WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal Homepage: www.ijrpr.com ISSN 2582-7421

Design and Implementation of a Bubble-Free Five-Stage Pipelined RISC-V Processor

Batana Harsha Vardhini [1]*, Ampolu Srihari [2]*, Gompa Dharmendra Kumar [3]*, Dr. Guntu Nooka Raju [4]*, Dr. Babji Prasad Chapa [5]*

[1-5] Department of Electronics and Communication Engineering, GMR Institute of Technology, Rajam, 532127, Andhra Pradesh, India.

A B S T R A C T

Conventional five-stage RISC-V processors experience performance degradation due to pipeline bubbles caused by data hazards, particularly the load-use dependency between a load (LW) instruction and its immediately following dependent instruction. In order to remove the load-use stall while maintaining the traditional five-stage pipeline topology, a bubble-free five-stage in-order RV32I RISC-V processor is designed and built in Verilog HDL. The suggested architecture includes an expanded Forwarding Unit to send loaded data straight to the execution stage, a Bubble Fixing Control Unit (BFCU) for hazard-driven control, and an Address Generator (AG) for early load-address computation. The proposed processor is simulated and synthesized using Xilinx Vivado 2024.2 and compared with a conventional five-stage RISC-V processor under identical operating conditions. Simulation results demonstrate a reduction in execution time from 110 ns to 100 ns, corresponding to an increase in throughput from 54.55 MIPS to 60 MIPS. Despite an approximately 2.5% increase in power consumption due to the additional hardware, the proposed architecture achieves a 6.8% reduction in energy consumption and a 7.31% improvement in energy efficiency. These results confirm that the proposed bubble-free architecture provides improved performance and pipeline utilization with a favorable energy-efficiency trade-off while preserving correct processor functionality.

Keywords: RISC-V, RV32I, five-stage pipeline, load-use hazard, bubble-free, forwarding, Verilog HDL.

1. Introduction

Based on the reduced instruction set computer (RISC) concept of employing a limited set of basic instructions, RISC-V is an open, royalty-free instruction set architecture (ISA) [15]. An ISA, which specifies a processor's instructions, registers, and memory model, is an agreement between software and hardware. This ISA is implemented by a digital circuit called a RISC-V processor, whose function is to execute a program by continually reading and executing instructions from memory. Any university or business can create, alter, or expand the ISA with additional instructions because it is open. RISC-V is therefore used in everything from edge-computing processors to tiny microcontrollers [4], [6]. The ease of building on FPGAs and ASICs is demonstrated by open-source cores such as SHAKTI [7], [8], Orca [9], Mriscv [10], VexRiscv [11], low RISC [12], and RI5CY [13]. The 32-bit base integer ISA, RV32I, is used in this study. Its main ideas are straightforward. Each of its 32 general-purpose registers, x0 to x31, is 32 bits wide, and x0 is always equal to zero. Decoding is simple because each instruction is 32 bits long and belongs to one of a few standard forms. Only load (LW) and store (SW) instructions access memory in this load-store architecture. Arithmetic and logic instructions like ADD, SUB, AND, and OR operate on registers, and branches like BEQ alter the program's flow.

Every instruction follows the same five steps. The instruction is read from the instruction memory at the address stored in the program counter (PC) when using Instruction Fetch (IF). The instruction is decoded and its source registers are read in Instruction Decode (ID). The ALU carries out the action or determines a memory address in Execute (EX). The data memory is loaded, read, stored, and written in Memory Access (MEM). The outcome is written to the destination register in Write Back (WB). All five processes are completed in a single, lengthy clock cycle by a single-cycle CPU, which is straightforward but sluggish. Five instructions occur in separate stages at once when a pipelined processor sets registers between the stages; ideally, one instruction completes in each clock cycle [15].

However, instructions are not always autonomous. Control and structural dangers originate from branching and shared hardware, while data hazards develop when an operation requires a result that a previous instruction has not yet generated [14], [15]. Forwarding, which transmits a result straight from a later step to the ALU input without waiting for write-back, resolves the majority of data dangers [14], [15]. Stalls, often known as bubbles, are the clock cycles during which the pipeline must still wait; they lower performance and produce no beneficial work. The load-use threat cannot be eliminated by forwarding [14]. The loaded word in LW x3, 0(x6) followed by ADD x5, x3, x4 is only accessible after the LW's MEM stage, while the ADD requires x3 at the beginning of its EX-stage, necessitating the insertion of a one-cycle bubble, as seen in Table 1. For low-end and edge devices, where both

performance and energy efficiency are critical, frequent bubbles increase the number of cycles a program needs and decrease pipeline utilization. Pre-fetching has also been studied as a way to reduce stalls [16].

