

Quote

- “Computers are incredibly fast, accurate and stupid;
- humans are incredibly slow, inaccurate, and brilliant;
- together they are powerful beyond imagination.”

- Albert Einstein

Important Prerequisites

- Logic Design
- Hands-On Assembly Language programming




Chapter 1


Computer Abstractions and Technology

Electrical & Computer Engineering
School of Engineering
THE COLLEGE OF NEW JERSEY

Introduction

- This course is all about how computers work
- But what do we mean by a computer? 

Introduction

- This course is all about how computers work
- But what do we mean by a computer?
 - Different types: desktop, servers, embedded devices
 - Different uses: automobiles, graphics, finance, genomics...

Introduction

- But what do we mean by a computer?
 - Different manufacturers: Intel, Apple, IBM, Microsoft, Sun...
 - Different underlying technologies and different costs!

Introduction

- Analogy: Consider a course on “automotive vehicles”
 - Many similarities from vehicle to vehicle (e.g., wheels)
 - Huge differences from vehicle to vehicle (e.g., gas vs. electric)

Introduction

- Best way to learn:
 - Focus on a specific instance and learn how it works
 - While learning general principles and historical perspectives

Introduction

- Rapidly changing field:
 - vacuum tube -> transistor -> IC -> VLSI (see section 1.4)
 - doubling every 1.5 years (since 1970):
 - memory capacity*
 - processor speed* (Due to advances in technology (~x30K: 0.1MHz to 3GHz) and organization (~x20: 10 CPI to 0.5 CPI))

Introduction

- 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)

Introduction

- Why learn this stuff?
 - you want to call yourself an “electrical engineer”, a “computer engineer”, or a “computer scientist”
 - you want to build software people use (need performance)
 - you need to make a purchasing decision or offer “expert” advice

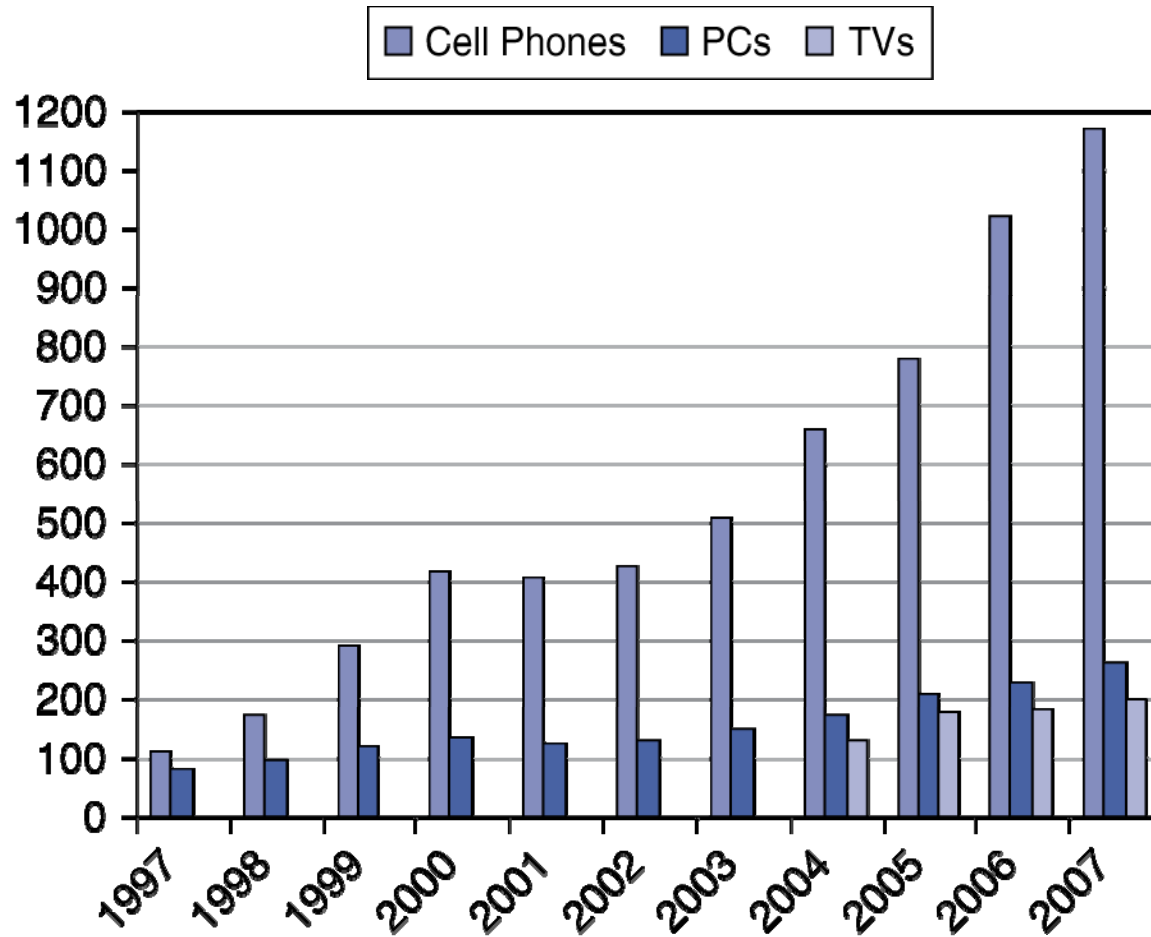
The Computer Revolution

- Progress in computer technology
 - Underpinned by Moore's Law
- Makes novel applications feasible
 - Computers in automobiles
 - Cell phones
 - Human genome project
 - World Wide Web
 - Search Engines
- Computers are pervasive

Classes of Computers

- Desktop computers
 - General purpose, variety of software
 - Subject to cost/performance tradeoff
- Server computers
 - Network based
 - High capacity, performance, reliability
 - Range from small servers to building sized
- Embedded computers
 - Hidden as components of systems
 - Stringent power/performance/cost constraints

The Processor Market



What You Will Learn

- How programs are translated into the machine language
 - And how the hardware executes them
- The hardware/software interface
- What determines program performance
 - And how it can be improved
- How hardware designers improve performance
- What is parallel processing

Understanding Performance

- Algorithm
 - Determines number of operations executed
- Programming language, compiler, architecture
 - Determine number of machine instructions executed per operation
- Processor and memory system
 - Determine how fast instructions are executed
- I/O system (including OS)
 - Determines how fast I/O operations are executed

What is a computer?

- Components:
 - input (mouse, keyboard)
 - output (display, printer)
 - memory (disk drives, DRAM, SRAM, CD)
 - network

What is a computer?

- Our primary focus: the processor (datapath and control)
 - implemented using millions of transistors
 - Impossible to understand by looking at each transistor
 - We need...

Abstractions

■ The BIG Picture

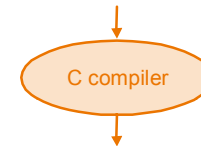
- Abstraction helps us deal with complexity
 - Hide lower-level detail
- Instruction set architecture (ISA)
 - The hardware/software interface
- Application binary interface
 - The ISA plus system software interface
- Implementation
 - The details underlying and interface

Abstraction

- Delving into the depths reveals more information

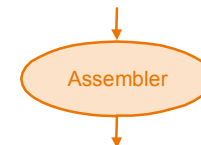
High-level language program (in C)

```
swap(int v[], int k)
{int temp;
  temp = v[k];
  v[k] = v[k+1];
  v[k+1] = temp;
}
```



Assembly language program (for MIPS)

```
swap:
  muli $2, $5,4
  add $2, $4,$2
  lw $15, 0($2)
  lw $16, 4($2)
  sw $16, 0($2)
  sw $15, 4($2)
  jr $31
```



Binary machine language program (for MIPS)

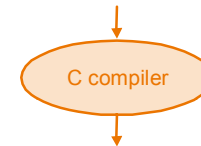
```
000000001010000100000000000011000
00000000100011100001100000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
0000001111100000000000000001000
```

Abstraction

- An abstraction omits unneeded detail, helps us cope with complexity

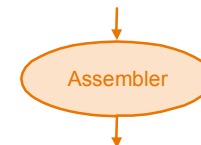
High-level language program (in C)

```
swap(int v[], int k)
{int temp;
 temp = v[k];
 v[k] = v[k+1];
 v[k+1] = temp;
}
```



Assembly language program (for MIPS)

```
swap:
  muli $2, $5,4
  add $2, $4,$2
  lw $15, 0($2)
  lw $16, 4($2)
  sw $16, 0($2)
  sw $15, 4($2)
  jr $31
```



Binary machine language program (for MIPS)

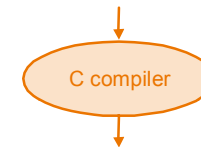
```
000000001010000100000000000011000
00000000100011100001100000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

Abstraction

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

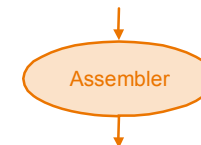
High-level
language
program
(in C)

```
swap(int v[], int k)
{int temp;
  temp = v[k];
  v[k] = v[k+1];
  v[k+1] = temp;
}
```



Assembly
language
program
(for MIPS)

