Design and Simulation of 16-Bit Pipelined RISC Processor

K. Jagadesh Reddy[1], G. Yashwanthi[2], J. Chandravally[3], J. Gnaneshwar[4], E. Prabhakar[5]
[1,2,3,4]*Dept of ECE, TKR College of Engineering and Technology, Hyderabad, India*
[5]*Assistant Professor, Dept of ECE, TKR College of Engineering and Technology,* Hyderabad, India

***Abstract*—The design of efficient processor architectures is essential for improving performance in modern digital systems. A basic single-cycle 16-bit RISC processor executes one instruction per clock cycle but suffers from low throughput due to long critical paths and inefficient hardware utilization. To address these limitations, this paper presents the design and implementation of a 16-bit pipelined RISC processor based on Harvard architecture. The proposed design employs a 5-stage pipeline consisting of Instruction Fetch, Decode, Execute, Memory Access, and Write Back stages, enabling parallel execution of multiple instructions. Hazard detection and forwarding units are incorporated to handle data and control hazards effectively, ensuring correct execution with minimal performance loss. The processor is modeled using Verilog HDL, verified through simulation, and synthesized using FPGA tools. The results demonstrate improved instruction throughput and efficient resource utilization compared to single-cycle architectures, making the design suitable for embedded and educational applications.**

***Index Terms*—RISC Processor, Pipelining, Harvard Architecture, Hazard Detection, Forwarding Unit, Verilog HDL, Embed-ded Systems.**

I. INTRODUCTION

The rapid advancement of digital technology and the in-creasing demand for high-performance computing systems have led to the development of efficient processor architectures. Modern applications such as embedded systems, communication devices, and real-time processing systems require processors that offer high speed, low power consumption, and efficient resource utilization. Among various processor architectures, Reduced Instruction Set Computer (RISC) architecture has emerged as a widely adopted approach due to its simplicity, predictable execution, and ability to achieve high performance with optimized hardware design.

RISC processors are based on the principle of executing a small set of simple and uniform instructions, each designed to complete in a short and consistent execution time. This approach contrasts with Complex Instruction Set Computer (CISC) architectures, where instructions are more complex and may require multiple clock cycles for execution. By simplifying instruction formats and reducing execution complexity, RISC architectures enable faster instruction decoding, efficient pipeline implementation, and improved overall system performance.

In traditional single-cycle processor designs, each instruction is executed within a single clock cycle, and the duration of this cycle is determined by the slowest instruction. This results in inefficient utilization of hardware resources, as faster instructions must wait for slower ones to complete. Consequently, the overall throughput of the processor is limited, and performance is significantly affected. Additionally, such designs fail to exploit the inherent parallelism available in instruction execution.

To overcome these limitations, pipelining is introduced as a fundamental technique in processor design. Pipelining divides the instruction execution process into multiple stages, allowing different instructions to be processed simultaneously in an overlapping manner. This technique improves instruction throughput by enabling multiple instructions to be in different stages of execution at the same time, similar to an assembly-line operation. A typical pipelined RISC processor consists of stages such

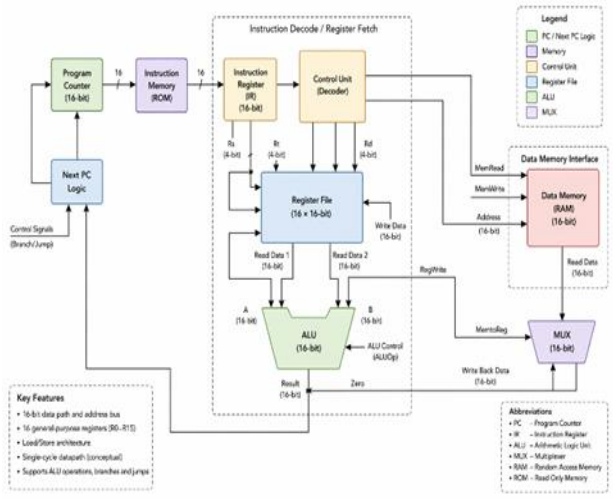

Fig. 1. 16-Bit RISC Block Diagram

as Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory Access (MEM), and Write Back (WB). Each stage performs a specific task and passes intermediate results to the next stage through pipeline registers. While pipelining improves performance, it also introduces challenges such as data hazards, control hazards, and structural hazards, which can affect correct execution if not properly handled.

