Chapter 1

Performance

Performance Metrics

- Purchasing perspective
  - given a collection of machines, which has the
    - best performance ?
    - least cost ?
    - best cost/performance?
- Design perspective
  - faced with design options, which has the
    - best performance improvement ?
    - least cost ?
    - best cost/performance?
- Both require
  - basis for comparison
  - metric for evaluation
- Our goal is to understand what factors in the architecture contribute to overall system performance and the relative importance (and cost) of these factors

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

- The answer depends on what we mean by “performace”
- How much faster is the Concorde compared to the 747?
  - Performance metric: speed
- How much bigger is the 747 than the Douglas DC-8?
  - Performance metric: capacity

Most important metric for Computer Performance: TIME, TIME, TIME

- Time is the **bottom line**!
- Response Time (latency): we want to reduce this.
  - 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: we want to increase this
  - How many jobs can the machine run at once?
  - What is the average execution rate?
  - How much work is getting done per unit time?

Metric for Computer Performance: TIME, TIME, TIME

- If we upgrade a machine with a new processor what do we increase?
  - Both response time and throughput improved
- If we add a new machine to the lab what do we increase?
  - Throughput. Response time stays the same, because the power of the machine would be the same.

Calculating Performance

- When metric is execution time, performance is an inverse function of execution time:
  - $Performance_X$ = Unit of work / $Execution\ time_X$
  - Book's definition: $Performance_x = 1/time_x$
  - Bigger performance is Better!
    - ➔ smaller execution time is better
- When comparing the performance of two machines, e.g., "X is n times faster than Y"
  - n = $Performance_X$ / $Performance_Y$

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

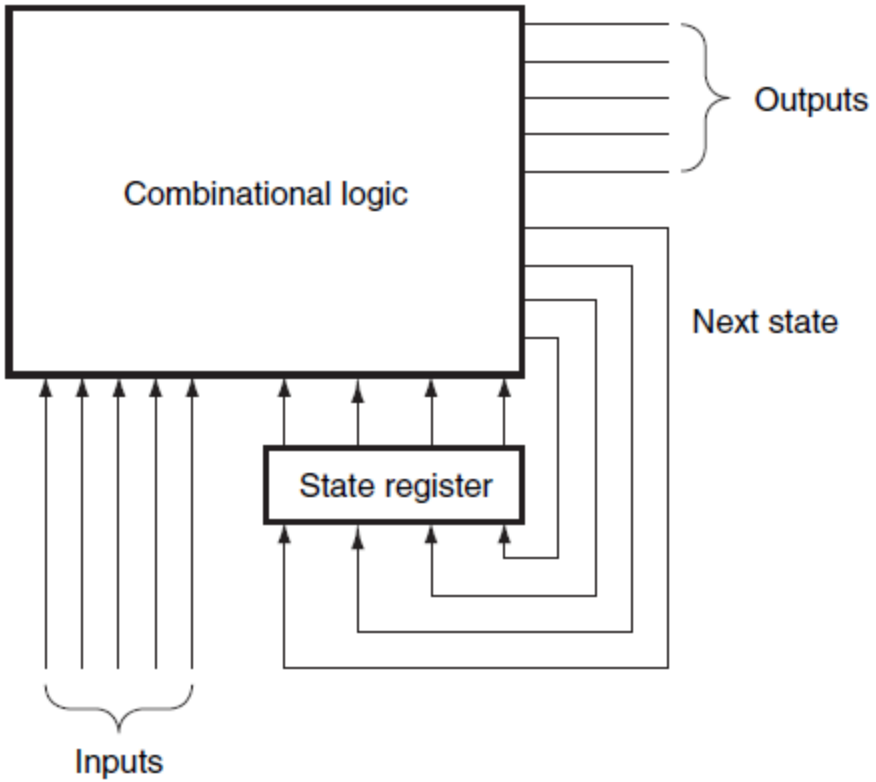
Textbook's Definition of Performance

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

$$\frac{perf_A}{perf_B} = n \Rightarrow \quad \frac{1/exec_A}{1/exec_B} = \frac{exec_B}{exec_A} = \frac{25}{20} = 1.25$$

