Communications on Applied Nonlinear Analysis
ISSN: 1074-133X
Vol 32 No. 3 (2025)

Optimal Design for High Performance Computing Systems Based on 5-Stage Pipeline 32-bit RISC-V Processor

Rajender Kumar[1], Sneha Bhattacharya[2]

[1] Assistant Professor, Electronics and Communication Engineering Department, NIT Kurukshetra. India.

[2] M. Tech Student, VLSI Design Department, NIT Kurukshetra. India.

Article History:

Received: 19-03-2025

Revised: 24-07-2025

Accepted: 21-09-2025

Abstract:

In an era where computational demands continually escalate, the quest for more efficient and powerful processors persists. Computer engineering and VLSI design industries are facing challenges with the trade-offs between the cost and performance of components in the implementation domain. Reduced Instruction Set Computer (RISC) architecture relies its focus mainly on scaling down the complexity and the number of instructions in the microprocessor. RISC-V is an Open-source Instruction set architecture (ISA) designed to be simple, modular, and customizable. The most important feature of RISC is that it supports load-store architecture. With this feature, an optimized 32-bit microprocessor has been designed with Verilog, simulated and synthesized in Xilinx Vivado. Verilog enables us to describe the behavior and structure of our processor at a register-transfer level. Overall, RISC-V's combination of simplicity, openness, and flexibility positions it as a promising ISA for a wide range of applications, from low-power IoT devices to high-performance computing systems.

Keywords: RISC-V, Open-source, Instruction set architecture (ISA), load-store architecture, Xilinx Vivado.

1. Introduction

1.1 Background

In the realm of digital design, we embark on a technical exploration aimed at crafting a 5-stage pipeline 32-bit RISC-V processor. At the core of our endeavor lies the 5-stage pipeline architecture, a cornerstone of modern processor design. RISC architecture was introduced with a basic instruction set that had consistent execution times. The added support to load/store architecture makes communication with memories faster and reliable. RISC helps achieve lesser silicon consumption when compared to other design architectures.

The pipeline architecture consists of five stages: Instruction fetch (IF), Instruction decode (ID), Execute (EX), Memory access (MEM), and Write-back (WB). Each stage is finely tuned to maximize throughput while minimizing latency. The pipeline incorporates various hazard detection and resolution mechanisms, including forwarding and stall insertion, to ensure correct execution of instructions.

1.2 Problem Statement

This paper presents the design and implementation of a 5-stage pipeline 32-bit RISC-V processor, adhering to the specifications of the RISC-V Instruction set architecture (ISA). The processor architecture is optimized for performance, area efficiency, and power consumption. The design is based on a modular approach, facilitating easy integration of additional

components and extensions.

2. Objective

The primary objective of this paper is to design and analyze a 5-stage pipelined 32-bit RISC-V processor with an emphasis on efficient instruction execution and program counter management. The work focuses on:

- Implementing the instruction execution flow consisting of instruction fetch, decode, execution, ALU operations, and program counter update.
- Demonstrating the role of multiplexers, control logic, and state elements in ensuring correct data flow and instruction sequencing.
- Highlighting how the Program Counter (PC) is updated either sequentially (PC + 4) or through branch offsets (PC + X), with sign-extension and left-shift operations for correct address alignment.
- Ensuring proper utilization of the ALU across instruction classes (R-type, I-type, and memory instructions), except in jump operations.
- Showcasing the benefits of the RISC pipelined architecture in achieving faster and efficient execution by overlapping instruction stages.

3. Instruction Execution in RISC

The execution of each instruction in a RISC processor follows a well-defined sequence of steps, ensuring consistent and efficient operation. The detailed stages are outlined below:

3.1 Instruction Fetch

- The **Program Counter (PC)** is sent to the memory where the code is stored.
- The instruction present at that memory address is fetched.

3.2 Instruction Decode / Register Read

- Based on the instruction fields, required registers are read.
- For a **load word (lw)** instruction, only one register is needed.
- In most other instructions, two registers are read.

3.3 Execution by Instruction Class

- Once register values are read, execution proceeds according to the instruction class.
- RISC has a simple and uniform instruction classification, allowing proper execution.
- All instructions use the **ALU** for execution, except the **JUMP** instruction.

3.4 ALU Operations

- After register values are read, they are sent to the **ALU**.
- The ALU performs different tasks depending on the instruction type:
  - **I-type instructions (Immediate type):**
    - ALU performs operations such as addition.
    - The result is stored in the destination register.
  - **R-type instructions (Register type):**
    - ALU performs arithmetic or logical operations using register values.

- **Memory instructions (lw, sw):**
  - ALU calculates the **memory address** for load/store operations.

3.5 Program Counter Update

- The next instruction address is calculated.
- This comes from one of the two **adders** (as shown in Fig. 1).
- The updated PC value is sent back to **instruction memory** for fetching the next instruction.

