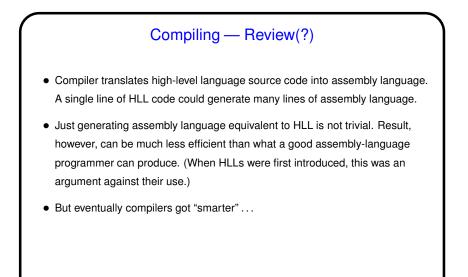


Slide 6



Compiling, Continued
One reason compilers are so big and complicated is that more and more they try to "optimize" (generate code that's more efficient than a naive translation), for example, by keeping values in registers to reduce the number of memory accesses.
Conventional wisdom now is that compilers can generate better assembly-language code than humans, at least most of the time.
Further, many architectures ("RISC", short for Reduced Instruction Set Computing) designed with the idea that most programs will be written in a high-level language, so ease of use for assembly-language programmers not a goal.

Slide 7

Slide 8

Compiling, Continued

- Textbook goes into some detail about compiling C code to loop through an array, showing a version that uses indices and one that uses pointers. They claim that a "good" compiler will likely generate the same code for both.
 Can (try to) test this with gcc write it both ways, compile with -S, and compare. Results with -O0 (no optimization) likely different, but with optimization (gcc defines several levels) should in principle be the same. I did get that result with some previous version of gcc but now don't.
- Standard advice write for clarity, trust compiler to generate good assembly code probably the way to go!

Compiling, Continued

 Note in passing that compiler optimizations can play havoc with attempts to time things: C compilers are allowed to just skip any code that doesn't have an observable effect (i.e., result isn't printed or otherwise used). (In practice they may or may not.)

Slide 10

