**Project Report: Tomasulo Algorithm Simulator**

1. **Introduction**The goal of this project was to develop a simulator for the Tomasulo algorithm, an approach used in computer architecture to execute instructions out-of-order to improve performance. The simulator was developed in JAVA programming language.
2. **Development Approach**

* Understanding the Algorithm: The first step involved a thorough study of the Tomasulo algorithm, focusing on its components - reservation stations, buffers, register file, cache, and execution queue.
* Code Structure:
  1. Input Handling: The simulator accepts MIPS instructions from a text file. Input parsing was implemented to handle different instruction formats and types.
  2. Core Simulation: The core simulation engine was designed to process instructions cycle by cycle, updating the status of reservation stations, buffers, register files, and caches accordingly and at the end of each cycle it prints the execution queue to display the progress of the code execution.
  3. User Interface: A command-line interface (CLI) was developed for input and real-time simulation display. It was made to ensure a user-friendly and clear display of simulation steps.
  4. GUI: As an additional feature, a basic graphical user interface (GUI) was developed to enhance user experience.
* Instruction Types: The simulator supports ALU operations like floating point operations (ADD.D, SUB.D, MUL.D and DIV.D), integer operations (ADDI, SUBI), loads, stores, and branch operations (BNEZ).
* User Customization: Users can input the latency of each instruction type and select the sizes of stations and buffers. The cache and register file can be pre-loaded or user-configured.
* Handling Instruction Hazards: The simulator effectively manages RAW, WAR, and WAW hazards.

1. **Handling Simultaneous Result Publication**

Our approach to manage scenarios where multiple instructions, attempt to publish results simultaneously is a First-In, First-Out (FIFO) mechanism. In this approach, instructions ready to write back their results are queued in “Writebackqueue” in the order they were issued. When a conflict arises—where multiple instructions complete and are ready to publish in the same cycle—the FIFO queue determines the order of access to the Common Data Bus (CDB). The instruction that entered the queue first is given priority, ensuring an orderly and fair resolution of write-back conflicts while maintaining program order and data consistency. This FIFO mechanism effectively mirrors real CPU behavior in handling concurrent execution and result publication.

1. **Testing**
2. Test Cases:
3. Basic Operations: Tests with simple instruction sets to verify basic functionality.
4. Complex Scenarios: Simulations with mixed instruction types, including loops and various hazards.
5. Edge Cases: Tests to ensure stability under unusual or extreme conditions, such as high instruction latencies or large buffer/reservations stations sizes.
6. User Feedback: Initial testing was followed by user feedback sessions, where a small group of users interacted with the simulator. Their feedback helped refine the CLI and GUI.
7. Debugging and Optimization: Based on test results and user feedback, debugging and performance optimization were carried out.
8. **Conclusion**

The Tomasulo algorithm simulator was successfully developed and tested, meeting the project requirements. It effectively demonstrates the Tomasulo algorithm's functionality and can handle a wide range of instruction sets and scenarios.