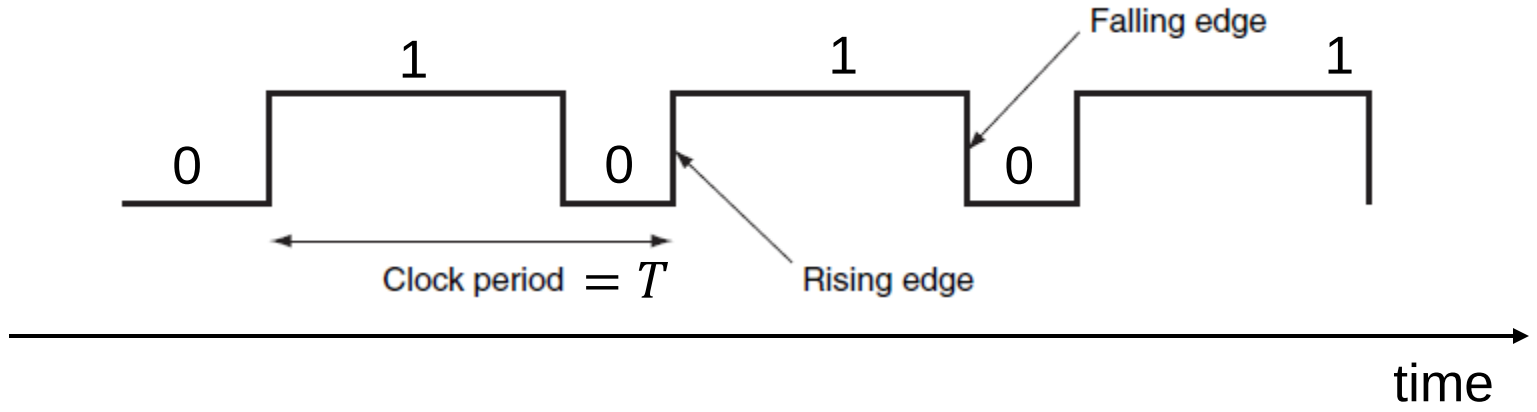
➔ Machine A has 1.25 times better performance.

Sequential circuits



What's a clock?

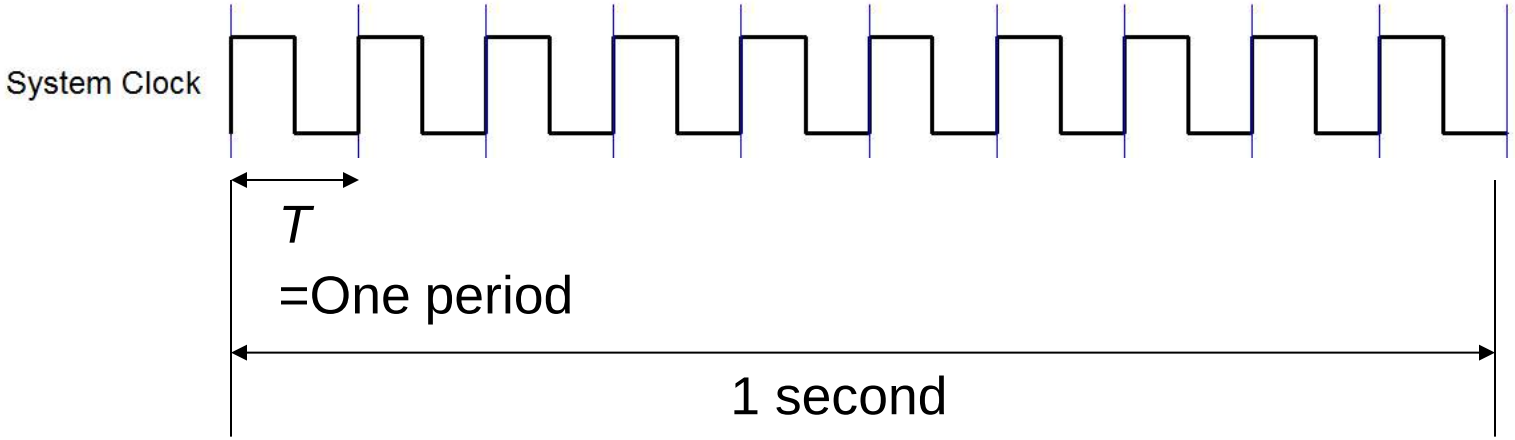
- It is a square wave with a fixed frequency.



T: period
f: frequency

$f = \frac{1}{T}$ and $T = 1/f$

How is clock period related to frequency?



