Slide 1

Administrivia

- Quiz 2 a week from today.
- Homework 2 on the Web; due March 3 (since we have not covered all material yet). Many problems, so start early?
- Reminder: Midterm scheduled for March 5 (the following Thursday). I'm willing to consider rescheduling for after spring break if there's consensus among students.

Slide 2

Minute Essay From Last Lecture

- Answer in notes. Most people got it, but not all. Review?

Slide 3

A Little (More) About Assembly Language and Assemblers

- We've done a few short examples of translating assembly language into machine language, punting on labels. (But now that we know about addressing modes, we can fill in details — next.)
- Normally this is done programmatically, by an "assembler". Accepts symbolic representations of instructions. Also uses some directives to help keep track of instructions, define character strings, etc. Details for MIPS assembler in Appendix A.

Slide 4

Translating Instructions With Labels — Example

- Look at an example — machine language for this C:

```
while (a[i] == k) {
        i = i + j;
}
```

Assume we're using `$s3` through `$s6` for `i`, `j`, `k`, address of `a`, and that code is in memory at (decimal) location 80000.

What does the machine code look like? first a digression . . .

Slide 5

Writing Complete Programs for the Simulator

- The simulator includes what's in essence a very primitive operating system, with facilities to load programs and do simple I/O. As in real operating systems, I/O is done by making "system calls".
- Complete programs can be run from the command line with, e.g., `spim -file hello.s`.

Slide 6

System Calls

- System calls are how user programs request service from the operating system — not just in MIPS, but in general. In MIPS the instruction is `syscall`; other architectures have something analogous.
- System calls similar to procedure calls in some ways — need to communicate to o/s which service you want (e.g., write some text to "standard output") and possibly parameters (e.g., the text to write). As with procedure calls, we do this by putting values in particular registers, but then rather than `jal` we use `syscall`.

Slide 7

System Calls, Continued

- An important distinction (discussed more in o/s courses, such as our CSCI 3323): Code for “system call” executes as part of the o/s, which means not subject to same restrictions as user programs (e.g., on memory access).
- Details (e.g., what services are offered) depend on the o/s. The very primitive o/s included in `spim` supports some for simple I/O; details in Appendix A.

Slide 8

Complete Programs — Examples

- We can now write some simple but complete programs for the simulator.
- (Examples on “sample programs” page.)

Slide 9

Example Revisited

- Now we have enough that we can translate that fragment of C into assembler and test the result(!).
- (Start by rewriting to use `goto`, then translate into assembler, then . . .)

Slide 10

Minute Essay

- None really — unless questions about writing programs for SPIM?