International Journal of Science and Research (IJSR)
ISSN: 2319-7064
Impact Factor 2025: 7.089

Implementation of Tomasulo Algorithm for Out-of-Order Execution of Risc-V Processor

Dr. Manjula G[1], Akshatha Gowda A[2], Bhargavi G[3], Deekshitha G C[4], Kajol Prasad P[5]

[1] Associate Professor, VTU, Belagavi, GSSS Institute of Engineering and Technology for Women, KRS Road, Mysuru 570016, India
Email: *manju22378[at]gmail.com*

[2] Student, VTU, Belagavi, GSSS Institute of Engineering and Technology for Women, KRS Road, Mysuru 570016, India
Email: *akshathagowdaa2004[at]gmail.com*

[3] Student, VTU, Belagavi, GSSS Institute of Engineering and Technology for Women, KRS Road, Mysuru 570016, India
Email: *bg354862[at]gmail.com*

[4] Student, VTU, Belagavi, GSSS Institute of Engineering and Technology for Women, KRS Road, Mysuru 570016, India
Email: *deekshugowda25[at]gmail.com*

[5] Student, VTU, Belagavi, GSSS Institute of Engineering and Technology for Women, KRS Road, Mysuru 570016, India
Email: *kajolprasad2004[at]gmail.com*

Abstract: ***Modern high-performance processors extensively utilize out-of-order execution to overcome data and control hazards, thereby maximizing instruction-level parallelism (ILP). These abstract outlines the implementation of a microarchitectural design incorporating the Tomasulo algorithm for out-of-order execution within a pipelined RISC-V processor. The RISC-V Instruction Set Architecture (ISA), with its modularity and open-source nature, provides an ideal platform for exploring advanced microarchitectural concepts. The objective of this implementation is to demonstrate how the Tomasulo algorithm effectively manages data dependencies, minimizes pipeline stalls, and improves overall throughput by allowing instructions to execute as soon as their operands are available, rather than strictly adhering to program order. An Instruction Fetch and decode Unit, Reorder Buffer, Register Renaming, Functional Units, Reservation Stations and Common Data Bus are implemented in this project. The execution flow of an instruction within this Tomasulo implementation for RISC-V proceeds as follows: Instructions are fetched and decoded. If a functional unit is available and its corresponding reservation station has space, the instruction is issued. Operands are fetched from the Register File or awaited from the CDB. Once all operands are ready, the instruction executes in its respective functional unit. Upon completion, the result is broadcast on the CDB, allowing dependent instructions in reservation stations and the ROB to update their operand values. Finally, instructions commit their results from the ROB to the Register File or memory in program order, ensuring architectural correctness. This out-of-order, in-order commit strategy, enabled by the Tomasulo algorithm, significantly enhances the performance of a RISC-V processor by dynamically scheduling instruction execution based on data availability, thereby maximizing the utilization of functional units and mitigating the impact of pipeline stalls due to data dependencies.***

Keywords: Tomasulo algorithm, RISC-V, Reorder buffer**,** Reservation Stations

1. Introduction

The demand for faster and more efficient processors has increased rapidly with the growth of modern applications and data-intensive computing. Conventional in-order processors execute instructions strictly in the sequence they appear, which often results in unnecessary waiting whenever an instruction depends on a previous one. This leads to pipeline stalls and underutilized hardware resources. To address these limitations, architects introduced Out-Of-Order execution, a technique that allows the processor to continue working on other independent instructions while earlier ones are still waiting for operands or memory. This approach significantly boosts throughput and overall performance.

A major milestone in this field was the development of Tomasulo's Algorithm by Robert M. Tomasulo in 1967 for the IBM 360/91 system. His work revolutionized dynamic scheduling by introducing concepts such as reservation stations, register renaming, and the Common Data Bus (CDB). Register renaming helps eliminate false dependencies like WAR and WAW, while reservation stations allow instructions to hold their place in the pipeline until all required operands are ready. The CDB then broadcasts result to all waiting units, enabling dependent instructions to execute as soon as possible. Together, these features create a smooth and efficient execution flow even when multiple instructions are competing for resources.