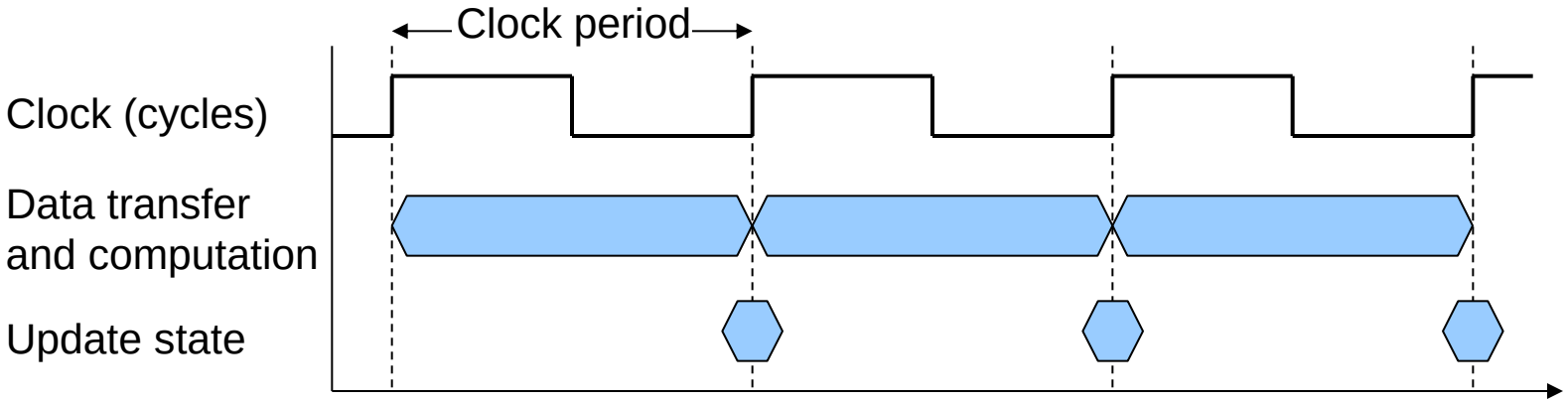
Count the *T*'s in one second.

f = how many *T*'s are there in 1 second = 1/*T*

Example: if *T* = 1 msec, then there are 1000 mseconds (or 1000 *T*'s) in 1 second. *T*=1 msec = 0.001 second; therefore, *f*=1/*T* = 1/0.001 = 1000 Hertz (cycles/second).

