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

- 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

The PostPC Era

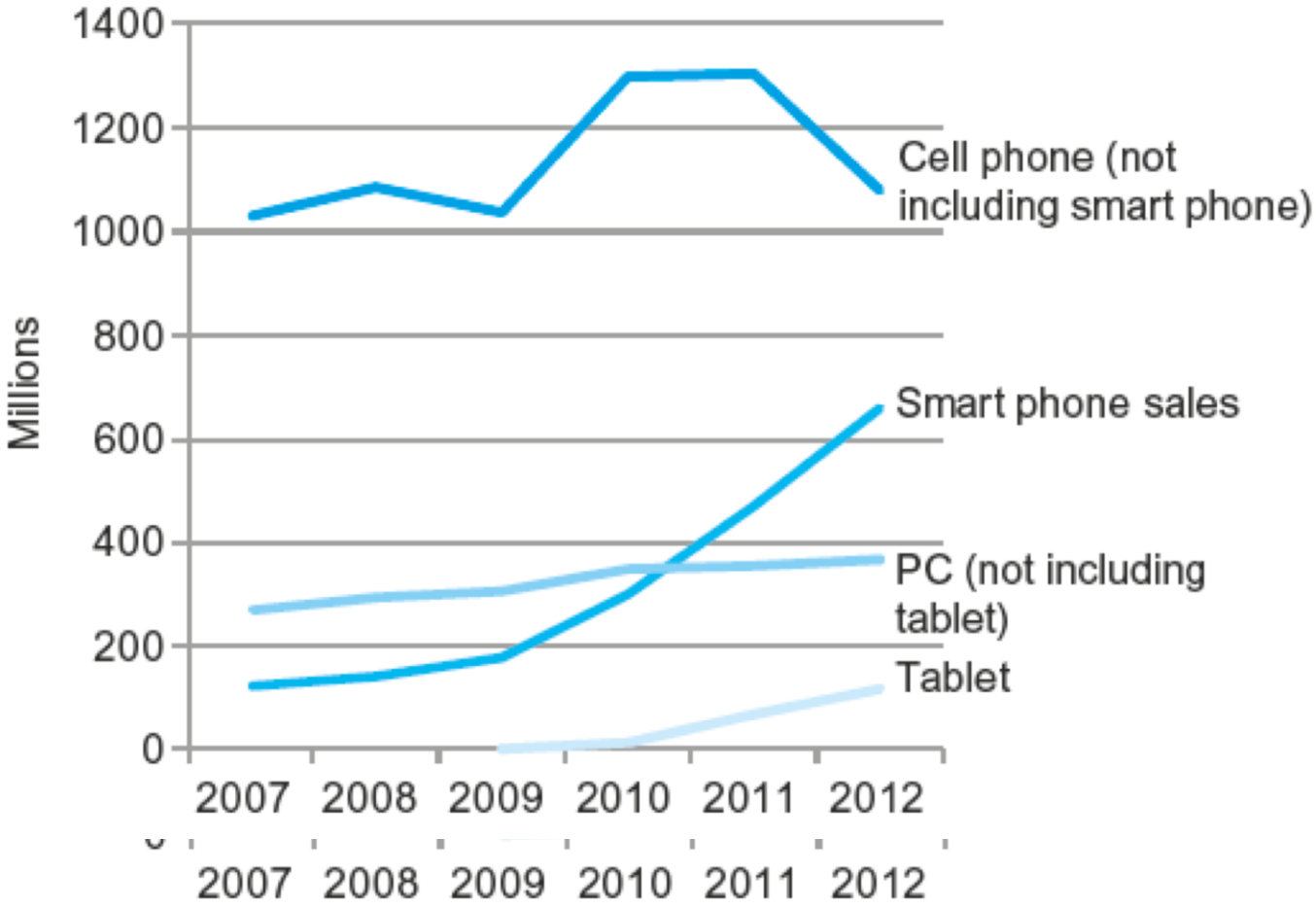


FIGURE 1.2 The number manufactured per year of tablets and smart phones, which reflect the PostPC era, versus personal computers and traditional cell phones. Smart phones represent the recent growth in the cell phone industry, and they passed PCs in 2011. Tablets are the fastest growing category, nearly doubling between 2011 and 2012. Recent PCs and traditional cell phone categories are relatively flat or declining.

By the time you complete this book you will be able to answer the following questions:

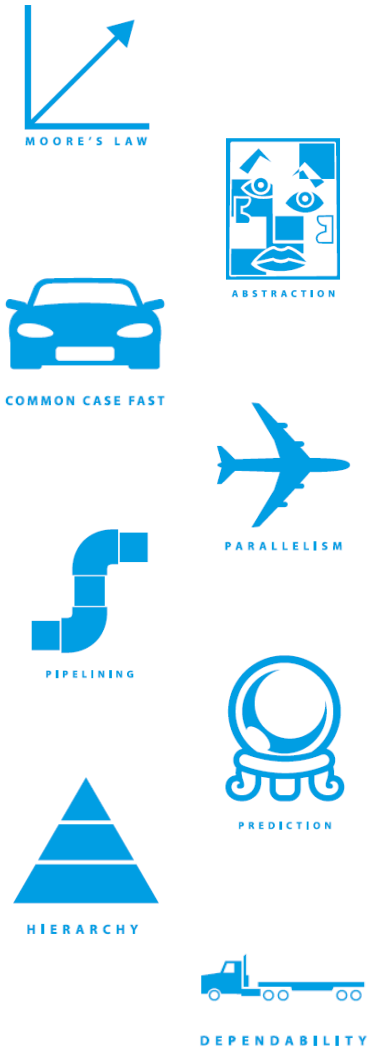
- high-level **language** → the language of the hardware,
- how does the **hardware execute** the resulting program?
- understanding the aspects of both the hardware and software that **affect program performance**.
- The software/ hardware **interface**, and how does software instruct the hardware to **perform** needed **functions**?
- What **determines the performance** of a program, and how can a programmer improve the performance?
- What **techniques** can be used by hardware **designers** to improve **performance**?
- What techniques can be used by hardware designers to improve **energy efficiency**? What can the programmer do to help or hinder energy efficiency?
- What are the reasons for and the consequences of the recent switch from sequential processing to **parallel processing**?

A Big Picture

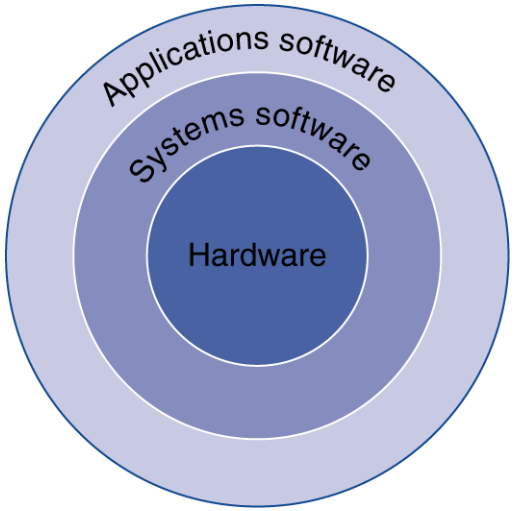
Hardware or software component	How this component affects performance	Where is this topic covered?
Algorithm	Determines both the number of source-level statements and the number of I/O operations executed	Other books!
Programming language, compiler, and architecture	Determines the number of computer instructions for each source-level statement	Chapters 2 and 3
Processor and memory system	Determines how fast instructions can be executed	Chapters 4, 5, and 6
I/O system (hardware and operating system)	Determines how fast I/O operations may be executed	Chapters 4, 5, and 6

