Pipelined Architecture of RISC-V Processor with Hazard Control

Bijaya Nanda Murmu (B220020)
Pratik Ray (B220039)
Siddharth Sahu (B220054)

under the guidance of
Dr. Pradyut Kumar Biswal

Department of Electronics & Telecommunications
International Institute of Information Technology Bhubaneswar
Bhubaneswar, Odisha, 751003, India
May, 2024

Motivation

The motivation behind RISC-V RV32I, the 32-bit integer base instruction set variant of the RISC-V architecture, lies in addressing the need for a standardized, open-source, and flexible instruction set architecture that caters to diverse computing requirements while promoting innovation and collaboration in the field of computer architecture.

Objective

- **Implement a High-Performance RISC-V Core:** Design and implement a 5-stage pipelined RISC-V RV32I processor core for efficient execution of integer instructions.

- **Achieve Full Hazard Control:** Integrate techniques like forwarding and stalling to ensure smooth instruction flow within the pipeline and avoid incorrect results.

- **Leverage Open-Source Advantages:** Utilize the open-source nature of RISC-V to customize the core for optimal performance within an embedded system application.

- **Demonstrate Scalability:** Highlight the modularity of RISC-V, allowing for future expansion with optional extensions based on evolving project needs.

- **Showcase Industry Relevance:** Emphasize the growing adoption of RISC-V by major companies and research institutions, demonstrating its practicality and potential impact.

Methodology

1) Design and Architecture Exploration:

- Study the RISC-V RV32I ISA specifications and understand the instruction set architecture.
- Research existing 5-stage pipelined processor designs and explore different pipelining techniques.
- `Design the data path and control path for the 5-stage pipeline, considering data hazards and control hazards.

2) Hazard Control Implementation:

- Implement forwarding techniques (data forwarding, control forwarding) to bypass pipeline stages and resolve data hazards.
- Implement stalling mechanisms to prevent incorrect results due to control hazards (branch prediction can be explored as an optimization).

3) Verification and Testing:

- Develop a comprehensive test suite covering various RISC-V instructions and functionalities.
- Utilize simulation tools to test the designed core and verify its functionality under different scenarios.
- Implement unit tests for individual pipeline stages to ensure proper operation.

Instruction Format

RISC-V defines six main instruction formats to balance regularity and simplicity in decoder hardware: R-type, I-type, S-type, B-type, U-type and J-type.