Table 1 - Load-use bubble in a conventional pipeline.

Instruction	Cycle 1	Cycle 2	Cycle 3	Cycle 4	Cycle 5	Cycle 6
LW	IF	ID	EX	MEM	WB	—
ADD	—	IF	ID	**ST (Bubble)**	EX	MEM

Research on RISC-V processors now focuses on pipelined and optimized microarchitectures rather than simple, energy-efficient solutions. An energy-efficient single-cycle RV32I microprocessor for edge computing was presented in [4]. It does not leverage instruction-level parallelism and each instruction must fit within a single clock period, however it has a simple datapath and uses little energy. To overcome this limitation, a five-stage pipelined RISC-V processor was developed and implemented on FPGA in [14]. Overlapping the five processes confirms the architecture's viability, but pipelining exposes structural, data, and control risks, making controlling bubbles and hazards challenging. An adaptive two-stage pipelined soft processor for edge computing was presented in [3] using a different trade-off; a shorter pipeline reduces overhead and energy but provides less instruction-level parallelism than a five-stage architecture. For energy-constrained low-end applications, a LW-driven bubble-free five-stage core was recently proposed in [1]. This work is most closely connected to this paper since it eliminates the load-use stall while maintaining the five-stage pipeline. Additionally, reliability has been investigated, for instance using error detection and correction methods in [2].

In conclusion, [3] prioritizes energy at the expense of parallelism, [4] is a straightforward and energy-efficient baseline with poor throughput, [14] has higher throughput but has stalls, and [1] tackles the risk brought on by loads. A straightforward bubble-free five-stage architecture that eliminates idle cycles while maintaining the traditional five-stage layout, accuracy, performance, and energy efficiency can still be developed and tested. In order to compare it with a conventional processor in terms of execution cycles, pipeline efficiency, throughput, and energy, this paper presents a bubble-free five-stage RV32I processor that was constructed by adding an Address Generator, a Bubble Fixing Control Unit, and an extended Forwarding Unit to a conventional pipeline. This section closes with a description of the conventional processor; Section 2 offers the suggested design; Section 3 provides the findings; and Section 4 wraps up.

1.1 Traditional RISC-V processor

The traditional processor, shown in Fig. 1, divides instruction execution into the IF, ID, EX, MEM and WB stages, separated by the IF/ID, ID/EX, EX/MEM and MEM/WB pipeline registers. Each register holds the instruction data and control signals needed by the next stage, so that up to five instructions are in flight in the same clock cycle [14], [15].

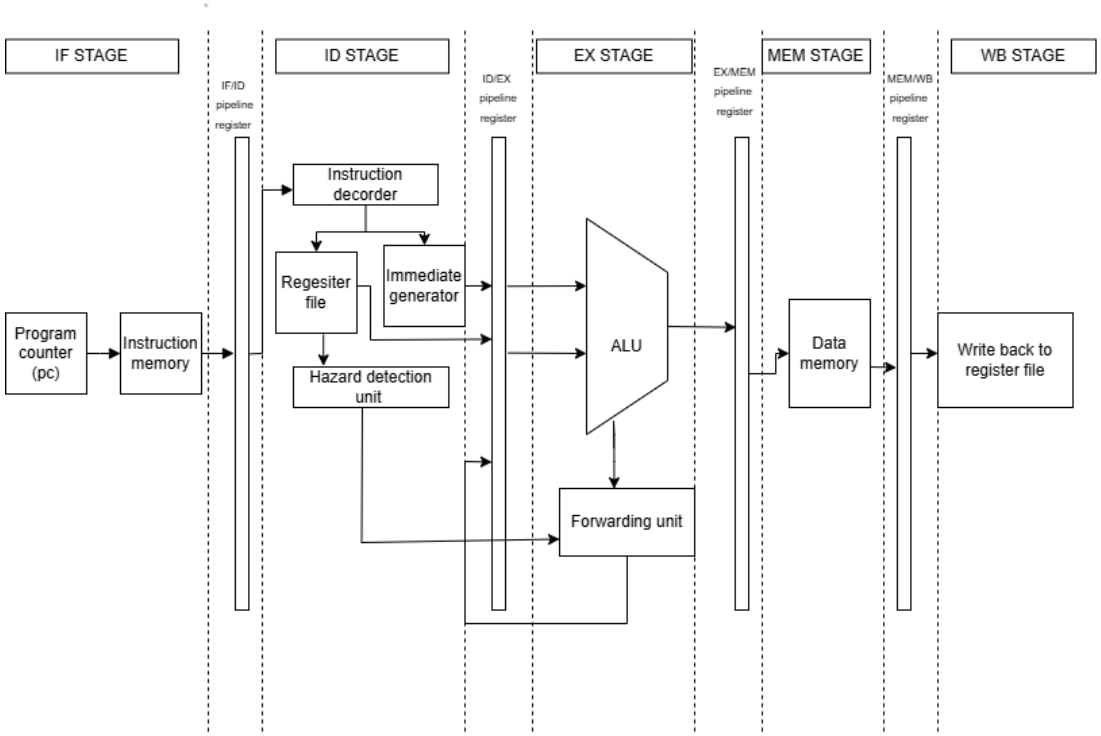


Fig. 1 - Block diagram of the traditional five-stage RISC-V processor.

