4-bit CPU Design Project Specifications

Objective:

Your task is to design and implement a **4-bit CPU** in SystemVerilog, along with a small memory module, that executes a set of instructions inspired by the **RISC-V ISA**. You will be responsible for designing both the **data path** and the **control unit**, as well as creating a memory system where instructions and data can be stored, read, and written. The project will involve verifying your design using a SystemVerilog testbench.

Requirements:

1. **Instruction Set Architecture (ISA)**: You will implement a simple **4-bit instruction set**, which includes basic arithmetic, logical, and control operations. These instructions will be executed by your CPU. Here is the instruction set you need to implement:

Mnemonic	Description	Opcode
`ADD`	Add two registers	`0000`
`SUB`	Subtract two registers	`0001`
`AND`	Bitwise AND	`0010`
`OR`	Bitwise OR	`0011`
`XOR`	Bitwise XOR	`0100`
`LD`	Load from memory into register	`0101`
`ST`	Store from register to memory	`0110`
`JMP`	Jump to address	`0111`
`BEQ`	Branch if equal	`1000`
`BNE`	Branch if not equal	`1001`

2. **Instruction Format**:
    - Instructions are **8 bits** wide.
    - The format for instructions is: `4-bit Opcode`, `2-bit Source Register`, `2-bit Destination Register` or `Immediate Value`.

Components to Design:

1. **Data Path**: You will design a data path that includes:
    - **Register File**: 4 general-purpose registers, each 4 bits wide.
    - **ALU**: An Arithmetic Logic Unit that performs operations like `ADD`, `SUB`, `AND`, `OR`, `XOR`, etc.
    - **Memory**: A 16x4-bit memory where both instructions and data will be stored.
    - **Program Counter (PC)**: A register that holds the address of the current instruction being executed.
2. **Control Unit**:
    - The control unit will decode the opcode of each instruction and generate the necessary control signals to drive the data path.
    - You are required to implement a **Finite State Machine (FSM)** in the control unit to manage the various stages of instruction execution:
        - **Instruction Fetch (IF)**: Fetch instruction from memory.
        - **Instruction Decode (ID)**: Decode the instruction.
        - **Execution (EX)**: Execute the ALU operation or control logic.
        - **Memory Access (MEM)**: Read from or write to memory (for `LD` and `ST` instructions).
        - **Write Back (WB)**: Write results back to registers.
3. **Memory Module**:
    - You will create a memory module that stores both instructions and data. This memory will support both **read** and **write** operations.
    - The memory will be organized as **16 locations**, each holding **4 bits**.
    - You must ensure that **write operations (e.g., `ST`) do not overwrite instructions** in memory by either:
        - Using separate instruction and data memory, or
        - Protecting the instruction region of memory.

Implementation Tasks:

1. **Design the CPU Architecture**:
    - Implement the **ALU** to handle operations like `ADD`, `SUB`, `AND`, `OR`, etc.
    - Design the **Register File** to store intermediate results.
    - Implement a **Memory Unit** that allows loading and storing values.
    - Use a **Program Counter (PC)** to manage instruction sequencing.
2. **Control Unit with FSM**:
    - The control unit must be designed using a **Finite State Machine (FSM)** to handle the stages of instruction execution (IF, ID, EX, MEM, WB).

    - The FSM will generate control signals that drive the behavior of the data path, such as when to read/write from memory, update the program counter, etc.
3. **Memory Access**:
    - Implement **load (`LD`)** and **store (`ST`)** operations where:
        - The `LD` instruction loads a value from memory into a register.
        - The `ST` instruction writes a register value back to memory.
    - Make sure that the **memory is protected from overwriting instructions** when performing `ST` operations.
4. **Testbench**:
    - You will write a **SystemVerilog testbench** to verify the correct functionality of your CPU.
    - The testbench should load a series of instructions into memory and observe the behavior of your CPU as it executes them.
    - Automate the testing by comparing expected results with actual results from the CPU during simulation.

Deliverables:

1. **SystemVerilog Code**:
    - Complete SystemVerilog code for the **4-bit CPU**, including the data path, control unit, and memory module.
    - Each group is expected to have well-commented and organized code. All members must fully understand the code and be able to explain its functionality.
2. **SystemVerilog Testbench**:
    - A comprehensive testbench to verify your CPU design.
    - The testbench should test a variety of instructions and corner cases to ensure your CPU works correctly under all scenarios.
    - Automation of verification with expected vs actual results comparison is encouraged.
3. **Presentation**:
    - A **10-minute presentation** showcasing your CPU design, its architecture, and the working of your code.
    - The presentation should demonstrate **teamwork**, with each team member presenting a portion of the project.
    - **All group members must have a full understanding of the entire project** and be ready to answer questions about any part of the design.
    - Presentations will take place on **November 26, 2024**.
4. **Weekly Status Reports**:
    - A **group weekly status report** must be submitted by a different team member each week to **Rashid (rashid.iqbal@chipxprt.com)**.
    - The report should detail the team's progress, any challenges faced, and the plan for the upcoming week.

    - The final weekly report must be submitted by **November 16, 2024**.
5. **Team Collaboration**:
    - Teams are expected to meet on their own schedule to complete the project.
    - Any questions or issues should be posted in the **LinkedIn group**, where the team and Rashid will collaborate to find solutions. This ensures that questions are shared and everyone benefits from the answers.

Deadlines:

- **Final Report Submission**: **November 16, 2024**
- **Final Presentation**: **November 26, 2024** (We all meet together for an hour to go over the presentations)