Slide 1

Administrivia

- Reminder: Homework 4 due today.
- Reminder: Exam 1 next Wednesday. Review sheet on the Web. We will do more review Monday.
- Homework 1 graded. I'm returning them in class, with a sample solution. (If you didn't turn anything in, ask for a sample solution.)

  Most people (though not all) did well.

  (And — the problem where I said you might consider writing a little throw-away program? only eight people did, but a range of languages — C++, C, Python, Scala, and one person mentioned a calculator).

Slide 2

Minute Essay From Last Lecture

- (Some interesting answers. I'll say more in next few slides and also try to respond to individual messages, soon.)

Slide 3

O/S Versus Object File Contents/Formats

- The machine code part should depend only on the architecture (and 32-bit versus 64-bits counts as part of "the architecture"). But the same compiler running under different operating systems might make different choices?
- Format seems like it would be similar but not identical across operating systems, but if there's no real incentive to standardize maybe it hasn't happened.

Slide 4

O/S Versus Executable File Contents/Formats

- Part of what's in an executable file is whatever information is needed for the O/S to "launch" the program. Windows `.exe` files don't run under Linux, right? and ELF executables don't run under Windows? (What about WINE? well, it's an emulator, isn't it?)
- So for example consider references to shared library code — Windows DLLs versus UNIX "shared libraries" versus . . .).
- Also might matter whether the linker can assume that programs will always be loaded starting at the same address.

Slide 5

O/S and System Calls

- At the hardware / machine code level, what matters is the architecture — switch to "all privileges" mode (if such a thing exists), transfer control to fixed location, noting return address.
- But then what's at that fixed location is supposed to be O/S code, and what it does clearly could vary. For example, for SPIM, to echo a line to standard output you set particular values in some registers and do the `syscall` instruction. Other O/S's for MIPS would provide this functionality in other ways (probably also involving a `syscall` but with different conventions for register usage, e.g.).

  Many of these services would be fairly high-level — e.g., "opening" a "file" — and would need to be supported by a fair bit of O/S code!

Slide 6

Designing a Processor — Overview

- Goal of Chapter 4 — sketch design of a (hardware) implementation of MIPS architecture in terms of some simple building blocks (AND and OR gates, inverters).
- Key components of the design (Figures 4.1 and 4.2):
  - Something to implement memory.
  - Something to implement instructions: "ALU" (arithmetic/logic unit).
  - Something to implement registers: "register file".
  - Something to implement fetch/decode/execute cycle: "control logic".

  The first three together make up the "data path". Analogy — it's a puppet, with "control" pulling its strings.

Slide 7

Circuit Design — Overview

- AND and OR gates implement Boolean-algebra functions of the same names; inverter implements "not".
- A word about notation: We'll use the textbook's notation for Boolean algebra, which alas is (probably?) different from what you used in CSCI 1323.

CSCI 2321	*CSCI 1323*
$a \cdot b$	$a \wedge b$
$a + b$	$a \vee b$
$\overline{a}$	a'

Slide 8

Implementing Logic Gates — Executive-Level Summary

- The ones and zeros of low-level software become two distinct voltages in hardware, and the logic of Boolean algebra is implemented using "switches" (things that connect an input to an output, or not, depending on the state of a control input).
- Currently these switches are (usually?) transistors. In widely-used "CMOS technology", there are two types of switches, one that's good if the input is "one" and one that's good if the input is "zero". These can be combined to implement logic. We looked earlier in the semester at a simple example (inverter). Link to description/explanation on "useful links" page coming soon.

Slide 9

Circuit Design — Overview Continued

- "Combinational logic" blocks implement Boolean functions/operations — map input(s) to output(s) without a notion of persistent state. (Think of these as "pure" functions that don't change any variables but can have multiple outputs.)
- "Sequential logic" blocks also implement Boolean functions/operations but include a notion of persistent state. (Think of these as methods in object-oriented programming, which map input(s) to output(s) but also have access to member variables that can be read/written.)

Slide 10

Minute Essay

- None – quiz.