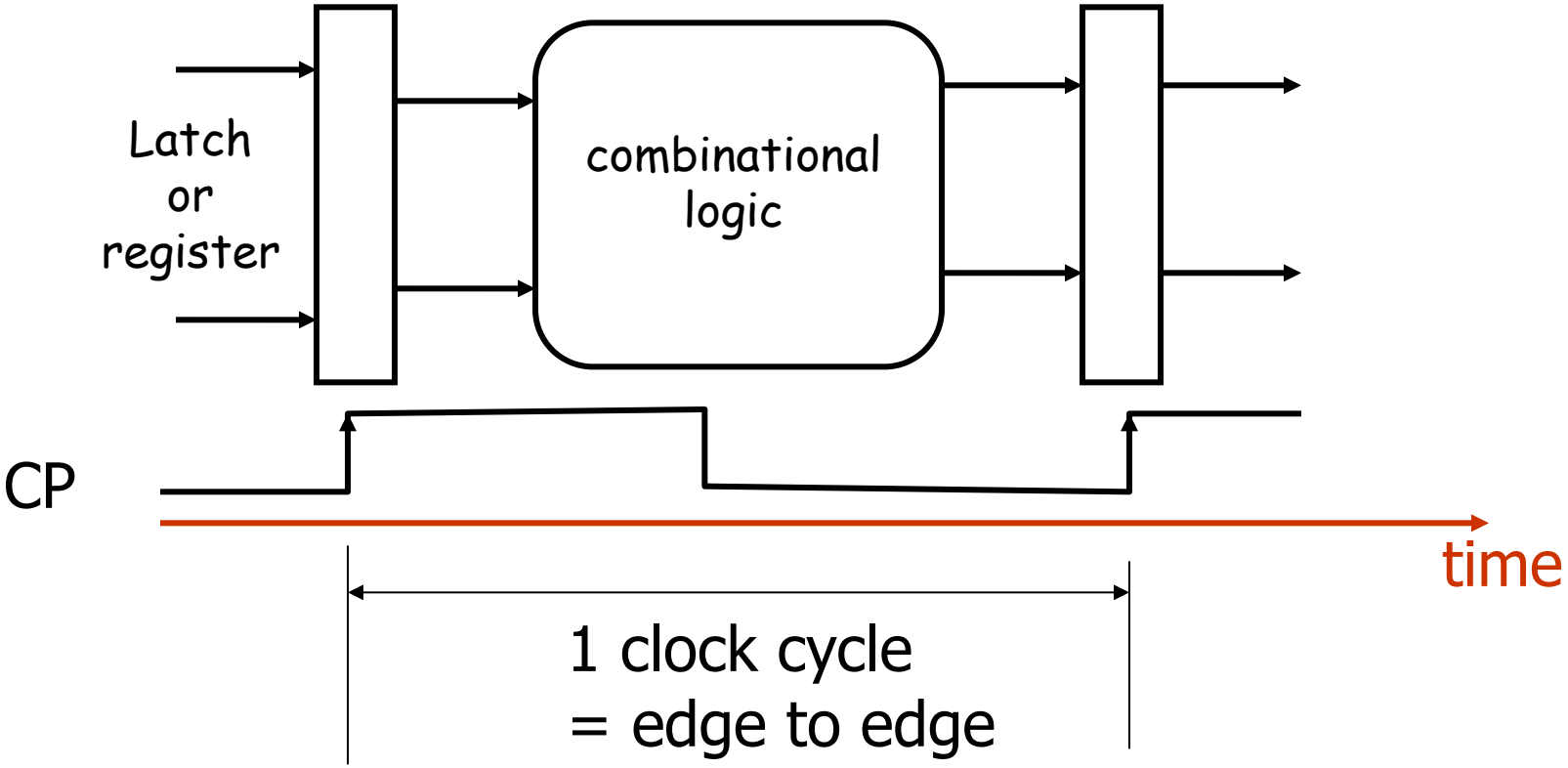
CPU Clocking

- Operation of digital hardware governed by a constant-rate clock



  - Clock period: duration of a clock cycle
    - e.g., 250ps = 0.25ns = 250×10^{-12}s
  - Clock frequency (rate): cycles per second
    - e.g., 4.0GHz = 4000MHz = 4.0×10^{9}Hz

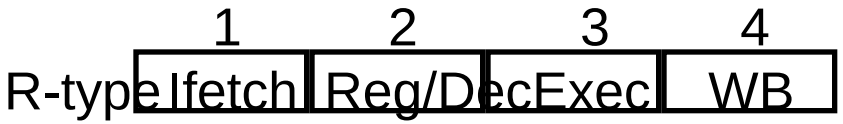
What's a Clock Cycle?



Why is clocking important?

- Typically, the execution of different instructions can take different number of clock cycles (we'll see later in Ch4).
  - Load instruction needs 5 clock cycles:

  - R-type (arithmetic/logic) and store instructions take 4 clock cycles:



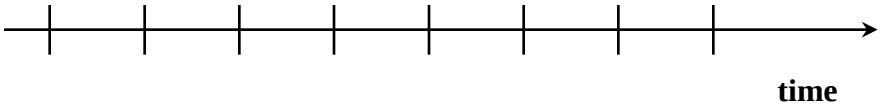
  - Branch instructions take 3 clock cycles. etc…
- ➔ Depending on the instruction mix in the program being run, the total number of clock cycles will vary!

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):

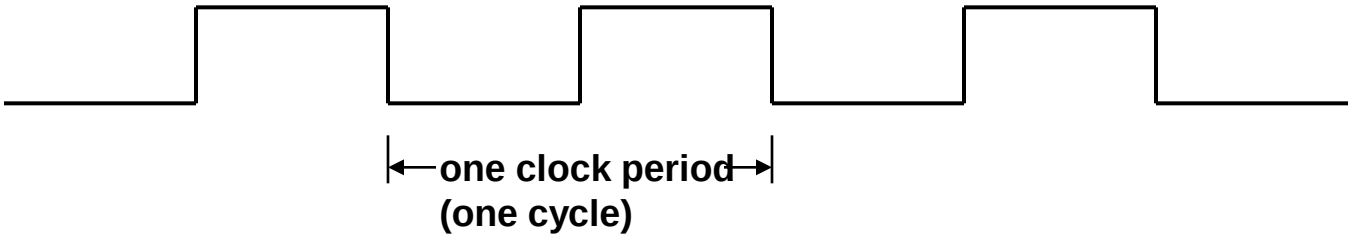


- cycle time = time between ticks = seconds per cycle

Review: Machine Clock Rate

- Definition: 1 Hz = 1 cycle/second
- Clock rate (MHz, GHz) is inverse of clock cycle time (clock period)

$$CC = 1 / CR$$



10 nsec clock cycle ➔ 100 MHz clock rate
5 nsec clock cycle ➔ 200 MHz clock rate
2 nsec clock cycle ➔ 500 MHz clock rate
1 nsec clock cycle ➔ 1 GHz clock rate
500 psec clock cycle ➔ 2 GHz clock rate
250 psec clock cycle ➔ 4 GHz clock rate
200 psec clock cycle ➔ 5 GHz clock rate

How to Improve Performance

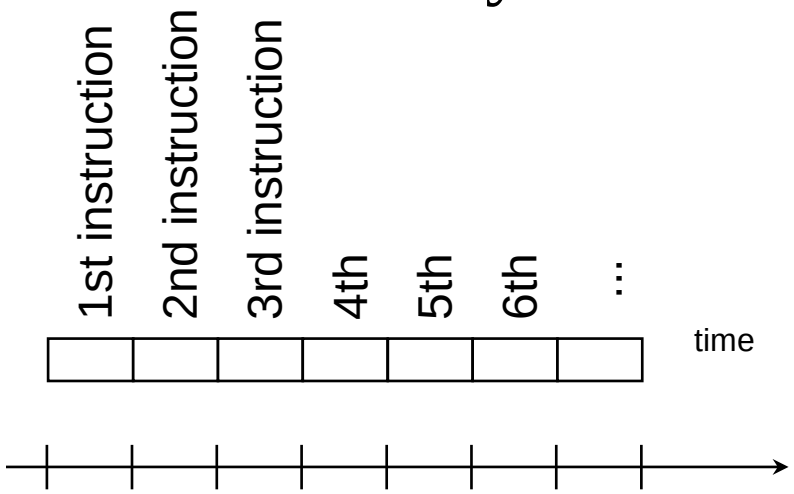
$$\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}$$

- 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.