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



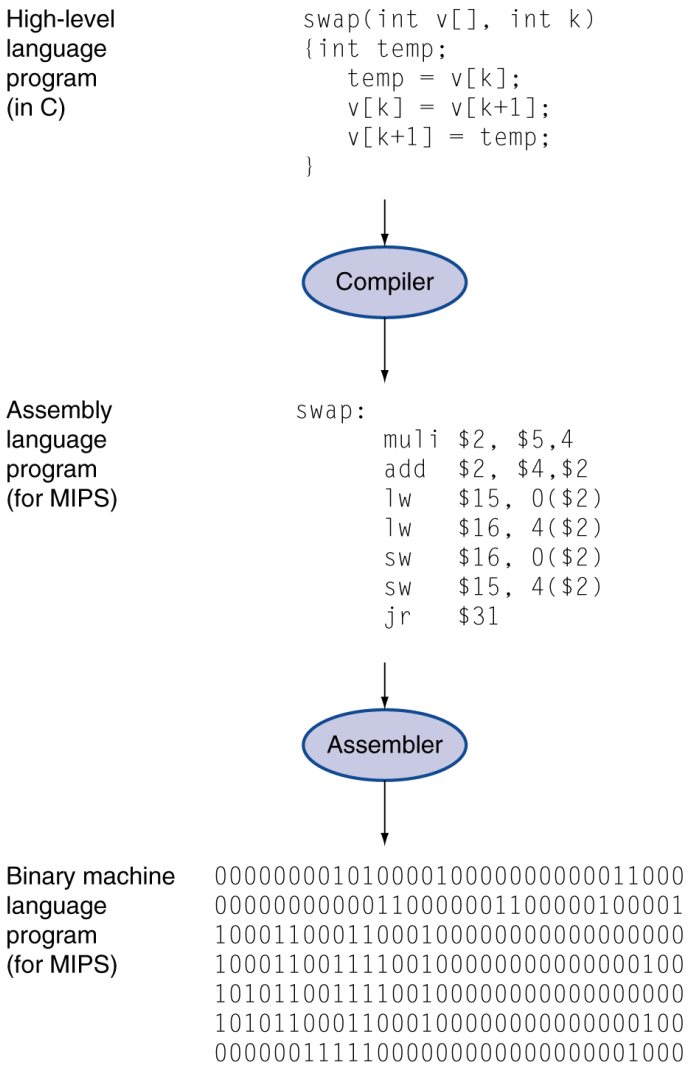
Below Your Program



- 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

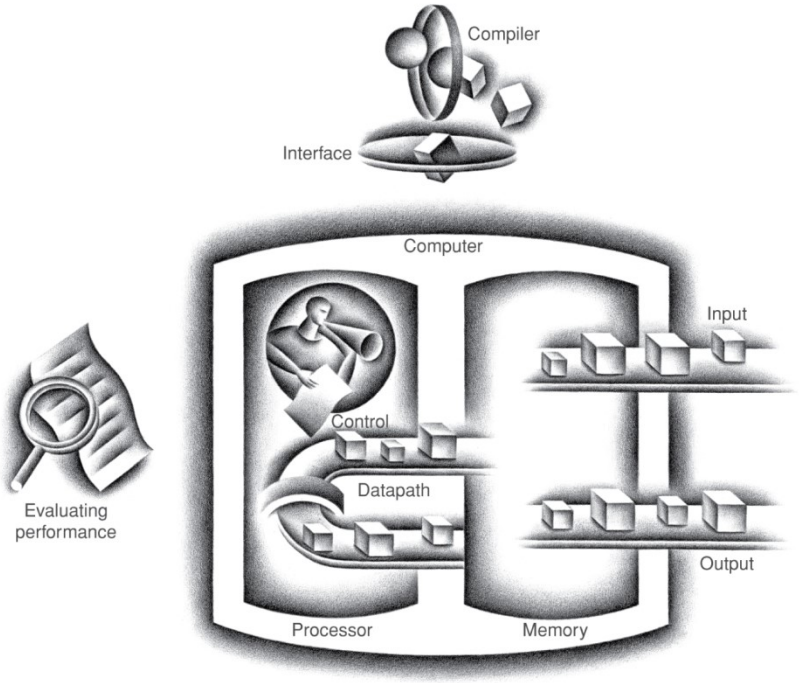
Levels of Program Code

- 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



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

Through the Looking Glass

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

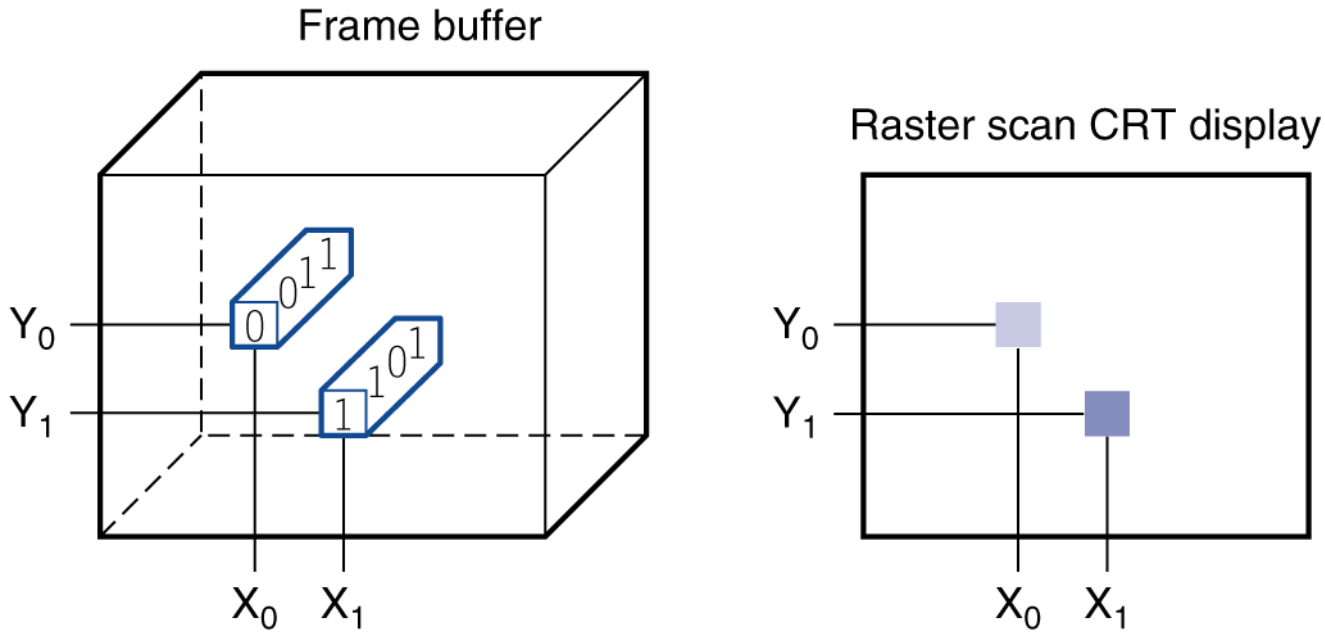
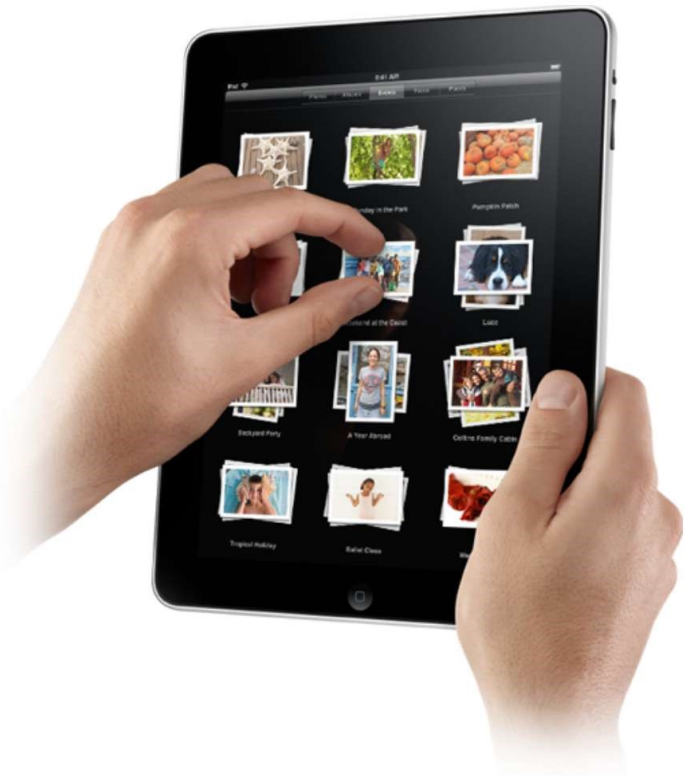


