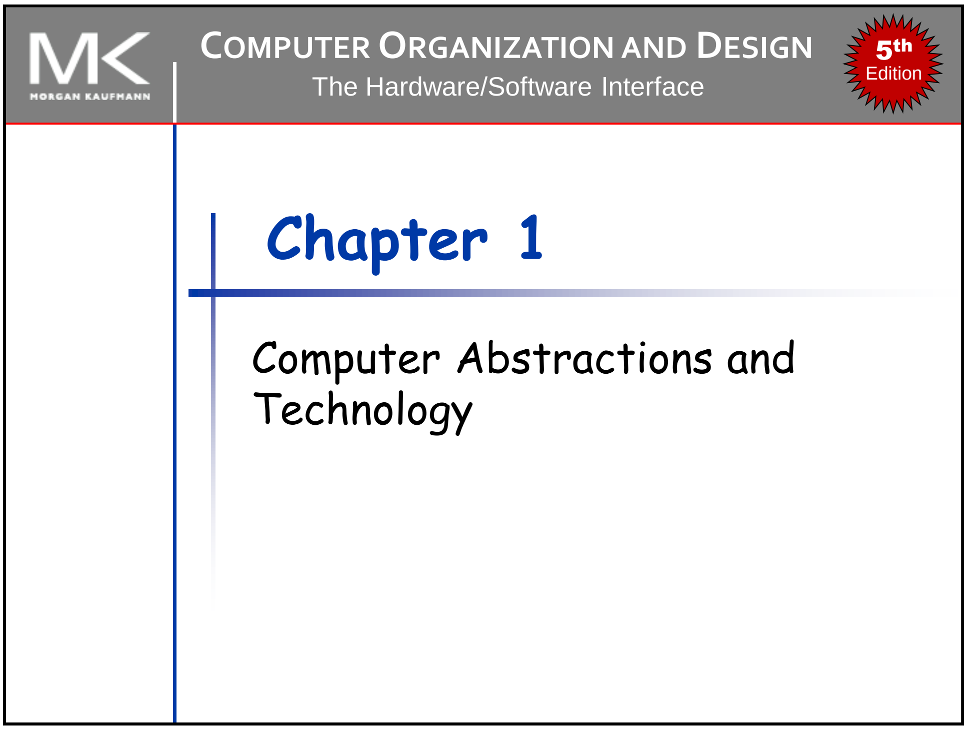


MORGAN KAUFMANN

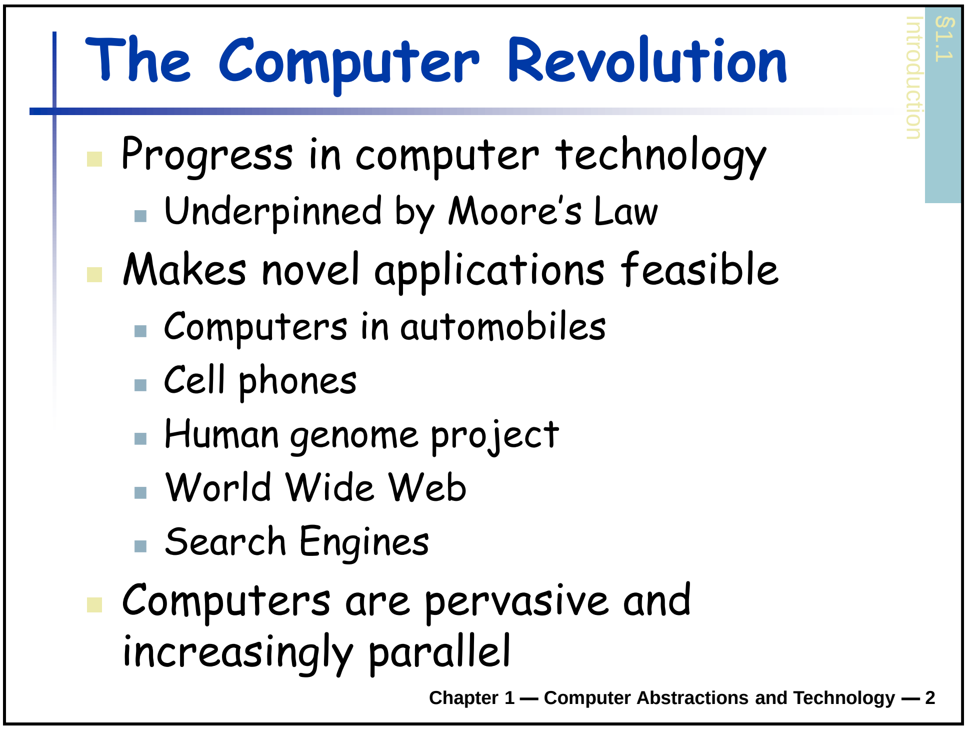
COMPUTER ORGANIZATION AND DESIGN

The Hardware/Software Interface

5th Edition

Chapter 1

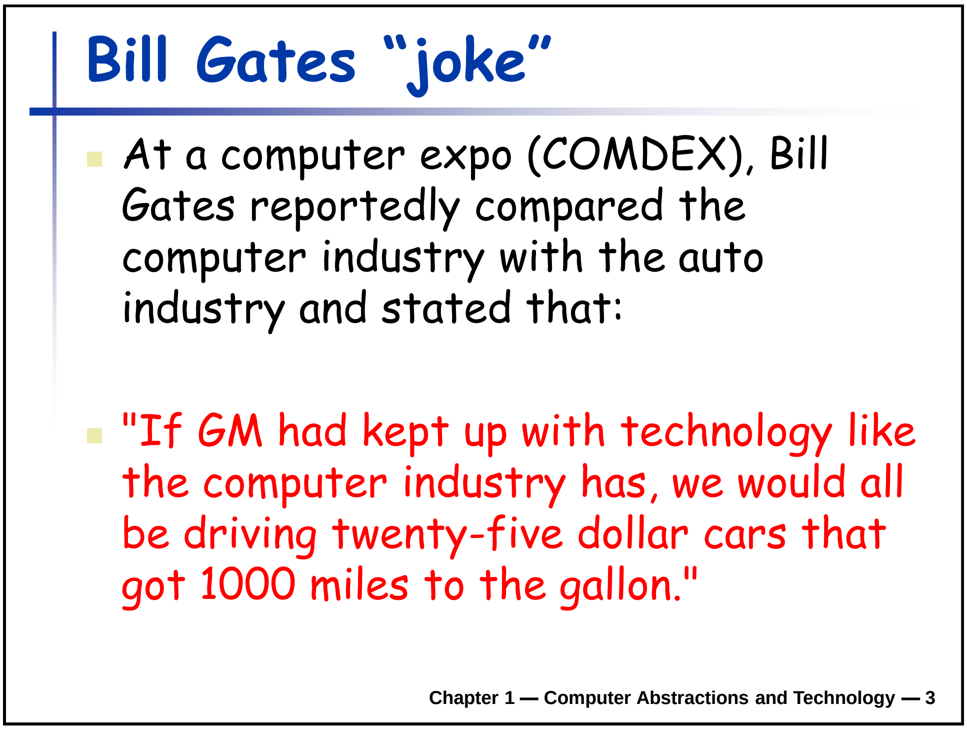
Computer Abstractions and Technology



The Computer Revolution

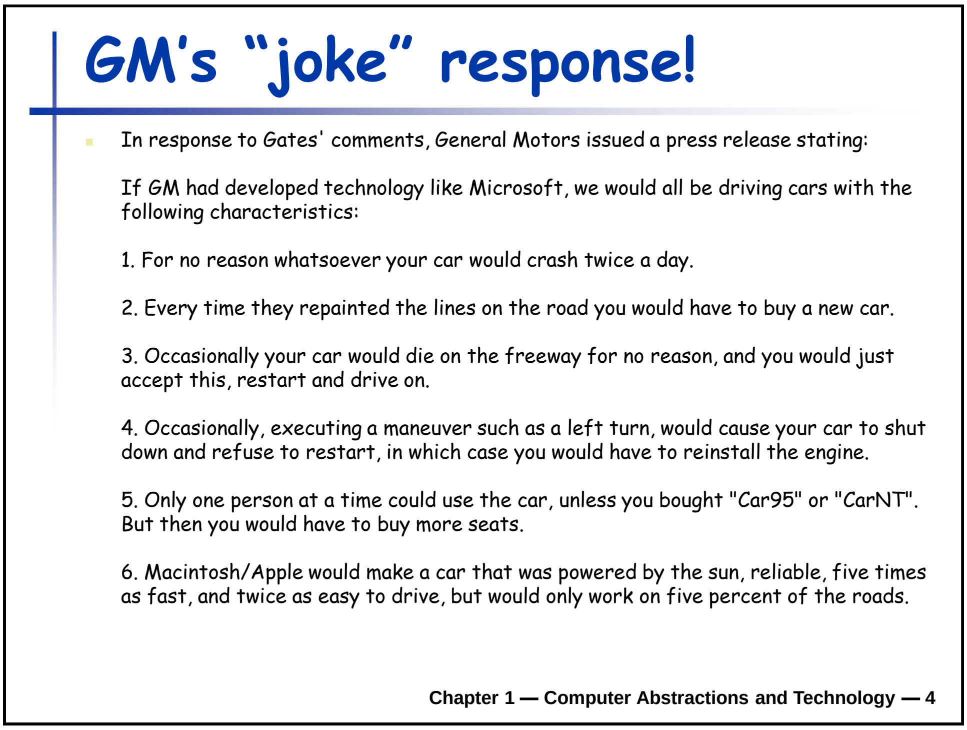
§1.1 Introduction

- 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 and increasingly parallel



Bill Gates "joke"

- At a computer expo (COMDEX), Bill Gates reportedly compared the computer industry with the auto industry and stated that:
- "If GM had kept up with technology like the computer industry has, we would all be driving twenty-five dollar cars that got 1000 miles to the gallon."



GM's "joke" response!

- In response to Gates' comments, General Motors issued a press release stating:

  If GM had developed technology like Microsoft, we would all be driving cars with the following characteristics:

  1. For no reason whatsoever your car would crash twice a day.

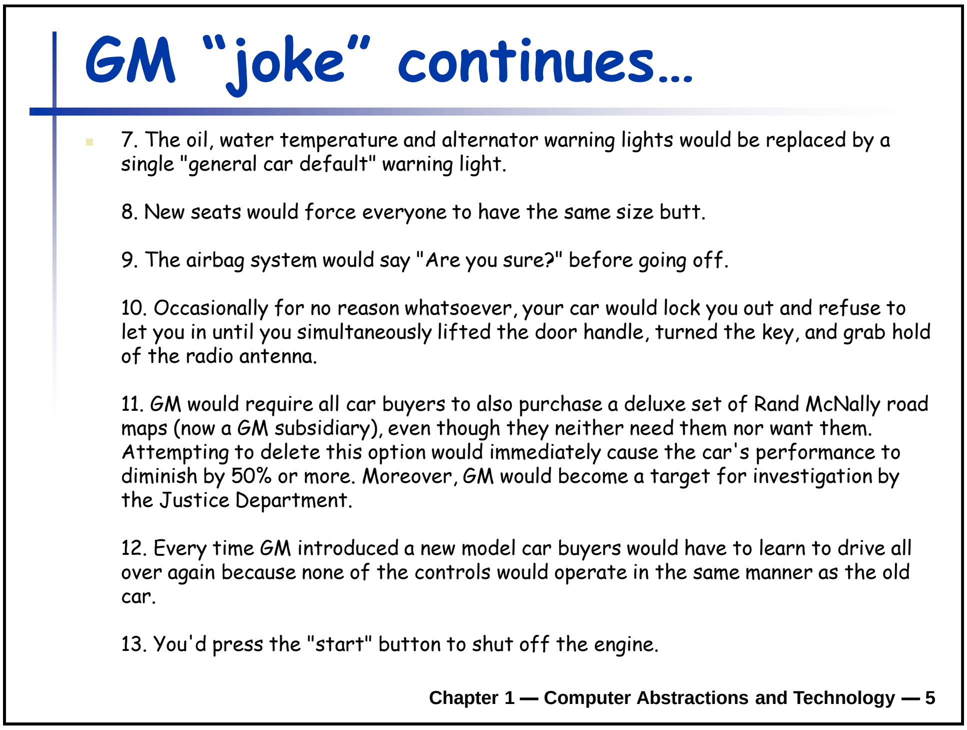
  2. Every time they repainted the lines on the road you would have to buy a new car.

