JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

# Design and Implementation of 32-Bit RISC Processor Using VHDL

**[1]J. Aparna, [2]C. R. K. Gayatri**
[1]M. Tech Student, [2]Assistant Professor
[1]Electronics and Communication Engineering,
[1]SVP College of Engineering, Visakhapatnam, India

***Abstract:*** The design of a high-performance, five-stage, 32-bit microprocessor that operates without interlocked pipeline stages (MIPS) and follows the Reduced Instruction Set Computing (RISC) architecture has presented. The primary objective of a RISC-based microprocessor is to efficiently execute a limited set of instructions, thereby enhancing processing speed. The CPU pipeline is structured into five key stages: Instruction Fetch (IF), Instruction Decode (ID), Execution (EX), Memory Access (MEM), and Write Back (WB). Various functional modules, including the Arithmetic Logic Unit (ALU), Control Unit, Program Counter, Multiplexer (MUX), Instruction Memory, Register File, Sign Extension, and Data Memory, are incorporated to facilitate execution and address calculation. The design is implemented using a Hardware Description Language (HDL) and is developed using Xilinx for constructing an RTL (Register Transfer Level) logic design.

**Index Terms - RISC, MIPS, ISA, CISC, Simulation**

## I. INTRODUCTION

Reduced Instruction Set Computer (RISC) processors, particularly those based on the MIPS architecture, are widely recognized for their simplicity, efficiency, and suitability for embedded and low-power applications. Despite these advantages, conventional processors often spend a significant amount of execution time performing redundant arithmetic operations, especially when operands include zero. Such unnecessary computations increase delay, power consumption, and hardware utilization without contributing to useful results.

This work presents the design and implementation of a 32-bit RISC processor using VHDL, with architectural enhancements in the Arithmetic Logic Unit (ALU) to optimize arithmetic performance. The proposed approach introduces an operand detection mechanism capable of identifying zero inputs early in the execution stage and bypassing redundant computations. For instance, multiplication with zero is resolved instantly without iterative operations, and addition or subtraction involving zero minimizes switching activity in the datapath.

To further strengthen performance, the design also incorporates a streamlined control unit that efficiently manages instruction decoding and execution sequencing in conjunction with the optimized ALU. This synergy ensures that instructions are not only processed faster but with reduced control overhead, maintaining synchronization across pipeline stages while leveraging the benefits of operand-based optimizations.

In addition to ALU-level enhancements, the processor design emphasizes modularity and scalability, enabling easy integration of future extensions such as custom instruction sets or peripheral interfaces. This modular approach not only simplifies testing and verification but also facilitates adaptability across a wide range of applications. Whether deployed in a compact embedded controller or as part of a larger system-on-chip (SoC), the processor maintains consistent performance and power-saving benefits without sacrificing design flexibility.

By integrating these optimizations, the processor achieves reduced execution time, higher throughput, improved FPGA resource utilization, and enhanced power efficiency. Beyond demonstrating a functional 32-bit processor, this project highlights how fine-grained arithmetic-level improvements can translate into overall architectural gains, making the design scalable and effective for embedded systems, digital signal processing, and low-power platforms. Such design strategies reinforce the practicality of customized RISC architectures for performance-critical and resource-constrained applications.

