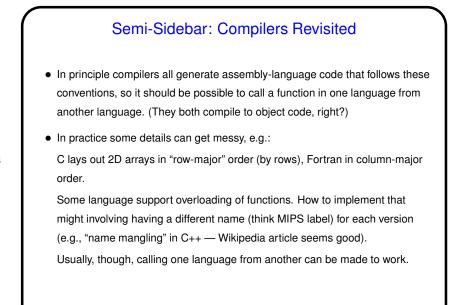


From Source to Execution — Big Picture Revisited

• Goal is to be able to translate programs written in a HLL or assembler language into something that the operating system can load into memory and run.

- Usually want this to be done in a way that supports separate compilation/assembly of source code files, possibly in different languages. (That sort of implies support for library functions, since a "library" basically consists of previously-compiled code.)
- A lot of the software conventions we've looked how procedures are called, memory use, etc. exist to make this work.

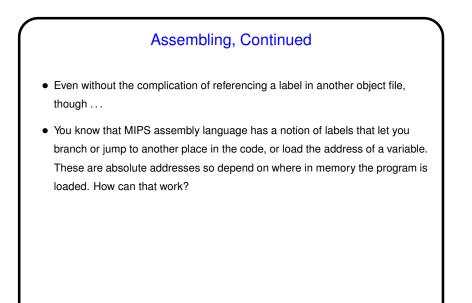


Assembling Revisited Job of the assembler is to produce "object code". Details vary among platforms ("platform" here means combination of architecture and operating system). Keeping in mind the big picture, object code needs to contain: Machine language for instructions, typically collected into "code segment", a.k.a. "text segment". Binary representation of any variables (.word, .asciiz, etc., in MIPS), typically collected into "data segment". Something that will make it possible for code in one object file to reference a global symbol (procedure or data) in another.

Slide 3

Slide 4

2



Assemblers — How They (Could?) Work (I admit I haven't looked at actual code for an assembler, but the job seems straightforward.) First start by establishing starting addresses for code and data segments. As each instruction or data declaration is encountered, add to appropriate segment and increment "next" address (by 4 for instructions, by size of data for data). (*NOTE* that labels themselves occupy no space, but pseudoinstructions might expand to multiple real instructions.) Also build "symbol table" of labels versus addresses and list of references to labels, and make note of any declared as "global".

Assemblers — How They (Could?) Work, Continued

 Resolve references to labels using symbol table and "patch" instructions accordingly. (Or maybe you make one pass through the code that only builds the symbol table and another that actually converts instructions. I think two passes are needed in any case.)

Slide 7

- Also keep track of any uses of absolute addresses, since these depend on where in memory the program gets loaded.
- When done, should have text and data segments, symbol table, and list of instructions that aren't right yet (because they reference external symbols or use absolute addresses).

Output all of that; format is part of platform's ABI.

Linkers — How They (Could?) Work

 Job of linker is combine one or more object files into "executable file" something the operating system can load into memory and execute.
 What does that imply ...

Slide 8

- Instructions that aren't complete yet because they reference procedures or data in another object file need to be corrected.
- Instructions that aren't complete/correct because they use absolute addresses need to be corrected.

Note that absolute addresses could still not be right, if it's not known at link time where in memory the program will be loaded.

Linkers — How They (Could?) Work, Contined

- So linker must do some things:
- Merge code segments, data segments.

• Use it to resolve unresolved references.

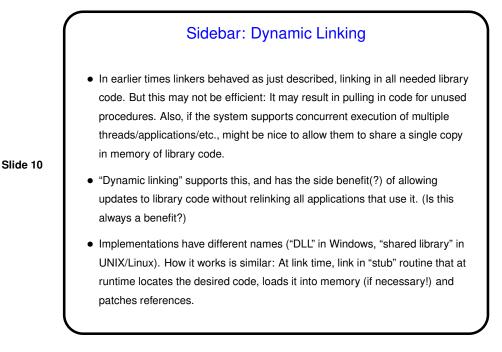
• Merge tables of "global" symbols into combined symbol table.

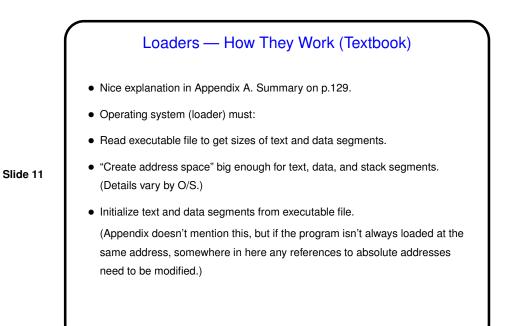
Slide 9

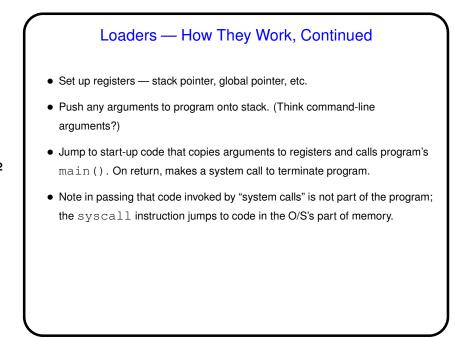
Modify any absolute addresses, keeping track of the instructions that use

them if they will need to be changed when the program is loaded.