  3. Occasionally your car would die on the freeway for no reason, and you would just accept this, restart and drive on.

  4. Occasionally, executing a maneuver such as a left turn, would cause your car to shut down and refuse to restart, in which case you would have to reinstall the engine.

  5. Only one person at a time could use the car, unless you bought "Car95" or "CarNT". But then you would have to buy more seats.

  6. Macintosh/Apple would make a car that was powered by the sun, reliable, five times as fast, and twice as easy to drive, but would only work on five percent of the roads.



GM "joke" continues…

- 7. The oil, water temperature and alternator warning lights would be replaced by a single "general car default" warning light.

  8. New seats would force everyone to have the same size butt.

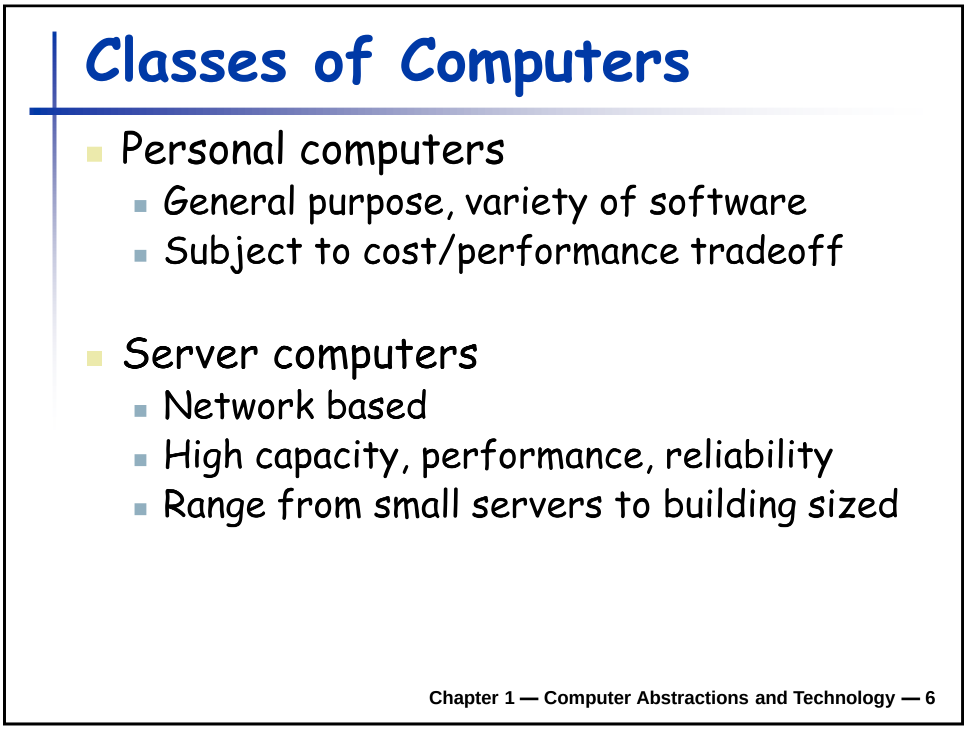
  9. The airbag system would say "Are you sure?" before going off.

  10. Occasionally for no reason whatsoever, your car would lock you out and refuse to let you in until you simultaneously lifted the door handle, turned the key, and grab hold of the radio antenna.

  11. GM would require all car buyers to also purchase a deluxe set of Rand McNally road maps (now a GM subsidiary), even though they neither need them nor want them. Attempting to delete this option would immediately cause the car's performance to diminish by 50% or more. Moreover, GM would become a target for investigation by the Justice Department.

  12. Every time GM introduced a new model car buyers would have to learn to drive all over again because none of the controls would operate in the same manner as the old car.

  13. You'd press the "start" button to shut off the engine.



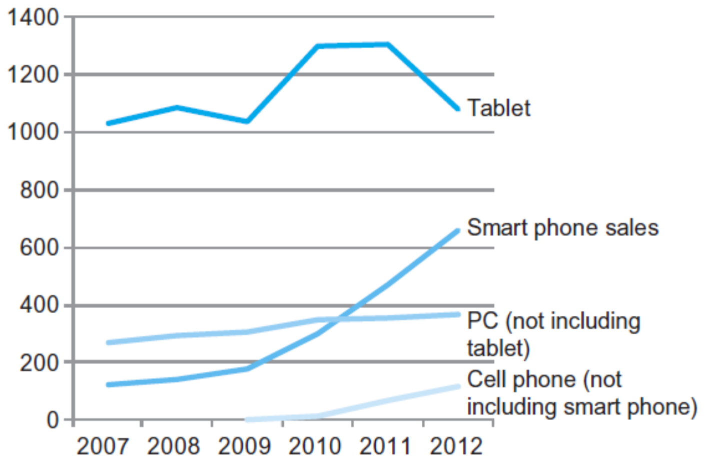
Classes of Computers

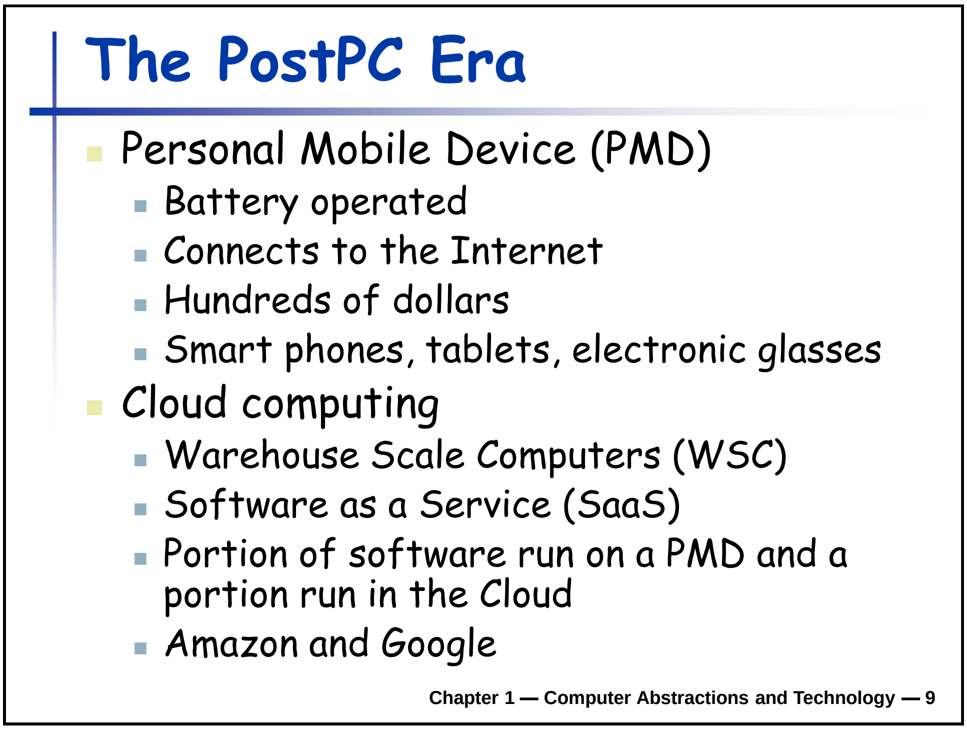
- Personal computers
  - General purpose, variety of software
  - Subject to cost/performance tradeoff
- Server computers
  - Network based
  - High capacity, performance, reliability
  - Range from small servers to building sized

