Slide 1

Administrivia

- Notes and sample programs from Friday before break on Web.
- Midterm grades mailed earlier today, including letter-grade estimate. Things to note if you're not happy with yours:
  - In computing the final grade, I'll drop the lowest quiz score. I didn't do that here.
  - Letter grades are conservative estimates.
  - There are still lots of points in play, and there will likely be a possibility of extra-credit points at the end of the term.

Slide 2

Building a Datapath, Review

- What we're doing is figuring out what "functional units" we need to implement a representative subset of MIPS instructions.
- Before the midterm, we talked about what's needed to store the program and step through it, and then what's needed for most of the instructions we want to implement.
- Recall also that we have two kinds of functional units — CL blocks and state elements.

Building a Datapath — Storing and Running Programs

- What we need to do: Store programs, fetch instructions one at a time in sequence.
- Datapath elements: instruction memory, program counter, adder.

Building a Datapath — R-format Instructions

- What we need to do: read contents of two registers, combine, store result in another register.
- Datapath elements: register file, ALU.

Slide 5

Building a Datapath — Load/Store Instructions

- What we need to do: read address from register, add offset from instruction, transfer info from register to data memory or vice versa.
- Datapath elements: register file, ALU, sign-extension unit.

Slide 6

Building a Datapath — Control-Flow Instructions

- What we need to do: For conditional branches, compute branch address from PC+4 and offset in instruction, subtract two registers and see if result is zero. For unconditional jumps, just replace PC with address from instruction.
- Datapath elements: register file, ALU, sign-extension unit, unit to do “left shift 2 bits”(really just routing), adder for PC.

  (Why can’t we just use the ALU to compute PC+4?)
- Connect them up as in igure 5.10.

Slide 7

A Little More About Branches

- We've oversimplified a little how branches actually work: Because of pipelining (to be discussed later), branches are "delayed" — instruction after branch is always executed.
- For simplicity, we'll show how to implement nondelayed branches.
- Similar caveat applies to what we say about load instructions — "delayed" in that value isn't available for next instruction.
- (Why didn't we mention this in writing MIPS-assembler programs? Assembler and simulator implement a "virtual machine" with nondelayed branches and loads. Compare with assembly-language version of code produced by compiler.)

Slide 8

Preview of Next Topic — Building a Simple "Control"

- Goal of next section — finish designing a simple implementation of representative group of instructions.
- Simplify by requiring that all instructions must be completed in one clock cycle. Not optimally efficient, but simpler.
- 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. (E.g., the ALU needs an input telling it which operation to perform — "add" for loads/stores/branches, something based on instruction itself for R-format instructions, etc.)

Minute Essay

- An easy one: How was your spring break?