The design of a pipelined RISC processor requires careful consideration of instruction flow, data dependencies, and efficient hardware organization. Understanding these aspects is essential for developing processors that balance performance, complexity, and resource utilization. In this context, the study of a 16-bit RISC processor provides a simplified yet effective platform to explore fundamental concepts of processor design, including instruction set architecture, datapath organization, and pipelined execution.

This work focuses on the design and simulation of a 16-bit RISC processor with pipelining concepts, aiming to enhance instruction throughput and demonstrate efficient processor operation. The design serves as a practical approach for understanding modern processor architectures and provides a foundation for further advancements in high-performance and scalable computing systems.

II. RELATED WORK

Several research efforts have been carried out in the design and implementation of RISC processors to improve performance, efficiency, and hardware utilization. Early studies on RISC architecture emphasized the use of simple instruction sets and uniform execution cycles to achieve faster processing compared to traditional CISC architectures. These designs focused on minimizing instruction complexity and enabling efficient pipeline implementation.

In [1], a 16-bit RISC processor was implemented using FPGA, demonstrating improved execution speed and reduced hardware complexity through the use of a simplified instruction set. The study highlighted the advantages of load-store architecture and the importance of register-based operations in enhancing performance. In [2], a basic RISC processor design was presented, focusing on instruction set organization and efficient data path design. The work emphasized the role of control unit design in coordinating processor operations.

Further advancements in RISC processor design include pipelined architectures, which significantly improve instruction throughput. In [3], a pipelined RISC processor was developed with emphasis on stage-wise execution and synchronization between pipeline stages. The results showed that pipelining enables multiple instructions to be processed simultaneously, thereby increasing system efficiency. Similarly, in [4], a 5-stage pipelined MIPS-based RISC processor was designed using Verilog HDL, incorporating forwarding and stalling mechanisms to handle data hazards and ensure correct execution.

Recent research has also focused on optimizing processor performance through hardware-efficient designs and improved hazard management techniques. Many implementations include forwarding units, hazard detection units, and pipeline control mechanisms to minimize delays caused by data dependencies and control flow changes.

Despite these developments, most existing designs either focus on higher bit-width processors or involve complex architectures that are not suitable for educational purposes or low-resource systems. Therefore, there is a need for a simple yet efficient pipelined RISC processor that demonstrates core architectural concepts while maintaining good performance and scalability.

III. PROPOSED METHODOLOGY

A. Overview of Proposed Architecture

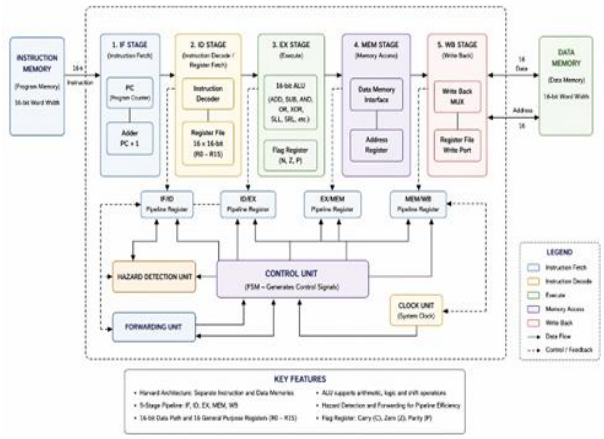

Fig. 2. Architecture of the proposed 16-Bit Pipelined RISC Processor.

The proposed system is a 16-bit pipelined RISC processor based on Harvard architecture, where instruction memory and data memory are separated to enable simultaneous access. As shown in Fig.2 , the processor is organized into a 5-stage pipeline consisting of Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory Access (MEM), and Write Back (WB) stages.

The architecture includes key components such as Program Counter (PC), Instruction Memory, Instruction Decoder, Register File, Arithmetic Logic Unit (ALU), Data Memory,

Control Unit, and pipeline registers. Additional modules such as the Hazard Detection Unit and Forwarding Unit are in-corporated to improve pipeline efficiency and ensure correct execution of instructions.