```
swap:
  muli $2, $5,4
  add $2, $4,$2
  lw $15, 0($2)
  lw $16, 4($2)
  sw $16, 0($2)
  sw $15, 4($2)
  jr $31
```



Binary machine
language
program
(for MIPS)

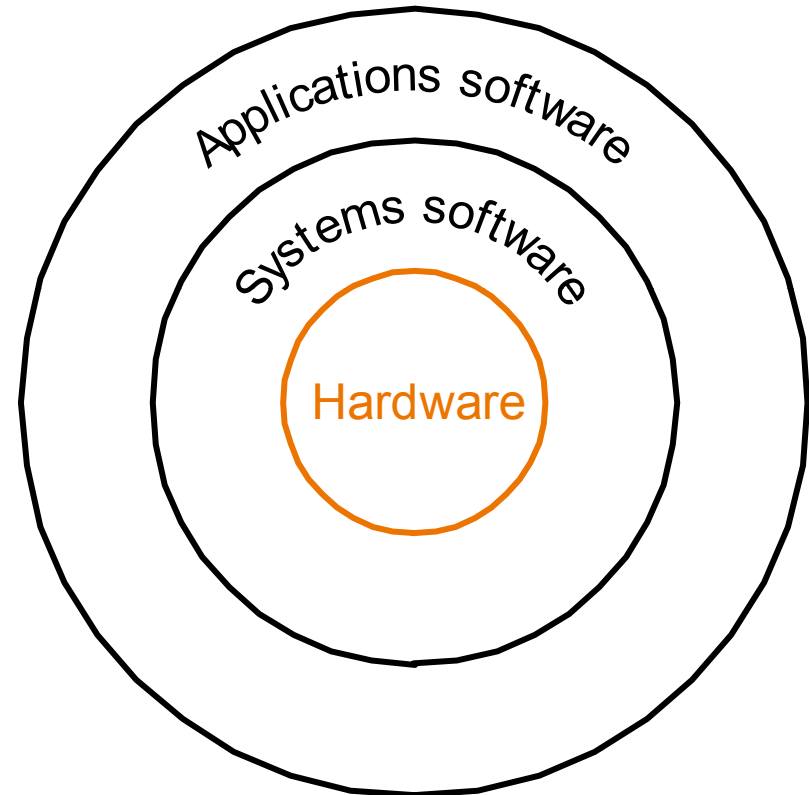
```
000000001010000100000000000011000
00000000100011100001100000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
0000001111100000000000000001000
```

How do computers work?

- Need to understand abstractions such as:
 - Applications software
 - Systems software
 - Assembly Language
 - Machine Language
 - Architectural Issues: i.e., Caches, Virtual Memory, Pipelining
 - Sequential logic, finite state machines
 - Combinational logic, arithmetic circuits
 - Boolean logic, 1s and 0s
 - Transistors used to build logic gates (CMOS)
 - Semiconductors/Silicon used to build transistors
 - Properties of atoms, electrons, and quantum dynamics
- So much to learn!

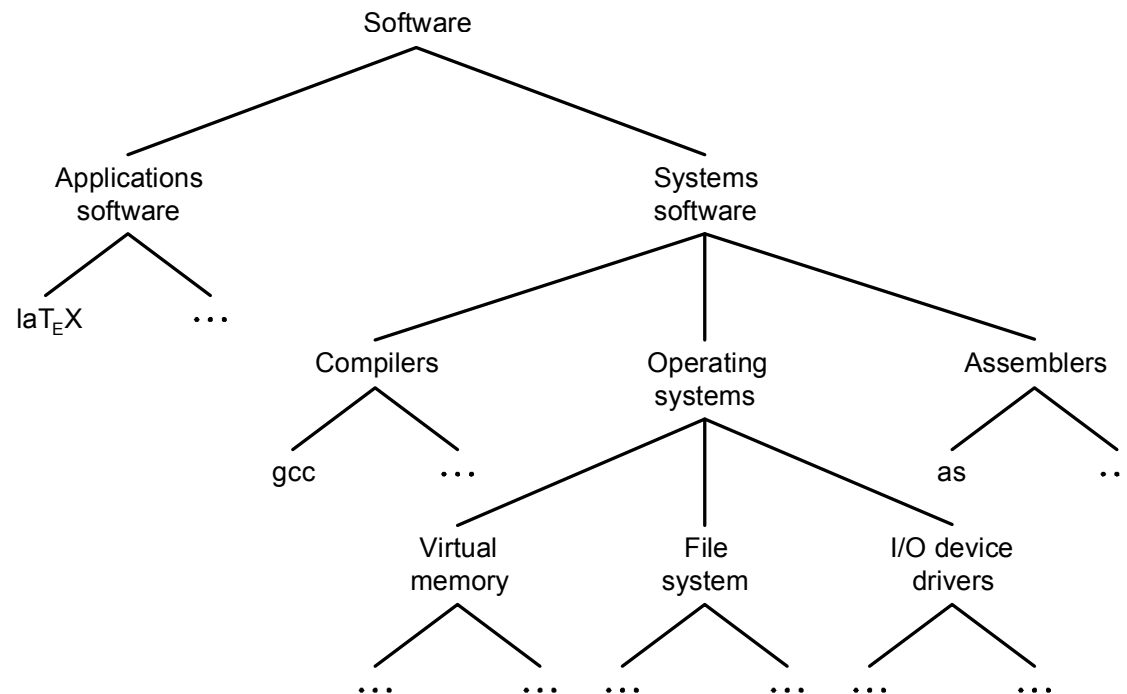
Hierarchy

- Helps us understand the pieces of a whole, and the building block interaction between them



(Software) Dichotomy

- Helps us catalogue and classify ... concepts, types ...



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*

Instruction Set Architecture

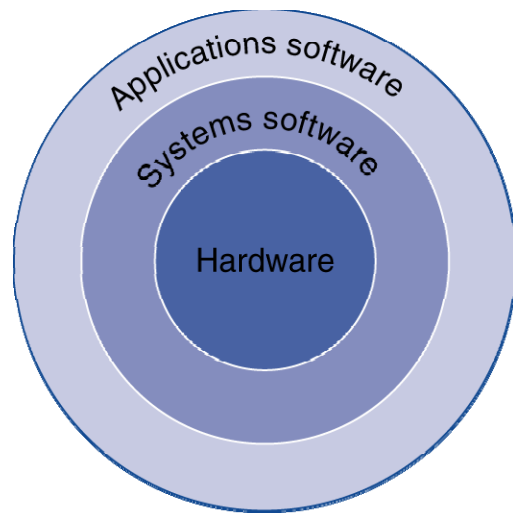
- A very important abstraction
 - disadvantage: *sometimes prevents using new innovations*

True or False: Binary compatibility is extraordinarily important? 

Instruction Set Architecture

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

Below Your Program



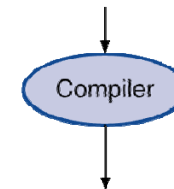
- Application software
 - Written in high-level language
- System software
 - Compiler: translates HLL code to machine code
 - Operating System: service code
 - Handling input/output
 - Managing memory and storage
 - Scheduling tasks & sharing resources
- Hardware
 - Processor, memory, I/O controllers

Levels of Program Code

- High-level language
 - Level of abstraction closer to problem domain
 - Provides for productivity and portability
- Assembly language
 - Textual representation of instructions
- Hardware representation
 - Binary digits (bits)
 - Encoded instructions and data

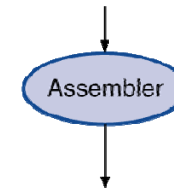
High-level
language
program
(in C)

```
swap(int v[], int k)
{int temp;
  temp = v[k];
  v[k] = v[k+1];
  v[k+1] = temp;
}
```



Assembly
language
program
(for MIPS)

```
swap:
  muli $2, $5, 4
  add $2, $4, $2
  lw $15, 0($2)
  lw $16, 4($2)
  sw $16, 0($2)
  sw $15, 4($2)
  jr $31
```

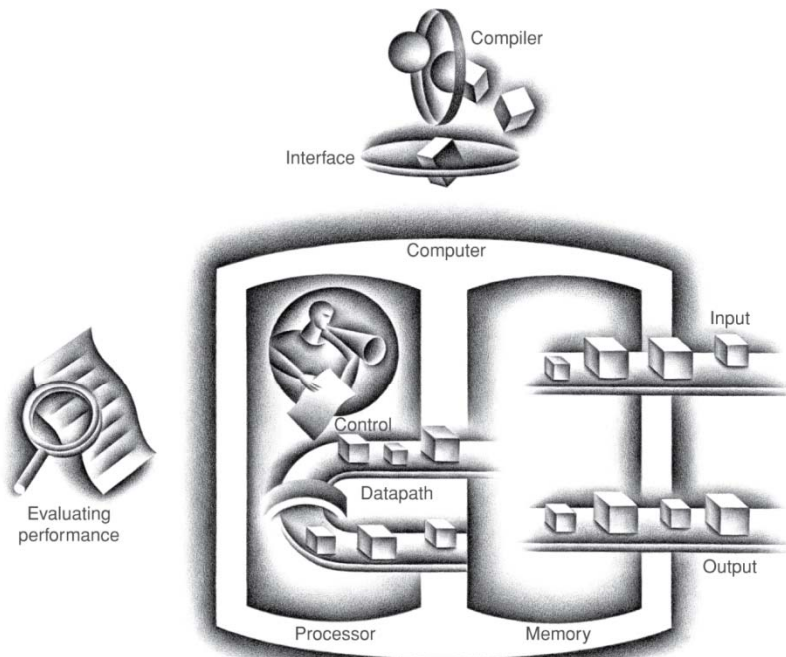


Binary machine
language
program
(for MIPS)

