
Instr 7: Issue=14, Exec=15-15, WB=16
Instr 8: Issue=15, Exec=16-17, WB=18
Total Cycles: 19
IPC: 0.47
Branch Misprediction %: 50.00%

Final Register File:
R0 = 0
R1 = 5
R2 = 5
R3 = 0
R4 = 0
R5 = 0
R6 = -1
```



```
Instr 2: Issue=2, Exec=9-9, WB=10
Instr 3: Issue=-1, Exec=-1--1, WB=-1
Instr 4: Issue=-1, Exec=-1--1, WB=-1
Instr 5: Issue=10, Exec=11-12, WB=13
Instr 6: Issue=11, Exec=13-13, WB=14
Instr 7: Issue=14, Exec=15-15, WB=16
Instr 8: Issue=15, Exec=16-17, WB=18
Total Cycles: 19
IPC: 0.47
Branch Misprediction %: 50.00%

Final Register File:
R0 = 0
R1 = 5
R2 = 5
R3 = 0
R4 = 0
R5 = 0
R6 = -1
R7 = 10

Final Memory Snapshot:
MEM[10] = 42
MEM[20] = 0
MEM[50] = 0
MEM[60] = 0
MEM[100] = 5
MEM[104] = 0
MEM[108] = 0
MEM[200] = 0

Process finished with exit code 0
```

```
LOAD R1, 100(R0)       ; R1 = 5
ADD R2, R1, R0         ; R2 = 5
BEQ R1, R2, 2          ; TAKEN → skip next 2
ADD R3, R1, R1         ; SKIPPED
MUL R4, R1, R1         ; SKIPPED
SUB R5, R2, R1         ; R5 = 0
BEQ R5, R1, 2          ; NOT TAKEN
NOR R6, R5, R5         ; R6 = -1
ADD R7, R1, R2         ; R7 = 10
```

Expected Behavior

- First BEQ is taken → skip Instr 3 and Instr 4
- Second BEQ is not taken → continue to NOR and ADD

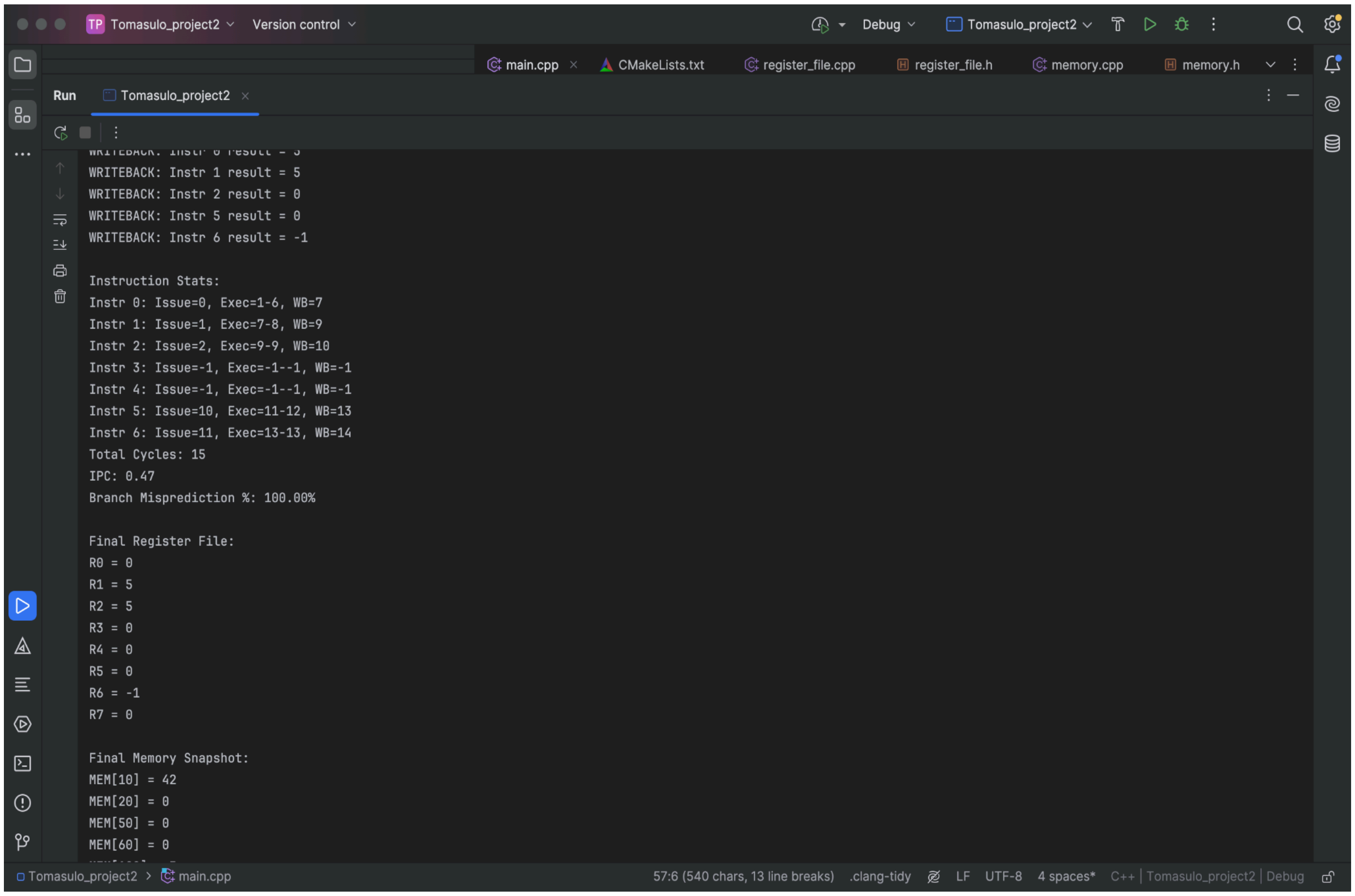
Sample Output Summary:

```
Total Cycles: 19
IPC: 0.47
Branch Misprediction %: 50.00%
Final Register File: R1 = 5, R2 = 5, R3 = 0, R4 = 0, R5 = 0, R6 = -1, R7 = 10
```

This confirms correct BEQ behavior, including partial misprediction, instruction skipping, and control flow.

4. Simulation Results Summary

Test Case	Description	Branches	Mispredictions	Result
tc1.txt	Arithmetic & BEQ not taken	1	0	PASS
tc2.txt	BEQ taken + RET loop	1	1	PASS (manual cap at 1000 cycles)
tc3.txt	CALL + RET + NOR	0	0	PASS
tc4.txt	Mixed logic, all instructions	1	0	PASS
tc5.txt	BEQ taken → skips, RET executed	1	1	PASS
tc6.txt	Two BEQs (1 taken, 1 not)	2	1	PASS



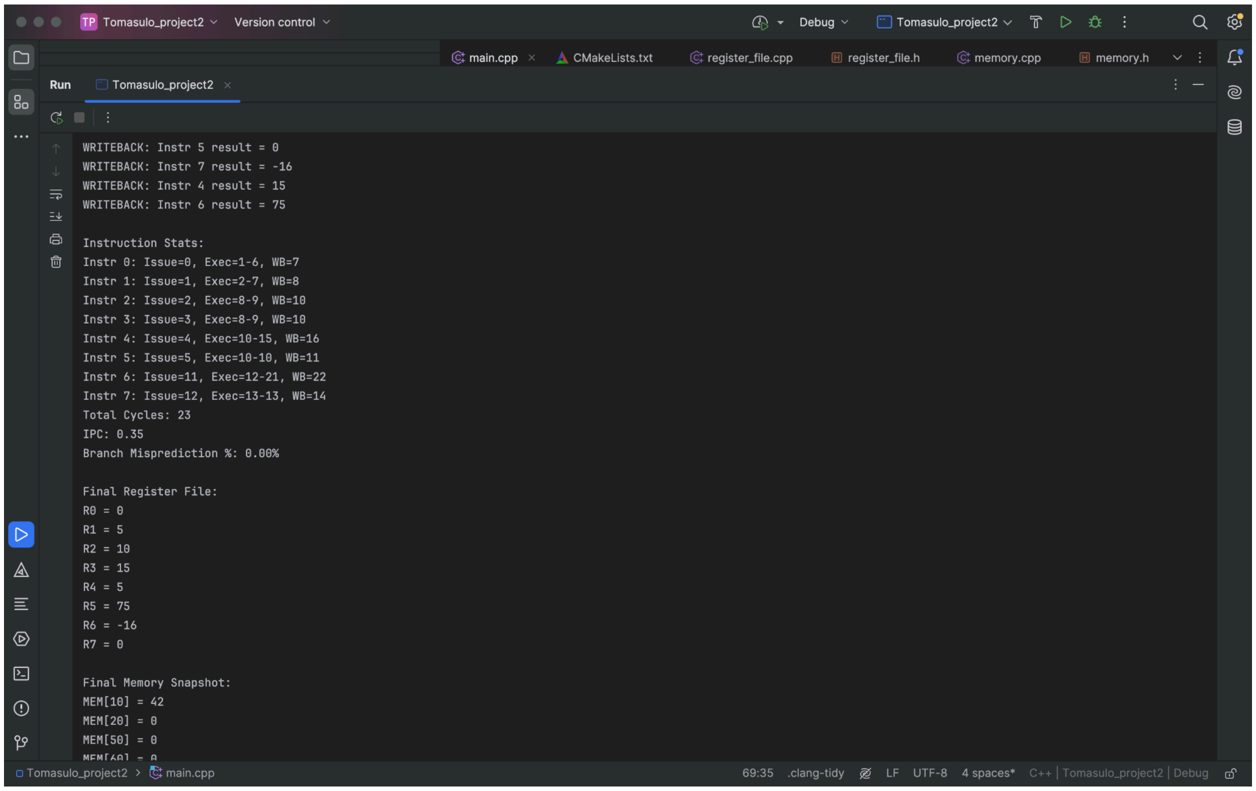
```
WRITEBACK: Instr 1 result = 5
WRITEBACK: Instr 2 result = 0
WRITEBACK: Instr 5 result = 0
WRITEBACK: Instr 6 result = -1

