Asian Journal of Electrical Sciences
ISSN: 2249-6297 (P); 2583-7923 (E)
Vol.14 No.2, 2025, pp.44-52

www.crijournals.org
DOI: https://doi.org/10.70112/ajes-2025.14.2.4300

Design, Implementation, and Functional Verification of a Single-Cycle 8-Bit RISC Processor Based on Von Neumann Architecture

Muskan Anwar[1], Amna Bazzal[2], Muhammad Sohail[3] and Zaryab Basharat[4*]
[1,2,3]Department of Electrical Engineering, University of Engineering and Technology,
Lahore (Narowal Campus), Narowal, Pakistan
[4*]MOE Key Laboratory of Thermo-Fluid Science and Engineering, Xi'an Jiaotong University, Xi'an, China
E-mail: muskananwar629@gmail.com, amnabazzal7@gmail.com, msohail9800@gmail.com
*Corresponding Author: 2020me521@student.uet.edu.pk, engr.zaryab@stu.xjtu.edu.cn
(Received 1 August 2025; Revised 3 October 2025; Accepted 21 October 2025; Available online 2 November 2025)

Abstract **- Computer architecture is fundamentally grounded in the core principles of Computer Organization and Architecture (COA), including digital logic design and the structural configuration of processing units. For students to fully comprehend these concepts, theoretical instruction alone is often insufficient; practical exposure through simulation significantly enhances understanding. One effective method is the development of a simple 8-bit Central Processing Unit (CPU) model that demonstrates how architectural components interact in real time. The proposed design follows the Von Neumann architecture model, incorporating essential elements such as registers, a bus interface, an Arithmetic Logic Unit (ALU), and memory, along with their internal interconnections. The CPU simulation is implemented using Logisim, a digital logic simulation tool that enables users to construct and test circuits efficiently. Logisim supports hierarchical circuit design, allowing small combinational and sequential subcircuits to be integrated into a fully functional CPU within a unified environment. Undergraduate students frequently encounter difficulties when studying COA concepts solely through textbooks and lectures, as traditional teaching methods often describe system architecture without demonstrating the dynamic, step-by-step operational flow. To address this gap, this research presents the systematic design and implementation of an 8-bit CPU in Logisim, inspired by the simplified Mic-1 model. The stepwise construction approach helps students visualize how individual components contribute to overall processor functionality, thereby strengthening conceptual understanding before progressing to more complex, application-specific computer designs.**

Keywords: **8-bit CPU, Logisim Simulation, Von Neumann Architecture, Computer Organization and Architecture, Digital Logic Design**

I. INTRODUCTION

The architecture of an 8-bit RISC (Reduced Instruction Set Computer) processor is designed to maximize computational efficiency while maintaining a simple and fast design [1]. The processor comprises several core components, each with a specific role: the Arithmetic Logic Unit (ALU), Control Unit (CU), Universal Shifter, and Barrel Shift Rotator [2]. These components work together to perform arithmetic, logical, and data manipulation operations on 8-bit operands, enabling the processor to execute a variety of instructions efficiently.

This processor follows a load/store Von Neumann architecture, meaning it uses a single shared memory for both instructions and data. Communication with memory occurs over a single data bus and a single address bus that connect the processor to memory.

In this architecture, both instructions and data are accessed sequentially, which simplifies memory management and reduces latency. Since instructions and data share the same bus, the processor must carefully schedule memory accesses to avoid conflicts and ensure smooth operation. To improve processing speed and throughput, the processor employs three-stage instruction pipelining, enabling several instructions to be processed concurrently across different execution stages [3]. The stages are as follows:

A. Fetch Stage

In this first stage, the processor retrieves the instruction from memory along with any required operand data. The Program Counter (PC) provides the address of the next instruction, which is placed on the address bus. The instruction and data are then transferred to the processor via the data bus. Fetching instructions in advance ensures that the processor always has work to perform, reducing idle cycles.

B. Decode Stage

Once the instruction is fetched, the Control Unit interprets the opcode to determine the operation to be performed. During decoding, the instruction is separated from its operands, and the Control Unit generates the necessary control signals to activate the appropriate hardware components, such as the ALU, Shifter, or Rotator. This stage ensures that the correct registers are selected and that the operation is properly configured before execution.

