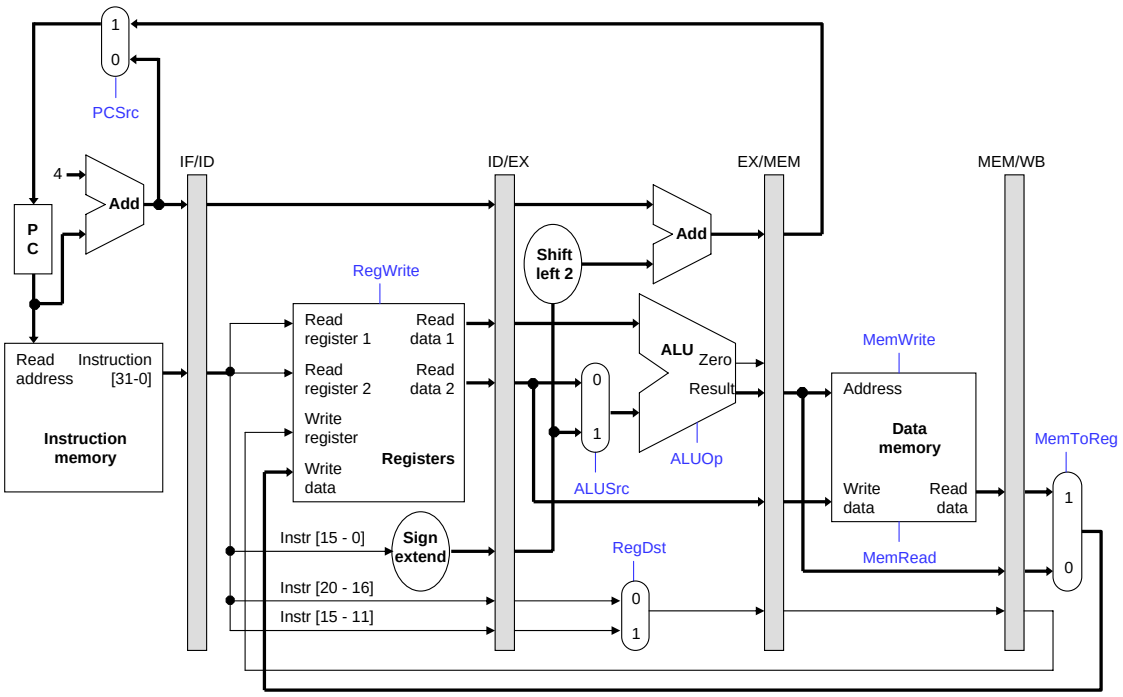
Recall from lecture that there are three types of hazards: **data hazards**, **control hazards** and **structural hazards**. In the 5-stage MIPS pipelined processor, all data hazards except load-use hazards (*e.g.*, `lw $t0, 0($a0)` followed by an instruction that reads `$t0`) can be eliminated via forwarding. Control hazards can be eliminated by stalling, although branch prediction can be used to reduce the number of stalls required.

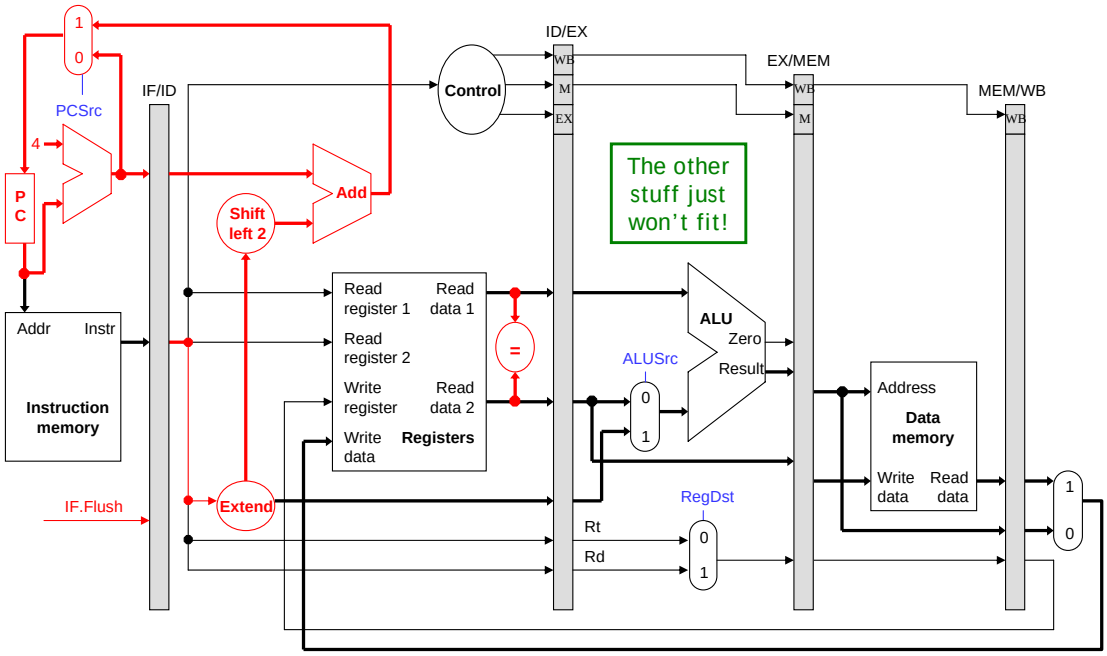


Consider the pipeline shown above.