```
0000000010100001000000000011000
0000000000011000000110000100001
1000110001100010000000000000000
1000110011110010000000000000100
1010110011110010000000000000000
1010110001100010000000000000100
000000111110000000000000001000
```

Components of a Computer

The BIG Picture



- Same components for all kinds of computer
 - Desktop, server, embedded
- Input/output includes
 - User-interface devices
 - Display, keyboard, mouse
 - Storage devices
 - Hard disk, CD/DVD, flash
 - Network adapters
 - For communicating with other computers

Anatomy of a Computer

■ Output device



■ Network cable



■ Input device

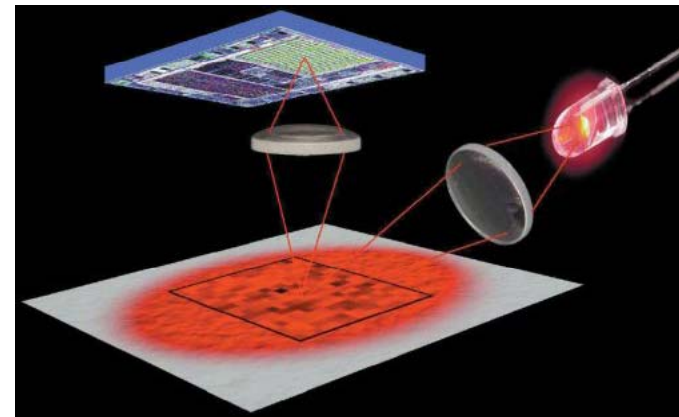


■ Input device



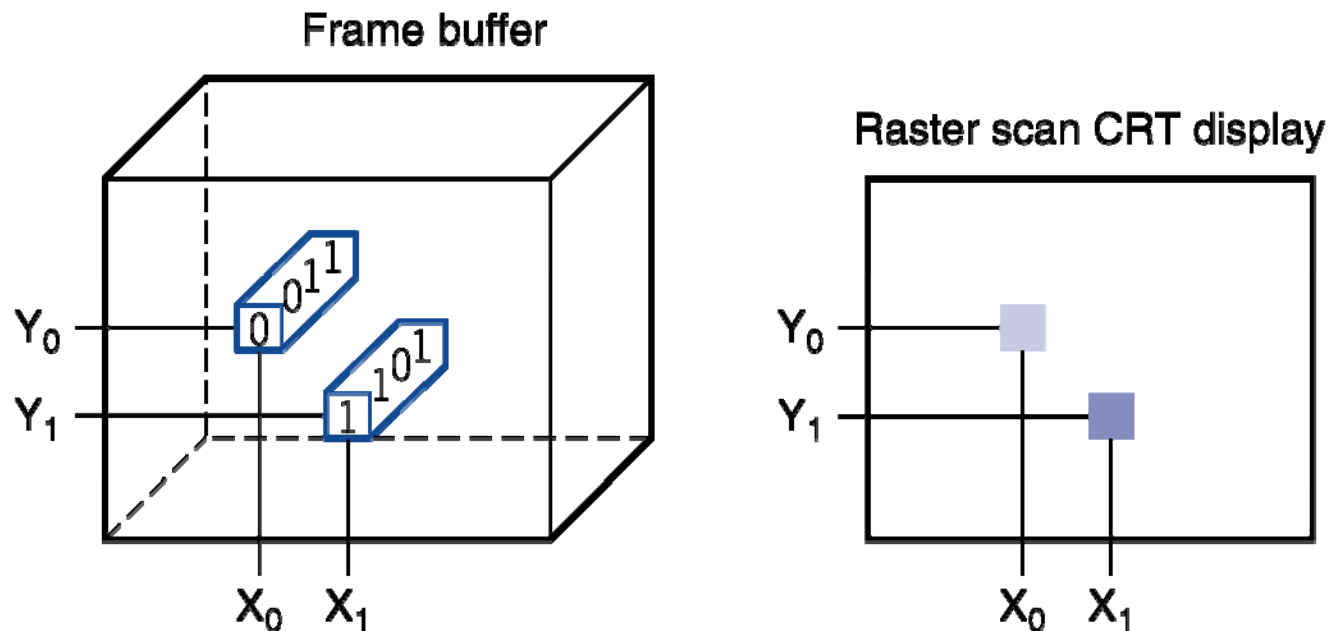
Anatomy of a Mouse

- Optical mouse
 - LED illuminates desktop
 - Small low-res camera
 - Basic image processor
 - Looks for x, y movement
 - Buttons & wheel
- Supersedes roller-ball mechanical mouse

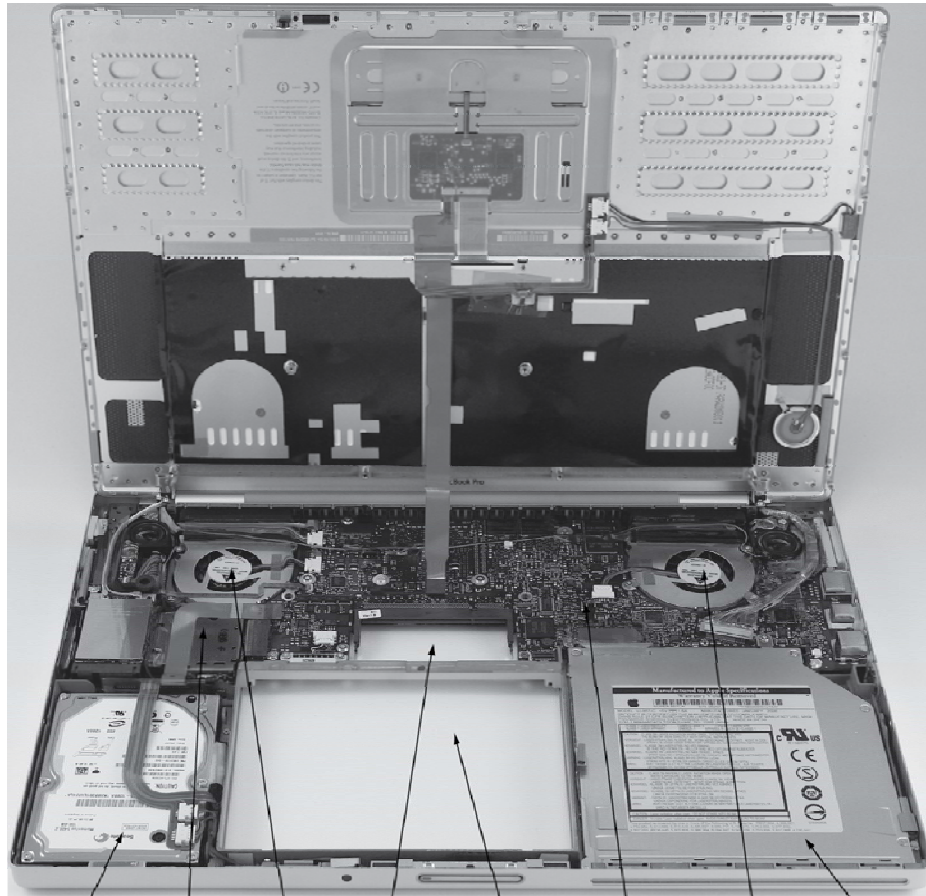


Through the Looking Glass

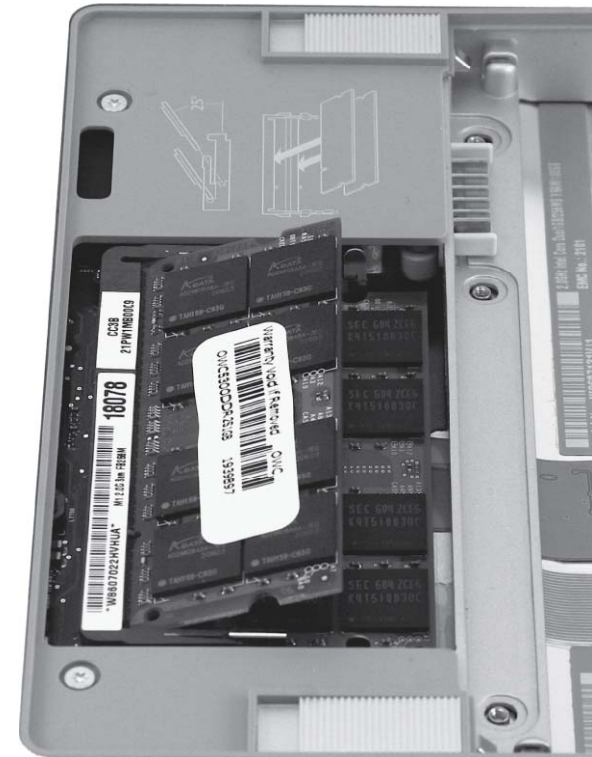
- LCD screen: picture elements (pixels)
 - Mirrors content of frame buffer memory



Opening the Box



Hard drive Processor Fan with cover Spot for memory DIMMs Spot for battery Motherboard Fan with cover DVD drive cover

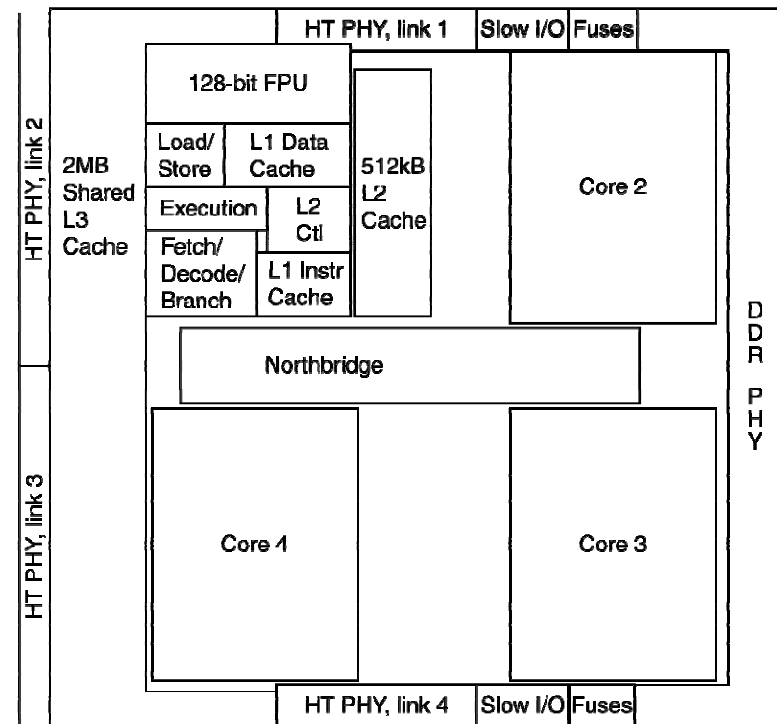
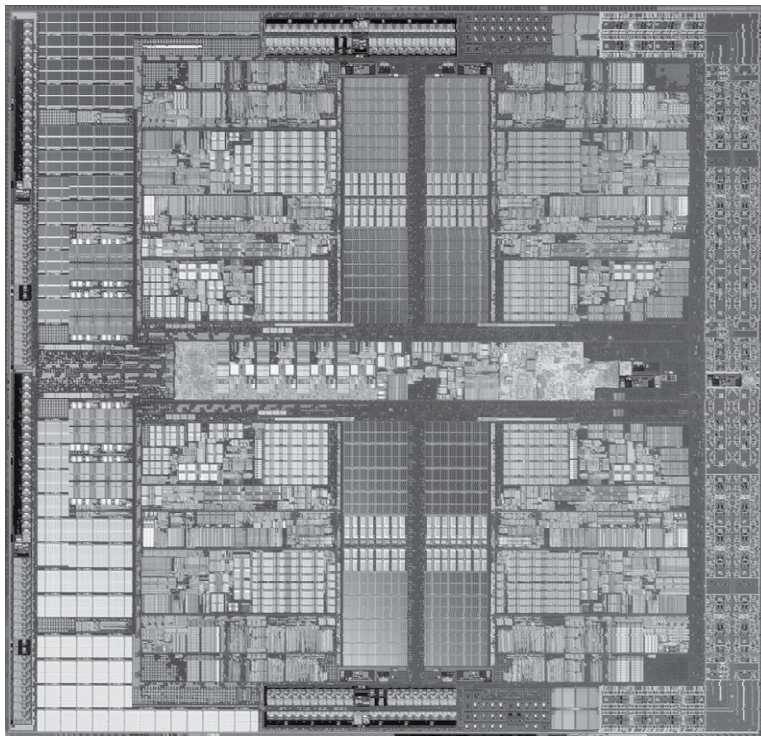


Inside the Processor (CPU)

- Datapath: performs operations on data
- Control: sequences datapath, memory, ...
- Cache memory
 - Small fast SRAM memory for immediate access to data

Inside the Processor

- AMD Barcelona: 4 processor cores



A Safe Place for Data

- Volatile main memory
 - Loses instructions and data when power off
- Non-volatile secondary memory
 - Magnetic disk
 - Flash memory
 - Optical disk (CDROM, DVD)



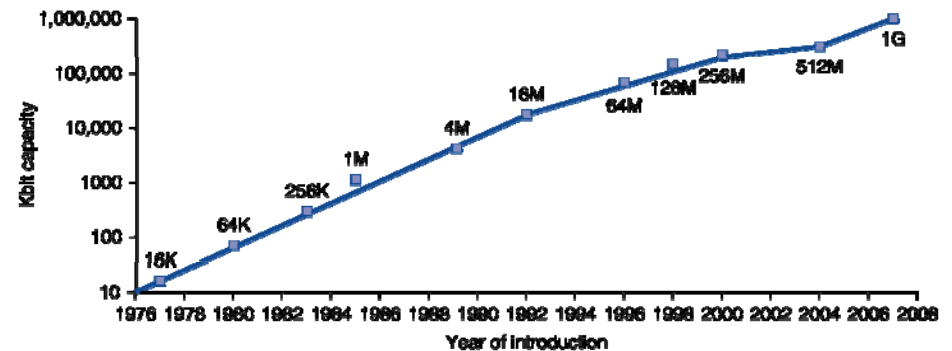
Networks

- Communication and resource sharing
- Local area network (LAN): Ethernet
 - Within a building
- Wide area network (WAN): the Internet
- Wireless network: WiFi, Bluetooth



Technology Trends

- Electronics technology continues to evolve
 - Increased capacity and performance
 - Reduced cost



■ DRAM capacity

Year	Technology	Relative performance/cost
1951	Vacuum tube	1
1965	Transistor	35
1975	Integrated circuit (IC)	900
1995	Very large scale IC (VLSI)	2,400,000
2005	Ultra large scale IC	6,200,000,000

Performance

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

Performance

Why is some hardware better than others for different programs?

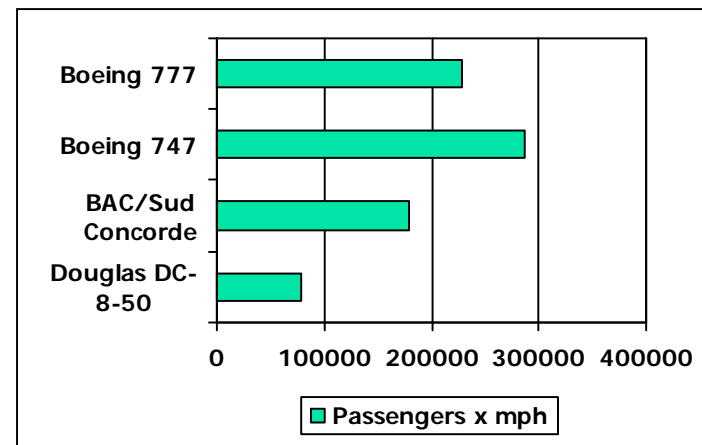
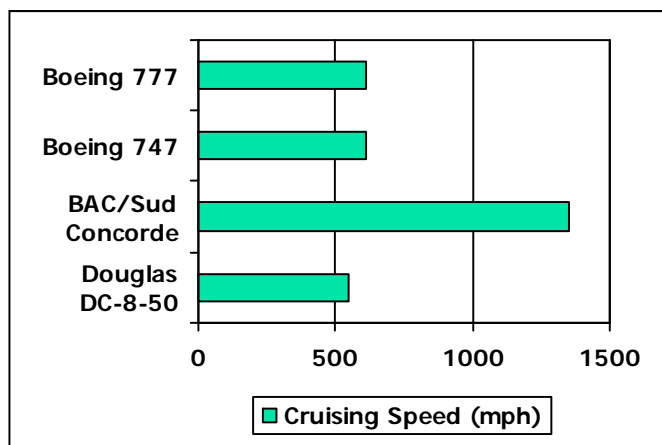
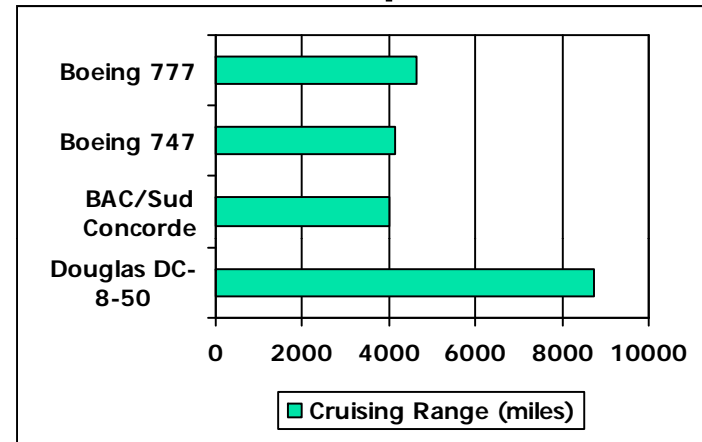
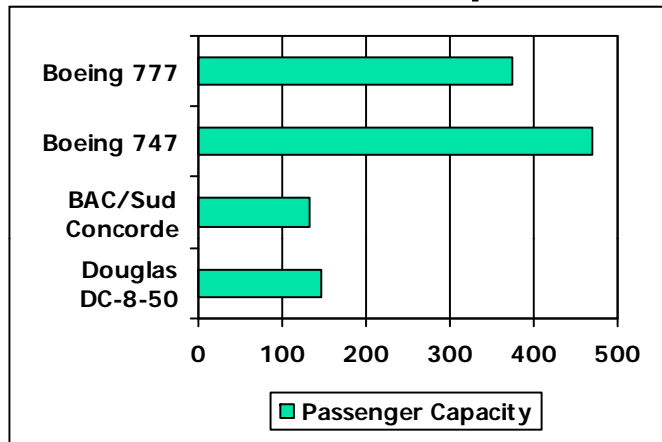
What factors of system performance are hardware related?

(e.g., Do we need a new machine, or a new operating system?)

How does the machine's instruction set affect performance?

Defining Performance

- Which airplane has the best performance?



Response Time and Throughput

- Response time
 - How long it takes to do a task
- Throughput
 - Total work done per unit time
 - e.g., tasks/transactions/... per hour
- How are response time and throughput affected by
 - Replacing the processor with a faster version?
 - Adding more processors?
- We'll focus on response time for now...

