Slide 1

Administrivia

- Quiz 4 graded. Most people did well! Solution online, linked from "Lecture topics etc." I graded on paper and plan to scan the graded papers and put them on Google Drive.

  (FYI — I haven't forgotten about Exam 1 but am working on grading!)
- Homework 7 due next Wednesday. One more homework, to be due the following Monday.
- We could do Quiz 5 Monday but it might be easier to wait until Wednesday since it covers about the same material?

Slide 2

More Administrivia

- FYI, I renamed the Google Drive folders for the course (to "CSCI2321-shared" and "CSCI2321-individual"), though I doubt that will affect how you access them.
- If you miss a scheduled class and watch the recording instead, you can still claim your attendance point by sending me a minute essay for it. You can help me keep accurate records by mentioning the date in the Subject line.

Slide 3

Quizzes (and Homeworks) Revisited

- I'll ask you in the minute essay how the way I asked you to do Quiz 4 worked for you, but — a show of "hands" maybe on whether it worked for you? (click "participants" and select . . .)
- For me it's more work, but I think collecting and printing everyone's work will go more smoothly next time (I'll get to test that with Homework 6 soon), and the main annoyance will be scanning in graded papers.

  Some minor problems go away if each of you makes sure your name appears *on every page* of what you turn in. (I won't ask that you apply this to Homework 6.) I'll remind you!
- Overall I plan to do the remaining quizzes the same way, unless there are objections.

Slide 4

Designing a Processor — Recap/Review

- We're working through the design of a processor that implements a subset of the MIPS architecture.
- Design starts with Figure 4.1, and previously we got as far as Figure 4.17, which (with the supporting tables) shows a complete design for `lw/sw`, selected arithmetic and logic instructions, and `beq`.

Slide 5

Instruction Execution Details — Examples Continued

- Example `add` previously. (Solution online as part of Homework 7 assignment.)
- Example `lw` previously. (Solution online as part of Homework 7 assignment.)
- Example `beq`. (Solution online as part of Homework 7 assignment.)

Slide 6

Homework 7 Help — Tracing Operation of the Processor Circuit

- (I'll go through these slides quickly in class; they may be a useful summary when you start doing the assignment. Time permitting I'll also put this information in the assignment too.)
- In this homework, you'll trace through what the circuit in Figure 4.17 is actually doing. Examples in lectures for 4/13 and 4/15. Idea is for you to trace through what the circuit actually does rather than what you think it should do. But the two should match!
- So, you start with what you know — current saved value of the PC and what's at that address (in instruction memory) and contents of selected registers and data memory locations — and work from there. Taking the first few steps . . .

Slide 7

Homework 7 Help, Continued

- Right away you can write down output of PC and input/output of instruction memory. The problems give you the machine language for the instruction; it may be helpful to split it into fields before going on.
- Now you can write down all the control signals, the inputs and output of the top left adder, and the register-number inputs to the register file. You can get the control signals from the table in Figure 4.18.
- Once you have those, you can write down outputs of the register file and start figuring out what the main ALU is doing. You can also determine whether the top right adder and the data memory will be used (based on control signals).

Slide 8

Homework 7 Help, Continued

- Figuring out what the ALU does . . . You need to determine what operation it's doing (based on the `ALUop` control signal and the instruction function field, as shown in Figure 4.13). You also need to determine what the second operand is (contents of a register? sign-extended value from instruction?), again using control signals.
- "And so forth" . . .

Slide 9

Designing a Processor, Continued

- 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 10

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 . . .

  (Minute-essay question.)

Slide 11

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 . . .
- Is what's being added enough that the instruction can work?
- What should the values of the control signals be? (Think about this on your own. Potential quiz/exam question!)

Slide 12

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, most 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 13

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 14

Simple Multi-Cycle Implementation

- One approach: Stick to the idea of executing one instruction at a time, but break things up so instructions potentially take multiple cycles.

  (This kind of implementation . . . Remember the discussion back in Chapter 1, in which different instructions took different numbers of cycles?)
- How's *that* going to help? Well . . .

Slide 15

Simple Multi-Cycle Implementation, Continued

- One potential payoff is skipping unused phases: E.g.., R-format (arithmetic/logic) instructions don't need to access data memory,
- Also, we don't need separate instruction/data memories.
- However, control logic becomes more complex: Must do everything we were doing before, plus keep track of which phase we're in. We can do that with a finite state machine (discussed in Appendix B, and it looks like we have time to say more about it now).
- Some previous editions of the textbook lay out a design for this.

Slide 16

Finite State Machines

- Typically represent sequential logic blocks as "finite state machines", consisting of
  - Input(s).
  - Output(s).
  - Current state (one of a set of possible states).

  (For those of you who've taken Theory: These are the finite automata probably covered there.)
- Define FSM by Boolean expressions that map
  - Current state and input(s) to next state.
  - Current state and (optionally) input(s) to output(s).

Slide 17

Finite State Machines

- Appendix B example: Controlling a traffic light. (Figures B.10.1 through B.10.3 and surrounding text.)
- In general, idea is to:
  - Assign numbers to states, and figure out how many bits are needed to represent this (only one for example, more if more than two states).
  - Write down Boolean expressions for bits of next state (one for each bit) based on bits of current state and inputs.
  - Write down Boolean expressions for output bits based on bits of current state and inputs.

Slide 18

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, doesn't make individual instructions faster, but means you can get more of them done in a given time.
- Like the simple multi-cycle implementation, it means added hardware complexity . . .

  (To be continued!)

Slide 19

Minute Essay

- The design sketched so far has two separate memory blocks, one for instructions and one for data. This turns out to be needed for the simplest implementation, one in which each instruction executes in a single cycle. Why? is there something different about the types of values to be stored, or is there some other reason? (*Hint:* Think about what has to happen for `lw`.)
- How did the way I asked you to do Quiz 4 work for you? Anything that would make it work better? I feel like there's just no good way to deal with quickly drawing pictures but hope you found an option that works okay for you?

Slide 20

Minute Essay Answer

- For `lw`, you need to be to both load the instruction and also load something from the specified address. (This is an open-ended version of one of the textbook's "check yourself" questions for section 4.3.)