Section 3: Pipelining

TA: Scott Beamer

Very Open Ended Question

Suppose you are an architect who has just completed the design of your 5-stage pipeline MIPS processor, that has all the possible bypass paths and logic to control them. It even handles exceptions correctly. Your manager congratulates you on designing a high performance processor, but breaks the bad news that you must reduce its cost by reducing the amount of bypass hardware/logic. This will reduce the performance, but you want to minimize that degradation. With a fully bypassed processor you can potentially forward from 3 stages. Could you obtain reasonable performance by only forwarding from 2 or even just 1 stage? Which stages would you forward from?

You aren't allowed to change the ISA for this problem, and currently it has no delayed branch, but if you really want you can spend additional hardware (to the manager's displeasure) to move the branch comparison to the decode stage. You can encourage programmers and compiler writers to order instructions in certain ways, but your processor must be correct for all valid uses of the ISA. Thus in this problem you will be thinking about hazards in the pipeline. Which ones are possible, how often they will occur, what penalty they will bring, and what ways there are to negate them. Clearly it may be hard to come up with cost numbers or program statistics instantly on your own, so feel free to also discuss methodologies you would use to satisfy your manager's request. You should definitely work in groups and discuss your ideas with others.

A Note on Exceptions

When dealing with exceptions, it is good to not loose sight of the end goal: seamlessly implementing the ISA. For most of the ISA's we study in this course, we give the programmer the illusion that instructions execute sequentially and only one executes at a time. For a single-cycle machine, this isn't too hard to do, but as pointed out in lecture, when we are pipelining, we are actually speculating that we will not have an exception. This is a good solution because they happen so rarely, but it is still important to get it correct. To do this there are two things to keep in mind:

- *Oldest Instruction's Exception First* - As a consequence of the sequential ISA, the exception that is from the oldest instruction should always win. Note this isn't the same as the oldest exception. If an instruction happens early in the pipeline and then a few cycles later an exception happens for an older instruction, the older instruction's exception needs to win.
- *Keep Architectural State Consistent* - The only way the system will ever be incorrect is if an error is visible in architectural state. Architectural state is what is visible to the programmer, and it includes the register file and main memory. If we mis-speculate on an instruction so it will have to be killed, its fine as long as its not somehow able to contaminate the architectural state.

To accomplish these goals we have a commit point, meaning we don't handle an exception or change architectural state until we are sure its the right thing to do and nothing else could go wrong (like another exception).