Slide 1

Administrivia

- (See e-mail of earlier this week for reminders about due dates etc.)

Slide 2

Digression — Drawing Figures Programmatically

- I only had one person express interest in using LaTeX to draw figures as for Homework 5, but it's kind of cool, so I'll digress here . . .
- As I was preparing a sample solution for Homework 5 in a previous year, I got interested in whether there wasn't some nice tool to do this programmatically — rather than me drawing a bunch of gates with a drawing program and connecting them, well, it just seemed like something a computer could help a lot with, and similarly with the state machines.
- Being a LaTeX fanatic, I looked for LaTeX-based approaches, and found . . .

Slide 3

Digression — Drawing Figures Programmatically

- . . . something called TikZ (short for German for "TikZ is not a drawing program). There's quite a learning curve, but the results can be really nice.

  Examples on "sample programs" page.

  (I got carried away and spent part of my summer drawing some of the figures in Chapter 4 with it! And I *think* it really is easier for me now to produce nice-looking diagrams like the ones in Appendix B.)

- Take-home message, maybe: LaTeX is really good in general at converting "logical markup" into something more graphical. That this can apply to turning a logical(?) representation of a figure into something graphical — maybe surprising, maybe not? Other tools could work the same way (and maybe some do)?

Slide 4

Designing a Processor — Review/Recap

- So we've sketched the design of a processor that implements a supposedly representative set of instructions.
- A few more things to fill in . . .

Slide 5

Why Separate Instruction Memory and Data Memory?

- Design shows instruction and data memory separate.
- Why? isn't it all just ones and zeros? Yes, but . . . (Think about it a minute.)

Slide 6

Why Separate Instruction Memory and Data Memory? Continued

- Think about what has to happen on a `lw`. (Is this possible with a single memory?)
- (This is one of the textbook's "check yourself" questions.)

Slide 7

Implementing Jumps

- Discussion so far has omitted the `j` instruction. How should that work?
- We need to be able to get 26 bits from the instruction, shift them 2 bits left, combine with high-order bits of the current PC, and use that as the new PC. Figure 4.24 shows how.

Slide 8

Multi-Cycle Implementations

- So, we have a sketch for an implementation that executes one instruction per cycle. But clearly this isn't how all real systems work (if nothing else, many don't separate instruction memory from data memory).
- Why not? means cycle time is limited by length of longest path through the whole circuit, while many instructions can be done faster.
- What to do? break up work into multiple pieces . . .

Slide 9

Instruction Phases

- Work involved in fetching and executing a MIPS instruction can be split into phases:
  - Fetch instruction.
  - Read register operands and (at the same time) decode instruction. “At the same time” since inputs to the register file and inputs to the main control block all come from the instruction itself.
  - Do operation or address calculation.
  - Access data memory.
  - Write register result.
- How does this help? Two possibilities . . .

Slide 10

Simple Multi-Cycle Implementation

- One approach is to stick to the idea of executing one instruction at a time, but break things up so instructions potentially take multiple cycles. (How’s *that* going to help? Well . . .)
- Control logic becomes more complex — must do everything we were doing before, plus keep track of which phase we’re in. (Recall discussion of finite state machines from Appendix B.)

Slide 11

Simple Multi-Cycle Implementation, Continued

- However, one potential payoff is skipping unused phases — e.g.., the R-format (arithmetic/logic) instructions don't need to access data memory, and indeed we don't need separate instruction/data memories.

  (This kind of implementation — remember the discussion back in Chapter 1, in which different instructions took different numbers of cycles?
- A previous edition of the textbook lays out a design for this (details later, maybe).

Slide 12

Pipelined Implementation

- Another approach is to use "pipelining": Modeled after assembly line; many real-world analogies possible. Textbook describes a laundry "assembly line", with stages corresponding to washing, drying, folding, and putting away.
- Could base a pipelined implementation of MIPS on the same phases used for a multi-cycle implementation, with one pipeline stage per phase.
- How does this help? well, it doesn't make individual instructions faster, but it means you can get more of them done in a given time.
- Like the simple multi-cycle implementation, it means added hardware complexity . . .

Slide 13

Pipelining — Implementation Overview

- First might observe that the five phases into which we've divided instruction processing seem to map onto the picture of our datapath — what we're doing is breaking up the flow of information through it into steps(!).
- So the idea will be to somehow partition the datapath so we can have each piece working on a different instruction. But for that to work, we have to add groups of registers between pieces, so we save the results of one step for the next step.
- To be continued . . .

Slide 14

Minute Essay

- None really — just sign in.