B. Pipeline Stage Operation

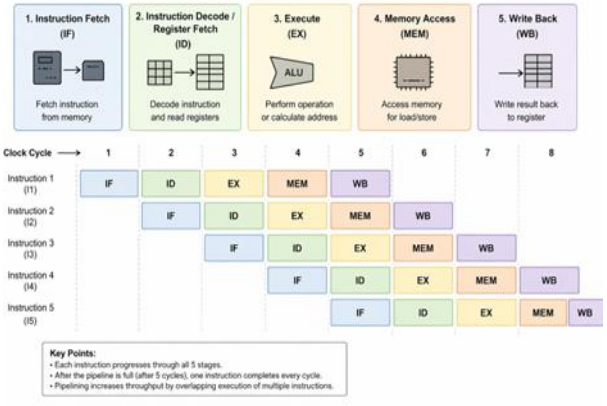

Fig. 3. Pipeline Stages.

The proposed processor employs a 5-stage pipeline architecture consisting of Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory Access (MEM), and Write Back (WB) stages. As illustrated in Fig.3, each instruction progresses through these stages sequentially, while multiple instructions are executed concurrently in different stages of the pipeline.

In the Instruction Fetch (IF) stage, the processor retrieves the instruction from instruction memory using the Program Counter (PC), and the PC is updated to point to the next instruction. In the Instruction Decode (ID) stage, the instruc-tion is decoded to determine the operation, and the required operands are read from the register file. Control signals nec-essary for execution are also generated in this stage.

During the Execute (EX) stage, the Arithmetic Logic Unit (ALU) performs the required arithmetic or logical operation. For memory-related instructions, the effective address is also calculated in this stage. In the Memory Access (MEM) stage, data memory is accessed for load and store operations based on the instruction type. Finally, in the Write Back (WB) stage, the result obtained either from the ALU or memory is written back to the destination register.

The timing diagram in Fig.3 shows how instructions (I1, I2, I3, etc.) are executed in an overlapping manner across successive clock cycles. Initially, the pipeline requires a few cycles to fill. Once the pipeline is filled, one instruction completes execution in every clock cycle. This overlapping of instruction execution increases throughput and improves hardware utilization.

Although pipelining enhances performance, it may intro-duce hazards due to data dependencies or control flow changes. These issues are handled using hazard detection and for-warding mechanisms, ensuring smooth and correct pipeline operation.

C. Hazard Detection and Forwarding

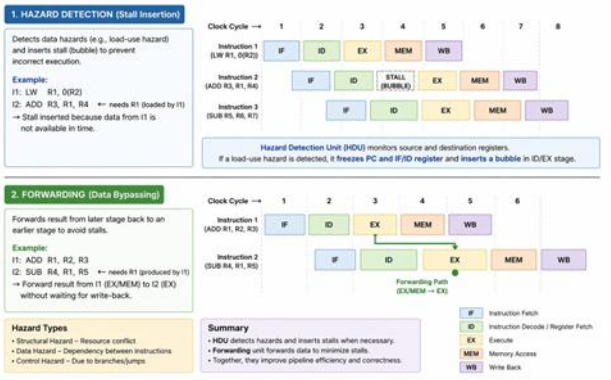

Fig. 4. Detailed Architecture of Hazard Detection and Forwarding.

Pipeline hazards arise when instructions depend on the results of previous instructions or when control flow changes occur. As illustrated in Fig.4, the proposed design handles these issues using a Hazard Detection Unit (HDU) and a For-warding Unit to ensure correct and efficient pipeline execution.

1) Hazard Detection (Stall Insertion)

The Hazard Detection Unit monitors the source and desti-nation registers of instructions in different pipeline stages to identify data hazards, particularly load-use hazards. When a dependent instruction requires data that is not yet available, the HDU introduces a stall (bubble) into the pipeline.

As shown in Fig.4, during a load-use condition, the processor temporarily freezes the Program Counter (PC) and IF/ID pipeline register, while inserting a bubble in the ID/EX stage. This ensures that the required data becomes available before the dependent instruction proceeds, thereby preventing incorrect execution.

2) Forwarding (Data Bypassing)

To reduce performance degradation caused by stalls, a Forwarding Unit is implemented to directly pass intermediate results from later stages to earlier stages. Instead of waiting for the Write Back stage, the result is forwarded from the EX/MEM or MEM/WB stage to the Execute stage of the dependent instruction.

As depicted in Fig.4, the forwarding path allows data to be transferred from the EX/MEM stage of one instruction to the EX-stage of the next instruction. This eliminates unnecessary stalls and improves pipeline efficiency.

3) Hazard Types

The processor may encounter different types of hazards, including data hazards due to instruction dependencies, struc-tural hazards due to resource conflicts, and control hazards caused by branch or jump instructions. The combination of hazard detection and forwarding effectively minimizes these issues.

