COMPUTER ARCHITECTURE & ORGANIZATION

Dr. Randa Mohamed

FCIS-Ainshams University Spring 2021

Level2

WHY

- It’s one of the most interesting courses in the faculty.
- Give you insight on how the hardware you’re using works.
- Ties Logic Design with Assembly, Compiler, and OS, and is at the core of all these sciences.
- It’s also about programming, the best programmer is one who knows the hardware the best.

WHAT TO LEARN

- Know how your computer works
- Know how assemblers work
- Learn how to design a processor
- Learn about testing your work

BY THE TIME YOU COMPLETE THIS COURSE 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?

TEXTBOOK

"Computer Organization and Design" 5th Edition by David A. Patterson and John L. Hennesy

CHAPTERS

Chapters 1 : Computer Abstraction and technology (1 lecture)

Chapter 2: Instructions: Language of the computer (2~3 Lectures)

Chapter 4: The processor (4~5 Lectures)

Chapter 5: Exploiting memory hierarchy (1~2 Lectures)

ASSESSMENT / GRADING

Item	Percentage
Quizzes/Assignments	5 Marks
Practical Hands-on	10 Marks
Midterm	15 Marks
Practical Exam	20 Marks
Final Exam	50 Marks

LOGISTICS

- Attendance in lectures & labs is mandatory (your attendance week)
- Studying the videos is mandatory (your home week)
- Late assignments/ Hands-on are not allowed
- Cheating in anyway is taken seriously
- Excuses for absence in exams should be officially approved in advance

COMMUNICATION

Lectures, labs and class announcements will be published on LMS.

Contact e-mail: randa_aboelfatoh@cis.asu.edu.eg

Let’s Start

Chapter 1

Computer Abstractions and Technology

Dr. Heba Khaled
Edited by: Dr. Randa Mohamed

Chapter 1

- Introduction to Computer Architecture (1.1,1.2)
- Below Your Program (Interface between SW and HW) (1.3,1.4)
- Computer Performance (1.6)
- Other sections are for your own knowledge and not included in exam.

Introduction

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

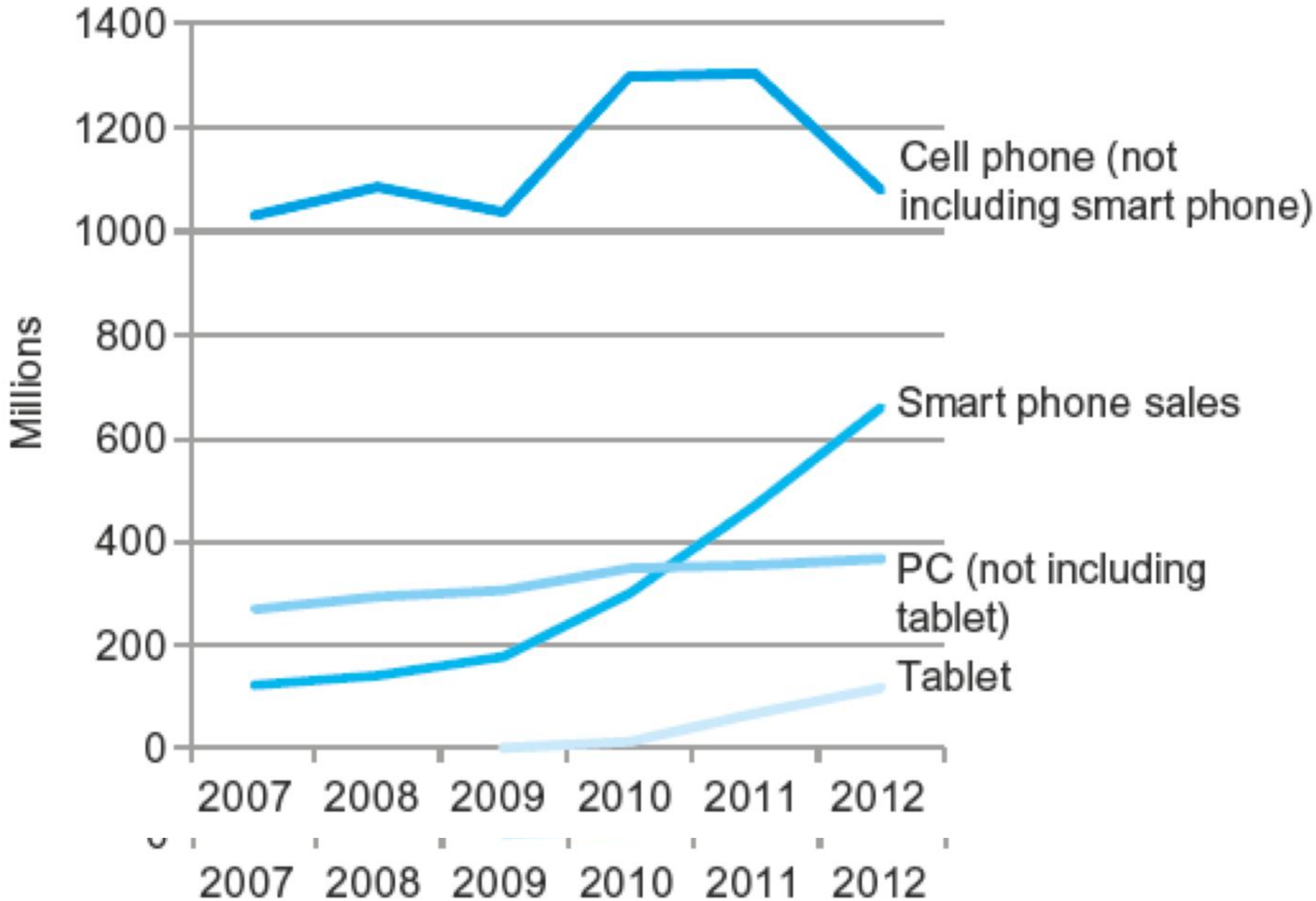
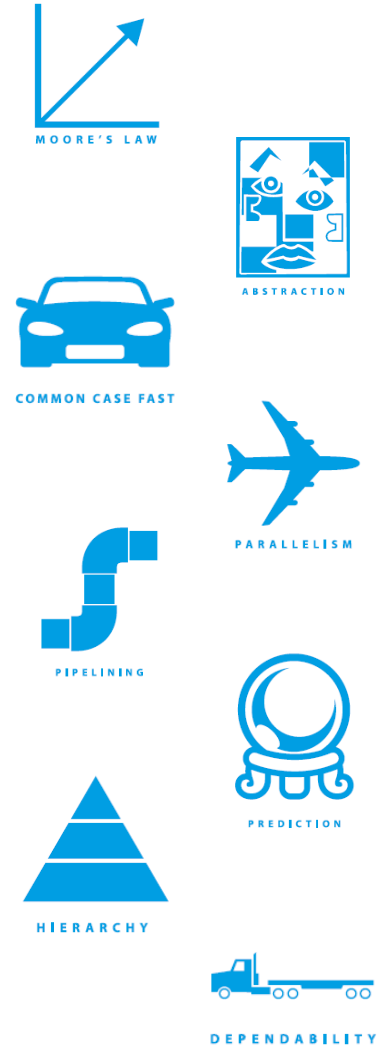


