Project Report: 16-bit RISC Processor in Verilog HDL

1. Introduction

This project involves the design and implementation of a 16-bit RISC processor using Verilog HDL.
The processor follows a reduced instruction set architecture and supports arithmetic, logical, memory, and control operations.

The main goal of this project was to understand CPU architecture, datapath design, and control signal generation through RTL-level design and simulation.
The design was verified using simulation tools such as Icarus Verilog and GTKWave.

2. Objective

To design a 16-bit RISC processor at the RTL level using Verilog HDL.
To develop separate modules for ALU, Control Unit, Register File, and Memories.
To integrate these modules into a complete CPU.
To verify the design functionality through simulation and waveform analysis.

3. System Overview

The processor architecture follows a modular design consisting of:

Arithmetic Logic Unit (ALU):
Performs arithmetic and logic operations.

Alu control:
Generates the operation codes for Alu operations depending on opcode and ALU_op which is generated by the control unit.

Control Unit:

Decodes instructions and generates control signals.

Register File:
Holds general-purpose registers.

Instruction Memory:
For this processor iStores program instructions.

Data Memory:
Used for memory read and write operations.

CPU Top Module:
Connects all submodules and controls execution flow.

The design supports a basic instruction set with operations like:
ADD, SUB, AND, OR, NOT, COMPARE, LEFT SHIFT AND RIGHT SHIFT (Arithmetic & Logic)
LOAD, STORE (Memory)
JMP (Control)
BEQ(Control)
BNQ(Control)

4. Design Methodology

The project followed the standard RTL design flow:

1. Design Entry:
Each module was written in Verilog HDL, following a hierarchical approach.
Separate files were created for each component (e.g., `alu.v`, `control_unit.v`, etc.).

2. Functional Simulation:
Testbenches were written to verify module behavior.
Simulations were executed using Icarus Verilog, and waveform outputs were viewed using GTKWave

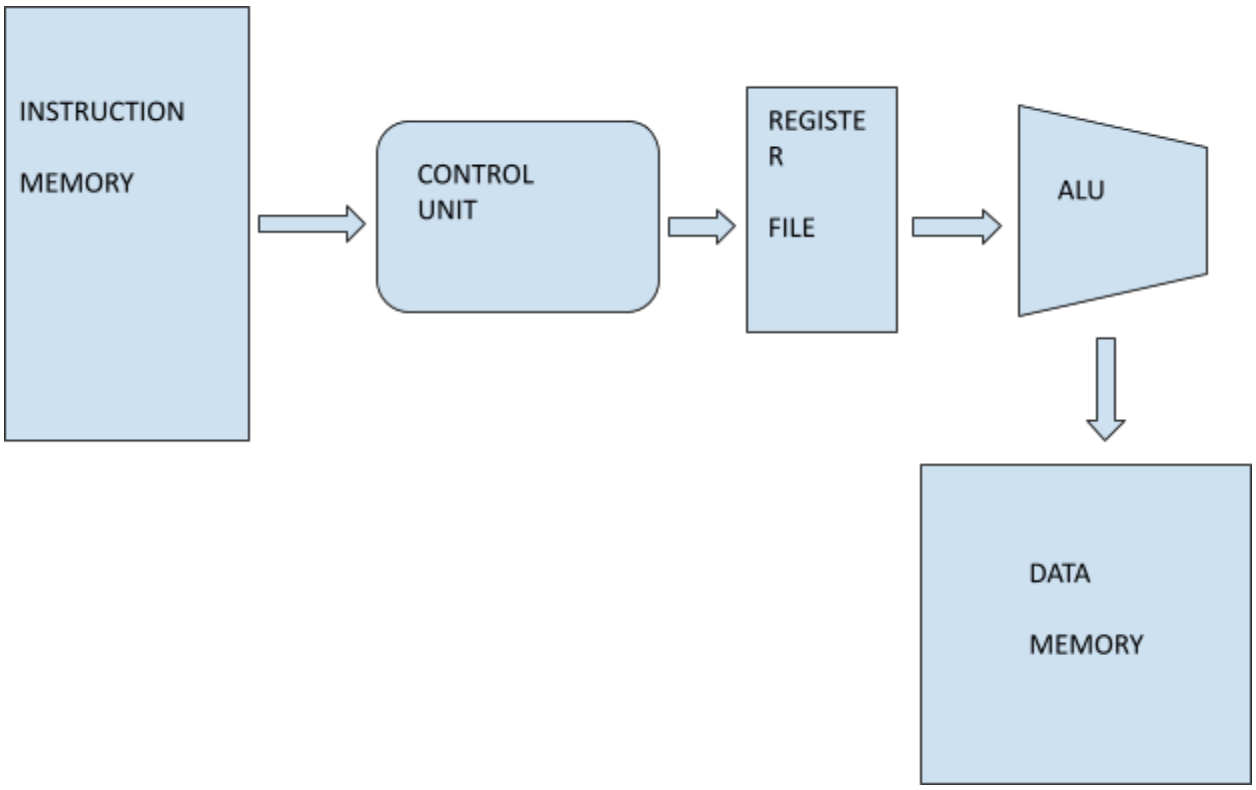
3. Synthesis:
The complete design was synthesized using Yosys to confirm hardware compatibility and generate netlists.

4. Verification:

Waveforms were analyzed to ensure correct data flow and control signal generation for all supported instructions.

5. Architecture Diagram

Below is a simplified representation of the system:



6. Module Descriptions

6.1 ALU

The ALU performs arithmetic and logic operations based on the control signals.
Supported operations: addition, subtraction, AND, OR, XOR.