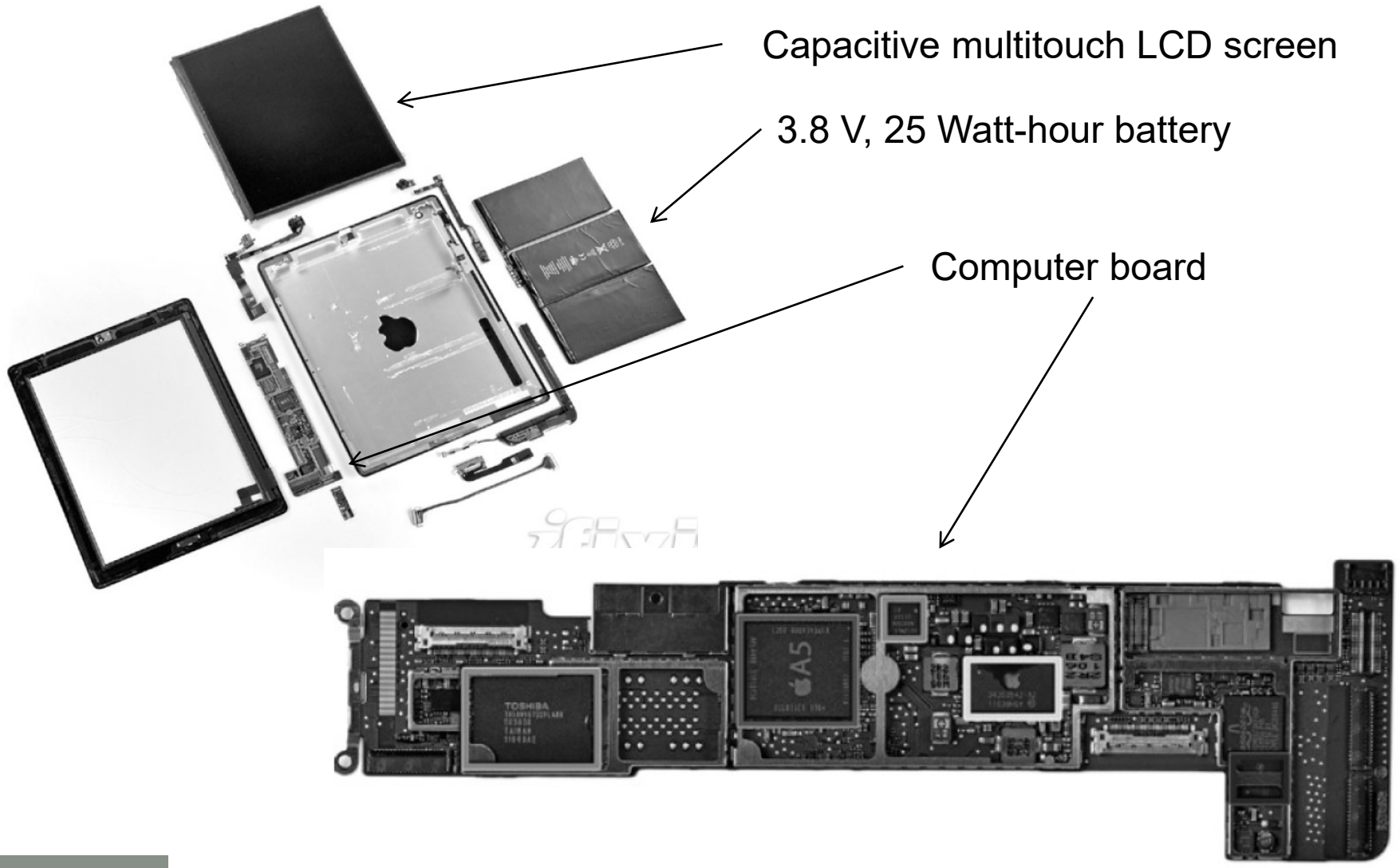
FIGURE 1.6 Each coordinate in the frame buffer on the left determines the shade of the corresponding coordinate for the raster scan CRT display on the right. Pixel (X_0, Y_0) contains the bit pattern 0011, which is a lighter shade on the screen than the bit pattern 1101 in pixel (X_1, Y_1).