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

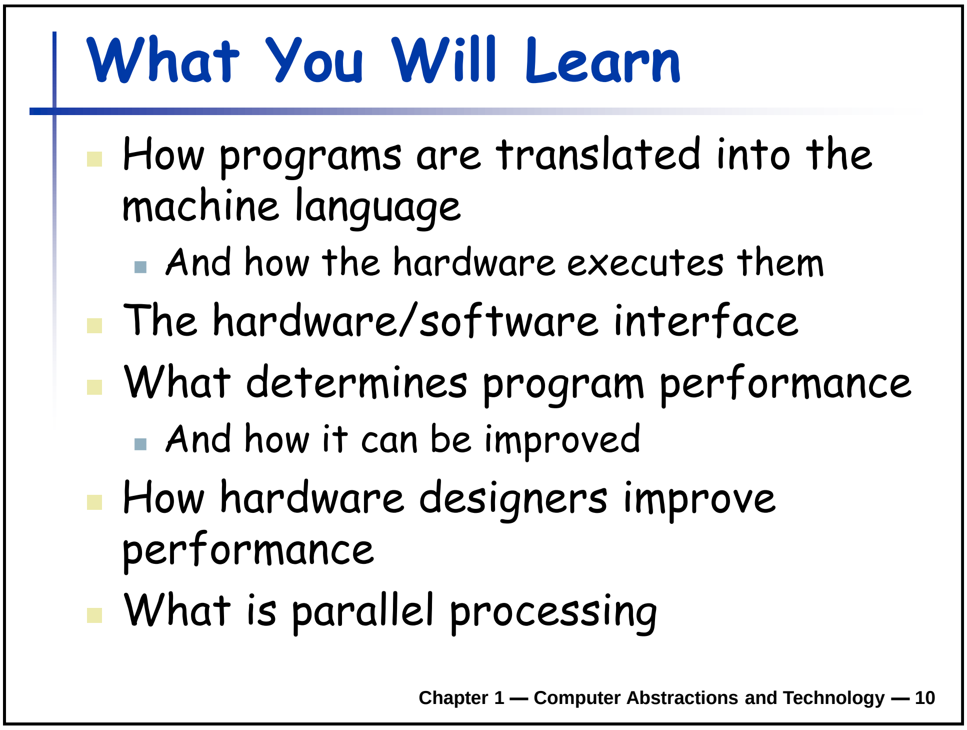
The PostPC Era





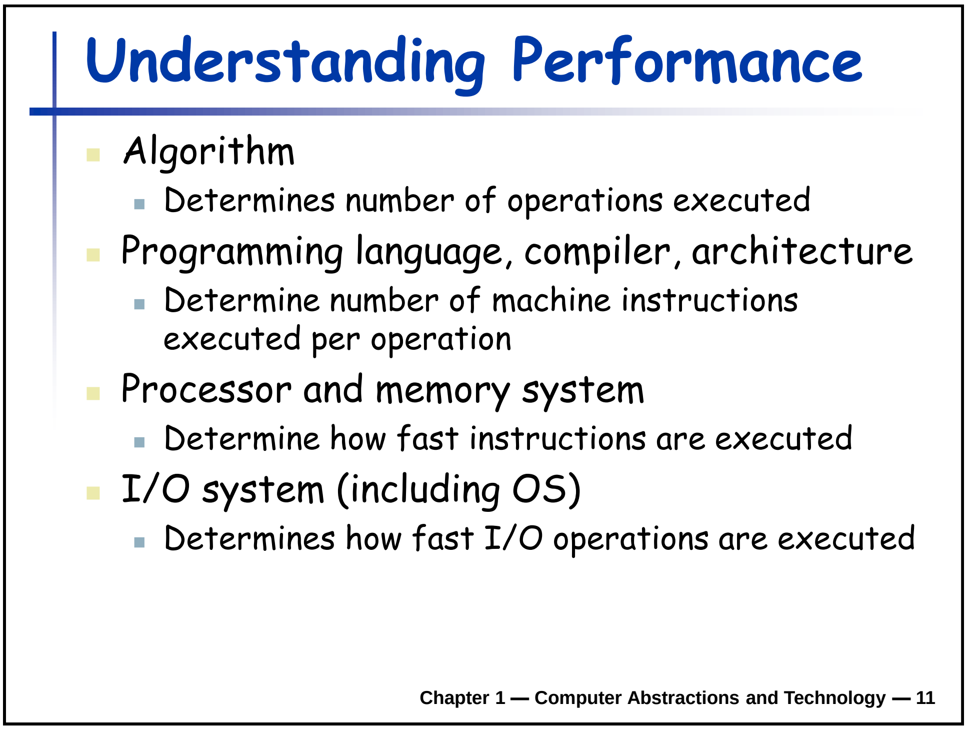
The PostPC Era

- 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

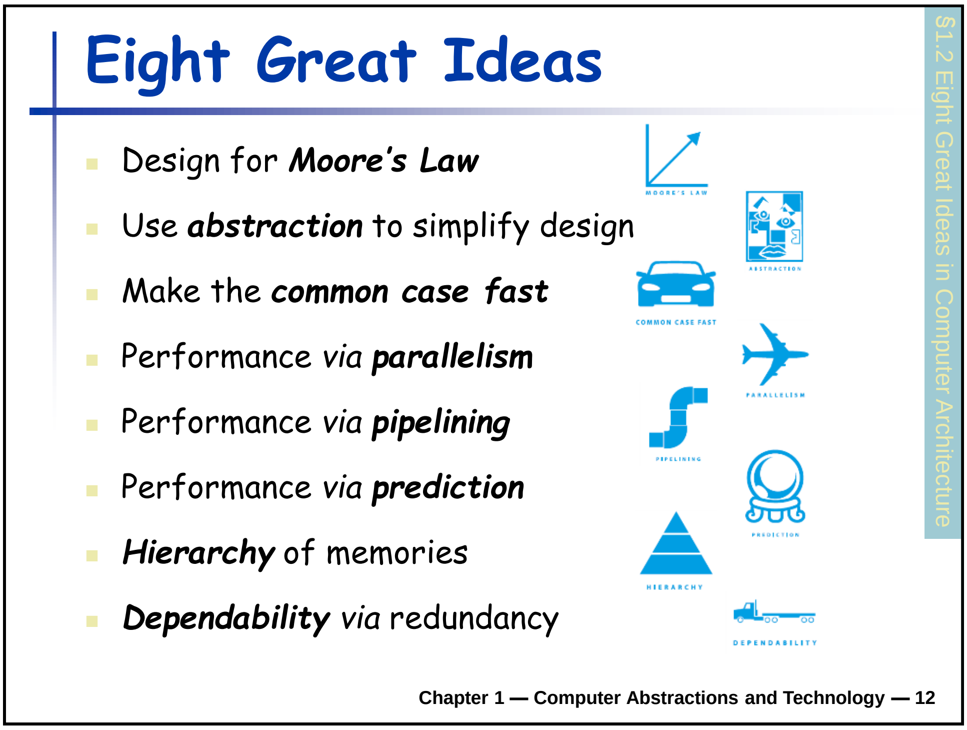
- How programs are translated into the machine language
  - And how the hardware executes them
- The hardware/software interface
- What determines program performance
  - And how it can be improved
- How hardware designers improve performance
- What is parallel processing



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

Chapter 1 — Computer Abstractions and Technology — 11



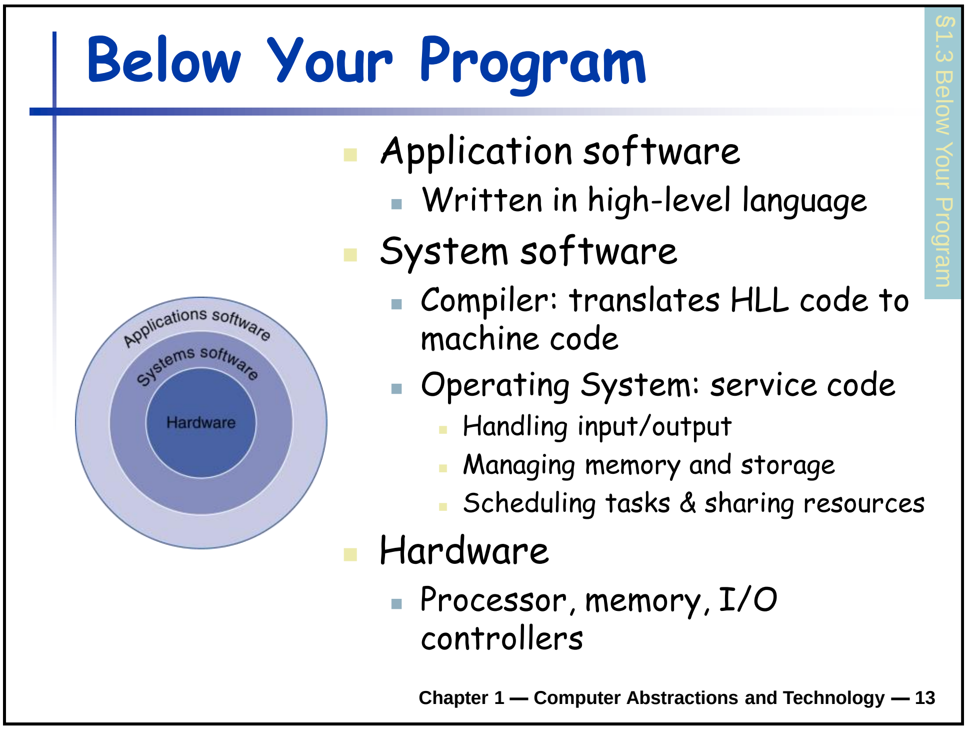
Eight Great Ideas

§1.2 Eight Great Ideas in Computer Architecture

- 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

MOORE'S LAW
ABSTRACTION
COMMON CASE FAST
PARALLELISM
PIPELINING
PREDICTION
HIERARCHY
DEPENDABILITY

Chapter 1 — Computer Abstractions and Technology — 12



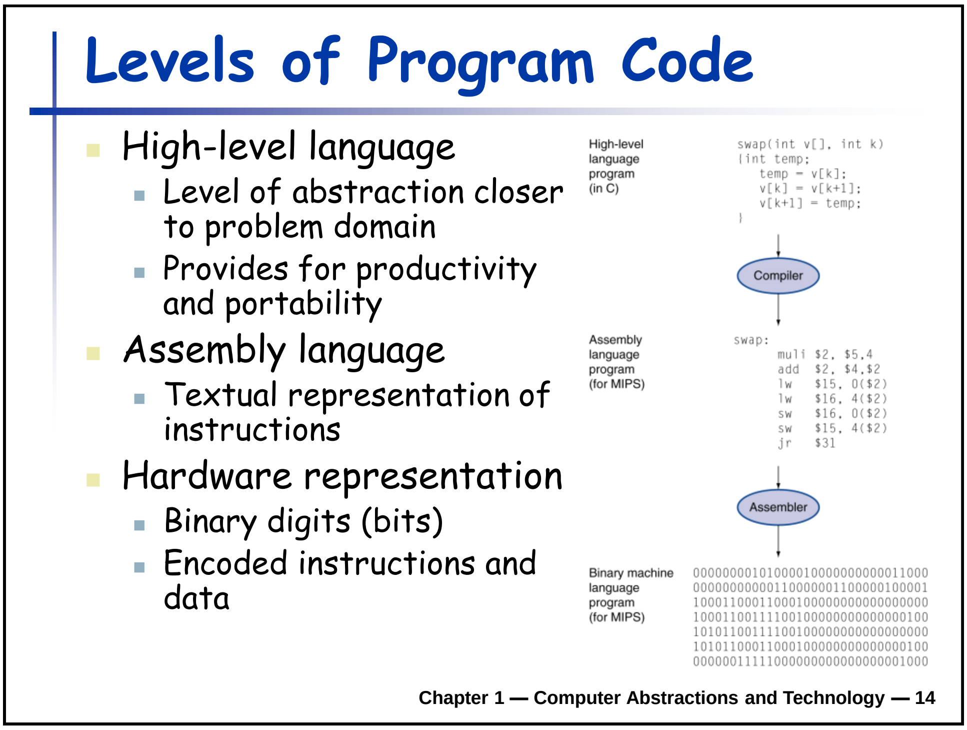
Below Your Program

§1.3 Below Your Program

Applications software
Systems software
Hardware

- 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

Chapter 1 — Computer Abstractions and Technology — 13



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)

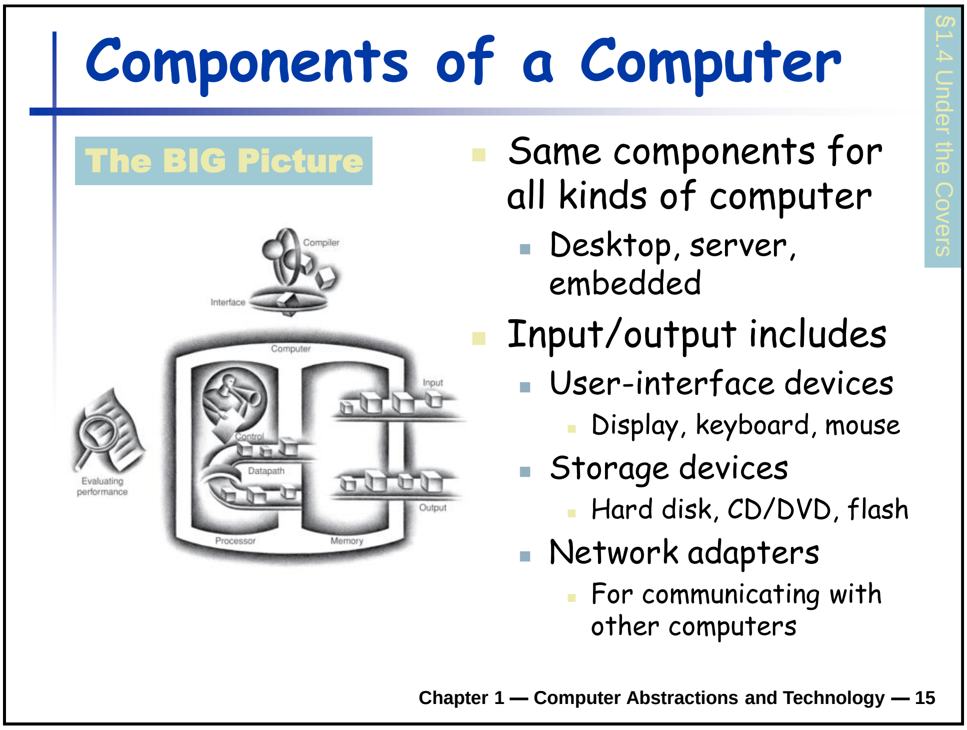
```
swap(int v[], int k)
{int temp;
   temp = v[k];
   v[k] = v[k+1];
   v[k+1] = temp;
}
```