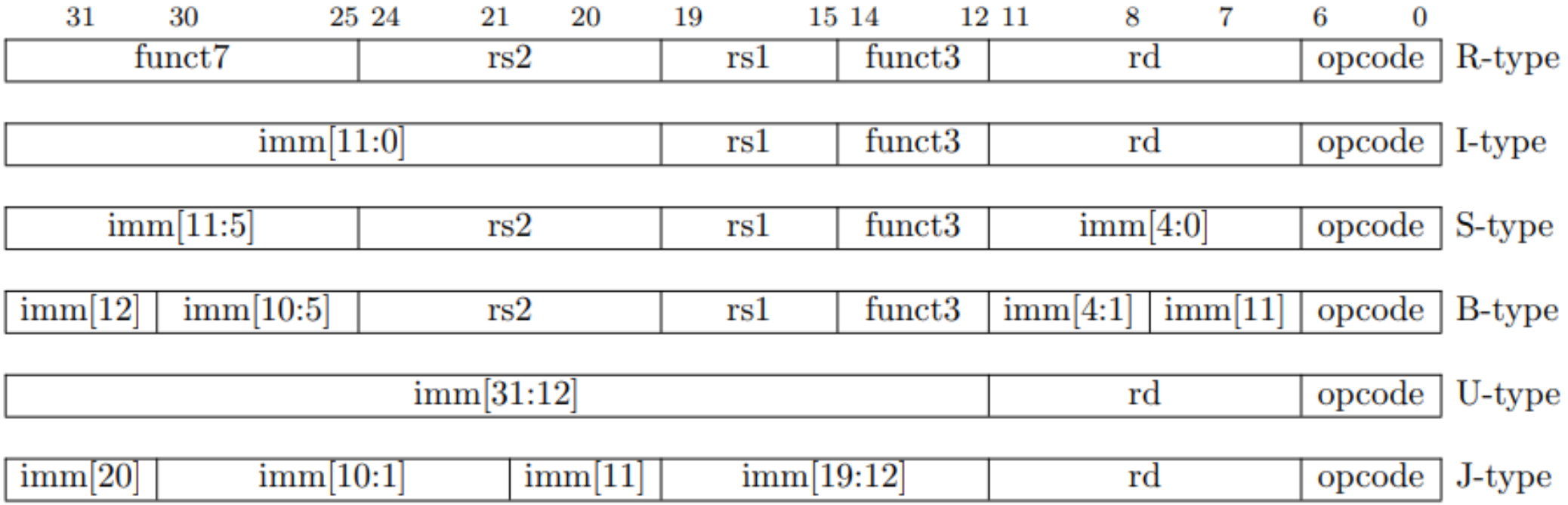


Figure 1: RISC-V base instruction formats.

Flowchart

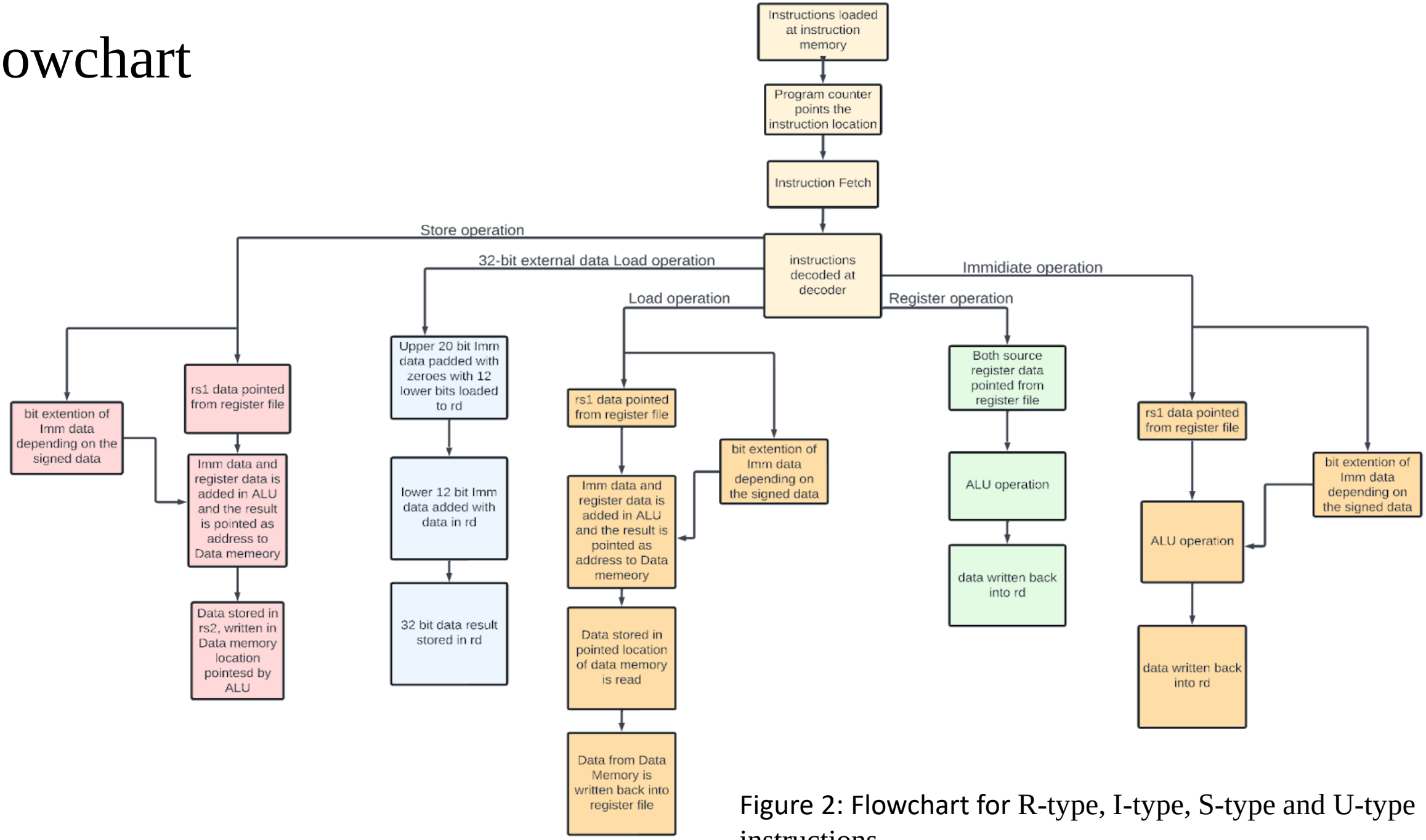


Figure 2: Flowchart for R-type, I-type, S-type and U-type instructions.