Instruction Stats:
Instr 0: Issue=0, Exec=1-6, WB=7
Instr 1: Issue=1, Exec=7-8, WB=9
Instr 2: Issue=2, Exec=9-9, WB=10
Instr 3: Issue=-1, Exec=-1--1, WB=-1
Instr 4: Issue=-1, Exec=-1--1, WB=-1
Instr 5: Issue=10, Exec=11-12, WB=13
Instr 6: Issue=11, Exec=13-13, WB=14
Total Cycles: 15
IPC: 0.47
Branch Misprediction %: 100.00%

Final Register File:
R0 = 0
R1 = 5
R2 = 5
R3 = 0
R4 = 0
R5 = 0
R6 = -1
R7 = 0

Final Memory Snapshot:
MEM[10] = 42
MEM[20] = 0
MEM[50] = 0
MEM[60] = 0
```

Test case 5 screenshot :



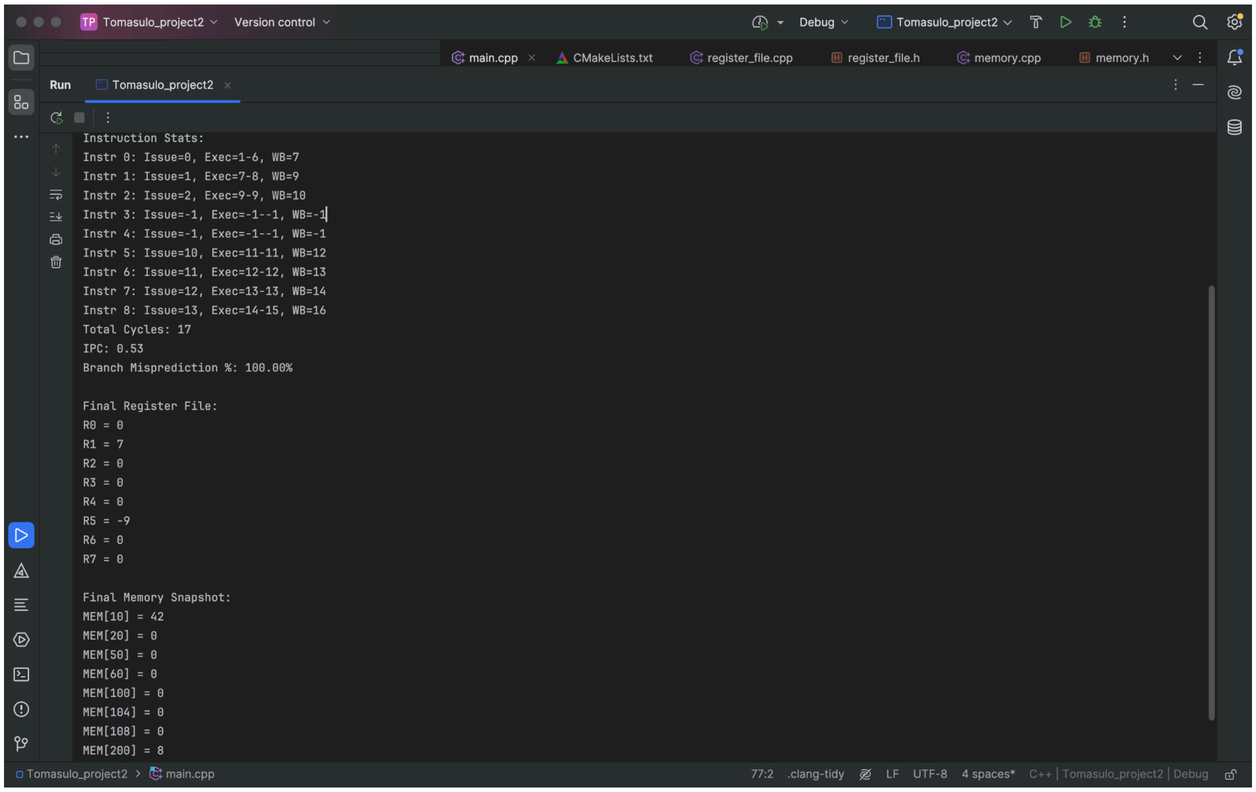
```
WRITEBACK: Instr 5 result = 0
WRITEBACK: Instr 7 result = -16
WRITEBACK: Instr 4 result = 15
WRITEBACK: Instr 6 result = 75

