So, to improve performance (everything else being equal) you can either (increase or decrease?)

_______ the # of required cycles for a program, or
_______ the clock cycle time or, said another way,
_______ the clock rate.

How to Improve Performance

• Could assume that number of cycles equals number of instructions

This assumption is incorrect,
different instructions take different amounts of time on different machines.
Why? hint: remember that these are machine instructions, not lines of C code

Example

• Our favorite program runs in 10 seconds on computer A, which has a 4 GHz. clock. We are trying to help a computer designer build a new machine B, that will run this program in 6 seconds. The designer can use new (or perhaps more expensive) technology to substantially increase the clock rate, but has informed us that this increase will affect the rest of the CPU design, causing machine B to require 1.2 times as many clock cycles as machine A for the same program. What clock rate should we tell the designer to target?”

• Don’t Panic, can easily work this out from basic principles
Now that we understand cycles

• A given program will require
  – some number of instructions (machine instructions)
  – some number of cycles
  – some number of seconds
• We have a vocabulary that relates these quantities:
  – cycle time (seconds per cycle)
  – clock rate (cycles per second)
  – CPI (cycles per instruction)
    - a floating point intensive application might have a higher CPI
  – MIPS (millions of instructions per second)
    - this would be higher for a program using simple instructions

Performance

• Performance is determined by execution time
• Do any of the other variables equal performance?
  – # of cycles to execute program?
  – # of instructions in program?
  – # of cycles per second?
  – average # of cycles per instruction?
  – average # of instructions per second?
• Common pitfall: thinking one of the variables is indicative of performance when it really isn’t.

CPI Example

• Suppose we have two implementations of the same instruction set architecture (ISA).
  For some program,
  Machine A has a clock cycle time of 250 ps and a CPI of 2.0
  Machine B has a clock cycle time of 500 ps and a CPI of 1.2
  What machine is faster for this program, and by how much?

  • If two machines have the same ISA which of our quantities (e.g., clock rate, CPI, execution time, # of instructions, MIPS) will always be identical?

# of Instructions Example

• A compiler designer is trying to decide between two code sequences for a particular machine. Based on the hardware implementation, there are three different classes of instructions: Class A, Class B, and Class C, and they require one, two, and three cycles (respectively).
  The first code sequence has 5 instructions: 2 of A, 1 of B, and 2 of C
  The second sequence has 6 instructions: 4 of A, 1 of B, and 1 of C.
  Which sequence will be faster? How much?
  What is the CPI for each sequence?
MIPS example

- Two different compilers are being tested for a 4 GHz machine with three different classes of instructions: Class A, Class B, and Class C, which require one, two, and three cycles (respectively). Both compilers are used to produce code for a large piece of software.

The first compiler’s code uses 5 million Class A instructions, 1 million Class B instructions, and 1 million Class C instructions.

The second compiler’s code uses 10 million Class A instructions, 1 million Class B instructions, and 1 million Class C instructions.

- Which sequence will be faster according to MIPS?
- Which sequence will be faster according to execution time?

Benchmarks

- Performance best determined by running a real application
  - Use programs typical of expected workload
  - Or, typical of expected class of applications e.g., compilers/editors, scientific applications, graphics, etc.
- Small benchmarks
  - nice for architects and designers
  - easy to standardize
  - can be abused
- SPEC (System Performance Evaluation Cooperative)
  - companies have agreed on a set of real program and inputs
  - valuable indicator of performance (and compiler technology)
  - can still be abused

Benchmark Games

- An embarrassed Intel Corp. acknowledged Friday that a bug in a software program known as a compiler had led the company to overstate the speed of its microprocessor chips on an industry benchmark by 10 percent. However, industry analysts said the coding error... was a sad commentary on a common industry practice of “cheating” on standardized performance tests... The error was pointed out to Intel two days ago by a competitor, Motorola... came in a test known as SPECint92... Intel acknowledged that it had “optimized” its compiler to improve its test scores. The company had also said that it did not like the practice but felt compelled to make the optimizations because its competitors were doing the same thing... At the heart of Intel’s problem is the practice of “tuning” compiler programs to recognize certain computing problems in the test and then substituting special handwritten pieces of code...

---

Saturday, January 6, 1996 New York Times

SPEC ‘89

- Compiler “enhancements” and performance

---
Experiment

- Phone a major computer retailer and tell them you are having trouble deciding between two different computers, specifically you are confused about the processors strengths and weaknesses (e.g., Pentium 4 at 2GHz vs. Celeron M at 1.4 GHz)

- What kind of response are you likely to get?

- What kind of response could you give a friend with the same question?

Amdahl’s Law

- **Execution Time After Improvement =** 
  \[
  \text{Execution Time Unaffected} + \left( \frac{\text{Execution Time Affected}}{\text{Amount of Improvement}} \right)
  \]

- **Example:**
  
  "Suppose a program runs in 100 seconds on a machine, with multiply responsible for 80 seconds of this time. How much do we have to improve the speed of multiplication if we want the program to run 4 times faster?"

  How about making it 5 times faster?

- **Principle:** Make the common case fast
Suppose we enhance a machine making all floating-point instructions run five times faster. If the execution time of some benchmark before the floating-point enhancement is 10 seconds, what will the speedup be if half of the 10 seconds is spent executing floating-point instructions?

We are looking for a benchmark to show off the new floating-point unit described above, and want the overall benchmark to show a speedup of 3. One benchmark we are considering runs for 100 seconds with the old floating-point hardware. How much of the execution time would floating-point instructions have to account for in this program in order to yield our desired speedup on this benchmark?