The PC stores the address of the instruction to be fetched, and execution starts in the IF stage. For sequential 32-bit instructions, the PC is updated as PC_next = PC + 4. The PC addresses the instruction memory, and the fetched instruction is kept in the IF/ID register. The next instruction can be retrieved while the preceding one is being decoded thanks to the IF/ID register.

The instruction decoder and control unit separate the instruction into its opcode, register, function, and immediate fields during the ID stage. They also identify the type of instruction (for instance, ADD is a register-register operation, LW is a load, SW is a store, and BEQ is a conditional branch) and produce the control signals for the subsequent stages. In the WB stage, the register file, which has 32 registers of 32 bits, receives the final result and provides the source operands, such as x1 and x2 for ADD x5, x1, x2. The immediate field, such as the offset 8 in LW x5, 8(x2), is extracted and sign-extended by the immediate generator. By integrating the register file and immediate generator outputs into the address information of loads and stores, Base Register + Immediate Offset, the address generator (AG) of the baseline aids memory operations. The hazard detection unit (HDU) keeps an eye on the instructions while they are in flight and recognizes when an instruction requires information that has not yet been generated by a previous instruction. The ID/EX register is then used to contain the operands, immediate, destination register, ALU, memory, and register-write controls.

The ALU calculates the effective address of memory instructions, such as R[x2] + 8 for LW x5, 8(x2), and executes the arithmetic and logic operations (addition, subtraction, AND, OR, XOR, and comparison) chosen by the control signals during the EX stage. The majority of data dangers are eliminated via a forwarding unit (FU): for ADD x5, x1, x2 followed by SUB x6, x5, x3, the ADD result is given straight to the ALU rather than waiting for write-back, allowing the SUB to execute without a bubble [15]. When forwarding is insufficient, the pipeline replaces the delayed instruction with a NOP, an instruction with no useful architectural consequence, using the bubble/NOP path shown above the EX stage in Fig. 1. The EX/MEM register contains the store data, the destination register, the control information, and the ALU result or effective address.

While other instructions just pass their ALU result through, load instructions read the data memory at the address calculated in EX and store instructions write their value to it during the MEM step. The loaded data or ALU result, the destination register, and the register-write control are then carried by the MEM/WB register to the WB stage, which writes the result into the register file for reading by subsequent instructions.

This arrangement is still ineffective for the sequence LW x5, 0(x1) followed by ADD x6, x5, x2. Since the ADD needs the data at the start of EX but the LW can only get it after the MEM step, forwarding is pointless. Consequently, the HDU keeps the PC (pc_write goes low) while the bubble/NOP route inserts a NOP, producing the one-cycle bubble of Table 1. This load-use bubble represents the penalty that the proposed architecture removes.

2. Proposed methodology

The primary goal of the suggested processor is to prevent the dependent instruction from waiting for the load while maintaining the same five-stage in-order RV32I structure as the conventional one. The idea consists of three parts. First, the load's address is established early, instead of waiting for the ALU in the EX-stage. Second, the data-memory read starts earlier than the conventional time. Third, the loaded data is transmitted directly to the ALU for the dependent instruction instead of going through MEM/WB and the register file first. Only the idle cycle brought on by the LW is eliminated; no out-of-order execution is utilized. The strategy is based on Kumar and Ray's LW-driven bubble-free method [1].

LW x5, 0(x1) and ADD x6, x5, x2 are where the difference is most noticeable. The data arrives at the ALU too late for the ADD in the conventional pipeline since the address is calculated in EX and the memory is read in MEM, resulting in an inevitable stall. The ADD moves forward in the cycle following the LW without a bubble in the suggested pipeline since the address is generated early, the memory is read early, and the data is sent to the ALU. An Address Generator, a Bubble Fixing Control Unit (BFCU), and an expanded Forwarding Unit which collaborates with the Hazard Detection Unit are added to the baseline in order to do this.

