Tomasulo Simulator Project Report

1. Implementation Overview

This project implements a simplified Tomasulo algorithm simulator in C++ for a custom 16-bit RISC processor. It models key architectural features of out-of-order execution, including reservation stations, register renaming, and common data bus broadcasting. The core of the simulator involves:

- Instruction parsing and issuing
- Reservation station management
- Execution scheduling based on operand availability
- Register file and status table tracking

Bonus features implemented:

- Assembly Program Instruction parsing
- GUI

2. AI Assistance Description

AI assistance was used in the following ways during the development process:

Design Assistance

- Clarifying the architecture of Tomasulo's algorithm, especially the role of reservation stations and register renaming.
- Recommending data structures to represent reservation stations, instructions, and pipeline tracking.
- Outlining edge cases for branching and memory-related instructions.

Coding Assistance

- Writing reusable structures for instructions and stations
- Debugging logic in reservation station issue rules

- Creating a simulation loop that correctly transitions between stages
- Formatting output to clearly show the instruction states at each cycle

All AI suggestions were carefully reviewed and integrated manually to align with the project specifications.

3. User Guide

Running the Simulation
1- Compile the code in your favourite compiler.
2 - Add Assembly Program in a text file "test.txt"

Assembly Input Example:

```
LOAD R2, 0(R0)
LOAD R3, 1(R0)
ADD R4, R2, R3
SUB R5, R2, R3
NOR R6, R2, R3
MUL R7, R2, R3
STORE R4, 2(R0)
STORE R5, 3(R0)
BEQ R2, R3, 2
ADD R4, R4, R4
CALL 3
ADD R1, R1, R0
label: SUB R7, R7, R0
```

Step-by-step Cycle Output:

At each step, the terminal output shows:

- Current cycle number
- Status of each reservation station
- Register file and status table
- Any instructions issued or completed

4. Simulation Results

The simulator correctly processes all example programs provided.

Results:

- Instructions issued in order
- Execution overlapped for ADD and MUL due to reservation station availability
- Correct final values computed and written to registers

5. Discussion of Results

The Tomasulo simulator successfully demonstrates dynamic scheduling and out-of-order execution capabilities:

- Dependency resolution: Proper register renaming and forwarding eliminate read-after-write hazards.
- Efficient utilization: Instructions execute as soon as operands are ready, improving throughput.
- Branch handling: Basic BEQ and JMP logic implemented with always not taken branch prediction.

Limitations and Future Work:

- Always not taken branch prediction is not very efficient, 2 bit or 3 bit branch prediction would be better.
- Performance metrics (e.g., CPI) could be added for deeper analysis.