Chair of Computer Architecture and Operating Systems
TUM School of Computation, Information and Technology
Technical University of Munich

# Multi-cycle RISC V Processor

**Chitsidzo Varaidzo Nemazuwa**
TUM School of Computation, Information and Technology, Technical University of Munich
c.nemazuwa@tum.de
December 19, 2025

## 1 Complete Multi-cycle RISC V Diagram

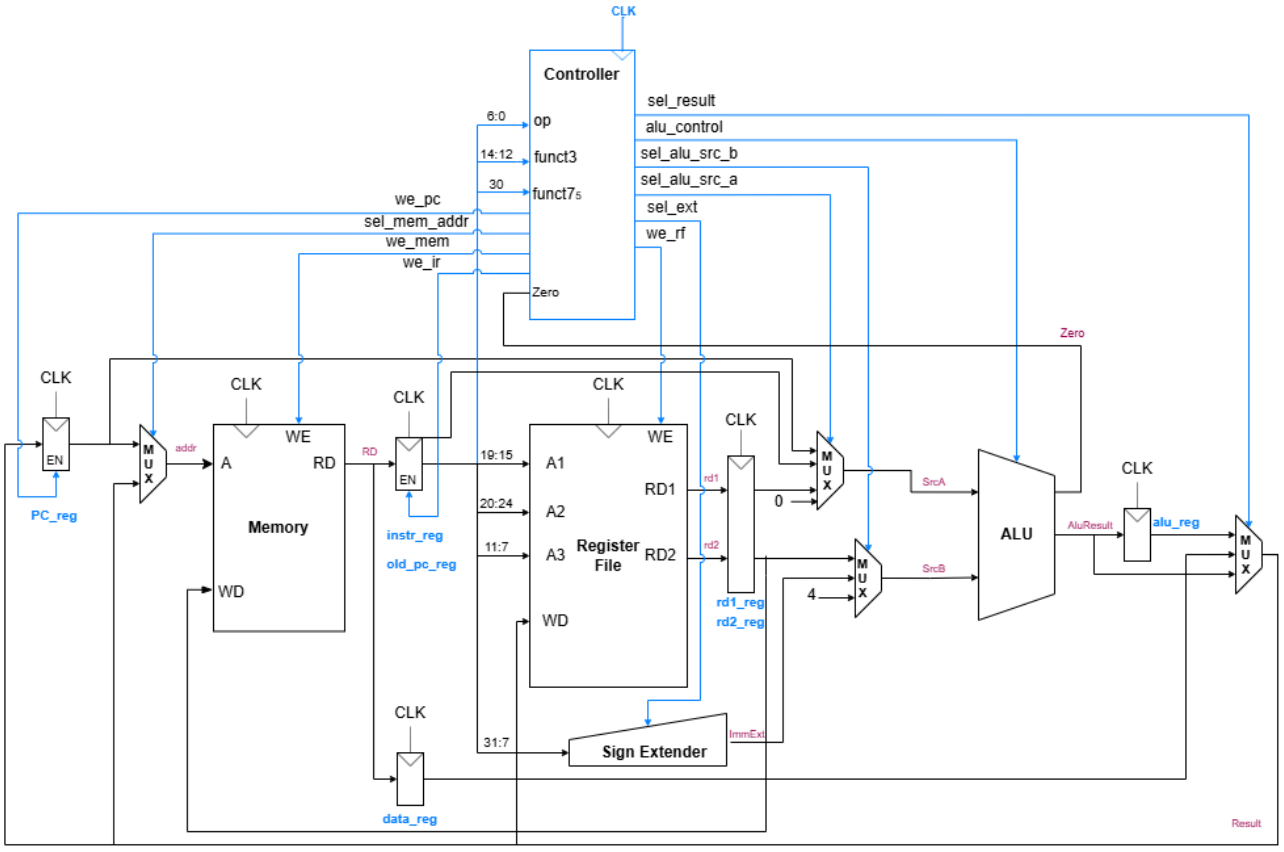


**Figure 1** Complete multicycle processor

Blue labels represent non-architecture registers, whilst the Purple labels represent wires connecting components.

