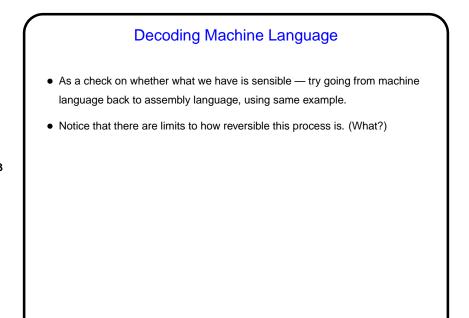


Slide 2



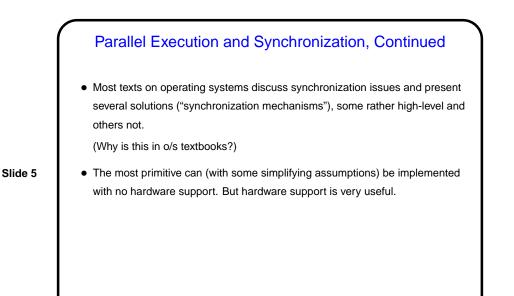
Slide 3

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.

Slide 4

• 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").



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). so

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 11 — i.e., if one can regard the pair of instructions as forming a single atomic load/store.

Slide 6