## II. RISC ARCHITECTURE

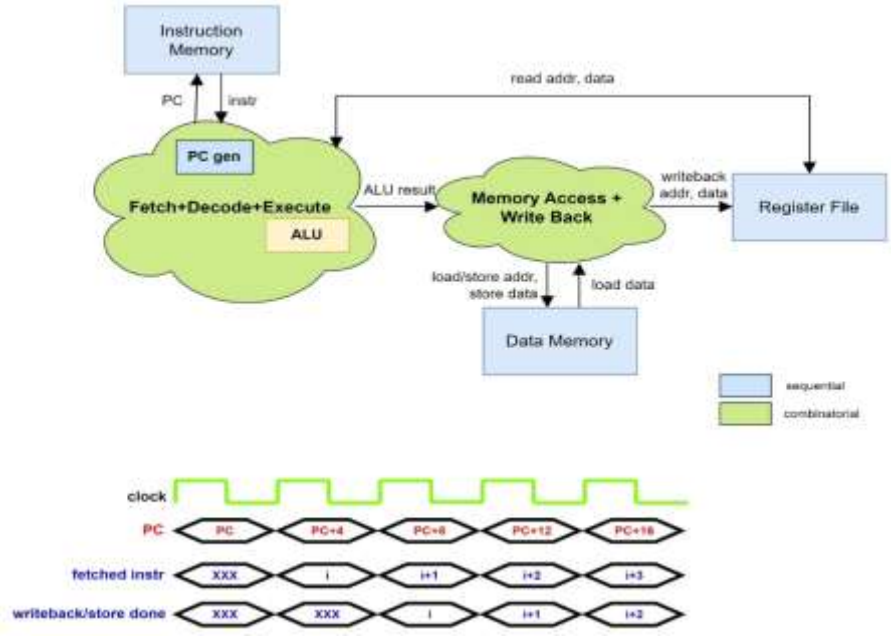


Fig: 2.1 block diagram of RISC architecture

The Reduced Instruction Set Computer (RISC) architecture, introduced in the late 1970s and early 1980s, emphasizes simplicity and efficiency. By using a small, uniform set of instructions that typically execute in a single clock cycle, RISC reduces hardware complexity, improves clock speed, and enables higher instruction throughput. Memory operations are restricted to explicit load and store instructions, while arithmetic and logic operations are performed only on registers, ensuring predictable and streamlined execution. Fixed-length instructions (commonly 32 bits) simplify decoding and support efficient pipelining, usually implemented as a five-stage pipeline: Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory Access (MEM), and Write Back (WB). Large register files minimize memory access delays, and the regular instruction structure allows multiple instructions to be processed in parallel with minimal conflicts.

The efficiency of the RISC design stems from its streamlined approach to instruction handling and execution. By limiting instruction complexity, RISC processors can maintain a high clock frequency and reduce the number of clock cycles per instruction. This approach not only simplifies the control logic but also enables easier pipelining and parallelism, which are critical for achieving high instruction throughput. Additionally, the fixed instruction size enhances the predictability of instruction fetching and decoding, making hardware implementation more straightforward and efficient. These characteristics contribute to the widespread adoption of RISC architectures in various domains, from embedded systems to high-performance computing.

Implementing RISC processors on FPGAs offers distinct advantages, as the architecture aligns well with FPGA resources and design methodologies. The register files are effectively mapped onto the FPGA's block RAMs, while arithmetic and logic units utilize lookup tables (LUTs) and digital signal processing (DSP) slices, optimizing resource utilization. Pipeline registers correspond naturally with flip-flop resources, enabling high-speed operation and efficient clock domain management. Furthermore, incorporating enhancements such as operand detection in the Arithmetic Logic Unit (ALU) can reduce unnecessary computations, decreasing power consumption and improving overall performance. This integration of RISC principles with FPGA technology not only validates the architecture's adaptability but also demonstrates how targeted optimizations at the arithmetic level can translate into tangible architectural benefits.

In this project, a 32-bit RISC processor is designed and implemented using VHDL targeting the Xilinx Spartan-7 FPGA. The design adheres to the classic RISC philosophy, following a load/store execution model and utilizing a fixed 32-bit instruction format alongside a pipelined architecture. Beyond replicating core RISC principles, the processor incorporates arithmetic-level enhancements, such as early zero operand detection within the ALU, to optimize execution efficiency. This combination of a traditional RISC framework with specific performance improvements showcases the continuing relevance and applicability of RISC architectures in modern FPGA-based systems, particularly for embedded applications requiring a balance of speed, power efficiency, and resource utilization.

## III. PROPOSED DESIGN METHOD

This section outlines the architectural design and VHDL-based implementation of a 32-bit RISC-V processor featuring a five-stage pipeline: Instruction Fetch (IF), Instruction Decode (ID), Execution (EX), Memory Access (MEM), and Write Back (WB). The processor integrates key components such as the Program Counter, Instruction Memory, Register File, ALU, Control Unit, and Data Memory. It supports essential instructions including ADD, SUB, AND, OR, ADDI, LW, and J, enabling arithmetic, logical, memory, and control operations. Designed with a modular approach, each component can be independently tested, simplifying verification and debugging.

The Fig.3.1 illustrates the execution flow of the 32-bit RISC-V instruction and $6, $6, $3, which performs a bitwise AND operation between the contents of registers $6 and $3 and stores the result back into register $6. Initially, the Program Counter (PC) holds the address of the current instruction and sends this address to the Instruction Memory (INST_MEM) to fetch the instruction. The PC then increments by 4 using an adder to point to the next instruction in the sequence.