# 2 Main Finite State Maschine Diagrams

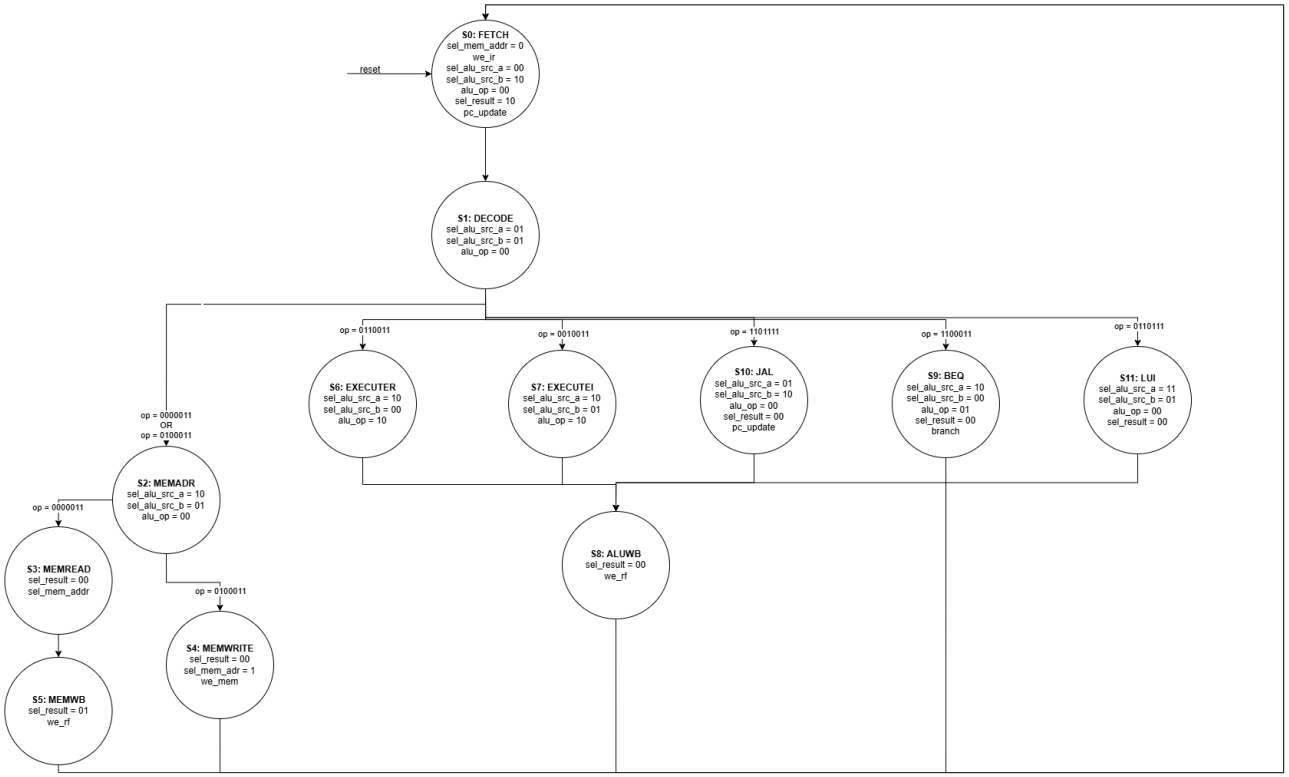


**Figure 2** Completed FSM Diagram for controller

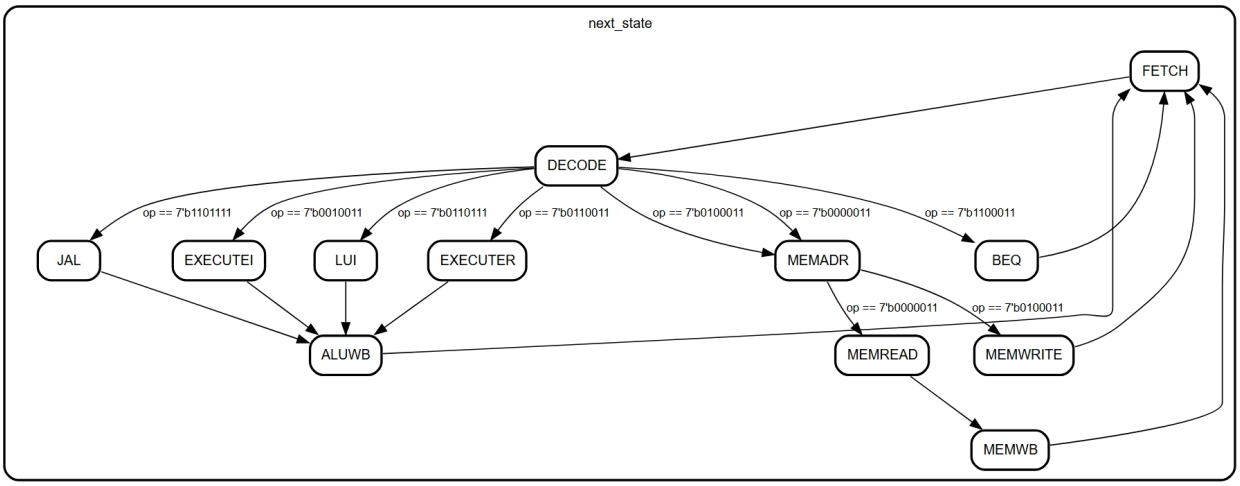


**Figure 3** FSM generated from code

# 3 Performance Analysis Solution Table

## 3.1 Theoretical CPI and Instruction Distribution

| Instruction Type | Theoretical CPI | Count in Program |
|---|---|---|
| R-Type (ADD, SUB, etc.) | 4 | 20 |
| I-Type Arithmetic (ADDI, XORI, etc.) | 4 | 30 |
| Load (LW) | 5 | 4 |
| Store (SW) | 4 | 4 |
| Branch (BEQ, BNE) | 3 | 7 |
| Jump (JAL) | 4 | 1 |
| Upper Immediate (LUI) | 3 | 6 |
| **Total Instructions** | **—** | **72** |

**Table 1** Theoretical CPI and Instruction Distribution

The multicycle processor requires three cycles for branches, four for R-type, I-type ALU, jump, and store instructions, and five for loads. The number of clock cycles per instruction (CPI) depends on the relative likelihood that each instruction is used.[1]

## 3.2 Actual results from testbench

| Instruction Type | Count | Total Cycles | Measured CPI |
|---|---|---|---|
| R-Type (ADD, SUB, etc.) | 20 | 80 | 4.00 |
| I-Type Arithmetic (ADDI, XORI, etc.) | 30 | 120 | 4.00 |
| Load (LW) | 4 | 20 | 5.00 |
| Store (SW) | 4 | 16 | 4.00 |
| Branch (BEQ, BNE) | 7 | 18 | 2.57 |
| Jump (JAL) | 1 | 4 | 4.00 |
| Upper Immediate (LUI) | 6 | 24 | 4.00 |
| **Overall** | **72** | **282** | **3.92** |

**Table 2** Measured CPI from Testbench Simulation

# 4 Multi-cycle Architecture Questions

**Question:** In Figure 1, some registers have write enable (en) while some do not. Explain why en is necessary or not.

**Answer:** Write enable signals are required for registers that must be updated selectively.

Registers with write enable:

- **Instruction Register (we_ir):** In a multi-cycle processor, the instruction must be held constant across multiple cycles. Without a write enable, a new instruction would overwrite the current one each cycle.
- **Old PC Register:** Updated together with the instruction register to store the current PC for branch and `jal` instructions.
- **Program Counter (we_pc):** The PC is only updated at specific times (instruction completion, branches, jumps). Without enable, it would increment every cycle and skip instructions.

