Slide 1

Administrivia

- (Via e-mail?)

Slide 2

Designing a Processor — Overview Revisited

- 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`).
  - 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. Note 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 3

Some Components We Want

- A register file.
- Some memory, which for simplicity we'll separate into instruction memory and data memory. Why? Simplifies some aspects of the design.
- 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.)
- Starter version of overall plan in Figure 4.1. How does this relate to what we need to do . . . First a small digression about 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 change as inputs change. (Some details in Appendix B. Textbook reviews some detail in Chapter 4. Okay to skim! The key point is that clocking is needed to avoid hardware race conditions.)
- Length of cycle determines how complicated the various logic blocks can be (or vice versa).

Slide 5

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 6

Memory-Access Instructions

- Instruction includes two registers (one for base address, one for where to load into / store from), 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?

Slide 7

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?

Slide 8

Control-Flow Instructions (`beq`)

- (`j` later.)
- Instruction includes two registers (values to compare), 16-bit displacement used to find target of branch.
- Needed computation:
  - Compare contents of two registers.
  - Compute address of branch target (PC+4 plus displacement).
  - Use result of comparison to choose value for next PC.
- How does this map to Figure 4.1?

Slide 9

Overview Revisited

- 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.
- 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 stores one back into the PC.) This is very typical of how things work at this level!

Slide 10

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.

  Figure 4.2 shows that addition.

Slide 11

Minute Essay

- Is this all making sense so far?