-

Think about where PC ALU result should come from

1. **How many stages is the datapath you've drawn? (i.e. How many cycles does it take to execute 1 instruction?)**
   There are 3 stages in the datapath we've drawn (FD/XM/W).

2. **How do you handle ALU → ALU hazards?**
   addi x1, x2, 100
   addi x2, x1, 100

   We take the output of the write-back stage MUX and feed it into the A_Mux and B_Mux. If the operand registers rs1, rs2 of Inst_XM are the same as the destination register rd of Inst_W, then the control logic selects the write-back stage value to be fed as inputs to the ALU.

3. **How do you handle ALU → MEM hazards?**
   addi x1, x2, 100
   sw x1, 0(x3)

   This is also handled with the same forwarding logic as above.

4. **How do you handle MEM → ALU hazards?**
   lw x1, 0(x3)
   addi x1, x1, 100

   This is also handled with the same forwarding logic as above.

5. **How do you handle MEM → MEM hazards?**
   lw x1, 0(x2)
   sw x1, 4(x2)

   also consider:
   lw x1, 0(x2)
   sw x3, 0(x1)

   This is also handled with the same forwarding logic as above.

6. **Do you need special handling for 2 cycle apart hazards?**

```
addi x1, x2, 100
nop
addi x1, x1, 100
```

We have two MUXes before the A and B registers in the Fetch-Decode stage to handle 2 cycle apart hazards.

7. **How do you handle branch control hazards? (What is the mispredict latency, what prediction scheme are you using, are you just injecting NOPs until the branch is resolved, what about data hazards in the branch?)**

   We predict that branches are always not taken, and when there is a mispredict we inject a NOP instruction into the XM stage. The control logic will handle whether a NOP should be inserted. The mispredict latency is 1 clock cycle.

8. **How do you handle jump control hazards? Consider jal and jalr separately. What optimizations can be made to special-case handle jal?**

   Both the jal and jalr control hazards are already handled by our forwarding logic. To special-case handle jal, we forward the immediate into an adder with PC and update the PC value with it.

9. **What is the most likely critical path in your design?**

   The most likely critical path in our design involves the forwarding logic, branch comparator, ALU, and control logic.

10. **Where do the UART modules, instruction, and cycle counters go? How are you going to drive uart_tx_data_in_valid and uart_rx_data_out_ready (give logic expressions)?**

    The cycle counter will just be an independent register that increments every clock cycle. The instruction counter will also be an independent register that increments only when a non-NOP instruction is being executed.
    uart_tx_data_in_valid = (addr == 32'h8000008 && instruction is store)
    uart_rx_data_out_ready = (addr == 32'h80000004 && instruction is load)

11. **What is the role of the CSR register? Where does it go?**
    The CSR register is separate from the RegFile and memory and its role is to contain some state that the RISC-V ISA testbench uses to check whether it is done. When a non-zero

value is written to the CSR register, the simulation ends. If the value is 1, this represents a success. If the value is greater than 1, this represents a failure.

12. **When do we read from BIOS for instructions? When do we read from IMem for instructions? How do we switch from BIOS address space to IMem address space? In which case can we write to IMem, and why do we need to write to IMem? How do we know if a memory instruction is intended for DMem or any IO device?**

    The top nibble (4 bits) of the address is used to partition the address space for different purposes. We read from BIOS for instructions when the top nibble of the address is 4'b0100. We read from IMem for instructions when the top nibble of the address is 4'b0001. In both cases, the address for the read must come from the program counter. We can only write to IMem when PC[30] == 1'b1. We need to write to IMem to load the program in from the BIOS. If the top 4 bits of the address computed by the ALU are 4'b00x1, the memory instruction is intended for DMem. If they are 4'b1000, the memory instruction is intended for I/O.

    Use a jump instruction to switch from BIOS to IMem PC

    Add reset signal to set PC to base of BIOS memory (0x40000000)