Flowchart

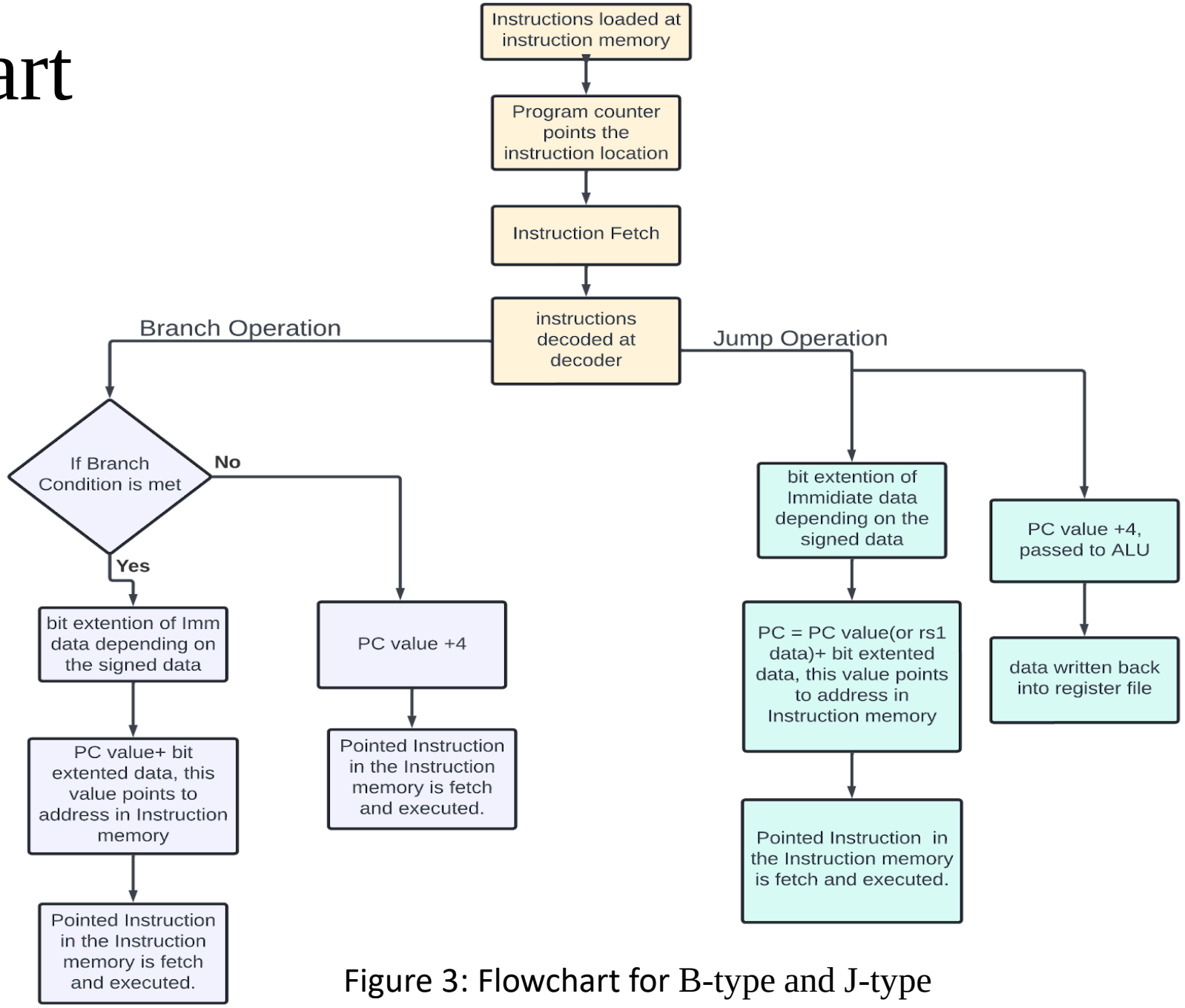


Figure 3: Flowchart for B-type and J-type

Block Diagram