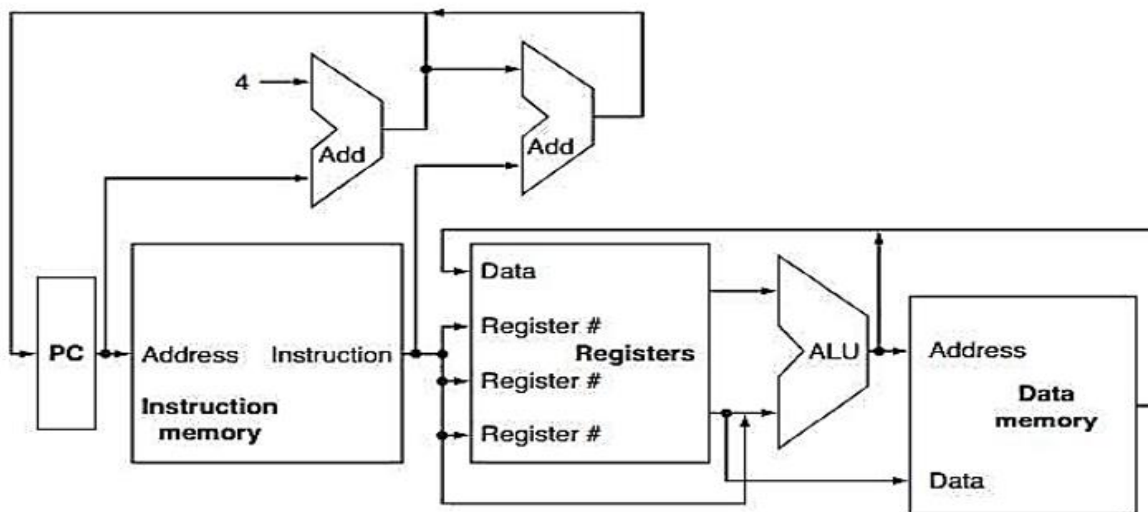

Fig.1 An abstract view of the implementation of the RISC-V subset showing the major functional units and the major connections between them

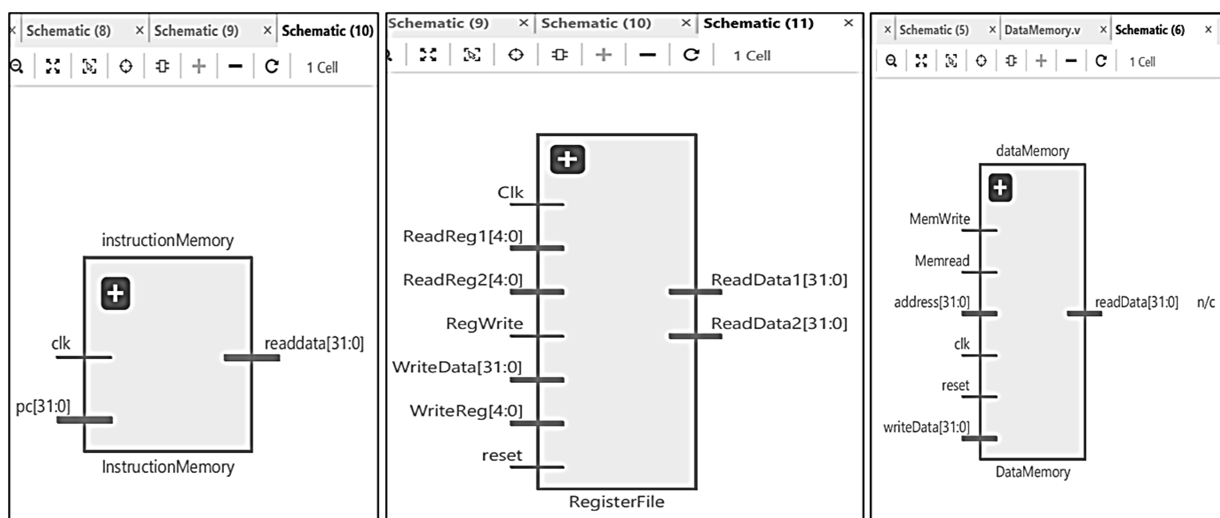

Fig.2 Schematic of RISC-V subset

3.6 Role of Multiplexers, Control Unit, and State Elements in RISC Execution

The top multiplexer determines what should be the next address that has to be fetched. Either it could be PC+4 or PC+X where X determines the sign extended address location. The middle multiplexer determines whether the input to the ALU is the second source register or the sign extended value which is used in lw, sw instructions or in branch instructions. The output multiplexer selects whether to output the data back to the instruction memory or the address in

the data memory which is to be given to the instruction memory. All the select lines are to be controlled by the control unit.

Instruction memory and the Program counter are the state elements. The Instruction Memory can be considered as a combination block. The Program Counter is a 32-bit block which is used to calculate the next instruction to be fetched.

Fig.3 The basic implementation of the RISC subset, including the necessary multiplexors and control lines.