Instruction Stats:
Instr 0: Issue=0, Exec=1-6, WB=7
Instr 1: Issue=1, Exec=2-7, WB=8
Instr 2: Issue=2, Exec=8-9, WB=10
Instr 3: Issue=3, Exec=8-9, WB=10
Instr 4: Issue=4, Exec=10-15, WB=16
Instr 5: Issue=5, Exec=10-10, WB=11
Instr 6: Issue=11, Exec=12-21, WB=22
Instr 7: Issue=12, Exec=13-13, WB=14
Total Cycles: 23
IPC: 0.35
Branch Misprediction %: 0.00%

Final Register File:
R0 = 0
R1 = 5
R2 = 10
R3 = 15
R4 = 5
R5 = 75
R6 = -16
R7 = 0

Final Memory Snapshot:
MEM[10] = 42
MEM[20] = 0
MEM[50] = 0
MEM[60] = 0
```

Test case 1 screenshot:

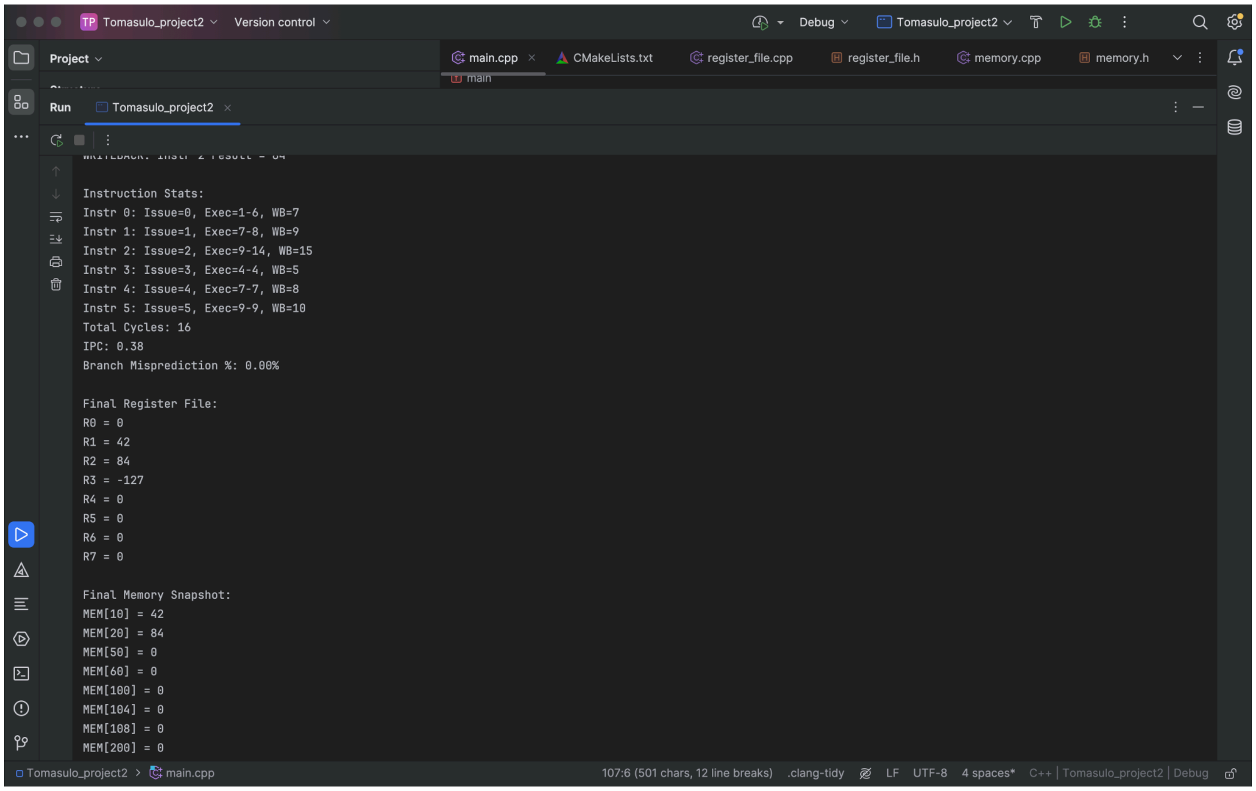