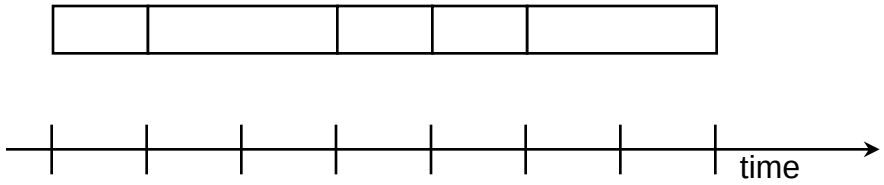
How many cycles are required for a program?

- Could assume that # of cycles = # of instructions



- This assumption is incorrect
  - different instructions may take different amounts of time on different machines. Why? (see next slide)

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 (and often can have a random element)
- Important point: changing the cycle time may change the number of cycles required for various instructions due to design constraints.

Clock cycles per instruction

- Not all instructions take the same amount of time to execute
  - One way to think about execution time is that it equals the number of instructions executed multiplied by the average time per instruction

$$\#\text{ of CPU cycles for a program} = \#\text{ of instructions} \times \underbrace{\text{average CPI}}_{\text{statistical quantity}}$$

- Clock cycles per instruction (CPI) – the average number of clock cycles each instruction takes to execute
  - A way to compare two different implementations of the same ISA

	CPI for instruction class		
	A	B	C
CPI	1	2	3

Effective CPI

- Computing the overall effective CPI is done by looking at the different types of instructions and their individual cycle counts and averaging

$$\text{CPU clock cycles} = \sum_{i=1}^{n} (CPI_i \times C_i)$$

$$\text{overall effective CPI} = \frac{\text{CPU clock cycles}}{\text{total \# of instructions}} = \frac{\text{CPU clock cycles}}{\sum_{i=1}^{n} C_i}$$

  - Where C_i is the instruction count of the number of instructions of class i executed
  - CPI_i is the (average) number of clock cycles per instruction for that instruction class
  - n is the number of instruction classes
- The overall effective CPI varies by instruction mix – a measure of the dynamic frequency of instructions across one or many programs

The Performance Equation

- Our basic performance equation is then

$$\text{CPU time} = \text{Instruction count} \times \textit{CPI} \times \text{clock cycle time}$$

$$\text{CPU time} = \frac{\text{Instruction count} \times \text{CPI}}{\text{clock rate}}$$

- These equations separate the three key factors that affect performance
  - Can measure the CPU execution time by running the program
  - The clock rate is usually given
  - Can measure overall instruction count by using profilers/ simulators without knowing all of the implementation details
  - CPI varies by instruction type and ISA implementation for which we must know the implementation details

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?“
- Don't Panic, can easily work this out from basic principles

Solution

$$CPUtime_A = \frac{clockcycles_A}{clockrate_A}$$

$$10\,\text{sec} = \frac{clockcycles_A}{400 \times 10^6\, cycles/\text{sec}}$$

$$clockcycles_A = 10\,\text{sec} \times 400 \times 10^6\, cycles/\text{sec}$$

$$= 4 \times 10^9\, cycles$$

Continued on next page

Solution (continued)

$$1.2 \times clockcycles_A = clockcycles_B$$

$$= 1.2 \times 4 \times 10^9 \, cycles = 4.8 \times 10^9 \, cycles$$

$$CPUtime_B = \frac{clockcycles_B}{clockrate_B} \Rightarrow$$

$$clockrate_B = \frac{clockcycles_B}{CPUtime_B} = \frac{4.8 \times 10^9 \, cycles}{6 \sec}$$

$$= 0.8 \times 10^9 \, cycles / \sec = 800 MHz$$

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 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? (Solution on next page)*

Solution

$$CPUtime_A = \underbrace{clockcycles_A}_{\#\text{ of instructions} \times \text{CPI}} \times \underbrace{cycletime_A}_{10ns}$$

$$\underbrace{\qquad\qquad\qquad\qquad}_{\#\text{ of instructions} \times 2 \times 10\text{ns}}$$

$$CPUtime_B = (\#\text{ of instructions}) \times 1.2 \times 20ns$$

Solutions (cont)

Let I = # of instructions

$$\frac{CPUperf_A}{CPUperf_B} = \frac{CPUtime_B}{CPUtime_A} = \frac{I \times 1.2 \times 20}{I \times 2 \times 10}$$

$$= \frac{2.4}{2} = 1.2$$

Machine A is 1.2 times faster than B

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.

