Design of 16-bit RISC Processor

Liyanage D.L.S.B. : EG/2019/3655
Sadeepa P.M.A.S. : EG/2019/3726

***Abstract*— This project focuses on creating a 16-bit RISC processor using Verilog, a hardware description language. The processor follows a straightforward design with separate memory for instructions and data. It supports 13 instructions with three addressing modes and includes 16 general-purpose registers, each storing 16-bit data.**
Key components like a 16-bit ALU capable of performing arithmetic and logical operations and a flag register for result status are integrated. The project employs modular design principles to ensure efficient and error-free implementation.
The resulting processor offers versatility for various applications, making it an essential exploration of digital system design and computer architecture
Keywords—RISC, 16-bit CPU, Verilog

I. INTRODUCTION

This Due to the increasing demand for faster and more efficient processors, two prominent approaches were developed in computer architecture: Complex Instruction Set Computers (CISC) and Reduced Instruction Set Computers (RISC).

CISC processors were originally favored because of their comprehensive sets of instructions. However, CISC architectures struggled to deliver the performance expected as computers grew in complexity. There was a significant issue with the interaction between the processor and memory, which often slowed the execution.

To address this, RISC architectures were developed. Their instruction sets were simplified, focusing on essential operations like loading and storing. As a result of this simplification, RISC instructions were able to be executed rapidly, typically within a single clock cycle. Moreover, the introduction of pipelining, a technique that enables multiple instructions to be processed simultaneously in stages, accelerated performance even further.

The term "Reduced" in RISC does not mean fewer instructions; rather, it means that RISC instructions accomplish their task in a single memory cycle, unlike more complex instructions that require multiple memory cycles in CISC CPUs.

A majority of today's microprocessors are based on the RISC or the CISC architecture. It has been demonstrated through research that RISC architecture significantly improves computer speed, particularly for frequently used operations. Important characteristics of RISC-based systems include:

1. **Pre-fetching**: This method optimizes the flow of instructions by retrieving the upcoming instruction(s) into a queue before the current one is finished.
2. **Pipelining**: This technique increases processor efficiency by allowing instructions to be issued before the current one is completed.
3. **Superscalar operation**: Superscalar processors can carry out several instructions at once, speeding up computation.

II. ARCHITECTURE

The project's primary objective is to design a 16-bit RISC processor following the Harvard architecture while minimizing functional units. The processor's architecture, illustrated in Figure 1, features a 16-bit ALU capable of executing 11 arithmetic and logical operations, a 16-bit program counter, a 32-bit instruction register, sixteen 16-bit general-purpose registers (R0 through R32), and a flag register indicating zero`.

The processor operates in four states: idle, fetch, decode, and execute. A dedicated control unit manages signal interactions to perform expected functions in each state. The program counter, a 16-bit register, indicates the memory address for instruction fetching. After executing the current instruction, the program counter is incremented by one, unless a 'JUMP' instruction is encountered, in which case it can be incremented or decremented by a specified offset.

Instructions are stored in the instruction memory, and the program counter output serves as input to the instruction memory. The instruction corresponding to the program counter's address is fetched, stored, and decoded in the instruction register. During decoding, the destination register, source register, memory location address, or immediate value is assigned based on the opcode. The opcode guides the ALU in performing the requested operation and generates necessary control signals.

Additionally, the processor offers 32, 16-bit general-purpose registers named R0 through R32, featuring two read ports and one write port. When the 'RegWrite' signal is enabled, data is written into the register indicated by the write address. Otherwise, two registers indicated by the read addresses are read. The ALU performs 11 arithmetic and logical operations, with operands sourced from registers, and the operation results are written back into the specified destination register through a multiplexer. Together, the 16-bit program counter and instruction memory constitute the fetch unit.

Decode Unit
The primary role of the instruction decode unit is to take the 32-bit instruction obtained from the preceding fetch unit and utilize it to access the register file, retrieving the necessary data values from registers. This decode unit is a collaborative effort involving the instruction register, control unit, and the registers themselves.

