Multi-Cycle RISC-V Processor Implementation Report

Beyrem Hadj Fredj

December 14, 2025

Repository Link: `https://github.com/BeyremHF/RISC-V-Processor-in-Verilog/tree/RV-MC`

1 Implementation Overview

This report documents the implementation of a multi-cycle RISC-V processor (RV32I subset) that executes instructions over multiple clock cycles using a Finite State Machine (FSM) controller.

1.1 Implemented Instructions

The processor supports 24 RISC-V instructions across 6 instruction types:

- **R-type:** ADD, SUB, AND, OR, XOR, SLT, SLTU, SLL, SRL, SRA
- **I-type (arithmetic):** ADDI, ANDI, ORI, XORI, SLTI, SLTIU, SLLI, SRLI, SRAI
- **Load:** LW
- **Store:** SW
- **Branch:** BEQ
- **Jump:** JAL
- **Upper Immediate:** LUI

1.2 Test Results

The processor was validated using a comprehensive test program covering all 24 instructions; 38/38 tests passed.

2 FSM Controller Design

The multi-cycle controller is implemented as a 12-state FSM that orchestrates instruction execution across multiple clock cycles.

2.1 State Descriptions

State	Description
S0 (FETCH)	Fetch instruction from memory using PC, compute PC+4
S1 (DECODE)	Decode instruction, read register file
S2 (COMPUTE)	Execute ALU operation for address/target calculation
S3 (MEM_ADDR)	Compute memory address for LW/SW
S4 (MEM_READ)	Read data from memory (LW only)
S5 (MEM_WB)	Write memory data to register file
S6 (MEM_WRITE)	Write data to memory (SW only)
S7 (EXEC_R)	Execute R-type ALU operation
S8 (EXEC_I)	Execute I-type ALU operation
S9 (ALUWB)	Write ALU result to register file
S10 (JAL)	Write PC+4 to register file for JAL
S11 (LUI)	Write immediate to register file for LUI

