

# Chapter 1

## Computer Abstractions and Technology

# 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

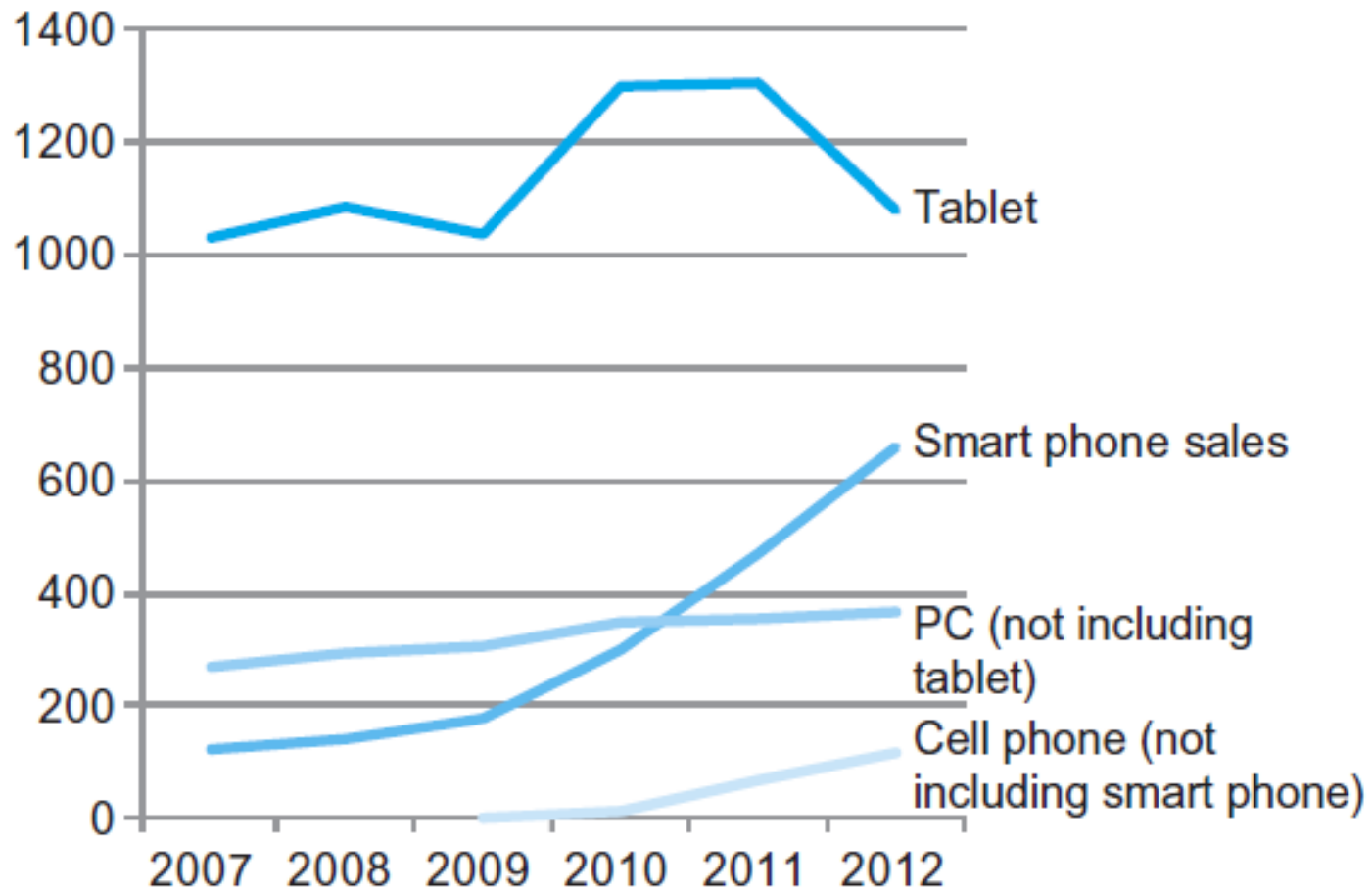
# Classes of Computers

- Personal 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

# Classes of Computers

- Supercomputers
  - High-end scientific and engineering calculations
  - Highest capability but represent a small fraction of the overall computer market
- Embedded computers
  - Hidden as components of systems
  - Stringent power/performance/cost constraints

# The PostPC Era



**Number (in Millions) manufactured per year of tablets and smart phones**

# The PostPC Era

- Personal Mobile Device (PMD)
  - Battery operated
  - Connects to the Internet
  - Hundreds of dollars
  - Smart phones, tablets, electronic glasses
- Cloud computing
  - Warehouse Scale Computers (WSC)
  - Software as a Service (SaaS)
  - Portion of software run on a PMD and a portion run in the Cloud
  - Amazon and Google