C. Execute Stage

In this final stage, the actual computation or data manipulation occurs. For arithmetic and logical operations, the ALU takes input operands from Register A and Register B, performs the specified operation (e.g., addition,

subtraction, AND, OR, XOR), and stores the result in the Accumulator or another destination register. For shift and rotate operations, the Universal Shifter or Barrel Shift Rotator manipulates data from Source Register A according to the instruction. The processed result is then written back to the appropriate register or memory location.

The Control Unit acts as the brain of the processor, coordinating all internal activities. It interprets the binary opcodes, produces timing signals, and manages data flow between registers, the ALU, and memory [4]. By issuing precise control signals, it ensures that each stage of the pipeline operates in synchrony, enabling simultaneous execution of multiple instructions at different stages, which significantly enhances overall throughput.

Registers play a crucial role in this architecture. Register A and Register B serve as primary sources for ALU operations, while the Accumulator holds results from arithmetic, logical, and shift/rotate operations. Source Register A specifically provides input for shifting and rotating functions. These registers allow for rapid, intermediate data storage, reducing dependency on slower off-chip memory resources.

Finally, the pipelined structure and modular design of this 8-bit RISC processor ensure high instruction throughput, low cycle time, and simplified instruction decoding, which are key advantages of RISC architectures. By limiting instruction complexity and focusing on a small set of essential operations, the processor achieves efficient, deterministic performance suitable for embedded systems and applications requiring fast and predictable computation.

II. LITERATURE REVIEW

The design and implementation of 8-bit processors have long served as a fundamental area of study in computer architecture and digital systems education [5]. Over the decades, these processors have not only demonstrated their practical utility but have also provided an accessible framework for understanding core concepts in processor design, instruction execution, and digital logic [6].

Early microprocessors such as the Intel 8008 and Intel 8080 were pioneering examples of 8-bit architectures, demonstrating the feasibility of processing 8 bits of data in a single operation. These early designs highlighted essential principles such as the interaction among the Arithmetic Logic Unit (ALU), control logic, registers, and memory, laying the groundwork for the development of more complex microprocessors and teaching tools [7].

Their historical significance lies in illustrating how compact instruction sets, efficient control mechanisms, and basic pipelining concepts can enable functional computation within limited hardware resources.

In terms of pedagogy, multiple studies have underscored the educational value of 8-bit processor design. Ichsan and Kurniawan [8] demonstrated the use of Logisim, a visual digital logic simulation tool, to construct an 8-bit CPU architecture. This approach is particularly effective for undergraduate education because it allows students to visually trace data paths, control unit operations, and instruction decoding, providing an interactive and intuitive understanding of processor workflows. By simulating processor operations in a controlled environment, students can experiment with modifications, observe the impact of control signals, and understand how instructions move through the fetch–decode–execute pipeline without the need for physical hardware.

This hands-on, simulation-based methodology reinforces theoretical knowledge and strengthens practical comprehension of digital logic principles. Further contributions, such as the work by Uma [9], focused on the design of a RISC-based 8-bit processor using Xilinx hardware development tools. This implementation emphasized single-cycle instruction execution and simplified control logic, aligning well with recommended Instruction Set Architecture (ISA) principles for educational CPUs.

The RISC (Reduced Instruction Set Computer) approach reduces instruction complexity, streamlines the datapath, and enables a clear separation of functional units, which simplifies both teaching and implementation. The resulting architecture provides students and engineers with a manageable platform for exploring instruction scheduling, ALU operations, and control signal generation, while also serving as a practical solution for low-resource embedded systems requiring predictable and efficient computation.

Another important contribution is from Trivedi and Asati [10], who explored a modular 8-bit core processor design suitable for small-scale systems and intellectual property (IP) cores. Their work emphasized compact ALU integration, minimized control units, and modular design practices.

Such approaches not only enhance learning outcomes—by clearly demonstrating how processor components can be designed and interconnected—but also show practical applicability for lightweight computational tasks in embedded systems. Their research highlights how modularity and simplicity in 8-bit architectures enable efficient hardware implementation while maintaining instructional clarity.

Overall, these studies reflect a clear consensus in the literature: 8-bit processor projects are highly effective for teaching digital design fundamentals. They provide a balance between conceptual clarity and practical implementation, allowing learners to grasp essential topics such as pipelining, control signal generation, instruction decoding, and data path management.

Tools such as Logisim and Xilinx further enhance this learning experience by offering visual and hardware-based simulation capabilities, reinforcing theoretical understanding while enabling experimentation and hands-on practice.

Consequently, 8-bit processor design remains a cornerstone in computer engineering education, bridging the gap between basic digital logic and more advanced processor architectures.