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
```

```
00000000101000010000000000011000
00000000000110000001100000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

Chapter 1 — Computer Abstractions and Technology — 14



Components of a Computer

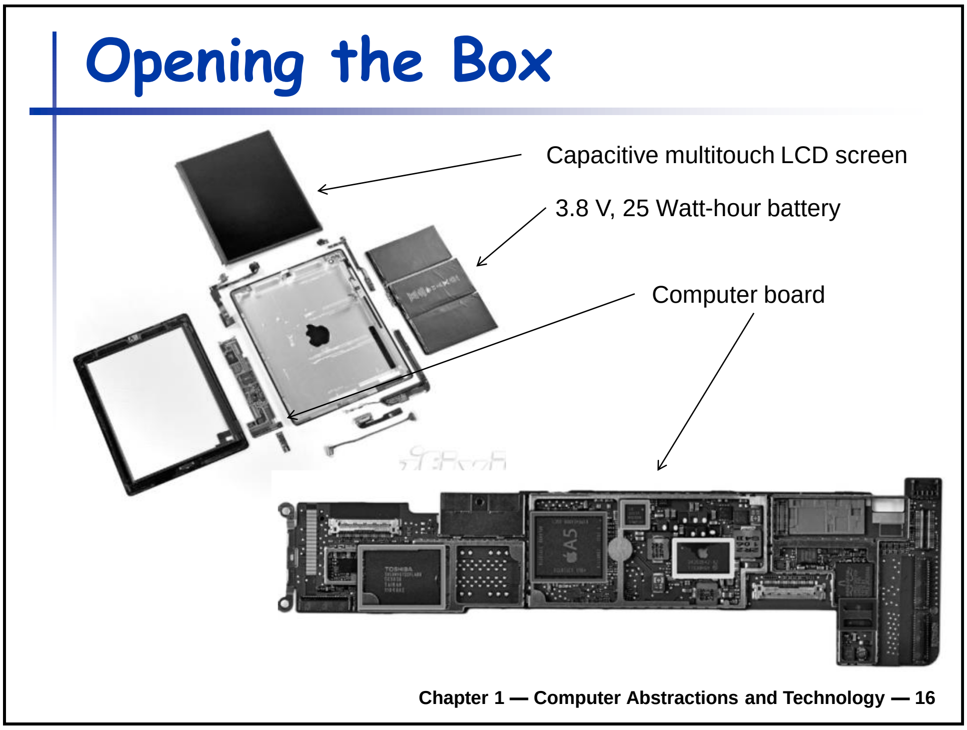
§1.4 Under the Covers

The BIG Picture

Compiler
Interface
Computer
Input
Control
Datapath
Output
Processor
Memory
Evaluating performance

- 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

Chapter 1 — Computer Abstractions and Technology — 15



Opening the Box

Capacitive multitouch LCD screen

3.8 V, 25 Watt-hour battery

Computer board

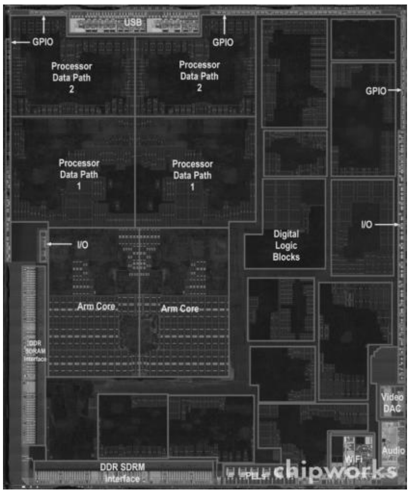
Chapter 1 — Computer Abstractions and Technology — 16

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

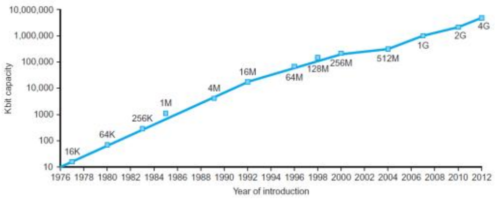
The BIG Picture

- Abstraction helps us deal with complexity by hiding lower-level details
- Instruction set architecture (ISA)
  - The hardware/software interface
  - This is the assembly-level programming we cover in this course
    - (Beneath this is machine-level programming, which is entirely ones and zeros....)

Technology Trends

§1.5 Technologies for Building Processors and Memory

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



DRAM capacity

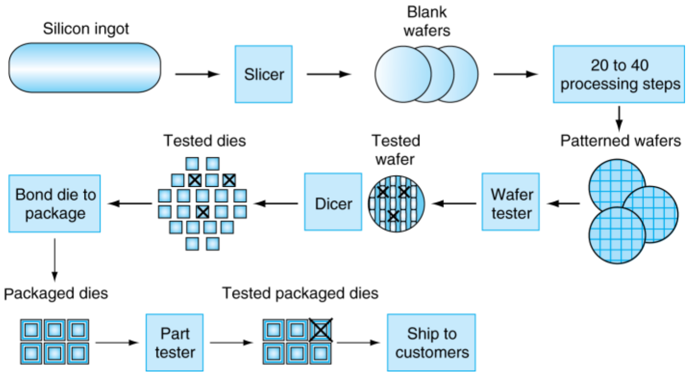
Year	Technology	Relative performance/cost
1951	Vacuum tube	1
1965	Transistor	35
1975	Integrated circuit (IC)	900
1995	Very large scale IC (VLSI)	2,400,000
2013	Ultra large scale IC	250,000,000,000

Semiconductor Technology

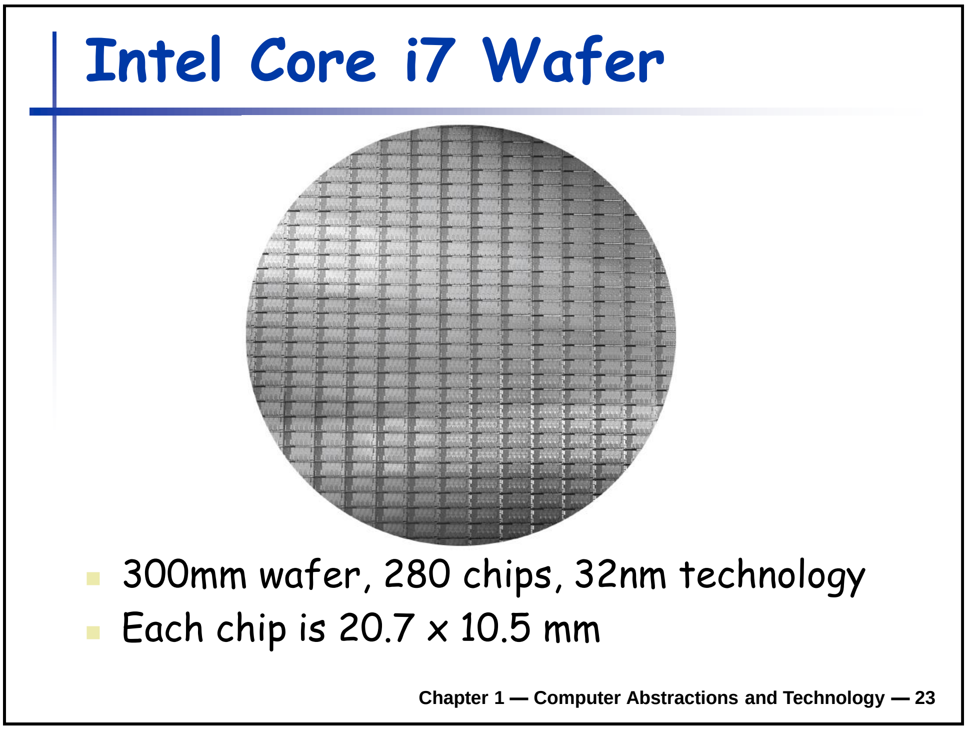
半导体

- Silicon: semiconductor
- Add materials to transform properties:
  - Conductors
  - Insulators 绝缘子
  - Switch