FIGURE 1.2 The number manufactured per year of tablets and smart phones, which reflect the PostPC era, versus personal computers and traditional cell phones. Smart phones represent the recent growth in the cell phone industry, and they passed PCs in 2011. Tablets are the fastest growing category, nearly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

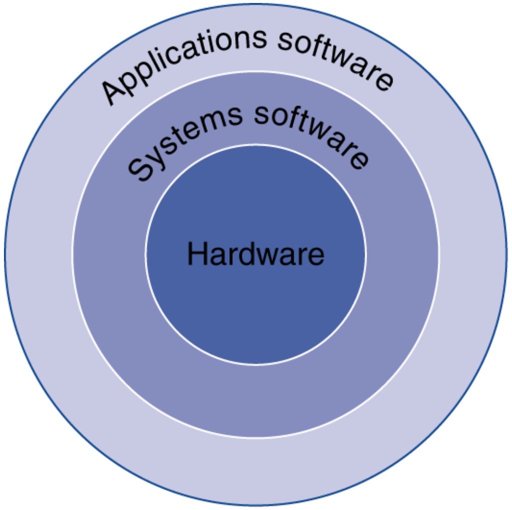
Eight Great Ideas

- 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

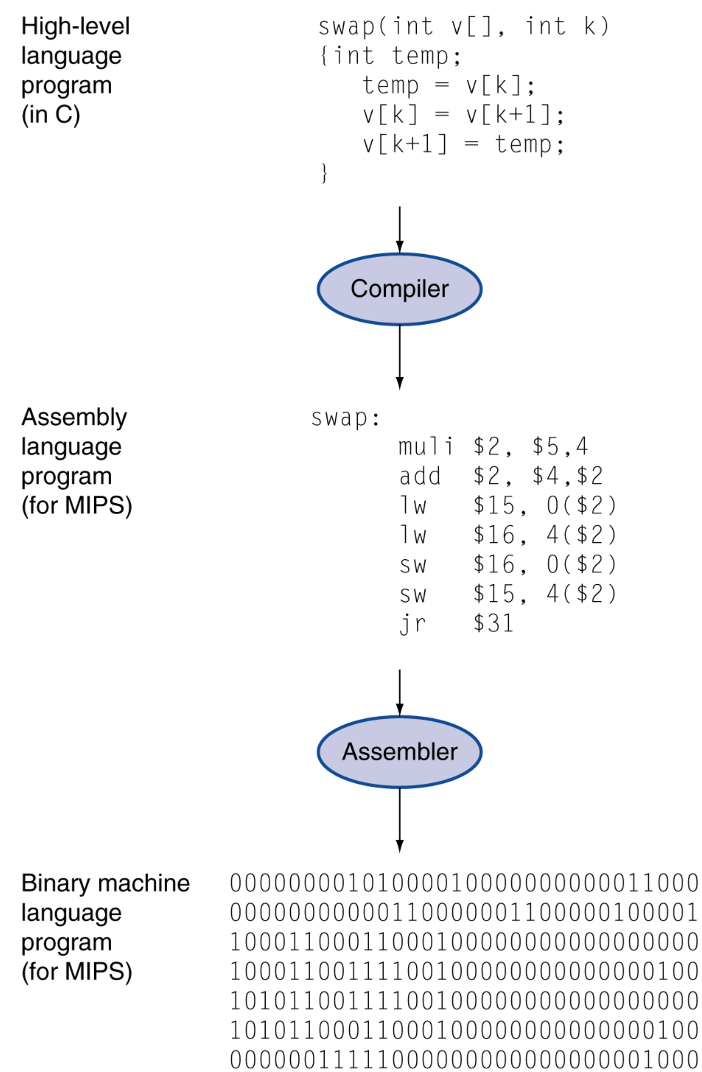
Below Your Program