Computer Performance: TIME, TIME, TIME

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?



Book's Definition of Performance

- For some program running on machine X ,

$$\text{Performance}_X = 1 / \text{Execution time}_X$$

- "X is n times faster than Y"

$$\text{Performance}_X / \text{Performance}_Y = n$$



Book's Definition of Performance

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

Relative Performance

- Define Performance = 1/Execution Time
- "X is n time faster than Y"

$$\begin{aligned} & \text{Performance}_X / \text{Performance}_Y \\ &= \text{Execution time}_Y / \text{Execution time}_X = n \end{aligned}$$

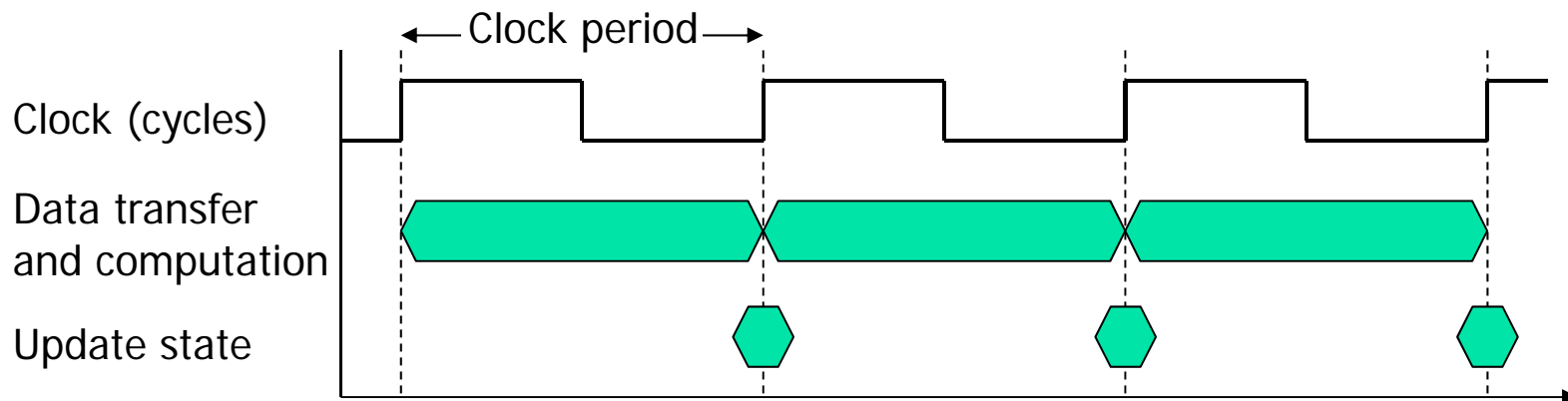
- Example: time taken to run a program
 - 10s on A, 15s on B
 - $\text{Execution Time}_B / \text{Execution Time}_A$
= 15s / 10s = 1.5
 - So A is 1.5 times faster than B

Measuring Execution Time

- Elapsed time
 - Total response time, including all aspects
 - Processing, I/O, OS overhead, idle time
 - Determines system performance
- CPU time
 - Time spent processing a given job
 - Discounts I/O time, other jobs' shares
 - Comprises user CPU time and system CPU time
 - Different programs are affected differently by CPU and system performance

CPU Clocking

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



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

CPU Time

$$\begin{aligned}\text{CPU Time} &= \text{CPU Clock Cycles} \times \text{Clock Cycle Time} \\ &= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}\end{aligned}$$

- Performance improved by
 - Reducing number of clock cycles
 - Increasing clock rate
 - Hardware designer must often trade off clock rate against cycle count

CPU Time Example

- Computer A: 2GHz clock, 10s CPU time
- Designing Computer B
 - Aim for 6s CPU time
 - Can do faster clock, but causes $1.2 \times$ clock cycles
- How fast must Computer B clock be?

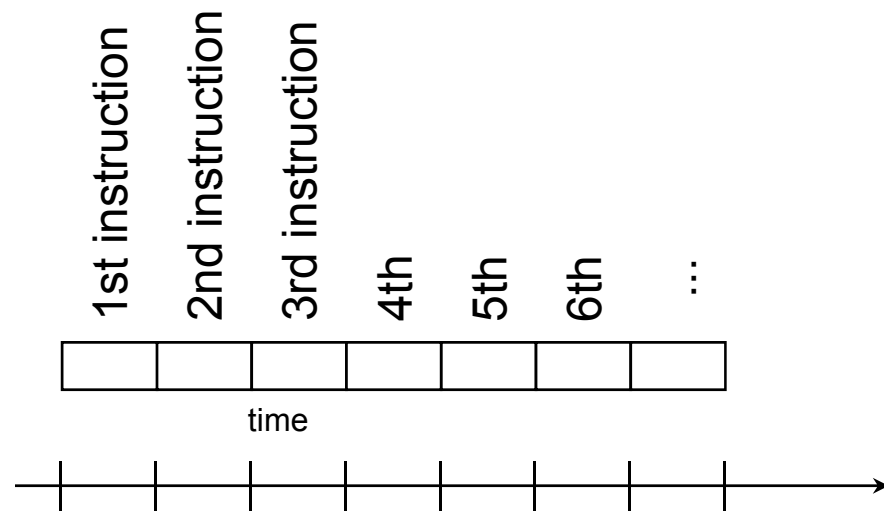
$$\text{Clock Rate}_B = \frac{\text{Clock Cycles}_B}{\text{CPU Time}_B} = \frac{1.2 \times \text{Clock Cycles}_A}{6\text{s}}$$

$$\begin{aligned} \text{Clock Cycles}_A &= \text{CPU Time}_A \times \text{Clock Rate}_A \\ &= 10\text{s} \times 2\text{GHz} = 20 \times 10^9 \end{aligned}$$

$$\text{Clock Rate}_B = \frac{1.2 \times 20 \times 10^9}{6\text{s}} = \frac{24 \times 10^9}{6\text{s}} = 4\text{GHz}$$

How many cycles are required for a program?

- Could assume that # of cycles = # of instructions

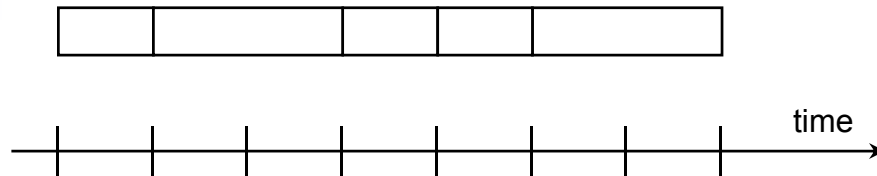


This assumption is incorrect,

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

Different numbers of cycles for different instructions

- *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?"
- 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

Now that we understand cycles

TIME TO EXECUTE A PROGRAM =
(INSTRUCTIONS / PROGRAM) x
(CYCLES / INSTRUCTION) x
(SECONDS / CYCLE)

- INSTRUCTIONS / PROGRAM
 - PROCESSOR ARCHITECTURE AND ORGANIZATION
 - COMPILER TECHNOLOGY
- CYCLES / INSTRUCTION
 - PROCESSOR ARCHITECTURE AND ORGANIZATION
- SECONDS / CYCLE
 - PROCESSOR ARCHITECTURE AND ORGANIZATION
 - SILICON TECHNOLOGY

Now that we understand cycles

- 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*

Instruction Count and CPI

