Slide 1

Administrivia

- Reminder: Homework 5 due today.

  (Show LaTeX examples?)

- Homework 6 on the Web. Due in a week (or possibly the following Monday — we can revisit this next time).

Slide 2

Minute Essay From Last Lecture

- I didn't tally responses exactly, but at least a plurality said the workload seemed about right, with a few saying it seemed less than typical for a 3-credit course and more saying they thought they spent more than the expected/intended amount of time.

Slide 3

Designing a Processor — Recap

- The goal is to sketch out an implementation of a small but (we hope) representative selection of MIPS instructions, consisting of three groups:
  - Memory-access instructions (`lw`, `sw`).
  - Arithmetic/logical instructions (`add`, `sub`, `and`, `or`, `slt`). (Sidebar: Should we review additions to the ALU for `slt`? Figures B.5.10 and B.5.11?)
  - Control-flow instructions (`beq`, `j`).
- Implementation is in terms of combinational logic blocks and state elements, all ultimately constructed from AND and OR gates and inverters. Notice however the frequent use of layers of abstraction.
- To make it possible for state elements to be changed in some controlled way, we use "clocking".

Slide 4

Clocking — Recap/Review

- Hardware will include something that implements a "clock cycle".
- State elements' inputs are "sampled" during one phase of this cycle, and outputs can change during another phase.
- Length of cycle determines how complicated the various logic blocks can be (or vice versa).

Slide 5

Some Components We Want

- A register file.
- Some memory (which for simplicity we'll separate into instruction memory and data memory).
- Some way of representing where to find the "next" instruction — a "special purpose" register typically called "program counter" (PC).
- One or more ALUs (why more than one? should become obvious soon).
- "Control logic". (More soon.)
- Figures 4.1 and 4.2 sketch overall plan. How does Figure 4.1 relate to what we need to do . . .

Slide 6

Fetching Instructions and Updating PC

- For all instructions, start by getting instruction from memory. (What do we need? How does this map to Figure 4.1?)
- For most instructions, at some point we need to increment PC. (What do we need? How does this map to the figure?)
- And then the three groups of instructions do different things, but there are some commonalities . . .

Slide 7

Memory-Access Instructions

- Instruction includes two registers (one for base address, one for where to load into / store from) and a 16-bit displacement.
- Needed computation:
  - Add displacement to register containing address.
  - Use result to access memory, loading/storing to/from register containing data.
- How does this map to Figure 4.1? (Also see Figure 4.19.)

Slide 8

Arithmetic/Logic Instructions

- Instruction includes three registers (two for input operands, one for result).
- Needed computation:
  - Perform operation (with ALU) using values from two registers as inputs.
  - Save result in target register.
- How does this map to Figure 4.1? (Also see Figure 4.20.)

Slide 9

Control-Flow Instructions (`beq`)

- (`j` later.)
- Instruction includes two registers (data to compare) and a 16-bit displacement used to find target of branch.
- Needed computation:
  - Compare contents of two registers.
  - Compute address of branch target (PC plus displacement).
  - Use result of comparison to choose value for next PC.
- How does this map to Figure 4.1? (Also see Figure 4.21.)

Slide 10

Overview Revisited

- Notice that Figure 4.1 seems to have ways to do everything we need to do — paths for data to flow from one place to another, including into ALU(s) for computation.
- Notice also that for every instruction we're in some sense doing the same things (have each ALU compute something), but some results are essentially discarded. (Example — `beq` computes two "next instruction" addresses, but only saves one of them.) This is very typical of how things work at this level.

Slide 11

The "Datapath" — What's Missing

- Inputs to some blocks (e.g. PC) can come from more than one source. *That* can't work. So we need multiplexors to control which is used.
- Inputs to ALU / adder are 32 bits, but for some instructions we want to get one of them from 16 bits in insturuction. So we need something to extend that to 32 bits by extending sign.
- Both control-flow instructions include something that needs to be shifted two bits before being used to compute a target address, so we need to support that.
- Add these to "datapath" part of Figure 4.1 to get Figure 4.11. Leaves out "control" part, substituting not-connected-yet control inputs (blue in figures.)
- Right now we're showing the whole instruction as input to all elements that need part of it; we'll refine this later.

Slide 12

Control Logic

- So we have a "datapath" that can do things, but there are some inputs that aren't connected to anything. An analogy — the datapath is a puppet, and these inputs are its strings.
- Who/what pulls the strings? the "control logic" — combinational logic whose input is the current instruction plus any other needed information and whose output is those disconnected inputs to datapath.
- As mentioned in Appendix B, tools exist to transform truth tables into combinational logic, so our job is to come up with ones that will generate the signals we need for the datapath.
- Section 4.4 works through details. A lot of it should seem like common sense (viewed from the right angle?). Only potentially tricky part is input to ALU "which operation?" . . .

Slide 13

ALU Control Input

- ALU as designed in Appendix B uses 4 bits to represent which operation is to be done (2-bit input to multiplexor plus 2 "inverted input" signals). Seems like it would be simple enough for the main control unit to generate these directly, no?
- However, turns out to be even simpler to split functionality into two parts — generate a 2-bit "ALU operation" from just the opcode field, and then use that plus (for some instructions) the function field to tell the ALU what to do.

Slide 14

Instruction Execution Details

- Section 4.4 gives some details of what happens for each kind of instruction in the subset (initially omitting jumps). What we need to add for jumps — end of section.
- We won't discuss more in class, but you should read carefully — not to memorize, but to understand. May be useful to try to write down, for an example instruction, inputs to all the combinational logic blocks and state elements — as Homework 6 asks you to do. (Examples as time permits.)

Slide 15

Minute Essay

- None — quiz.