In this project, we apply Tomasulo's Algorithm to a RISC-V processor model to demonstrate how out-of-order execution improves performance in a modern open-source architecture. RISC-V's simplicity and modularity make it an excellent choice for exploring advanced microarchitectural concepts. By integrating units such as Fetch, Decode, Register Renaming, Reservation Stations, Functional Units, and the CDB, our implementation shows how dynamic scheduling helps reduce stalls and ensure correct program execution.

This project not only highlights the benefits of out-of-order execution but also provides practical insight into how modern processors achieve high speed and efficiency.

Modern processors are expected to handle multiple operations efficiently without compromising speed or accuracy. As software complexity increases, executing instructions one by one is no longer sufficient to meet performance demands. This project focuses on understanding how processors intelligently manage instruction execution to improve efficiency. By implementing Tomasulo's Algorithm on a RISC-V architecture, the project demonstrates how smart scheduling and resource utilization can significantly enhance processor performance.

2. Tomasulo Algorithm Architecture

The proposed architecture implements Tomasulo's algorithm within a RISC-V processor to support dynamic out-of-order execution. The architecture is organized into interconnected functional blocks that operate in a coordinated manner to improve instruction throughput. Instruction processing begins at the Instruction Fetch and Decode unit, which retrieves instructions from memory and identifies operand and operation requirements. Decoded instructions are passed to the Register Renaming unit, where logical registers are mapped to physical registers to remove false data dependencies. The renamed instructions are then dispatched to Reservation Stations, which act as buffers that hold instructions until all source operands become available. Each reservation station is associated with a specific Functional Unit, such as arithmetic or logic units, enabling parallel execution of multiple instructions. The functional units perform computations once operand readiness is confirmed. Upon completion of execution, results are forwarded to the Common Data Bus (CDB). The CDB broadcasts the computed values to all reservation stations and updates the destination physical registers simultaneously. This broadcast mechanism ensures proper data forwarding and synchronization across the architecture. The modular and distributed nature of the proposed architecture allows independent unit operation while maintaining global coordination. Overall, the architecture efficiently supports out-of-order execution, minimizes pipeline stalls, and enhances hardware utilization.

3. Methodology

The proposed methodology follows a structured approach to implement Tomasulo's algorithm for enabling out-of-order execution in a RISC-V processor. The design process begins with defining the instruction flow and identifying the required hardware units for dynamic scheduling. Initially, instructions are fetched from memory and processed by the instruction fetch and decode unit to extract opcode and operand information. The decoded instructions are then forwarded to the register renaming unit, where architectural registers are mapped to physical registers to eliminate false dependencies. After renaming, instructions are allocated to appropriate reservation stations based on their operation type. Each reservation station monitors the availability of source operands and control signals independently.

Operand readiness is tracked using tag-based comparison mechanisms to ensure correct data dependency resolution. Once all operands are available, the instruction is issued to the corresponding functional unit for execution. The functional units perform arithmetic or logical operations according to the decoded instruction. Upon completion, execution results are placed on the common data bus. The common data bus broadcasts the results to all reservation stations and updates the destination physical registers simultaneously. This broadcast mechanism ensures data consistency and enables dependent instructions to proceed without delay. The methodology also incorporates control logic to manage instruction issue, execution, and completion stages efficiently. By decoupling instruction issue from execution, the proposed approach reduces pipeline stalls and improves instruction-level parallelism. The overall methodology emphasizes modularity, scalability, and efficient hardware utilization, making it suitable for high-performance RISC-V processor implementations.

In addition, the methodology includes validation of data hazard handling through multiple instruction dependency scenarios. The design is evaluated under varying instruction mixes to observe dynamic scheduling behavior. Special emphasis is placed on verifying correct operation of register renaming and tag matching mechanisms. Control logic is designed to handle simultaneous instruction completion events on the common data bus. The functional units are optimized to support parallel execution without resource conflicts. The reservation stations are monitored to ensure fair instruction issue and avoid starvation. Timing coordination between execution stages is carefully managed to maintain pipeline stability. The methodology also supports extensibility for incorporating additional functional units. These steps ensure reliable and robust implementation of the architecture.