Fig. 2 shows the resulting block diagram. The IF, ID, MEM and WB stages and the four pipeline registers follow the baseline of Fig. 1: the PC addresses the instruction memory, the instruction is stored in IF/ID and decoded in ID, and the HDU examines the dependencies between the instructions in flight. The changes are concentrated around the EX stage and the hazard-control loop. Compared with Fig. 1, the AG and BFCU appear as new blocks and no bubble/NOP path is drawn.

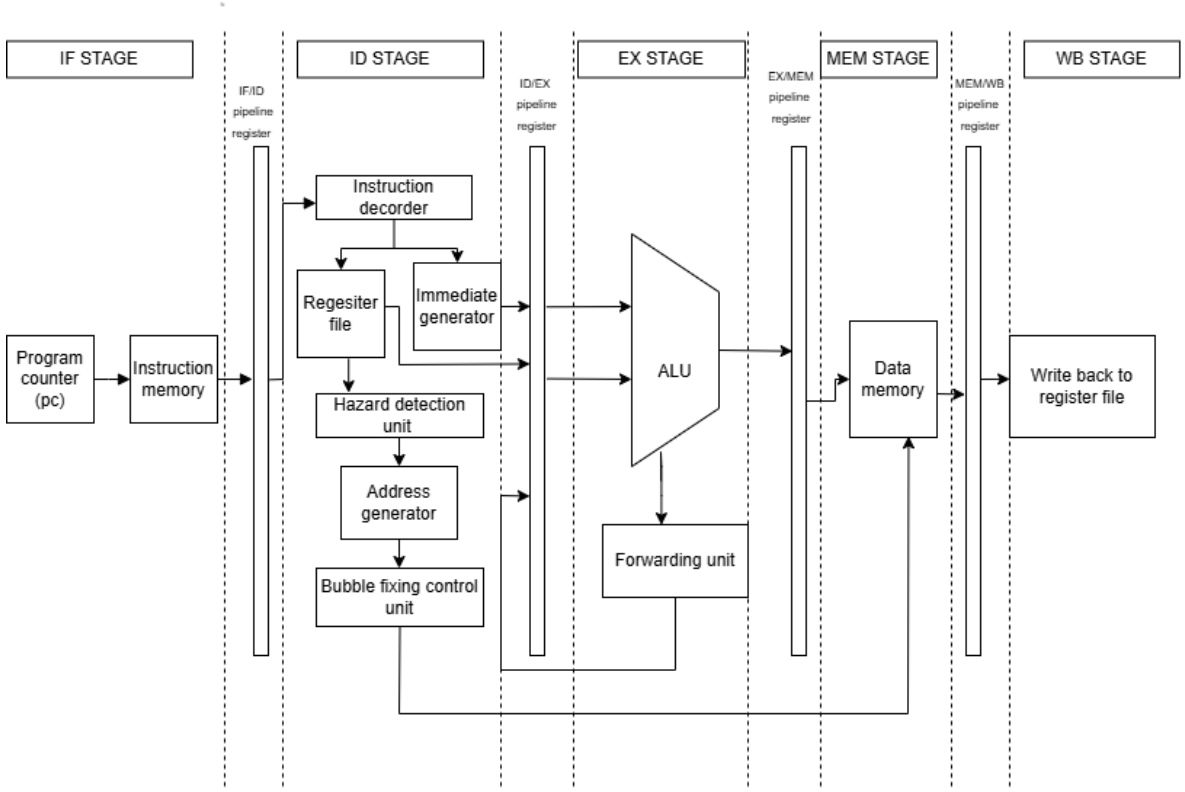


Fig. 2 - Block diagram of the proposed bubble-free five-stage RISC-V processor.

The Hazard Detection Unit responds differently, but it still checks the load's destination register with the subsequent instruction's source registers. A traditional HDU would inject a bubble on a match and freeze a portion of the pipeline. In the suggested approach, the bubble-fixing mechanism, which permits an alternate data channel in lieu of a stall, receives the identification of the LW danger from the HDU. The HDU in Fig. 2 receives feedback from the Data Memory and Write Back outputs while controlling the PC and the instruction decode and control block. By keeping hazard detection and bubble-fixing management separate, the processor may maintain its typical five-stage structure while providing a specialized method for load dependency.

The Address Generator is the main architectural enhancement. In the baseline, the effective address of a load is formed by the ALU in the EX stage from the base register and the immediate offset, so for LW rd, offset(rs1) the memory address is

$$A_{MEM} = R[rs1] + offset.$$

In the suggested architecture, the Data Memory may be addressed without waiting for the ALU result because the AG, which is depicted next to the ALU in Fig. 2, generates this address early and sends it straight to the EX/MEM register. Because early address generation allows data-memory access to begin before the traditional ALU address computation would have finished, it is essential for eliminating the bubble.