Clock Cycles = Instruction Count \times Cycles per Instruction

CPU Time = Instruction Count \times CPI \times Clock Cycle Time

$$= \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}$$

- Instruction Count for a program
 - Determined by program, ISA and compiler
- Average cycles per instruction
 - Determined by CPU hardware
 - If different instructions have different CPI
 - Average CPI affected by instruction mix

CPI

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

- This is the average CPI for a program or for a machine/ISA
- Program:
 - Look at traces
 - CPI_i is the CPI for a given type/class of instruction
 - C_i is the count of those instructions in the program
- Machine/ISA
 - CPI_i is the CPI for a given type/class of instruction
 - C_i is the count of those instructions in the ISA

CPI in More Detail

- If different instruction classes take different numbers of cycles


$$\text{Clock Cycles} = \sum_{i=1}^n (\text{CPI}_i \times \text{Instruction Count}_i)$$

- Weighted average CPI


$$\text{CPI} = \frac{\text{Clock Cycles}}{\text{Instruction Count}} = \sum_{i=1}^n \left(\text{CPI}_i \times \frac{\text{Instruction Count}_i}{\text{Instruction Count}} \right)$$

Relative frequency

CPI Example

- Computer A: Cycle Time = 250ps, CPI = 2.0
- Computer B: Cycle Time = 500ps, CPI = 1.2
- Same ISA 
- Which is faster, and by how much?

CPI Example

- Computer A: Cycle Time = 250ps, CPI = 2.0
- Computer B: Cycle Time = 500ps, CPI = 1.2
- Same ISA 
- Which is faster, and by how much?

$$\begin{aligned} \text{CPU Time}_A &= \text{Instruction Count} \times \text{CPI}_A \times \text{Cycle Time}_A \\ &= 1 \times 2.0 \times 250\text{ps} = 1 \times 500\text{ps} \end{aligned}$$

A is faster...

$$\begin{aligned} \text{CPU Time}_B &= \text{Instruction Count} \times \text{CPI}_B \times \text{Cycle Time}_B \\ &= 1 \times 1.2 \times 500\text{ps} = 1 \times 600\text{ps} \end{aligned}$$

$$\frac{\text{CPU Time}_B}{\text{CPU Time}_A} = \frac{1 \times 600\text{ps}}{1 \times 500\text{ps}} = 1.2$$

...by this much

CPI Example

- Alternative compiled code sequences using instructions in classes A, B, C

Class	A	B	C
CPI for class	1	2	3
IC in sequence 1	2	1	2
IC in sequence 2	4	1	1

Sequence 1: IC = 5

Clock Cycles

$$= 2 \times 1 + 1 \times 2 + 2 \times 3$$

$$= 10$$

$$\text{Avg. CPI} = 10/5 = 2.0$$

Sequence 2: IC = 6

Clock Cycles

$$= 4 \times 1 + 1 \times 2 + 1 \times 3$$

$$= 9$$

$$\text{Avg. CPI} = 9/6 = 1.5$$

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

 ic the # of required cycles for a program, or

 ic the clock cycle time or, said another way,

 ic the clock rate.

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.

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?

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

Benchmarks

- 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)

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...*

Benchmark Games

- *...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.*

Benchmark Games

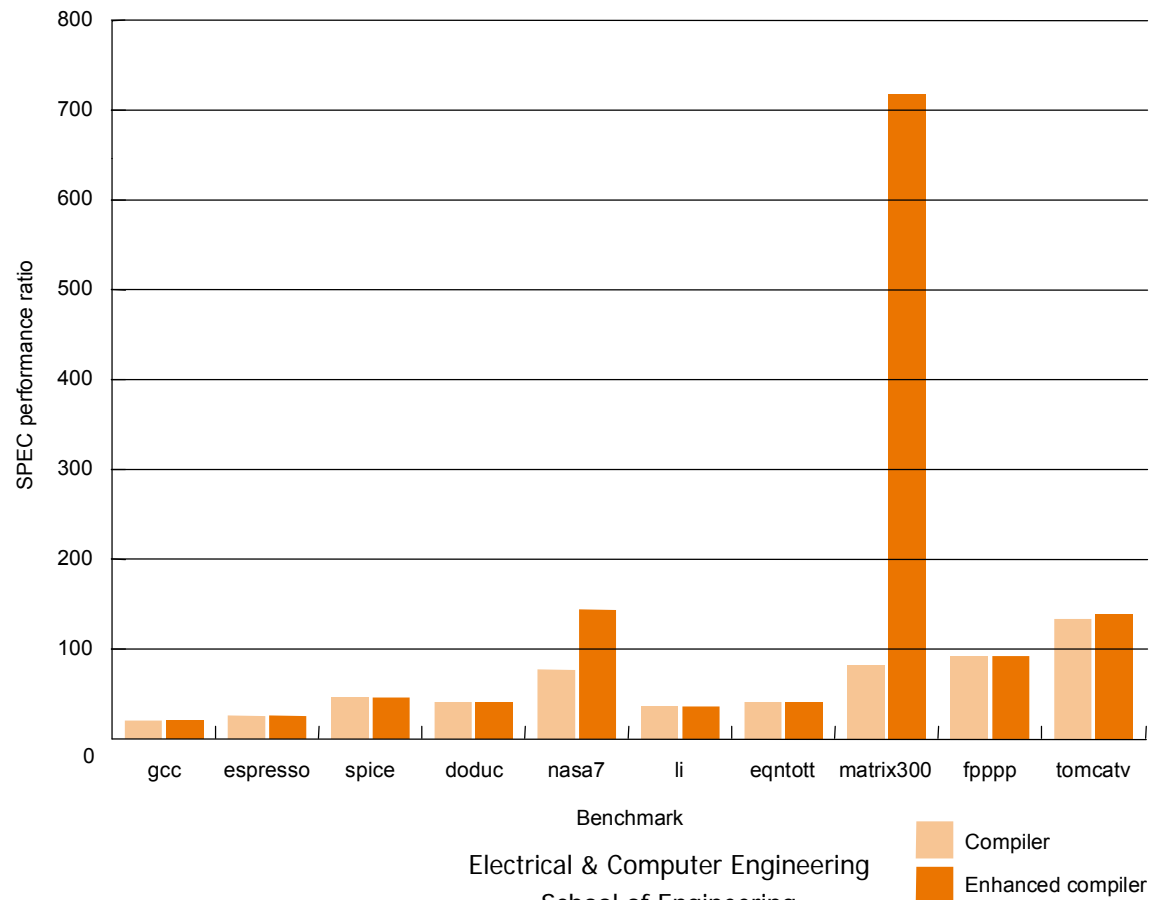