1. In which stage is the PC incremented?
2. How many stalls do branches incur in the above implementation?

3. Can we do better?

Now consider a modified pipeline.



4. What about data hazards relating to branch instructions in the above pipeline?

Problems

1. Unconditional jumps:

   Your friend is job-hunting. In an interview, he is asked to explain what kinds of hazards are caused by `j` (unconditional jump) instructions in the classic 5-stage MIPS pipeline, and to explain how to deal with such hazards. He is given five minutes to answer, so he phones the only current CS 232 student he knows: you! Can you help your friend?

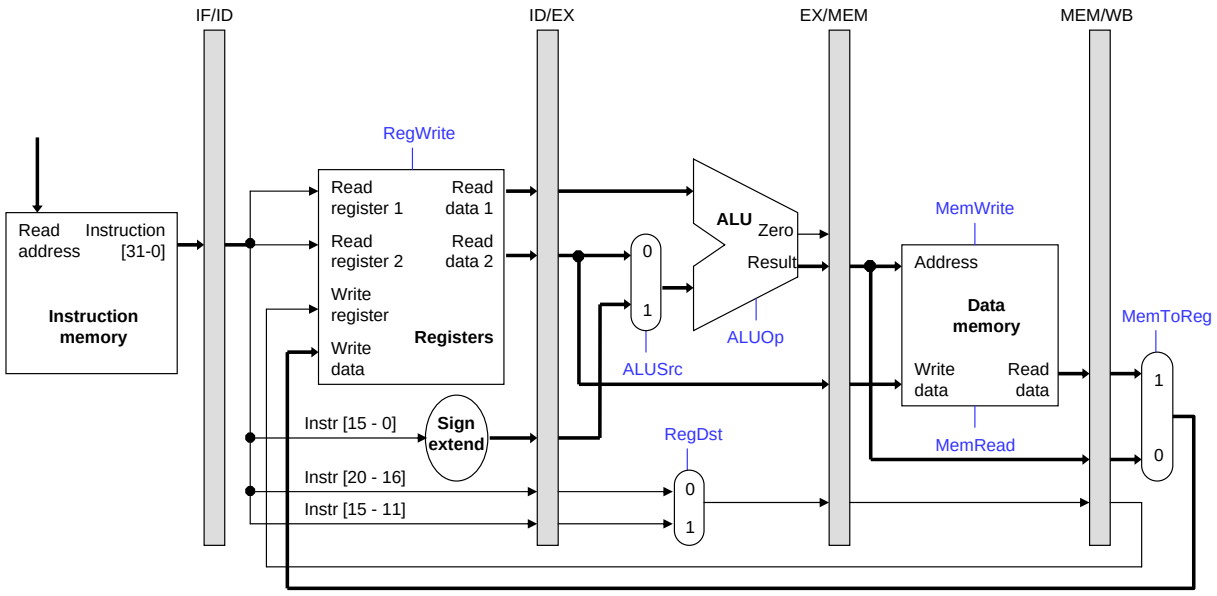
   Hint: Assume that the target address for a j instruction is computed in the ID stage.

2. Unconditional jumps revisited:

   Thinking over your friend's question some more, you realize that it is possible to make `j` less hazardous by resolving the target of the jump earlier.

   (a) On the datapath skeleton below, show the modifications that you will need to implement your idea. Note: The instruction `j target` sets the PC to the address of the instruction labeled target. This address is computed as follows: the two lowest bits are 00, the next 26 bits are bits 25-0 of the `j` instruction, and the 4 most significant bits are the PC bits 31-28.

   (b) For each new control signal that you add, show the necessary control logic. To simplify matters, specify the control logic using expressions like `(opcode == j)`, etc.



3. Control hazards:

   You have your dream (nightmare?) job as the head of the Architecture division of a cell-phone manufacturer that uses the classic 5-stage MIPS pipelined processor. An analysis of the programs that run on your cell-phones reveals many load-use hazards of the following special type: a `lw` instruction followed by a branch instruction that uses the value read from memory by the `lw`. For example:

   ```
   lw $t0, 0($a0)
   beq $t0, $t1, Label
   ```

   The problem with such sequences is that the pipeline is stalled (for at least one cycle) even when the branch outcome is correctly predicted. Your colleague wants the company to switch to a 6-stage MIPS pipelined processor with the following pipeline stages: IF, ID, EX, MEM, BR, WB. The new stage, BR, is the Branch Resolution stage in which both the target address and the resolution of the branch are computed.

   (a) Your colleague claims that all load-use hazards of the special type above can be eliminated with forwarding on the 6-stage pipeline. Convince yourself that she is right by writing a pipeline diagram that shows where forwarding must be performed.

Instruction	1	2	3	4	5	6	7	8
`lw`	IF							
`beq`		IF						

   (b) Your colleague also claims that the 6-stage pipelined processor does not require any fancy branch-prediction, because the optimal strategy is to not predict at all. Is she right?

   (c) A drawback of predicting not-taken in the proposed pipeline is that branches aren't resolved until after the predicted instructions reach the MEM stage. Why might this be a problem? Write a hazard detection equation to stall the pipeline to avoid this problematic case.

   (d) On the 6-stage pipeline, if the branch target was computed in the ID stage instead of the BR stage, which simple prediction strategy (no prediction, always predict taken, always predict not taken) would you recommend? Why?