Slide 1

Administrivia

- Reminder: Homework 5 due Monday.

Slide 2

Generating Control Signals, Continued

- What's left for our single-cycle implementation? details of "control" CL block.
- First write out tables showing its inputs, outputs. Result is table in figure 5.27.
- Turn into AND and OR gates as described in Appendix C. Basic idea is to write down, for each output, a Boolean expression in terms of the 6 bits of opcode. Turning this into gates, we get figure C.5.

Slide 3

Single-Cycle Implementation, Summary

- The good news: Now we (supposedly) know how to build something from AND and OR gates and inverters that will execute (some) MIPS instructions.
- The not-so-good news: This isn't how we'd do it if we were designing a real processor — assumption that every instruction can be completed in a single cycle leads to inefficiency, because
  - Cycle time is determined by the longest instruction. This isn't too bad for the instructions we've discussed (though, e.g., `lw` takes longer than `j`), but would really be a problem if we were implementing multiply, divide, etc. Also, it means we can't "make the common case fast."
  - We need to duplicate some elements (e.g., ALU).
- So our next step will be to design a "multiple-cycle implementation" — divide (at least some) instructions into steps, with one step per cycle.

Slide 4

Multiple-Cycle Implementation

- Idea is to break each instruction down into steps (e.g., "fetch instruction from memory and increment PC"), and execute one *step* per cycle. Different instructions can take different numbers of cycles.
- Obviously this is more complicated to do. What are the benefits?
  - Presumably each step will involve less work than a full instruction, so clock cycle can be shorter. That might speed things up. (Is it guaranteed to?)
  - If we break things up "right", maybe we can reduce the amount of duplicated hardware.

Slide 5

Datapath for Multiple-Cycle Implementation

- First step is to (re)design the datapath, with the goal of eliminating duplicate hardware — single memory for instructions/data, single ALU.
- We'll assume this time that clock cycle is long enough for any of the following. (Why? Should become clear later.)
  - Memory access (read *or* write).
  - Register-file access (two reads *or* one write).
  - ALU operation.

  (How does this compare with cycle time needed for single-cycle implementation?)

Slide 6

Datapath for Multiple-Cycle Implementation, Continued

- First sketch of datapath — figure 5.30. Verify that this includes all needed pathways for data. We'll also need some "temporary storage" areas for things that need to be retained from one "step" to another. (This should become clearer later.)
- Add needed multiplexors to get figure 5.31. Compare this to datapath for single-cycle implementation (figure 5.17).

Slide 7

Minute Essay

- Give Boolean expressions for two more rows of truth table in figure 5.27 — `MemtoReg` and `Branch`. (Inputs, recall, are the 6 bits of the opcode.)