The control for this unique LW route is provided by the Bubble Fixing Control Unit. After obtaining the address and control data from the EX/MEM register, the BFCU in Fig. 2 drives the Data Memory access and reports to the HDU. It links hazard detection with early memory access and forwarding by determining when the bubble-fixing process is triggered and producing the multiplexer control signals that choose the LW data path and direct the forwarding multiplexers while a load dependency is present. Its activity manifests as the bubble fix signal in simulation.

The Forwarding Unit provides operands to the ALU right away before they are written back to the register file. By directing data read from the Data Memory to the ALU via the path Data Memory → Forwarding Unit → ALU, the proposed architecture builds upon traditional forwarding, which controls ALU-to-ALU interactions. As a result, the dependant instruction can use the loaded value without waiting for write-back. As a result, the hazard-management logic between the forwarded operands and the ordinary operands from the ID/EX register is used by the ALU at the EX stage to make a choice. The EX/MEM register retains the early-generated address and the bubble-fixing control information in addition to its regular contents as the instruction moves toward the Data Memory. The Data Memory keeps loading and storing data. A LW receives the address generated by the AG, A_{DM} = R[rs1] + Imm, and both the forwarding and write-back routes are able to retrieve the read data. The gap between the LW and its dependant instruction is reduced since the memory access occurs earlier due to the earlier generation of the address. The loaded data is still written into the register file via the path Data Memory → MEM/WB → Register File since neither the MEM/WB register nor the WB stage have changed. The distinction is that write-back simply modifies the architecture registers, whereas forwarding transfers the data to the dependent instruction earlier. The additional AG, BFCU, and multiplexer logic are the approach's cost; the impact on power is discussed in the following section.

3. Results and discussion

Through simulation and synthesis in Xilinx Vivado 2024.2, the performance of the suggested bubble-free processor was compared to the conventional five-stage processor, with a focus on the Artix-7 xc7a200tfbg676-2 device with a clock period of 10 ns (100 MHz). The same six-instruction RV32I program, with machine codes of 0x00500093, 0x00A00113, 0x002081B3, 0x00302023, 0x00002203, and 0x003202B3, was run by both CPUs. The program has only one load-use dependency because the last instruction depends on the value loaded by the previous lw instruction. The expected final value is x5 = 30 (0x1E). The simulated waveforms, execution cycles, pipeline efficiency, throughput, power, and energy of the two architectures are compared.

The simulation waveform of the traditional processor is shown in Fig. 3. In successive cycles, the first three instructions yield the ALU results 0x5, 0xA, and 0xF. The danger detection unit asserts the stall and control_nop signals between 60 and 70 ns when the load-use dependency is detected. Concurrently, a NOP is introduced into the pipeline and pc_write is driven low, keeping the program counter at 0x18. As a result, there is only one bubble in the pipeline, and each subsequent operation is one clock cycle behind.

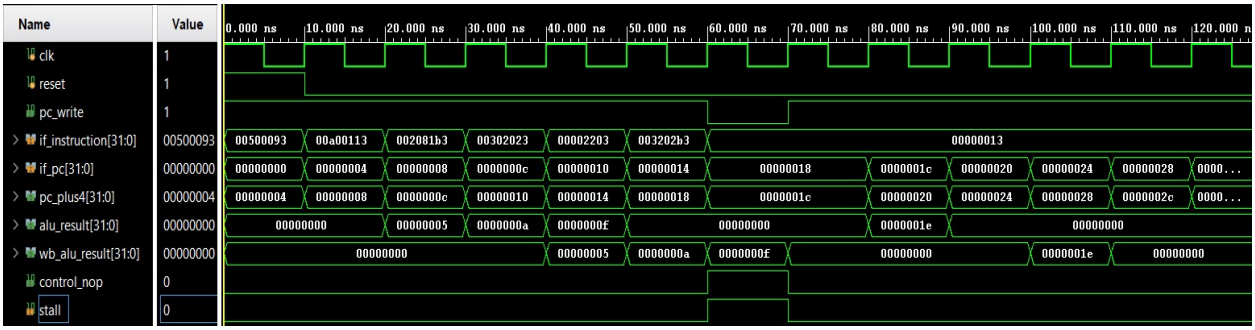


Fig. 3 - Simulation waveform of the traditional five-stage processor (stall and control_nop asserted at 60–70 ns).

The bubble-free processor's waveform for the same program is displayed in Fig. 4. It is confirmed that the additional hardware has no effect on the execution of independent instructions because the ALU results of the first three instructions are the same as those of the conventional CPU. However, the program counter keeps advancing, control_nop stays low, and pc_write stays high during the 60–70 ns interval. Instead of a stall, the bubble fix signal is asserted for one clock cycle, activating the early-address and forwarding channel described in Section 2. Consequently, the dependent instruction proceeds without a bubble, and the correct result 0x1E appears on the ALU output one clock cycle ahead of the traditional processor.

