Single-Cycle RISC-V Processor

By: Aryan Karnati

1. Introduction

1.1 What Is RISC-V

RISC-V is an open-source instruction set architecture (ISA) that is used to develop processors for a wide range of applications, from embedded systems to high-performance computing. It is based on the reduced instruction set computer (RISC) principle, which emphasizes a streamlined set of instructions that can be executed rapidly and efficiently. Unlike proprietary ISAs such as ARM (used by companies like Apple) or x86 (used by AMD and Intel), RISC-V is royalty-free and modular. This allows designers to tailor the architecture to specific needs without licensing constraints. RISC-V's open-source model empowers developers with full control over processor customization, making it a powerful choice for cutting-edge applications in IoT, automotive systems, aerospace, and AI. Its flexibility and cost-efficiency have also made it increasingly popular in academic research and startup innovation, where proprietary architectures can be limiting.

1.2 Single Cycle Processor

A single-cycle processor is a type of CPU architecture in which each instruction is executed in exactly one clock cycle, regardless of its complexity. This design simplifies control logic and makes the instruction execution flow easy to understand and implement, making it especially useful for educational purposes and early-stage processor development. All operations, such as instruction fetch, decode, execute, memory access, and write-back, occur sequentially within the same clock cycle. While this architecture is straightforward and suitable for small-scale systems, it sacrifices performance and clock speed scalability due to accommodating the longest instruction path within one cycle. As a result, single-cycle processors are typically used in academic settings, low-power embedded systems, and FPGA-based prototypes.

1.3 Goals Of The Project

This project aimed to design and implement a working single-cycle RISC-V processor using SystemVerilog. The processor was built to support a core subset of the RV32I instruction set, including arithmetic, logic, memory, and branch operations. A key objective was to create a complete datapath and control unit that could execute each instruction in a single clock cycle. Along the way, the project aimed to reinforce a practical understanding of how processors fetch, decode, and execute instructions, while providing hands-on experience with digital design tools like ModelSim. This project also served as a stepping stone toward more advanced architectures, such as pipelined and multi-cycle processors, and helped build the foundational skills needed for work in hardware design, embedded systems, and computer architecture.

