Slide 1

Administrivia

- We'll stay with six quizzes but have the next one next Wednesday.
- Next homework coming soon (I hope!).

Slide 2

Implementing the MIPS Architecture

- Goal of chapter 4 is to show how we could use the low-level building blocks described in Appendix C to implement a proof-of-concept subset of the architecture (instructions, registers, etc.) we've defined.
- "Proof of concept"? yes, the subset we'll implement may not be enough to do anything useful or interesting, but it should be enough to illustrate how we could implement the rest of the architecture.

Slide 3

Subset to Implement

- Representative memory-access instructions (`lw`, `sw`).
- Representative arithmetic/logical instructions (`add`, `sub`, `and`, `or`, `slt`).
- Representative control-flow instructions (`beq`, `j`).

Slide 4

Overview

- Very simplified view of what a processor does: Fetch next instruction. Figure out what it is and execute it. Lather, rinse, repeat.

  Implicit in this description is a notion of "next instruction", which normally moves through the stored program in sequence but not always (e.g., for control-flow instructions).
- What we have to work with:
  - Combinational logic blocks (Appendix C).
  - Sequential logic blocks ("state elements", Appendix C).

Slide 5

Clocking — Executive-Level Summary

- 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 6

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?).
- “Control logic”. (More soon.)
- Figures 4.1 and 4.2 sketch overall plan. How does Figure 4.2 relate to what we need to do . . .

Slide 7

Fetching Instructions and Updating PC

- For all instructions, start by getting instruction from memory. (What do we need? How does this map to Figure 4.2?)
- 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 8

Memory-Access Instructions

- Instruction includes two registers (one for memory address, one for data) 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 4.2?

Slide 9

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 4.2?

Slide 10

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 4.2?

Slide 11

Overview Revisited

- Notice that Figure 4.2 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 12

Minute Essay

- None really — sign in, unless questions.