
Chapter 1

Computer Abstractions and Technology

Classes of Computers

- Personal computers
 - General purpose, variety of software
 - Subject to cost/performance tradeoff
- Server computers (*=> cloud computing*)
 - Network based
 - High capacity, performance, reliability
 - Range from small servers to building sized
- 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





Notations Used in Textbook

- Figure 1.1

| Decimal term | Abbreviation | Value | Binary term | Abbreviation | Value | % Larger |
|--------------|--------------|-----------|-------------|--------------|----------|----------|
| kilobyte | KB | 10^3 | kibibyte | KiB | 2^{10} | 2% |
| megabyte | MB | 10^6 | mebibyte | MiB | 2^{20} | 5% |
| gigabyte | GB | 10^9 | gibibyte | GiB | 2^{30} | 7% |
| terabyte | TB | 10^{12} | tebibyte | TiB | 2^{40} | 10% |
| petabyte | PB | 10^{15} | pebibyte | PiB | 2^{50} | 13% |
| exabyte | EB | 10^{18} | exbibyte | EiB | 2^{60} | 15% |
| zettabyte | ZB | 10^{21} | zebibyte | ZiB | 2^{70} | 18% |
| yottabyte | YB | 10^{24} | yobibyte | YiB | 2^{80} | 21% |

- Processor Cores*

What You Will Learn

- How programs are translated into the machine language
 - And how the hardware executes them
- The hardware/software interface
 - How software can instruct hardware
- What determines **program performance**
 - And how it can be improved
- How hardware designers improve performance
- Also, **multicore, GPU, and parallel processing**

Understanding Program Performance

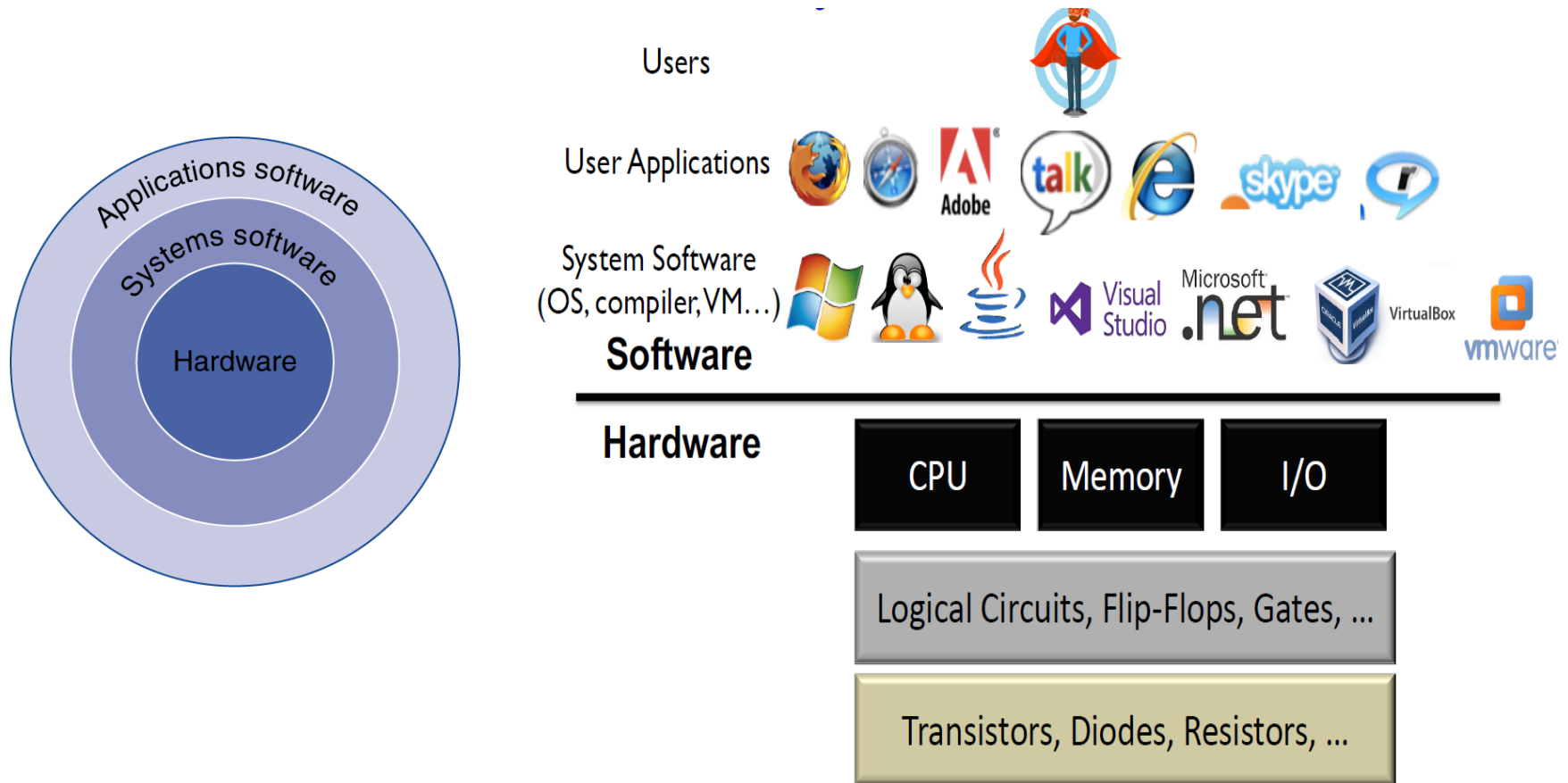
- Algorithm
 - Determines number of operations executed
- Programming language, compiler, architecture
 - Determine number of machine instructions executed per operation
- Processor and memory system
 - Determine how fast instructions are executed
- I/O system (including OS)
 - Determines how fast I/O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