2. Processor Architecture Overview

2.1 Processor Block Diagram

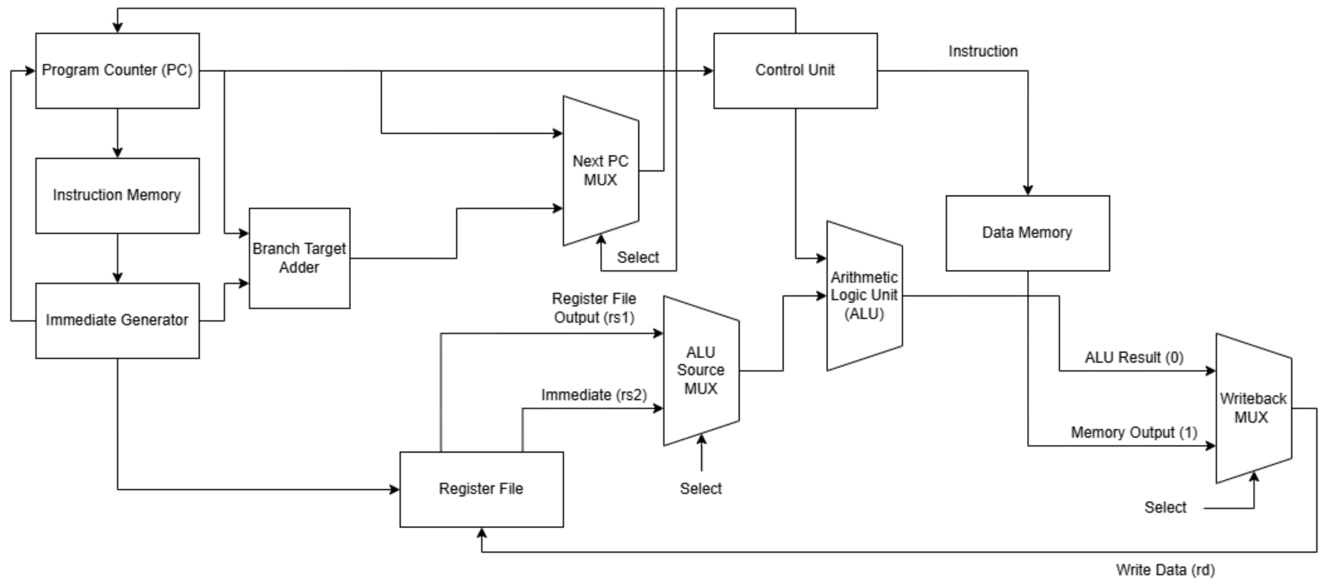


Figure 1: Block Diagram Of Data Path

2.2 ALU

The ALU performs arithmetic and logical operations such as addition, subtraction, AND, OR, and set-on-less-than. Operands come from the register file or immediate values, and the result is sent to either memory or the register file.

2.3 ALU Control Unit

The ALU Control Unit connects the main Control Unit and the ALU. Its main function is to generate the correct control signal to determine which operation the ALU should perform for a given instruction. The main Control Unit interprets the instruction's opcode to determine the instruction type, and the ALU Control Unit looks at the funct3 and funct7 fields to specify the exact ALU operation.

The Control Unit outputs a general ALUOp signal that classifies the instruction. The ALU Control Unit receives the ALUOp signal and the bits from the funct3 and funct7 instructions. Based on that information, the ALU Control Unit outputs a 4-bit control code telling the ALU what operation to perform.

2.4 Branch Target Adder

The Branch Target Adder is a component in the processor that calculates the destination address for branch instructions. When a branch instruction is executed, the processor may need to jump to a different part of the program instead of continuing to the next instruction. To figure out where to jump, the Branch Target Adder adds the current Program Counter (PC) value to an immediate offset from the instruction. This immediate value is extracted and sign-extended by the Immediate Generator. The result is the branch target address, which is the location in memory that the processor should go to if the branch condition is true. This value is then sent to the Next PC Mux, which decides whether to use it based on the outcome of the branch condition.

2.5 Control Unit

The Control Unit is responsible for interpreting the type of instruction being executed and generating the appropriate control signals to coordinate the rest of the processor. It uses the opcode field of the 32-bit instruction and determines how data should flow through the datapath and which operations should be performed during that clock cycle.

In a single-cycle processor, all operations occur in one clock cycle. The Control Unit takes a series of steps to ensure this happens correctly.

1. Decodes the opcode
2. Generates the control signals based on the instruction type
3. Enables or disables necessary components

2.5 Immediate Generator

The Immediate Generator is a component that extracts and prepares immediate values from instructions. In the RISC-V instruction set, many instructions include a number directly within the instruction itself, which is called an "immediate". This is used for things like address offsets, arithmetic, or comparisons. These immediate values are stored in different formats depending on the instruction type (I-type, S-type, B-type, etc.), and they often appear in non-contiguous bits. The Immediate Generator takes these bits, arranges them into the correct order, and performs sign extension so the value fits into a 32-bit format. This processed immediate value is then sent to other parts of the processor, such as the ALU or Branch Target Adder.

2.6 Instruction Memory

Instruction Memory stores the program’s instructions. Each instruction is 32 bits wide and is located at a specific address in memory. At the start of every clock cycle, the Program Counter (PC) provides an address to the Instruction Memory, and the instruction stored at that address is sent out to the rest of the processor. This instruction is then decoded and executed within the same cycle. In a single-cycle processor, the Instruction Memory is read-only during normal operation, meaning instructions are only fetched, not changed. It acts as the source of the program’s logic, supplying one instruction at a time in the order determined by the PC. Whether the processor executes a simple addition or jumps to another part of the program, the Instruction Memory is always the starting point for each instruction’s execution.