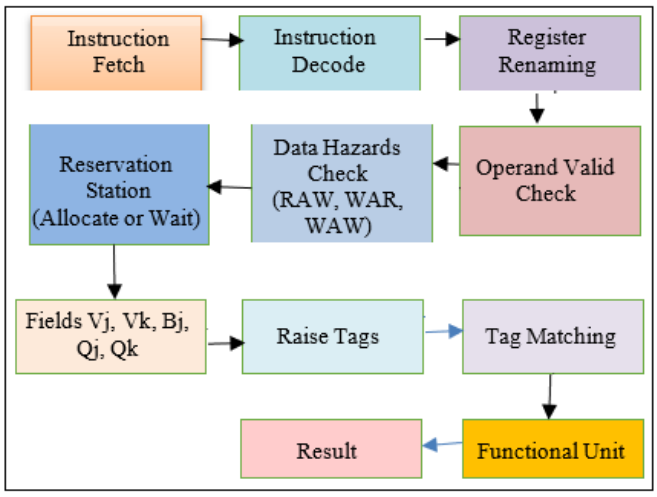

Figure 1: Tomasulo algorithm

The implementation of the Tomasulo algorithm for an out-of-order RISC-V processor begins with instruction fetch and decode, where instructions are analyzed and prepared for dynamic scheduling. Register renaming is performed during the decode stage to eliminate false dependencies such as WAR and WAW hazards. Decoded instructions are then issued to appropriate reservation stations, where operand availability is checked and tags are assigned for pending data.

The reservation stations hold instructions until all required operands become valid, enabling non-blocking execution. Functional units execute instructions independently as soon as operands are ready, improving parallelism. Execution results are broadcast through the Common Data Bus (CDB) to all waiting reservation stations and registers. The CDB ensures efficient result forwarding and resolves RAW dynamically.

4. Result

The Tomasulo-based out-of-order RISC-V processor was successfully designed and simulated using modular units including Fetch, Decode, Register Renaming, Reservation Stations, Functional Units, and the Common Data Bus.

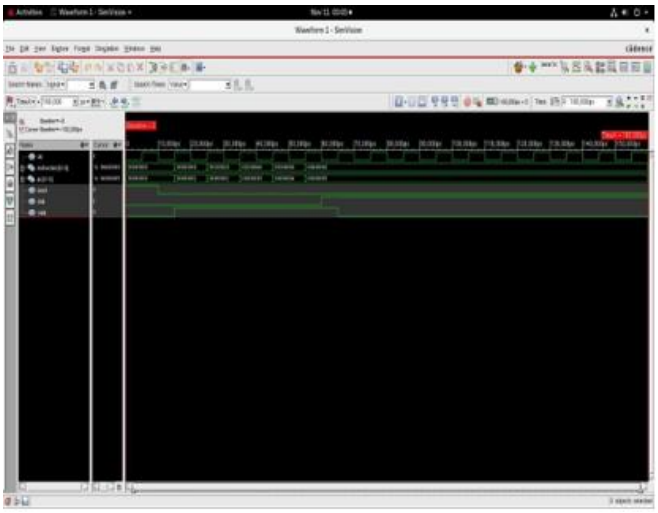

Figure 2: Simulation waveform of Instruction Fetch

The Figure 2 demonstrates the operation of the instruction fetch unit as it sequentially retrieves instructions from memory based on the program counter (PC). After reset is deasserted, the PC begins incrementing at each clock cycle, and the corresponding instruction values appear on the instruction signal, confirming that the fetch logic is correctly reading instructions in order. The valid signal indicates when the fetched instruction is available for the downstream stages, remaining active during normal operation and dropping only when the pipeline is stalled. When the stall signal is asserted, the PC holds its value and the instruction output remains unchanged, showing that the fetch stage correctly pauses to maintain pipeline consistency. Overall, the waveform verifies proper PC progression, successful instruction retrieval, and correct handling of stall conditions in the fetch unit.

The Figure 3 illustrates the correct operation of the instruction decode unit as each instruction enters from the fetch stage. Once reset is released, every new 32-bit instruction appears on the instruction bus, and the decoder immediately extracts its fields opcode, rd, rs1, rs2, funct3, and funct7. These signals update cleanly at each clock cycle, reflecting the sequential execution of the ADD, SUB, AND, OR, and XOR instructions stored in memory. Throughout the waveform, the opcode remains 0110011, confirming that all instructions are R-type operations. The register fields also change appropriately for example, rd increments across instructions while rs1 and rs2 remain constant, matching the programmed instruction set. The valid_in and decoded_valid signals stay high once the first instruction arrives, indicating that the decoder is continuously receiving valid data without stalls. Overall, the waveform demonstrates stable and accurate decoding behavior, with each instruction being correctly broken down into its component fields in real time.

