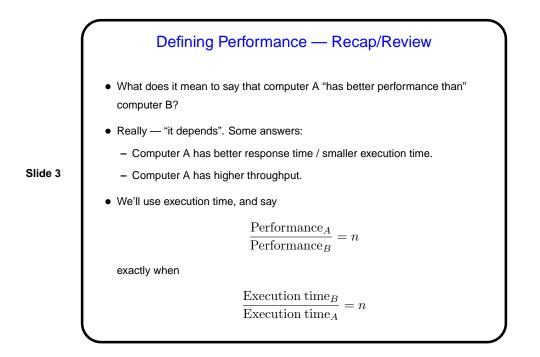
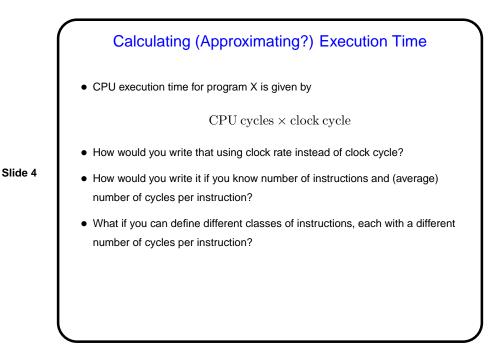
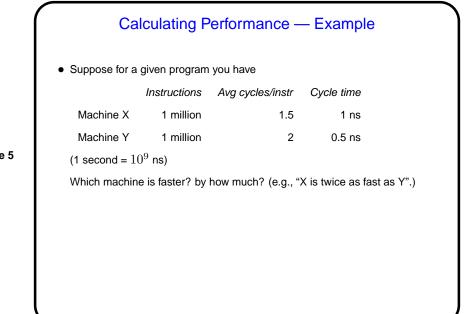
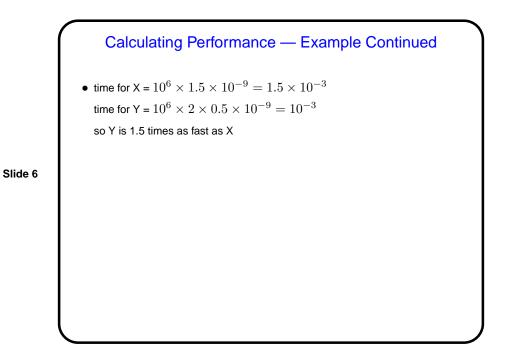


Minute Essay From Last Lecture
What kinds of products probably use embedded processors?
Some answers that seem likely: car, microwave, washer, dryer, radio, calculator(?), digital watch, TV, router, Google glass(?), insulin pump, thermostat, GPS system, printer,
Some answers I'm skeptical about: iPad, tablet, PDA.
Possibly not very clear where to draw the line ...









One More Thing About Performance — Amdahl's Law

- Parallel-computing version: Can define "speedup" gained by using *P* processors as ratio of execution time using 1 processor to execution time using *P* processors. (So, in a perfect world it would be *P*).
- But most "real programs" have some parts that have to be done sequentially. Gene Amdahl (principal architect of early IBM mainframe(s)) argued that this limits speedup — "Amdahl's Law":

If γ is the "serial fraction", speedup on P processors is (at best — this ignores overhead)

$$S(P) = \frac{1}{\gamma + \frac{1 - \gamma}{P}}$$

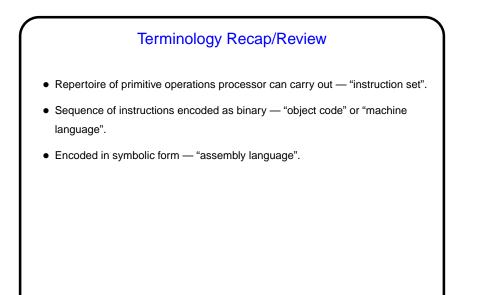
and as P increase, this approaches $\frac{1}{\gamma}$ — upper bound on speedup.

• Textbook points out that this is more broadly applicable!

"Architecture" as Interface Definition

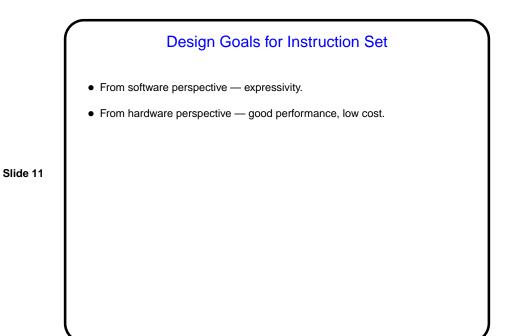
- From software perspective, "architecture" defines lowest-level building blocks

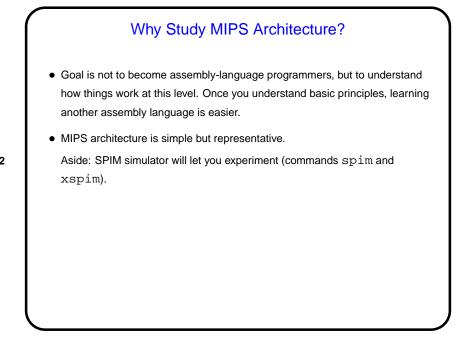
 what operations are possible, what kinds of operands, binary data formats, etc.
- From hardware perspective, "architecture" is a specification designers must build something that behaves the way the specification says.