2.7 Next PC Mux

The Next PC Mux’s job is to decide what the Program Counter (PC) should be updated to at the end of each clock cycle. Normally, the processor moves to the next instruction by adding 4 to the current PC, since each RISC-V instruction is 4 bytes long. However, if the instruction is a branch or jump, the processor may need to go to a different address instead. The Next PC Mux chooses between the regular next address (PC + 4), the branch target address (PC + immediate), or the jump target address. It decides based on control signals from the Control Unit and the result of the branch condition. This decision happens within the same clock cycle, so the correct instruction can be fetched right away in the next cycle. By selecting the appropriate path, the Next PC Mux helps control the flow of the program and ensures that jumps and branches work as expected.

2.8 Program Counter

The Program Counter (PC) is a 32-bit register that holds the memory address of the instruction currently being executed. At the beginning of each clock cycle, the PC sends this address to the instruction memory to fetch the corresponding instruction. In the absence of a jump or branch, the PC increments by 4 to point to the next sequential instruction, as all RISC-V instructions are 32 bits (4 bytes) long. If a control transfer instruction is encountered, the PC instead updates to a new target address computed by adding an immediate offset to the current PC value.

However, if the current instruction is a control transfer, the PC is updated to a new target address instead. This address is typically calculated by adding an immediate offset to the current PC value. This update happens every clock cycle, allowing the processor to either continue with the next sequential instruction or jump to a new location based on the program's control flow.

2.9 Register File

The Register File is a collection of registers that are used to temporarily hold data that the processor needs during instruction execution. The RISC-V architecture utilizes 32 general-purpose registers, which are each 32 bits wide. They are labelled x0-x31.
During every clock cycle, the processor can read two source registers and write to one destination register. The instruction tells the processor which registers to read from and which ones to write to. These values are often used for calculations in the ALU or for memory operations. If the instruction needs to save a result, the processor writes it back to the specified register, as long as the write signal is active. Register x0 always holds the value zero, and it cannot be changed. This makes it useful when a constant zero is needed.

3. Module Descriptions

Module	Inputs	Outputs
ALU	- A - B - ALUControl	- result - zero
ALU Control Unit	- ALUOp - funct3 - funct7	- ALUControl
Branch Target Adder	- pc - imm_gn	- branch_target
Control Unit	- opcode	- RegWrite - ALUSrc - MemWrite - MemToReg - Branch - ALUOp
Immediate Generator	- instr	- imm_out
Instruction Memory	- addr	- instr
Next PC Mux	- pc_plus_4 - branch_target - jump_target - pc_src	- next_pc
Program Counter	- clk - reset - pc_next	- pc_out
Register File	- clk - we - rs1 - rs2 - rd - wd	- rd1 - rd2

4. Control Signals & Instruction Support

Instruction	Confirmation	Expected Result	Signals

add x3, x2, x1	Add x1 + x2	x3 = 15	RF.rd = 3
			RD.wd = 15
			ALU_result = 15
			RF.we = 1

sub x4, x2, x1	Subtract x2 - x1	x4 = 5	RF.rd = 4
			RD.wd = 5
			ALU_result = 5
			RF.we = 1

and x5, x1, x2	Bitwise AND x1 & x2	x5 = 0	RF.rd = 5
			RD.wd = 0
			ALU_result = 0
			RF.we = 1

or x6, x1, x2	Bitwise Or x1’	x2’	x6 = 15

xor x7, x1, x2	Bitwise XOR x1 ^ x2	x7 = 15	RF.rd = 7
			RD.wd = 15
			ALU_result = 15
			RF.we = 1

slt x8, x1, x2	Set if x1 < x2	x8 = 1	RF.rd = 8
			RD.wd = 1
			ALU_result = 1
			RF.we = 1

sw x1, 0(x0)	Store x1 to memory	DataMemory[0] = 5	DM.addr = 0
			DM.wd = 5
			DM.MemWrite = 1

lw x9, 0(x0)	Load from memory[0] into x9	x9 = 5	RF.rd = 9
			RD.wd = 5
			DM.rd = 5
			RF.we = 1

beq x1, x1, label	Compare equal, branch taken	Skip next instruction	ALU.zero = 1
			CU.Branch = 1
			Pc jumps to label