III. ARCHITECTURE OF 8-BIT RISC PROCESSOR

The architecture of the 8-bit RISC processor is composed of several critical components that collectively enable efficient instruction execution. The Arithmetic Logic Unit (ALU) functions as the central computational component of the processor, executing arithmetic operations such as addition and subtraction, as well as logical operations including AND, OR, and XOR. The Control Unit (CU) manages the overall operation of the processor by interpreting instruction opcodes, generating control signals, and coordinating data flow among the ALU, registers, memory, and other components.

Additionally, the processor incorporates a Universal Shifter and a Barrel Shift Rotator, which enable bit-level manipulation of data, including left and right shifts and rotations, thereby supporting efficient implementation of various arithmetic, logical, and control functions.

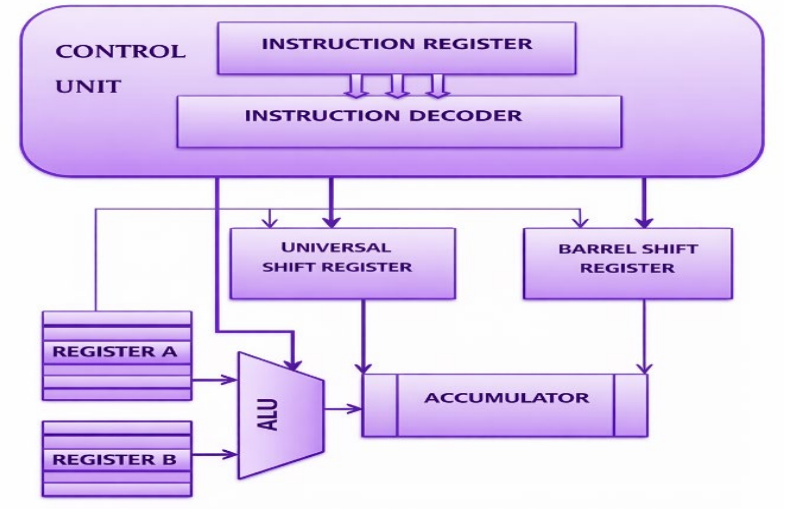

Fig.1 Architecture of 8-Bit Risc Processor

This processor is designed based on the Von Neumann architecture, meaning that a single shared memory stores both instructions and data, and all memory accesses are conducted sequentially through a common data bus and address bus. This sequential access simplifies memory organization and ensures predictable instruction flow, although it requires careful scheduling to avoid memory access conflicts. To further enhance performance, the processor employs a three-stage instruction pipeline consisting of the fetch, decode, and execute stages.

During the fetch phase, the processor retrieves the next instruction along with any necessary operands from memory. In the subsequent decode phase, the Control Unit analyzes the instruction and configures the required components to carry out the execution process.

Finally, during the execution phase, the ALU or the shifter/rotator unit performs the required operation, and the computed result is subsequently written back to the designated destination register or memory location. This pipelined organization enables multiple instructions to progress through different stages concurrently, improving overall throughput while minimizing processor idle time.

The instruction set of this processor follows simplified RISC principles, emphasizing a reduced number of basic operations that can be executed efficiently. The core instructions include LOAD (to move data from memory to registers), STORE (to write data from registers to memory), ADD and SUB (for arithmetic operations), JMP (to alter program flow), and HLT (to halt the processor).

Each of these instructions is designed to execute within a single clock cycle, minimizing instruction latency and simplifying control logic. By adhering to these principles, the processor achieves both operational simplicity and computational efficiency, making it suitable for educational purposes, embedded applications, and small-scale digital systems. The architecture, as shown in Figure 1, visually illustrates the interconnection among the ALU, Control Unit, Shifter, Rotator, registers, and memory, providing a clear representation of the processor's datapath and control flow.

IV. METHODOLOGY OF DESIGN OF 8-BIT RISC PROCESSOR

A. Control Unit

The 8-bit RISC processor is structured around several key design modules, each responsible for specific operations within the processor's datapath and control framework. The Control Unit (CU) forms the core of instruction management and is implemented as a Finite State Machine (FSM) to coordinate all logical, arithmetic, shift, and rotate operations.

The CU relies on two decoders to generate precise control signals for the ALU and shifting units, ensuring that each instruction activates the correct hardware components at the appropriate time.

B. Arithmetic Logic Unit (ALU)

The Arithmetic Logic Unit (ALU) is subdivided into two functional blocks: one responsible for logical operations such as AND, NAND, OR, NOR, XOR, and XNOR, and the other responsible for arithmetic operations, primarily addition and subtraction, based on the control inputs provided by the CU.

