Slide 1

## Administrivia

- Reminder: Quiz 3 Thursday.
- Homework 3 due a week from today (Web site was wrong).

Slide 2

## Minute Essay From Last Lecture

- A few people thought the 64-bit ALU would be faster. Um, no . . .
- One person (correctly but somewhat pedantically?) pointed out that we did say all ALU operations are supposed to take the same amount of time.

**Slide 3**

## The Big Picture, Revisited

- We've sketched what we need for the "datapath" part of a MIPS processor — logic blocks to perform arithmetic/logic operations (ALU) and store information (register file, RAM).
- Now we need something to control it — a sequential logic block.

**Slide 4**

## 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).
- Define FSM by Boolean expressions that map
  - Current state and input(s) to next state.
  - Current state and (optionally) input(s) to output(s).
- Appendix B example — controlling a traffic light. (Figures B.10.1 through B.10.3 and surrounding text.)

## Implementing the MIPS Architecture

- Goal of Chapter 4 is to show how we could use the low-level building blocks described in Appendix B 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 5**

## 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 6**

Slide 7

## 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: Two kinds of “logic blocks” described in Appendix B.

Slide 8

## Clocking — Executive-Level Summary

- (Discussed in more detail in Appendix B.)
- 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 9

## 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? should become obvious soon).
- "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 10

## 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 . . .

## Memory-Access Instructions

Slide 11

- 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?

## Arithmetic/Logic Instructions

Slide 12

- 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 13

## 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 14

## 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 15

## The “Datapath”

- As discussed in class (and in more detail in section 4.2), we will need instruction memory, data memory, register file, PC, a full ALU, and a couple of adders.
- Did we leave anything out? yes:
  - Input to ALU / adder is two 32-bit quantities, but for some instructions what we have in the instruction is 16 bits — so we need something to extend that to 32 bits by extending the sign.
  - Both control-flow instructions include something that needs to be shifted two bits before being used to compute a target address, so we need to support that.
- Combine with “datapath” part of Figure 4.2 to get Figure 4.11, which leaves out the “control” part, substituting not-connected-yet control inputs (blue in the text).

Slide 16

## 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.
- As mentioned in Appendix B, tools exist to transform truth tables into combinational logic, so our job is to come up with ones that will generate the signals we need for the datapath.
- Section 4.4 works through details. A lot of it should seem like common sense (viewed from the right angle?).

## 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?

**Slide 17**

## Minute Essay Answer

- This is one of the textbook's “check yourself” questions (p. 259), and the answer is at the end of the chapter.

**Slide 18**