Simulation & Testing

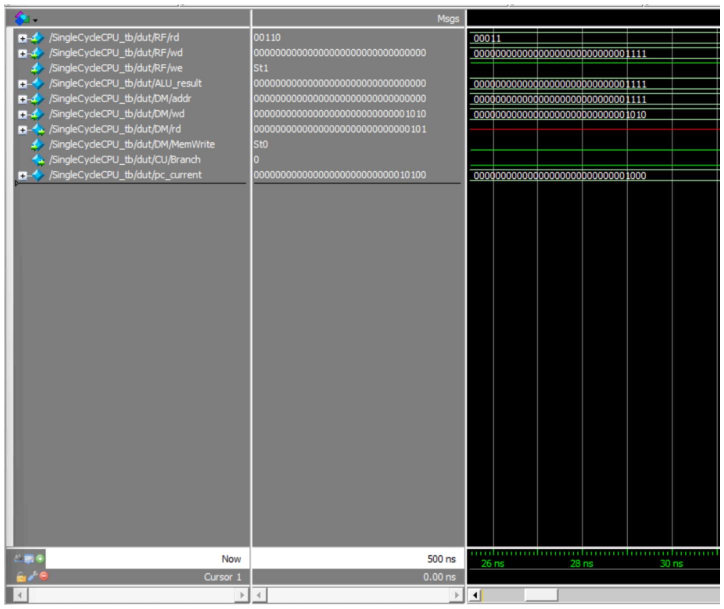
Each module in the processor was first tested individually using SystemVerilog testbenches. These testbenches provided controlled input values and monitored outputs to verify that each unit functioned correctly under various conditions. Once all modules passed unit testing, they were integrated into the SingleCycleCPU top-level module for system-level verification.

Simulation was conducted in ModelSim, where the instruction memory was loaded with test programs written in RISC-V assembly and converted to machine code. These test programs exercised various datapath combinations, including arithmetic sequences, memory access patterns, and conditional branching. UART-based debugging and $display statements were used to trace intermediate register values and ALU results throughout the simulation.

Key observations included:

- Register file reads and writes occurred in the correct cycle
- ALU results matched expectations across all tested operations
- Branch logic correctly updated the PC when conditions were met
- Memory loads and stores interacted correctly with data memory