C. Universal Shift Register

For bit-level data manipulation, the processor includes a Universal Shift Register capable of performing load, right-shift, left-shift, and hold operations. This module employs 4×1 multiplexers and combinational logic gates to control shifting operations accurately and efficiently.

D. Barrel Shift Rotator

Complementing the shift register, the Barrel Shift Rotator facilitates multi-bit rotational operations using 8×1 multiplexers, allowing rapid rotation of data across multiple positions without requiring sequential shifting.

E. General Purpose Register

The processor also incorporates a General Purpose Register, an 8-bit storage element implemented using D flip-flops and AND gates. It supports synchronous data storage, reset functionality, and rapid read/write access, serving as a flexible workspace for intermediate computations.

F. Instruction Set Format Rules

The processor adheres to strict instruction set format rules to simplify design and maximize efficiency. All instructions are executed in a single clock cycle, with ALU operations consistently involving two operands, and all data transfers between modules occur synchronously, coordinated by the system clock to ensure timing integrity.

G. Processor Top Module Design

At the system level, the top module design integrates all individual components, connecting the Program Counter (PC), Instruction Fetch, Instruction Decode, and Execution Stage, which includes the ALU, Register File, and memory modules. Synchronization of all modules occurs on the positive edge of the clock, ensuring consistent operation and preventing timing hazards. This modular, pipelined, and synchronous design ensures that the processor operates efficiently, executing instructions predictably while maintaining clarity in both educational and practical contexts.

V. PROCEDURE

The development of the 8-bit RISC processor follows a structured and systematic procedure to ensure functional correctness and modular clarity. The process begins with defining the processor specifications, which involves identifying the processor's operational width as 8 bits and specifying the fundamental instruction set, including ADD, SUB, LOAD, STORE, JUMP, and HLT. These instructions form the foundation of the processor's operations, enabling arithmetic computations, data movement between registers and memory, program flow control, and halting of execution.

The next step involves the development of core modules, each serving a dedicated function within the processor. The Arithmetic Logic Unit (ALU) is designed to handle both arithmetic (addition and subtraction) and logical operations (AND, OR, XOR, etc.). The Register File provides temporary storage for operands and results, enabling rapid access during instruction execution.

The Memory Unit stores both instructions and data, following the Von Neumann architecture to support sequential fetch operations. Meanwhile, the Control Unit is implemented to decode opcodes and generate the appropriate control signals that orchestrate the operation of the ALU, registers, shifter, and rotator.

Following module development, a top-level processor design is created by integrating all core components into a single cohesive unit. This integration ensures proper interconnection of the ALU, registers, memory, and control logic, establishing a complete datapath and control flow for instruction execution. Special attention is given to control flow implementation, where the JUMP (JMP) instruction updates the Program Counter (PC) to allow non-sequential instruction execution, and the HALT (HLT) instruction stops PC updates, effectively pausing processor activity.

Once integration is complete, simulation and validation of each module are performed independently to verify their correctness under various test cases. The ALU, shifter, register file, and Control Unit are individually tested for functionality and timing accuracy.

Finally, full system testing is conducted after integration to confirm the correct operation of the processor as a whole, ensuring that all instructions execute correctly, pipelining functions as expected, and data flows reliably through the processor. This stepwise procedure ensures that the design is robust, modular, and fully functional, while also providing a clear educational framework for understanding 8-bit processor architecture and operations.

VI. RESULTS AND DISCUSSION

The implementation of the 8-bit RISC processor was carried out using Verilog HDL, with each functional block modeled as a separate module to ensure modularity, clarity, and ease

of testing. The processor consists of core modules, including the Arithmetic Logic Unit (ALU), Register File, Memory, and Control Unit, all integrated into a top-level processor module that manages instruction fetch, decode, and execution. The design follows the Von Neumann architecture, in which instructions and data share a single memory accessed sequentially.

A. ALU Module

The ALU performs basic arithmetic and logical operations based on a 2-bit control signal (alu_op). As shown in the implementation, the module supports addition (2'b00), subtraction (2'b01), and a pass-through operation (2'b10) for moving data. The ALU operates on 8-bit operands sourced from the accumulator register (acc_out) and memory data (mem_data). The use of a combinational always @(*) block ensures that the ALU output updates immediately in response to changes in operands or control signals, thereby providing real-time computation within a single clock cycle.