Great Ideas in Computer Architecture

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



Below Your Program



Levels of Program Code

- High-level language
 - Level of abstraction closer to problem domain
 - Provides for productivity and portability
sort(), printf()
- Assembly language
 - Textual and symbolic representation of instructions
add, sub
- Machine code (object code or binary)
 - Binary digits of 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)

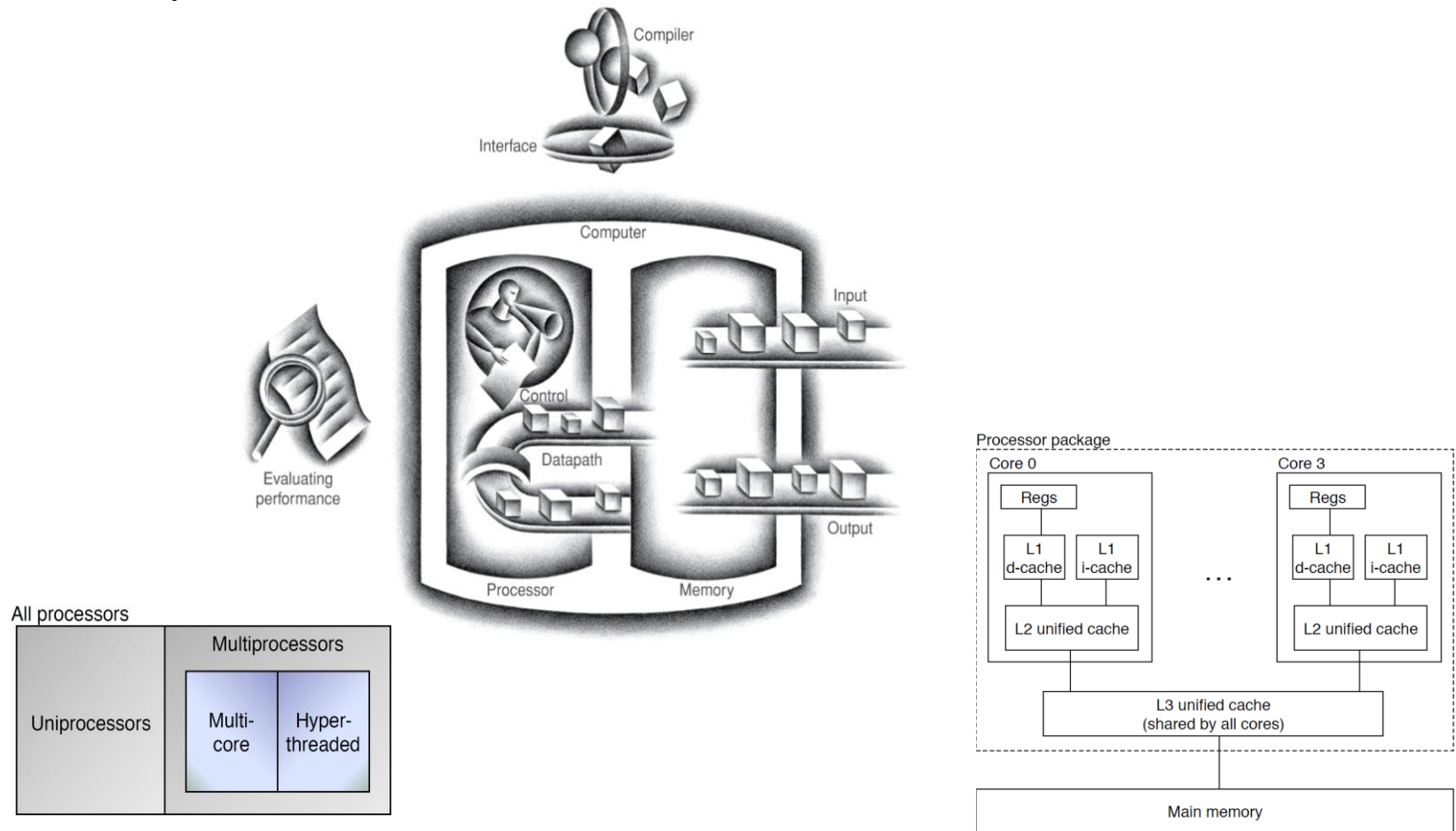
```
000000001010000100000000000011000
000000000000110000001100000100001
100011000110001000000000000000000
100011001111001000000000000000100
101011001111001000000000000000000
101011000110001000000000000000100
00000011111000000000000000001000
```

