CSE4372/5392 (Spring 2025)
Lab #2

This lab is due by February 13, with a 20% penalty per week day for being late.

In this lab, you will write the first iteration of the EX stage top-level module including any needed support modules. The EX stage will take input from the ID stage and output data to the MEM stage (and later forward data back to the ID stage for hazard resolution). The final module will have around 28 ports in total, but in this first iteration, we will limit the port list to just the 7 signals needed for the ALU operation.

The following steps will guide you through this process:

1. Create a module, rv32_ex_top, with the following port list:

```
// system clock and synchronous reset
input clk,
input reset,

// from id
input [31:0] pc_in,
input [31:0] iw_in,
input [31:0] rs1_data_in,
input [31:0] rs2_data_in,

// to mem
output reg [31:0] alu_out);
```

2. In your project top, use a push button (including the synchronizer from the seq_logic project in class) to provide the synchronous reset for the design. Use the 100 MHz clock (CLK100) as your system clock (we can use a slower clock later to make it easier to close timing).

3. You will need to decode the instruction word (iw_in) to calculate the operand, funct3, funct7, and the immediate/shamt values. The ALU output will be based on the value of the PC (pc_in), RS1 (rs1_data_in), and RS2 (rs2_data_in), and the immediate/shamt fields.

4. For steps 5-10, the ALU will be used to generate the value written to the destination register (rd) about 3 clocks later after leaving the WB stage. For step 11, the ALU will

be used to generate the address (ADDGEN) used for memory reads and writes after 1 clock in the MEM stage. The ALU output will be registered into a 32-bit flip-flop (alu_out) on the positive edge of the clock. If reset is asserted, alu_out should be loaded with 0 on the positive clock edge.

5. Using only combinational logic, design the logic to output the correct value of ALU output (alu_out) for the following arithmetic operations in the provided RV32I ISA spreadsheet:
- ADD (add regs), ADDI (add reg and immediate)
- SUB (subtract regs)

6. Using only combinational logic, design the logic to output the correct value of ALU output (alu_out) for the following logical instructions in the provided RV32I ISA spreadsheet:
- AND (and regs), ANDI (and reg and immediate)
- OR (or regs), ORI (or reg and immediate)
- XOR (exclusive-or regs), XORI (exclusive-or reg and immediate)

7. Using only combinational logic, design the logic to output the correct value of ALU output (alu_out) for the following shift instructions in the provided RV32I ISA spreadsheet:
- SLL (shift left by reg), SLLI (shift left by immediate)
- SRL (shift right logical by reg), SRLI (shift right logical by immediate)
- SRA (shift right arithmetic by reg), SRAI (shift right arithmetic by immediate)

8. Using only combinational logic, design the logic to output the correct value of ALU output (alu_out) for the following conditional set instructions in the provided RV32I ISA spreadsheet:
- SLT (store 1 if signed less than reg)
- SLTI (store 1 is signed less than immediate)
- SLTU (store 1 is unsigned less than reg)
- SLTIU (store 1 is unsigned less than immediate)

9. Using only combinational logic, design the logic to output the correct value of ALU output (alu_out) for the following jump instructions in the provided RV32I ISA spreadsheet:
- JALR (jump to address relative to reg value and store return address in reg)
- JAL (jump to address relative to PC and store return address in reg)

10. Using only combinational logic, design the logic to output the correct value of ALU output (alu_out) for the following load/add upper immediate instructions in the provided RV32I ISA spreadsheet:

- LUI (load immediate into upper 20 bits of reg)
- AUIPC (add immediate to upper bits of PC and store in reg)

11. Using only combinational logic, design the logic to output the correct value of ALU output (alu_out) for the following load and store instructions in the provided RV32I ISA spreadsheet:
- LB (load byte signed), LBU (load byte unsigned)
- LH (load half-word signed), LHU (load half-word unsigned)
- LW (load word)
- SB (store byte), SH (store half-word), SW (store word)

12. Use switches, LEDs, and signal tap as you wish to verify that the ALU output for all of these 31 operations is correct. Your test plan should be very detailed as an overlooked mistake now could cause a significant debugging problem later.

13. Demonstrate the requested operations to the TA and send your Verilog source .v or .sv files (don't send the entire project) to the TA for credit. 3 files (one top level, one for the ex stage module, one for the alu module) should be sent.