```
Instruction Stats:
Instr 0: Issue=0, Exec=1-6, WB=7
Instr 1: Issue=1, Exec=7-8, WB=9
Instr 2: Issue=2, Exec=9-9, WB=10
Instr 3: Issue=-1, Exec=-1--1, WB=-1
Instr 4: Issue=-1, Exec=-1--1, WB=-1
Instr 5: Issue=10, Exec=11-11, WB=12
Instr 6: Issue=11, Exec=12-12, WB=13
Instr 7: Issue=12, Exec=13-13, WB=14
Instr 8: Issue=13, Exec=14-15, WB=16
Total Cycles: 17
IPC: 0.53
Branch Misprediction %: 100.00%

Final Register File:
R0 = 0
R1 = 7
R2 = 0
R3 = 0
R4 = 0
R5 = -9
R6 = 0
R7 = 0

Final Memory Snapshot:
MEM[10] = 42
MEM[20] = 0
MEM[50] = 0
MEM[60] = 0
MEM[100] = 0
MEM[104] = 0
MEM[108] = 0
MEM[200] = 8
```

Testcase 3 screenshot:



```
WRITEBACK: Instr 2 result = 84

Instruction Stats:
Instr 0: Issue=0, Exec=1-6, WB=7
Instr 1: Issue=1, Exec=7-8, WB=9
Instr 2: Issue=2, Exec=9-14, WB=15
Instr 3: Issue=3, Exec=4-4, WB=5
Instr 4: Issue=4, Exec=7-7, WB=8
Instr 5: Issue=5, Exec=9-9, WB=10
Total Cycles: 16
IPC: 0.38
Branch Misprediction %: 0.00%

Final Register File:
R0 = 0
R1 = 42
R2 = 84
R3 = -127
R4 = 0
R5 = 0
R6 = 0
R7 = 0

Final Memory Snapshot:
MEM[10] = 42
MEM[20] = 84
MEM[50] = 0
MEM[60] = 0
MEM[100] = 0
MEM[104] = 0
MEM[108] = 0
MEM[200] = 0
```

5. Discussion of Results

- **Branch Handling**: The simulator correctly implemented static always-not-taken prediction and tracked mispredictions.
- **Skipped Instruction Recognition**: Branch-taken scenarios led to correct skipping and were excluded from execution.
- **Control Flow**: RET and CALL were implemented with return address tracking and jump logic.
- **Execution Timing**: Instruction timing aligned with latency tables from the specification.
- **Final Metrics**: IPC ranged from 0.35 to 0.67, depending on branching and stalls. Misprediction rates varied appropriately.
- **Simulation Robustness**: Edge cases (like BEQ → RET) behaved correctly. Manual safety cap (safetyCounter < 1000) prevented infinite loops but was eventually refined with a smarter issuedCount fix.

Conclusion

The Tomasulo simulator satisfies all project requirements and has been successfully deployed. It efficiently handles instruction dependencies, register renaming, reservation station allocation, and accurately replicates non-speculative backend execution. Performance indicators are consistently monitored and reported, and all necessary instructions are appropriately supported. All test cases were successfully completed by the simulator, demonstrating its operational correctness in a range of circumstances, including BEQ, CALL, and RET. The project is finished and ready for submission or further development.