RISC-V Single Cycle CPU

Ouail Slama

Yahia Kilany

Department of computer science and engineering

CSCE 3301: Computer Architecture

Dr. Shrief Salama

Nov 10, 2025

Introduction

The RISC-V single-cycle CPU is a fundamental implementation of the RISC-V instruction set architecture (ISA), designed to execute one instruction per clock cycle. emphasizing simplicity, modularity, and efficiency. This project aims to design and implement a single-cycle processor capable of executing a subset of RISC-V instructions, including arithmetic, logical, load/store, and branch operations.

In this design, every stage of instruction execution, fetch, decode, execute, memory access, and write-back, is completed within a single clock cycle. Although this architecture is not optimized for speed or power efficiency compared to pipelined processors, it provides a clear understanding of the internal datapath and control signal interactions that govern instruction execution.

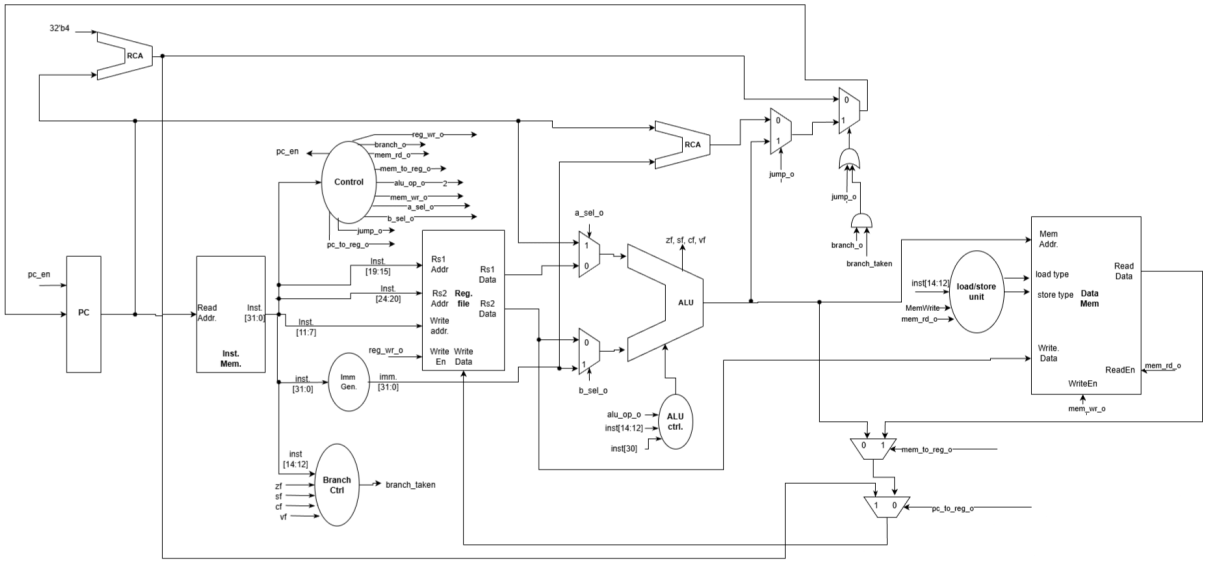
The implemented CPU integrates several key components, such as the program counter, instruction memory, control unit, register file, arithmetic logic unit (ALU), data memory, and multiplexers that manage data flow. Additional modules, including the immediate generator, control units that ensure correct instruction decoding and execution. The processor also features LED and seven-segment display outputs to visualize internal signals and verify proper operation during testing for the Nexys A7 100T FBGA.

The project implements 42 instructions of the RISC-V RV32I. However, it treats the fence and ecall ebreak instructions as terminate instructions that stop the program.

Implementation

The implementation was done using Verilog on the Vivado platform with the purpose of being modular and reusable.

Datapath



The proposed data path includes 9 main modules and several muxes and logic gates for control

1. The Program Counter (PC)

holds the address of the current instruction being executed. After each instruction, the PC is updated to point to the next instruction — typically PC + 4 — or a branch/jump target address if control flow changes occur. It serves as the entry point for instruction fetching.

2. Instruction Memory

The Instruction Memory stores the program instructions. The PC provides the address to this memory, and the corresponding 32-bit instruction is output to be decoded. This memory is read-only during program execution.

3. Control Unit

The Control Unit interprets the opcode field of the instruction and generates all necessary control signals to direct data flow through the datapath. It determines operations such as

register writes, ALU operations, memory access, branching, and selection of multiplexer paths.

4. Register File

The Register File contains 32 general-purpose registers used to store operands and results. It allows two simultaneous reads (for source registers) and one write (for the destination register). Operands for the ALU and store operations are fetched from this module.

5. Immediate Generator (Imm Gen)

The Immediate Generator extracts and sign-extends the immediate value from the instruction based on its type (I-type, S-type, B-type, etc.). This immediate is used in arithmetic operations, branch offsets, and memory addressing.