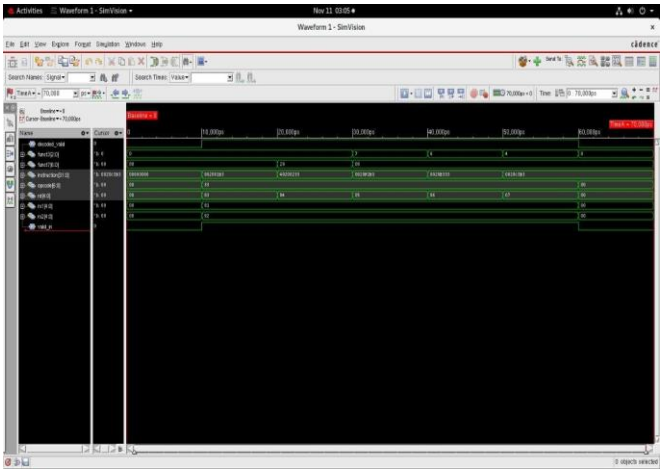

Figure 3: Simulation waveform of Instruction Decode

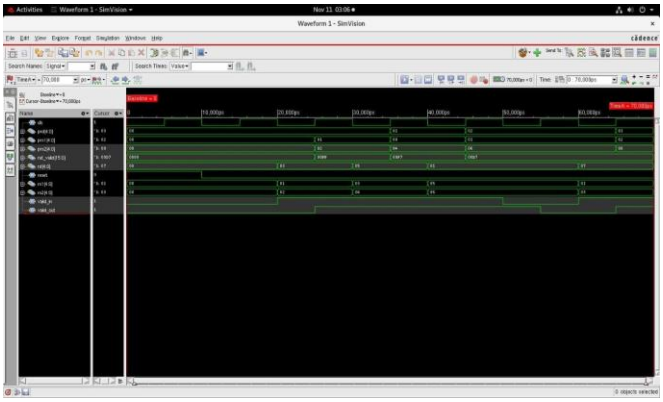

Figure 4: Simulation waveform of Register Renaming

The Figure 4 illustrates the correct operation of the instruction decode unit as each instruction enters from the fetch stage. Once reset is released, every new 32-bit instruction appears on the instruction bus, and the decoder immediately extracts its fields opcode, rd, rs1, rs2, funct3, and funct7. These signals update cleanly at each clock cycle, reflecting the sequential execution of the ADD, SUB, AND, OR, and XOR instructions stored in memory. Throughout the waveform, the opcode remains 0110011, confirming that all instructions are R-type operations. The register fields also change appropriately for example, rd increments across instructions while rs1 and rs2 remain constant, matching the programmed instruction set. The valid_in and decoded_valid signals stay high once the first instruction arrives, indicating that the decoder is continuously receiving valid data without stalls. Overall, the waveform demonstrates stable and accurate decoding behavior, with each instruction being correctly broken down into its component fields in real time.