- *The company had also said that it did not like the practice but felt to compelled to make the optimizations because its competitors were doing the same thing...At the heart of Intel's problem is the practice of "tuning" compiler programs to recognize certain computing problems in the test and then substituting special handwritten pieces of code...*

Benchmark Games

- *Saturday, January 6, 1996 New York Times*

SPEC '89

- Compiler "enhancements" and performance

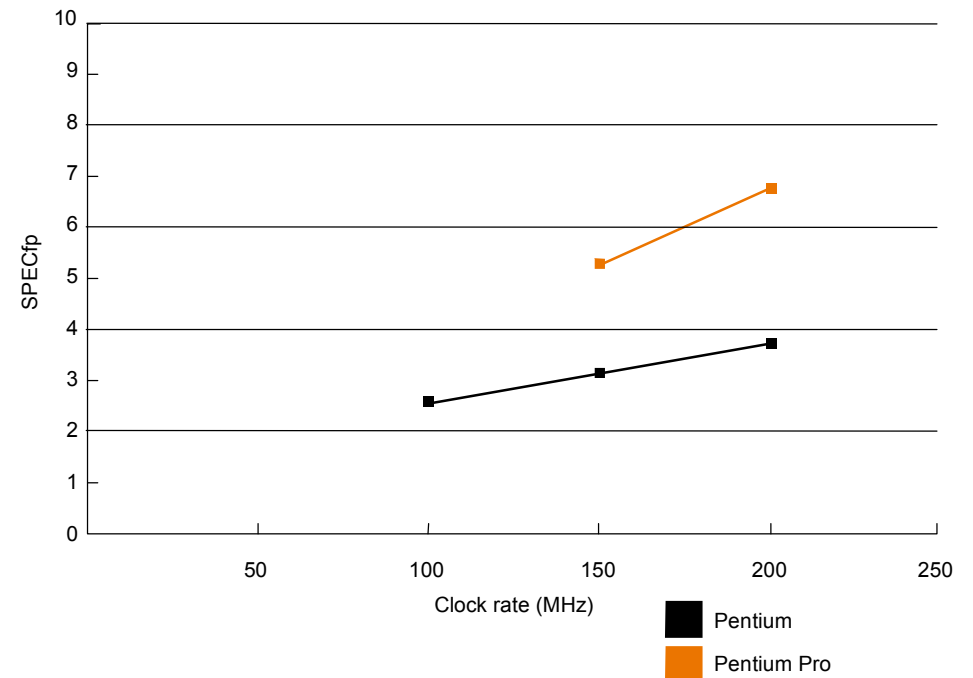
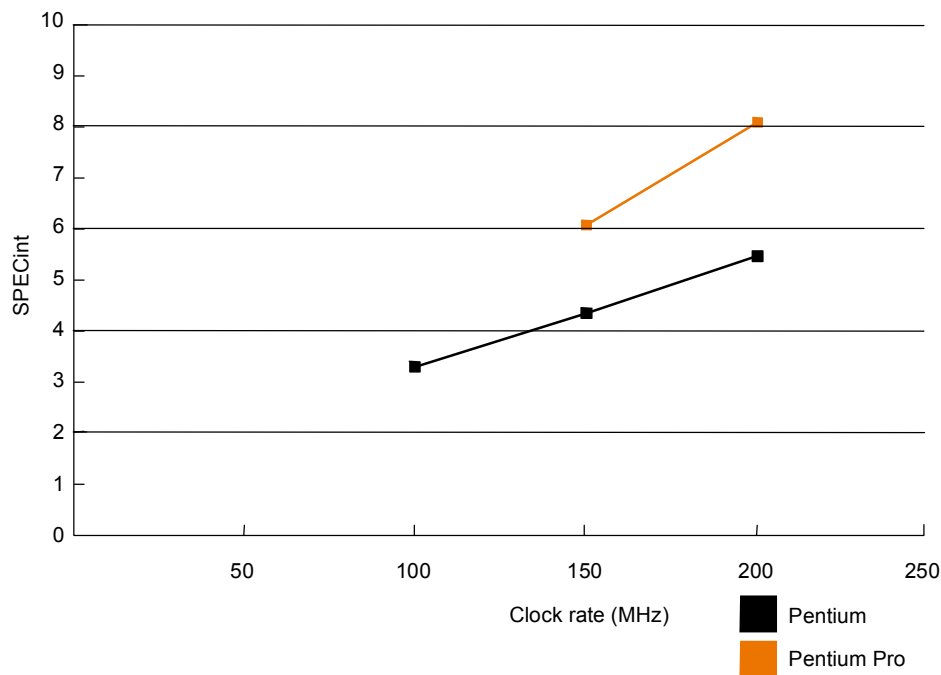


SPEC '95

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
ljpeg	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 Navier 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

SPEC '95

Does doubling the clock rate double the performance?
Can a machine with a slower clock rate have better performance?



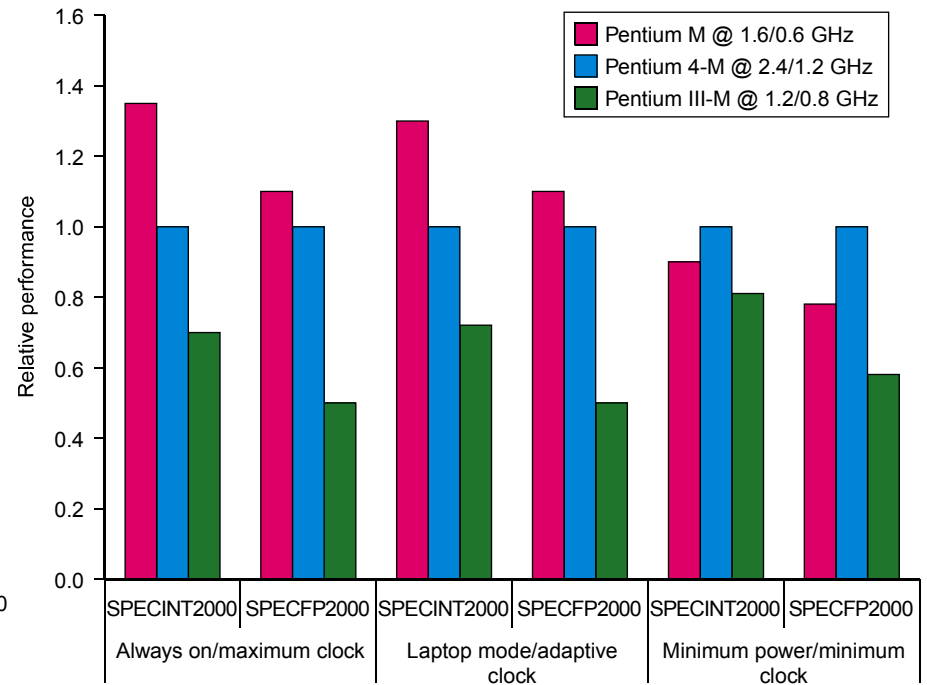
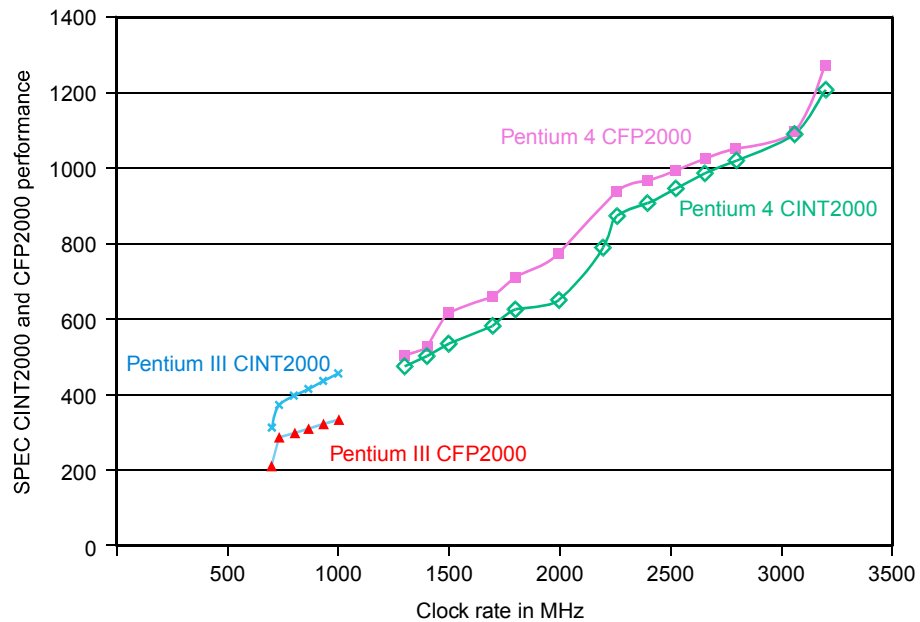
SPEC CPU2000

Integer benchmarks		FP benchmarks	
Name	Description	Name	Type
gzip	Compression	wupwise	Quantum chromodynamics
vpr	FPGA circuit placement and routing	swim	Shallow water model
gcc	The Gnu C compiler	mgrid	Multigrid solver in 3-D potential field
mcf	Combinatorial optimization	applu	Parabolic/elliptic partial differential equation
crafty	Chess program	mesa	Three-dimensional graphics library
parser	Word processing program	galgel	Computational fluid dynamics
ecn	Computer visualization	art	Image recognition using neural networks
perlbnk	perl application	equake	Seismic wave propagation simulation
gap	Group theory, Interpreter	facerec	Image recognition of faces
vortex	Object-oriented database	ampp	Computational chemistry
bzip2	Compression	lucas	Primality testing
twolf	Place and rote simulator	fma3d	Crash simulation using finite-element method
		sixtrack	High-energy nuclear physics accelerator design
		apsi	Meteorology: pollutant distribution

FIGURE 4.5 The SPEC CPU2000 benchmarks. The 12 integer benchmarks in the left half of the table are written in C and C++, while the floating-point benchmarks in the right half are written in Fortran (77 or 90) and C. For more information on SPEC and on the SPEC benchmarks, see www.spec.org. The SPEC CPU benchmarks use wall clock time as the metric, but because there is little I/O, they measure CPU performance.

SPEC 2000

Does doubling the clock rate double the performance?
Can a machine with a slower clock rate have better performance?



SPEC CPU Benchmark

- Programs used to measure performance
 - Supposedly typical of actual workload
- Standard Performance Evaluation Corp (SPEC)
 - Develops benchmarks for CPU, I/O, Web, ...
- 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)

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

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

High cache miss rates



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

Experiment

- What kind of response are you likely to get?
- What kind of response could you give a friend with the same question?

Performance Summary

■ The BIG Picture

$$\text{CPU Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

- Performance depends on
 - Algorithm: affects IC, possibly CPI
 - Programming language: affects IC, CPI
 - Compiler: affects IC, CPI
 - Instruction set architecture: affects IC, CPI, T_c


Amdahl's Law

Execution Time After Improvement =