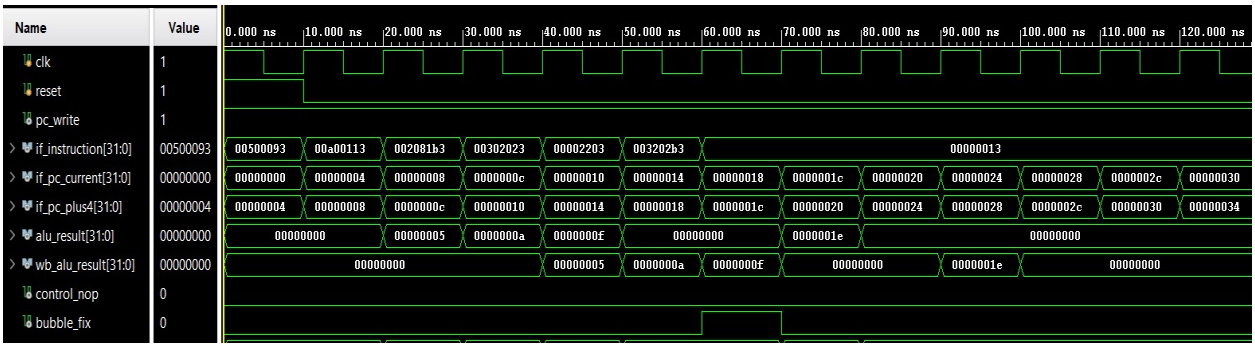


Fig. 4 - The bubble-free processor's simulation waveform (bubblefix asserted, no stall, pc_write remains high).

Table 2 summarizes the main distinctions between the two waveforms. The update maintains functional correctness because the program produces the same architectural result, x5 = 0x1E, in both scenarios. The only distinction is timing: the bubble-free processor substitutes this stall with a single bubble fix pulse and keeps fetching, whereas the conventional processor freezes the program counter and inserts a NOP bubble. Consequently, the final value is generated one clock cycle (10 ns) ahead of schedule.

Table 2 - Key differences between the simulation waveforms.

Observation	Traditional (Fig. 3)	Bubble-free (Fig. 4)
ALU results of I1–I3 (0x5, 0xA, 0xF at 20–50 ns)	Same	Same
pc_write at 60–70 ns	Low (PC frozen)	Stays high
Hazard signals at 60–70 ns	stall and control_nop asserted	control_nop stays low; no bubble
bubble_fix	Not used	Asserted for one cycle

Observation	**Traditional (Fig. 3)**	**Bubble-free (Fig. 4)**
PC moves from 0x18 to 0x1C	At about 80 ns	At about 70 ns
Result x5 = 0x1E on ALU output	From about 80 ns	From about 70 ns (one cycle earlier)

Standard pipeline measures are used to evaluate the two processors' performance [15]. The optimal number of clock cycles for a pipeline with K stages carrying out N commands without stalls is provided by

$$C_{ideal} = N + (K - 1). \quad (1)$$

The actual number of clock cycles and the execution time when Cstall stall cycles are present are

$$C_{actual} = C_{ideal} + C_{stall}, \quad (2)$$

$$T_{exec} = C_{actual} \times T_{clk}, \quad (3)$$

where T_{clk} is the clock period. For N = 6, K = 5 and T_{clk} = 10 ns, (1) gives C_{ideal} = 6 + 4 = 10 cycles for both processors. The traditional processor incurs one load-use stall (C_{stall} = 1), so that C_{actual} = 11 cycles and T_{exec} = 110 ns. The bubble-free processor incurs no stall (C_{stall} = 0), so that C_{actual} = 10 cycles and T_{exec} = 100 ns.

The pipeline efficiency is defined as the ratio of useful instruction cycles to the total number of pipeline cycles, and the relative efficiency is defined with respect to the ideal pipelined execution:

$$\eta = (N / C_{actual}) \times 100\ \%, \quad (4)$$

$$\eta_{rel} = (C_{ideal} / C_{actual}) \times 100\ \%. \quad (5)$$

With N = 6 useful cycles, the traditional processor achieves η = 6/11 × 100 % = 54.55 % and η_{rel} = 10/11 × 100 % = 90.91 %, whereas the bubble-free processor achieves η = 6/10 × 100 % = 60 % and η_{rel} = 10/10 × 100 % = 100 %. The throughput and the number of cycles per instruction (CPI) are given by