What kinds of data do we need to manipulate?

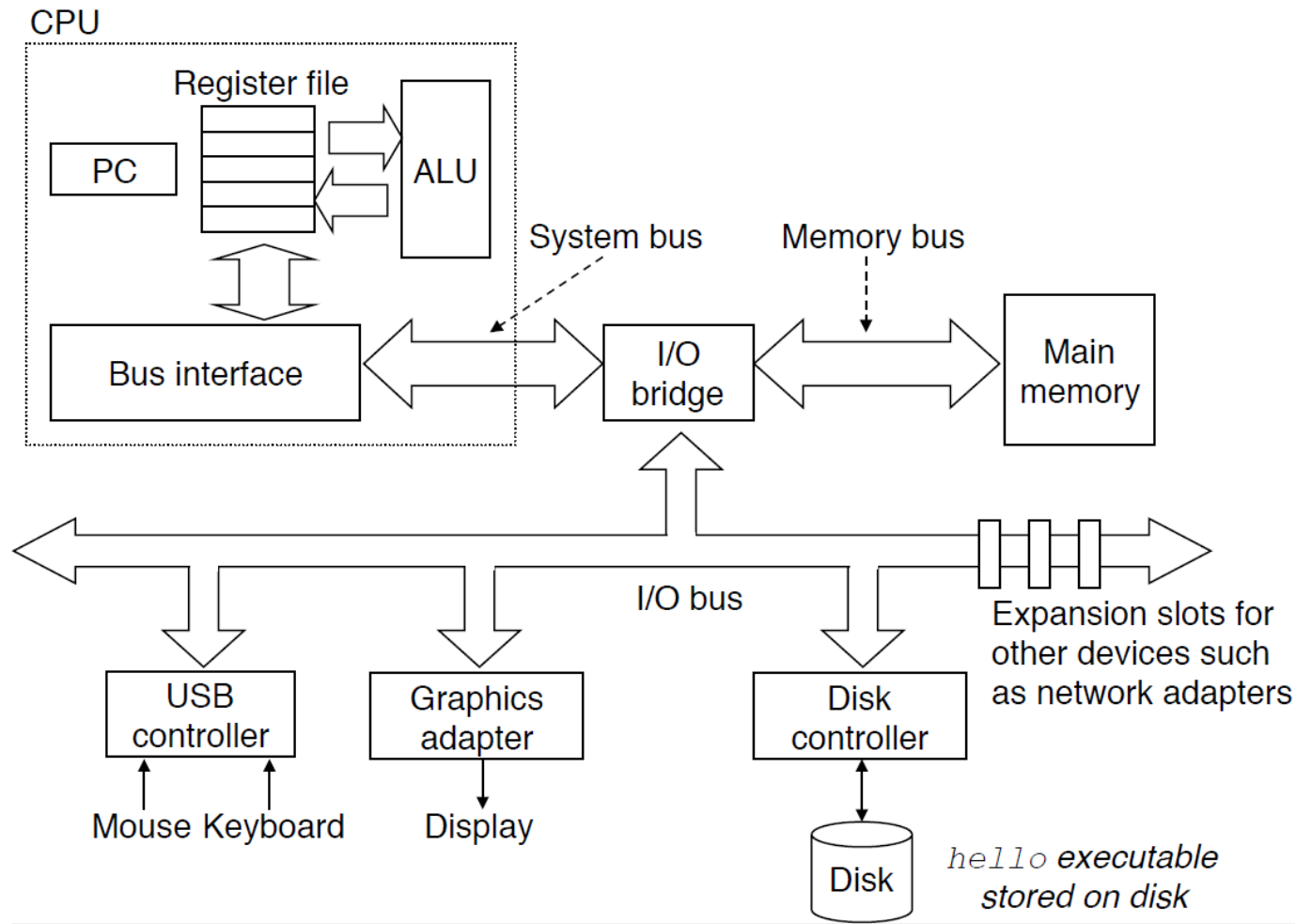
- Instructions
 - operations performed by computer
- Numbers
 - integers, floating point, complex, rational, irrational, ...
- Logical
 - true, false
- Text
 - characters, strings, ...
- Images
 - pixels, colors, shapes, ...
- Sound