Touchscreen

- PostPC device
- Supersedes keyboard and mouse
- Resistive and Capacitive types
  - Most tablets, smart phones use capacitive
  - Capacitive allows multiple touches simultaneously



Opening the Box

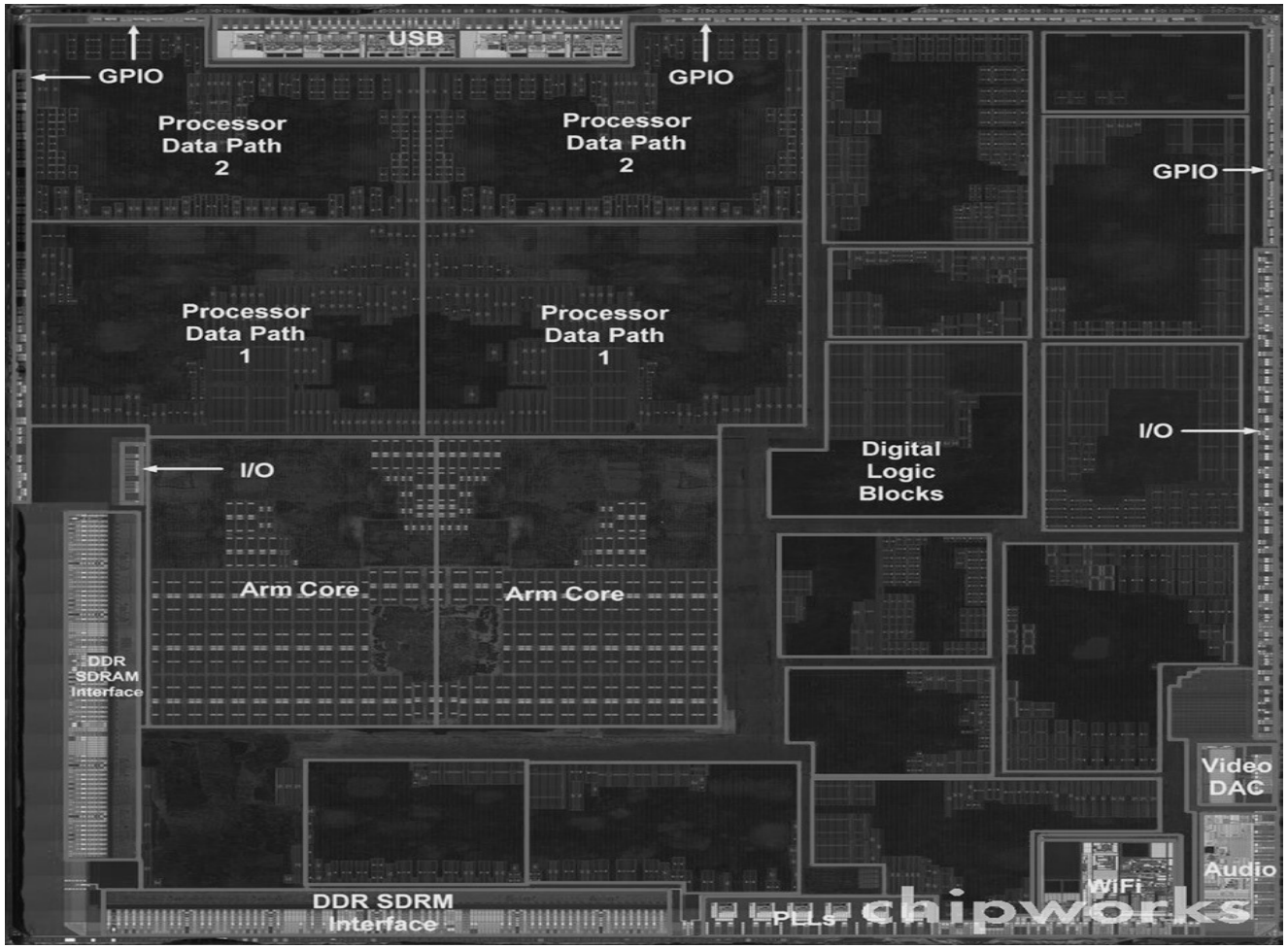


Inside the Processor (CPU)