Instr class	**CPI**
A	1
B	2
C	3

	A	**B**	**C**
Code1	2	1	2
Code2	4	1	1

of Instructions Example

- Which sequence will be faster? How much?
  What is the CPI for each sequence?

$$clock cycle\ s_1 = (2 \times \underbrace{1}_{A}) + (1 \times \underbrace{2}_{B}) + (2 \times \underbrace{3}_{C}) = 2 + 2 + 6 = 10$$

Faster=> $clock cycle s_2 = (4 \times 1) + (1 \times 2) + (1 \times 3) = 4 + 2 + 3 = 9$

$$CPI_1 = \frac{cycles_1}{(\#\ \text{of instructions})_1} = \frac{10}{5} = 2$$

$$CPI_2 = \frac{cycles_2}{(\#\ \text{of instructions})_2} = \frac{9}{6} = 1.5$$

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?

Solution

$$Execution = \frac{clock\ cycles}{clock\ rate}$$

$$time_1 = \frac{(5mil \times 1) + (1mil \times 2) + (1mil \times 3)\ cycles}{100 \times 10^6\ cycles / \sec}$$

$$= \frac{(5+2+3) \times 10^6\ cycles}{100 \times 10^6\ cycles / \sec} = \frac{10}{100} \sec = 0.1 \sec$$

$$time_2 = \frac{(10mil \times 1) + (1mil \times 2) + (1mil \times 3) cycles}{100 \times 10^6\ cycles / \sec}$$

$$= \frac{(10+2+3) \times 10^6}{100 \times 10^6} = \frac{15}{100} \sec = 0.15 \sec$$

Compiler 1 Faster than Compiler 2 For execution Time

Solution

Compiler 2
faster using MIPS

$$MIPS = \frac{\#\text{ of instructions}}{\text{execution time}} \times 10^{-6}$$

$$MIPS_1 = \frac{7 \times 10^6 instr}{0.1\text{sec}} \times 10^{-6} = 70 MIPS$$

$$MIPS_2 = \frac{12 \times 10^6 instr}{0.15} \times 10^{-6} = 80 MIPS$$

Amdahl's Law

- Pitfall: Improving an aspect of a computer and expecting a proportional improvement in overall performance
- Amdahl's law says:
  Execution Time After Improvement =

  Execution Time Unaffected +

  (Execution Time Affected / Amount of Improvement)
- Gained overall improvement is limited by the part of the system that cannot be changed.

Amdahl's Law

- 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?"

- Answer: exec time after improvement=100/4 =25sec

$$25\,\text{sec} = \frac{80}{n} + 20$$

➔ n=16 times

Amdahl's Law

How about making it 5 times faster?

Answer: exec time after improvement = 100/5 = 20 sec

$$20 = \frac{80}{n} + 20$$

$$\Rightarrow \frac{80}{n} = 0$$

$$\Rightarrow n = \infty$$

Can't be done!

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?

Answer: 5/5 + 5 = 6 seconds

Example

- We are looking for a benchmark to show off the new floating-point unit described above (new FP unit runs 5 times faster), 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?

- Solution on next page.

Example (solution)

$$\text{speedup of } 3 \Rightarrow \frac{100}{3} = 33\text{sec new exec time}$$

$$33 = \frac{x}{5} + (100 - x) = \frac{x}{5} + \frac{500 - 5x}{5} = \frac{500 - 4x}{5}$$

$$165 = 500 - 4x$$

$$4x = 500 - 165 = 335$$

$$x \cong 84 \text{ seconds must be FP operations}$$

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 (Standard Performance Evaluation Corp): www.spec.org
  - Originally: 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)
  - Develops benchmarks for CPU, I/O, Web, …