Manufacturing ICs

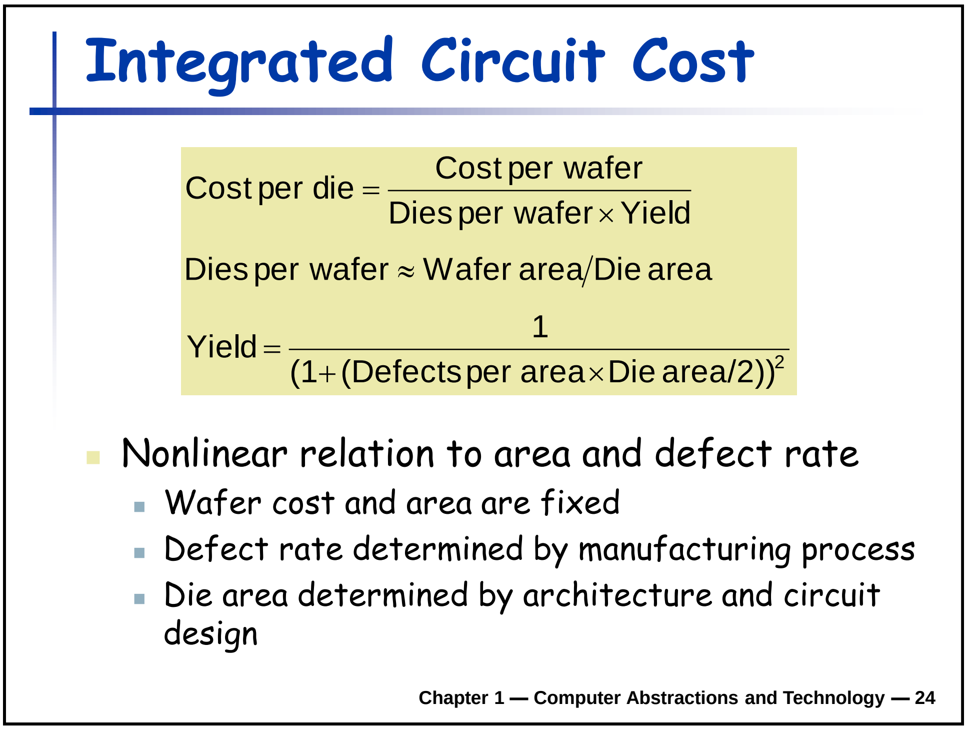


- Yield: proportion of working dies per wafer



Intel Core i7 Wafer

- 300mm wafer, 280 chips, 32nm technology
- Each chip is 20.7 x 10.5 mm



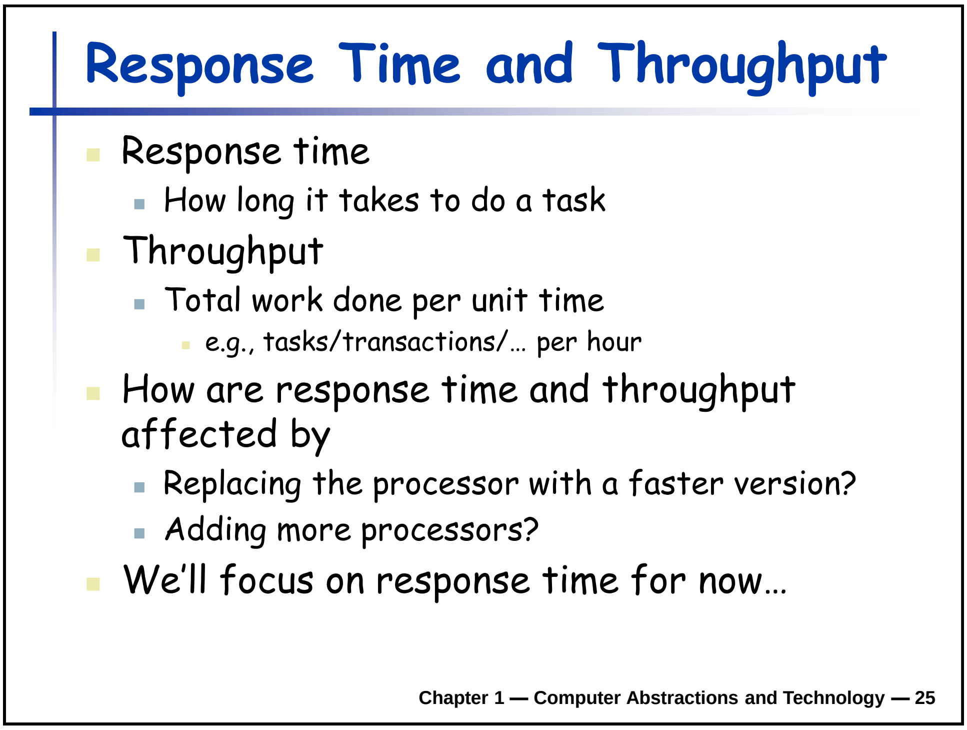
Integrated Circuit Cost

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

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

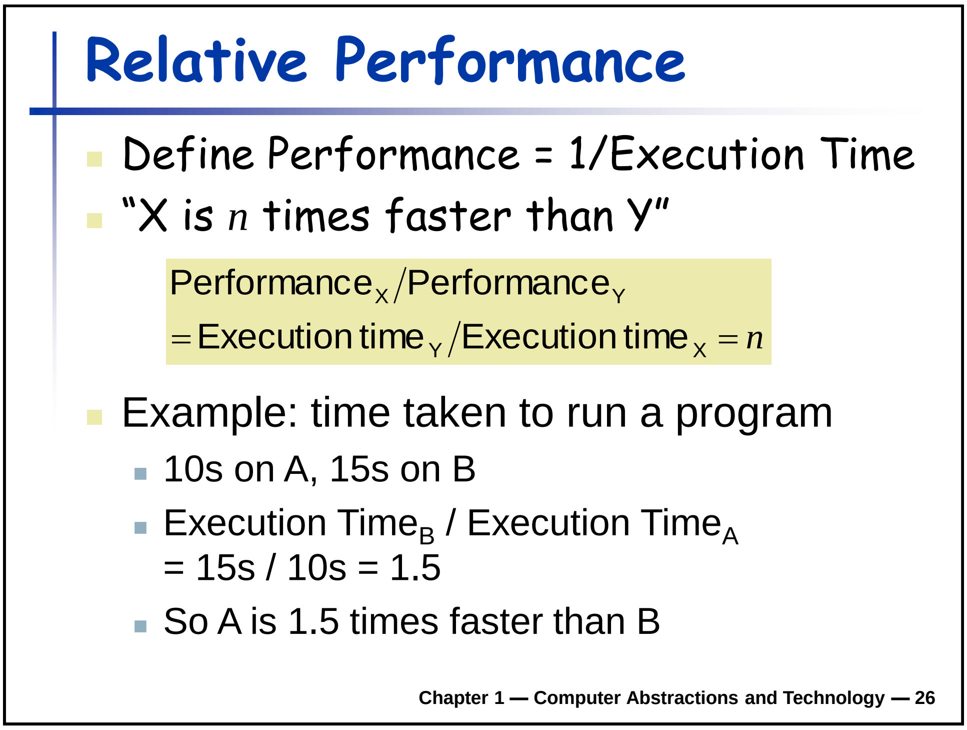
$$\text{Yield} = \frac{1}{(1 + (\text{Defects per area} \times \text{Die area/2}))^2}$$

- 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



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"

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

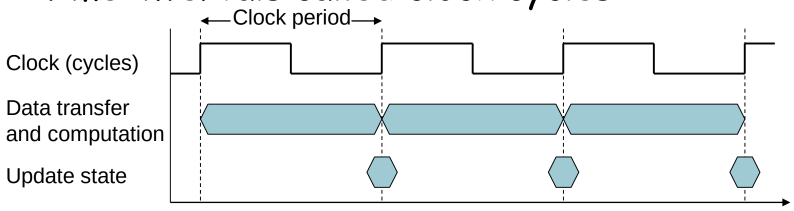
- Example: time taken to run a program
  - 10s on A, 15s on B
  - Execution Time$_B$ / Execution Time$_A$ = 15s / 10s = 1.5
  - So A is 1.5 times faster than B

Measuring Execution Time

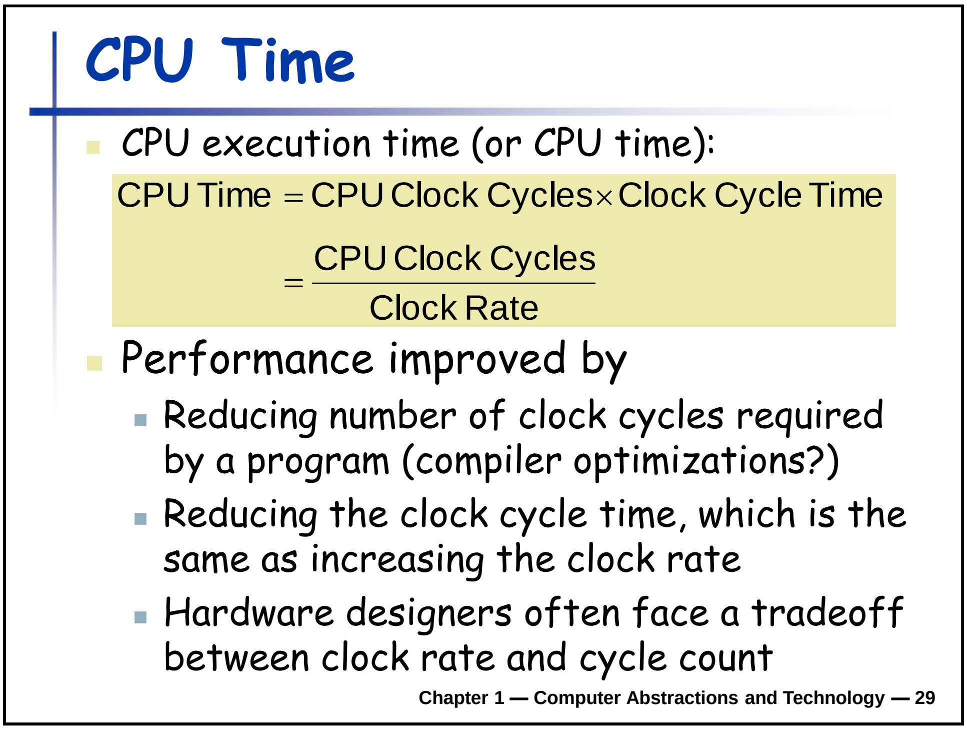
- Elapsed time
  - Total response time, including all aspects
    - Processing, I/O, OS overhead, idle time
  - Determines system performance
- CPU time
  - Time spent processing a given job
    - Discounts I/O time, other jobs' shares
  - Comprises user CPU time and system CPU time
  - Different programs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

CPU Clocking

- Operation of digital hardware is governed by a constant-rate clock that defines discrete time intervals called clock cycles



- 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



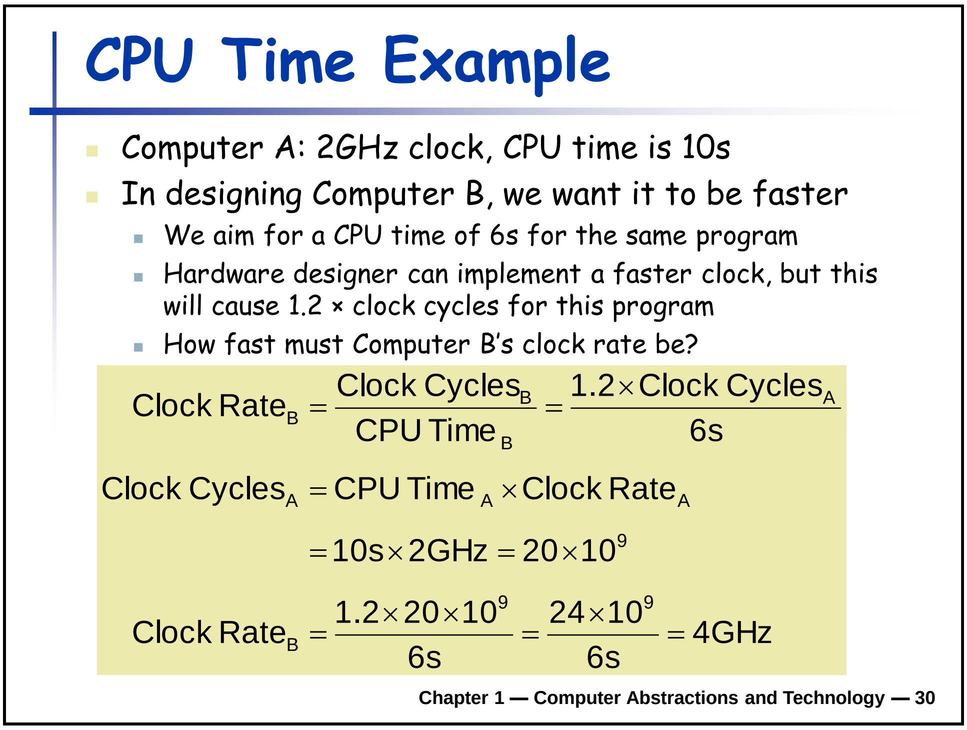
CPU Time

- CPU execution time (or 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 required by a program (compiler optimizations?)
  - Reducing the clock cycle time, which is the same as increasing the clock rate
  - Hardware designers often face a tradeoff between clock rate and cycle count

