Single-Cycle RISC-V Processor Design

Nile University

ITCS School

Fall'25

—

CSC311 – Intro. To Computer Architecture

Project Overview

In this project, you will design, implement, and simulate a complete **Single-Cycle RISC-V Processor** using Verilog.

This assignment helps you understand how a CPU works internally - from fetching an instruction to executing it - by building every part of the processor yourself.

Your design will follow the single cycle datapath introduced in Lecture 8. To simplify the implementation, you will merge the ALU Control Unit with the Main Control Unit into one module.

Learning Objectives

By completing this project, you will:

- Understand the internal structure of a modern processor.
- Design each building block of a RISC-V CPU using Verilog.
- Integrate all modules into a complete single-cycle datapath.
- Simulate and debug your hardware using Vivado's built-in simulator**.**
- Load and run machine-code instructions in your processor.
- Analyze waveforms to verify correct execution.
- Produce a professional, IEEE-style hardware design report.
- Present your design clearly and confidently.

Required Tools

You will use:

- Vivado Design Suite (simulation only — you will NOT run on FPGA hardware)
- Verilog HDL
- Any RISC-V assembler or online tool to generate machine code

---

Instruction Set Requirements

Your single-cycle RISC-V processor must be able to correctly execute all required instruction categories, including:

I. R-Type Instructions (integer arithmetic & logic)

Your ALU must support all standard R-type operations, including but not limited to:

- add, sub
- and, or, xor
- sll (logical left shift), srl (logical right shift)

II. I-Type Arithmetic

- addi, andi, ori, xori

III. Shift Instructions

- Immediate shifts: slli, srli
- Standard R-type shifts (as listed above)

IV. Memory Access Instructions

Use 64-bit (doubleword) memory operations:

- Load: ld
- Store: sd

V. Branching

- beq

Your CPU must be capable of fetching, decoding, executing, and verifying the correct behavior of these instructions using your single-cycle datapath.

Components You Must Implement

***You will merge the ALU Control Unit with the Main Control Unit.**

***For both Instruction Memory and Data Memory, allocate enough space to comfortably support your chosen design, inputs and system requirements.**

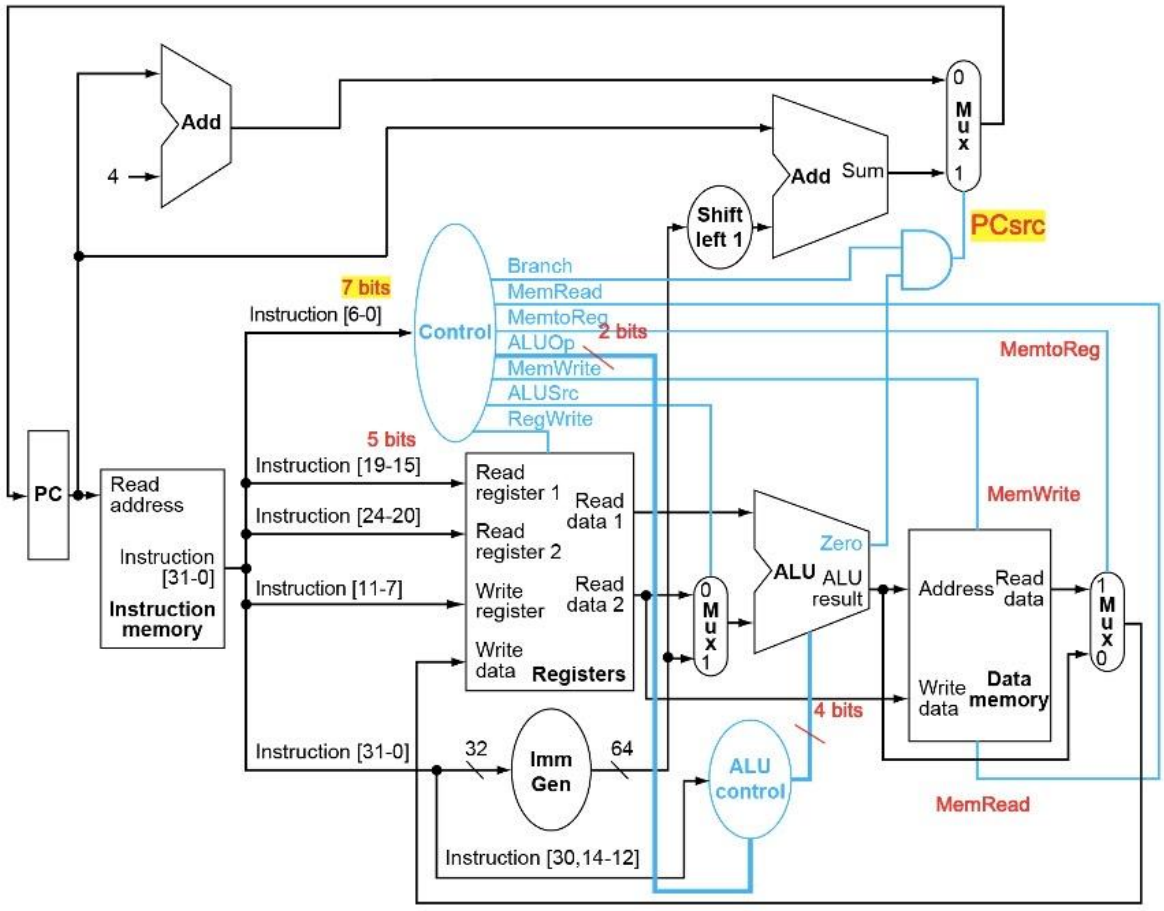

Figure 1 Single-Cycle RISC-V Processor

Project Deliverables

i. Part 1 — Individual Work

Each student must submit:

- One Verilog module they implemented
- A clear explanation of the module's purpose
- A simple testbench showing correct functionality

ii. Part 2 — Group Work

The team should be a **group of 5**. As a group, you will submit:

1. The complete CPU with all modules integrated
2. The instruction memory file containing your test machine code
3. Simulation results showing:
    - PC updates
    - Instruction execution
    - Register writes
    - ALU outputs
    - Memory reads/writes
    - Branching
4. A professional IEEE-format report (PDF) containing:
    - Introduction
    - Datapath diagram
    - Design methodology
    - Control-unit truth table
    - Explanation of modules
    - Full simulation waveforms
    - Conclusion
    - **Download the template of the IEEE paper**: Conference-template-letter.doc
5. A 10–15 minute presentation demonstrating your design and results

Datapath Reference

You must include a version of the **single-cycle RISC-V datapath** in your report. This diagram must guide your implementation and appear in the documentation.

Example Instructions to Test Your CPU

You will be provided later with a small **test program** from our side to verify your system. The results of running this program should be clearly documented both in your report and during the discussion/presentation. The program will include instructions from the instruction set you have already implemented. You will need to convert these instructions into **machine code** and load them into your instruction memory.

In the meantime (until the official test program is provided), you may test your implementation using RISC-V instruction such as: *"This is only an example"*

- addi x1, x0, 5
- beq x13, x14, Label
- sd x3, 0(x0)
- slli x4, x9, 3

To do so you will need to convert these instructions into **machine code** and load them into your instruction memory.

Grading Rubric

A. Individual Component

- Correct Verilog implementation
- Testbench completeness
- Ability to explain the design

B. Group Integration

- CPU executes instructions correctly
- Proper datapath connections
- Control signals are correct

C. Report

- IEEE format
- Correct diagrams
- Clear explanations
- Strong simulation results

D. Presentation

- Clarity
- Organization
- Technical accuracy
- Team participation

Five Individual Student Tasks

Each student in the group must be assigned **one task only**, and they will be evaluated based on the quality, correctness, clarity, and simulation of *their* part.

Task 1 — ALU & Arithmetic Logic Design

You will design and implement the **Arithmetic Logic Unit (ALU)**, which must support:

- add, sub,
- and, or, xor,
- sll, srl ….etc

Your responsibilities:

- Derive ALU operation codes
- Implement ALU behavior in Verilog
- Ensure correct outputs for all R-type and shift operations
- Provide an individual testbench that proves correct ALU operation

---

Task 2 — Control Unit (Main Control + ALU Control Merged)

You will design and implement the **Control Unit**, responsible for generating all internal control signals, including:

- RegWrite
- MemWrite
- MemRead
- ALUSrc
- MemToReg
- Branch
- ALUControl (merged inside)

Your responsibilities:

- Build the control table for all required instructions
- Implement instruction decoding logic
- Generate ALUControl directly from opcode + funct3/funct7
- Provide a testbench that applies opcodes and checks correct control signals

---

Task 3 — Register File & Immediate Generator

You will implement:

A. Register File

- 32 registers
- x0 always 0
- two read ports, one write port
- synchronous write, asynchronous read

B. Immediate Generator

- Correct extraction and sign-extension of immediates for addi, andi, xori, ori, ld, sd, beq, shift-immediates

Your responsibilities:

- Implement both modules in Verilog
- Create a testbench to verify register reads/writes and immediate formats

---

Task 4 — Program Counter, Branch Logic & PC Update

You will design all components related to instruction sequencing:

- PC register (updates every cycle)
- PC + 4 adder
- Branch target calculation
- Branch decision logic
- MUX for selecting the next PC
- Handling branching correctly

Your responsibilities:

- Implement PC logic in Verilog
- Verify correct branching behavior in a testbench

Task 5 — Instruction Memory, Data Memory & Top-Level Integration

You will implement:

A. Instruction Memory

- Preloaded
- Word-aligned addressing

B. Data Memory

- Support for ld and sd
- Word-addressed RAM

C. CPU Top-Level Module

You will integrate:

- PC
- Instruction memory
- Control unit
- Register file
- ALU
- Immediate generator
- Data memory
- Branch logic
- All multiplexers and internal wires

Your responsibilities:

- Make sure all modules connect correctly
- Run the full-core simulation
- Run RISC-V program to test the processor

Submission & Deadline

- Only **one member** submits the project.
- In this single submission, each team member must attach their task using the following form: NAME_ID_TASK_NO.v
- **Complete project + report + presentation** due: The Discussion week before the finals.

Academic Integrity

All submitted work must be your own.

Copying Verilog code or design files from another group or external sources is **strictly prohibited** and will result in disciplinary action.