- 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

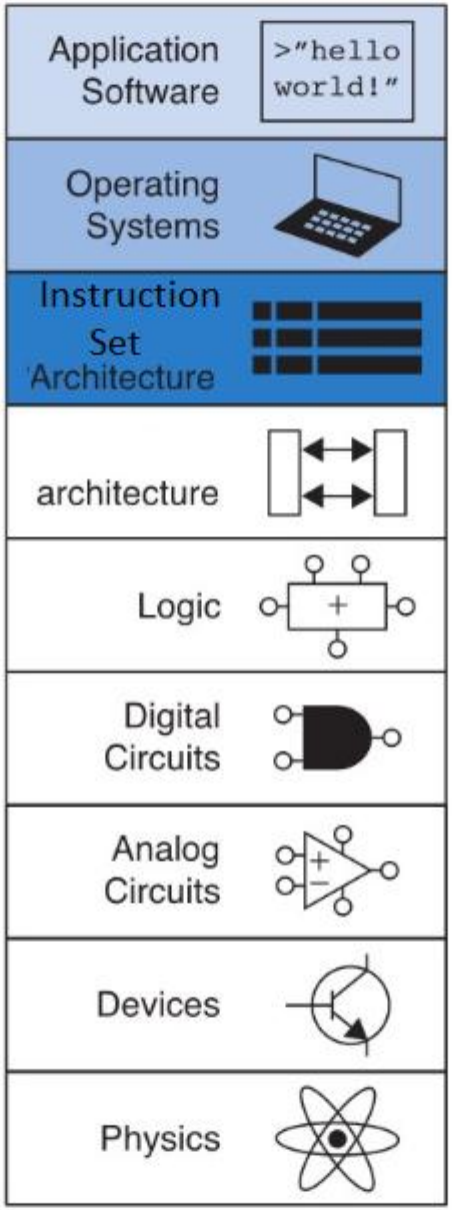
Below Your Program

- 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



Instruction set Architecture

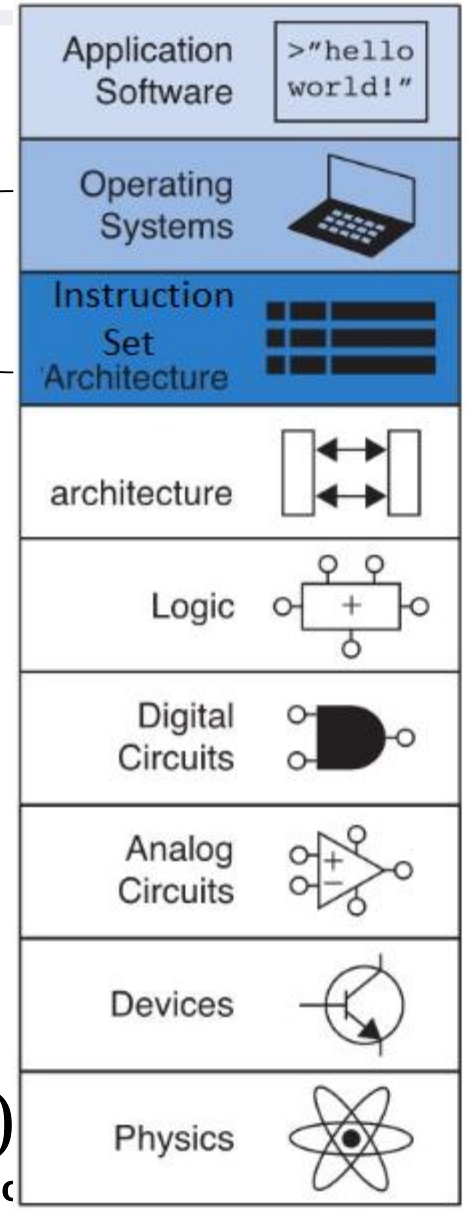
- Is the programmer's view of a computer.
- It is defined by the instruction set (language) and operand locations (registers and memory).
- Many different architectures exist, such as x86, MIPS, SPARC, ARM, and PowerPC.
- Can be implemented by different companies (ARM architecture is implemented by TI, NXP, Freescale)



