Slide 1

Administrivia

- (Via e-mail.)

Slide 2

Homework 4 Assemble/Link Problem — Why?

- Textbook's example of linking is not, in my opinion, all that clear. Further — I think it should go further in showing how the whole process works.

  So in Homework 4 — fill in more details, and in a way compatible with SPIM.
- Recycling similar problems from previous years not an option — I was wrong about about what's in the symbol table and relocation information.
- Getting it reasonable straight in *my* head — very long process, many frustrations! I tried to incorporate what I could find out about ELF format for object files, widely used in UNIXworld. But that about "devil is in the details"? So true here.

Slide 3

Homework 4 Assemble/Link Problem — Example

- Problem write-up includes a worked-through example, meant to be helpful.
- Work through steps . . .

Slide 4

Homework 4 Assemble/Link Problem — Example, Continued

- First step — "annotate" source: Expand pseudoinstructions, label each instruction with offset.
- Next build symbol table, relocation information. (I got a bit stuck here, thinking symbols all had to be global. Don't think that can work, though.)
- Finally combine symbol tables, relocation information sections, and "patch" instructions.
- Problem asks you to express things in hexadecimal. If that's hard — example directory index mentions some options.

Slide 5

Homework 4 Assemble/Link Problem — Concerns

- I'm still unsure about details of references to local-only labels:

  Seems like they must be available in some contexts (e.g., jumps to local-only labels inside same module), but not in others (jumps to local-only lables from another module).

  Further, can't require non-global labels to be unique, and — how can that work?

- But clearly possible, and — I'm leaving this as an open question for now.

Slide 6

Chapter 2 Wrap-Up

- "Real stuff" sections look interesting, but — skim if interested.
- One more topic worth saying a few words about . . .

Slide 7

Compiling — Review(?)

- 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" . . .

Slide 8

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 9

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!

Slide 10

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 11

Minute Essay

- Questions about Homework 4 now? though you may not really know until you start working on it.