RISC Processor Design using Verilog RTL

1. Project Overview

Title: 8-bit RISC Processor Design using Verilog RTL

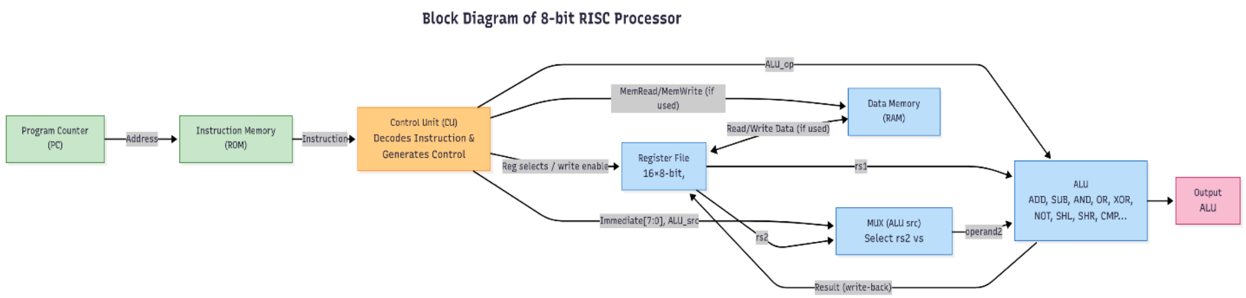
Description:
This project involves designing and simulating an **8-bit RISC processor** using **Verilog HDL**. It follows a **single-cycle architecture** where **fetch, decode, execute, memory access, and write-back** happen within **one clock cycle**. It includes an ALU, Register File, Control Unit, ROM, RAM, and supporting multiplexers. The processor supports arithmetic, logic, shift, comparison, move, jump, and halt instructions.

Objective:
To design, implement, and simulate a simple RISC processor that demonstrates instruction execution, register manipulation, and memory interaction.

Key Features:

- 8-bit data width
- 4-bit Program Counter
- 16 general-purpose 8-bit registers
- 16 × 16-bit ROM (instructions)
- 16 × 8-bit RAM (data)
- Single-cycle instruction execution
- Modular design for easy understanding and testing

2. Architecture Diagram


Block Diagram of 8-bit RISC Processor
Program Counter (PC)
Address
Instruction Memory (ROM)
Instruction
Control Unit (CU) Decodes Instruction & Generates Control
ALU_op
MemRead/MemWrite (if used)
Data Memory (RAM)
Read/Write Data (if used)
Reg selects / write enable
Register File 16×8-bit,
rs1
Immediate[7:0], ALU_src
rs2
MUX (ALU src) Select rs2 vs
operand2
ALU ADD, SUB, AND, OR, XOR, NOT, SHL, SHR, CMP...
Output ALU
Result (write-back)

Data Flow:

1. Instructions are fetched from ROM.
2. Control Unit decodes instructions and generates control signals.
3. ALU performs arithmetic/logical operations using Register File data.
4. Results are written back to Register File or stored in RAM.

3. Instruction Format

Bits	Field	Description
[15:12]	opcode	4-bit operation code
[11:8]	rs1	Source register 1
[7:4]	rs2	Source register 2 (or 0 for single-operand instructions)
[3:0]	rd	Destination register
[7:0]	immediate	8-bit immediate value (used in MOV/JMP)

Notes: MOV uses rd + immediate; JMP uses immediate as jump address.

4. Instruction Set

Opcode	Instruction	Function	Example
0000	ADD	rd = rs1 + rs2	ADD R1,R2,R3
0001	SUB	rd = rs1 - rs2	SUB R1,R2,R3
0010	MUL	rd = rs1 * rs2	MUL R4,R2,R3
0011	DIV	rd = rs1 / rs2	DIV R4,R2,R3
0100	AND	rd = rs1 & rs2	AND R5,R1,R2
0101	OR	rd = rs1 | rs2	OR R1,R2,R8
0110	XOR	rd = rs1 ^ rs2	XOR R6,R2,R1
0111	NOT	rd = ~rs1	NOT R3,R1
1000	SHIFT RIGHT	rd = rs1 >> rs2	SR R3,R1,R2
1001	SHIFT LEFT	rd = rs1 << rs2	SL R3,R1,R2
1010	LESS THAN	rd = (rs1 < rs2)	LT R1,R2,R3
1011	EQUAL	rd = (rs1 == rs2)	EQ R2,R3,R1
1100	NOT EQUAL	rd = (rs1 != rs2)	NE R1,R2,R3
1101	MOV	rd = immediate	MOV R1,0x0A
1110	JMP	Jump to immediate	JMP 0x04
1111	HALT	Stop processor	HALT