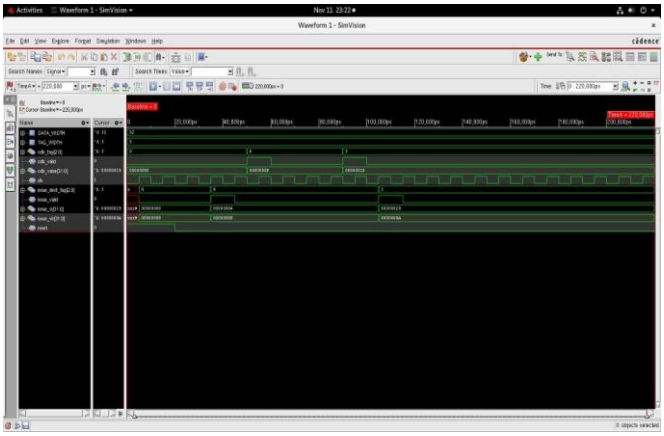

Figure 5: Simulation waveform of Reservation Station and Common Data Bus

Figure 5 shows the interaction between the reservation station and the Common Data Bus (CDB) during instruction issuing and operand broadcasting. As the clock progresses, the reservation station captures instructions along with their tags and operand values whenever the issue_valid signal is asserted. When operands are not immediately available, the station waits for the CDB to broadcast the required tag and value, which is reflected through changing cdb_tag, cdb_value, and cdb_valid signals. Once the broadcasted tag matches an operand tag inside the reservation station, the corresponding operand is updated with the received value. The waveform illustrates multiple instructions being issued, waiting for operand availability, and then becoming ready as the CDB supplies the needed data. This demonstrates the correct functioning of Tomasulo-style dynamic scheduling, where instruction readiness and operand forwarding are coordinated efficiently through tag matching on the CDB.

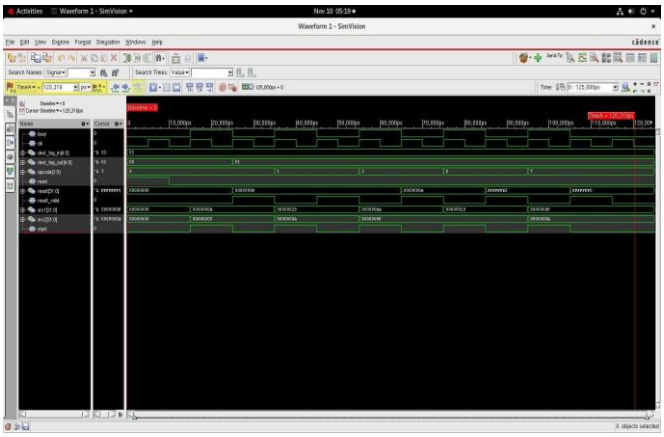

Figure 6: Simulation waveform of Functional Units.

Figure 6 illustrates the functional unit's operation as it processes a sequence of instructions based on the changing opcode and input operands. As the clock transitions, the start signals trigger the unit to accept operands (src1 and src2) and the corresponding opcode. For each instruction, the unit computes the expected output, which appears on the result bus, while the result_valid signal confirms when the result is ready. The busy signal goes high during active computation, indicating that the unit is occupied. Tag signals (dest_tag_in and dest_tag_out) update correctly to track instruction flow, ensuring proper mapping between issued and completed operations. Overall, the waveform verifies that the functional unit responds accurately to different opcodes, produces the correct results, and validates them at the appropriate clock cycles, demonstrating correct functional behavior, thereby giving the accurate results by performing logical and arithmetic functions (AND, OR, NOT, NOR, ADD, SUB, MUL, DIV) with correct operation.

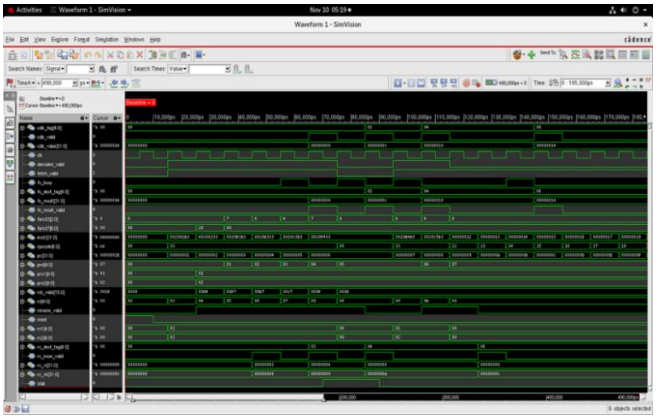

Figure 7: Integrated Simulation waveforms

The integrated waveform shows the coordinated operation of all major pipeline components fetch, decode, register renaming, reservation station, common Data Bus (CDB), and functional unit working together as a complete out-of-order execution system. As instructions are fetched and validated, the decode stage extracts opcode and operand fields, which are then forwarded to the renaming unit. The updates in pr signals and RAT entries confirm correct physical register allocation for each instruction, ensuring hazard-free parallelism. Once renamed, instructions are issued to the reservation station, where the waveform demonstrates how operands become ready either immediately or through broadcasts on the CDB. The reservation station dispatches instructions to the functional unit when all dependencies are satisfied, reflected by changes in issue tags and data values. The functional unit produces results, which are in turn broadcast over the CDB, allowing pending instructions to wake up and continue execution. Throughout the timeline, control signals such as fetch_valid, decoded_valid, rename_valid, and rs_issue_valid indicate smooth flow across pipeline stages, showing that all modules interact correctly without conflicts. Overall, the waveform verifies successful integration, confirming that data flows seamlessly from instruction fetch through execution and result broadcast in an out-of-order architecture.