3.7 Address Calculation in RISC processor

The RISC processor is a 32-bit processor hence all its addresses will be in multiples of 4. For calculating the next instruction bit, the sign extended value can be multiplied by 4 or left shifted by 2 to find the appropriate address by adding with the PC. It is known that the offset has been sign-extended from 16 bits, the shift will give away only sign bits. Control logic is used to find whether the pc should be added by PC+4 or PC+X where X is the sign extended value.

Fig.4 Datapath for branch instruction showing ALU-based condition evaluation and branch target calculation using an adder with PC+4 and the shifted, sign-extended offset

Communications on Applied Nonlinear Analysis
ISSN: 1074-133X
Vol 32 No. 3 (2025)

ALU Control

The Arithmetic Logic Unit (ALU) plays a central role in the execution of RISC instructions, performing a variety of arithmetic and logical operations depending on the instruction type:

3.8 Arithmetic and Logical Operations

- For instructions such as addition, subtraction, and logical operations, the ALU directly executes the operation and forwards the result to the data memory or destination register.

3.9 Load and Store Instructions

- In the case of load (lw) and store (sw) instructions, the ALU is responsible for calculating the effective memory address.
- This is achieved by adding the sign-extended immediate value to the base register value obtained from the instruction.

3.10Branch Instructions

- For branch instructions, the ALU evaluates the specified condition.
- If the condition is satisfied, the ALU computes the target address and updates the Program Counter (PC) to jump to the branch location.

3.11ALU Control Mechanism

- The ALU is governed by a 4-bit ALU control signal, which determines the specific operation to be performed.
- Among these control bits, the last two bits are designated as the ALUOp, which provide higher-level instruction classification (e.g., R-type, branch, memory access).
- For R-type instructions, the exact ALU operation is further refined using the 6-bit function (funct) code from the instruction field.

ALU Control Lines	Function
0000	AND
0001	OR
0010	Add
0110	Subtract
0111	Set on less than
1100	NOR

Instruction opcode	ALUOp	Instruction operation	Funct field	Desired ALU action	ALU control input
LW	00	Load word	XXXXXX	Add	0010
SW	00	Store word	XXXXXX	Add	0010
Branch Equal	01	Branch Equal	XXXXXX	Subtract	0110
R-Type	10	Add	100000	Add	0010
R-Type	10	Subtract	100010	Subtract	0110
R-Type	10	AND	100100	AND	0000
R-Type	10	OR	100101	OR	0001
R-Type	10	Set on less than	101010	Set on less than	0111

Fig.5 ALU control signal generation based on ALUOp control bits and function codes for R-type instructions

4. Main Control Unit

The control unit is responsible for generating all control signals required for instruction execution. These signals are primarily determined by the opcode field of the instruction (bits 31–26), with the exception of the PCSrc control line, which is influenced by the branch control logic. The PCSrc line is specifically used to select the branch predictor output.

5.1 Inputs to the Control Unit

- The opcode field (6 bits) is provided as input to the control unit.
- Based on this input, the control unit generates the necessary control signals to regulate the datapath.

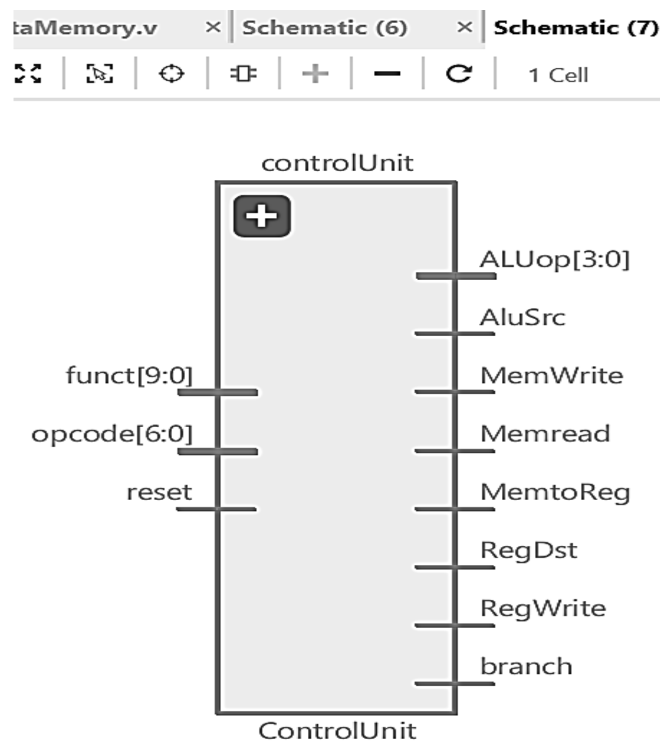

Fig.6 Schematic of the control unit.

5.2 Control Signals

A total of eight control signals are produced, which serve as inputs to different components of the processor. These include:

- RegWrite: Determines whether a value is written into the register file.
- ALUOp: Directs the ALU control block to perform the required arithmetic or logical operation.
- MemRead / MemWrite: Control signals for data memory, selecting whether to read from or write into memory.
- MemToReg: Select signal for the multiplexer that decides whether the data to be written back to the register comes from the ALU or from memory.
- ALUSrc: Select signal that determines whether the ALU's second input is taken from a register or a sign-extended immediate value.
- PCSrc: Selects whether the next PC value is PC + 4 (sequential execution) or PC + X (branch

execution).

