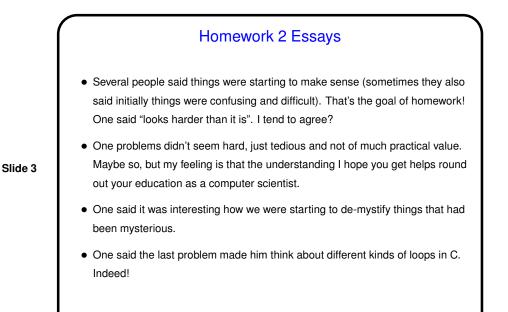


Minute Essay From Last Lecture • Pretty much unanimously people liked doing a problem in class! I'll try to do more of this, at least from time to time.

Slide 2



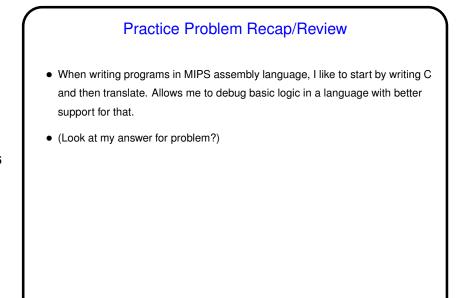
Homework 2 Essays, Continued

- One said he was thankful for HLLs and compilers. Indeed!
- Perhaps the most interesting comment: Examples of compiling C seem long and tedious; hard to imagine writing a whole compiler!

Indeed! but the way I understand it, people don't: If starting from scratch (unlikely these days), typically start by writing a sort of starter compiler, enough to translate at least some of the language, no optimization. Then use it to develop a slightly more complicated compiler ... "Lather, rinse, repeat"?

Slide 4

2

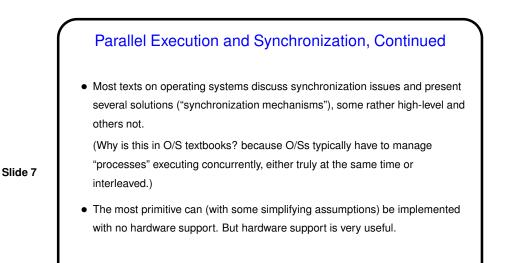


Sidebar(?): Parallel Execution and Synchronization

 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 6

• 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").



Sidebar: Why is Implementing a Lock Hard? It might seem like it would be straightforward to implement a lock — just have

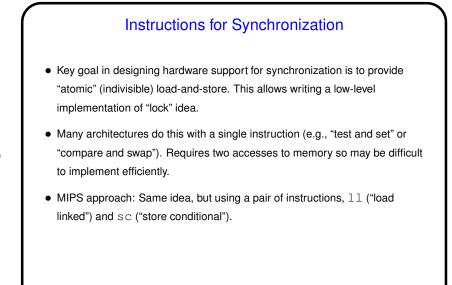
an integer variable, with value 0 meaning "unlocked" and anything else meaning "locked". And then you "lock" by looping until the value is 0, then setting to nonzero:

Slide 8

while (lock != 0) {}

and "unlock" by setting back to 0.But this doesn't work! (Why not?)

lock = 1;



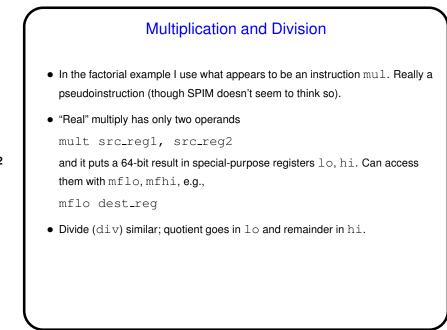
MIPS Instructions for Synchronization
11 loads a value from memory and somehow remembers the location and value. Syntax:

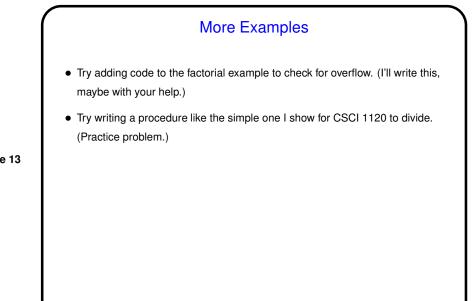
reg1, displacement (reg2)
Operands used as for 1w.

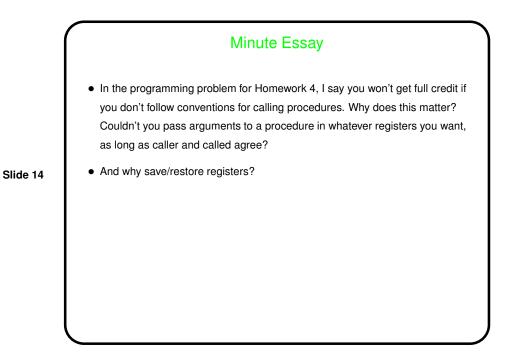
Sc stores a value into memory — *IF* the location has not changed since a previous 11 from that address.
sc reg1, displacement (reg2)
Operands used almost as for 1w, except that reg1 is set to indicate whether the store "succeeded" (i.e., value had not changed since 11). So one can regard a (11, sc) pair as forming a single atomic load/store.
(How to make this work? Hardware designers' problem! glib answer but maybe all we can do in this course.)

Slide 9

5







Minute Essay Answer

• If you write both the calling program and its called procedures, it might seem like it hardly matters how you communicate between caller and called. But think about how well (or not well) this would work for a larger project! and even for small projects, isn't it easier to always follow convention rather than inventing one for each procedure?

Slide 15

• Saving/restoring registers ... You can skip this if the procedure doesn't modify any of the registers normally saved/restored. I say probably good style to do it anyway; better to just copy boilerplate than try to think through exactly what each case needs?