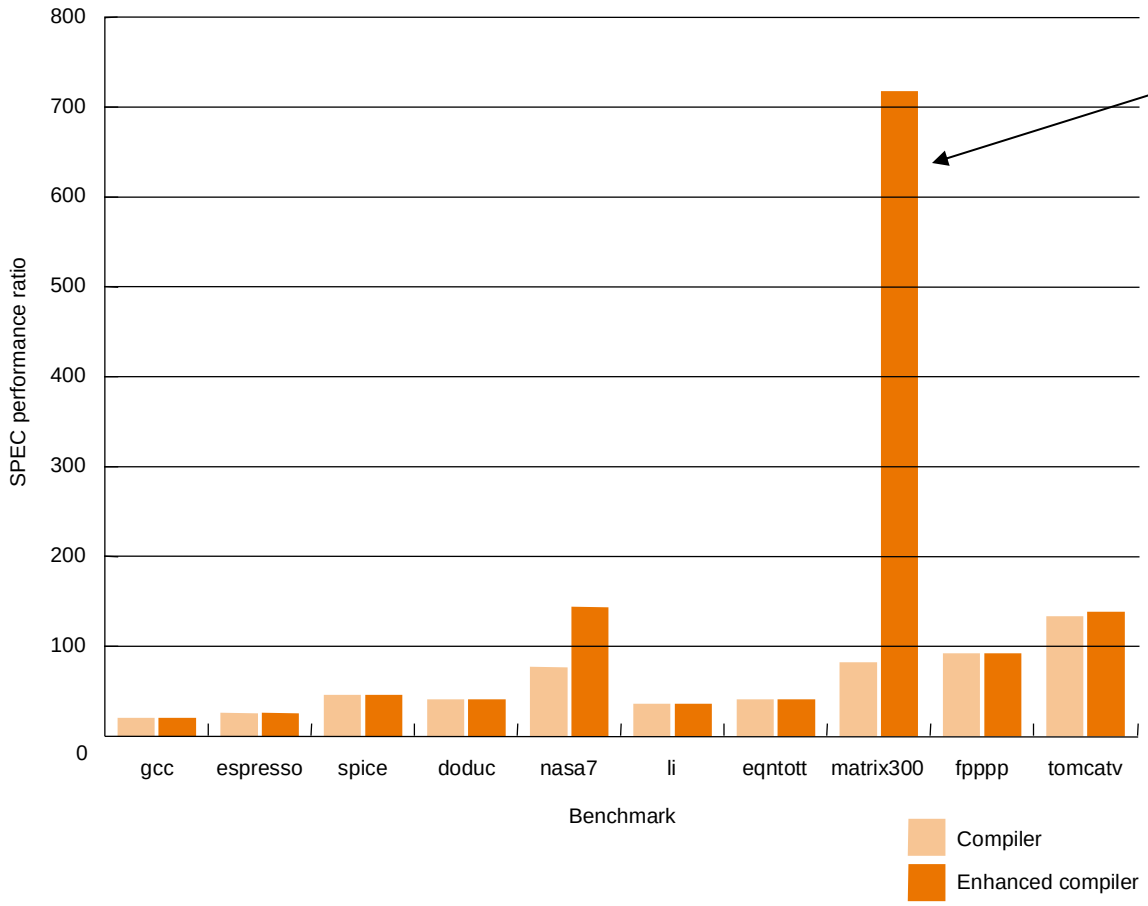
Benchmarks

$$SPECmark = \frac{\text{time on SPARCstation 10/40}}{\text{time on target machine}}$$

SPECmark larger ➔ performance better

SPEC ‘89

- Compiler “enhancements” and performance



Spike is because the enhanced compiler was optimized for 1 statement in matrix multiply which was executed 99% of the 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

