CSE4372/5392 (Spring 2025)
Lab #9

This lab is due at the time of the project defense on the last day of class.

In this lab, you will resolve the data hazard the occurs when reading a register after a register value is loaded from memory. The needed data in this case will only be available in the WB stage of the pipeline. The resolution of this hazard will require a combination of stalling and data forwarding for the best performance.

The following steps will guide you through this process:

1. Start with the project from Lab 8.

2. Add these signals to the rv32_id_top port list:

```
// register df from ex
input       df_wb_from_mem_ex,

// register df from mem
input       df_wb_from_mem_mem,
```

3. Add these signals to the rv32_ex_top port list:

```
// register df from wb (from mem_read)
input       df_wb_from_mem_wb,
input [4:0]  df_wb_reg,
input [31:0] df_wb_data,
```

4. In Lab 8, there was a writeback from memory control signal that is generated in ID that propagates through the registers to the WB stage. Connect this signal (hereafter called wb_from_mem) at project top to the ID stage from the EX and MEM stages (for stall generation) and to the EX stage from the WB stage (for data forwarding).

5. Connect the writeback data and writeback register signals from the WB stage to the EX stage in the project top (these are used for data forwarding).

6. Add logic in the ID stage to detect a data hazard between a register read of rs1 or rs2 in ID and a pending writeback from memory of an instruction in the EX stage. Generate a stall under this condition. This addresses ALU operations (rs1_data and/or rs2_data), address generation (rs1_data), memory write data (rs2_data), and branch condition (rs1_data and rs2_data) instructions.

7. Add logic in the ID stage to detect a data hazard between a register read of rs1 or rs2 when a branch instruction is decoded in ID and a pending writeback from memory of an instruction in the MEM stage. Generate a stall under this condition. This addresses the branch instruction rs1_data and rs2_data dependencies.

8. As long as the stall is asserted in the ID stage (for 1 or 2 clocks), insert a bubble (NOP) into the pipeline for the EX stage in the next clock cycle.

9. When the stall is asserted in the ID stage, create a signal to disable writes to the PC register in the IF stage. Route this signal through the project top back to the IF stage.

10. During the first clock in which a stall is asserted in the ID stage, temporarily store the PC and IW values currently present in the ID stage so that they are not lost when the new PC and IW values are clocked from the IF stage into the ID stage. On the next clock associated with the last stall for the instruction, these stored values of PC and IW should be registered into the EX stage instead of the normal IW value or the PC or NOP value. Generation of branch address and enable signals should also be controlled from this stored value in this situation.

11. Add logic to register rs1_reg and rs2_reg from ID to EX.

12. Add logic in the EX stage to detect a data hazard between the read from a register in ID and a pending writeback from memory of an instruction in the WB stage. In this case, rs1_data and/or rs2_data were read before the data from the memory was available. If this hazard is detected, use the data value from the WB stage instead of the corrupted data from ID that was read before the data was available.

13. Demonstrate operation of branch and ALU register read after a load instruction, showing the expected register results to the TA.

14. Send your Verilog source .v or .sv files (don’t send the entire project) to the TA for credit.