Instruction set Architecture

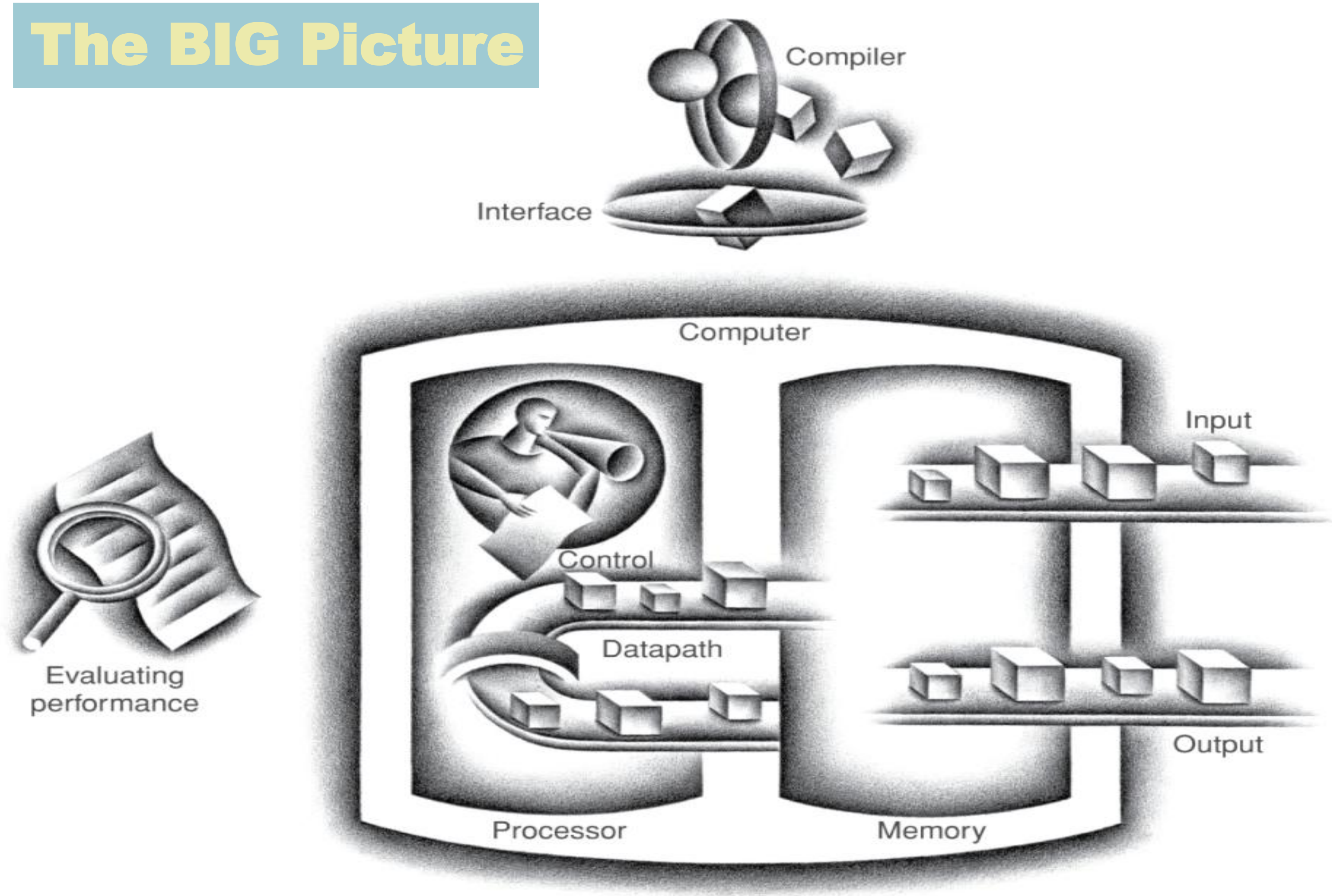
- Is the programmer's view of a computer.
- It is defined by the instruction set (language) and operand locations (registers and memory).
- Many different architectures exist, such as x86, MIPS, SPARC, ARM, and PowerPC.
- Can be implemented by different companies (ARM architecture is implemented by TI, NXP, Freescale)

Application binary interface



Components of a Computer (Architecture)