Table 1: FSM State Descriptions

2.2 FSM State Diagram

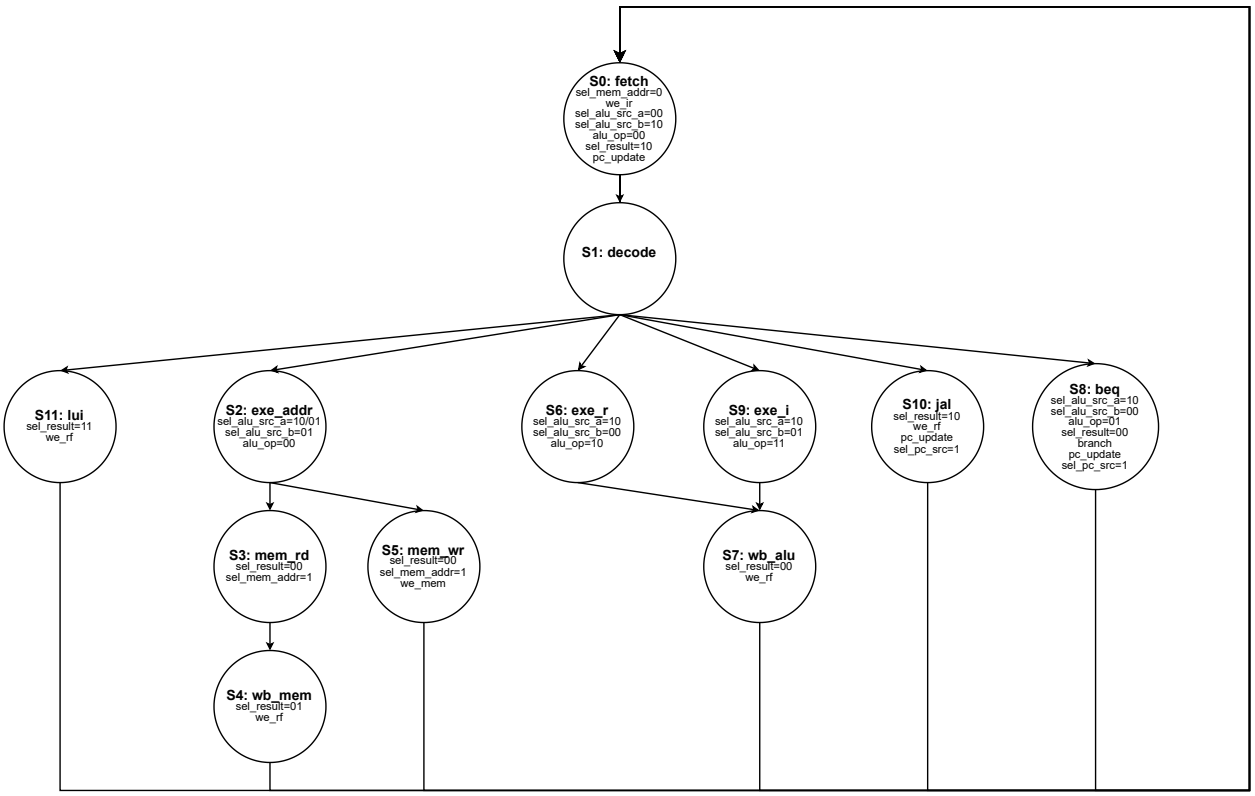


Figure 1: Complete FSM State Diagram with Control Signals

3 Control Signal Tables

3.1 Main Decoder Control Signals

State	we_pc	we_ir	we_rf	we_mem	sel_mem _addr	sel_alu _src_a	sel_alu _src_b	sel _result
S0	1	1	0	0	0	00	10	XX
S1	0	0	0	0	X	XX	XX	XX
S2	0	0	0	0	X	01	01	XX
S3	0	0	0	0	X	10	01	XX
S4	0	0	0	0	1	XX	XX	XX
S5	0	0	1	0	X	XX	XX	01
S6	0	0	0	1	1	XX	XX	XX
S7	0	0	0	0	X	10	00	XX
S8	0	0	0	0	X	10	01	XX
S9	0	0	1	0	X	XX	XX	00
S10	1	0	1	0	X	XX	XX	10
S11	0	0	1	0	X	XX	XX	11

Table 2: Control Signals per State (X = don't care)

4 Performance Analysis

4.1 Cycle Counts per Instruction Type

Instruction Type	Cycles
LW (Load Word)	5
SW (Store Word)	4
R-type	4
I-type (arithmetic)	4
BEQ (Branch)	4
JAL (Jump and Link)	4
LUI (Load Upper Imm)	3

Table 3: Clock Cycles per Instruction Type

4.2 Execution Paths per Instruction Type

Instruction Type	State Sequence
LW (Load Word)	S0 → S1 → S3 → S4 → S5
SW (Store Word)	S0 → S1 → S3 → S6
R-type	S0 → S1 → S7 → S9
I-type (arithmetic)	S0 → S1 → S8 → S9
BEQ (Branch)	S0 → S1 → S2 → S0
JAL (Jump)	S0 → S1 → S2 → S10 → S0
LUI	S0 → S1 → S11

