# Design and Simulation of a 32-bit RISC-V Single Cycle Processor

Project Report

**Nithish Reddy**
Department of Electronics and Communication Engineering
College of Engineering, Guindy
Anna University, Chennai, India
nithishreddy.k.v.s@gmail.com

July 13, 2025

**Abstract**

This project involves the design, coding, and simulation of a 32-bit RISC-V single-cycle processor using Verilog HDL. A single-cycle processor is a simplified architecture where all instructions are executed in a single clock cycle. The processor integrates fundamental blocks such as Program Counter, Instruction Memory, Register File, ALU, Immediate Generator, Control Units, and Data Memory. This document provides an in-depth description of each component, their interconnection, simulation insights, and future plans for pipelined and GDSII-based implementations.

# Contents

# 1 Introduction

A single-cycle processor completes every instruction in a single clock cycle, meaning that fetch, decode, execute, memory access, and write-back all occur within one cycle. Although not as performance-efficient as pipelined architectures, single-cycle designs are simpler and ideal for foundational understanding of processor behavior.

The RISC-V instruction set architecture (ISA) is modular, open-source, and widely adopted in academia and industry. This project implements a subset of RISC-V instructions (e.g., `ADDI`, `BEQ`, `LW`, `SW`) using Verilog.

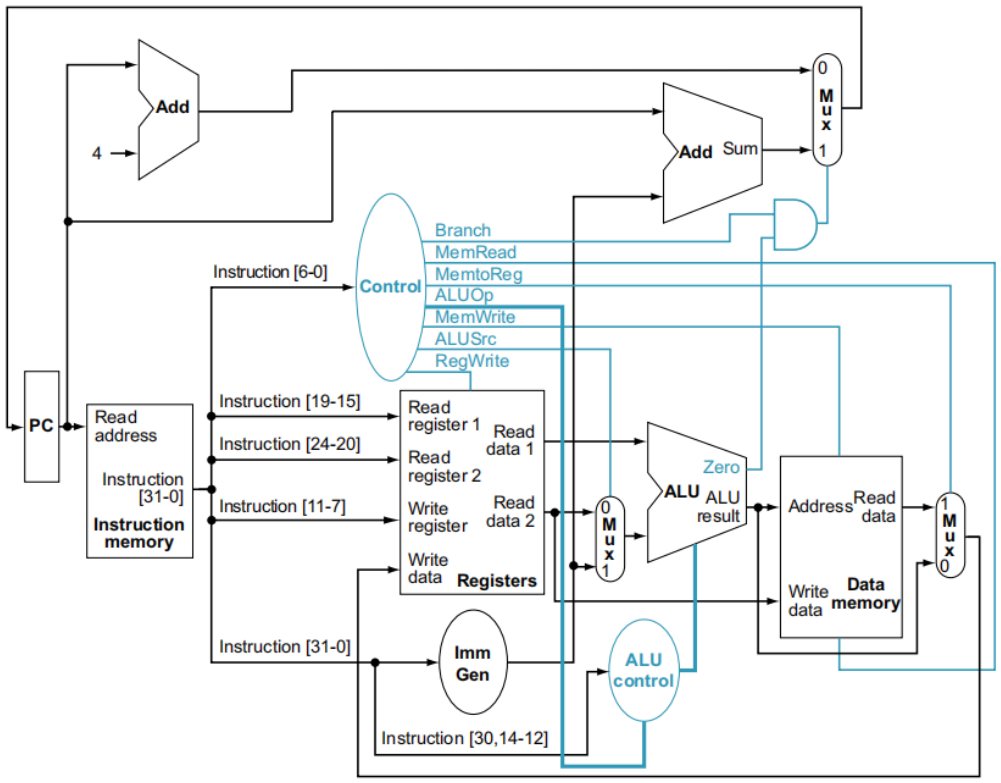


Figure 1: 32-bit Single-Cycle RISC-V Processor Architecture

# 2 Processor Architecture Overview

The major functional blocks of the processor include:

- Program Counter (PC)
- Instruction Memory
- Control Unit (Main Decoder + ALU Decoder)
- Register File
- Immediate Generator (Extend Unit)
- ALU

- Data Memory
- PC Update Logic and Multiplexers

Each of these modules is explained below.

