Slide 1

Administrivia

- (A little about me.)
- First homework to be on the Web soon; likely due date is a week from Monday. I will send e-mail.

Slide 2

From Programs to Executables — Recap/Review

- Source code is translated into assembly language (symbolic representation of machine language via a compiler, then converted to object code (machine language, plus other information) via an assembler. Note that all compilers/assemblers follow some of the same conventions for passing of arguments, etc. — this is part of an ABI (“application binary interface”). Another part of the ABI defines how application programs make requests of the operating system.
- Object code is linked with library code via a so-called linker, making use of that “other information” (such as references to library code) to form an “executable” file, which conforms to the part of the ABI that specifies a format specific to architecture and operating system. Typically this file contains machine language plus extra information such as size.

Slide 3

From Programs to Executables, Continued

- At runtime, operating system loads machine language from executable file and transfers control to address of starting instruction. This is also the point at which calls to dynamically-linked library code are resolved.
- (As noted last time, not all languages work this way, but this is the “compile to native code” model used by, e.g., C and C++.)

Slide 4

A Little About Integrated Circuits

- Conceptual view of hardware:
  - *Transistor* — on/off switch controlled by electrical current.
  - Combine/connect a lot of transistors to get *circuit* that does interesting things (e.g., addition).
  - Put a bunch of circuits together to get a *chip* / *integrated circuit* (IC). If lots of transistors, *VLSI chip*.
- (Example of how to use this idea to build a simple circuit to invert one bit linked from course “Useful links” page. In the later part of the course we’ll talk about using inverters and other simple “gates” as building blocks for a processor.)

Slide 5

A Little About Integrated Circuits, Continued

- Manufacturing process starts with a thin flat piece of silicon, adds metal and other stuff to make wires, insulators, transistors, etc.
- Of course, this is all automated! Low-level chip designers use CAD-type tools, which save designs in a standard format, which the chip designers simulate/test with other software, and then send off to be *fabricated*.
- Typically make many chips on a *wafer*, discard those with defects, bond each good one to something larger with *pins* to allow connections to other parts of computer.

Slide 6

Defining Performance

- What does it mean to say that computer A “has better performance than” computer B?
- Really — “it depends”. Some answers:
  - Computer A has better response time / smaller execution time.
  - Computer A has higher throughput.
- We’ll use execution time, and say

$$\frac{\text{Performance}_A}{\text{Performance}_B} = n$$

exactly when

$$\frac{\text{Execution time}_B}{\text{Execution time}_A} = n$$

Slide 7

Measuring Performance

- If we use execution time as criterion, how to measure?
- Wall-clock time seems fairest, since it includes
  - Time for CPU to execute instructions.
  - Any waiting for memory access.
  - Any waiting for I/O.
  - Any waiting for operating system.
- Is that easy to measure reliably / repeatably?

Slide 8

Measuring Performance, Continued

- No — to get repeatable measure of wall clock time, need an otherwise unused system.
- So instead we could use "CPU performance" — amount of time CPU needs to run program. Easier to measure, more consistent, and at least says *something* about the processor.
- Even that, though, is not as simple as it might seem.

Slide 9

Measuring Performance, Continued

- CPU execution time for program X is given by

  $$\text{CPU cycles} \times \text{clock cycle time}$$

  and then CPU cycles in turn is the product of count of instructions and cycles per instruction.

  (“Cycles”? processors typically are mostly-synchronous devices, in which all the parts do some basic operation at fixed intervals called cycles.)

- And then it *might* seem like we can say something meaningful about what happens if we change one of these numbers — but only if all other things remain the same, which might or might be true!

Slide 10

Evaluating / Comparing Performance

- Trickier than it might seem to come up with one number that means something.
- Approaches include
  - Use the actual workload, on the actual hardware platform(s), and compare times.
  - Put together a representative simulated workload — “benchmark”; run and compare times.
  - Compare code size.
  - Compare number of instructions per second (“MIPS” or “MFLOPS”, once).
- Alas, all of these are flawed in some way.

  (In particular, paraphrasing someone whose name I don’t remember, “peak MIPS is just the number you can’t go any faster than.”)

Slide 11

Minute Essay

- Suppose you are trying to decide which of two computers, call them `Foo` and `Bar`, will give you the best performance. You run two test programs on `Foo` and observe execution times of 10 seconds for one and 20 seconds for the other. If the first program takes 5 seconds on `Bar`, how long does the second program take? (Hint: This might be something of a trick question.)
- Other questions?

Slide 12

Minute Essay Answer

- It might seem like that second program would take 10 seconds on `Bar`, but in truth you probably can't be sure without doing the experiment, since the two machines, or the two test programs, could differ in ways that would make this obvious answer wrong.