Components of a Computer

- **Same components** for all kinds of computer
 - Desktop, server, embedded



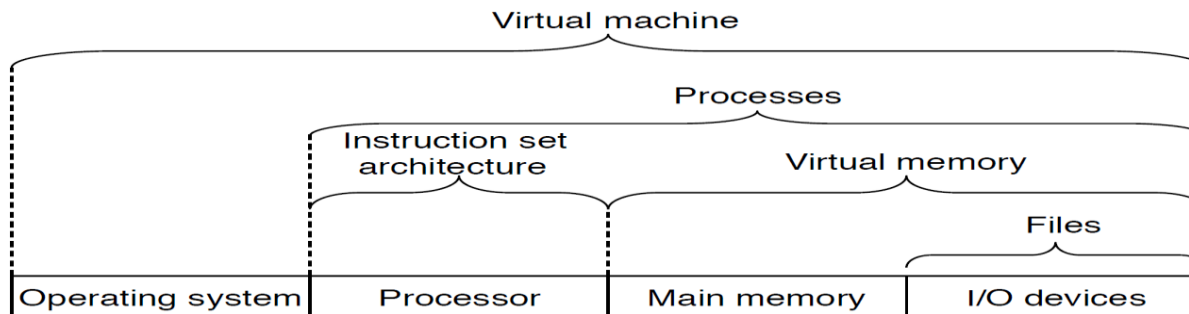
Hardware Organization of a Typical System



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



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...

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
 - $\text{Execution Time}_B / \text{Execution Time}_A$
 $= 15\text{s} / 10\text{s} = 1.5$
 - 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