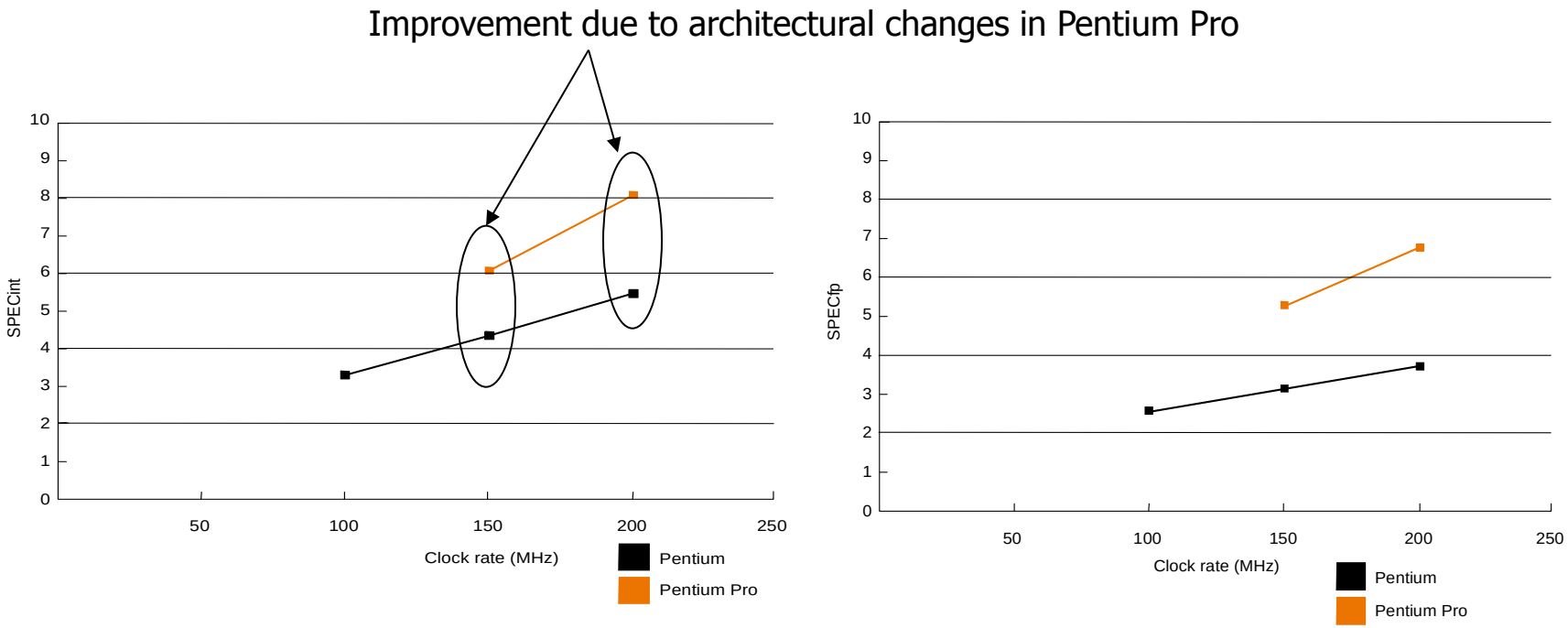
8 integer

10 FP apps

SPEC ‘95

Does doubling the clock rate double the performance?

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



Lack of 1:1 improvement from increasing clock rate due to slowness of memory

Spec2000 benchmarks

- URL: http://www.specbench.org/cpu2000/
- Integer benchmarks

Benchmark	**Description**
gzip	Compression
vpr	FPGA Circuit Placement and Routing
gcc	Programming Language Compiler
mcf	Combinatorial Optimization
Crafty	Game Playing: Chess
Parser	Word Processing
Eon	Computer Visualization
Perlbmk	PERL Programming Language
Gap	Group Theory, Interpreter
Vortex	Object-oriented Database
Bzip2	Compression
Twolf	Place and Route Simulator

Spec2000 benchmarks

- FP benchmarks

Benchmark	Description
wupwise	Physics / Quantum Chromodynamics
swim	Shallow Water Modeling
mgrid	Multi-grid Solver: 3D Potential Field
applu	Parabolic / Elliptic Partial Differential Equations
mesa	3-D Graphics Library
galgel	Computational Fluid Dynamics
art	Image Recognition / Neural Networks
equake	Seismic Wave Propagation Simulation
facerec	Image Processing: Face Recognition
ammp	Computational Chemistry
lucas	Number Theory / Primality Testing
fma3d	Finite-element Crash Simulation
sixtrack	High Energy Nuclear Physics Accelerator Design
apsi	Meteorology: Pollutant Distribution

SPEC CPU Benchmark

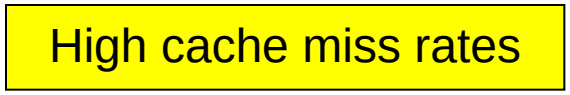
- SPEC CPU2006
  - Elapsed time to execute a selection of programs
    - Negligible I/O, so focuses on CPU performance
  - Normalize relative to reference machine
  - Summarize as geometric mean of performance ratios
    - CINT2006 (integer) and CFP2006 (floating-point)

$$\sqrt[n]{\prod_{i=1}^{n} \text{Execution time ratio}_i}$$

Why geometric mean? Because we are comparing *ratios*.

CINT2006 for Opteron X4 2356

Name	Description	IC × 10^9	CPI	Tc (ns)	Exec time	Ref time	SPECratio
perl	Interpreted string processing	2,118	0.75	0.40	637	9,777	15.3
bzip2	Block-sorting compression	2,389	0.85	0.40	817	9,650	11.8
gcc	GNU C Compiler	1,050	1.72	0.47	24	8,050	11.1
mcf	Combinatorial optimization	336	10.00	0.40	1,345	9,120	6.8
go	Go game (AI)	1,658	1.09	0.40	721	10,490	14.6
hmmer	Search gene sequence	2,783	0.80	0.40	890	9,330	10.5
sjeng	Chess game (AI)	2,176	0.96	0.48	37	12,100	14.5
libquantum	Quantum computer simulation	1,623	1.61	0.40	1,047	20,720	19.8
h264avc	Video compression	3,102	0.80	0.40	993	22,130	22.3
omnetpp	Discrete event simulation	587	2.94	0.40	690	6,250	9.1
astar	Games/path finding	1,082	1.79	0.40	773	7,020	9.1
xalancbmk	XML parsing	1,058	2.70	0.40	1,143	6,900	6.0
Geometric mean							11.7



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!