



- Many, many factors influence execution time for programs, from choice of algorithm to "processor speed" to system load, as discussed previously.
- Textbook chooses to focus in this chapter on "execution time" by which the authors mean processor time only, excluding delays caused by other factors. Might not be meaningful for comparing systems but seems like reasonable way to compare processors at least.
- (Parallelism in 1/17 slides, starting with "Parallelism (Hardware)".)

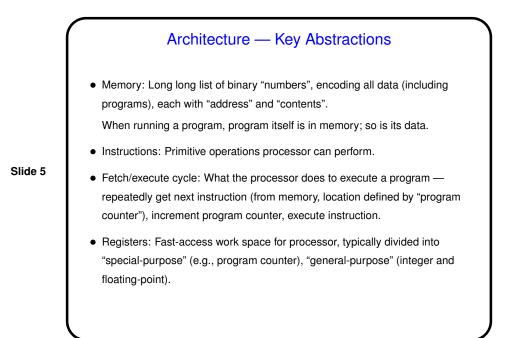
"Architecture" as Interface Definition

- "Architecture" here means "instruction set architecture" (ISA), a key abstraction.
- From software perspective, "architecture" defines lowest-level building blocks

 what operations are possible, what kinds of operands, binary data formats, etc.

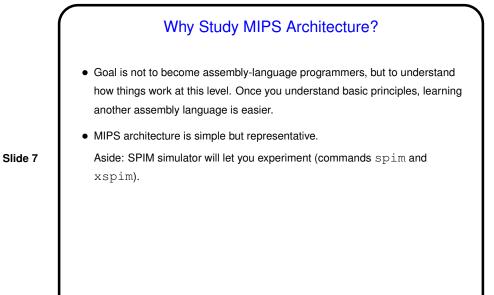
Slide 4

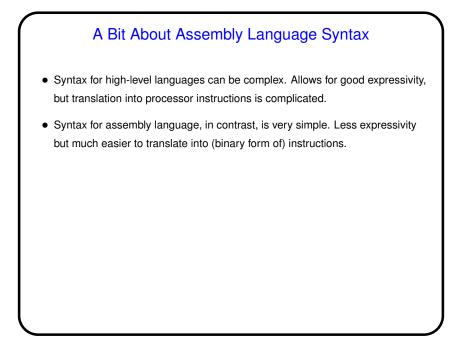
• From hardware perspective, "architecture" is a specification — designers must build something that behaves the way the specification says.

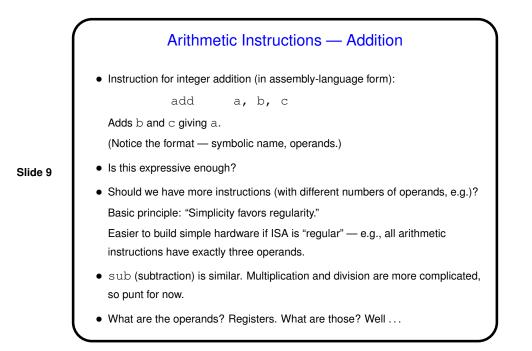


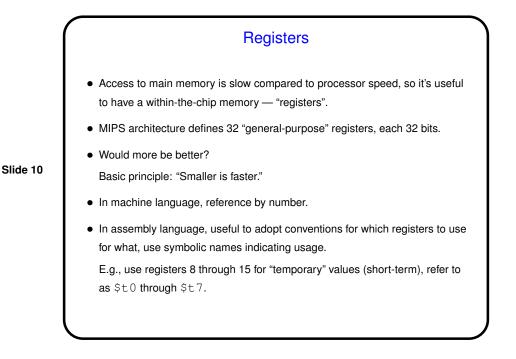
Design Goals for Instruction Set

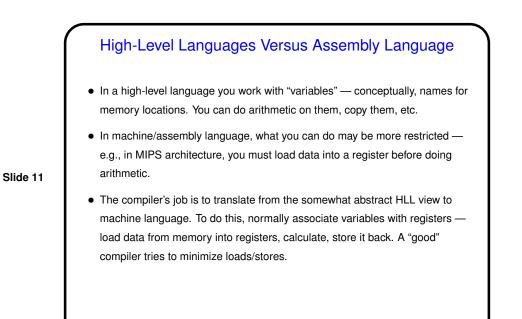
- From software perspective expressivity.
- From hardware perspective good performance, low cost.
- (Yes, these can sometimes be opposing forces!)