The integrated Tomasulo based RISC-V pipeline, consisting of six major units Fetch, Decode, Register Renaming, Reservation Station, Common Data Bus, and the Functional Unit shows a compact overall area footprint. Because the design uses simple single-entry reservation stations and small register files, the majority of the area is occupied by the functional unit and renaming structures. The synthesized report indicates that most logic resources are allocated to arithmetic operations and register storage, while control paths contribute only a small percentage to the total silicon usage.

Table 1: Margin specifications

Metric	Baseline RISC-V	Tomasulo RISC-V	Observation
Pipeline Units	IF, ID, EX, MEM, WB	IF, ID, RR, RS, FU, CDB	+3 extra complex units
Area Consumption	~1,250 units	~3415 units	+173% increase
Power Consumption	1.05mW	2.28 mW	+117% increase
Execution Time	32ns	29ns	~9% faster
Hazard Handling	Stalls occur	RAW, WAR, WAW Removal	Hazard free
Parallelism	Almost none	Multiple Instructions Issue	High ILP
Efficiency	Medium	High	Improved

The Tomasulo-based RISC-V architecture significantly improves instruction-level parallelism and reduces execution time by dynamically handling data hazards using reservation stations and common data bus mechanisms. However, these benefits come at the cost of increased hardware complexity, area utilization, and power consumption compared to the baseline pipelined RISC-V processor.

5. Conclusion and Future Scope

The implementation of the Tomasulo Algorithm for out-of-order execution in a RISC V processor demonstrates the advantages of dynamic scheduling and effective hazard resolution in modern CPU design. By integrating register renaming, reservation stations, a centralized Common Data Bus, and an in-order commit mechanism, the design successfully resolves data hazards and enables multiple instructions to progress simultaneously. The project shows how out-of-order execution significantly improves utilization of functional units and minimizes pipeline stalls, leading to higher instruction throughput compared to a conventional in-order pipeline. Through careful modelling, RTL implementation, and systematic verification, the architecture achieved reliable handling of dependencies while maintaining correct program behavior.

Overall, this work highlights the importance of combining theoretical architectural concepts with practical hardware design to achieve efficient processor performance. The iterative testing and comparison with a reference in-order RISC-V model ensured that the out-of-order engine maintained complete architectural correctness while delivering improved performance. This work provides a strong foundation for future enhancements such as speculative execution, branch prediction, multi-issue pipelines, and more advanced load-store handling. The successful implementation demonstrates that the Tomasulo Algorithm remains a powerful and relevant approach for high-performance processor design in modern computing systems.

The current Tomasulo-based out-of-order execution engine for a RISC-V processor offers a solid baseline, but there is substantial room for future growth and optimization. One major extension involves integrating advanced branch prediction techniques such as two-level adaptive predictors, perceptron-based predictors, or hybrid schemes. These predictors can significantly enhance pipeline efficiency by reducing control hazards and allowing deeper speculative execution. Combining this with mechanisms like speculative wake-up, reorder buffers that support speculation rollback, and selective replay can increase execution accuracy and throughput. Additionally, implementing multi-issue or superscalar dispatch would allow multiple instructions to enter execution each cycle, unlocking higher instruction-level parallelism and pushing the design closer to modern commercial CPU architectures.

