EE 705: VLSI DESIGN LAB 2025
PROJECT- ALU & Decode

TAs: Lakshmi + Omkar

Note:

1. Submission is **only** through Moodle in the form of a PDF file upload.
2. Keep the port names and module names as mentioned in the question (including the case—whatever is capital must be capital & whatever is small must be small).
3. Code must be in **Verilog**

PART 1. ALU

1) Design the ALU module (module name: **riscv_alu**) for a RISCV processor, that comprises of the following ports:
    a. Input Ports:
        i. alu_op_i[3:0]:Specify which operation ALU will perform.
        ii. alu_a_i[31:0]:First 32 bit Operand
        iii. alu_b_i[31:0]: First 32 bit Operand.
    b. Output Ports:
        i. alu_p_o[31:0]: Result of ALU operation.
2) Functional Description of the riscv_ALU module:
    a. The ALU will perform several operations, including arithmetic operations, logical operations, and bitwise shifts. The ALU will be integrated into a RISC-V core, and its operations will be controlled by a 4-bit ALU opcode input.
        i. No operation (NOP).
        ii. Logical shift left. (Designed during Assignment -- remake for 32bit)
        iii. Logical shift right. (Designed during Assignment -- remake for 32bit)
        iv. Arithmetic shift right. (Need to be designed for 32 bit (don't use inbuild function)
        v. Addition of two operands. (Using BrentKung—designed during Assignment)
        vi. Subtraction of two operands. (Using BrentKung—designed during Assignment)
        vii. Bitwise AND operation.
        viii. Bitwise OR operation.
        ix. Multiply (16 bit * 16 bit = 32 bit). (Designed during Assignment)
        x. Comparison to check if the first operand is less than the second (unsigned comparison).
        xi. Comparison to check if the first operand is less than the second (signed comparison).

3) Make a test bench to test the complete module, showing all the input and output signals properly
4) The hardware needs to be tested on FPGA using ILA and VIO
5) Ensure that your complete module is compliant with RISCV ISA (RV32I)
6) Draw a block diagram showing the other modules which are dependent on this module + modules that ALU depends on. The diagram should clearly represent how the integration of the ALU module will happen with the other modules.

ADDITIONAL Materials and hints: None

PART 2: DECODE

1) Design the Decoder module (module name: **riscv_decode**) for a RISCV processor, that comprises of the following ports:

    a. **Input Ports:**

        i. Clk_i,rst_i,

        ii. Instruction Fetch Signals:

            1. fetch_valid_i: Indicates if the fetched instruction is valid.
            2. fetch_instr_i [31:0]: The 32-bit instruction to be decoded.
            3. fetch_pc_i [31:0]: The program counter (PC) value of the fetched instruction.

        iii. Branch Signals:

            1. branch_request_i: Signals a branch request.
            2. branch_pc_i [31:0]: The target PC for a branch.
            3. branch_csr_request_i: Indicates if the branch is due to a CSR (Control and Status Register) operation.
            4. branch_csr_pc_i [31:0]: The PC for a CSR-based branch.

        iv. Writeback Signals:

            1. writeback_exec_idx_i [4:0]: Destination register index for execution unit writeback.
            2. writeback_exec_squash_i: Indicates whether the execution writeback should be squashed.
            3. writeback_exec_value_i [31:0]: Value to be written back from execution stage.
            4. writeback_mem_idx_i [4:0]: Destination register index for memory stage writeback.
            5. writeback_mem_squash_i: Indicates whether the memory writeback should be squashed.
            6. writeback_mem_value_i [31:0]: Value to be written back from memory stage.
            7. writeback_csr_idx_i [4:0]: Destination register index for CSR writeback.
            8. writeback_csr_squash_i: Indicates whether the CSR writeback should be

squashed.

9. writeback_csr_value_i [31:0]: Value to be written back from CSR operations.
10. writeback_muldiv_idx_i [4:0]: Destination register index for multiplication/division unit writeback.
11. writeback_muldiv_squash_i: Indicates whether the multiplication/division writeback should be squashed.
12. writeback_muldiv_value_i [31:0]: Value to be written back from the multiplication/division unit.

v. Stall Signals:

1. exec_stall_i: Execution unit stall signal.
2. lsu_stall_i: Load/store unit stall signal.
3. csr_stall_i: CSR unit stall signal.

b. Output Ports:

i. Fetch Control Outputs:

1. fetch_branch_o → Signals whether a branch should be taken.
2. fetch_branch_pc_o → Specifies the target PC for a branch.
3. fetch_accept_o → Indicates if the decoder is ready to accept a new instruction.
4. fetch_invalidate_o → Invalidates the current fetched instruction (e.g., on branch misprediction).

ii. Opcode Validation Outputs:

1. exec_opcode_valid_o → Indicates that the decoded instruction is valid for ALU execution.
2. lsu_opcode_valid_o → Indicates that the instruction is a Load/Store operation.
3. csr_opcode_valid_o → Indicates that the instruction involves a CSR operation.
4. muldiv_opcode_valid_o → Indicates that the instruction is a multiplication/division operation.

iii. Decoded Instruction Outputs:

1. opcode_instr_o [57:0] → Encoded representation of the decoded instruction.
2. opcode_opcode_o [31:0] → Stores the original 32-bit fetched instruction.
3. opcode_pc_o [31:0] → Stores the program counter (PC) of the decoded

instruction.

iv. Register Operand Outputs:

1. opcode_rd_idx_o [4:0] → Index of the destination register (rd).
2. opcode_ra_idx_o [4:0] → Index of the first source register (rs1).
3. opcode_rb_idx_o [4:0] → Index of the second source register (rs2).
4. opcode_ra_operand_o [31:0] → Value of the first source register (rs1).
5. opcode_rb_operand_o [31:0] → Value of the second source register (rs2).

2) Functional Description of the riscv_decode module:

a. **Instruction Decoding**: Extracts opcode, source/destination registers, and immediate values from the fetched instruction. Manage register file access with multiple simultaneous read/write ports.

b. **Control Signal Generation**: Produces control signals for ALU operations, memory accesses, branching, and CSR interactions.

c. **Branch Handling**: Manages program flow changes based on branch instructions.

d. **Writeback Management**: Handles the result writeback from different execution units.

e. **Stall Handling**: Ensures proper operation by pausing execution when required due to dependencies or memory accesses.

i.

3) Make a test bench to test the complete module, showing all the input and output signals properly

4) The hardware needs to be tested on FPGA using ILA and VIO

5) Ensure that your complete module is compliant with RISCV ISA (RV32I)

6) Draw a block diagram showing the other modules which are dependent on this module + modules that decode depends on. The diagram should clearly represent how the integration of the decode module will happen with the other modules.

ADDITIONAL Materials and hints: None