Chapter 1 — Computer Abstractions and Technology — 29



CPU Time Example

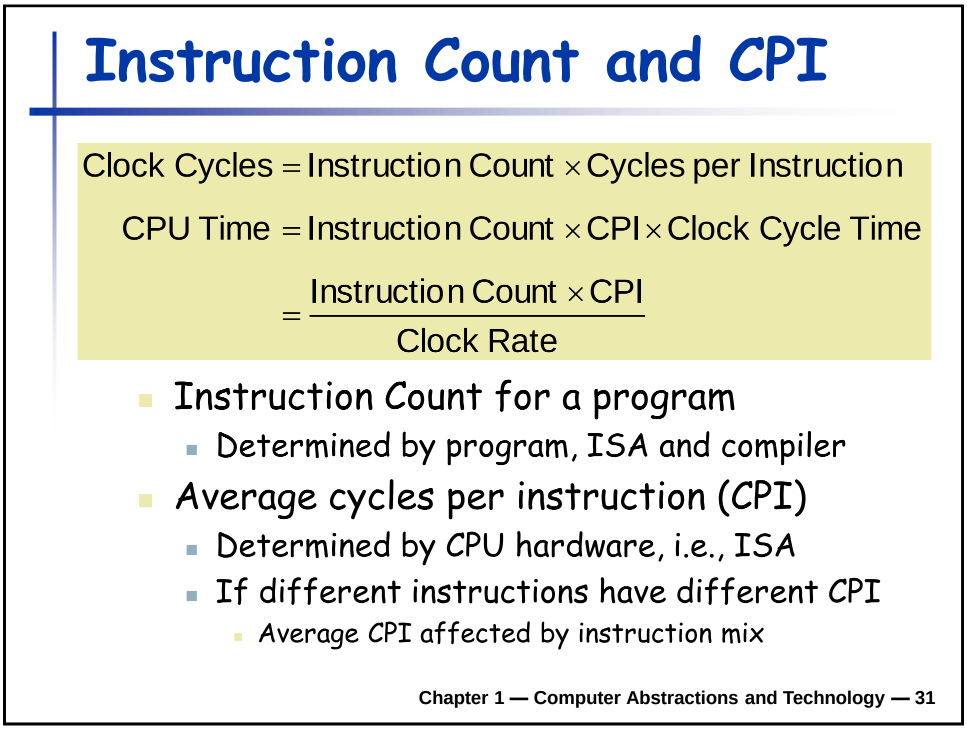
- Computer A: 2GHz clock, CPU time is 10s
- In designing Computer B, we want it to be faster
  - We aim for a CPU time of 6s for the same program
  - Hardware designer can implement a faster clock, but this will cause 1.2 × clock cycles for this program
  - How fast must Computer B's clock rate be?

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

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

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

Chapter 1 — Computer Abstractions and Technology — 30



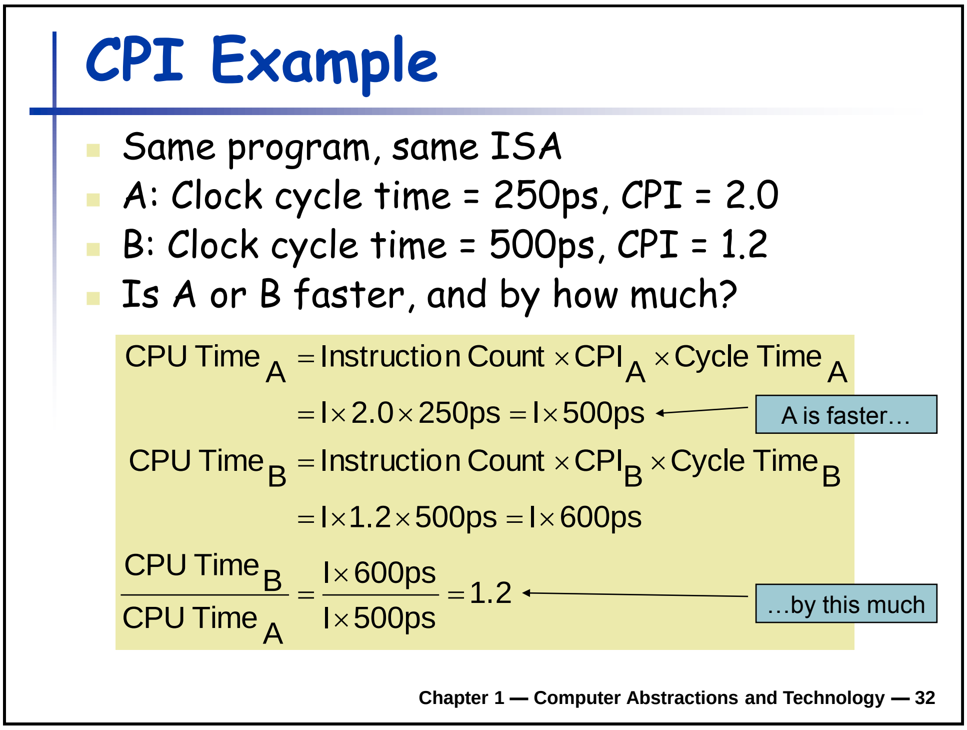
Instruction Count and CPI

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

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

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

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



CPI Example

- Same program, same ISA
- A: Clock cycle time = 250ps, CPI = 2.0
- B: Clock cycle time = 500ps, CPI = 1.2
- Is A or B faster, and by how much?

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

$$= \text{I} \times 2.0 \times 250\text{ps} = \text{I} \times 500\text{ps}$$

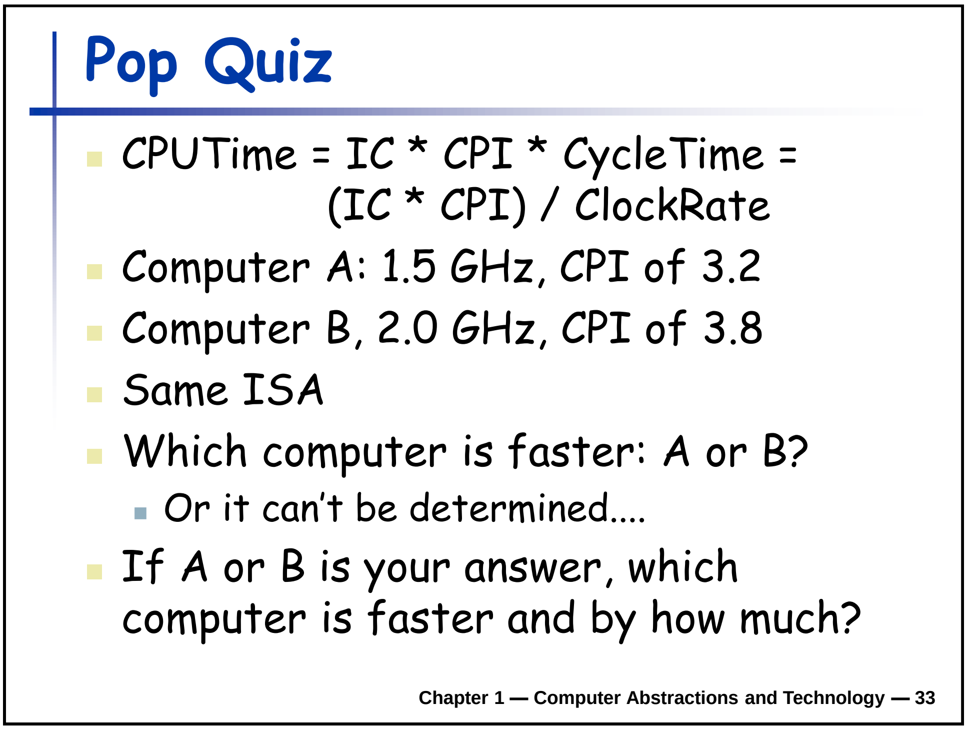
A is faster…

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

$$= \text{I} \times 1.2 \times 500\text{ps} = \text{I} \times 600\text{ps}$$

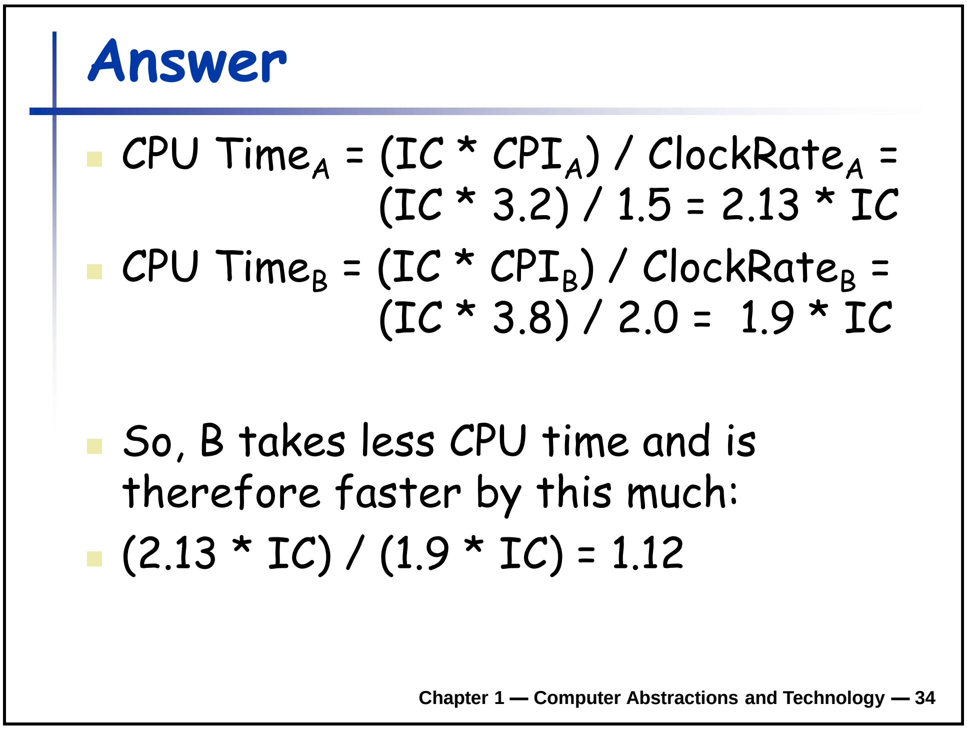
$$\frac{\text{CPU Time}_B}{\text{CPU Time}_A} = \frac{\text{I} \times 600\text{ps}}{\text{I} \times 500\text{ps}} = 1.2$$

…by this much



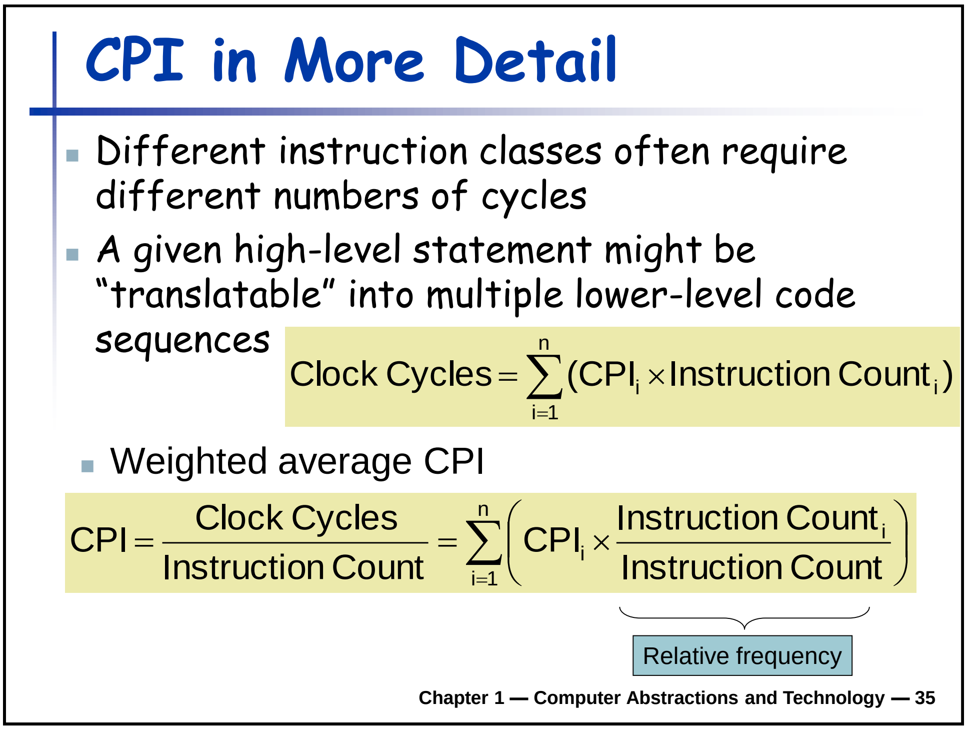
Pop Quiz

- CPUTime = IC * CPI * CycleTime =
  (IC * CPI) / ClockRate
- Computer A: 1.5 GHz, CPI of 3.2
- Computer B, 2.0 GHz, CPI of 3.8
- Same ISA
- Which computer is faster: A or B?
  - Or it can't be determined....
- If A or B is your answer, which computer is faster and by how much?



