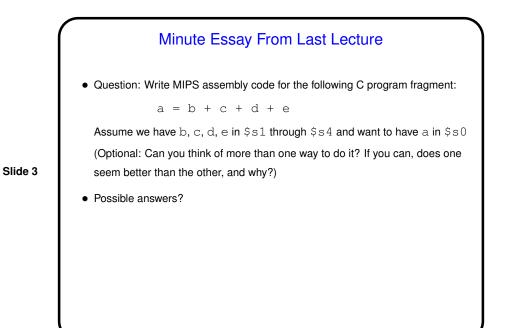


High-Level Languages Versus Assembly Language

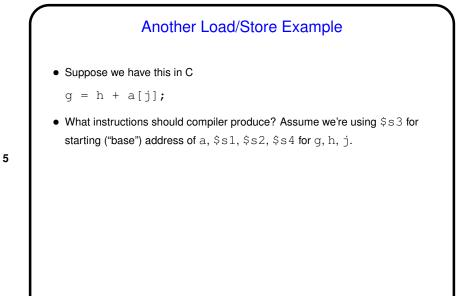
- In a high-level language you work with "variables" conceptually, names for memory locations. You can do arithmetic on them, copy them, etc.
- In machine/assembly language, what you can do may be more restricted e.g., in MIPS architecture, you must load data into a register before doing arithmetic).
- The compiler's job is to translate from the somewhat abstract HLL view to machine language. To do this, normally associate variables with registers load data from memory into registers, calculate, store it back. A "good" compiler tries to minimize loads/stores.



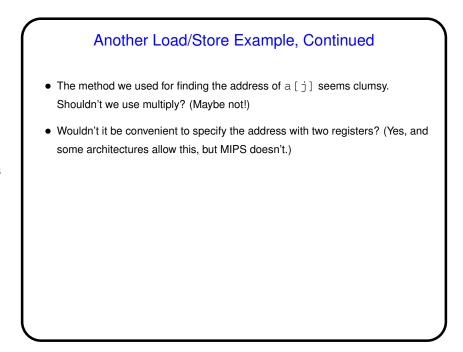
Load/Store Example
Suppose we have this in C

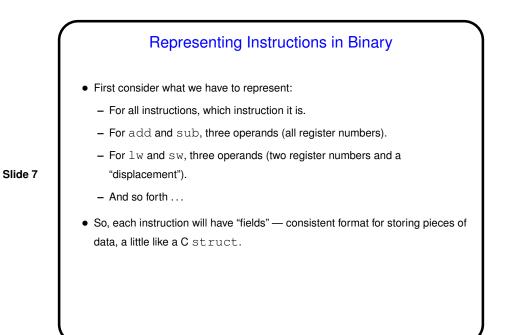
a[12] = h + a[8];

What instructions should compiler produce? Assume we're using \$s3 for starting ("base") address of a, \$s2 for h.



Slide 5





## Representing Instructions in Binary, Continued

- So, can we use the same format for all instructions? Some data ("which instruction") is common to all, but operands may need to be different.
- Can we / should we make all instructions the same length? For MIPS, yes (other architectures differ), and then define different ways of dividing up the length "formats".

"Design Principle 3: Good design involves good compromises."