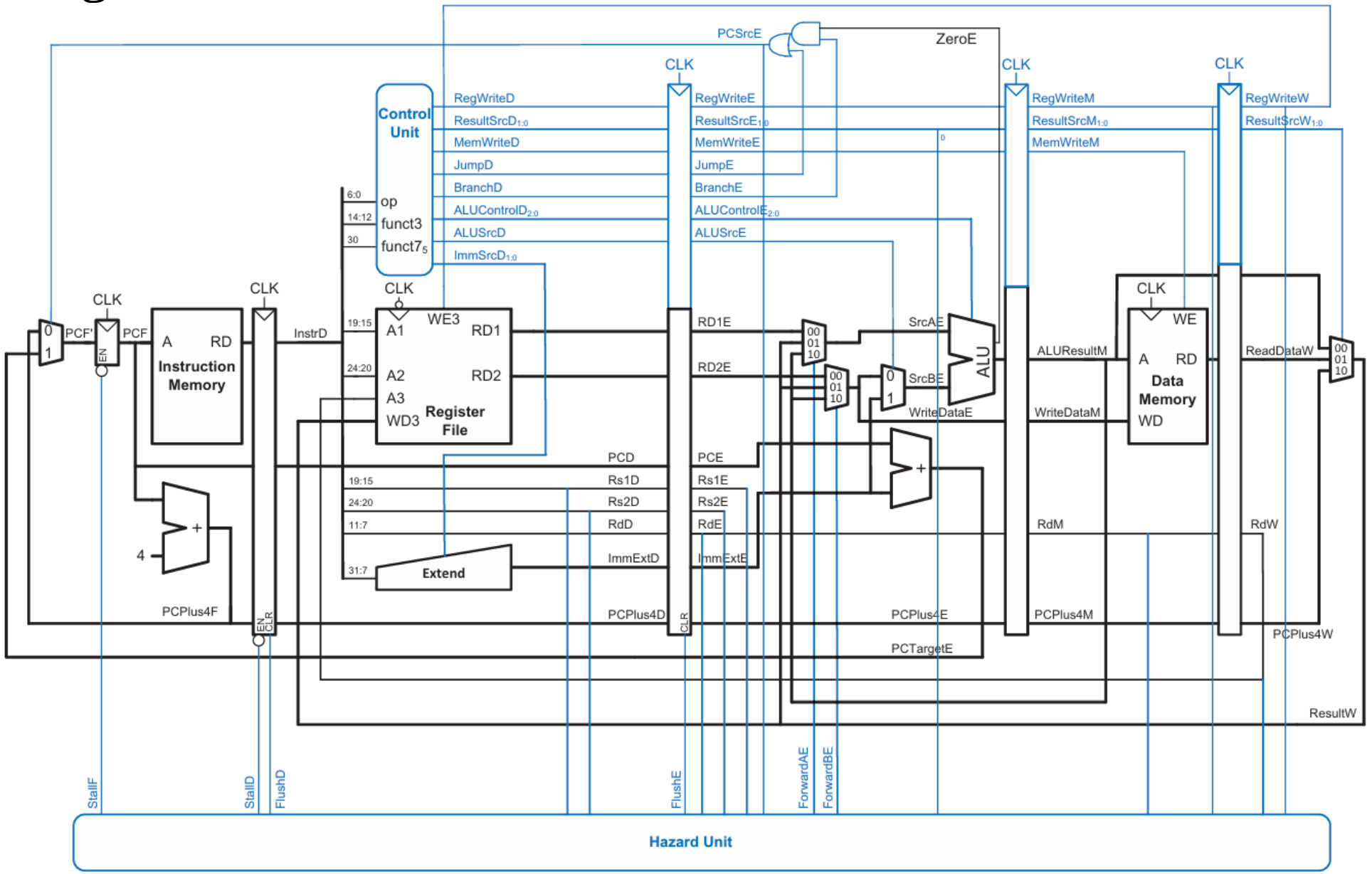


Figure 4: Block diagram for pipelined architecture RV32I with full hazard control [8]