Overall, the integration of the Hazard Detection Unit and Forwarding Unit ensures correct instruction execution while maintaining high pipeline performance with minimal stalls.

D. Data Path and Memory Organization

The datapath of the proposed 16-bit RISC processor is designed to efficiently transfer data between different func-tional units such as the register file, ALU, and memory modules. It consists of interconnected components including multiplexers, pipeline registers, the ALU, and the register file, which together enable smooth execution of instructions across pipeline stages.

The processor follows a Harvard architecture, where in-struction memory and data memory are separated. Instruction memory stores program instructions and is accessed during the Instruction Fetch stage, while data memory is used for load and store operations in the Memory Access stage. This separation allows simultaneous instruction fetching and data access, reducing memory conflicts and improving overall performance.

The register file consists of sixteen 16-bit general-purpose registers that provide fast access to operands. Data from the register file is supplied to the ALU for computation, and the ALU result is either forwarded to the next stage or written back to the register file. Multiplexers are used to select between different data sources, such as ALU output and memory data, based on control signals.

The datapath also supports efficient data movement between pipeline stages through pipeline registers, ensuring proper synchronization. Overall, the design of the datapath and memory organization enables efficient instruction execution with reduced delays and improved throughput.

IV. RESULTS AND DISCUSSION

A. Simulation Results

The proposed 16-bit pipelined RISC processor is implemented using Verilog HDL and verified

through behavioral simulation. The simulation results confirm correct operation of the processor across all pipeline stages, including instruction fetch, decode, execute, memory access, and write back.

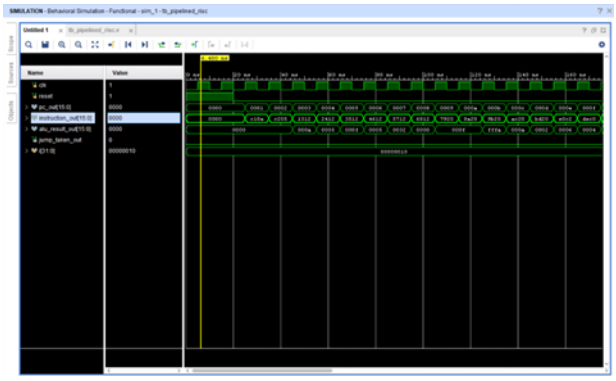

Fig. 5. Simulation Result with Reset ON.

Fig.5 shows the waveform when the reset signal is asserted (reset = 1). During this phase, the processor is initialized to a known state. The Program Counter (PC) is reset to zero, and the instruction output remains at its default value. The ALU output and control signals remain inactive, indicating that no instruction execution takes place. This verifies that the reset mechanism correctly initializes the processor and prevents unintended operations.

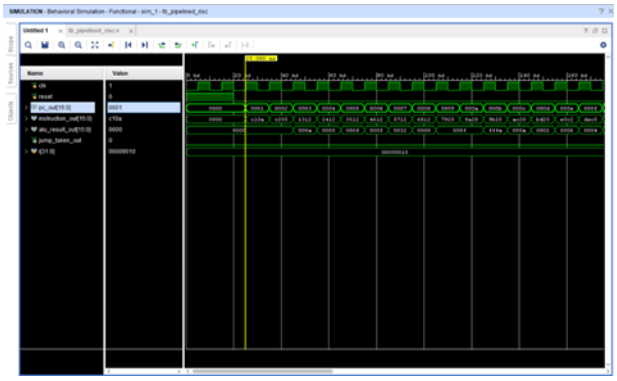

Fig. 6. Simulation Result with Reset OFF.

Fig.6 shows the waveform when the reset signal is de-asserted (reset = 0), where normal processor operation begins. The Program Counter increments sequentially, indicating proper instruction fetching from memory. The instruction output changes according to the stored instruction sequence, and corresponding ALU results are generated. The outputs confirm that arithmetic and logical operations are executed correctly.

The simulation also demonstrates proper pipeline behavior, where multiple instructions are processed simultaneously in different stages. The overlapping execution of instructions across clock cycles verifies correct functioning of pipeline registers and stage synchronization. No incorrect data propagation or timing issues are observed, indicating stable pipeline operation.