Registers without write enable: Temporary registers are meant to capture new values every cycle and therefore do not require write enables.

**Question:** Why is the output of the sign extender not registered?

**Answer:** The output of the sign extender is made up of a combinational function of the instruction and therefore, wouldn't change whilst the current instruction is being processed, hence why it's not registered.

**Question:** Where is the slowest datapath (critical path) between two registers, given the delays of the components are constants as shown in Table 1?

**Answer:** The critical path taking the pseudo delays in table 1 into account is the path to read data from memory : From the alu_reg register(Register read - 100) through the Result and Adr multiplexers(30 * 2) to read memory(Mem Read - 200) into the Data register(data_reg).

**Question:** How does a multi-cycle architecture allow the clock period to be shorter than in a single-cycle one? What is the benefit?

**Answer:**

The execution time depends on the number of cycles and the cycle time. Single-cycle processors execute all instructions in a single cycle. In contrast, multi-cycle processors use varying numbers of cycles for different instructions. However, the multi-cycle processor does less work in a single cycle; therefore, it has a shorter cycle time. This results in the CPI being dependent on the relative likelihood that each instruction is used.

**Question:** Why might a multi-cycle design be easier (or harder) to extend with new instructions compared to a single-cycle design?

**Answer:**

In general, multi-cycle designs tend to be easier to extend with new instructions compared to single-cycle designs due to their flexibility and FSM control architecture. In a multi-cycle processor, new instructions can take as many clock cycles as needed by adding states to the FSM, allowing complex operations to be implemented without affecting the performance of simpler instructions. The clock period remains constant since it's determined by the slowest combinational path within a single state, not the entire instruction. This enables hardware reuse; the same ALU, memory ports, and datapaths can be orchestrated differently across multiple cycles for new instructions, avoiding the need for duplicated hardware.

In contrast, a single-cycle design requires all instructions to complete in one clock period, so adding a complex instruction forces the clock to be slowed down for every instruction, degrading overall performance. Additionally, single-cycle designs may require parallel datapaths or additional hardware resources to accommodate new instructions within the one-cycle constraint.

The primary trade-off is that multi-cycle designs require more complex control logic updates (modifying the FSM and adding state transitions), but this is significantly easier than redesigning the entire datapath and critical path timing required in single-cycle processors.

**Question:** Briefly describe the bug you have encountered.

**Answer:**

When extending the multicycle processor's ISA, several categories of bugs were encountered. FSM state transition bugs were common, where new instruction opcodes weren't properly decoded in the DECODE state, causing incorrect state transitions or infinite loops.

1. Control signal timing bugs occurred when write enables (we_rf, we_mem) or multiplexer select signals were asserted in the wrong states, leading to data being written prematurely or routed incorrectly through the datapath.

2. ALU control decoding errors arose when the mapping between alu_op and alu_control wasn't updated for new instructions, resulting in wrong arithmetic or logical operations being executed.

3. Immediate extension bugs occurred when the wrong sel_ext value was selected for new instruction formats, causing incorrect immediate values to be sign-extended and used in calculations.

These bugs were typically identified through waveform analysis in simulation, where comparing register values and control signals across clock cycles revealed which states had incorrect control signal combinations or where data wasn't being latched into pipeline registers properly.

## 5 GitHub Link

Github Repo: https://github.com/chits-nema/RISC-V-Processor/tree/RV_MC

## References

[1] Sarah L. Harris and David Harris. *Digital Design and Computer Architecture: RISC-V Edition*. Morgan Kaufmann, 1st edition, 2021.