6.2 Control Unit

The Control Unit decodes the instruction opcode and generates the required control signals for ALU operations, register selection, and memory access.

6.3 Register File
Contains general-purpose registers used for intermediate storage during instruction execution.

6.4 Instruction Memory
Stores the machine instructions. It is read-only during program execution.

6.5 Data Memory
Handles read/write operations for data load and store instructions.

6.6 CPU Top Module
Integrates all submodules and connects the datapath with the control path.

7. Simulation and Results

7.1 Simulation Setup

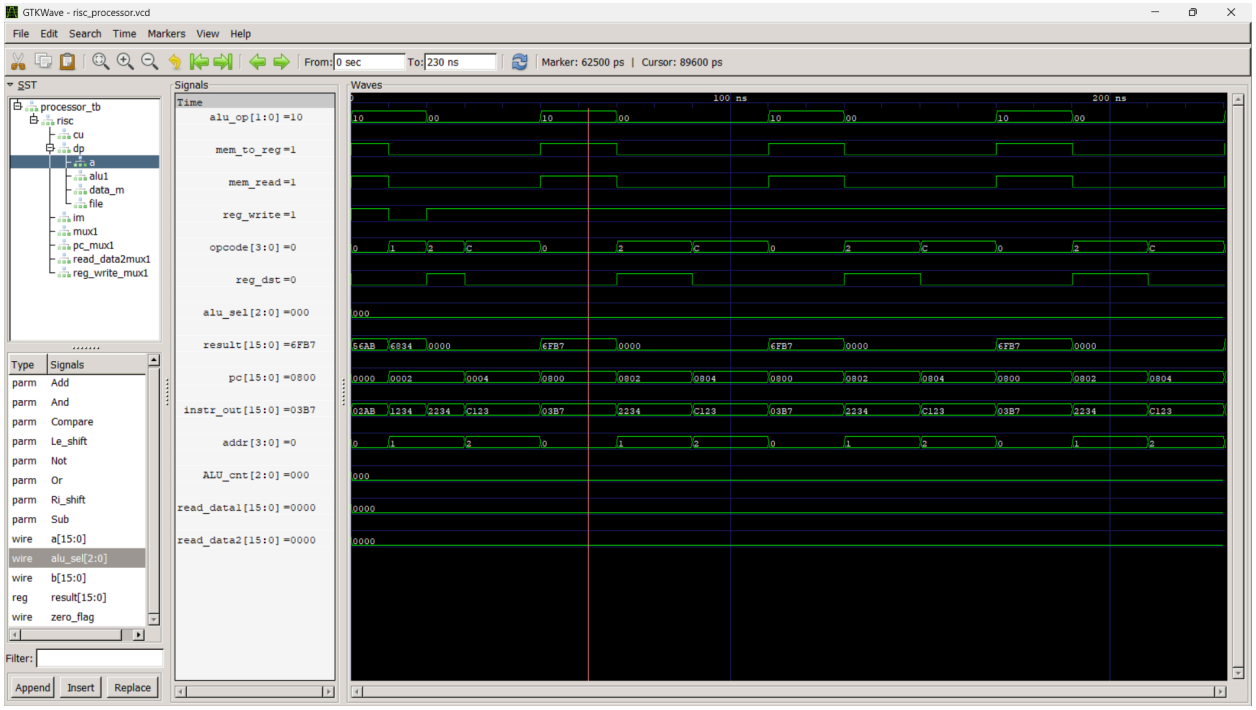
Simulator : Icarus Verilog

Waveform Viewer: GTKWave

Testbench : different testbenches are written to test individual modules
for example "control_unit_testbench.v"

7.2 Simulation Output
Simulation confirmed correct execution of all supported instructions.
Example results:
- `ADD R1, R2, R3` → Result stored in `R1` = R2 + R3
- `LOAD R4, [MEM]` → Data loaded from memory into R4

Example waveform:



8. Synthesis

The design was synthesized using Yosys to verify logic synthesis correctness.
The tool generated a netlist and a gate-level schematic confirming that all RTL elements are synthesizable.

Example command sequence:

```
yosys
# Read all module files
read_verilog data_path_modules.v
read_verilog data_path.v
read_verilog control_unit.v
read_verilog risc_processor.v

# Set top-level module
hierarchy -top risc_processor

# Flatten everything into top-level
flatten risc_processor
```

```
# Generate Graphviz .dot file
show -format dot -prefix risc_processor_flat risc_processor
```

The Yosys-generated netlist for the processor is included in this repository. You can open and explore it to see the full module hierarchy and interconnections.

9. Results and Observations

The CPU successfully executed all tested instructions.
All module interconnections were verified.
Timing and logical correctness confirmed through simulation.
The design is synthesizable for FPGA implementation.

10. Future Enhancements

Implement pipelining for improved performance.
Integrate UART module for I/O communication.
Extend instruction memory for larger programs

11. Tools used and their purpose

Icarus Verilog	:	Compilation & Simulation of written verilog code.
GTKWave	:	Waveform Analysis of different signals in design.
Yosys	:	RTL Synthesis (generates netlist for written synthesizable verilog code)
VS Code	:	HDL Editing
Windows/Linux	:	Execution Environment

12. Conclusion

The 16-bit RISC processor was successfully designed, simulated, and verified using Verilog HDL.
The modular architecture allowed for easy testing and future scalability.
This project deepened understanding of digital design concepts, datapath control, and RTL simulation techniques.

13. References

Samir Palnitkar, Verilog HDL: A Guide to Digital Design and Synthesis
David Money Harris & Sarah Harris, Digital Design and Computer Architecture
Yosys HQ Documentation
Icarus Verilog Wiki