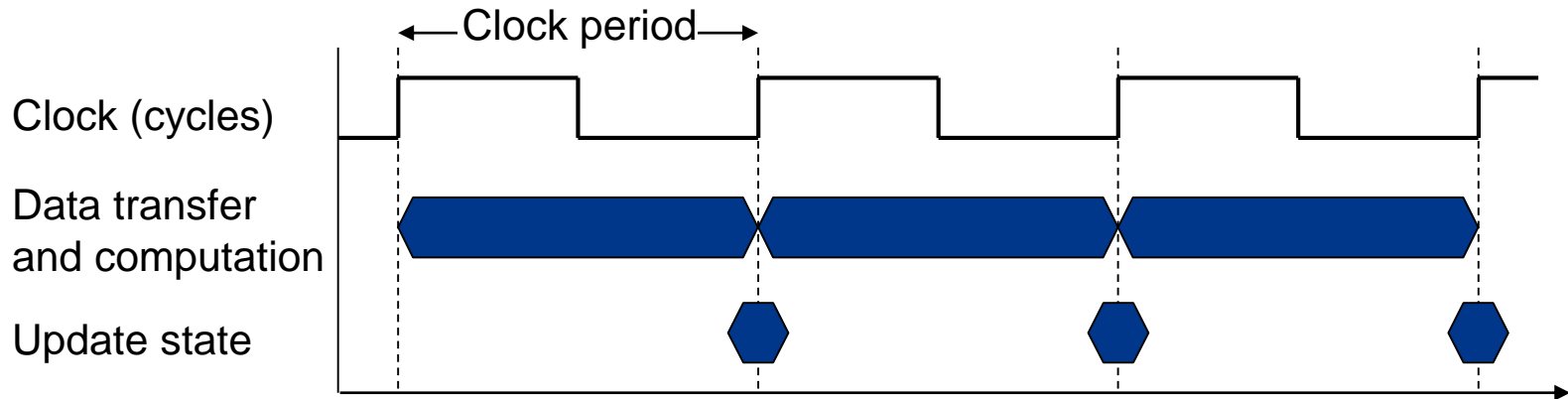
e.g. We are executing 'prog.sh' along with 'time' command.

| | | |
|-------------------|------|----------|
| \$ time ./prog.sh | Real | 0m8.803s |
| Initiating merge | User | 0m0.010s |
| Merge completed | Sys | 0m0.000s |

Check Yourself! (pp. 33)

CPU Clocking and CPU Time

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



- $$\text{CPU Time} = \text{clock cycle} \times \text{clock cycle time}$$
$$= IC \times CPI \times \text{clock cycle time}$$

Instruction Count for a program (IC)

Determined by program, ISA and compiler

Average cycles per instruction (CPI)

Determined by CPU hardware

If different instructions have different CPI

Average CPI affected by instruction mix

Comparing Code Segments (pp. 37)

