Design of the MiniRISC Processor

The purpose of this task is to design and implement a complete processor core called **MiniRISC** in SystemVerilog. The processor must implement the specified instruction set architecture (ISA), with a unified register file, arithmetic and logic instructions, immediate operations, and basic control-flow instructions (branches and jumps). The design must be synthesizable and fully functional under simulation.

Your design will be evaluated against **diverse test cases** to check **functional correctness**.

Processor Interface

The processor must expose the following external interface signals:

Signal	Direction	Width	Description
clk_i	Input	1	Clock input
resetn_i	Input	1	Active-low reset
instr_i	Input	32	Instruction input
pc_o	Output	32	Program counter (increments by 4 each cycle)
reg_write_o	Output	1	Register write enable
reg_waddr_o	Output	4	Register write address
reg_wdata_o	Output	32	Register write data

Register File

The processor must include a single unified register file with the following properties:
- 16 registers, each 32 bits wide.
- Register x0 (address 0000) must be hardwired to zero. Any attempt to write to this register must be ignored.
- All registers must reset to zero when resetn_i is asserted low.
- The register file must support single-cycle read and single-cycle write behavior.

Single-Cycle Requirement

The MiniRISC processor must be a **single-cycle processor**. Once the processor outputs a program counter value on pc_o, the testbench will immediately return the corresponding instruction on instr_i. That instruction must be fully executed in the same cycle, with the final result (register write, if any) visible at the end of the cycle. No multi-cycle execution or pipelining is allowed.

Instruction Set

Each instruction is encoded in **32 bits**. The opcode is always located in the **least significant 4 bits** (instr[3:0]). The remaining fields vary depending on the instruction type. The tables below describe the exact format of each instruction type.

R-type (Arithmetic and Logic)

Bits 31–16	Bits 15–12	Bits 11–8	Bits 7–4	Bits 3–0
unused (16 bits)	rs2 (4 bits)	rs1 (4 bits)	rd (4 bits)	opcode (4 bits)

Opcode	Mnemonic	Description
0000	ADD	rd = rs1 + rs2
0001	SUB	rd = rs1 - rs2
0010	AND	rd = rs1 & rs2
0011	OR	rd = rs1 | rs2
0100	SLL	rd = rs1 << (rs2[4:0])
0101	SRL	rd = rs1 >> (rs2[4:0]) (logical)
0110	SRA	rd = rs1 >>> (rs2[4:0]) (arithmetic)
0111	XOR	rd = rs1 ^ rs2

I-type (Immediate Instructions)

Bits 31–12	Bits 11–8	Bits 7–4	Bits 3–0
imm (20 bits)	rs1 (4 bits)	rd (4 bits)	opcode (4 bits)

Opcode	Mnemonic	Description
1000	ADDI	rd = rs1 + imm

Branch Instructions

Bits 31–12	Bits 11–8	Bits 7–4	Bits 3–0
imm (20 bits)	rs2 (4 bits)	rs1 (4 bits)	opcode (4 bits)

Opcode	Mnemonic	Description
1001	BEQ	if (rs1 == rs2) pc = pc + imm
1010	BNE	if (rs1 != rs2) pc = pc + imm

Jump Instructions

JUMPI format

Bits 31–4	Bits 3–0
imm (28 bits)	opcode (4 bits)

JUMPR format

Bits 31–8	Bits 7–4	Bits 3–0
imm (24 bits)	rs1 (4 bits)	opcode (4 bits)

Jump Opcodes

Opcode	Mnemonic	Description
1011	JUMPI	pc = pc + imm
1100	JUMPR	pc = rs1 + imm

Reset Behavior

When resetn_i is deasserted (low):
- pc_o must reset to zero.
- All registers in the register file must reset to zero.
- reg_write_o must be low, and no writes to the register file may occur.

Testing

Miners must test their design thoroughly to ensure correctness of every instruction defined above. While final evaluation will be performed by **validators**, miners are expected to verify functionality using their own testbenches before submission.

The evaluation environment will monitor pc_o, reg_write_o, reg_waddr_o, and reg_wdata_o to check that register file updates are correct at every step of execution.

Correctness will be determined by comparing the observed register write operations against a golden reference model across **diverse test cases**.

Simulation Requirements

Simulation will be performed using Verilator. To make your design testable:
- Provide a file list named rtl.f including all RTL sources.
- The file paths in rtl.f must be relative to the rtl.f location.
- Testbench compilation will be run from the same directory where rtl.f resides.

If your file paths are incorrect and the design fails to compile, the submission will receive **zero**.
- Only RTL must be submitted (no testbench code).
- The top module must be named minirisc_top.
- RTL must be synthesizable (no delays, DPI, or non-synthesizable constructs).

Evaluation Notes

Your processor will be evaluated by **validators** based on:
1. Correctness of execution for all defined instructions.
2. Proper PC behavior, including sequential updates, branches, and jumps.
3. Register file updates (writes observed via reg_write_o, reg_waddr_o, reg_wdata_o must match expectations).