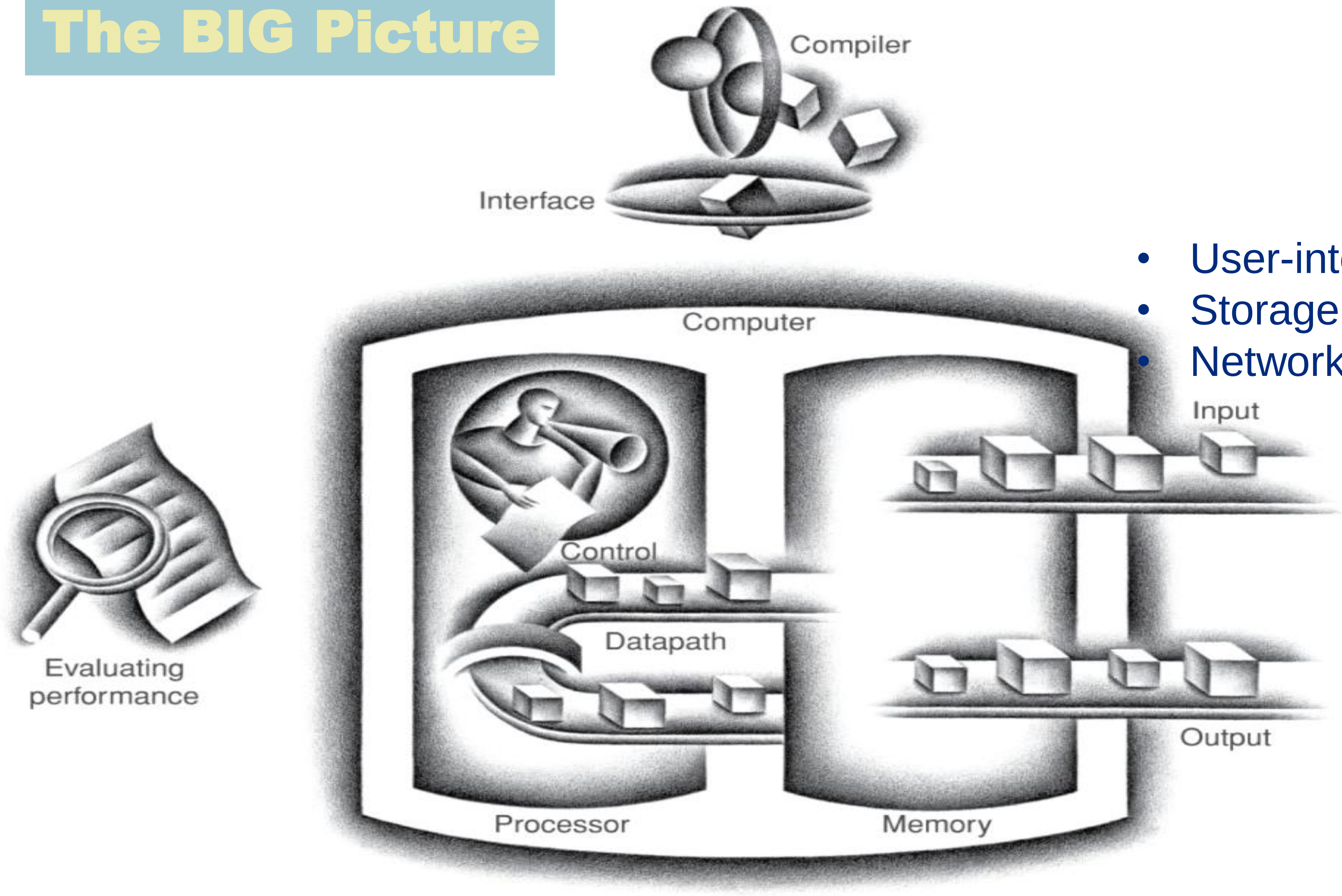
The BIG Picture



Components of a Computer (Architecture)

§1.4 Under the Covers

The BIG Picture



- User-interface devices
- Storage devices
- Network Adaptors

Processor (CPU)

- Datapath: performs operations on data
- Control: sequences datapath, memory, I/O devices.

Memory

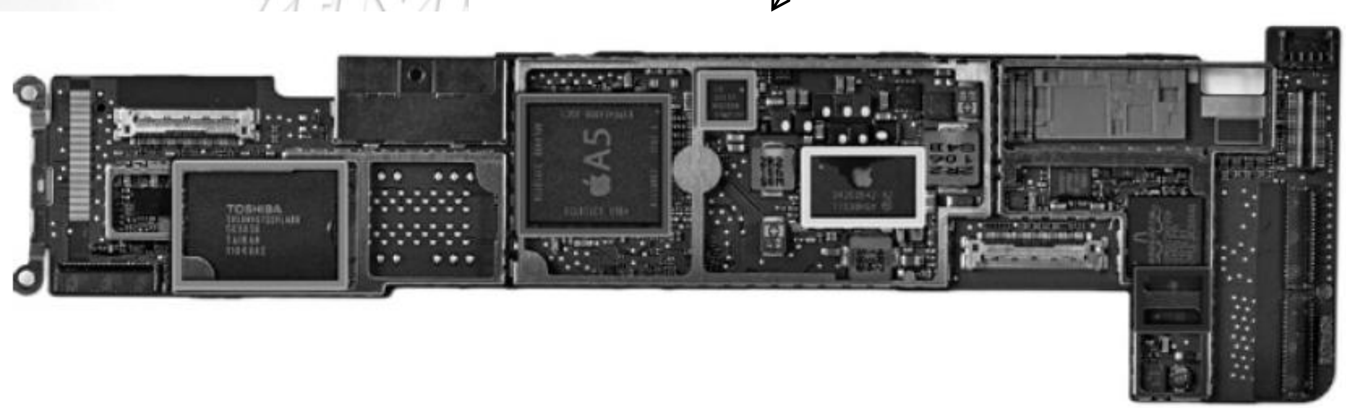
- **Main memory/primary memory:** Memory used to hold programs while they are running. Typically consists of DRAM.
- **Secondary memory Nonvolatile:** Memory used to store programs and data between runs; typically consists of flash memory in PMDs and magnetic disks in servers.
- **Cache memory:** Consists of a small, fast memory that acts as a buffer for the DRAM memory. Cache is built using a different memory technology, static random access memory (SRAM). SRAM is faster but less dense, and hence more expensive, than DRAM.