Execution Unit

The execution unit within the processor is composed of the arithmetic logic unit (ALU), responsible for executing the operation as specified by the opcode in the instruction. This unit performs the actual arithmetic and logical computations mandated by the instruction.

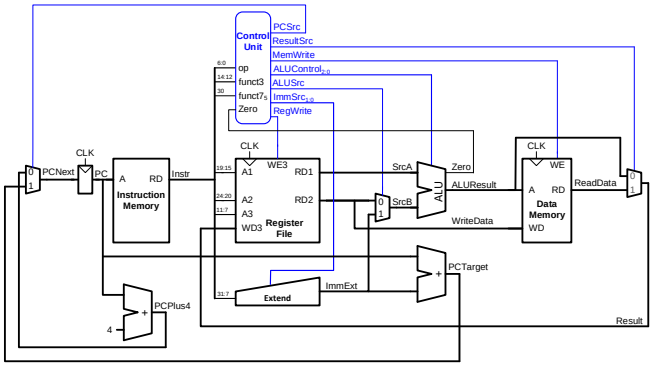

Figure 1: Architecture

III. INSTRUCTION SET

The instruction word has a width of 32 bits, and there are seven different types of instructions. Each type of instruction has a unique structure, which is illustrated in Table 1

Table 1 : Instruction set

31:25	24:20	19:15	14:12	11:7	6:0	
funct7	rs2	rs1	funct3	rd	op	R-Type
$imm_{11:0}$		rs1	funct3	rd	op	I-Type
$imm_{11:5}$	rs2	rs1	funct3	$imm_{4:0}$	op	S-Type
$imm_{12,10:5}$	rs2	rs1	funct3	$imm_{4:1,11}$	op	B-Type
$imm_{20,10:1,11,19:12}$				rd	op	J-Type
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits	

Based on the instruction set outlined above, it is possible to perform 13 arithmetic and logical operations, as indicated in the table presented in Figure 3. Each arithmetic operation is associated with a distinct structure, and for proper execution, the 32-bit word must adhere to that specific structure.

Table 2

Description	Opcode	Funct3	Funct7
Add	0110011	000	0000000
Sub	0110011	000	0100000
And	0110011	111	0000000
Or	0110011	110	0000000
Set less than	0110011	010	0000000
Add immediate	0010011	000	-
And immediate	0010011	111	-
Or immediate	0010011	110	-
Set less than immediate	0010011	010	-
Branch if =	1100011	000	-
Load word	0000011	010	-
Store word	0100011	010	-
Jump and link	1101111	-	-

The instructions can be classified into the following five functional categories: data transfer operations, arithmetic operations, logical operations, branching operations and control operations.

IV. METHODOLOGY

1. Processor Architecture Design:

Design a 16-bit RISC processor with a Harvard architecture, featuring separate data and instruction memory, an ALU, program counter, instruction register, general-purpose registers, and a flag register, and create a block diagram to visually illustrate the processor's architecture.

2. Instruction Set Definition:

Here RISK-V architecture is used. According to that architecture, instruction set is defined for opcode, source and destination registers, immediate values, and encoding schemes for different instruction types, and document opcode assignments, operand formats, and detailed execution behaviors for each instruction to serve as a reference during

3. Verilog Coding:

Write Verilog code for individual processor components like ALU, program counter, registers, and control unit as separate modules to ensure modularity and facilitate debugging, while focusing on proper functionality and adherence to the defined architecture and instruction set.

4. Simulation and Testing:

Utilize Verilog simulation tools such as Vivado Xilinx to simulate the processor design, creating testbench code to verify module functionality and overall processor behavior, and conduct comprehensive testing to validate correct instruction execution, data handling, and flag management, addressing and resolving any issues that surface during testing.

5. Integration and Verification:

Combine all processor modules to create a complete processor design and verify their interoperability, and test the integrated processor with various test cases and programs to confirm it adheres to the specified architecture and instruction set, ensuring it performs as expected.