Answer

- CPU Time_A = (IC * CPI_A) / ClockRate_A =
  (IC * 3.2) / 1.5 = 2.13 * IC
- CPU Time_B = (IC * CPI_B) / ClockRate_B =
  (IC * 3.8) / 2.0 = 1.9 * IC
- So, B takes less CPU time and is therefore faster by this much:
- (2.13 * IC) / (1.9 * IC) = 1.12



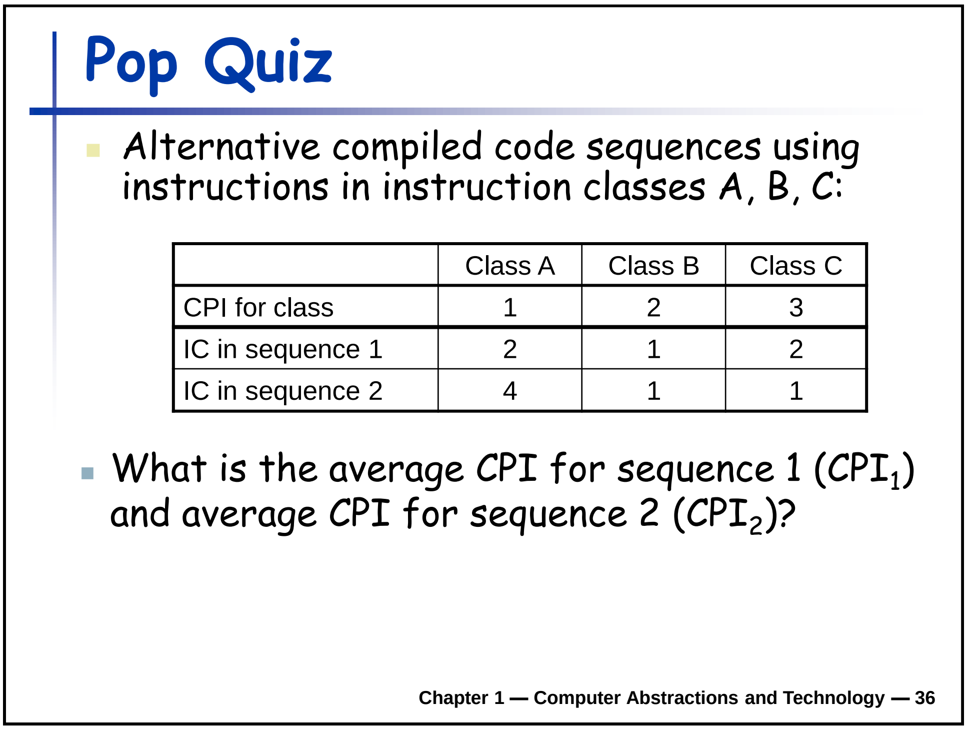
CPI in More Detail

- Different instruction classes often require different numbers of cycles
- A given high-level statement might be "translatable" into multiple lower-level code sequences

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

- Weighted average CPI

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

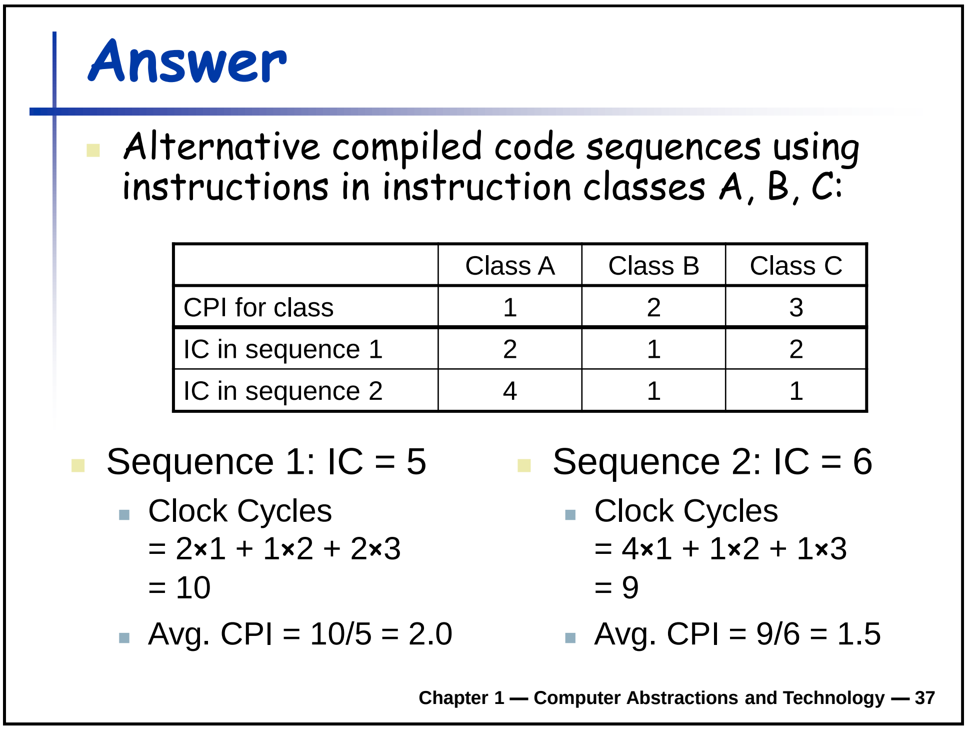


Pop Quiz

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

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

- What is the average CPI for sequence 1 (CPI_1) and average CPI for sequence 2 (CPI_2)?

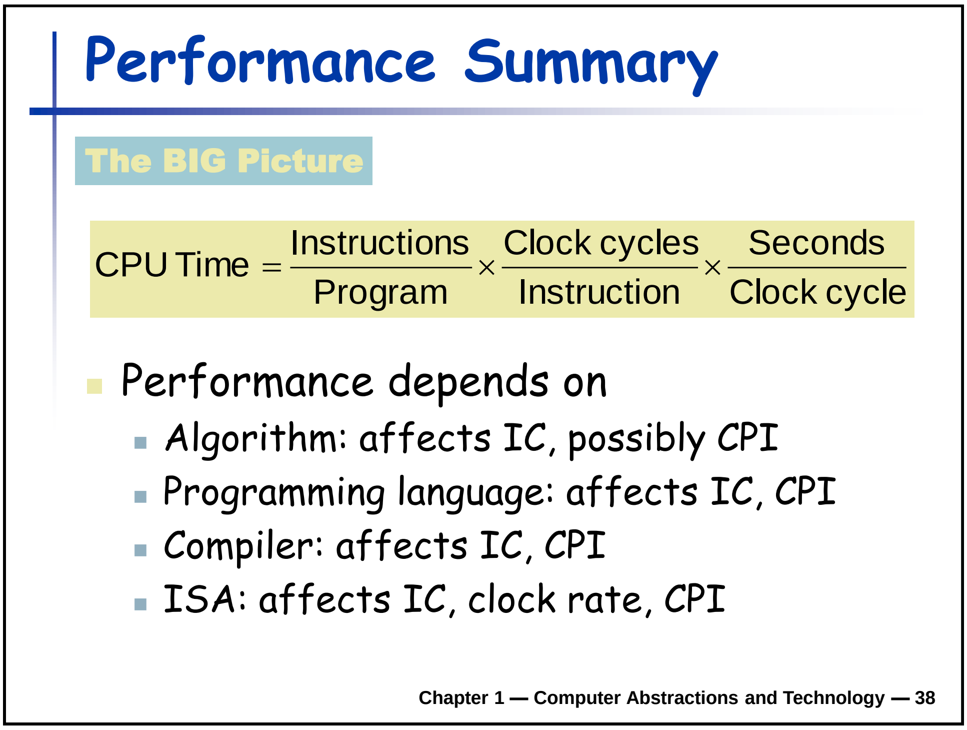


Answer

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

	Class A	Class B	Class 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
    = 2×1 + 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



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
  - ISA: affects IC, clock rate, CPI

Pitfall: Amdahl's Law

§1.10 Fallacies and Pitfalls

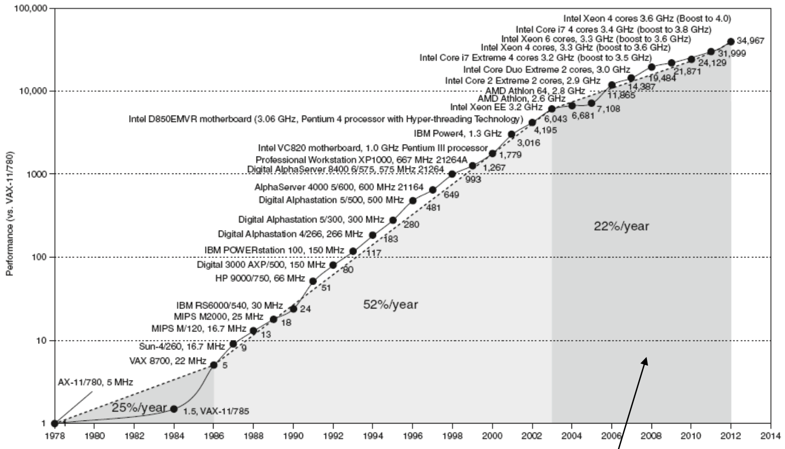
- Improving an aspect of a computer and then expecting a proportional improvement in overall performance….

