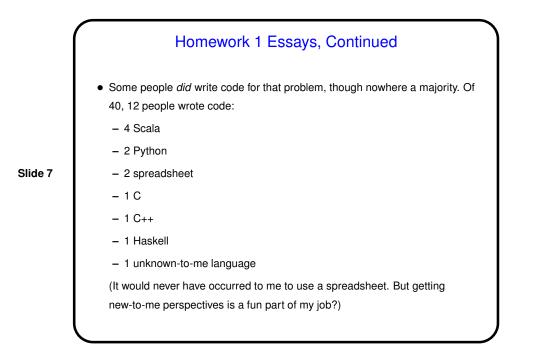


Homework 1 Essays Many people found *something* about the problems interesting, though many also found them somewhat tedious, and one person said "busy-work". "Yeah well"? Several people commented on the problem about fallacies. This is one of the things I like about this discussion — performance not as simple as one number. One person commented that it's interesting that after discussion of how hard it is to quantify performance we then talk about "best performance". Good point! For the problem about computing parallel execution times, several people didn't realize "sequential time" and "parallel time on 1 processor" were distinct things. But they are, and this is not atypical of real programs.

3



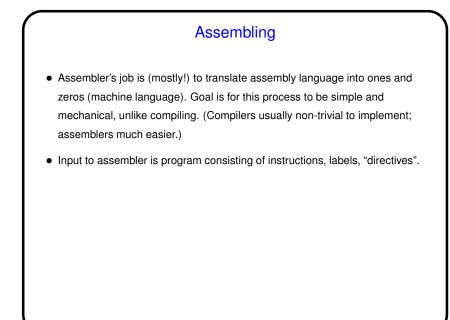


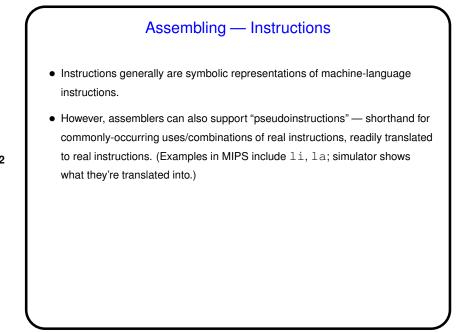
- Four main phases, conceptually at least compile, assemble, link, load.
- Real systems (or simulators) may combine steps, in appearance or even in reality — e.g., a compiler might go directly from high-level source to object code, in appearance or in fact, and the SPIM simulator assembles "on the fly".

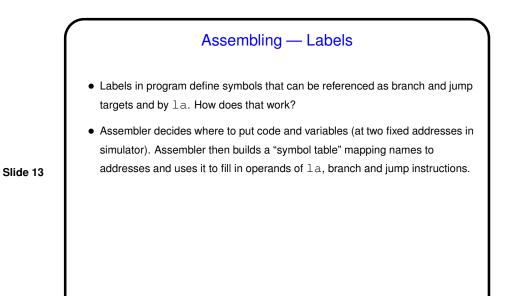


- Compiler translates high-level language source code into assembly language. A single line of HLL code could generate many lines of assembly language.
- Just generating assembly language equivalent to HLL is not trivial. Result, however, can be much less efficient than what a good assembly-language programmer can produce. (When HLLs were first introduced, this was an argument against their use.)
- But eventually compilers got "smarter" ...

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.
Slide 10
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.
Some compilers will show you the assembly-language result (e.g., gcc with the -S flag). (A bit more about this another time.)







Assembler directives (starting . in MIPS) tell the assembler — something. Examples include .word to define a 4-byte constant, .end.
Two worth additional mention here — .text, .data: Typically output includes "text (code) segment" consisting of machine-language instructions and "data segment" containing fixed/static data.
.text, .data tell assembler which of the these to use for following code.

Slide 14

7

Linking

• For small programs assembling the whole program works well enough. But if the program is large, or if it uses library functions, seems wasteful to recompile sections that haven't changed, or to compile library functions every time (not to mention that that requires having their source code).

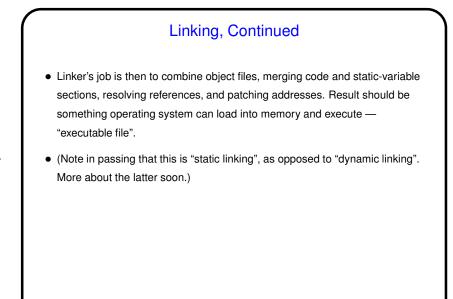
Slide 15

- So we need a way to compile parts of programs separately and then somehow put the pieces back together i.e., a "linker" (a.k.a. "linkage editor").
- To do this, have to define a mechanism whereby programs/procedures can reference addresses outside themselves and can use absolute addresses even though those might change.

Linking, Continued

 How? define format for "object file" — machine language, plus additional information about size of code, size of statically-allocated variables, symbols, and instructions that need to be "patched" to correct addresses. Format is part of complete "ABI" (Application Binary Interface), specific to combination of architecture and operating system.

So, output of assembler is one of these, including information about symbols defined in this code fragment and about unresolved (external) references.



	Loaders
	• So what's left
	 "Executable file" contains all machine language for program, except for any dynamically-linked library procedures. What does the operating system have to do to run the program? Well
18	• Obviously it needs to copy the static parts (code, variables) into memory. (How big are they?) Also it needs to set up to transfer control to the main program, including passing any parameters. And what about those absolute addresses?
	 So as with object code, executable files contain more than just machine language. File format, like that of object code, is part of ABI. (More soon.)