Opening the Box (Apple iPad2)

Capacitive multitouch LCD screen

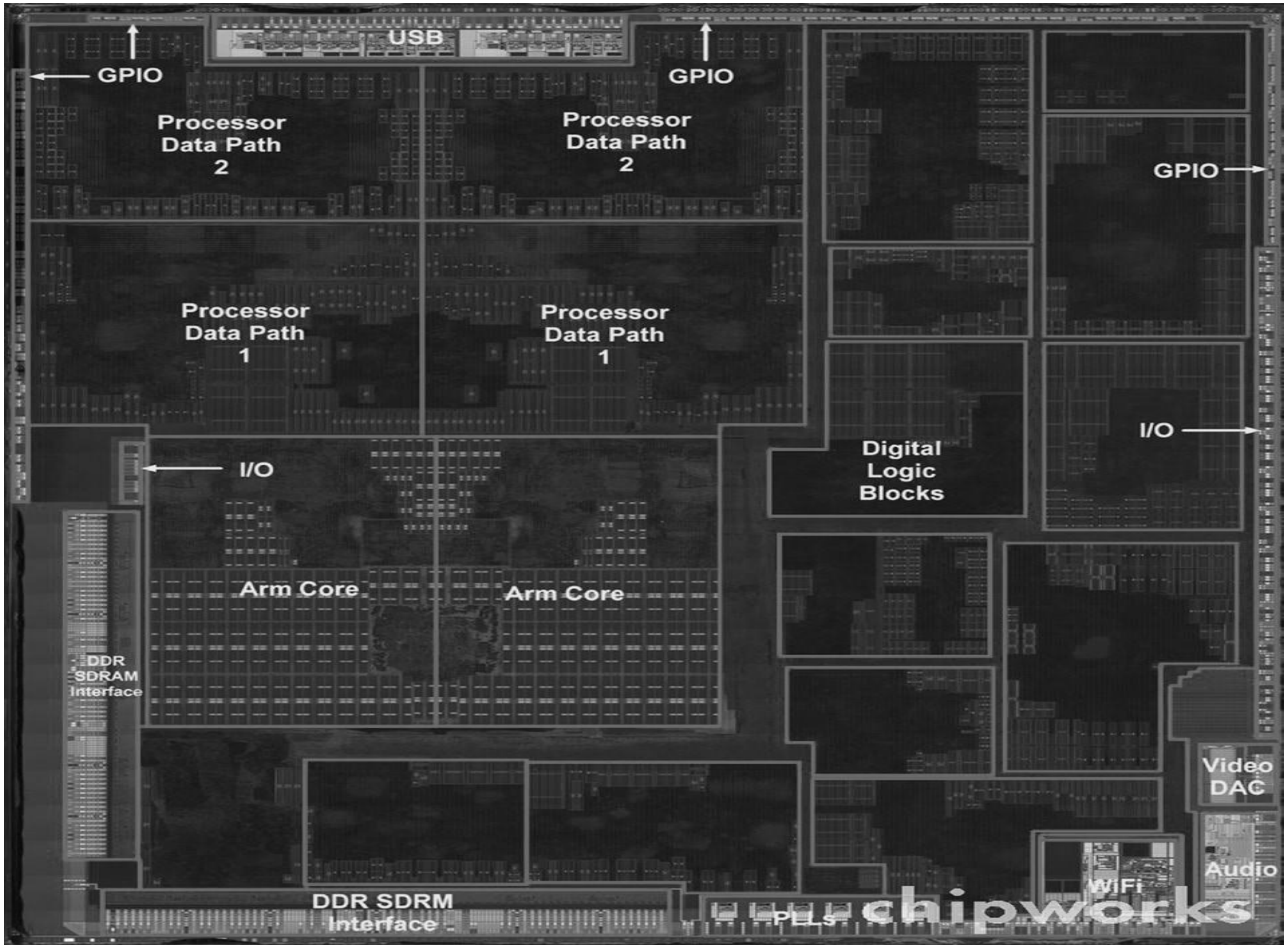
3.8 V, 25 Watt-hour battery

Computer board



Inside the Processor

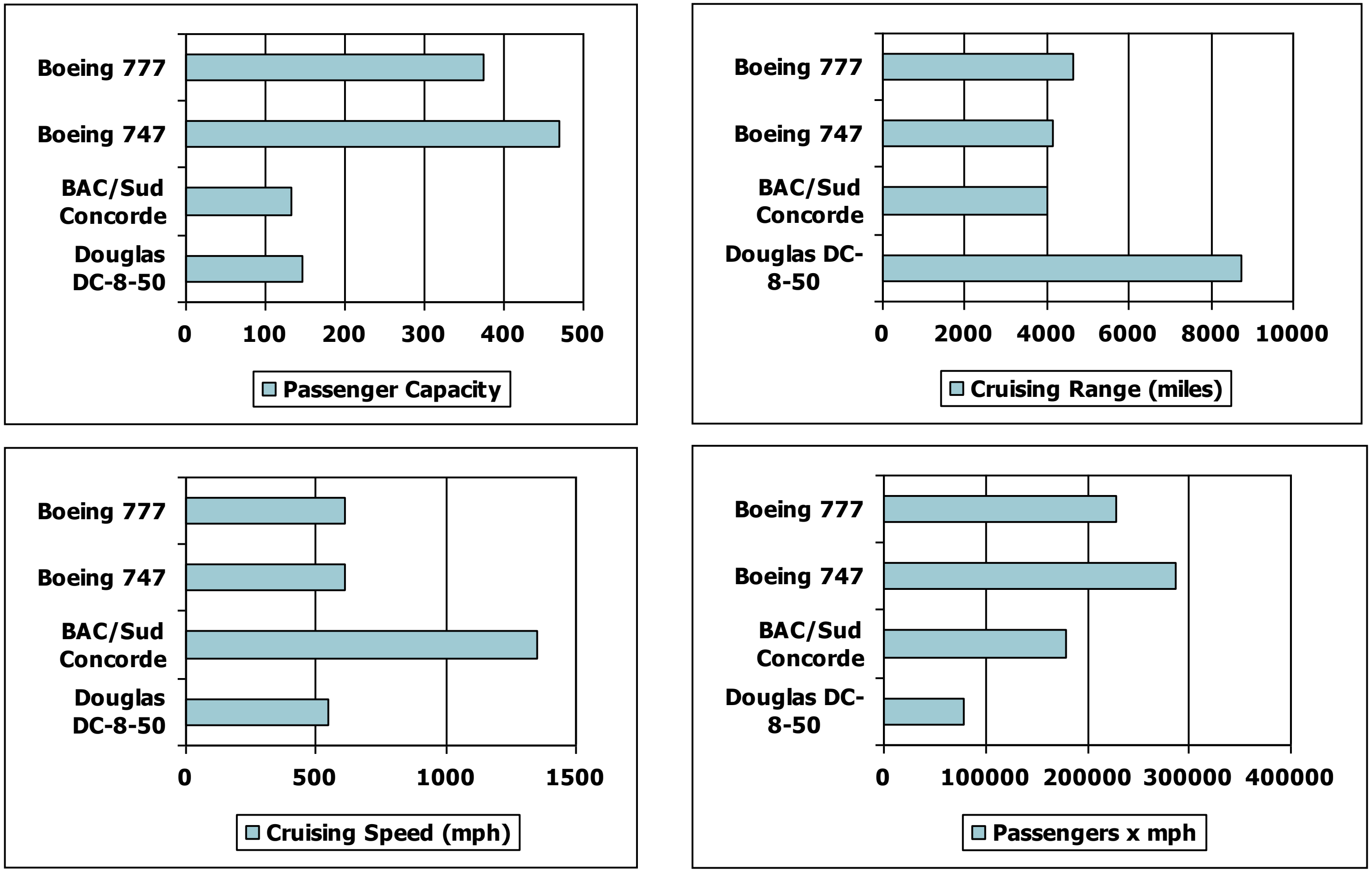
- Apple A5



Performance

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
- To compare the performance of two machines (or CPUs) "A", "B" running a given specific program (A is *n* times faster than B)

$$\text{Performance}_A = 1 / \text{Execution Time}_A$$
$$\text{Performance}_B = 1 / \text{Execution Time}_B$$

$$\text{Speedup} = n = \frac{\text{Performance}_A}{\text{Performance}_B} = \frac{\text{Execution TimeB}}{\text{Execution Time}_A}$$

Relative Performance

Example: For a given program:

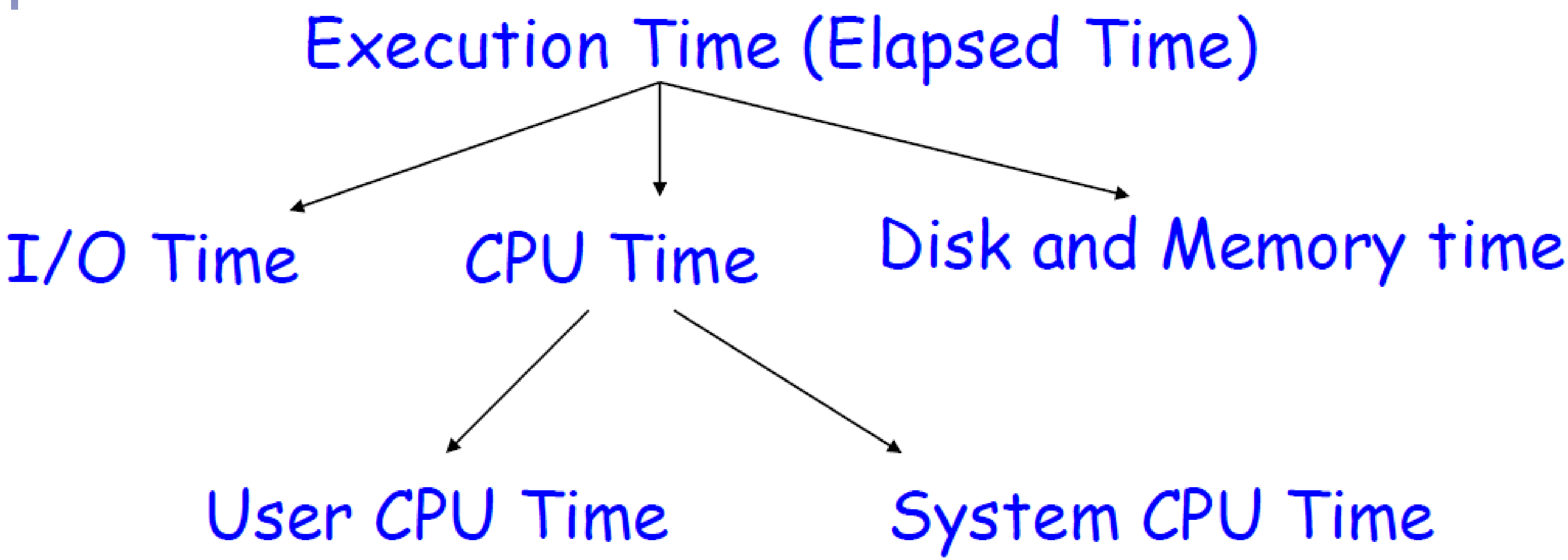
Execution time on machine A: Execution_A= 10 second

Execution time on machine B: Execution_B= 15 seconds

- Speedup= PerformanceA/ PerformanceB= Execution Time_B /Execution Time_A=15 /10 = 1.5
- The performance of machine A is 1.5 times the performance of machine B when running this program, or Machine A is said to be 1.5 times faster than machine B when running this program

Measuring Execution Time

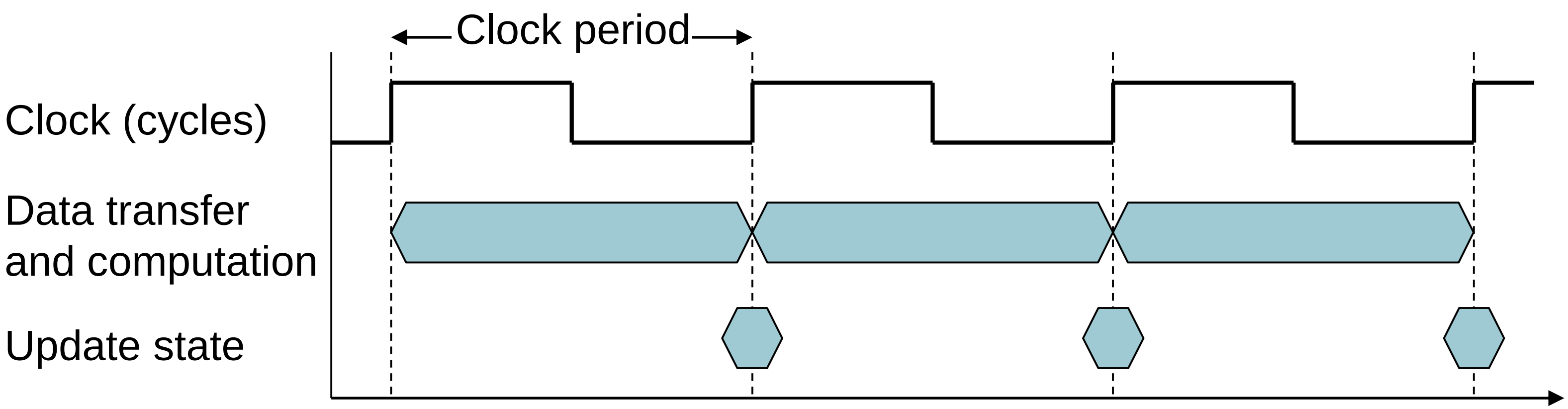
- Elapsed time
  - Total response time, including all aspects
    - Processing, I/O, OS overhead, idle time
  - Determines system performance
- CPU time
  - Time spent processing a given job
    - Discounts I/O time, other jobs' shares
  - Comprises user CPU time and system CPU time
  - Different programs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



MK MORGAN KAUFMANN

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 *T*

R = 1

- 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?

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

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

$$= 10s \times 2GHz = 20 \times 10^9$$

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

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

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

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, $CPI = \sum_{i=1}^{n}(CPI_i\ x\ C_{fraction_i})$

  - 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

- Sequence 1: IC = 5
  - Clock Cycles
    $= 2\times1 + 1\times2 + 2\times3$
    $= 10$
  - Avg. CPI = 10/5 = 2.0
- Sequence 2: IC = 6
  - Clock Cycles
    $= 4\times1 + 1\times2 + 1\times3$
    $= 9$
  - 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:

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

or

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

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

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

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

or

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

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)				√	√	√

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

Problems to Solve

- 1.5, 1.6,1.7
- Note that E9 for example means 10^9.

Thank you