Implementation

1. **Program Counter**
2. **Register File**
3. **Instruction Fetch**
4. **Fetch Decode Pipeline Stage**
5. **Instruction Decoder**
6. **Decode Execute Pipeline**
7. **ALU**
8. **Execute Memory Pipeline Stage**
9. **Data Memory (Memory Access)**
10. **Memory Writeback**
11. **Stall controller**
12. **D-cache & I-cache**

Specifications

No. of registers in Register bank	32
Size of each register	32bits
Register Bank Size	128 Bytes
Instruction memory size	Upto 4 GB
Data memory size	Upto 4 GB
Clock Frequency	50MHz
Bus size (address bus and data bus)	32-bit
Pipeline Stages	5
RISC-V Base Integer set	RV32I

Architecture (Schematic)

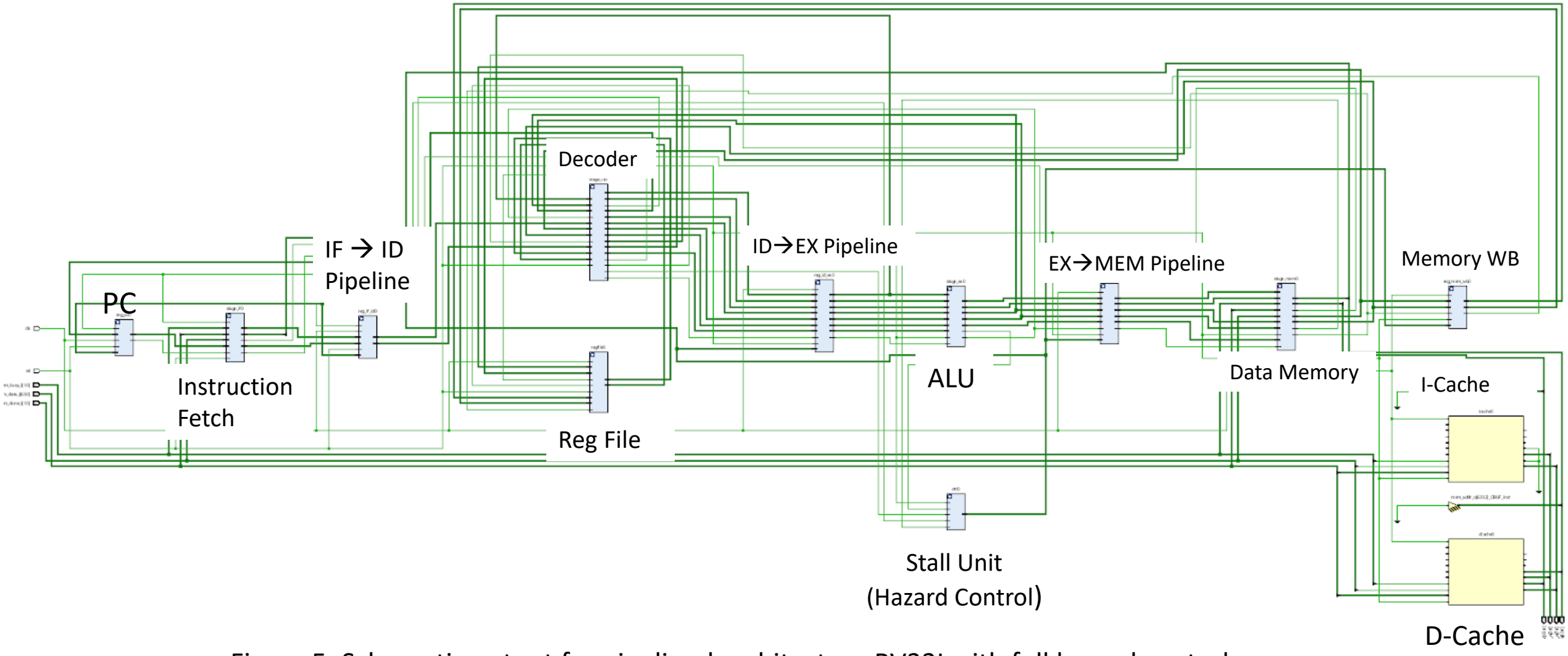


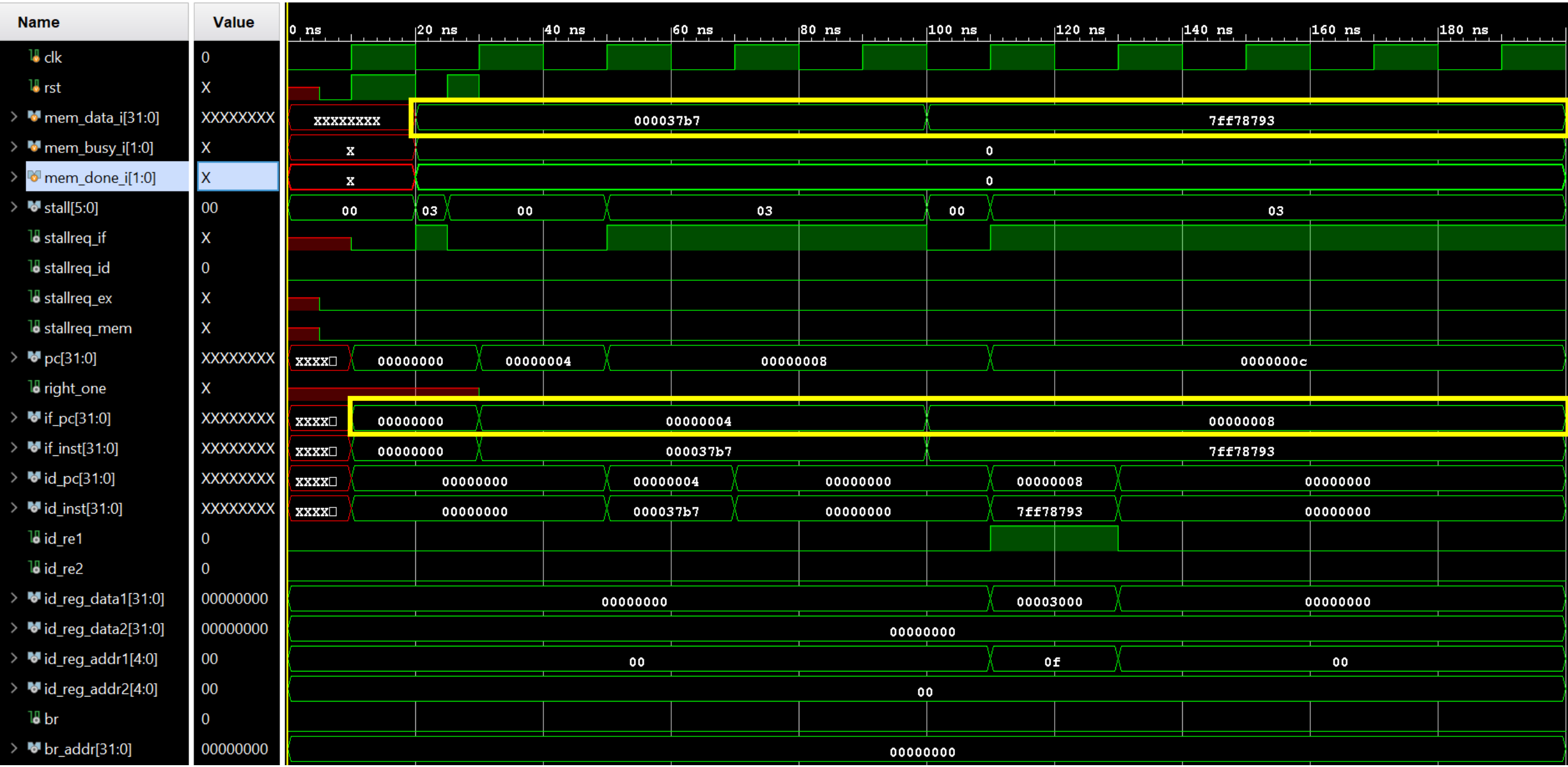
Figure 5: Schematic output for pipelined architecture RV32I with full hazard control