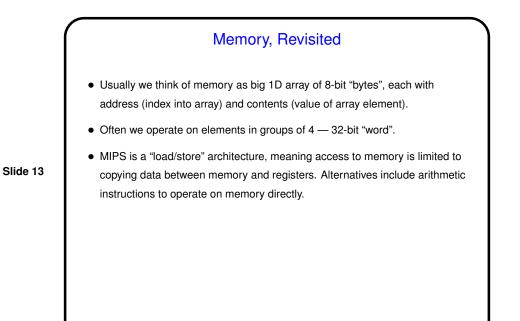




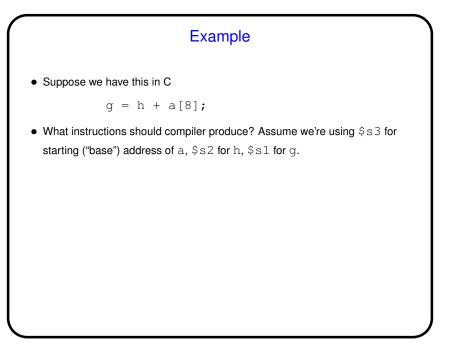


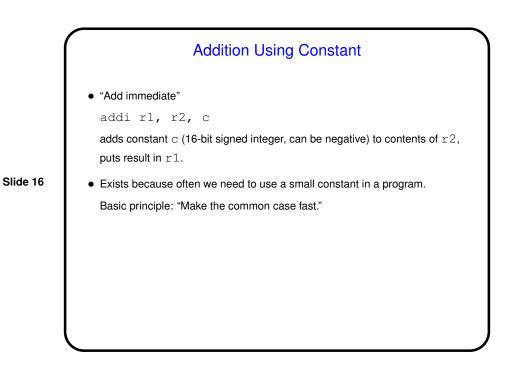


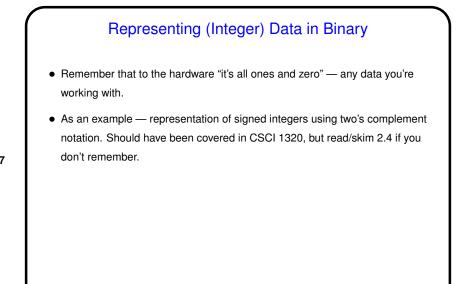
Example
• Suppose we have this in C f = (g + h) - (i + j)
 What instructions should compiler produce? Assume we're using \$s0 for f, \$s1 for g, \$s2 for h, \$s3 for i, \$s4 for j. (Symbolic register names starting \$s are used for for slightly longer-term storage than the ones starting \$t.) (Where do values come from? Next topic)



Memory-Access Instructions — Load
Goal is to get one 32-bit word from memory and put in a register.
How to specify location in memory? Seems most useful to have address in a register. For a little more flexibility, specify address in terms of "base" and "displacement".
lw r, d(b)
Address specified by contents of register b plus (constant) d. Loads word into register r.
sw ("store word") instruction is similar.







A Little About the Simulator As mentioned, installed on our machines is a simulator you can use to try your programs. It simulates a MIPS processor running a *very* primitive operating system (just enough to load programs and do some simple console I/O). It assembles programs on the fly. Slide 18 Your code goes in a file with extension . s. (Sample starter code on "Sample programs" page. Contains many things we haven't talked about yet but could still be useful for trying things out.) Start it with command xspim (spim for command-line version). (Short demo.)

Minute Essay
Write MIPS assembly code for the following C program fragment:

a = b + c + d + e

Assume we have b, c, d, e in \$s1 through \$s4 and want to have a in \$s0
Optional: Can you think of more than one way to do it? If you can, does one seem better than the other, and why?
OR
Write MIPS assembler code to exchange the values of a [0] and a [1]. Assume register \$s0 contains the address of a (start of the array), and a is an array of integers.
If you haven't filled in Dr. Lewis's survey for next semester's classes, please do so now.

	Minute Essay Answer
	• One way:
	add \$s0, \$s1, \$s2
	add \$s0, \$s0, \$s3
	add \$s0, \$s0, \$s4
	Another way (not as good since uses more registers?):
Slide 20	add \$t0, \$s1, \$s2
	add \$t1, \$s3, \$s4
	add \$s0, \$t0, \$t1
	One way:
	lw \$t0, 0(\$s0)
	lw \$t1, 4(\$s0)
	sw \$t0, 4(\$s0)
	sw \$t1, 0(\$s0)