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

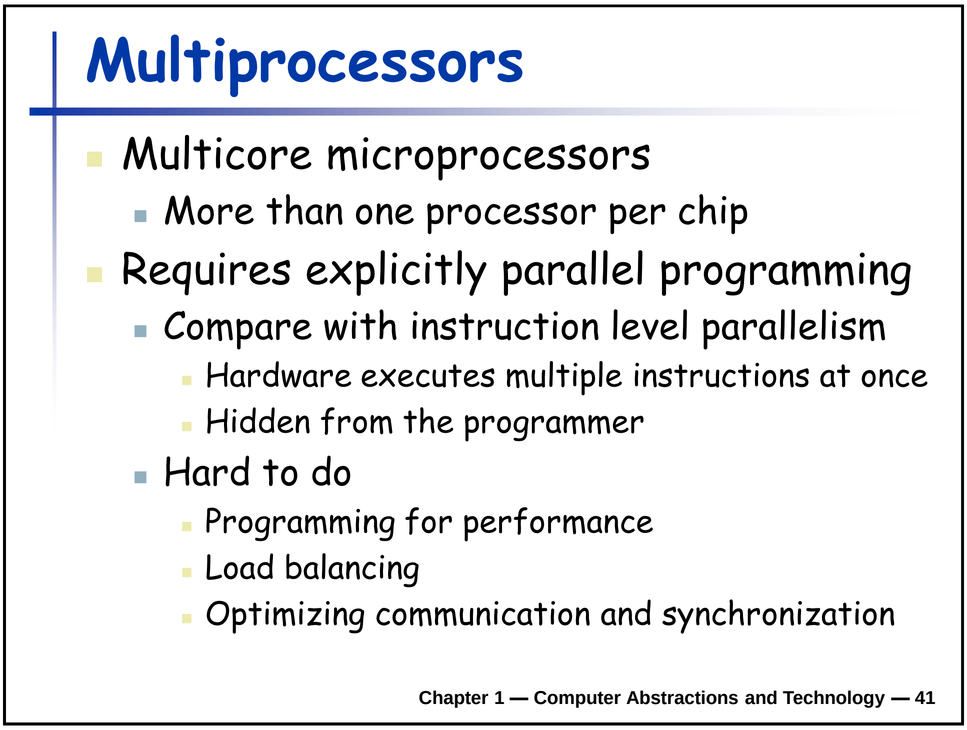
- Example: multiplication accounts for 80s of a program that runs for 100s
  - How much do I need to improve the speed of multiplication to run my program five times faster overall? $20 = \frac{80}{n} + 20$
  - Can't be done!
- **Corollary: make the common case fast**

Uniprocessor Performance

§1.8 The Sea Change: The Switch to Multiprocessors

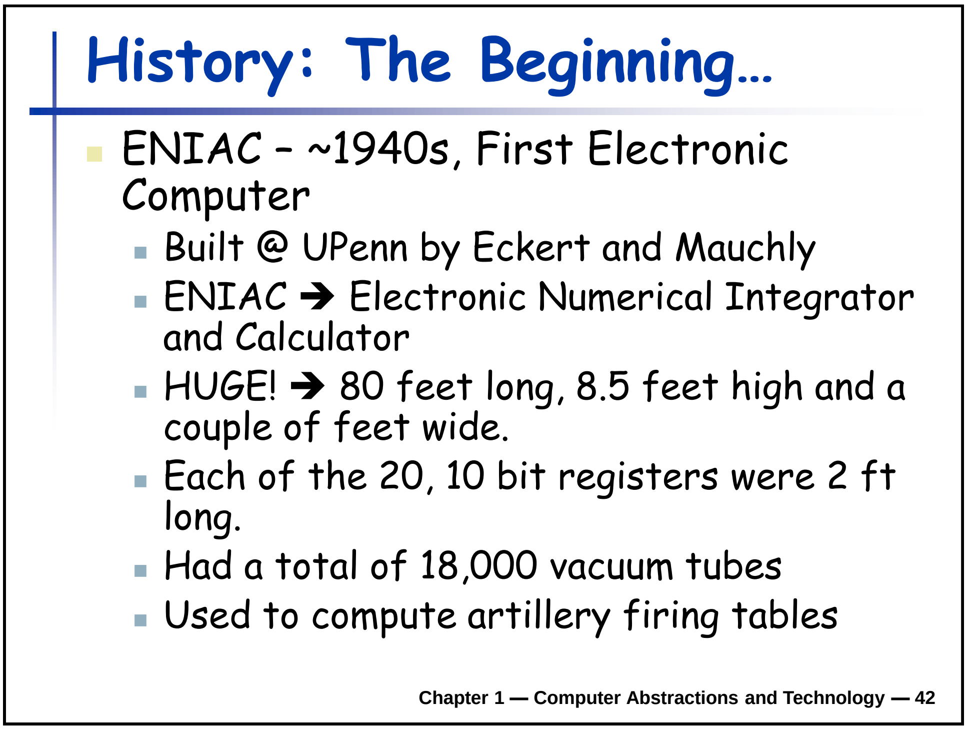


Constrained by power, instruction-level parallelism, memory latency



Multiprocessors

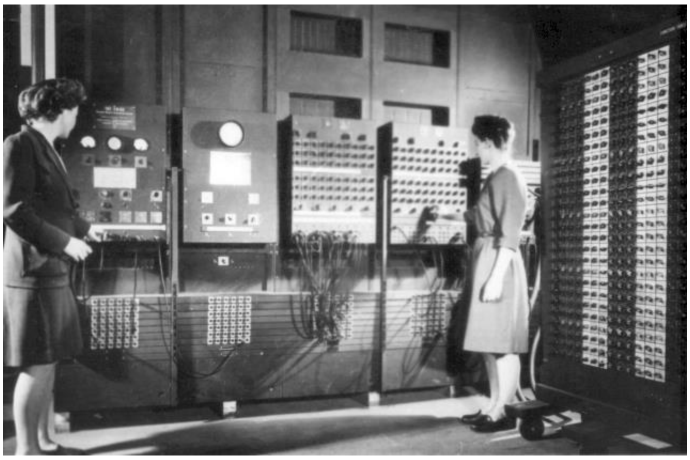
- Multicore microprocessors
  - More than one processor per chip
- Requires explicitly parallel programming
  - Compare with instruction level parallelism
    - Hardware executes multiple instructions at once
    - Hidden from the programmer
  - Hard to do
    - Programming for performance
    - Load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



History: The Beginning…

- ENIAC – ~1940s, First Electronic Computer
  - Built @ UPenn by Eckert and Mauchly
  - ENIAC ➔ Electronic Numerical Integrator and Calculator
  - HUGE! ➔ 80 feet long, 8.5 feet high and a couple of feet wide.
  - Each of the 20, 10 bit registers were 2 ft long.
  - Had a total of 18,000 vacuum tubes
  - Used to compute artillery firing tables

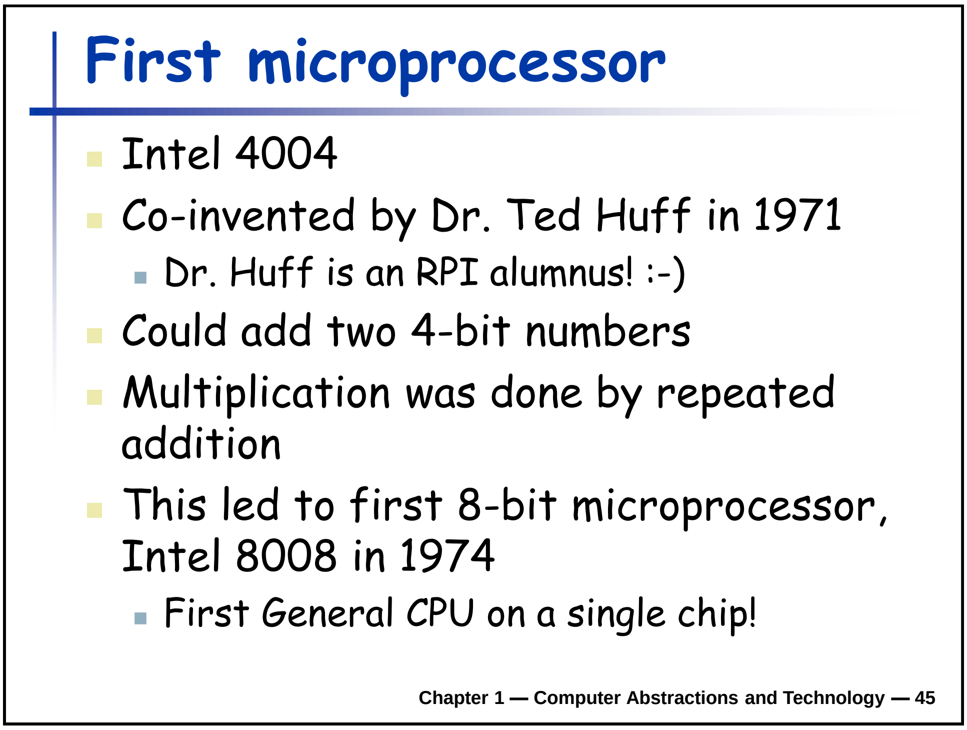
Pictures of ENIAC



Programmers/Early "Computer Geeks" Betty Jean Jennings (left) and Fran Bilas (right) operate the ENIAC's main control panel at the Moore School of Electrical Engineering. (U.S. Army photo from the archives of the ARL Technical Library)

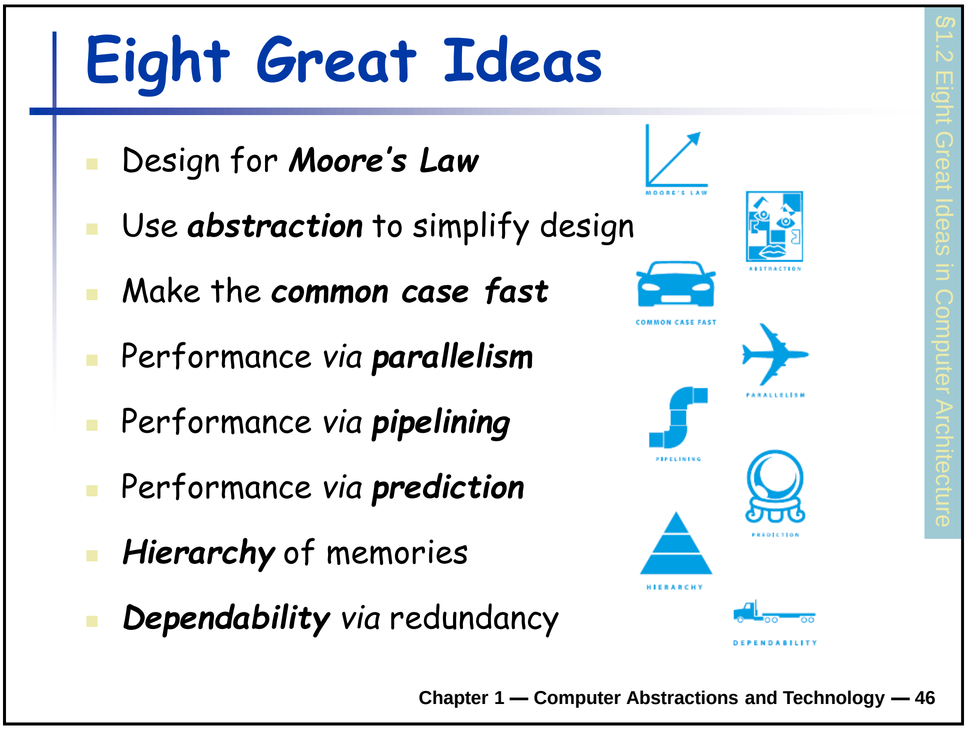
The von Neumann Computer

- 1944: John von Neumann proposes idea for a "stored program" computer – e.g., that a "program" can be stored in memory just like data…
- EDVAC – Electronic Discrete Variable Automatic Computer
- All modern day computer systems are built with this key concept!



First microprocessor

- Intel 4004
- Co-invented by Dr. Ted Huff in 1971
  - Dr. Huff is an RPI alumnus! :-)
- Could add two 4-bit numbers
- Multiplication was done by repeated addition
- This led to first 8-bit microprocessor, Intel 8008 in 1974
  - First General CPU on a single chip!



Eight Great Ideas

§1.2 Eight Great Ideas in Computer Architecture

- 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

MOORE'S LAW
ABSTRACTION
COMMON CASE FAST
PARALLELISM
PIPELINING
PREDICTION
HIERARCHY
DEPENDABILITY