B. Register File Module

The Register File module stores intermediate computation results and facilitates communication between the ALU and memory. Implemented using D flip-flops, it allows synchronous data storage with a write-enable signal and reset functionality. When write_en is active, the ALU result is written into the accumulator (acc); when reset is asserted, the accumulator is cleared to zero. This design supports single-cycle updates of register values and ensures deterministic data availability for subsequent instructions.

C. Memory Module

The Memory module provides storage for both instructions and data. It is implemented as a 256 × 8-bit memory array, initialized to zero at the start of simulation. The module supports both read and write operations, controlled by the mem_write signal. Reads are asynchronous through the statement assign data_out = mem[addr];, allowing the fetched instruction or operand to be immediately available to the processor. Write operations occur synchronously on the positive clock edge, ensuring proper timing and avoiding race conditions.

D. Control Unit

The Control Unit (CU) orchestrates the operation of the processor. Implemented as a combinational block, it interprets the 8-bit opcode and generates control signals (alu_src, mem_write, reg_write, alu_op, jump, and halt) for the ALU, register file, and memory. The CU defines the behavior for the instruction set, including LOAD, STORE, ADD, SUB, JMP, and HLT. For example, LOAD activates the register write signal, STORE triggers memory writes, and JMP updates the program counter (PC). The design ensures single-cycle instruction execution and synchronous control of all modules.

E. Top-Level Processor Module

The top-level processor module integrates all components, coordinating the fetch, decode, and execute stages. The Program Counter (PC) provides sequential instruction addresses and is updated at every positive clock edge unless a jump or halt instruction is executed. During each cycle, the instruction is fetched from memory (IR <= mem_data) and decoded by the control unit to activate ALU, memory, or register operations. The ALU performs the requested operation on the accumulator and memory data, and the results are written back to the accumulator or memory as required. This integration demonstrates a fully functioning single-cycle 8-bit RISC processor, with all instructions executing deterministically.

F. Discussion of Results

The processor successfully executes the complete instruction set with deterministic timing. The single-cycle execution model ensures that all instructions, including arithmetic, memory, and control-flow operations, are completed within one clock cycle, thereby simplifying timing analysis and pipeline control. The modular design allows easy verification of individual units, with the ALU, register file, memory, and control unit validated independently before full system integration.

The top-level processor module demonstrates correct PC updates, instruction fetch, execution, and halting behavior, thereby confirming the functional correctness of the designed 8-bit RISC architecture. This implementation highlights the advantages of RISC design principles, including a simplified instruction set, reduced control complexity, and predictable execution timing. The combination of Verilog modeling and modular design provides a strong framework for educational purposes, as well as a practical reference for embedded systems requiring compact and low-latency processors.