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

Inside the Processor

- Apple A5



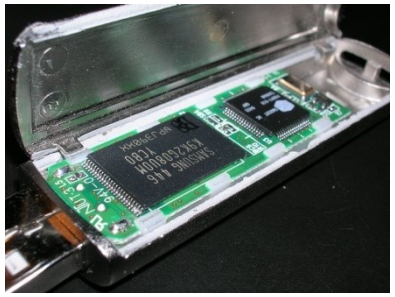
Abstractions

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

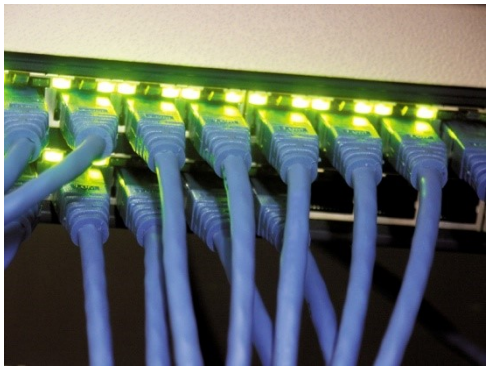
A Safe Place for Data

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



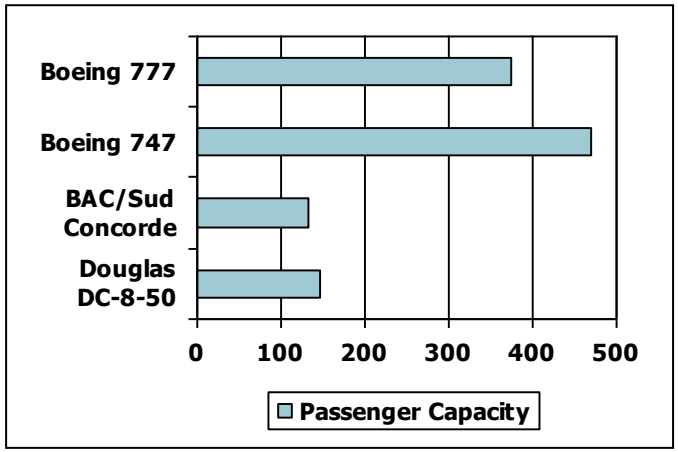
Networks

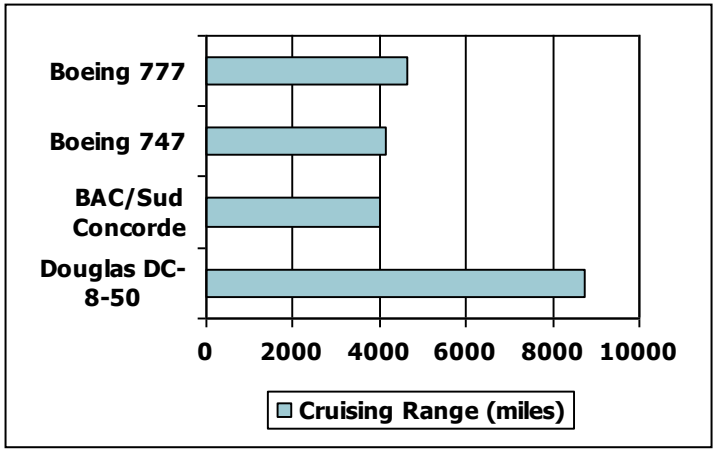
- Communication, resource sharing, nonlocal access
- Local area network (LAN): Ethernet
- Wide area network (WAN): the Internet
- Wireless network: WiFi, Bluetooth

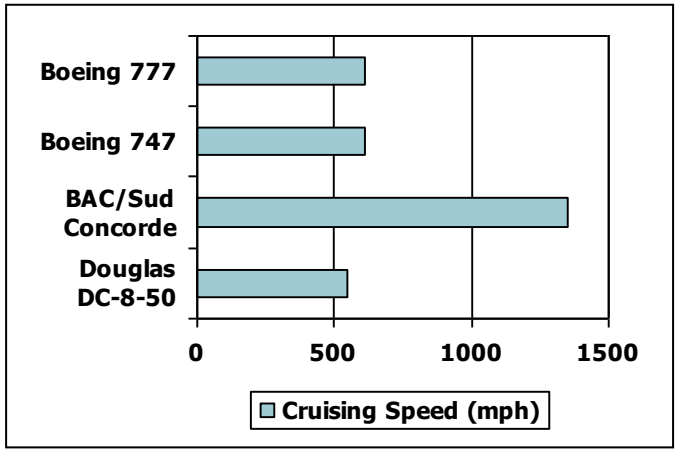


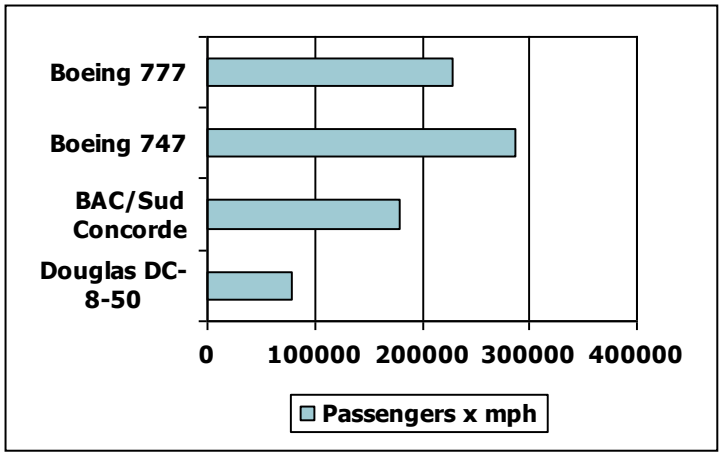
Defining Performance

- Which airplane has the best performance?