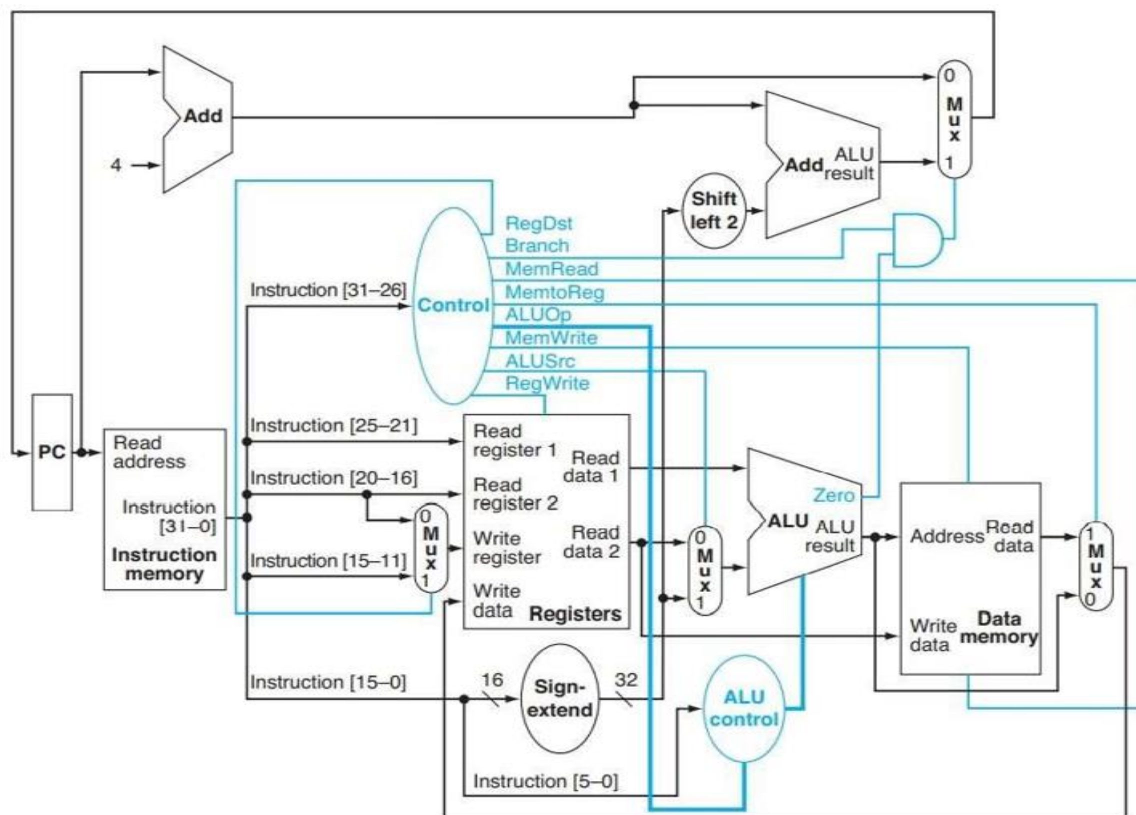

Fig.7 The simple Datapath with the control unit.

5. Pipelined Datapath And Control

Pipelining is a fundamental design technique in RISC architectures, enabling parallel execution of instructions by dividing the instruction cycle into distinct stages. In a 5-stage RISC pipeline, multiple instructions are executed simultaneously, with each stage processing a different instruction. This significantly reduces the overall execution time for a sequence of instructions.

The five stages of the RISC pipeline are as follows:

5.1 Instruction Fetch (IF)

- The instruction fetch stage retrieves the instruction from instruction memory using the address stored in the Program Counter (PC).
- The PC is then updated (typically PC + 4) to point to the next instruction.

5.2 Instruction Decode and Register File Read (ID)

- The fetched instruction is decoded to determine its type and operands.
- The required registers are read from the register file.
- This stage also prepares data for possible write-back to registers.