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

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?" 

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?

Example

$$I = 5 \quad E_B = 10 \text{ s} \quad E_{BFP} = 5 \text{ s} \quad E_{BO} = 5 \text{ s} \quad n = ?$$

$$E_A = E_{BO} + \frac{E_{BFP}}{I}$$

$$n = \frac{E_B}{E_A} = \frac{E_B}{E_{BO} + \frac{E_{BFP}}{I}} = \frac{10 \text{ s}}{5 \text{ s} + \frac{5 \text{ s}}{5}} = 1.\bar{6} \approx 67\%$$

Example

- 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?

Example

$$I = 5 \quad \frac{E_B}{E_A} = 3 \quad E_B = 100 \text{ s} \quad E_{BFP} = ? \quad E_{AFP} = ?$$

$$E_A = E_{BO} + \frac{E_{BFP}}{I} \quad \frac{E_B}{3} = (E_B - E_{BFP}) + \frac{E_{BFP}}{I}$$

$$E_{BFP} = \frac{\frac{E_B}{3} - E_B}{-1 + \frac{1}{I}} = \frac{\frac{100 \text{ s}}{3} - 100 \text{ s}}{-1 + \frac{1}{5}} = 83.\bar{3}$$

Example

$$I = 5 \quad \frac{E_B}{E_A} = 3 \quad E_B = 100 \text{ s} \quad E_{BFP} = ? \quad E_{AFP} = ?$$

$$E_{BO} = E_B - E_{BFP} = 100 \text{ s} - 83.\bar{3} = 16.\bar{6}$$

$$E_{AFP} = E_A - E_{AO} = \frac{E_B}{3} - E_{BO} = \frac{100 \text{ s}}{3} - 16.\bar{6} \text{ s}$$

$$E_{AFP} = 16.\bar{6} \text{ s}$$

Example (easier way ...)

$$I = 5 \quad \frac{E_B}{E_A} = 3 \quad E_B = 100 \text{ s} \quad E_{BFP} = ? \quad E_{AFP} = ?$$

$$E_{AFP} = \frac{E_{BFP}}{I} = \frac{83.\bar{3} \text{ s}}{5} = 16.\bar{6} \text{ s}$$

Remember

- Performance is specific to a particular program/s
 - Total execution time is a consistent summary of performance
 - How much do you use each program?

Remember

- For a given architecture performance increases come from:
 - increases in clock rate (without adverse CPI affects)
 - improvements in processor organization that lower CPI

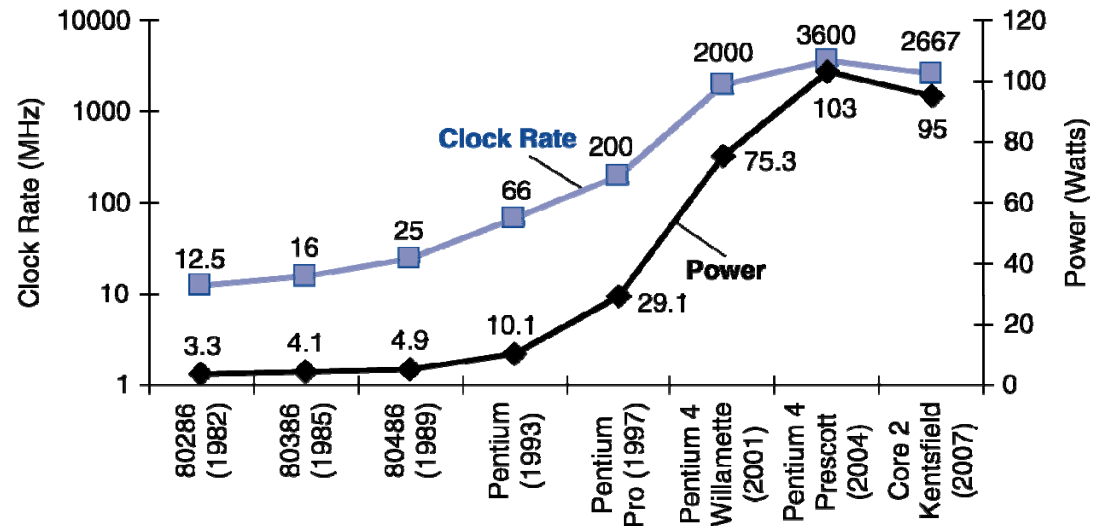
Remember

- For a given architecture performance increases come from:
 - compiler enhancements that lower CPI and/or instruction count
 - Algorithm/Language choices that affect instruction count

Remember

- 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!

Power Trends



- In CMOS IC technology

$$\text{Power} = \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency}$$

× 30

5V → 1V

× 1000

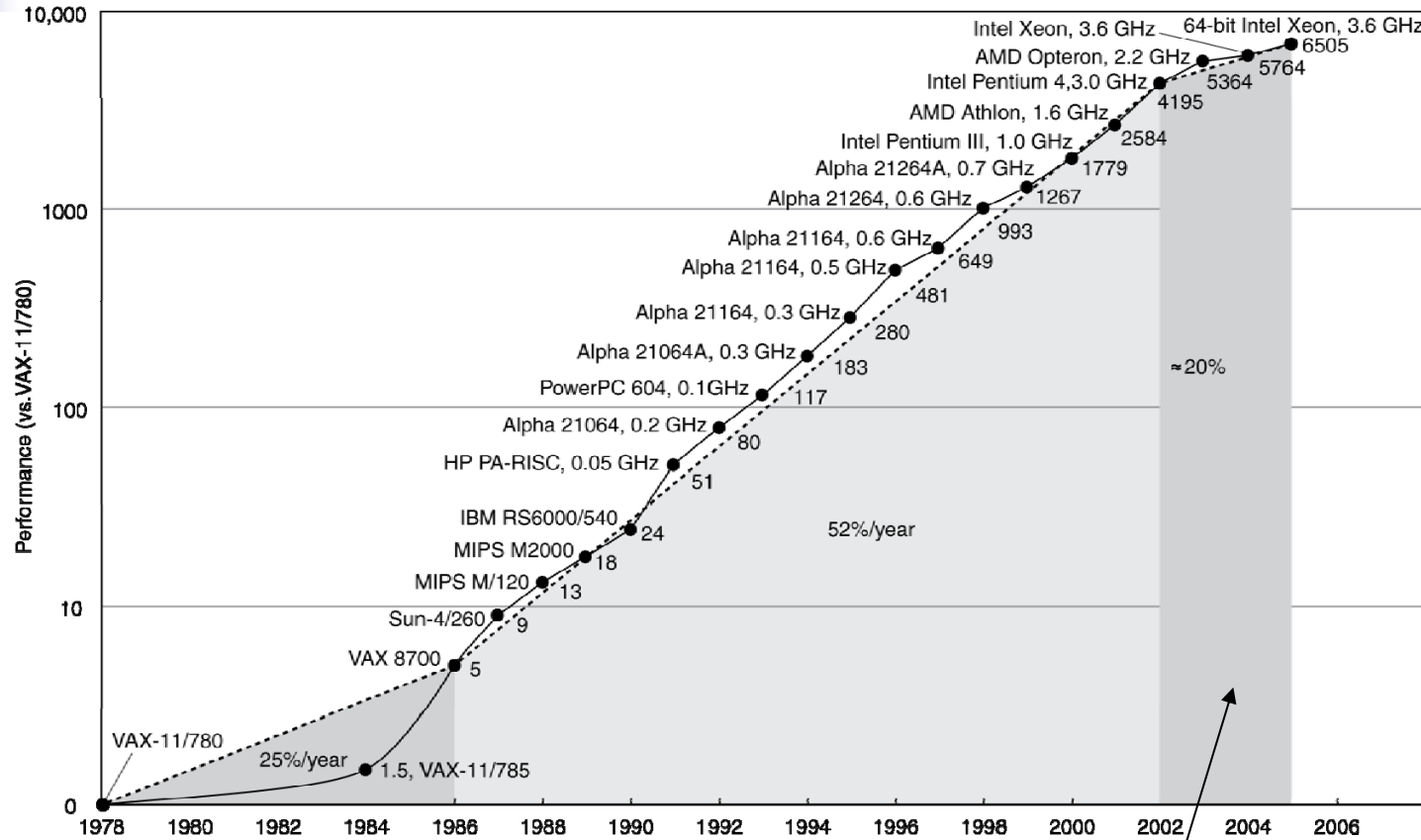
Reducing Power

- Suppose a new CPU has
 - 85% of capacitive load of old CPU
 - 15% voltage and 15% frequency reduction

$$\frac{P_{\text{new}}}{P_{\text{old}}} = \frac{C_{\text{old}} \times 0.85 \times (V_{\text{old}} \times 0.85)^2 \times F_{\text{old}} \times 0.85}{C_{\text{old}} \times V_{\text{old}}^2 \times F_{\text{old}}} = 0.85^4 = 0.52$$

- The power wall
 - We can't reduce voltage further
 - We can't remove more heat
- How else can we improve performance?

Uniprocessor Performance

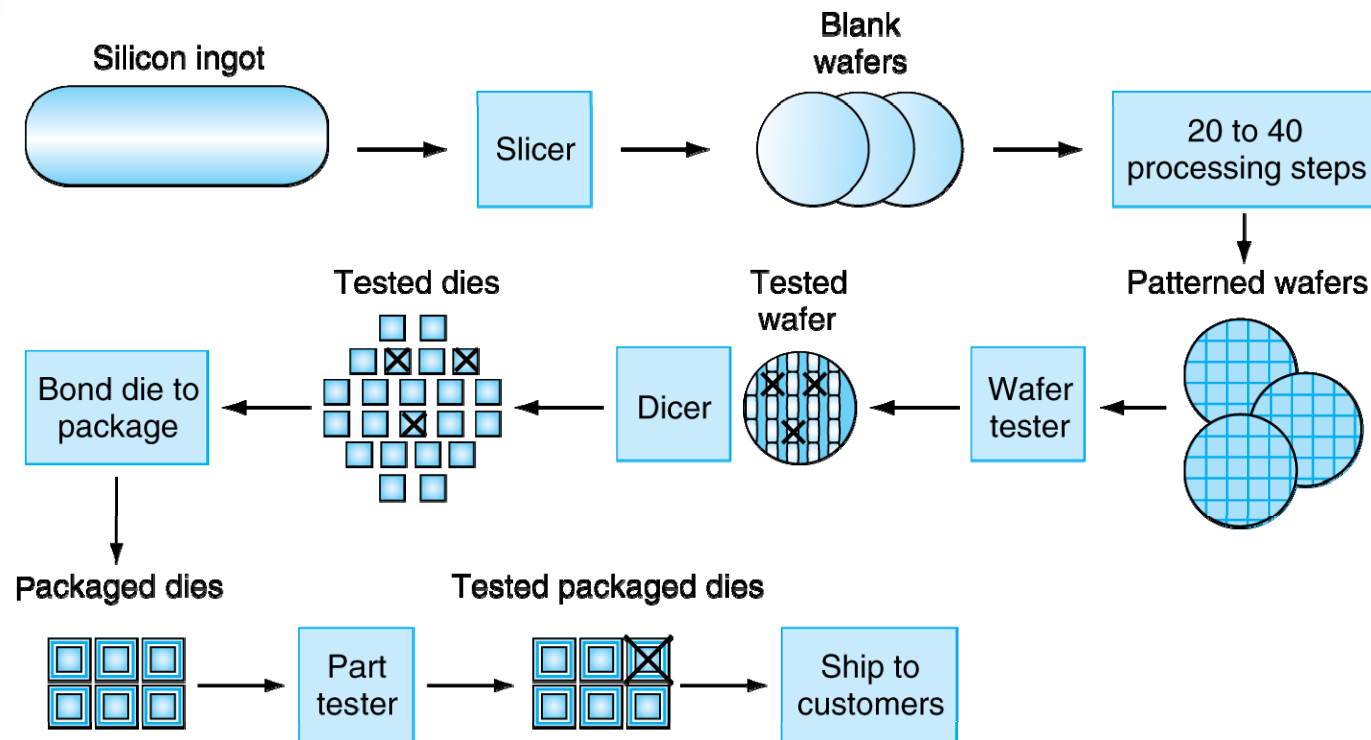


Constrained by power, instruction-level parallelism, memory latency

Multiprocessors

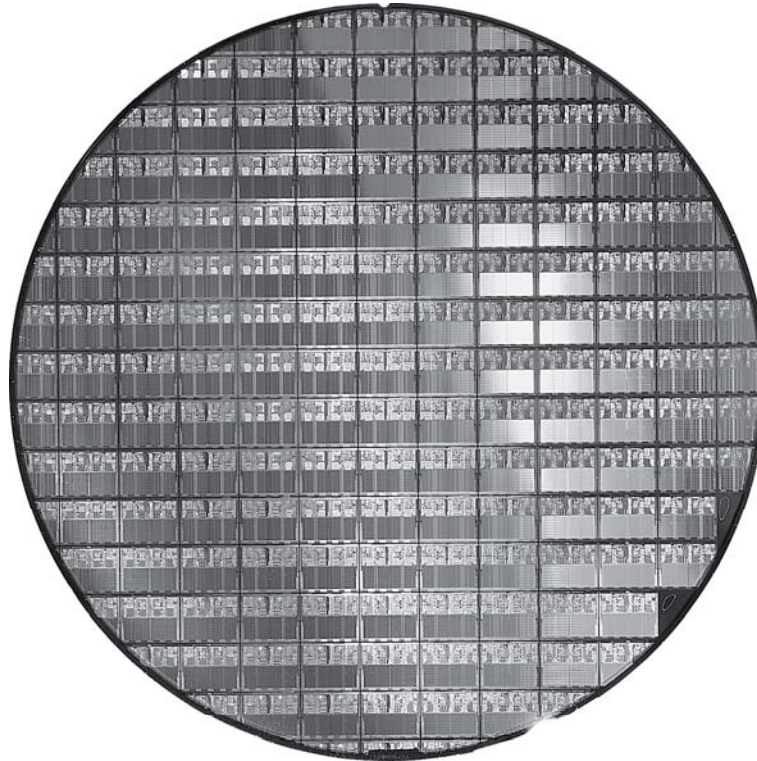
- Multicore microprocessors
 - More than one processor per chip
- Requires explicitly parallel programming
 - Compare with instruction level parallelism
 - Hardware executes multiple instructions at once
 - Hidden from the programmer
 - Hard to do
 - Programming for performance
 - Load balancing
