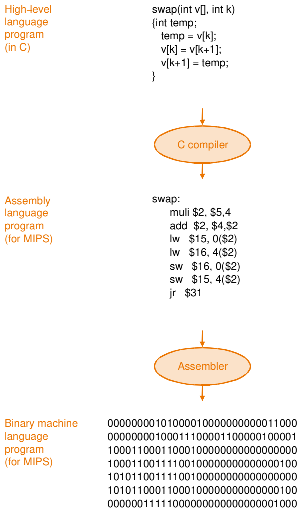
Abstraction

- Delving into the depths reveals more information
- An abstraction omits unneeded detail, helps us cope with complexity

What are some of the details that appear in these familiar abstractions?



Introduction

- Rapidly changing field:
  - vacuum tube –> transistor –> IC –> VLSI (see section 1.4)
  - doubling every 1.5 years:
    memory capacity
    processor speed *(Due to advances in technology <u>and</u> organization)*
- Things you'll be learning:
  - how computers work, a basic foundation
  - how to analyze their performance (or how not to!)
  - issues affecting modern processors (caches, pipelines)
- Why learn this stuff?
  - you want to call yourself a "computer scientist"
  - you want to build software people use (need performance)
  - you need to make a purchasing decision or offer "expert" advice

Instruction Set Architecture

- A very important abstraction
  - interface between hardware and low–level software
  - standardizes instructions, machine language bit patterns, etc.
  - advantage: *different implementations of the same architecture*
  - disadvantage: *sometimes prevents using new innovations*

  True or False: Binary compatibility is extraordinarily important?

- Modern instruction set architectures:
  - 80x86/Pentium/K6, PowerPC, DEC Alpha, MIPS, SPARC, HP

What is a computer?

- Components:
  - input (mouse, keyboard)
  - output (display, printer)
  - memory (disk drives, DRAM, SRAM, CD)
  - network
- Our primary focus: the processor (datapath and control)
  - implemented using millions of transistors
  - Impossible to understand by looking at each transistor
  - We need...

Where we are headed

- Performance issues (Chapter 2)   *vocabulary and motivation*
- A specific instruction set architecture (Chapter 3)
- Arithmetic and how to build an ALU (Chapter 4)
- Constructing a processor to execute our instructions (Chapter 5)
- Pipelining to improve performance (Chapter 6)
- Memory: caches and virtual memory (Chapter 7)
- I/O (Chapter 8)

Performance

- Measure, Report, and Summarize
- Make intelligent choices
- See through the marketing hype
- Key to understanding underlying organizational motivation

Why is some hardware better than others for different programs?

What factors of system performance are hardware related?
(e.g., Do we need a new machine, or a new operating system?)

How does the machine's instruction set affect performance?

Which of these airplanes has the best performance?

Airplane	Passengers	Range (mi)	Speed (mph)
Boeing 737−100	101	630	598
Boeing 747	470	4150	610
BAC/Sud Concorde	132	4000	1350
Douglas DC−8−50	146	8720	544

- How much faster is the Concorde compared to the 747?
- How much bigger is the 747 than the Douglas DC−8?

Computer Performance: TIME, TIME, TIME

- Response Time (latency)
  —How long does it take for my job to run?
  —How long does it take to execute a job?
  —How long must I wait for the database query?
- Throughput
  —How many jobs can the machine run at once?
  —What is the average execution rate?
  —How much work is getting done?

- *If we upgrade a machine with a new processor what do we increase?*

If we add a new machine to the lab what do we increase?

Execution Time

- **Elapsed Time**
  - **counts everything *(disk and memory accesses, I/O , etc.)***
  - **a useful number, but often not good for comparison purposes**
- **CPU time**
  - **doesn't count I/O or time spent running other programs**
  - **can be broken up into system time, and user time**

- **Our focus: user CPU time**
  - **time spent executing the lines of code that are "in" our program**

Definition of Performance

- **For some program running on machine X,**

  Performance$_X$ = 1 / Execution time$_X$

- **"X is n times faster than Y"**

  Performance$_X$ / Performance$_Y$ = n

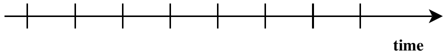
- **Problem:**
  - **machine A runs a program in 20 seconds**
  - **machine B runs the same program in 25 seconds**

Clock Cycles

- **Instead of reporting execution time in seconds, we often use cycles**

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}$$

- **Clock "ticks" indicate when to start activities (one abstraction):**



- **cycle time = time between ticks = seconds per cycle**
- **clock rate (frequency) = cycles per second (1 Hz. = 1 cycle/sec)**

A 200 Mhz. clock has a $\frac{1}{200 \times 10^6} \times 10^9 = 5 \text{ nanoseconds}$ **cycle time**

How to Improve Performance

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}$$

So, to improve performance (everything else being equal) you can either

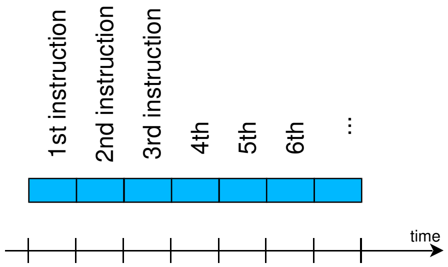
________ the # of required cycles for a program, or

________ the clock cycle time or, said another way,

________ the clock rate.

How many cycles are required for a program?

