Slide 1

Administrivia

- FYI: Quiz solutions will be online after class(es).

Slide 2

Example, Continued

- Finish example (of turning one of those snippets of C code into a complete MIPS assembly program) from previous class.
- Look at binary version of instructions, especially branches and jumps.

Slide 3

Decoding Machine Language

- As a check on whether what we have is sensible — try going from machine language back to assembly language, using same example.
- Notice that there are limits to how reversible this process is. (What?)

Slide 4

Sidebar: Parallel Execution and Synchronization

- A lot of commodity hardware these days features multiple processing units ("cores") sharing access to memory. One reason for this is that in theory we can make individual applications faster by splitting computation up among processing elements.
- Having processing elements share memory makes parallel programming easier in some ways but has risks ("race conditions"). Avoiding the risks requires some way to control access to shared variables (e.g., to implement notion of "lock").

Slide 5

Parallel Execution and Synchronization, Continued

- Most texts on operating systems discuss synchronization issues and present several solutions ("synchronization mechanisms"), some rather high-level and others not.

  (Why is this in o/s textbooks?)

- The most primitive can (with some simplifying assumptions) be implemented with no hardware support. But hardware support is very useful.

Slide 6

Instructions for Synchronization

- Key goal in designing hardware support for synchronization is to provide "atomic" (indivisible) load-and-store. This allows writing a low-level implementation of "lock" idea.
- Many architectures do this with a single instruction (e.g., "test and set" or "compare and swap"). Requires two accesses to memory so may be difficult to implement efficiently.
- MIPS approach — same idea, but using a pair of instructions, `ll` ("load linked") and `sc` ("store conditional"). Example of use in textbook (p. 122). `sc` "succeeds" only if value at target location has not changed since previous `ll` — i.e., if one can regard the pair of instructions as forming a single atomic load/store.

Slide 7

Minute Essay

- None — quiz.