Communications on Applied Nonlinear Analysis
ISSN: 1074-133X
Vol 32 No. 3 (2025)

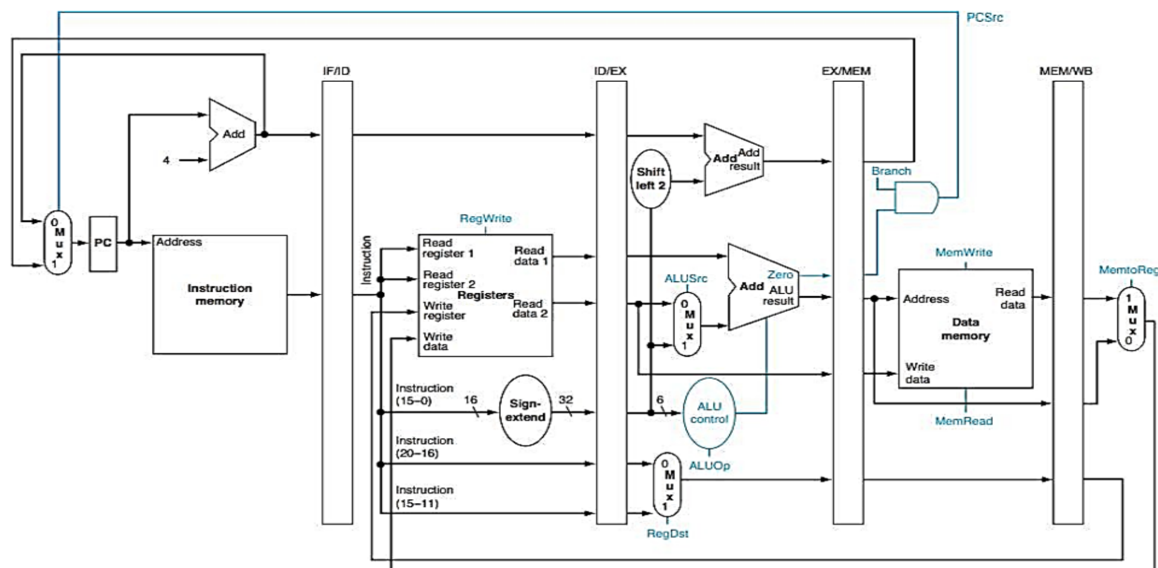

Fig.8 The pipelined Datapath with the control signals identified

5.3 Execution or Address Calculation (EX)

- The ALU performs arithmetic or logical operations as defined by the instruction opcode.
- For memory-related instructions, the ALU calculates the effective address by adding the sign-extended immediate value to the base register.

5.4 Memory Access (MEM)

- If the instruction requires memory access, the data memory is addressed using the result from the ALU.
- For load instructions (lw), data is read from memory.
- For store instructions (sw), data is written into memory.

5.5 Write-Back (WB)

- The final stage updates the register file with the result of execution.
- For arithmetic/logical instructions, the ALU result is written back.
- For load instructions, the value fetched from memory is written back to the register.

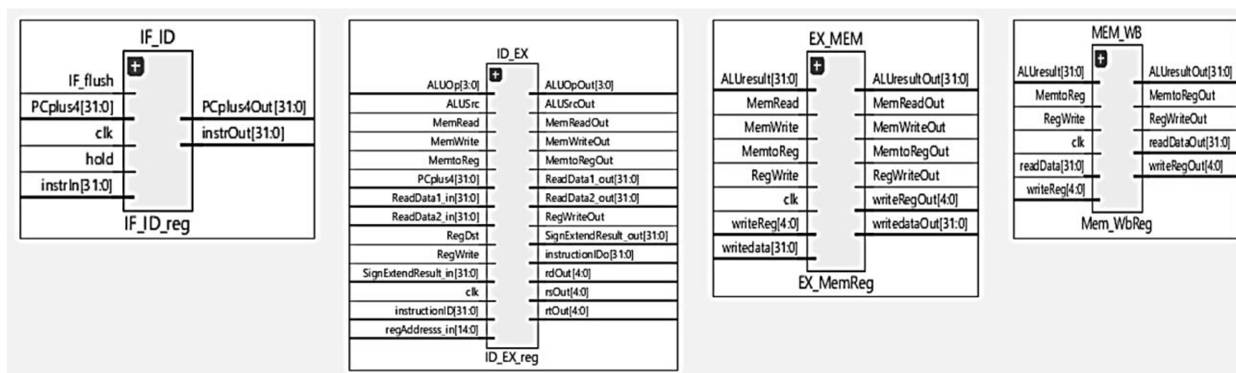

Fig.9 Schematic of Register unit of pipelined datapath

6. Hazard Detection And Stalling

Pipeline hazard are some situations in which the instructions are such that the pipeline cannot happen in the normal way and we need to stop for some clock cycles in order to properly execute the operations. There are mainly 3 different types of hazards:

6.1 Structural Hazard

This type of hazard occurs if the data memory and instruction memory are not separate. Hence in the fourth stage when there will writing to the data memory, some instruction might read from the instruction memory. Since reading and writing from the same memory at the same time is practically not possible in the same control cycle, this gives rise to Structural Hazard.

6.2 Data Hazard

Let us consider an example:

add t0, t1, t2

add t4, t0, t3

Now from the above example we can see that after the execution of first instruction we would get the value of t0 which can be used in the execution of the second instruction. Hence either we need to stall the second instruction till the execution unit of the first or do forwarding in which the as soon as the execution happens the data is sent to the instruction decode stage thereby saving a clock cycle.