$$Throughput = N / T_{exec}, \quad (6)$$

$$CPI = C_{actual} / N. \quad (7)$$

The traditional processor therefore delivers 6 / (110 × 10^{-9} s) = 54.55 MIPS with a CPI of 11/6 = 1.833, while the bubble-free processor delivers 6 / (100 × 10^{-9} s) = 60 MIPS with a CPI of 10/6 = 1.667. The CPI values exceed unity mainly because the test program is short: the four-cycle pipeline fill is spread over only six instructions, and the traditional processor additionally incurs one stall cycle. The relative improvement of the bubble-free processor over the traditional processor in any metric X is defined as

$$\Delta = ((X_{bf} - X_{trad}) / X_{trad}) \times 100\ \%. \quad (8)$$

Applied to the throughput, (8) gives Δ = (60 − 54.55) / 54.55 × 100 % ≈ 10 %.

The power consumption was obtained from the Vivado power analysis. The traditional processor consumes a total on-chip power of 10.367 W (post-implementation estimate: dynamic power 10.193 W, of which 8.994 W is I/O power; device static power 0.174 W; junction temperature 44.4 °C), and the bubble-free processor consumes 10.627 W (post-synthesis estimate: dynamic power 10.451 W, of which 9.599 W is I/O power; device static power 0.176 W; junction temperature 44.9 °C). The bubble-free processor thus draws approximately 2.5 % more power, which is attributable to the additional AG, BFCU and forwarding logic. The total energy consumed, the energy per instruction and the energy efficiency are computed as

$$E = P \times T_{exec}, \quad (9)$$

$$E_{instr} = E / N, \quad (10)$$

$$EE = N / E. \quad (11)$$

For the traditional processor, E = 10.367 W × 110 ns = 1.14037 µJ, E_{instr} = 190.06 nJ and EE = 6 / (1.14037 × 10^{-6} J) = 5.261 MIPS/W. For the bubble-free processor, E = 10.627 W × 100 ns = 1.0627 µJ, E_{instr} = 177.12 nJ and EE = 6 / (1.0627 × 10^{-6} J) = 5.646 MIPS/W. Applying (8) to the energy efficiency gives Δ = (5.646 − 5.261) / 5.261 × 100 % = 7.31 %. Although the bubble-free processor consumes slightly more power, it completes the program sooner and therefore consumes less energy. It should be noted that I/O power accounts for 88 % and 92 % of the total power of the two designs, that Vivado reports a low confidence level for these vectorless estimates, and that the two power values originate from different stages of the design flow. The relative trend is therefore more meaningful than the absolute values.

The two CPUs are compared on all of the aforementioned measures in Table 3. By doing away with the single load-use stall, the bubble-free processor decreases the energy per instruction by about 6.8%, boosts throughput by about 10%, improves pipeline efficiency by 5.45 percentage points, and saves one clock cycle (10 ns). These improvements come at the expense of somewhat increased power consumption and more AG and BFCU hardware. The

presented results are specific to the test program because it only has one load-use dependent. The benefit of the proposed design increases with the number of load-use dependencies in a program because each additional reliance would save one cycle.

Table 3 - Comparison of the traditional and bubble-free processors.

Parameter	Traditional	Bubble-free	Difference
Load-use stall cycles	1	0	−1 (eliminated)
Total clock cycles	11	10	−1 (9.09 % fewer)
Execution time (10 ns clock)	110 ns	100 ns	−10 ns (9.09 % faster)
CPI	1.833	1.667	−0.167
Pipeline efficiency	54.55 %	60.00 %	+5.45 points
Relative efficiency	90.91 %	100 %	+9.09 points
Throughput	54.55 MIPS	60 MIPS	≈ +10 %
Power-analysis stage in Vivado	Implemented netlist	Synthesized netlist	—
Total on-chip power	10.367 W	10.627 W	+0.26 W (+2.5 %)
Total energy (P × T)	1140.37 nJ	1062.7 nJ	−77.67 nJ (−6.8 %)
Energy per instruction	190.06 nJ	177.12 nJ	−12.94 nJ (−6.8 %)
Energy efficiency	5.261 MIPS/W	5.646 MIPS/W	+7.31 %
Load-use hazard handling	Stall + NOP bubble	Early address + forwarding	—
Extra hardware	HDU, FU	HDU, AG, BFCU, extended FU	AG + BFCU added

4. Conclusion

The performance constraints brought on by load-use data hazards in traditional pipelined architectures are effectively addressed by the suggested bubble-free five-stage RV32I RISC-V processor. The proposed architecture eliminates the traditional load-use pipeline stall while preserving the five-stage in-order pipeline structure by integrating an Address Generator (AG), Bubble Fixing Control Unit (BFCU), and an improved Forwarding Unit to enable early load-address computation and direct forwarding of necessary data.

