Slide 1

Administrivia

- Most/all remaining lectures will likely be “executive summaries”. There will be assigned reading, but you should read for the big picture / key ideas and skim/skip details.

Slide 2

Minute Essay From Last Lecture

- Questions:
  - Write a sequence of MIPS instructions that would be guaranteed to cause an arithmetic overflow exception.
  - Write a sequence of MIPS instructions that would be guaranteed to cause an “undefined instruction” exception.
- Answers?

Slide 3

Pipelining — Basic Idea

- Modeled after assembly line; many real-world analogies possible. Textbook describes a laundry "assembly line", with stages corresponding to washing, drying, folding, and putting away.
- Could base a pipelined implementation of MIPS on the steps we defined for the multi-cycle implementation, with one pipeline stage per step, as in figure 6.3.

Slide 4

Pipelining Complications — "Structural Hazards"

- Idea is that two things we want to do at the same time conflict — e.g., read instruction from memory and read data from memory.
- Only solution is to avoid. For MIPS, we could go back to separate instruction and data memories.

Slide 5

Pipelining Complications — "Control Hazards"

- Idea is that we need to make a decision but can't yet — e.g., we can't know what instruction should logically follow a conditional branch until we have the branch partly executed.
- Several possible solutions:
  - Stall — just wait until we can be sure.
  - Predict — make a guess, and if we guess wrong undo/redo.
  - Use delayed branches — always execute instruction after conditional branch, then jump / don't jump. (This is what MIPS does.)

Slide 6

Pipelining Complications — "Data Hazards"

- Idea is that we need data computed by one instruction before it would normally be available — e.g., two successive R-type instructions, or a load followed by an R-type instruction.
- Several possible solutions:
  - Stall — just wait until data is available. (Probably not a good solution.)
  - Add hardware for "forwarding" — special hardware to route results to next instruction in addition to regular destination. May or may not be possible.
  - Use delayed loads — don't allow instruction after a "load" to use the result. (This is what original MIPS did.)

Slide 7

Pipelining and MIPS — Sketch

- Basic idea shown in figure 6.12. Getting all details right — avoiding control and data hazards, flushing pipeline on exceptions — is nontrivial.

Slide 8

Pipelining — Summary

- Can improve instruction throughput, but individual instructions don't execute faster.
- Works better if instructions are "regular" and can be broken into equal-time steps.
- Nontrivial to get right!

Slide 9

Minute Essay

- None — quiz.