V. SIMULATION RESULTS

lw

As lw is an I-type instruction rs1 (source register) contains the base address. The lw instruction also requires an offset. The offset is stored in the immediate field of the instruction. It is a signed value, so it must be sign-extended to 16 bits by using the Extend unit. The processor adds the base address to the offset to find the address to read from memory. This addition is performed by ALU. Then Result of the ALU is sent to the data memory as the address to read. The data is read from the data memory and then written back to the destination register (rd) at the end of the clock cycle.

Instruction:111111111100_00100_010_01011_0000011
Immediate : 111111111100
rs1 : 00100
rd : 01011
opcode : 0000011
function3 : 010

As the instruction is lw opcode and function3 should be 0000011 and 010 respectively. In the example instruction source register is 00100 and it is the 4th index of the register file. Here index 4 contains 0x0007. As immediate is 111111111100 it is equivalent to -4. Therefore the ALU operation gives us 3. So the processor reads index 3 from the RAM and writes back it into the destination register of the register file. In this example RAM[3] initializes with the value 9. So 9 will write back to the destination register 01011 of the register file. Therefore index 11 of the register file stores value 9 after executing this instruction. This operation can be seen in the following waveforms.

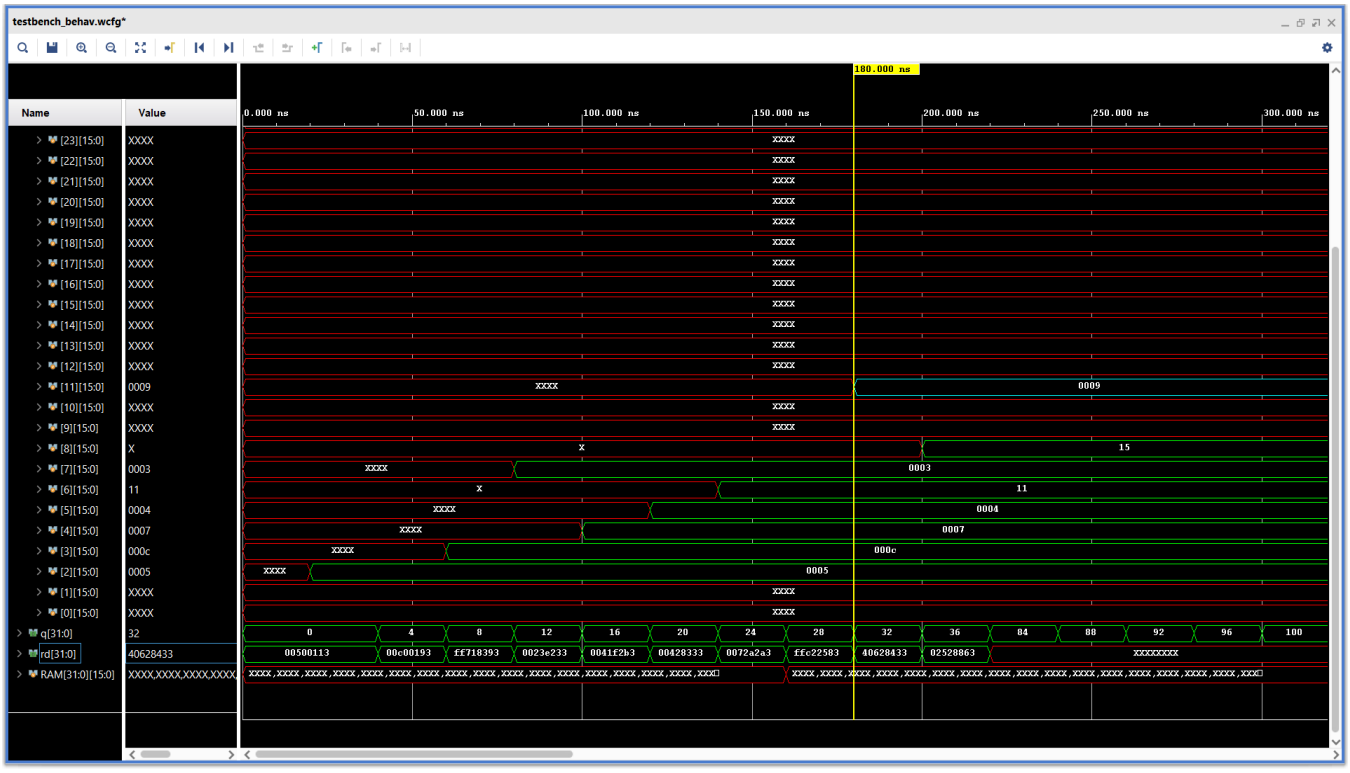

Figure 2 : Load word Instruction

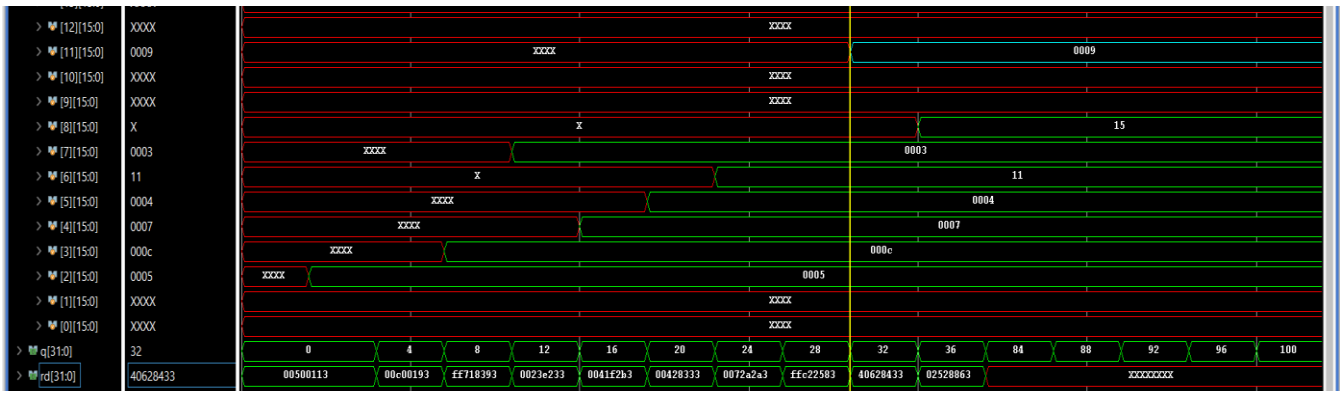

Figure 3 : Load word Instruction

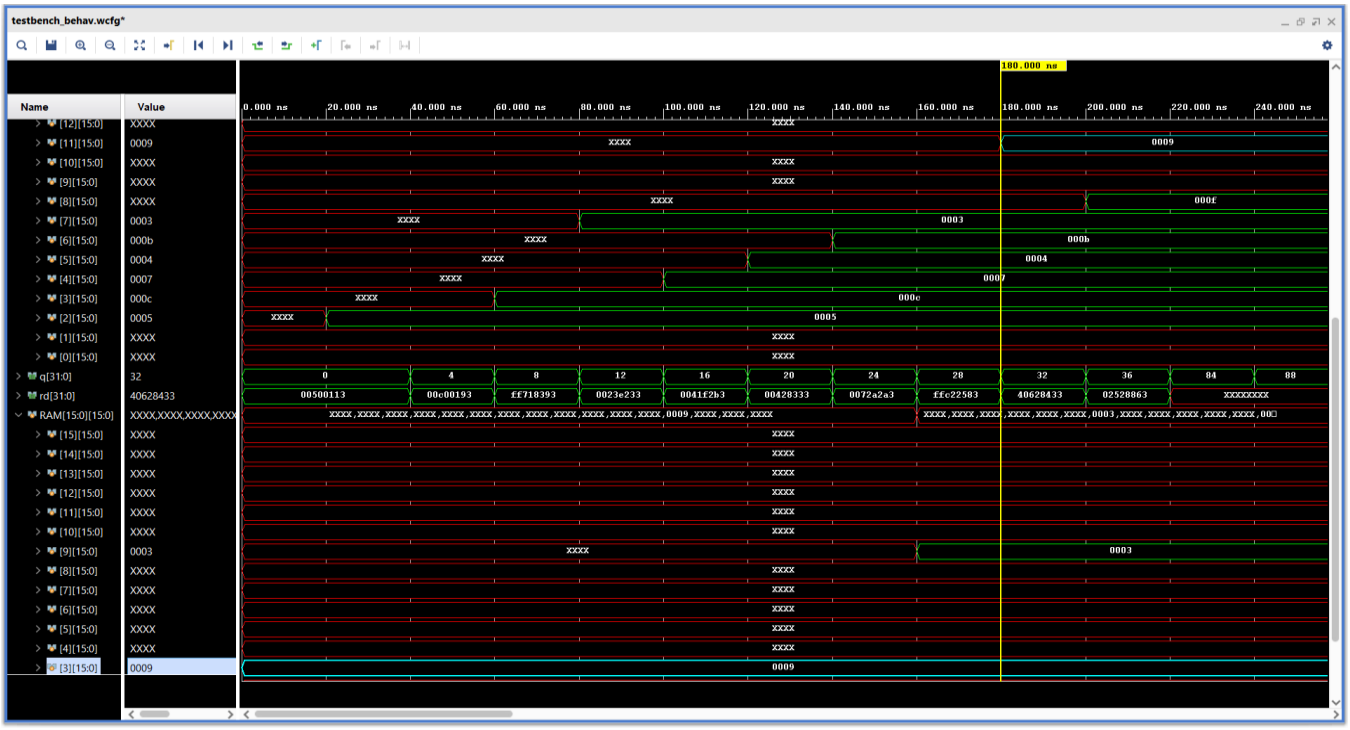

Figure 4 : Load word instruction

Sw

This is an S-type instruction. sw reads a base address from the register file and sign-extends the immediate. The ALU adds the base address to the immediate to find the memory address. ALU adds the base address to the immediate to find the memory address. The sw instruction also reads a second register from the register file and writes its contents to the data memory.

Instruction: 0000000_00111_00101_010_00101_0100011
Immediate: 0000000_00101
rs2 : 00111
rs1 : 00101
opcode : 0100011
function3 : 010

When this instruction executes, the register file reads base address 0x0004 from index 5. Immediate is equal to 5. The ALU computes 0x0004 + 5 = 0x0009. As the rs2 is 00111 register file reads the value 3 from that index and sends it to data memory. Therefore data memory writes 3 to RAM[9]. This instruction can be seen in following waveforms.