6.3 Control Hazard

Consider the following example:

beq t0, t1, L1

Sub t0, t1, t2

L1: add t3, t4, t5

From the above example, we can see that after the first instruction execution, we would get that which instruction we can execute or not. So either we need to stall the next instruction fetch after the execution unit or we use branch predictors. In branch predictors, we have schemes from which we can select if branch is taken or not in the beginning of the instruction fetch and proceed accordingly. If after the execution unit result, the prediction is alright, then we can proceed accordingly. If the prediction is false we need to flush the entire instruction and again proceed with the correct instruction.

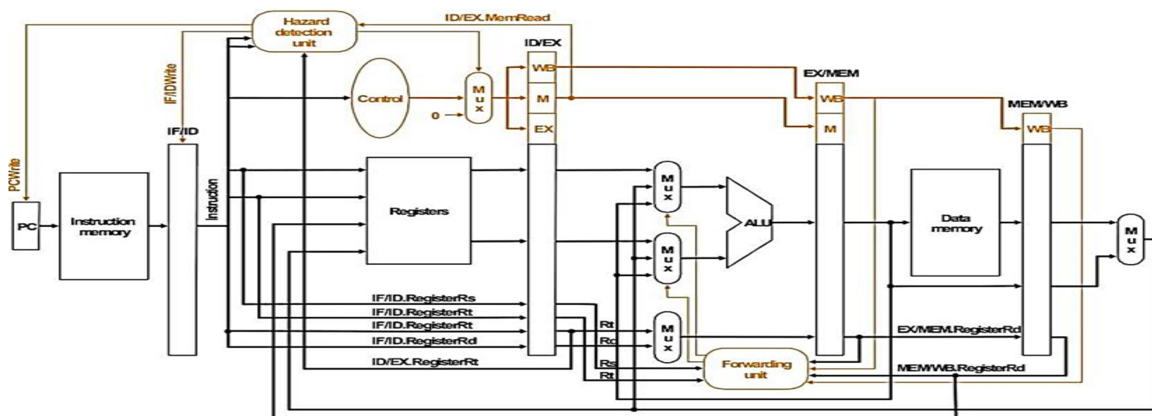

Fig.10 Hazard Detection Unit

In our processor design, hazard detection primarily addresses two categories of hazards: data hazards and control hazards.

Data hazards occur when an instruction attempts to use a value that has not yet been updated, leading to incorrect computations. To detect such conditions, the design checks whether the destination register of the EX/MEM stage matches either of the source registers of the ID/EX stage. This is expressed as the condition:

(ID_EX_Rt == IF_ID_Instr[25:21]) || (ID_EX_Rt == IF_ID_Instr[20:15])

When this condition is met, control signals are generated accordingly. The holdPC signal ensures that the Program Counter (PC) retains its current value, thereby preventing the fetch of the next instruction. Simultaneously, the IF/ID pipeline register holds its previous value, and the control signals are zeroed out to effectively stall subsequent stages.

Control hazards arise in branch instructions, since the outcome of a branch can only be determined in the execution stage. By this time, the following instruction has already begun execution, leading to potential inconsistencies. To handle this, the design either stalls the pipeline until branch resolution or flushes instructions that were incorrectly fetched.