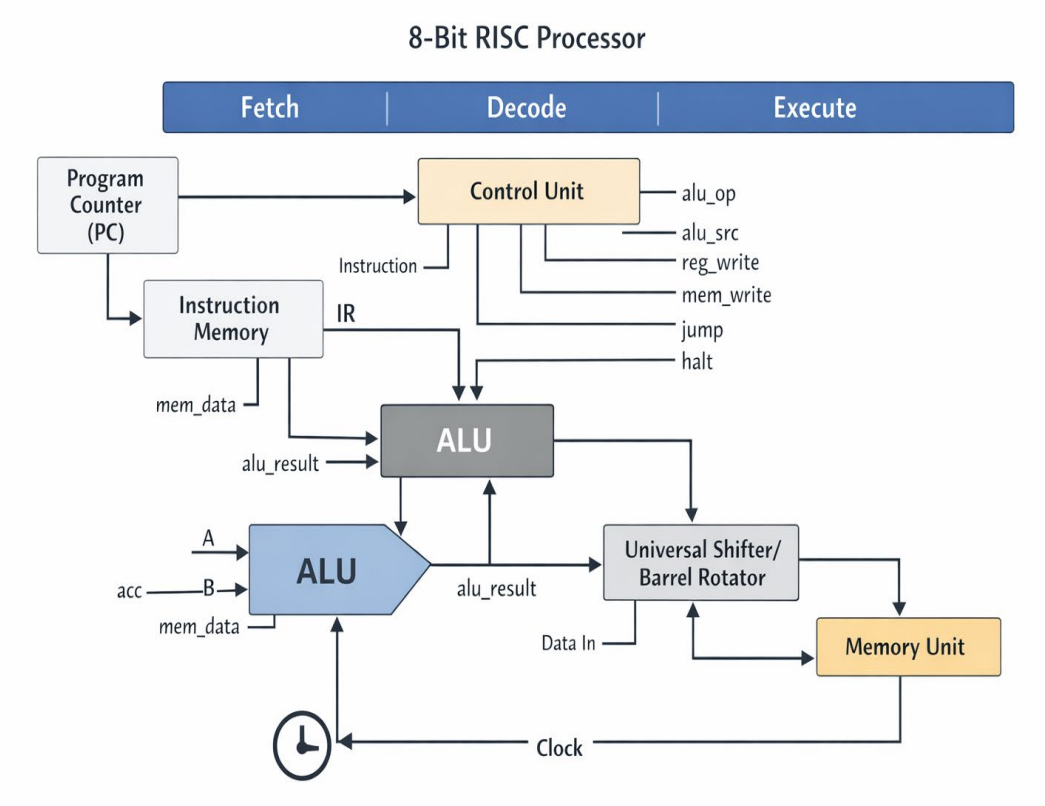

Fig.2 RISC Processor Architecture Flowchart Diagram

VII. PROCESSOR OUTPUT AND SIMULATION

The simulation results validate the correct functional behavior of the designed 8-bit RISC processor under various operational scenarios. Using HDL-based simulation tools, each instruction type—arithmetic, memory, and control flow—was executed and monitored through waveform analysis to ensure proper signal timing and data propagation.

For arithmetic operations, the ALU correctly performed addition (ADD) and subtraction (SUB) based on the control signal (alu_op). The accumulator register updated synchronously on the positive clock edge when the reg_write signal was asserted, confirming proper interaction between the ALU and the register file.

Output waveforms verified that the computed results matched the expected values for different operand combinations, demonstrating the functional accuracy of the combinational and sequential logic. In terms of memory operations, the LOAD and STORE instructions functioned as intended. During LOAD execution, data from the specified memory address was successfully transferred to the accumulator.

For STORE operations, the accumulator value was written back to memory when the mem_write signal was enabled. Simulation waveforms confirmed correct address selection, proper data transfer, and synchronous memory updates without timing conflicts.

For program control instructions, the JUMP (JMP) instruction correctly modified the Program Counter (PC), allowing non-sequential instruction execution. The HALT (HLT) instruction successfully stopped PC incrementation, effectively pausing processor activity. These behaviors were verified through waveform observation of the PC signal across multiple clock cycles.