Figure 5 : Store word instruction

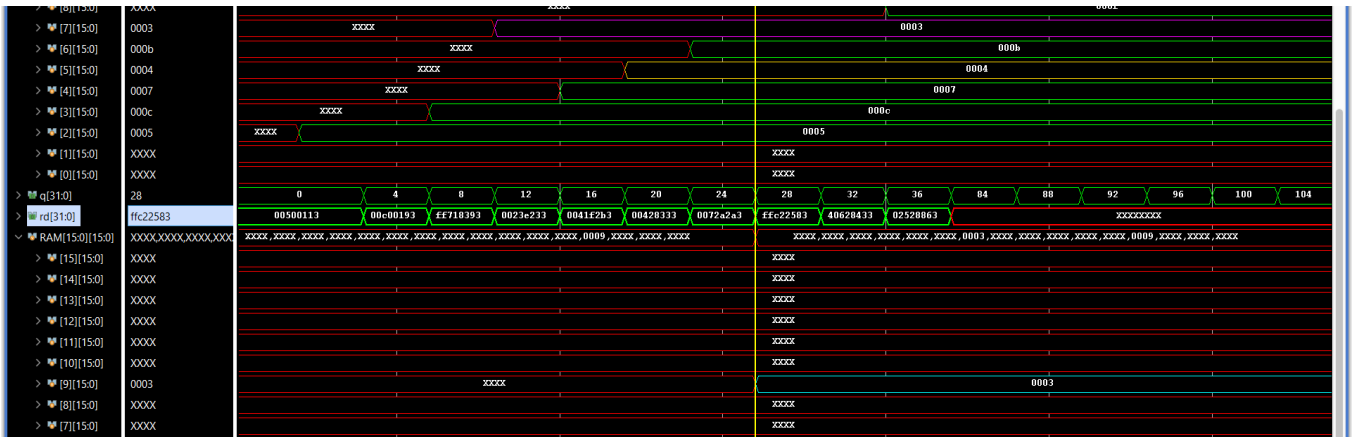

Figure 6 : Store word instruction

add

add is R-Type instruction. This instruction reads two source registers from the register file and performs some ALU operation on them, and writes the result back to the destination register.

Instruction: 0000000_00100_00101_000_00110_0110011
function 7 : 0000000
rs2 : 00100
rs1 : 00101
rd : 00110
funcion 3 : 000
opcode : 0110011

In this instruction source register 2 is the 4th index of the register file. It contains 7. Similarly source register 1 is the 5th index of the register file and it contains value 4. According to the opcode, function 3 and function 7 this is addition operation. After addition result value (7+4=11) will be stored in the destination register (6th index of register file). In the waveform it can be seen that after executing 6th instruction (addition instruction) 11 is stored in the register file at 6th index.