- Could assume that # of cycles = # of instructions

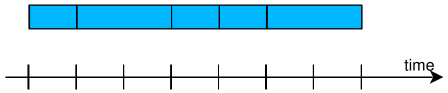


This assumption is incorrect,

different instructions take different amounts of time on different machines.

Why? hint: remember that these are machine instructions, not lines of C code

Different numbers of cycles for different instructions



- Multiplication takes more time than addition
- Floating point operations take longer than integer ones
- Accessing memory takes more time than accessing registers

- *Important point: changing the cycle time often changes the number of cycles required for various instructions (more later)*

Example

- Our favorite program runs in 10 seconds on computer A, which has a 400 Mhz. 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?"

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 vocubulary 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 10 ns. and a CPI of 2.0
  Machine B has a clock cycle time of 20 ns. 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 100 MHz. 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?

SPEC '95

Benchmark	Description
go	Artificial intelligence; plays the game of Go
m88ksim	Motorola 88k chip simulator; runs test program
gcc	The Gnu C compiler generating SPARC code
compress	Compresses and decompresses file in memory
li	Lisp interpreter
ijpeg	Graphic compression and decompression
perl	Manipulates strings and prime numbers in the special-purpose programming language Perl
vortex	A database program
tomcatv	A mesh generation program
swim	Shallow water model with 513 x 513 grid
su2cor	quantum physics; Monte Carlo simulation
hydro2d	Astrophysics; Hydrodynamic Naiver Stokes equations
mgrid	Multigrid solver in 3-D potential field
applu	Parabolic/elliptic partial differential equations
trub3d	Simulates isotropic, homogeneous turbulence in a cube
apsi	Solves problems regarding temperature, wind velocity, and distribution of pollutant
fpppp	Quantum chemistry
wave5	Plasma physics; electromagnetic particle simulation

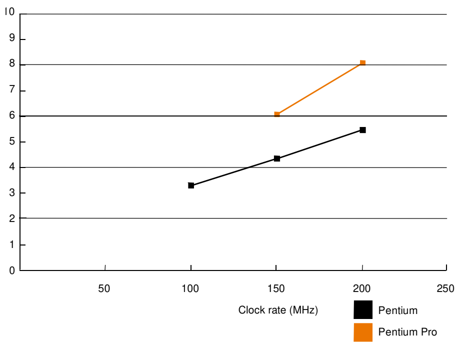
Benchmarks

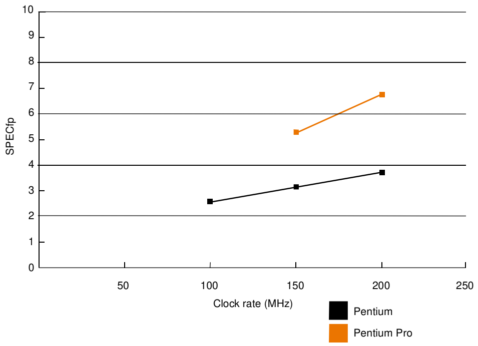
- Performance best determined by running a real application
  - Use programs typical of expected workload
  - Or, typical of expected class of applications
    e.g., compilers/editors, scientific applications, graphics, etc.
- Small benchmarks
  - nice for architects and designers
  - easy to standardize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

SPEC '95

Does doubling the clock rate double the performance?

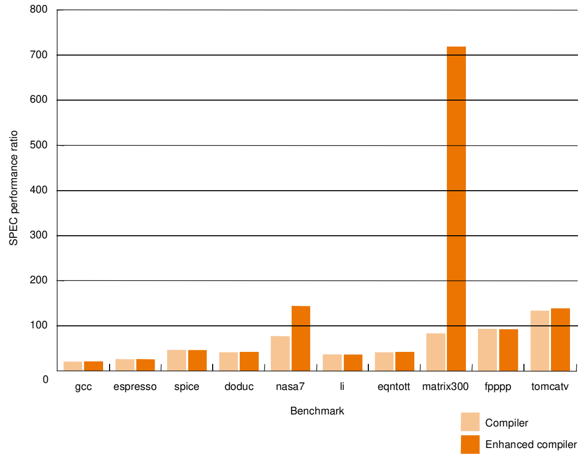
Can a machine with a slower clock rate have better performance?





SPEC '89

- Compiler "enhancements" and performance



Remember

- Performance is specific to a particular program/s
  - Total execution time is a consistent summary of performance
- For a given architecture performance increases come from:
  - increases in clock rate (without adverse CPI affects)
  - improvements in processor organization that lower CPI
  - compiler enhancements that lower CPI and/or instruction count
- Pitfall: expecting improvement in one aspect of a machine's performance to affect the total performance
- You should not always believe everything you read! Read carefully! (see newspaper articles, e.g., Exercise 2.37)

Amdahl's Law

Execution Time After Improvement =

Execution Time Unaffected +(Execution Time Affected / Amount of Improvement)

- **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***

Example

- **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?**