Under the identical operating circumstances as the traditional five-stage RISC-V processor, the CPU was constructed in Verilog HDL and assessed using Xilinx Vivado 2024.2. The findings show an increase in throughput from 54.55 MIPS to 60 MIPS and a decrease in execution time from 110 ns to 100 ns. The suggested architecture delivers a 6.8% decrease in energy consumption and a 7.31% increase in energy efficiency, despite the added hardware causing a 2.5% rise in power consumption.

These findings show that the suggested method effectively strikes a compromise between energy economy, performance enhancement, and hardware overhead.

By reducing needless delays and maintaining proper processor functioning, the suggested bubble-free architecture enhances pipeline usage overall. The findings suggest that pipelined RISC-V processors' performance and energy efficiency can be enhanced by early address creation in conjunction with specialized hazard-control and forwarding techniques. By addressing other pipeline dangers and investigating alternative optimization strategies for better performance RISC-V implementations, the suggested design can be further expanded.

5. References

[1] S. Kumar and K. C. Ray, “Micro-Architecture of LW Driven Bubble-Free Five-Stage Pipelined RISC-V Processor Core for Energy Constraint Low-End Application”, in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 73, no. 1, pp. 207-215, 2026.

[2] A. Sreekumar, B. S. Shankar, and B. N. K. Reddy, “Integrating error correction and detection techniques in RISC-V processor microarchitecture for enhanced reliability”, Integration, vol. 100, 2025.

[3] S. Kumar and K. C. Ray, “An energy-efficient adaptive two stage pipeline RISC-V based soft processor for edge-computing applications”, Int. J. Parallel, Emergent Distrib. Syst., pp. 1–15, 2024.

[4] S. Shukla, P. K. Jha, and K. C. Ray, “An energy-efficient single-cycle RV32I microprocessor for edge computing applications”, Integration, vol. 88, pp. 233–240, 2023.

[5] P. Tsigas and Y. Zhang, “A simple, fast parallel implementation of quicksort and its performance evaluation on SUN enterprise 10000”, in Proc. 11th Euromicro Conf. Parallel, Distrib. Netw. -Based Process., pp. 372–381, 2024.

[6] M. Gautschi et al., “Near-threshold RISC-V core with DSP extensions for scalable IoT endpoint devices”, in *IEEE Transactions on Very Large-Scale Integration (VLSI) Systems*, vol. 25, no. 10, pp. 2700-2713, 2017.

[7] S. Gupta, N. Gala, G. S. Madhusudan, and V. Kamakoti, “SHAKTI-F: A fault tolerant microprocessor architecture”, in *Proc. IEEE 24th Asian Test Symposium (ATS)*, pp. 163-168, 2015.

[8] P. Corner, P. George, A. Sahoo, A. Menon, and V. Kamakoti, "1.1 The SHAKTI processor program," Indian Institute of Technology Madras, Chennai, India, Tech. Rep., vol. 2, no. 3, 2018.

[9] "Orca RV32IM," VectorBlox Computing, GitHub repository. Accessed: Jul. 21, 2024. [Online].

[10] "Mriscv," OnChipUIS, GitHub repository. Accessed: Jul. 21, 2024. [Online].

[11] A. E. Wilson, S. Larsen, C. Wilson, C. Thurlow, and M. Wirthlin, “Neutron radiation testing of a TMR VexRiscv soft processor on SRAM-based FPGAs”, in *IEEE Transactions on Nuclear Science*, vol. 68, no. 5, pp. 1054-1060, 2021.

[12] Z. Mohseni and P. Reviriego, “Reliability characterization and activity analysis of lowRISC internal modules against single event upsets using fault injection and RTL simulation”, *Microprocessors and Microsystems*, vol. 71, 2019.

[13] "Ri5cy: A Small 4-Stage RISC-V Core," PULP Platform, GitHub repository. Accessed: Jul. 21, 2024.

[14] W. P. Kiat, K. M. Mok, W. K. Lee, H. G. Goh, and I. Andonovic, “A comprehensive analysis on data hazard for RISC32 5-stage pipeline processor”, in *Proc. 31st International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, pp. 154-159, 2017.

[15] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design RISC-V Edition: The Hardware Software Interface*. San Mateo, CA, USA: Morgan Kaufmann, 2018.

[16] M. Zulkifli, Y. Yudhanto, N. Soetharyo, and T. Adiono, “Reduced stall MIPS architecture using pre-fetching accelerator”, in *Proc. International Conference on Electrical Engineering and Informatics*, pp. 611-616, 2009.