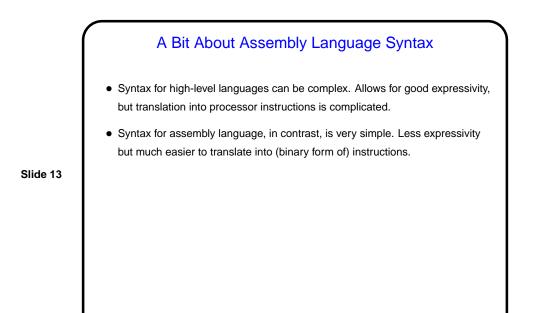


Slide 10

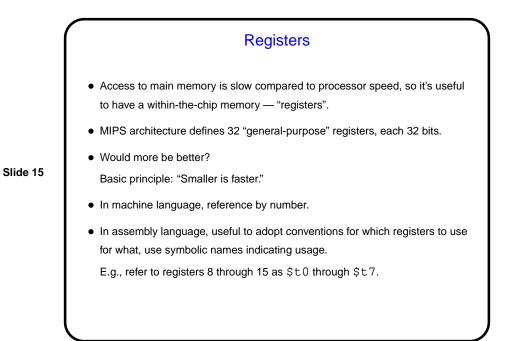
Architecture — Key Abstractions Memory: Long long list of binary "numbers", encoding all data (including programs), each with "address" and "contents". When running a program, program itself is in memory; so is its data. Instructions: Primitive operations processor can perform. Fetch/execute cycle: What the processor does to execute a program — repeatedly get next instruction (from memory, location defined by "program counter"), increment program counter, execute instruction. Registers: Fast-access work space for processor, typically divided into "special-purpose" (e.g., program counter), "general-purpose" (integer and floating-point).

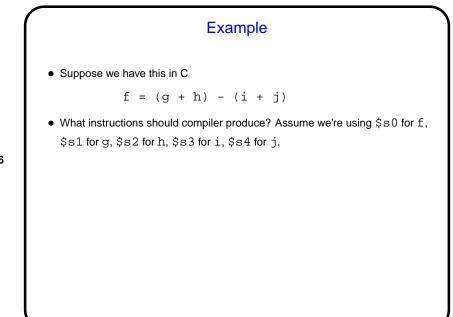


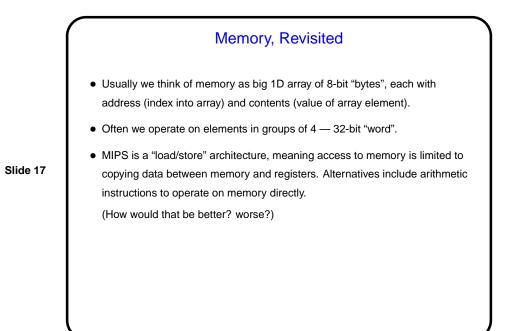


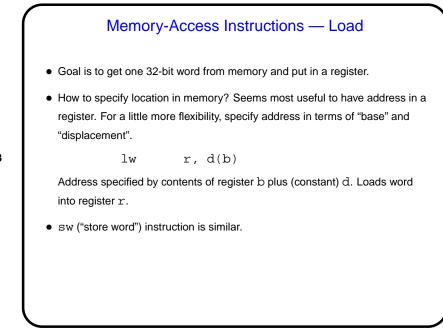


Arithmetic Instructions — Addition
Instruction for integer addition (in assembly-language form): add a, b, c
Adds b and c giving a. (Notice the format — symbolic name, operands.)
Is this expressive enough?
Should we have more instructions (with different numbers of operands, e.g.)? Basic principle: "Simplicity favors regularity." Easier to build simple hardware if ISA is "regular" — e.g., all arithmetic instructions have exactly three operands.
sub (subtraction) is similar. Multiplication and division are more complicated, so punt for now.
What are the operands? Registers.









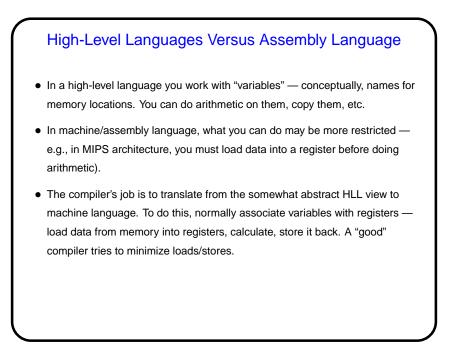


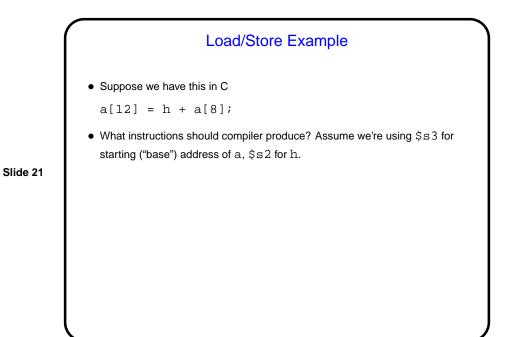
• Suppose we have this in C

g = h + a[8];

• What instructions should compiler produce? Assume we're using \$s3 for starting ("base") address of a, \$s2 for h, \$s1 for g.

Slide 19





 Addition Using Constant
 "Add immediate" addi r1, r2, c adds constant c (16-bit signed integer, can be negative) to contents of r2, puts result in r1.
 Exists because often we need to use a small constant in a program. Basic principle: "Make the common case fast."