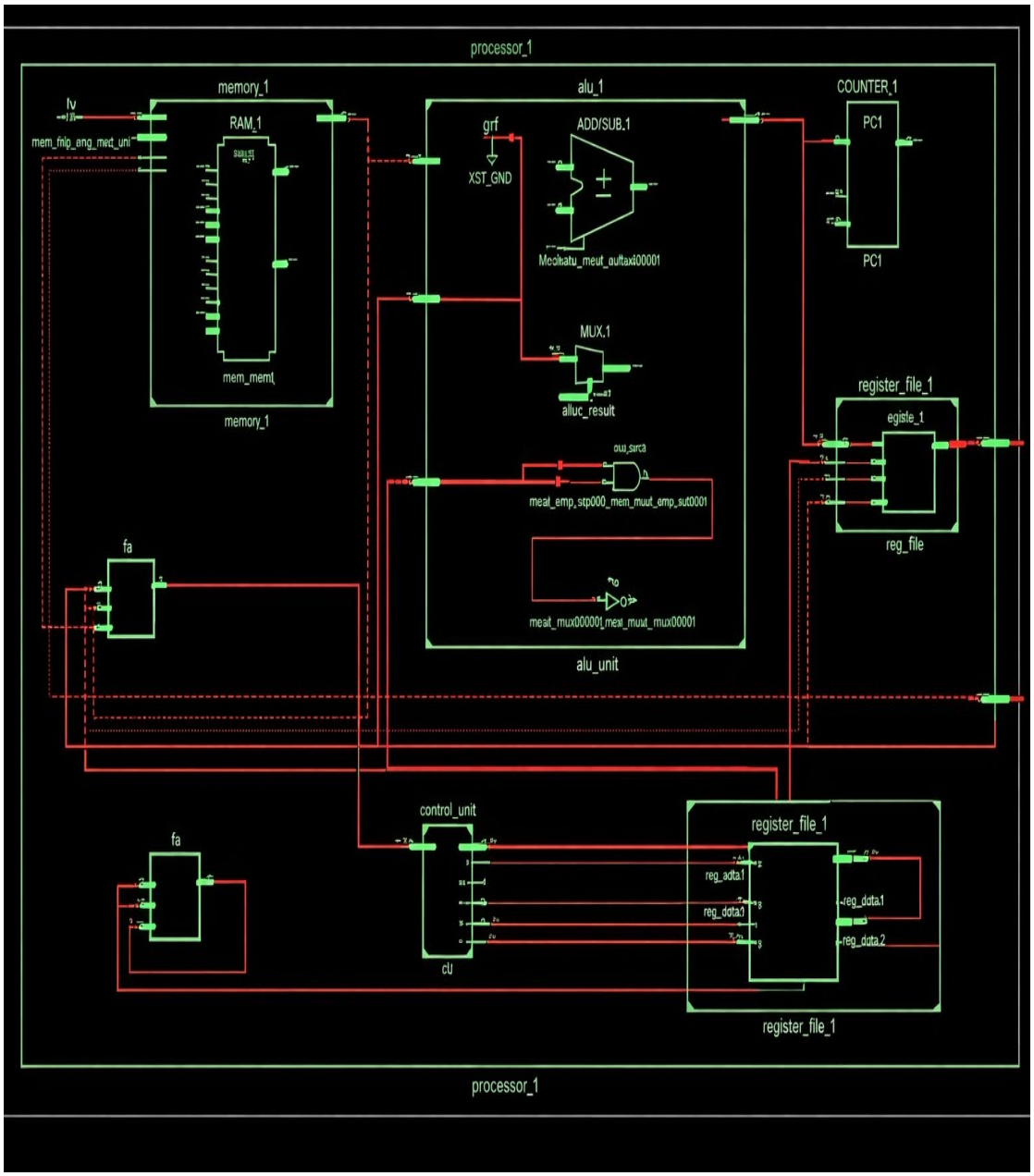

Fig.3 Processor Output

Overall, the simulation confirms that the processor correctly executes the full instruction set within a single clock cycle per instruction, maintaining synchronous operation and deterministic timing. The results demonstrate the successful integration of datapath and control logic, validating the architectural design and functional reliability of the 8-bit RISC processor.

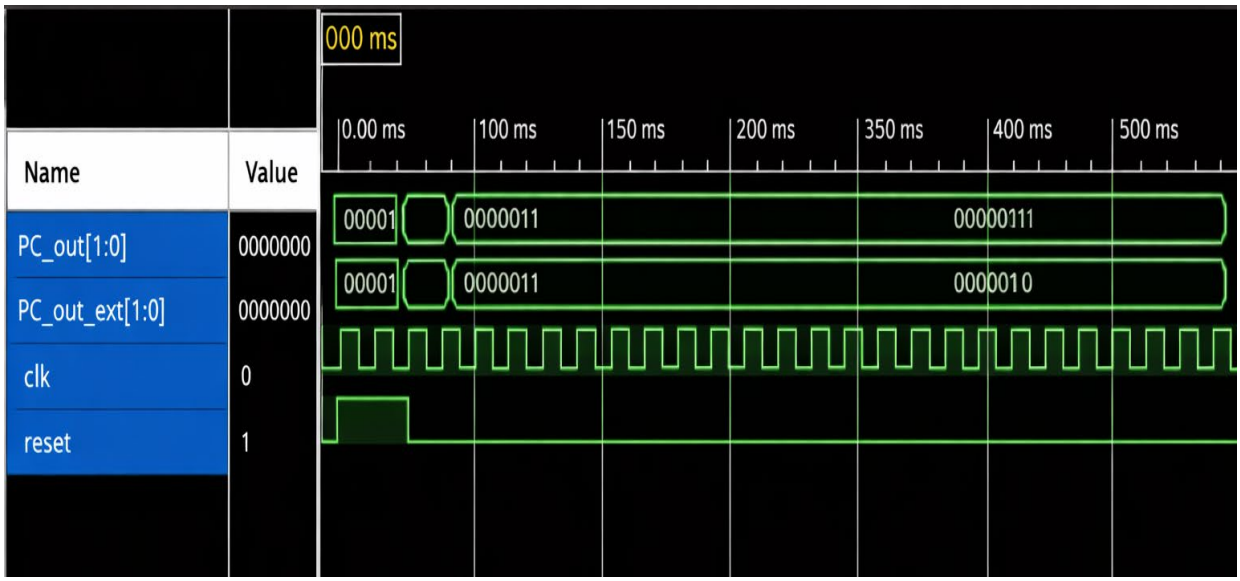

Fig.4 Simulation

.

