Slide 1

Administrivia

- None.

Slide 2

Minute Essay From Last Lecture

- Question: Sketch a combinational logic block with two inputs a and b and an output that's 1 when they're equal and 0 otherwise.
- Answer?

Slide 3

Combinational Logic Blocks and State Elements, Review

- Two types of “functional units” — CL blocks, which don’t have a notion of saved/internal state, and state elements, which do.
- The idea will be to combine them, and a clock, in such a way that every clock cycle:
  - At the start of the cycle the outputs of the state elements are based on their current values.
  - These values “ripple” through the CL blocks, generating new inputs for the state elements.
  - At the end of the cycle, these inputs are used to set new values for the state elements.

  See figures B.10 and B.11.

Slide 4

Combinational Logic Blocks, Recap / A Bit More

- Combinational logic blocks are circuits that “compute” outputs as Boolean functions of inputs.
- So, to design a CL block, it makes sense to first write down the function(s) to compute. A typical approach is to get this into a “standard” form, often a big OR whose terms are ANDs (“disjunctive normal form” — disjunction (OR) of conjunctions (ANDs)). Then it’s obvious how to turn this into gates and inverters. “Programmable logic arrays” (PLAs) are a standard approach.
- More about this, and examples, in Appendix B.

Slide 5

State Elements, Recap / A Bit More

- We looked last time at simple state elements.
- Recall that the point is to build something that "holds a value" and allows it to be set. This is why we need something like that feedback loop in the "set/reset latch" circuit — so the circuit's outputs (current state) depend not only on its current inputs but also on its previous inputs.
- We also typically want to connect these in a way that current state might connect to inputs that set next state — e.g., a counter. To make this work, we introduce a clock, and design the state elements so they only get updated once per "clock cycle". (Yes, this is the "clock cycle" of chapter 2.)

  (Aside: It's possible to design processors without a clock — "asynchronous" — but not how it's typically done now.)
- As a more complicated example, consider designing a "register file", as shown in figures B.18, B.19, B.20.

Slide 6

Building a Processor

- The overall idea involves two parts:
  - "Datapath" that stores values and does operations. Exactly what it does depends on "control signals" (e.g., "operation" input to ALU from chapter 4).
  - "Control" that generates control signals.
- A possibly useful analogy — datapath is a puppet, control is the thing pulling its strings.

Slide 7

Building a Datapath

- First we need a place to store instructions and a way to fetch them one at a time and execute them. For that we need:
  - Instruction memory (physically part of main memory and not processor, but logically can include in datapath).
    State element, right? What should values be? inputs? output?
  - Program counter.
    Also a state element, right? What should values be? inputs? output?
  - Adder (to increment program counter).
    CL block, right? what should inputs and outputs be?
- And then we connect these as shown in figure 5.5.

Slide 8

Minute Essay

- Here's what we talked about in chapters 3 and 4:
  1. MIPS instruction set.
  2. Translating C to MIPS assembly language.
  3. Translating assembly language to 1s and 0s.
  4. Representing numbers (integers and reals) in binary.
  5. Computer arithmetic.
  6. Turning Boolean functions into circuits.

  How well do you think you understand each of these?
- Reminder: Homework 4 due by 5pm today.