# 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

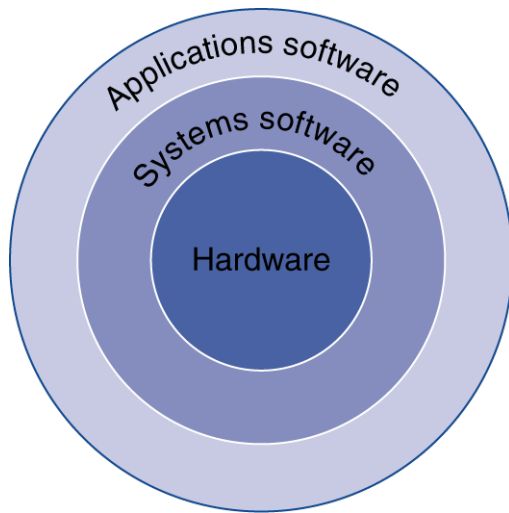


# Eight Great Ideas

- Design for *Moore's Law*
- Use *abstraction* to simplify design
- Make the *common case fast*
- Performance via *parallelism*
- Performance via *pipelining*
- Performance via *prediction*
- *Hierarchy* of memories
- *Dependability* via redundancy



# 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;
}
```

Compiler

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
```

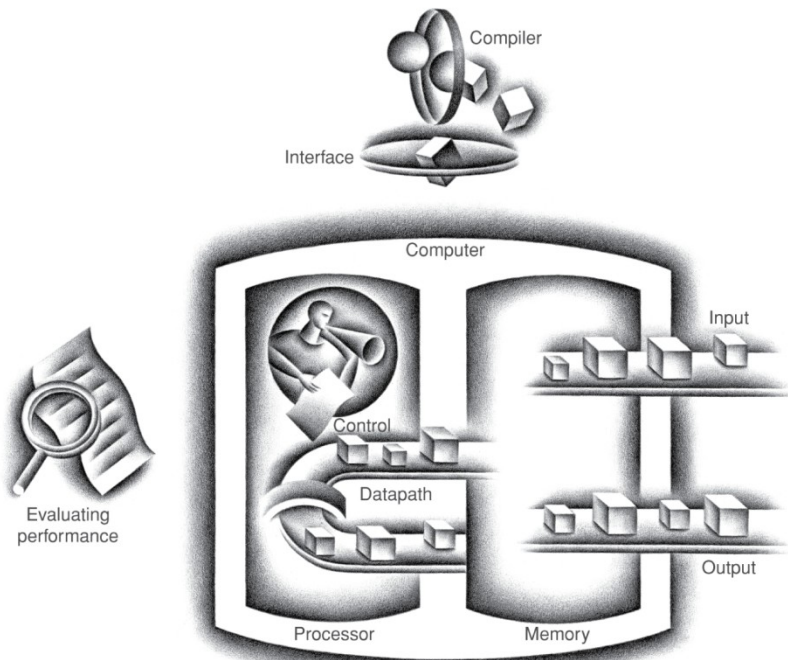
Assembler

Binary machine  
language  
program  
(for MIPS)

```
000000001010000100000000000011000
000000000000110000001100000100001
100011000110001000000000000000000
100011001111001000000000000000100
101011001111001000000000000000000
101011000110001000000000000000100
00000011111000000000000000001000
```

# 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

# Inside the Processor (CPU)

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

# 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

# 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...

# Relative Performance

- Define Performance =  $1/\text{Execution Time}$
- “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
  - $\text{Execution Time}_B / \text{Execution Time}_A$   
 $= 15\text{s} / 10\text{s} = 1.5$
  - 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?

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

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

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

# 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 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?

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

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

$$\frac{\text{CPU Time}_B}{\text{CPU Time}_A} = \frac{1 \times 600\text{ps}}{1 \times 500\text{ps}} = 1.2 \leftarrow \text{...by this much}$$

# 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

- 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  
 $= 2 \times 1 + 1 \times 2 + 2 \times 3$   
 $= 10$
  - Avg. CPI =  $10/5 = 2.0$
- Sequence 2: IC = 6
  - Clock Cycles  
 $= 4 \times 1 + 1 \times 2 + 1 \times 3$   
 $= 9$
  - Avg. CPI =  $9/6 = 1.5$



# 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,  $T_c$

# 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 Intel Core i7 920

Description	Name	Instruction Count x 10 <sup>9</sup>	CPI	Clock cycle time (seconds x 10 <sup>-9</sup> )	Execution Time (seconds)	Reference Time (seconds)	SPECratio
Interpreted string processing	perl	2252	0.60	0.376	508	9770	19.2
Block-sorting compression	bzip2	2390	0.70	0.376	629	9650	15.4
GNU C compiler	gcc	794	1.20	0.376	358	8050	22.5
Combinatorial optimization	mcf	221	2.66	0.376	221	9120	41.2
Go game (AI)	go	1274	1.10	0.376	527	10490	19.9
Search gene sequence	hmmer	2616	0.60	0.376	590	9330	15.8
Chess game (AI)	sjeng	1948	0.80	0.376	586	12100	20.7
Quantum computer simulation	libquantum	659	0.44	0.376	109	20720	190.0
Video compression	h264avc	3793	0.50	0.376	713	22130	31.0
Discrete event simulation library	omnetpp	367	2.10	0.376	290	6250	21.5
Games/path finding	astar	1250	1.00	0.376	470	7020	14.9
XML parsing	xalancbmk	1045	0.70	0.376	275	6900	25.1
Geometric mean	–	–	–	–	–	–	25.7

# 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

# 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

# Sections to Read from the Book

- 5<sup>th</sup> Edition Sections to read
  - 1.1
  - 1.2
  - 1.3
  - 1.4
  - 1.6
  - 1.9
  - 1.10