- 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 = $9/6 = 1.5$

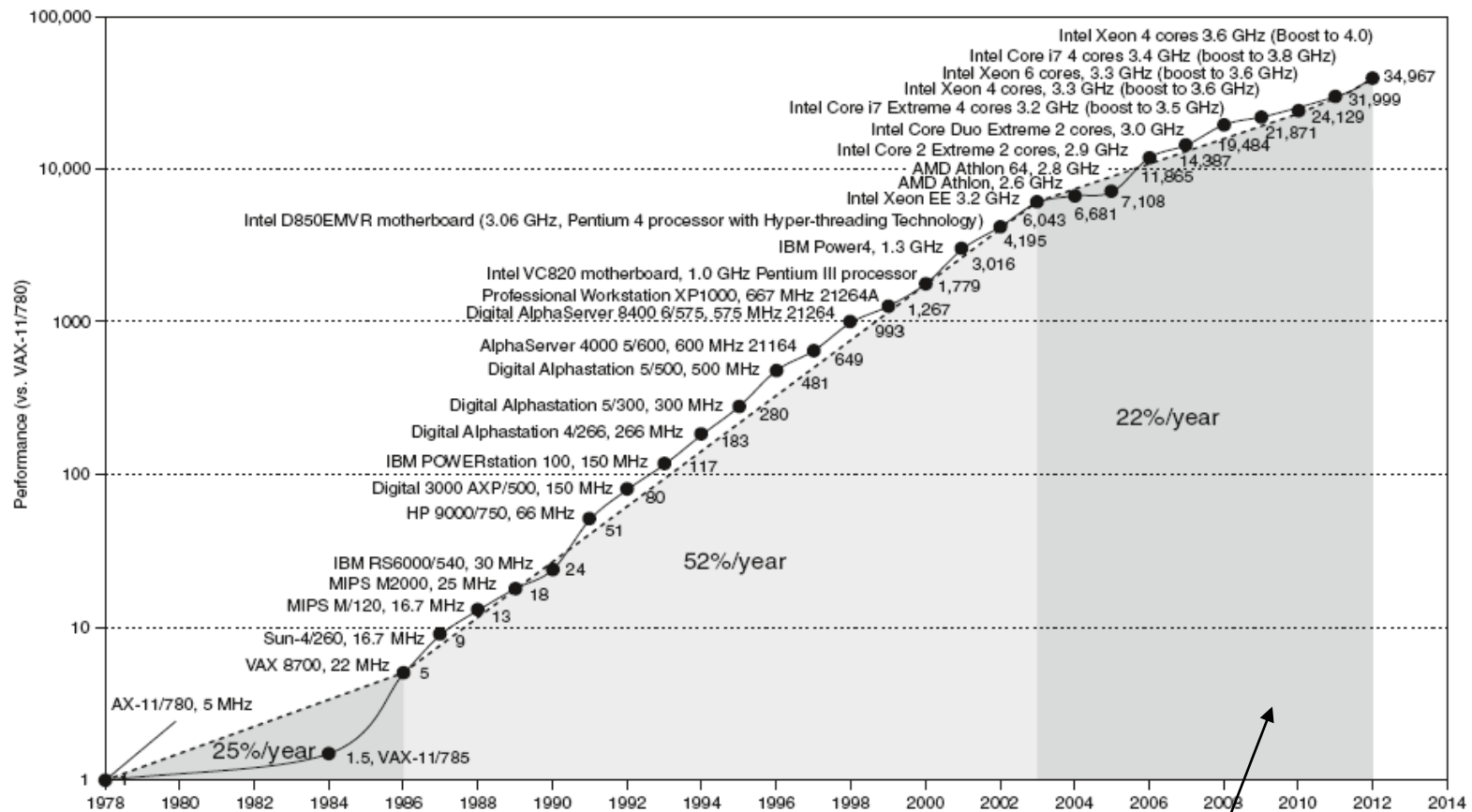
Performance Summary

The BIG Picture : Performance Equation (pp. 38)

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

- Performance depends on (pp. 39)
 - Algorithm: affects IC, possibly CPI
 - Programming language: affects IC, CPI
 - Compiler: affects IC, CPI
 - Instruction set architecture: affects IC, CPI, T_c
- Check Yourself! (pp. 40)

Processor Performance (Fig. 1. 17)



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

SPEC and TPC











- Programs used to measure performance
 - Supposedly typical of actual **workload**
- Standard Performance Evaluation Corp(SPEC)
 - <https://www.spec.org/> Figure 1. 18
 - **Embedded System** : <https://www.eembc.org>
EEMBC develops performance benchmarks for the hardware and software used in autonomous driving, mobile imaging, the Internet of Things, mobile devices, and many other applications. The EEMBC community includes **member companies**, **commercial licensees**, and **academic licensees** at institutions of higher learning around the world.
- Transaction Processing Performance Council (TPC)
 - Define **transaction processing** and **database** benchmarks
 - <http://www.tpc.org/>



Standard Performance Evaluation Corporation

[Home](#)[Benchmarks ▾](#)[Tools ▾](#)[Results ▾](#)[Contact](#)[Site Map](#)[Search](#)[Help](#)

Benchmarks

-  [Cloud](#)
-  [CPU](#)
-  [Graphics/Workstations](#)
-  [ACCEL/MPI/OMP](#)
-  [Java Client/Server](#)
-  [Mail Servers](#)
-  [Storage](#)
-  [Power](#)
-  [Virtualization](#)
-  [Web Servers](#)

Results Search

Submitting Results

[Cloud/CPU/Java/Power](#)
[SFS/Virtualization](#)
[ACCEL/MPI/OMP](#)
[SPECcapc/SPECviewperf/SPECwpc](#)

Tools

The Standard Performance Evaluation Corporation (SPEC) is a non-profit corporation formed to establish, maintain and endorse standardized benchmarks and tools to evaluate performance and energy efficiency for the newest generation of computing systems. SPEC develops benchmark suites and also reviews and publishes submitted results from our [member organizations](#) and other benchmark licensees.

What's New:

08/03/2020: The 12th ACM/SPEC International Conference on Performance Engineering, [ICPE 2021](#), is set to be held April 19-23, 2021, in Rennes, France.

07/03/2020: SPEC is [calling for nominations](#) for the Kaivalya Dixit Distinguished Dissertation Award 2020. The program is open to dissertations that have been defended between October 2019 and September 2020. Deadline for submissions is September 30, 2020.

06/03/2020: The serverless computing paradigm delivers major benefits for cloud applications — low-cost, fine-grained deployment, and management-free operation. Read more in this [SPEC Research Group report](#).

Summary

- Amdahl's Law (pp. 49)
- Performance Equation
- MIPS (Million Instructions Per Second)
- SPEC and TPC