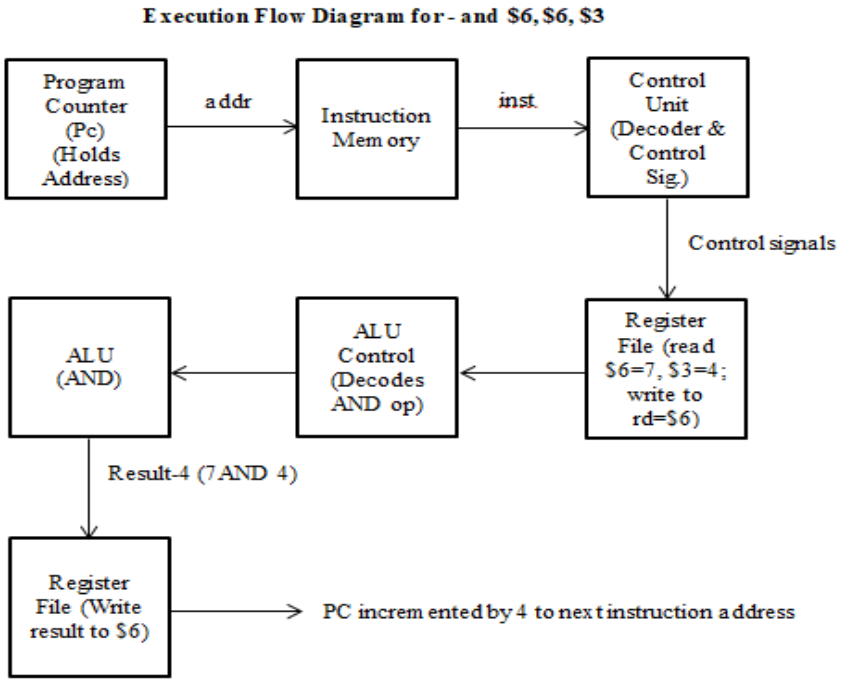


Fig: 3.1 execution flow diagram

The Instruction Memory retrieves the 32-bit instruction corresponding to the address from the PC and sends it simultaneously to the Control Unit and the Register File. The Control Unit decodes the instruction's opcode and function fields to generate the appropriate control signals that direct the processor's datapath components. These signals configure the ALU Control unit to select the AND operation and enable reading and writing of registers.

The Register File reads the contents of source registers $6 and $3. For this instruction, $6 initially contains the value 7, and $3 contains the value 4. These values are passed as operands to the Arithmetic Logic Unit (ALU). The ALU Control unit receives the decoded control signals and instructs the ALU to perform the bitwise AND operation. The ALU processes the inputs 7 (binary 0111) and 4 (binary 0100), computing the bitwise AND result as 4 (binary 0100). Since this is a register-to-register operation, the Data Memory is not accessed during this instruction cycle.

Finally, the ALU result is written back to the destination register $6 in the Register File, updating its value from 7 to 4. This completes the execution of the instruction, and the processor proceeds to the next instruction as indicated by the updated PC.

In essence, this instruction's execution highlights the seamless coordination of the PC, Instruction Memory, Control Unit, Register File, ALU Control, ALU, and Data Memory blocks in efficiently performing the bitwise AND operation within the RISC-V pipeline architecture.

## IV. SIMULATION RESULTS

The 32-bit RISC-V processor design was thoroughly validated through simulation (Fig.3) using the provided VHDL testbench. The testbench generates clock and reset signals to initialize and drive the processor module, while monitoring key outputs such as the program counter (pc_out) and ALU result (alu_result).

One of the key instructions tested was the AND $6, $6, $3, which performs a bitwise AND operation between the contents of registers $6 and $3. In the simulation, registers $6 and $3 were initially loaded with values 7 (binary 0111) and 4 (binary 0100), respectively. During execution, the instruction is fetched from the Instruction Memory using the current PC value, which is then incremented by 4 to point to the next instruction.

The Control Unit decodes the instruction and generates control signals that configure the ALU to perform the AND operation. The Register File reads the specified registers and forwards their values to the ALU. The ALU computes the bitwise AND of 7 and 4, yielding the result 4 (binary 0100). Since this is a register-to-register operation, the Data Memory is not accessed in this cycle.

The computed result is then written back into register $6, updating its stored value from 7 to 4. Throughout this process, the PC continues to increment properly, ensuring the processor fetches the subsequent instructions in sequence. The simulation waveforms confirm correct operation of the instruction execution cycle, with pc_out incrementing by 4 and alu_result accurately reflecting the ALU's output. The reset signal effectively initializes the processor before execution begins. Additional instructions such as ADD, SUB, OR, ADDI, LW, and J were also tested, demonstrating the processor's capability to handle a variety of operations in the RISC-V instruction set.