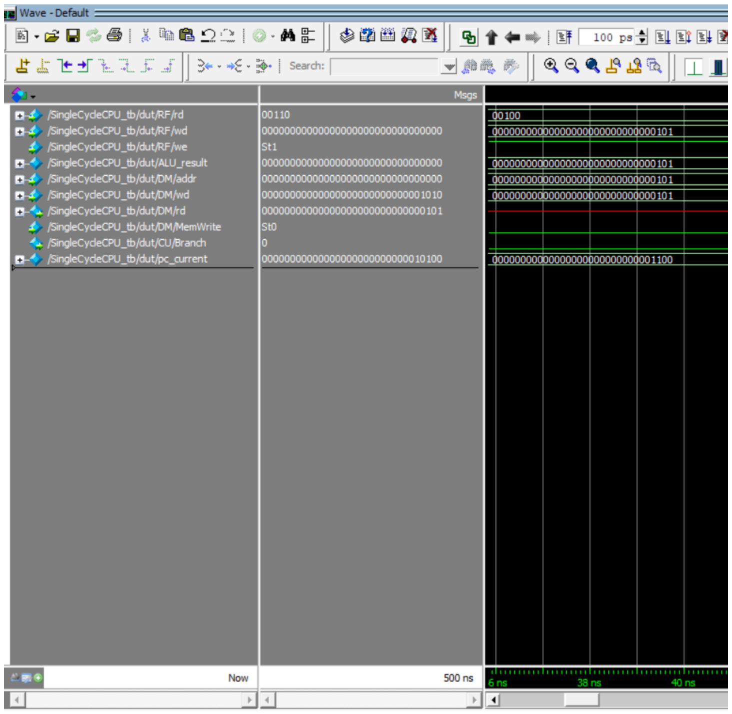
Waveform analysis helped resolve initial bugs in immediate generation and control decoding. The timing of control signals was verified to ensure that write enables, mux selects, and memory addresses were active during the proper phases of the cycle. Functional correctness was confirmed when all test programs produced the expected register and memory outputs.



```
                                   Msgs
/SingleCycleCPU_tb/dut/RF/rd          00110                               00011
/SingleCycleCPU_tb/dut/RF/wd          00000000000000000000000000000000    00000000000000000000000000001111
/SingleCycleCPU_tb/dut/RF/we          St1
/SingleCycleCPU_tb/dut/ALU_result     00000000000000000000000000000000    00000000000000000000000000001111
/SingleCycleCPU_tb/dut/DM/addr        00000000000000000000000000000000    00000000000000000000000000001111
/SingleCycleCPU_tb/dut/DM/wd          00000000000000000000000000001010    00000000000000000000000000001010
/SingleCycleCPU_tb/dut/DM/rd          00000000000000000000000000000101
/SingleCycleCPU_tb/dut/DM/MemWrite    St0
/SingleCycleCPU_tb/dut/CU/Branch      0
/SingleCycleCPU_tb/dut/pc_current     00000000000000000000000000010100    00000000000000000000000000001000

Now        500 ns      26 ns      28 ns      30 ns
Cursor 1   0.00 ns
```

Figure 2: Simulation Of ADD Instruction



```
Wave - Default
                                              Msgs
/SingleCycleCPU_tb/dut/RF/rd          00110                              00100
/SingleCycleCPU_tb/dut/RF/wd          00000000000000000000000000000000   00000000000000000000000000000101
/SingleCycleCPU_tb/dut/RF/we          St1
/SingleCycleCPU_tb/dut/ALU_result     00000000000000000000000000000000   00000000000000000000000000000101
/SingleCycleCPU_tb/dut/DM/addr        00000000000000000000000000000000   00000000000000000000000000000101
/SingleCycleCPU_tb/dut/DM/wd          00000000000000000000000000001010   00000000000000000000000000000101
/SingleCycleCPU_tb/dut/DM/rd          00000000000000000000000000000101
/SingleCycleCPU_tb/dut/DM/MemWrite    St0
/SingleCycleCPU_tb/dut/CU/Branch      0
/SingleCycleCPU_tb/dut/pc_current     00000000000000000000000000010100   00000000000000000000000000001100
Now                                   500 ns
```

Figure 3: Simulation Of SUB Instruction

Final Results

The single-cycle RISC-V processor was successfully implemented and simulated, achieving full functionality for a subset of the RV32I instruction set. All modules were built using synthesizable SystemVerilog and integrated into a complete datapath capable of executing one instruction per clock cycle.

Highlights of the final design include:

- Support for key instructions: ADD, SUB, AND, OR, XOR, SLT, LW, SW, and BEQ
- A modular and readable architecture with clearly separated functional blocks
- Successful simulation of arithmetic, memory, and branch operations
- Accurate control signal generation based on instruction decoding

While the processor is not optimized for speed or area, it serves as a strong foundation for more advanced architectures such as pipelined or multi-cycle processors. Future work may include hazard detection, forwarding logic, support for more instructions, and eventual synthesis onto an FPGA.