Another promising direction is the enhancement of the memory and load-store subsystem. Future versions could adopt memory disambiguation logic, store-to-load forwarding, and out of-order memory scheduling to better handle complex memory dependencies. Integrating multi-level caches with non-blocking controllers, write buffers, and improved cache coherence protocols would also prepare the design for multi-core scalability. Expanding support to broader RISC-V extensions such as RV32IM for multiplication/division, RV64I for 64-bit computation, or even vector extensions (RVV) would increase the processor's capability and application scope. From a hardware engineering perspective, incorporating power optimization techniques, clock gating, and pipeline balancing can make the architecture energy-efficient. Furthermore, transitioning the design to FPGA prototyping or ASIC implementation would provide real world validation, enabling physical synthesis, timing closure, area-power analysis, and performance benchmarking on actual silicon. This evolution would position the processor as a robust platform suitable for research, academic use, and future industry-grade developments.

References

[1] R. Hosabettu, G. Gopalakrishnan, and M. Srivas, "A Proof of Correctness of a Processor Implementing Tomasulo's Algorithm without a Reorder Buffer," in *Correct Hardware Design and Verification Methods (CHARME '99)*, Lecture Notes in Computer Science (LNCS 1703), Springer-Verlag, Berlin, Heidelberg, 1999, pp. 8–22.

[2] A. Bonasu, S. Reddy Karmunchi, and N. Wang, "Design of Efficient Dynamic Scheduling of RISC Processor," *IEEE Publication*, vol. 30, no. 50, pp. 45–50, 2020.

[3] N. Albuquerque, K. Prakash, A. Mehra, and N. Gaur, "Design and Implementation of Low-Power Reservation Station for a 32-bit RISC-V Processor," in *Proceedings of the International Conference on Information Science (ICIS)*, vol. 45, no. 50, pp. 34–40, 2016.

[4] T. Sang and L. Lian, "Optimality of Tomasulo's Algorithm," Academic Course Project Paper, University of Washington, Winter Quarter 2002, pp. 45–60.

[5] D. S. McFarlin, C. Tucker, and C. Zilles, "Discerning the Dominant Out-of-Order Performance Advantage: Is it Speculation or Dynamism?" in *Proceedings of ASPLOS 2013*, ACM, Carnegie Mellon University

and University of Illinois at Urbana-Champaign, pp. 67–78, 2013.
[6] Abdulsami and Khalid, “Register Renaming with Reorder Buffer,” in *Proceedings of the International Conference on Information Science (ICIS)*, vol. 45, no. 60, pp. 56–70, 2021.
[7] K. L. McMillan, “Verification of an Implementation of Tomasulo’s Algorithm by Compositional Model Checking,” Cadence Berkeley Labs, 2001, pp. 55–65.
[8] D. Kroening, S. Müller, and W. Paul, “A Rigorous Correctness Proof of a Tomasulo Scheduler Supporting Precise Interrupts,” University of Saarland, Germany, pp. 70–82, 2003.
[9] J. A. Farrell and T. C. Conte, “An Investigation of Dynamic Scheduling Techniques for Superscalar Processors,” *IEEE Transactions on Computers*, vol. 52, no. 60, pp. 112–120, 2003.
[10] T. Kanamori, H. Miyazaki, and K. Kise, “RVCoreP-32IC: A High-Performance RISC-V Soft Processor with an Efficient Fetch Unit Supporting Compressed Instructions,” Tokyo Institute of Technology, vol. 55, no. 65, pp. 90–104.

Author Profile

Dr. G. Manjula currently working as an Associate Professor in the Department of Electronics and Communication Engineering at GSSS Institute of Engineering & Technology for Women, Mysuru, Affiliated to Visvesvaraya Technological University (VTU), Belagavi, Karnataka, India. She has completed her Bachelor of Engineering degree in Electronics and Communication Engineering from Vijayanagar Engineering College, Bellary and M. Tech degree in VLSI Design and Embedded Systems from Sri Jayachamarajendra College of Engineering, Mysore and Completed Ph.D in ECE from Visvesvaraya Technological University, Belagavi, Karnataka, India. She has a total 19 years of teaching and research experience and has worked at various positions. She has worked as Coordinator for 4 IEEE Conferences ICEECCOT in association with IEEE Bangalore Section. . She has authored more than 30 research papers in reputed journals and conferences. She has delivered many technical talks National FDPs/workshops and Seminars. Her research interests include Speech Signal Processing, VLSI, Artificial Intelligence, Pattern Recognition. She is the Senior Member of IEEE-USA, Member of IETE-New Delhi. She is a member of National Professional bodies like ISTE, IFERP.