The successful execution of the AND instruction, along with other operations such as ADD, SUB, OR, ADDI, LW, and J, highlights the functional correctness and versatility of the implemented 32-bit RISC-V processor. Each instruction undergoes the full pipeline process—fetch, decode, execute, memory access (when required), and write-back—demonstrating that the control unit, register file, ALU, and data path components operate cohesively. Furthermore, the timing behavior observed in the simulation waveforms aligns with the expected control signal transitions and data flow across the pipeline stages. This not only validates the instruction handling but also confirms the correct synchronization and sequencing within the processor, reinforcing its readiness for deployment in embedded applications and further development for more complex instruction sets or system integration.

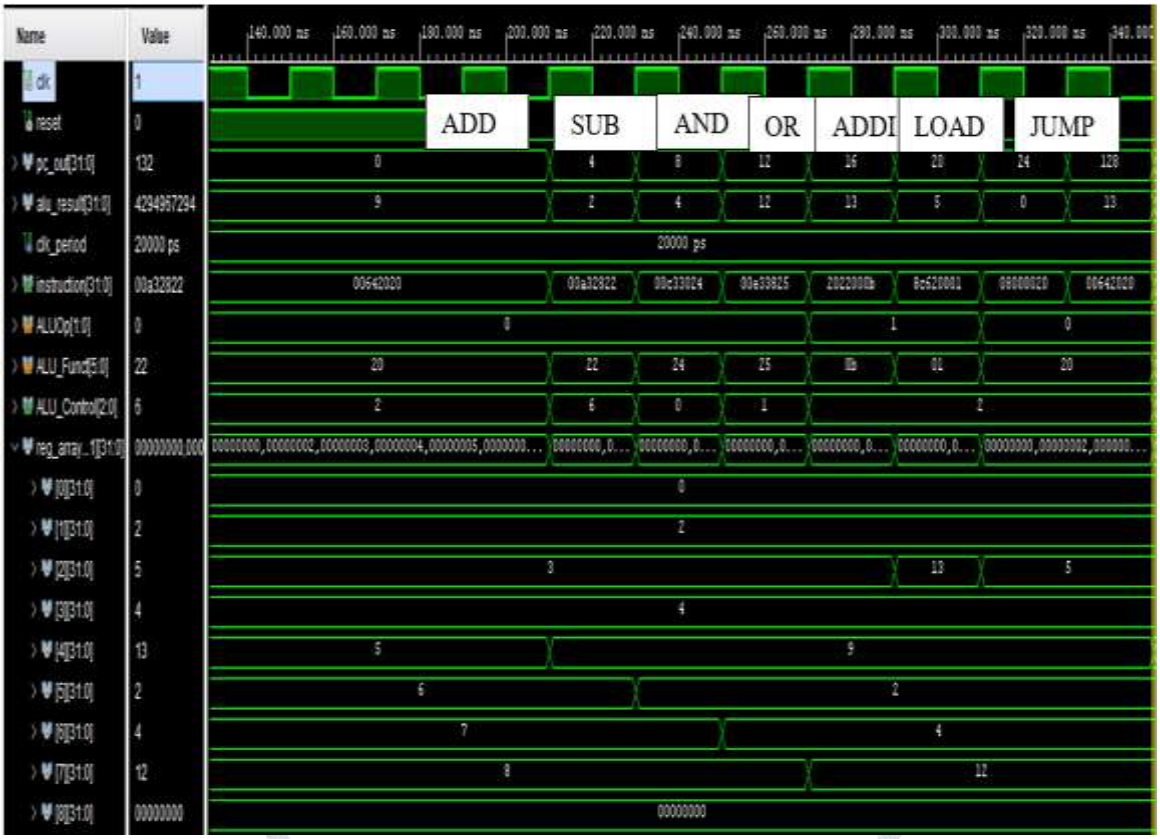


Fig: 4.1 simulation result

This testbench-based simulation verifies that the processor correctly implements the instruction execution flow and data path, providing confidence in the functionality and reliability of the VHDL design.

## V. CONCLUSION AND FUTURE SCOPE

In this project, a 32-bit RISC processor was successfully designed and implemented using VHDL. The processor supports important instructions like ADD, SUB, AND, OR, ADDI, LW, and J, and uses a five-stage pipeline to improve performance. The design was tested through simulation, showing correct execution of instructions such as AND $6, $6, $3.

The project demonstrates how a simple and efficient RISC processor can be built using hardware description language. For future improvements, more instructions can be added, and features like pipeline hazard handling and interrupts can be included. Also, the design can be tested on real hardware like FPGA to check its performance in practical situations.