5. Module Descriptions

Module	Description	Inputs / Outputs
ALU.v	Performs arithmetic and logic operations	input [7:0] A,B, input [3:0] opcode, output [7:0] result
RegisterFile.v	Holds 16 × 8-bit registers; handles read/write	input clk, write_enable, rd_addr1, rd_addr2, wr_addr, wr_data, output rd_data1, rd_data2
ControlUnit.v	Decodes 16-bit instruction and generates control signals	input [15:0] instruction, clk, reset, output reg opcode, rs1, rs2, rd, ALU_op, ALU_src, RegWrite, MemWrite, MemRead, jump_flag, halt, jump_address, immediate
RAM.v	Stores temporary 8-bit data	input clk, addr, data_in, write_enable, output data_out
ROM.v	Stores program instructions	input addr, output data
MUX.v	Selects between multiple data sources	input [7:0] in0,in1, input sel, output out
RISC_TB.v	Testbench to simulate processor	Generates clock & reset, monitors outputs

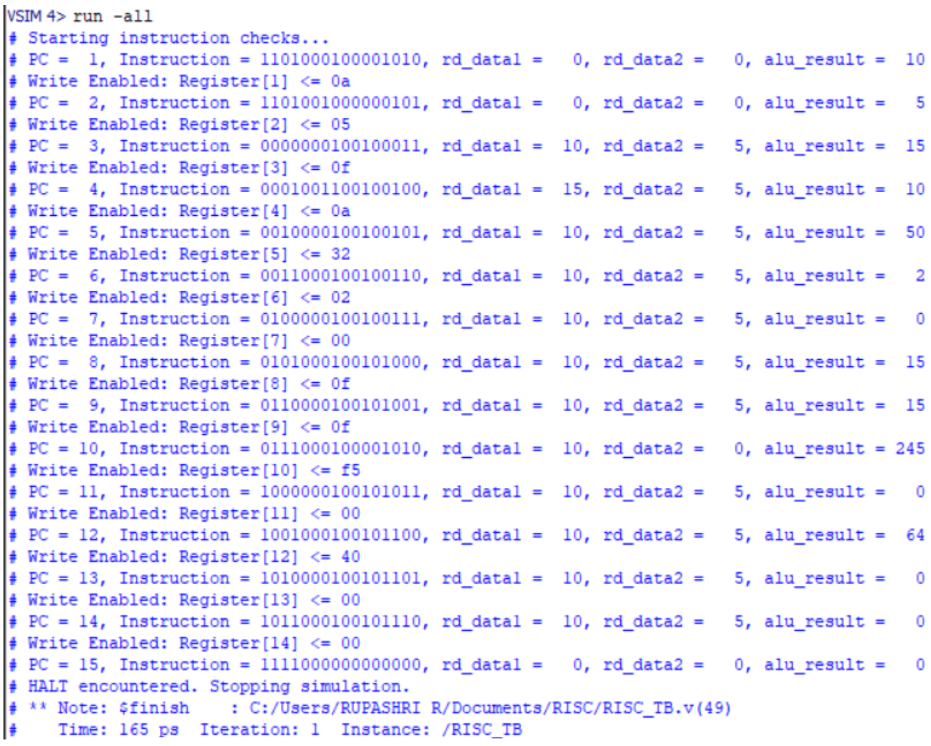
6. Testbench & Simulation

- Clock toggles every 10 ns to simulate timing.
- Reset initializes the processor.
- Instructions are read from ROM sequentially.
- Monitored signals: ALU result, register writes, memory accesses, program counter.
- Waveform outputs show instruction execution flow.

Sample Execution Flow:

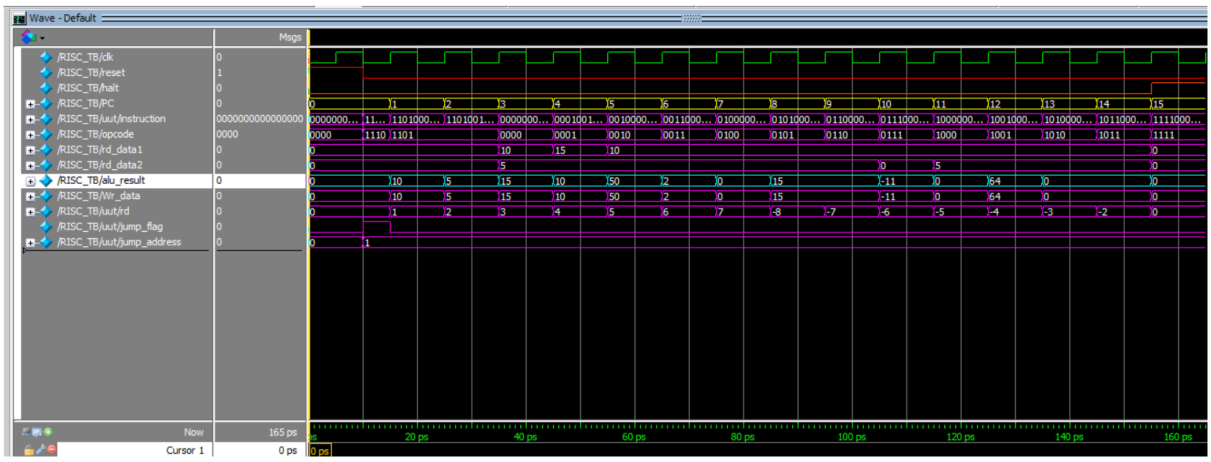
```
PC=1 : JMP

PC=2 : MOV R1,10

PC=3 : MOV R2,5

PC=4 : ADD R1,R2 -> R3

PC=5 : SUB R3,R2 -> R4
.
.
PC=15 : HALT
```