# 3 Module Descriptions

## 3.1 Program Counter

**Inputs:** `clk, reset, pc_in`
**Output:** `pc_out`
The PC holds the address of the current instruction. It is updated every clock cycle unless reset is high, in which case it resets to zero.

## 3.2 Instruction Memory

**Input:** `pc`
**Output:** `instruction`
Stores 32-bit machine instructions. Accessed using `pc[31:2]` to ensure word alignment.

## 3.3 Main Decoder

Decodes the instruction opcode and generates control signals like `RegWrite, MemWrite, Branch, ALUSrc, ALUop, ImmSrc`.

## 3.4 ALU Decoder

Based on `funct3` and `funct7` fields, along with ALUop, this block generates a 3-bit `ALUControl` signal to determine the ALU operation.

## 3.5 Register File

**Inputs:** `clk, WE3, A1, A2, A3, WD3`
**Outputs:** `RD1, RD2`
A 32x32 register file that supports two simultaneous reads and one write.

## 3.6 Immediate Generator

Extracts and sign-extends immediate fields from the instruction based on the instruction type (I, S, B, J, U).

## 3.7 ALU

**Inputs:** `SrcA, SrcB, ALUControl`
**Outputs:** `ALUResult, zero`
Performs arithmetic/logic operations. Sets the `zero` flag when `SrcA == SrcB`.

### 3.8 Data Memory

**Inputs:** `clk, address, write data, write enable`
**Output:** `read data`

Simulates main memory for load and store operations.

### 3.9 Multiplexers

- **MUX for PCSrc:** Chooses between `PC+4` and branch target.
- **MUX for ALUSrc:** Selects between register value and immediate.
- **MUX for ResultSrc:** Selects ALU output, memory read data, or PC+4 as the write-back result.

## 4 Instruction Example

The processor was tested with:

```
mem[0] = 32'h00500093; // addi x1, x0, 5
```

This instruction adds immediate 5 to x0 and stores result in x1.

**Observed Behavior:**

- Instruction was fetched from memory.
- ALU performed addition of 0 + 5.
- Register x1 updated with value 5.

## 5 Simulation and Results

Simulation was conducted using a testbench that toggles clock and reset signals. The waveforms observed include:

- PC incrementing correctly
- Instruction fetched accurately
- ALUResult reflecting correct computation
- Write-back to register file confirmed

Due to initial bugs, signals like `ALUResult` were seen as 'x'. After correcting register initialization and write-enable logic, valid values were observed.

## 6 Tools Used

- Vivado / ModelSim for simulation
- Verilog HDL for RTL
- GTKWave for waveform analysis
- (Future) OpenLane and Cadence for physical design

## 7 Future Work

- Fix edge case bugs and enhance instruction support
- Add jump instructions and exception handling
- Design a 5-stage pipelined version of this processor
- Perform full RTL-to-GDSII flow using open-source tools like OpenLane and Cadence Innovus
- Analyze timing and area metrics on synthesized design

## 8 Conclusion

This project successfully demonstrates the working of a 32-bit RISC-V single-cycle processor. The simulation confirms that the design performs instruction fetch, decode, execution, memory access, and write-back in one cycle. Although basic, this processor lays the foundation for advanced microarchitecture exploration.

## Appendix: Sample Testbench

```
module RISC_V_tb;
    reg clk, reset;
    RISC_V_Datapath uut(clk, reset);

    initial begin
        clk = 0;
        forever #5 clk = ~clk;
    end

    initial begin
        reset = 1;
        #10 reset = 0;
        #100 $finish;
    end
endmodule
```