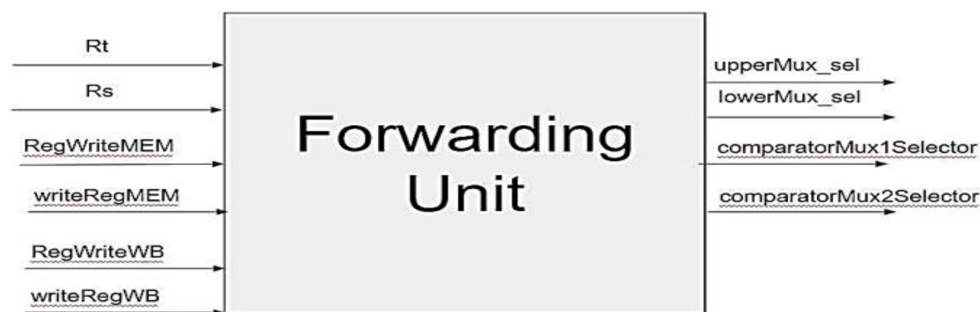

Fig. 11 Forwarding Element

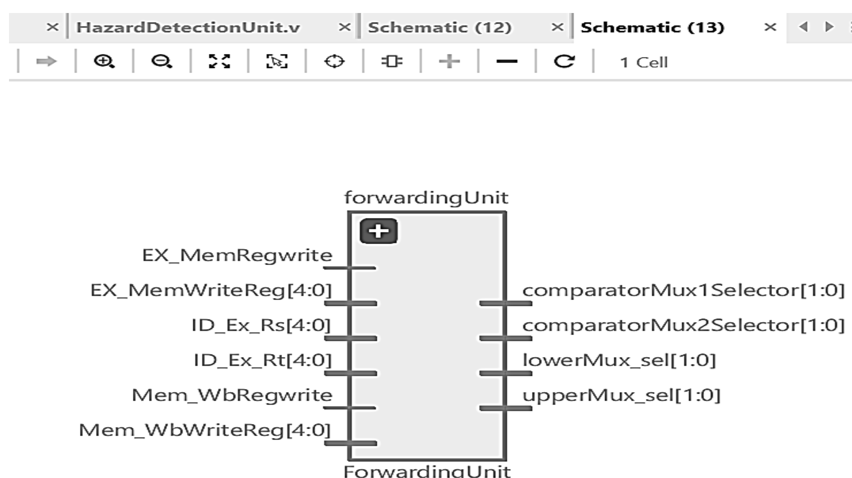

Fig. 12 Schematic of Forwarding Unit

Forwarding is employed to further mitigate data hazards. If the control signals MemRegWrite and MemWriteReg from the EX/MEM stage are active, and the destination register (MemWriteReg) matches the source register (Rs) of the ID/EX stage, a forwarding signal is generated. This enables the ALU output from an earlier stage to be directly forwarded to the dependent instruction, ensuring correctness without waiting for the write-back stage. Similar

checks are performed for the Rt register as well. Additionally, if the register being written to in the final stage coincides with the register currently being accessed, the forwarded value is taken from the last multiplexer output instead of the register file, since the register file still contains an outdated value.

7. Simulation Results

The simulation waveform shown in Fig. 13 validates the functionality of the designed 5-stage pipelined 32-bit RISC-V processor. The clock (clk) and reset (reset) signals operate as expected, ensuring proper synchronization and initialization of the pipeline. The Program Counter (nextPC, readPC, PCPlus4) increments correctly in steps of 4, confirming sequential instruction fetch.

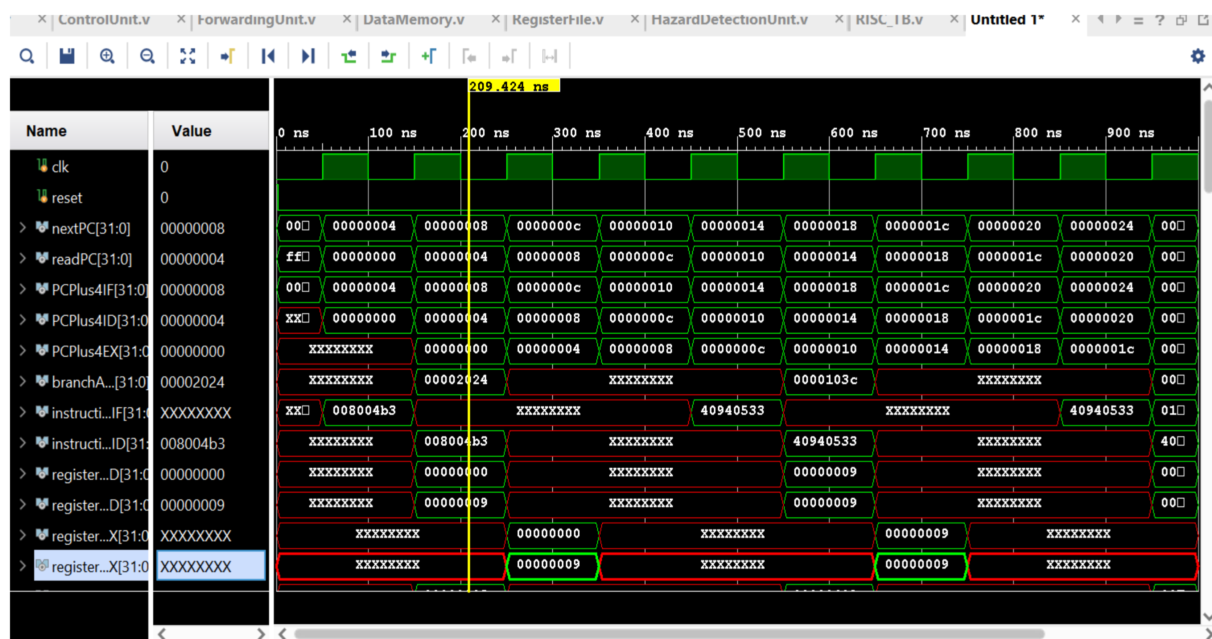

Fig. 13 Simulated timing diagram of top-level module

Overall, the timing diagram confirms that:

- The Program Counter is updated correctly in each cycle.
- Instruction fetch, decode, execute, memory, and write-back stages function as intended.
- Register values are updated with accurate results after execution.
- Hazard detection and forwarding mechanisms successfully prevent data inconsistencies.

Thus, the waveform verifies the functional correctness and pipeline efficiency of the implemented RISC-V processor.

