Slide 1

Administrivia

- Homework 4 graded. High (out of 40) 39, low 19, median 27.

Slide 2

Building "Control" for a Single-Cycle Implementation

- Goal of this section — finish designing a simple implementation of representative group of instructions.
- Simplify by requiring that all instructions must be completed in one clock cycle.
- Basic idea will be to combine datapath elements sketched so far, figure out what "control signals" we need, then figure out how to generate them — using as input the current state of the machine and the instruction being executed.
- Recall big picture: During each clock cycle, values from state elements "flow" through CL blocks, ultimately generating new values for (some) state elements, which will be stored at the end of the cycle.

Slide 3

Building "Control" for a Single-Cycle Implementation, Continued

- First step is to combine datapath pieces sketched out so far, noticing that:
  - If we need to use the same element for two different functions, we need two of them.

    (Does this explain why "instruction memory" is separate from "data memory"?)
  - If we need to get input for a particular element from more than one source, we need a multiplexor. E.g., second input from ALU can come from register or from sign extender.

Slide 4

Building "Control" for a Single-Cycle Implementation, Continued

- Combining the datapath elements talked about so far gives figure 5.13, which gives us what we need to fetch instructions in sequence and execute all but `j` instruction.

  (Notice that while most connections are shown as a single line, in reality they're multiple wires to carry multi-bit values.)
- We need multiplexors several places (where?).
- Multiplexors and ALU both have "control inputs". We have to figure out how to generate these (a.k.a. "control signals").

Slide 5

Generating Control Signals — ALU Control

- First look at generating control signals for ALU.
- Looking back at figure 4.17, ALU needs 3 bits of control input:
  - 2 bits to say whether output should come from adder, AND, OR, or less.
  - 1 bit to say whether adder's second input should be inverted.
- This allows for 8 possible combinations, but only 5 are used (which ones? see table on p. 353).
- How do these relate to the instructions we're implementing? Consider load/store, `beq`, and R-format instructions.

Slide 6

Generating Control Signals — ALU Control, Continued

- So we need to use the instruction to generate the 3 bits of ALU control input (as shown in figure 5.14). Do this in two parts:
  - Use instruction's op code to `ALUOp`.
  - Use `ALUOp` and instruction's function field to generate ALU control input.

  (Why two parts? sometimes simpler/smaller/faster to have two small "control units" than one big one.)
- We'll punt for now on the first part. To do the second part, start with a truth table showing desired outputs, as in figure 5.15.

  Notice that some inputs are "don't care" (output doesn't depend on this input).
- We can then turn this table into a CL block. Process can be completely mechanical, as described in Appendix C.

Slide 7

Minute Essay

- In figure 5.13, the multiplexor at the far right has two inputs. One comes from the "read data" output of the data memory and is 32 bits. Where does the other input come from, and how many bits is it? Where does the output go, and how many bits is it?
- In figure 5.13, there are three separate things that can perform addition (two adders and a full ALU). Do we need all three? Why or why not?