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

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

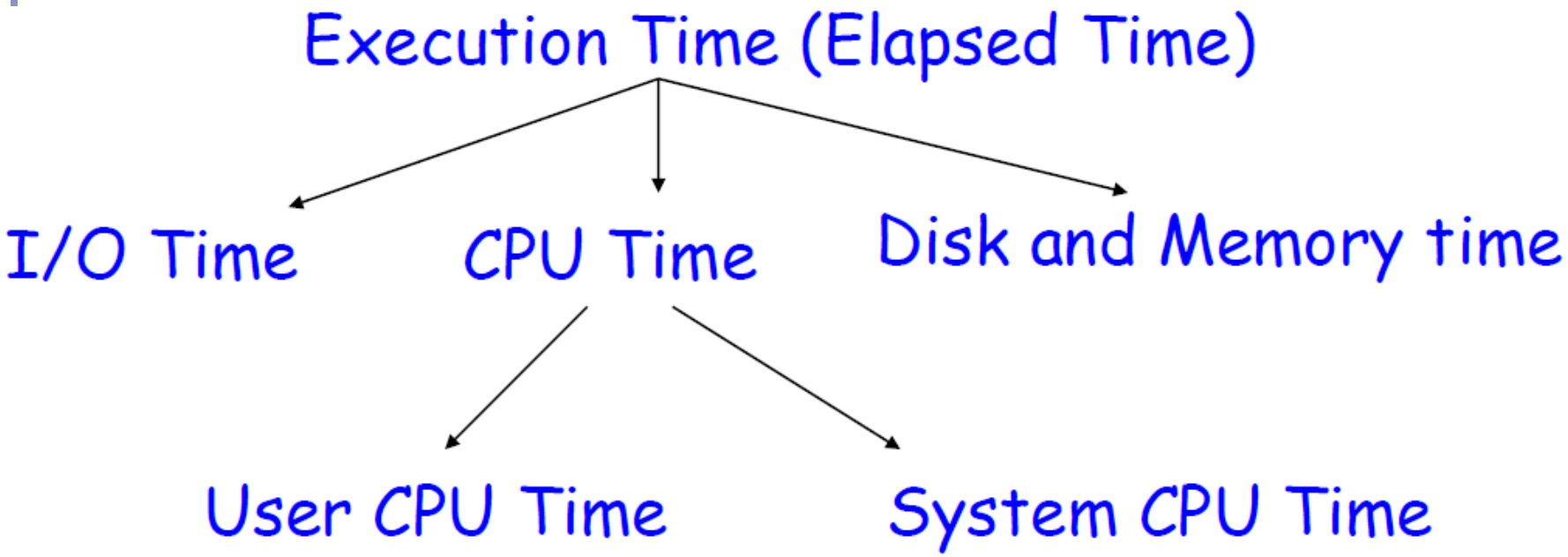
- Define Performance = 1/Execution Time
- "X is n times faster than Y"

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

- Example: time taken to run a program
  - 10s on A, 15s on B
  - Execution Time_B / Execution Time_A = 15s / 10s = 1.5
  - So A is 1.5 times faster than B

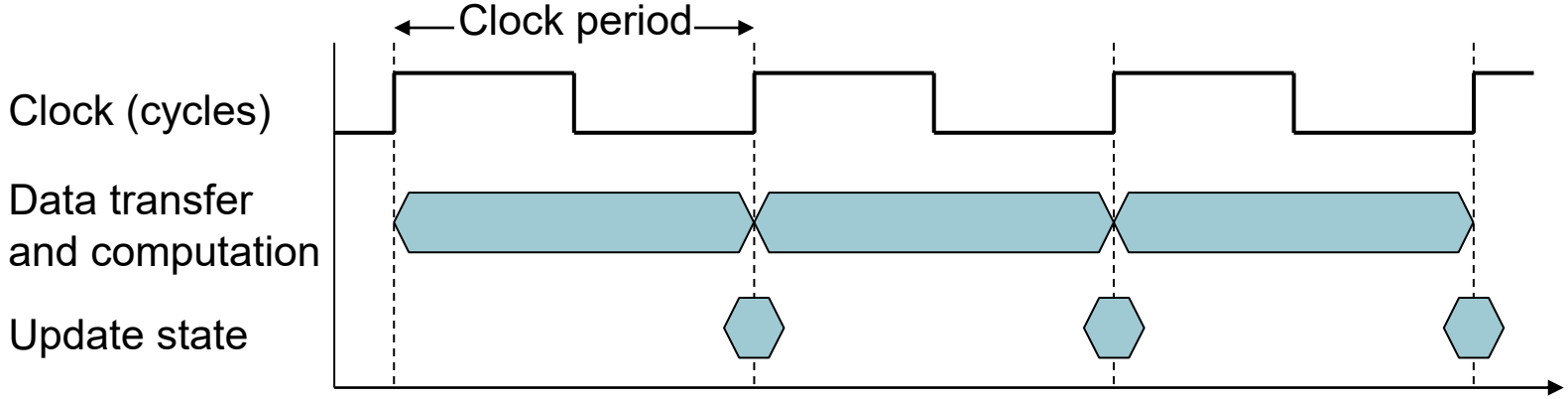
Measuring Execution Time

- 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 cycle time/period (T):**
  - time for a complete clock cycle
  - time between ticks
  - seconds per cycle
- **Clock rate/frequency (R):**
  - the inverse of the clock period, i.e.,
  - cycles per second

$$R = \frac{1}{T}$$

- 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

CPU Time

$$\text{CPU Time} = \text{CPU Clock Cycles} \times \text{Clock Cycle Time}$$

$$= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}$$

- Performance improved by
  - Reducing number of clock cycles
  - Increasing clock rate

- Our favorite program runs in 10 seconds on computer A, which has a 2 GHz clock. We are trying to help a computer designer build a computer, B, which will run this program in 6 seconds.Thee designer has determined that a substantial increase in the clock rate is possible, but this increase will affect the rest of the CPU design, causing computer B to require 1.2 times as many clock cycles as computer A for this program. What clock rate should we tell the designer to target?

- To run the program in 6 seconds, B must have twice the clock rate of A.

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 × clock cycles
- How fast must Computer B clock be?