 - Optimizing communication and synchronization

Manufacturing ICs



- Yield: proportion of working dies per wafer

AMD Opteron X2 Wafer



- X2: 300mm wafer, 117 chips, 90nm technology
- X4: 45nm technology

Integrated Circuit Cost

$$\text{Cost per die} = \frac{\text{Cost per wafer}}{\text{Dies per wafer} \times \text{Yield}}$$

$$\text{Dies per wafer} \approx \text{Wafer area} / \text{Die area}$$

$$\text{Yield} = \frac{1}{(1 + (\text{Defects per area} \times \text{Die area} / 2))^2}$$

- Nonlinear relation to area and defect rate
 - Wafer cost and area are fixed
 - Defect rate determined by manufacturing process
 - Die area determined by architecture and circuit design

Pitfall: Amdahl's Law

- Improving an aspect of a computer and expecting a proportional improvement in overall performance

$$T_{\text{improved}} = \frac{T_{\text{affected}}}{\text{improvement factor}} + T_{\text{unaffected}}$$

- Example: multiply accounts for 80s/100s
 - How much improvement in multiply performance to get 5× overall?

$$20 = \frac{80}{n} + 20 \quad \blacksquare \text{ Can't be done!}$$

- Corollary: Make the common case fast!

Fallacy: Low Power at Idle

- Look back at X4 power benchmark
 - At 100% load: 295W
 - At 50% load: 246W (83%)
 - At 10% load: 180W (61%)
- Google data center
 - Mostly operates at 10% – 50% load
 - At 100% load less than 1% of the time
- Consider designing processors to make power proportional to load

Pitfall: MIPS as a Performance Metric

- MIPS: Millions of Instructions Per Second
 - Doesn't account for
 - Differences in ISAs between computers
 - Differences in complexity between instructions

$$\begin{aligned}
 \text{MIPS} &= \frac{\text{Instruction count}}{\text{Execution time} \times 10^6} \\
 &= \frac{\text{Instruction count}}{\frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}} \times 10^6} = \frac{\text{Clock rate}}{\text{CPI} \times 10^6}
 \end{aligned}$$

- CPI varies between programs on a given CPU

Concluding Remarks

- Cost/performance is improving
 - Due to underlying technology development
- Hierarchical layers of abstraction
 - In both hardware and software
- Instruction set architecture
 - The hardware/software interface
- Execution time: the best performance measure
- Power is a limiting factor
 - Use parallelism to improve performance

Where we are headed

- Verilog or VHDL
 - The languages of computer design
- A specific instruction set architecture (Chapter 2)
- Arithmetic and how to build an ALU (Chapter 3)

Where we are headed

- Constructing a processor to execute our instructions ... AND ...
- Pipelining to improve performance (Chapter 4)
- Memory: caches and virtual memory (Chapter 5)
- I/O (Chapter 6)
- Multiprocessors (Chapter 7)

Where we are headed

Key to a good grade:
reading the book!
doing the homework!
asking questions!
not procrastinating!

Historical Perspective

- ENIAC built in World War II was the first general purpose computer
 - Used for computing artillery firing tables
 - 80 feet long by 8.5 feet high and several feet wide
 - Each of the twenty 10 digit registers was 2 feet long
 - Used 18,000 vacuum tubes
 - Performed 1900 additions per second



–Since then:

Moore's Law:

transistor capacity doubles every 18-24 months