6. ALU Control

The ALU Control Unit combines the high-level ALU operation signal from the main control unit with the function fields (funct3 and funct7) of the instruction to produce a specific ALU operation code. This ensures that the ALU performs the correct computation (e.g., add, subtract, AND, OR, shift).

7. Arithmetic Logic Unit (ALU)

The ALU performs all arithmetic and logical operations, such as addition, subtraction, bitwise AND/OR/XOR, and comparison for branch conditions. It also generates status flags (Zero, Sign, Carry, Overflow) used for branch decision-making.

8. Branch Control

The Branch Control module evaluates branch conditions based on ALU flags (e.g., Zero, Sign, Overflow) and the instruction's funct3 field. It determines whether the branch should be taken and provides a signal to update the PC accordingly.

9. Load/Store Unit

The Load/Store Unit interprets the instruction's funct3 field to determine the data width and sign-extension type for memory access (byte, halfword, or word). It provides load_type and store_type signals to the data memory to correctly handle LB, LH, LW, SB, SH, and SW operations.

10. Data Memory

The Data Memory is designed to be byte-addressable, enabling flexible access to 8-bit, 16-bit, or 32-bit quantities. The memory logic uses load_type and store_type signals from

the Load/Store Unit to select the correct byte lanes and perform sign or zero extension when reading

11. Multiplexers (MUXes)

Several Multiplexers control data flow between components:

- A-input MUX chooses between the register value and the PC value
- B-input MUX selects between register value and immediates for the ALU.
- Branch/jump MUXes select the next PC value (either sequential or target address).
- Write-back MUX chooses between the ALU result, memory output, or PC+4 for writing into registers.

12. Adders (RCA)

- Ripple-Carry Adders (RCAs) are used to compute new PC values:
- One adder calculates PC + 4 for sequential instruction flow.
- Another computes PC + immediate for the branch.

13. Interconnections and Control Signals

All modules are connected through buses that carry data and control signals. The control unit and multiplexers orchestrate the correct path of data flow for each instruction type, ensuring that only the relevant hardware components are active during each operation.

Implementation Files

The RISC-V single-cycle CPU is composed of modular Verilog files that correspond directly to datapath blocks. The *cpu.v* file integrates all components, while *top.v* handles external FPGA connections. Instructions are fetched from *inst_mem.v*, decoded by *control_unit.v*, and executed through the *alu.v* with control signals refined by *alu_control.v*.

The *reg_file.v* includes 32 general-purpose registers that supply operands to the ALU and store results, with multiplexers (*mux.v, nmux.v*) directing data flow. Immediate values are generated by *imm_gen.v* and branch_control.v determines branch outcomes. Memory operations use *load_store_unit.v* with *data_mem.v* to support byte and halfword addressing.

Supporting modules like *shifter.v*, *rca.v*, *register.v*, and *dff.v* provide arithmetic and storage primitives, while *defines.*v holds instruction constants. Display and debugging modules (*bcd.v, four_digit_seven_segment_driver.v*) handle output visualization. Together, these modules implement the five main stages, fetch, decode, execute, memory, and write-back, in a single clock cycle.

Data Paths per Instruction Format

R-Format

The R-type instructions are relatively simple. The control unit chooses both inputs for the ALU to come from the register file and enables writing to the register file.

I-Format:

The I type is similar to the R type except for the control unit choosing the immediate generator as the second source for the ALU.

JALR:

The JALR instruction is a special type of I instruction. The control unit chooses the immediate generator and the register value, and the two inputs to the ALU. The output is then passed to the PC MUXes so that the program can just to the appropriate instruction and save PC+4 to a register.

Load Instructions

The control unit activates the data memory for a read operation. The ALU computes the effective address by adding the base register and the immediate value generated by the immediate generator. The **load/store unit (LSU)** interprets the instruction type to determine whether to load a byte, halfword, or word, and performs sign extension if necessary. The data retrieved from memory is then written back to the destination register.

Store Instructions:

For store operations, the control unit enables memory write and disables register write. The ALU again calculates the effective address using the base register and the immediate. The **LSU** ensures the correct data width (byte, halfword, or word) is written to memory based on the instruction's function code.

B-Type (Branch) Instructions:

The branch control module evaluates the condition based on ALU flags (zero, sign, or overflow). If the condition is met, the branch target address is calculated using the immediate generator and added to the current program counter. Otherwise, the program continues sequentially with PC+4.

U-Type Instructions (LUI/AUIPC):

For U-type instructions, the immediate generator provides a 20-bit upper immediate. In the case of **LUI**, the immediate value is written directly to the destination register. For **AUIPC**, the ALU adds the immediate to the current PC to generate the result stored in the register.

J-Type (JAL) Instructions:

The **JAL** instruction functions as an unconditional jump. The ALU computes the new PC by adding the immediate offset to the current PC. Meanwhile, PC+4 is stored in the destination register to allow the program to return later.

FENCE/ECALL instructions

The control unit stops the program counter from updating. Effectively stopping the program.

Testing

A single comprehensive program with all the instructions was created and verified to be working correctly.(*Please see the testcases folder for screenshots).*