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

$$\text{Clock Cycles}_A = \text{CPU Time}_A \times \text{Clock Rate}_A$$

$$= 10\text{s} \times 2\text{GHz} = 20 \times 10^9$$

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

Instruction Count and CPI

$$\text{Clock Cycles} = \text{Instruction Count} \times \text{Cycles per Instruction}$$

$$\text{CPU Time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time}$$

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

- 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

Static and Dynamic Instruction Count

- **Static instruction count** is the number of instructions the program has
- **Dynamic instruction count** is the actual number of instructions executed by the CPU for a specific program execution
  - We usually use dynamic instruction count as if, for example, you have a loop in your program then some instructions get executed more than once
  - Also, in the presence of branches, some instructions may not be executed at all

- Suppose we have two implementations of the same instruction set architecture.
- Computer A has a clock cycle time of 250 ps and a CPI of 2.0 for some program, and computer B has a clock cycle time of 500 ps and a CPI of 1.2 for the same program.

- Which computer is faster for this program and by how much?

CPI Example

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

$$\text{CPU Time}_A = \text{Instruction Count} \times \text{CPI}_A \times \text{Cycle Time}_A$$
$$= I \times 2.0 \times 250\text{ps} = I \times 500\text{ps}$$

A is faster…

$$\text{CPU Time}_B = \text{Instruction Count} \times \text{CPI}_B \times \text{Cycle Time}_B$$
$$= I \times 1.2 \times 500\text{ps} = I \times 600\text{ps}$$

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

…by this much

CPI in More Detail

- Different instructions take different amounts of time depending on what they do:
  - **Multiplication** takes more time than **addition**
  - **Floating-point** operations take longer than **integer** ones
  - **Accessing memory** takes more time than accessing **registers**
- Instructions can be divided into classes of similar instructions
- Instructions in the same class have the same **Clock cycles Per Instruction (CPI)** value

CPI in More Detail (cont.)

- Total CPU clock cycles for a certain program can be calculated by looking at various instruction classes and their individual CPIs

- If different instruction classes take different numbers of cycles

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

- – CPIi is the clock cycles per instruction for class i (integer number),
- – Ci is the count of instructions executed from class i, and
- – n is the number of instruction classes

CPI in More Detail

- **Average CPI ($CPI_{average}$ or just CPI)** for a certain program is the average number of clock cycles each instruction takes to execute

$$CPI = \frac{CPU\ clock\ cycles\ for\ a\ program}{Instruction\ count} = \frac{CPU\ clock\ cycles\ for\ a\ program}{C}$$

thus, $$CPU\ clock\ cycles\ for\ a\ program = CPI \times C = \sum_{i=1}^{n}(CPI_i \times C_i)$$

then, $$CPI = \frac{\sum_{i=1}^{n}(CPI_i \times C_i)}{C} = \sum_{i=1}^{n}(CPI_i \times \frac{C_i}{C})$$

Relative frequency

- C is the number of instructions executed by the program (known as the **instruction count**, instruction path length, or dynamic program size)

- let the fraction of occurrence (relative frequency) of an instruction class in a program be $C_{fraction_i} = \frac{C_i}{C}$ then, $CPI = \sum_{i=1}^{n}(CPI_i \times C_{fraction_i})$

- Thus, CPI depends on the instruction mix (the dynamic frequency of instructions across the program)

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

5
6

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

Instructions Per Clock Cycle

- CPI provides one way of comparing two different implementations of the same ISA, since the number of instructions executed for a program will be the same
- Although we might expect that the minimum CPI is 1.0, some processors fetch and execute multiple instructions per clock cycle (e.g., multicore microprocessors as will be shown later)
- We could invert CPI to talk about **IPC**, or **instructions per clock cycle**

The CPU Performance Equation

- The CPU time for a program can be expressed in two ways:

$$\text{CPU execution time for a program} = \text{CPU clock cycles for a program} \times \text{Clock cycle time}$$

or

$$\text{CPU execution time for a program} = \text{CPU clock cycles for a program} \times \frac{1}{\text{Clock rate}}$$

- As $\text{CPU clock cycles for a program} = CPI \times C = \sum_{i=1}^{n} (CPI_i \times C_i)$

- Then, we could express the CPU performance equation as follows:

$$\text{CPU time} = CPI \times C \times T = \left(\sum_{i=1}^{n} (CPI_i \times C_i)\right) \times T$$

or

$$\text{CPU time} = CPI \times C \times \frac{1}{R} = \left(\sum_{i=1}^{n} (CPI_i \times C_i)\right) \times \frac{1}{R}$$

Performance Summary

The BIG Picture

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

- CPU performance is dependent upon three factors:

Factors \ Dependency	Algorithm	Programming	Compiler	ISA	Processor implementation	Technology
Average clock cycles per instruction (CPI)	√	√	√	√	√	
Instruction count (C)	√	√	√	√		
Clock cycle time (clock rate) (T or R)	X	X	X	√	√	√

- CPU time is equally dependent on these three factors: a 10% improvement in any one of them leads to 10% gain in CPU time

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

Pitfall: MIPS as a Performance Metric

- MIPS: Millions of Instructions Per Second

$$\text{MIPS} = \frac{\text{Instruction count}}{\text{Execution time} \times 10^6}$$

$$= \frac{\text{Instruction count}}{\frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}} \times 10^6} = \frac{\text{Clock rate}}{\text{CPI} \times 10^6}$$

- 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