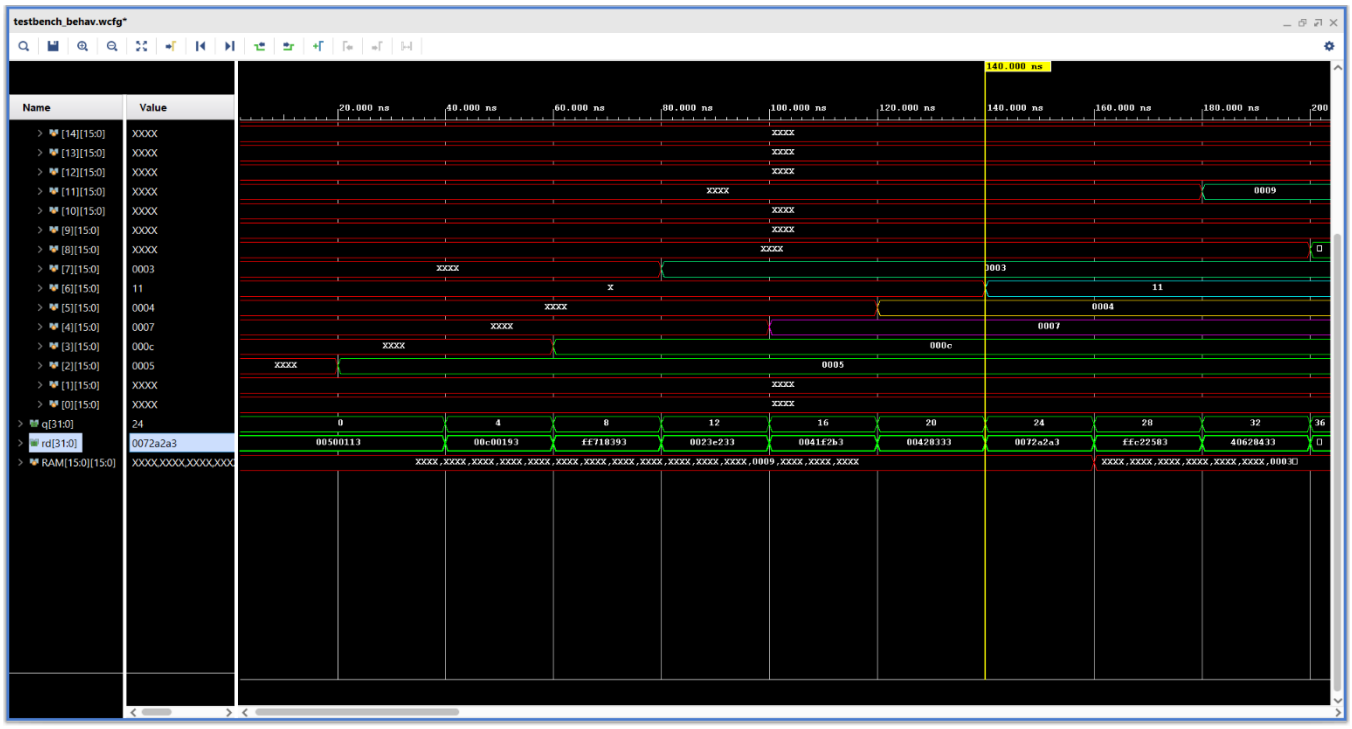

Figure 7 : Add instruction

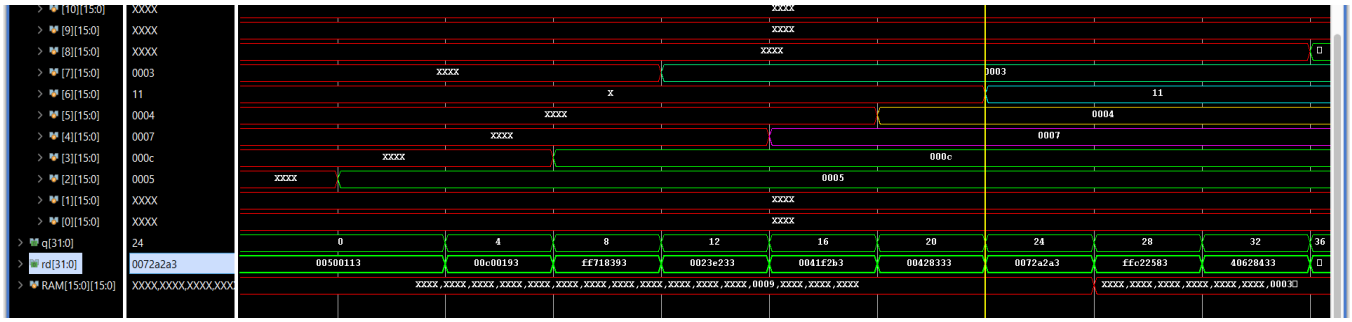

Figure 8 : Add Instruction

VI. CONCLUSION

In conclusion, this project has successfully designed and implemented a 16-bit RISC processor using Verilog, following the Harvard architecture. The processor features separate memory for instructions and data, a 16-bit ALU capable of performing 11 arithmetic and logical operations, a flag register for result status, and 16 general-purpose registers, each storing 16-bit data. The project adheres to modular design principles, ensuring efficiency and reliability in implementation.

The project's significance lies in its contribution to digital system design and computer architecture exploration. It exemplifies the advantages of RISC architecture, such as efficient instruction execution, pipelining, and superscalar operation, which are crucial for enhancing computer speed, especially for frequently used operations.

Through the defined instruction set and methodology, the project covers various aspects, from architecture design to Verilog coding, simulation, and testing. This comprehensive approach ensures that the resulting processor performs as expected, validating its adherence to the specified architecture and instruction set.

VII. REFERENCESS

[1] M. Bin, I. Reaz, M. Islam, and M. Sulaiman, “A Single Clock Cycle MIPS RISC Processor Design using VHDL,” 2002. Accessed: Sep. 12, 2023. [Online]. Available: https://17760100877370134570.googlegroups.com/attach/d2794bbb33f7feec/a1.pdf?part=0.1&vt=ANaJVrFxM6aBHUmu3Cov41dbUPZxGCOMWoxJMLPYUVBQ1ctTTwoHOqL5EHGqLrSQun6aXFlgxWg-O5lar7GIY-lA1XxfqBYta0p8zAsdiW_vcNMKtJpE2v0

[2] S. Gaonkar and A. M, “Design of 16-bit RISC Processor,” International Journal of Engineering Research & Technology, vol. 2, no. 7, Jul. 2013, doi: https://doi.org/10.17577/IJERTV2IS70828