B. Performance Analysis

The obtained results validate the functionality of the pro-posed pipelined RISC processor. The reset operation ensures a reliable starting condition, while the sequential increment of the Program Counter confirms correct instruction flow within the system. The correct generation of instruction outputs and ALU results demonstrates that the datapath and control logic are functioning as intended.

Furthermore, the pipeline structure enables continuous execution of instructions, improving throughput compared to a non-pipelined design. The synchronization between stages ensures smooth data transfer, and the design effectively supports concurrent instruction processing.

Overall, the simulation results confirm that the processor operates correctly under different conditions and achieves efficient performance through pipelining. The design is therefore suitable for implementation in embedded and FPGA-based systems.

V. CONCLUSION

In this work, a 16-bit pipelined RISC processor is successfully designed and simulated using a 5-stage pipeline architecture. The implementation of pipelining enables multiple instructions to be processed simultaneously, significantly improving instruction throughput compared to a single-cycle design. The use of Harvard architecture further enhances performance by allowing separate access to instruction and data memory.

The simulation results confirm the correct operation of the processor under both reset and normal execution conditions. The proper increment of the Program Counter, correct instruction decoding, and valid ALU outputs demonstrate the effectiveness of the datapath and control logic. The pipeline stages operate in a synchronized manner, ensuring smooth data flow across the system.

Overall, the proposed design achieves a good balance between simplicity and performance, making it suitable for embedded systems and educational applications. The processor also provides a strong foundation for further enhancements such as advanced hazard handling, increased word size, and integration with memory hierarchy.

REFERENCES

[1] R. D. Komati, A. Kolekar, K. Kasbekar, and A. Godbole, "Design and Implementation of 16-Bit RISC Processor on FPGA," *International Journal of Creative Research Thoughts (IJCRT)*, vol. 8, no. 6, 2020.

[2] S. Gaonkar and A. M., "Design of 16-bit RISC Processor," *International Journal of Engineering Research & Technology (IJERT)*, vol. 2, no. 7, 2013.

[3] S. B. Sudha, G. K. Sneha, S. Shashidhar, and A. S., "Design and Physical Synthesis of a 16 Bit RISC Processor," *International Research Journal of Modernization in Engineering, Technology and Science (IRJMETS)*, vol. 5, no. 6, 2023.

[4] A. C. S. Ankitha and K. V., "Design and Development of a 5-Stage Pipelined RISC Processor based on MIPS," *International Research Journal of Engineering and Technology (IRJET)*, vol. 9, no. 10, 2022.

[5] S. R. Vaidya and P. S. Patil, "Design and Implementation of 16-bit RISC Processor Using Verilog HDL," *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, vol. 4, no. 5, 2015.

[6] M. A. Khan and S. A. Hussain, "FPGA Implementation of 16-bit RISC Processor Architecture," *International Journal of Computer Applications (IJCA)*, vol. 98, no. 12, 2014.

[7] N. Sharma and R. Gupta, "Design of Pipelined RISC Processor Using Verilog," *International Journal of Engineering Science and Computing (IJESC)*, vol. 6, no. 6, 2016.

[8] P. K. Sahu and A. Verma, "Implementation of 5-Stage Pipeline RISC Processor on FPGA," *International Journal of Innovative Research in Science, Engineering and Technology (IJIRSET)*, vol. 5, no. 4, 2016.

[9] R. K. Singh and V. Tiwari, "Design and Simulation of RISC Processor Using VHDL," *International Journal of Electronics and Communication Engineering*, vol. 7, no. 3, 2015.

[10] A. Sharma and D. Singh, "Design of High Performance RISC Processor with Pipeline Architecture," *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering (IJAREEIE)*, vol. 5, no. 7, 2016.

[11] K. Patel and M. Shah, "Design and FPGA Implementation of 32-bit RISC Processor," *International Journal of Scientific & Engineering Research (IJSER)*, vol. 6, no. 9, 2015.

[12] V. R. Kumar and S. K. Rao, "Design of RISC CPU Using Verilog HDL and FPGA Implementation," *International Journal of VLSI Design & Communication Systems (VLSICS)*, vol. 3, no. 3, 2012.

[13] P. Mishra and N. Dutt, "Processor Architecture Design Using Hardware Description Languages," *IEEE Design & Test of Computers*, 2008.

[14] A. Agarwal and M. Lang, "Implementation of Pipelined Processor for High-Speed Applications," *International Journal of Computer Science and Information Technologies (IJCSIT)*, vol. 5, no. 2, 2014.