VIII. CONCLUSION

The design and implementation of the 8-bit RISC processor successfully demonstrate the core principles of computer architecture and digital system design. Through a structured and modular development approach, fundamental components, including the Arithmetic Logic Unit (ALU), Register File, Memory Unit, and Control Unit, were individually designed, verified, and subsequently integrated into a unified processor module. This systematic methodology ensured functional correctness, modular clarity, and efficient hardware organization. The processor adheres to simplified RISC principles, employing a reduced instruction set and single-cycle instruction execution to maintain deterministic timing and minimize control complexity.

The Von Neumann architecture, combined with synchronous clock-driven operation, enables predictable instruction flow and reliable data transfer between modules. The implementation of control flow instructions such as JUMP and HALT further validates the proper coordination between the Program Counter and the Control Unit. Simulation and full system testing confirmed that arithmetic, memory access, and control operations execute correctly within the defined architecture.

The modular Verilog-based design not only demonstrates practical hardware modeling techniques but also reinforces theoretical concepts such as datapath organization, instruction decoding, finite-state control, and synchronous system design. Overall, this 8-bit processor implementation provides a comprehensive educational platform for understanding CPU architecture, bridging the gap between digital logic fundamentals and advanced processor design. It establishes a strong foundation for further exploration into pipelined architectures, advanced instruction sets, and scalable processor systems.

ACKNOWLEDGEMENT

The authors sincerely acknowledge UET Lahore, Pakistan, for providing the facilities and opportunity to carry out this research work.

Declaration of Conflicting Interests

The authors declare no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The authors received no financial support for the research, authorship, and/or publication of this article.

Use of Artificial Intelligence (AI)-Assisted Technology for Manuscript Preparation

The authors confirm that no AI-assisted technologies were used in the preparation or writing of the manuscript, and no images were altered using AI.

ORCID

Muskan Anwar https://orcid.org/0009-0009-5740-5934
Amna Bazzal https://orcid.org/0009-0006-4849-956X
Muhammad Sohail https://orcid.org/0009-0000-2442-1721
Zaryab Basharat https://orcid.org/0009-0008-1914-3331

REFERENCES

[1] S. Hari Venkatesh, N. Vignesh, B. P. Pranav, P. Narun Ram, and G. Saisuriyaa, "8-Bit RISC-V for ALU operation," in *Proc. Int. Conf. Emerging Electronics and Automation*, Singapore: Springer Nature Singapore, Dec. 2024, pp. 429–441.

[2] T. Bräunl, "Central processing unit," in *Embedded Robotics: From Mobile Robots to Autonomous Vehicles with Raspberry Pi and Arduino*, Singapore: Springer Singapore, 2022, pp. 17–52.

[3] V. Lalu, "Design and implementation of 5-stage pipelined RISC-V processor on FPGA," in *Proc. 28th Int. Symp. VLSI Design and Test (VDAT)*, Sep. 2024, pp. 1–6.

[4] M. A. V. B. Raj and L. D. S. K. Andrews, *Computer System Architecture*. Magestic Technology Solutions (P) Ltd., 2023.

[5] D. Vujičić and S. Ranđić, "New challenges in computer architecture education," 2022.

[6] R. Muralidhar, R. Borovica-Gajic, and R. Buyya, "Energy efficient computing systems: Architectures, abstractions and modeling to techniques and standards," *ACM Computing Surveys*, vol. 54, no. 11s, pp. 1–37, 2022.

[7] F. A. Ali and S. Mali, "Fundamentals and advanced concepts of microprocessors," in *Handbook of Semiconductors*. CRC Press, 2024, pp. 137–151.

[8] A. Afrin and M. N. I. M. Nahin Ul Sadad, "SP-2: Extending 4-bit SP-1 CPU with addressing modes and interfacing features on Logisim for computer architecture education," *Journal of Engineering and Applied Science*, vol. 8, no. 2, pp. 90–96, 2024.

[9] T. J. G. da Silva, "Implementation study of a RISC-V based SoC for IoT using SKY130 open PDK," Master's thesis, Universidade NOVA de Lisboa, Lisbon, Portugal, 2022.

[10] P. Trivedi and D. Asati, "A new approach for 8 bit core processor & its IP design for small scale system," *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, vol. 1, no. 7, pp. 121–124, 2012.