Transcript

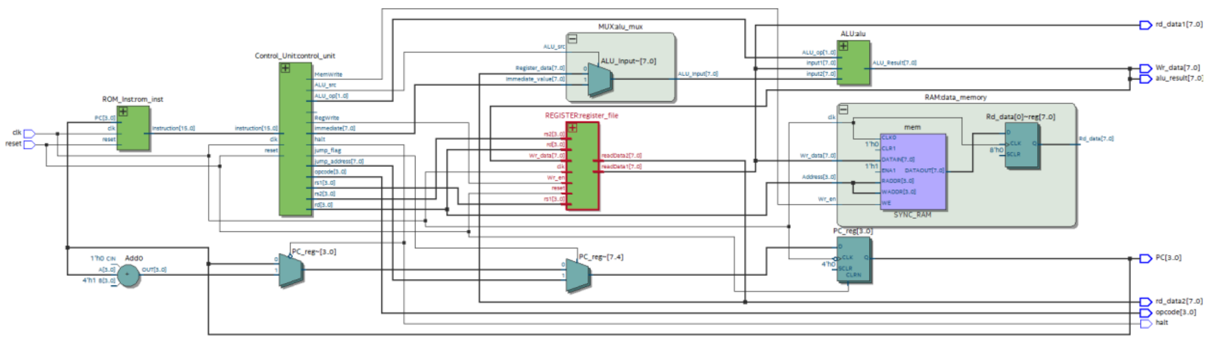


```
VSIM 4> run -all
# Starting instruction checks...
# PC =  1, Instruction = 1101000100001010, rd_data1 =   0, rd_data2 =   0, alu_result =  10
# Write Enabled: Register[1] <= 0a
# PC =  2, Instruction = 1101001000000101, rd_data1 =   0, rd_data2 =   0, alu_result =   5
# Write Enabled: Register[2] <= 05
# PC =  3, Instruction = 0000000100100011, rd_data1 =  10, rd_data2 =   5, alu_result =  15
# Write Enabled: Register[3] <= 0f
# PC =  4, Instruction = 0001001100100100, rd_data1 =  15, rd_data2 =   5, alu_result =  10
# Write Enabled: Register[4] <= 0a
# PC =  5, Instruction = 0010000100100101, rd_data1 =  10, rd_data2 =   5, alu_result =  50
# Write Enabled: Register[5] <= 32
# PC =  6, Instruction = 0011000100100110, rd_data1 =  10, rd_data2 =   5, alu_result =   2
# Write Enabled: Register[6] <= 02
# PC =  7, Instruction = 0100000100100111, rd_data1 =  10, rd_data2 =   5, alu_result =   0
# Write Enabled: Register[7] <= 00
# PC =  8, Instruction = 0101000100101000, rd_data1 =  10, rd_data2 =   5, alu_result =  15
# Write Enabled: Register[8] <= 0f
# PC =  9, Instruction = 0110000100101001, rd_data1 =  10, rd_data2 =   5, alu_result =  15
# Write Enabled: Register[9] <= 0f
# PC = 10, Instruction = 0111000100001010, rd_data1 =  10, rd_data2 =   0, alu_result = 245
# Write Enabled: Register[10] <= f5
# PC = 11, Instruction = 1000000100101011, rd_data1 =  10, rd_data2 =   5, alu_result =   0
# Write Enabled: Register[11] <= 00
# PC = 12, Instruction = 1001000100101100, rd_data1 =  10, rd_data2 =   5, alu_result =  64
# Write Enabled: Register[12] <= 40
# PC = 13, Instruction = 1010000100101101, rd_data1 =  10, rd_data2 =   5, alu_result =   0
# Write Enabled: Register[13] <= 00
# PC = 14, Instruction = 1011000100101110, rd_data1 =  10, rd_data2 =   5, alu_result =   0
# Write Enabled: Register[14] <= 00
# PC = 15, Instruction = 1111000000000000, rd_data1 =   0, rd_data2 =   0, alu_result =   0
# HALT encountered. Stopping simulation.
# ** Note: $finish    : C:/Users/RUPASHRI R/Documents/RISC/RISC_TB.v(49)
#    Time: 165 ps  Iteration: 1  Instance: /RISC_TB
```

Waveform



RTL View



7. How to Run

1. Open Quartus Prime → Add all Verilog files → Compile RISC_Top_Module.v.
2. Open ModelSim → Load RISC_TB.v.
3. Run simulation → Observe waveforms.
4. Modify ROM content to test different programs.

8. References

- *Digital Design and Computer Architecture*, David Harris & Sarah Harris
- Intel Quartus Prime and ModelSim User Guide
- Verilog tutorials from Component Byte