8. Conclusion and Future Scope

In this work, a 5-stage pipelined 32-bit RISC-V processor has been designed and implemented in Verilog on the Xilinx Vivado platform, incorporating a hazard detection unit to efficiently resolve data and control hazards, thereby minimizing pipeline stalls and enhancing execution throughput. The processor supports fundamental instruction types such as R-type, I-type, load/store, and branch instructions while maintaining strict compliance with the RISC-V ISA, ensuring seamless software compatibility. Comprehensive simulation and validation confirmed the functional correctness of the design, along with notable improvements in performance, low area utilization, and reduced power consumption achieved through architectural optimization. While the present implementation establishes a robust and efficient baseline architecture, future

enhancements may focus on the integration of advanced pipeline techniques, such as branch prediction and forwarding mechanisms, the extension of instruction set support to include floating-point and compressed instructions, the incorporation of cache memories for optimized memory hierarchy, and the adoption of low-power design methodologies. Furthermore, transitioning the design from FPGA-based prototyping to ASIC realization offers significant potential for improved scalability, efficiency, and applicability across embedded systems and high-performance computing domains.

References

1. Enfang Cui, Tianzheng Li, and Qian Wei, “RISC-V Instruction Set Architecture Extensions: A Survey,” *IEEE Access*, vol. 11, pp. 24696–24711, 2023.
2. Avinash N. J., Ishani Mishra, Tejas C. Ghorpade, Lokesh M., Nishanth H., and Manoj Kumar S., “Design and Implementation of 32-Bit MIPS RISC Processor with Flexible 5-Stage Pipelining and Dynamic Thermal Control,” in *2023 IEEE International Conference on Intelligent Systems, Smart and Green Technologies (ICISSGT)*, 2023.
3. T. Gokulan, Akshay Muraleedharan, and Kuruvilla Varghese, “Design of a 32-bit, Dual Pipeline Superscalar RISC-V Processor on FPGA,” in *2020 23rd Euromicro Conference on Digital System Design (DSD)*, Kranj, Slovenia, 2020, pp. 340–343.
4. Pankaj Nair V. M. and Lalu V., “Design and Implementation of 5-Stage Pipelined RISC-V Processor on FPGA,” in *2024 28th IEEE International Symposium on VLSI Design and Test (VDAT)*, Bengaluru, India, 2024.
5. Suseela Budi, Pradeep Gupta, Kuruvilla Varghese, and Amrutur Bharadwaj, “A RISC-V ISA Compatible Processor IP for SoC,” in *2018 IEEE International Symposium on Devices, Circuits and Systems (ISDCS)*, Howrah, India, 2018.
6. Y. Shi, Z. Jiang, R. Kumar, J. Chen, and R. Zhang, “Evaluating RISC-V for High-Performance Computing,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 5, pp. 1095–1108, May 2020.
7. K. Delić and V. Milutinović, “A Survey of RISC-V Microprocessor Implementations,” *IEEE Access*, vol. 8, pp. 19105–19134, 2020.
8. Rakesh M.R, “RISC Processor Design in VLSI Technology Using the Pipeline Technique”, International Journal Of Innovative Research In Electrical, Electronics, Instrumentation And Control Engineering Vol. 2, Issue 4, April 2014.
9. Supraj Gaonkar, Anitha M, “Design of 16-bit RISC Processor”, International Journal of Engineering Research & Technology (IJERT), ISSN: 2278-0181, Vol. 2 Issue 7, July – 2013.
10. Galani Tina G, Riya Saini and R.D.Daruwala, “Design and Implementation of 32-bit RISC Processor using Xilinx”, International Journal of Emerging Trends in Electrical and Electronics (IJETEE – ISSN: 2320-9569) Vol. 5, Issue. 1, July-2013.
11. Anil Kumar, worked on “Design and Implementation of a 32 bit VLSI RISC architecture”, Department of Computer Science and Engineering, Indian Institute of Technology, Kanpur, March 1993.
12. Pranjali.S.Kelgaonkar alongside Prof. Shilpa Kodgire, “Design of 32 bit MIPS RISC Processor Based SoC”, International Journal of Latest Trends in Engineering and Technology (IJLTET - ISSN:2278 621X) Vol. 6, Issue 3, January-2016.
13. Wael M ElMedany, Khalid A AlKooheji, “Design and Implementation of a 32bit RISC Processor on Xilinx FPGA”, Department of Communications and Electrical Engineering, Fayoum University, Egypt, Computer Engineering Department, Information Technology College, University Of Bahrain, Bahrain.
14. Mrs. Ussra Fathima, Dr. Baswaraj Gadgay and Mr. Veeresh Pujari, “A 32-Bit RISC Processor for Convolution Application”, Department of VLSI Design and Embedded

Systems, Visvesvaraya Technological University, Kalaburagi, Karnataka, India. IJISET - International Journal of Innovative Science, Engineering & Technology, Vol. 3 Issue 7, July 2016.

15. John L. Hennessy, and David A. Patterson, “Computer Architecture A Quantitative Approach”, 4th Edition.
16. Rainer Ohlendorf, Thomas Wild, Michael Meitinger, Holm Rauchfuss, Andreas Herkersdorf, “Simulated and measured performance evaluation of RISC based SoC platforms in network processing applications”, Journal of Systems Architecture 53 (2007) 703–718.