## VI. REFERENCES

[1] M. Rao, P. Niranjan, and D. Kumar, "Design and implementation of 32-bit RISC-V processor using Verilog," 2024 IEEE International Conference on Distributed Computing, VLSI, Electrical Circuits and Robotics (DISCOVER), 2024, doi: 10.1109/DISCOVER62353.2024.10750638.

[2] B. Poduel, P. Kansakar, S. R. Chhetri and S. R. Joshi, “Design and Implementation of Synthesizable 32-bit Four Stage Pipelined RISC Processor in FPGA Using Verilog/VHDL,” Nepal Journal of Science and Technology, vol. 15, no. 1, pp. 81–88, 2014.

[3] B. Ashok, Y. Bhargavi, V. Sirisha and M. M. G. Krishna, “Design and Implementation of 32-bit RISC Processor Using XILINX,” International Journal of Science, Engineering and Technology (IJSET), vol. 9, no. 4, pp. 362–366, 2021.

[4] N. D. D. Nischitha, K. S. Priya, S. D. Shivani, Y. K. L. Yashaswini and C. M. Patil, “Implementation of 32-bit RISC-V Processor,” International Research Journal of Engineering and Technology (IRJET), vol. 8, no. 5, pp. 1427–1431, 2021.

[5] H. Miyazaki, T. Kanamori, M. A. Islam and K. Kise, “RVCoreP: An Optimized RISC-V Soft Processor of Five-Stage Pipelining,” arXiv preprint arXiv:2002.03568, 2020.

[6] A. Singh, N. Franklin, N. Gaur and P. Bhulania, “Design and Implementation of a 32-bit ISA RISC-V Processor Core using Virtex-7 and Virtex-UltraScale,” 2020 IEEE 5th International Conference on Computing Communication and Automation (ICCCA), Greater Noida, India, pp. 126–130, 2020, doi: 10.1109/ICCCA49541.2020.9250850.

[7] P. Mantovani, R. Margelli, D. Giri and L. P. Carloni, “HL5: A 32-bit RISC-V Processor Designed with High-Level Synthesis,” 2020 IEEE Custom Integrated Circuits Conference (CICC), Boston, MA, USA, pp. 1–8, 2020, doi: 10.1109/CICC48029.2020.9075913.

[8] T. Gokulan, A. Muraleedharan and K. Varghese, “Design of a 32-bit, Dual Pipeline Superscalar RISC-V Processor on FPGA,” 2020 23rd Euromicro Conference on Digital System Design (DSD), pp. 17–24, 2020.

[9] S. Mangalwedhe, R. Kulkarni and S. Y. Kulkarni, “Low Power Implementation of 32-bit RISC Processor with Pipelining,” Proceedings of the Second International Conference on Microelectronics, Computing & Communication Systems (MCCS 2017), Springer Singapore, pp. 467–475, 2019.

[10] V. Jain, A. Sharma and E. A. Bezerra, “Implementation and Extension of Bit Manipulation Instruction on RISC-V Architecture using FPGA,” 2020 IEEE 9th International Conference on Communication Systems and Network Technologies (CSNT), Gwalior, India, pp. 167–172, 2020, doi: 10.1109/CSNT48778.2020.9115759.

[11] R. Höller, D. Haselberger, D. Ballek, P. Rössler, M. Krapfenbauer and M. Linauer, “Open-Source RISC-V Processor IP Cores for FPGAs — Overview and Evaluation,” 2019 8th Mediterranean Conference on Embedded Computing (MECO), Budva, Montenegro, pp. 1–6, 2019, doi: 10.1109/MECO.2019.8760205.
[12] K. Stangherlin and M. Sachdev, “Design and Implementation of a Secure RISC-V Microprocessor,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 30, no. 11, pp. 1705–1715, 2022.
[13] H. An, Y. Kim, J. Lee and S. Kang, “Single Cycle 32-bit RISC-V ISA Implementation and Verification,” arXiv preprint arXiv:2006.04550, 2020.
[14] J. Jeemon, “Pipelined 8-bit RISC Processor Design using Verilog HDL on FPGA,” 2016 IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT), pp. 1012–1016, 2016.
[15] A. R. Al-Ali and M. K. Qureshi, “Design and Simulation of 32-bit Pipelined RISC Processor using VHDL,” International Conference on Computational Intelligence and Computing Research (ICCIC), pp. 311–315, 2017.
[16] Chipmunk Logic, "Designing Pequeño RISC-V CPU from Scratch – Part 2: Specifications and Architecture," Available: https://chipmunklogic.com/digital-logic-design/designing-pequeno-risc-v-cpu-from-scratch-part-2-specifications-and-architecture/.