Table 4: FSM State Sequences for Each Instruction Type

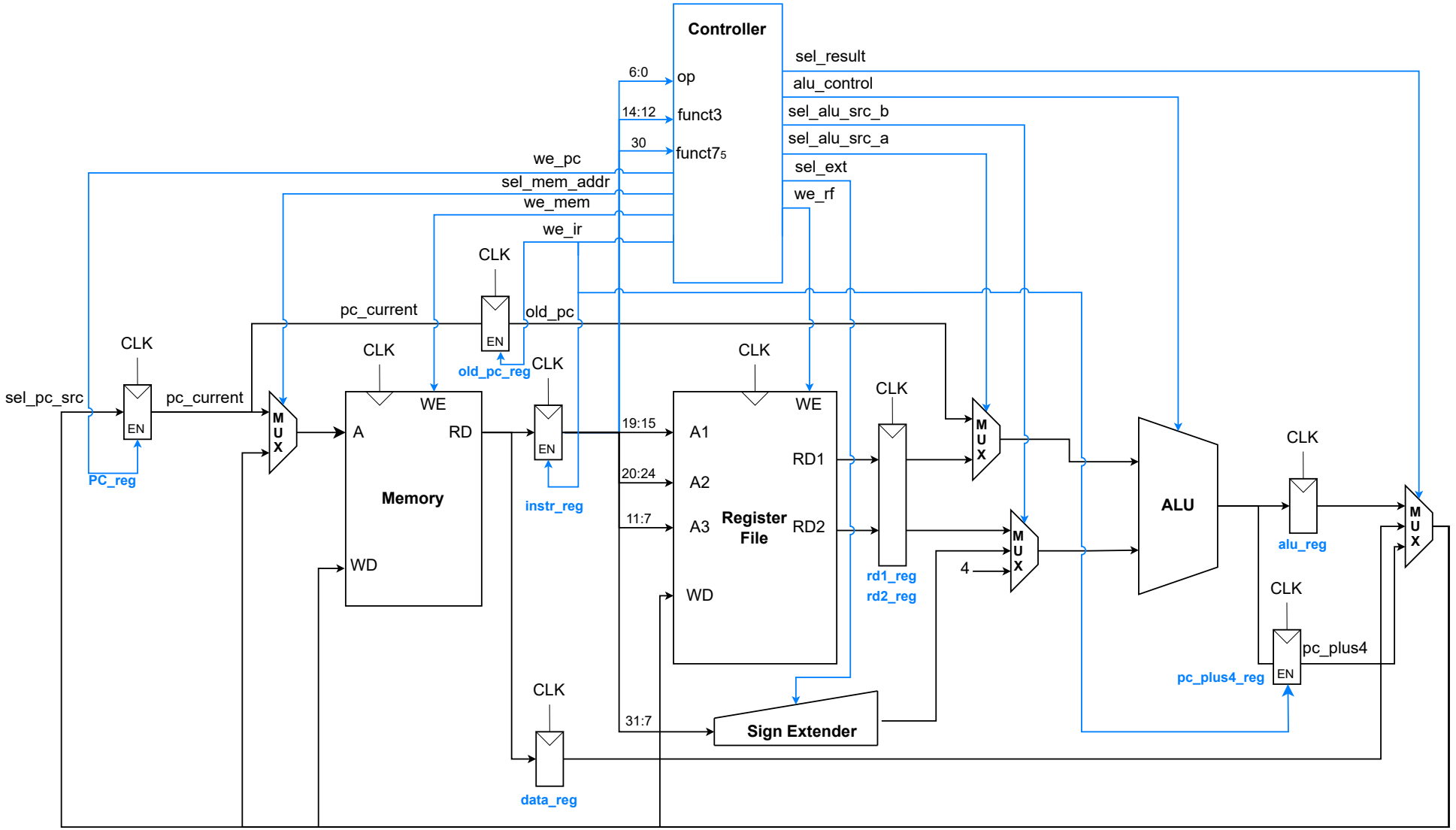
4.3 Average CPI Calculation

Based on the test program execution:

- **Total Instructions:** 38
- **Total Cycles:** 152
- **Average CPI:** 152 / 38 = **4.00**

Instruction Breakdown:

- R-type: 13 instructions × 4 cycles = 52 cycles
- I-type: 17 instructions × 4 cycles = 68 cycles
- LW: 1 instruction × 5 cycles = 5 cycles
- SW: 4 instructions × 4 cycles = 16 cycles
- BEQ: 1 instruction × 4 cycles = 4 cycles
- JAL: 1 instruction × 4 cycles = 4 cycles
- LUI: 1 instruction × 3 cycles = 3 cycles


Controller
op
funct3
funct7₅
6:0
14:12
30
sel_result
alu_control
sel_alu_src_b
sel_alu_src_a
sel_ext
we_rf
we_pc
sel_mem_addr
we_mem
we_ir
CLK
EN
pc_current
old_pc
old_pc_reg
sel_pc_src
PC_reg
MUX
A
WE
RD
Memory
WD
instr_reg
19:15
20:24
11:7
A1
A2
A3
Register File
RD1
RD2
WD
rd1_reg
rd2_reg
4
ALU
alu_reg
pc_plus4
pc_plus4_reg
31:7
Sign Extender
data_reg