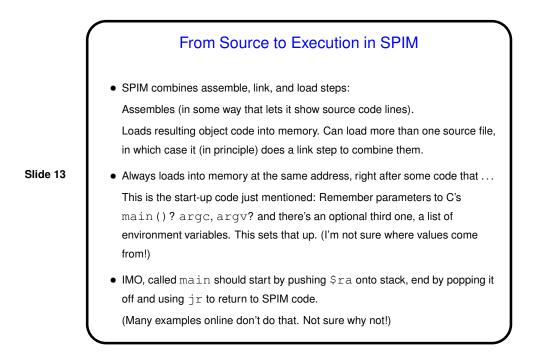
• Output all of that; format is part of platform's ABI.

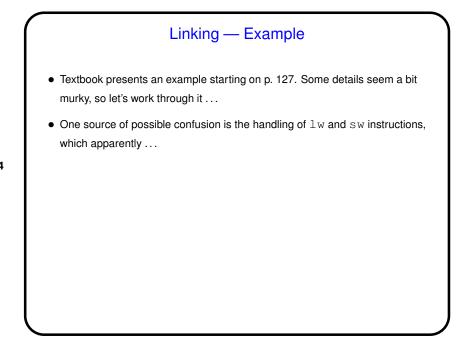


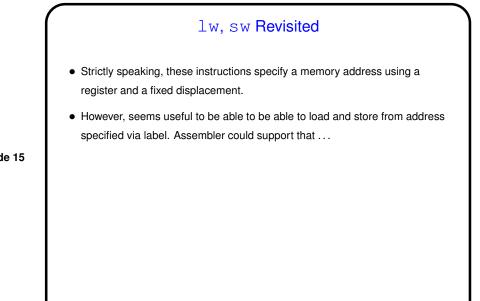


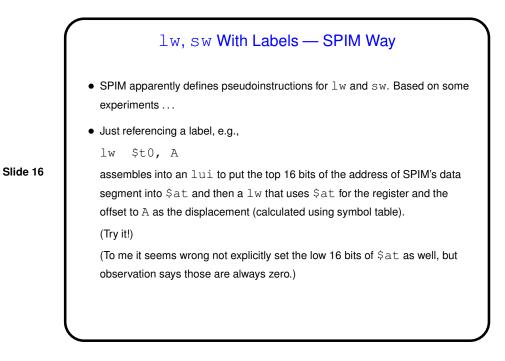


 $\mathbf{6}$



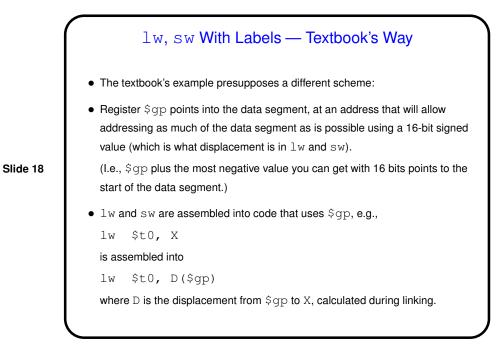


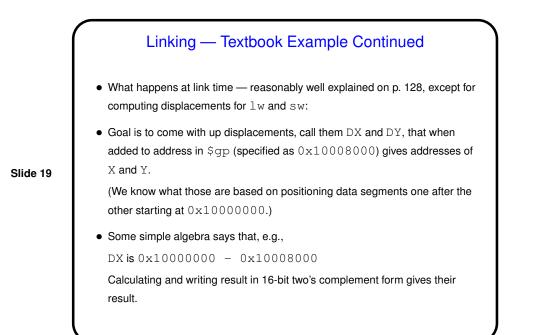




8

lw, sw With Labels — SPIM Way, Continued
Referencing a label and a register, e.g.,
lw \$t0, A(\$t1)
assembles similarly, except that the lui to set \$at to the address of the data segment is followed an addu (unsigned add) to add the contents of \$t1. Note that if \$t1 is an index into an array of "words" this won't do what you might want.





Homework 4 — Example of Assembling / Linking

- For the written problem in Homework 4, I ask you to do something along the same lines, but writing things in a way that I think makes more sense.
- Given two fairly meaningless source-code files, work through steps ...
- (Files with code etc. linked from homework writeup.)