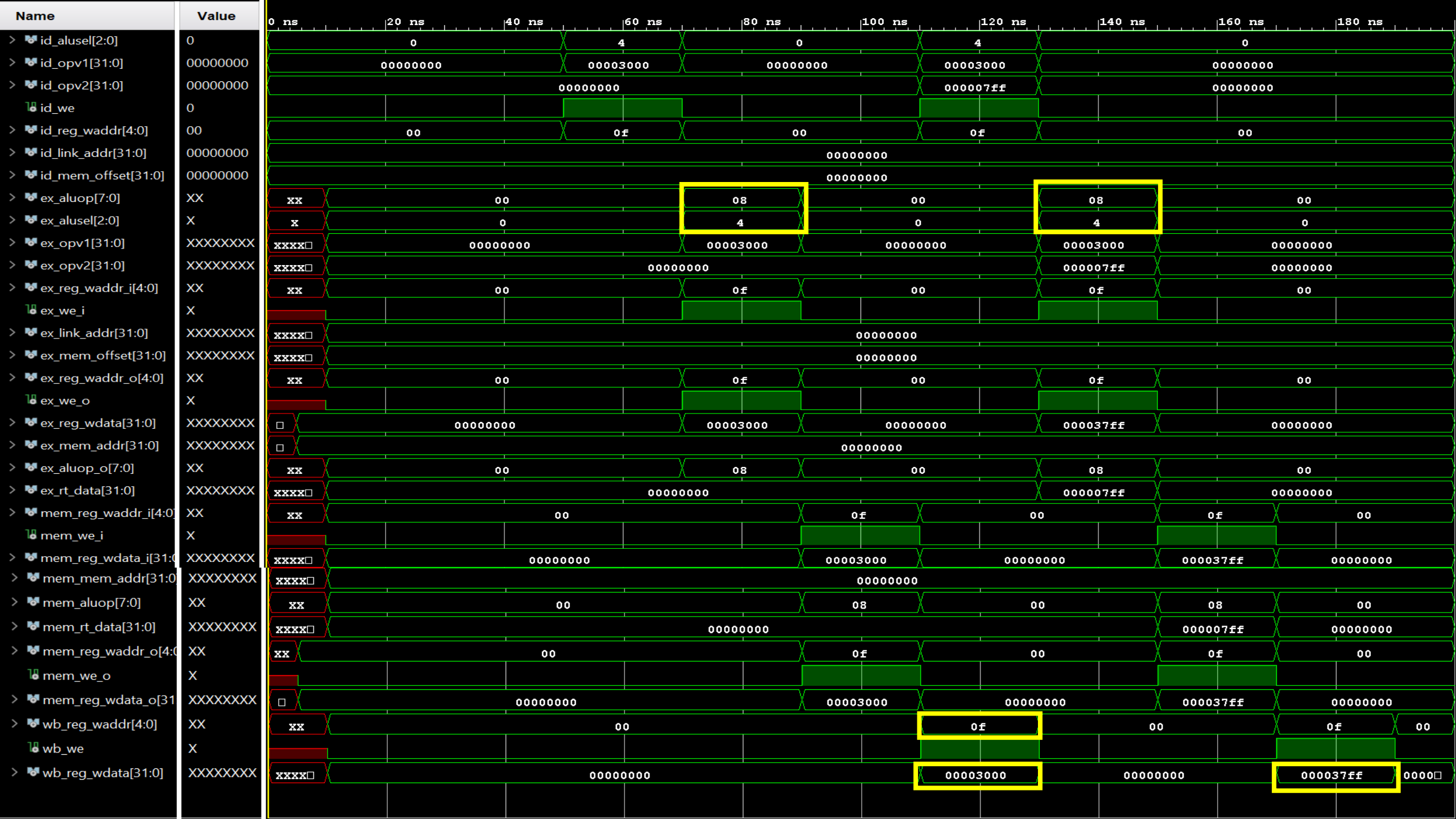
Testbench Simulation

```
//reset
initial begin
#5 rst =1'b0;
#5 rst =1'b1;
#5 rst =1'b1;
#5 rst =1'b0;
#5 rst =1'b1;
#5 rst =1'b0;
end

//clock
initial begin
clk= 1'b0;
forever #10 clk= ~clk;
end
initial begin
/* lui ->       U_imm           rd    opcode
        00000000000000000011 01111 0110111  */
#20 mem_data_i= 32'b00000000000000000011011110110111; mem_busy_i= 2'b00; mem_done_i= 2'b00;
/* addi ->      I_imm       rs     funct3   rd    opcode
             011111111111 01111    000   01111  0010011 */
#80  mem_data_i= 32'b01111111111101111000011110010011;
#20  $display("Register value at address %h after lui operation: %h", wb_reg_waddr, wb_reg_wdata);
#60  $display("Register value at address %h after addi operation: %h", wb_reg_waddr, wb_reg_wdata);
end
initial begin
#200 $finish ;
end
endmodule
```

Simulated Waveform




Name
Value
0 ns
20 ns
40 ns
60 ns
80 ns
100 ns
120 ns
140 ns
160 ns
180 ns
id_alusel[2:0]
id_opv1[31:0]
id_opv2[31:0]
id_we
id_reg_waddr[4:0]
id_link_addr[31:0]
id_mem_offset[31:0]
ex_aluop[7:0]
ex_alusel[2:0]
ex_opv1[31:0]
ex_opv2[31:0]
ex_reg_waddr_i[4:0]
ex_we_i
ex_link_addr[31:0]
ex_mem_offset[31:0]
ex_reg_waddr_o[4:0]
ex_we_o
ex_reg_wdata[31:0]
ex_mem_addr[31:0]
ex_aluop_o[7:0]
ex_rt_data[31:0]
mem_reg_waddr_i[4:0]
mem_we_i
mem_reg_wdata_i[31:0]
mem_mem_addr[31:0]
mem_aluop[7:0]
mem_rt_data[31:0]
mem_reg_waddr_o[4:0]
mem_we_o
mem_reg_wdata_o[31:0]
wb_reg_waddr[4:0]
wb_we
wb_reg_wdata[31:0]

Result

Lui (110-130ns)	Addi(170-190ns)
20bit data after padding = 0003000	Lower 12 bit = 7FF Addition Result = 00003000 + 000007FF= 000037FF
Address(5-bit) = 0 F (R15)	Address(5-bit) = 0 F (R15)

```
Register value at address 0f after lui operation: 00003000
Register value at address 0f after addi operation: 000037ff
```

Figure 6: Register State after lui and addi operation

Validation:

	lui operation		addi operation	
	Expected result	Actual Result	Expected Result	Actual Result
Destination Register address	0 1111		0 1111	
Data stored in destination address	0000 0000 0000 0000 0011 0000 0000 0000		0000 0000 0000 0000 0011 0111 1111 1111	

Conclusion:

- Our Xilinx Verilog simulation and testbench have definitively demonstrated the successful loading of 32-bit immediate data into a designated register using lui and addi instructions. This achievement conclusively validates the functionality of these core RISC-V instructions within the processor.
- Furthermore, the meticulously designed testbench extends beyond this singular test. It comprehensively verifies the processor's ability to handle the extensive instruction set provided by the RISC-V ISA. This comprehensive testing approach definitively confirms the processor's capacity to execute a broad spectrum of RISC-V programs.

References

[1] Kovačević N., Mišeljić, Đ., & Stojković A. (2022). “RISC-V vector processor for acceleration of machine learning algorithms”. In 2022 30th Telecommunications Forum (TELFOR) (pp. 1-6).

[2] An, Hyogeun, et al. "Single Cycle 32-bit RISC-V ISA Implementation and Verification."

[3] TOZLU, Y. S., & YILMAZ, Y. (2021). “DESIGN AND IMPLEMENTATION OF A 32-BIT RISC-V CORE”.

[4] Raveendran, A., Patil, V. B., Selvakumar, D., & Desalphine, V. (2016, January). “A RISC-V instruction set processor-micro-architecture design and analysis.” In 2016 International Conference on VLSI Systems, Architectures, Technology and Applications (VLSI-SATA) (pp. 1-7). IEEE.

[5] Katke, Sharda P., and G. P. Jain. "Design and implementation of 5 stages pipelined architecture in 32 bit risc processor." International Journal of Emerging Technology and Advanced Engineering 2.4 (2012): 340-346.

[6] Dörflinger, Alexander, et al. "A comparative survey of open-source application-class RISC-V processor implementations." Proceedings of the 18th ACM international conference on computing frontiers. 2021.

[7] Samanth, Rashmi, Ashwini Amin, and Subramanya G. Nayak. "Design and Implementation of 32-bit Functional Unit for RISC architecture applications." 2020 5th International Conference on Devices, Circuits and Systems (ICDCS). IEEE, 2020.

[8]. D. Harris and S. L. Harris, "Digital Design and Computer Architecture, RISC-V Edition," Elsevier, 2021, ch. 7, p. 453.