UNIVERSITÀ DEGLI STUDI DI TRIESTE

Dipartimento di Ingegneria e Architettura

Computer Abstractions and Technology

A. Carini – Digital System Architectures

Introduction

In the last decades, there have been a number of new computers whose introduction appeared to revolutionize the computing industry; these revolutions were cut short only because someone else built an even better computer.
This race to innovate has led to unprecedented progress since the inception of electronic computing in the late 1940s.
Had the transportation industry kept pace with the computer industry, for example, today we could travel from New York to London in a second for a penny.
Each time the cost of computing improves by another factor of 10, the opportunities for computers multiply. Applications that were economically infeasible suddenly become practical.
In the recent past, the following applications were “computer science fiction”:

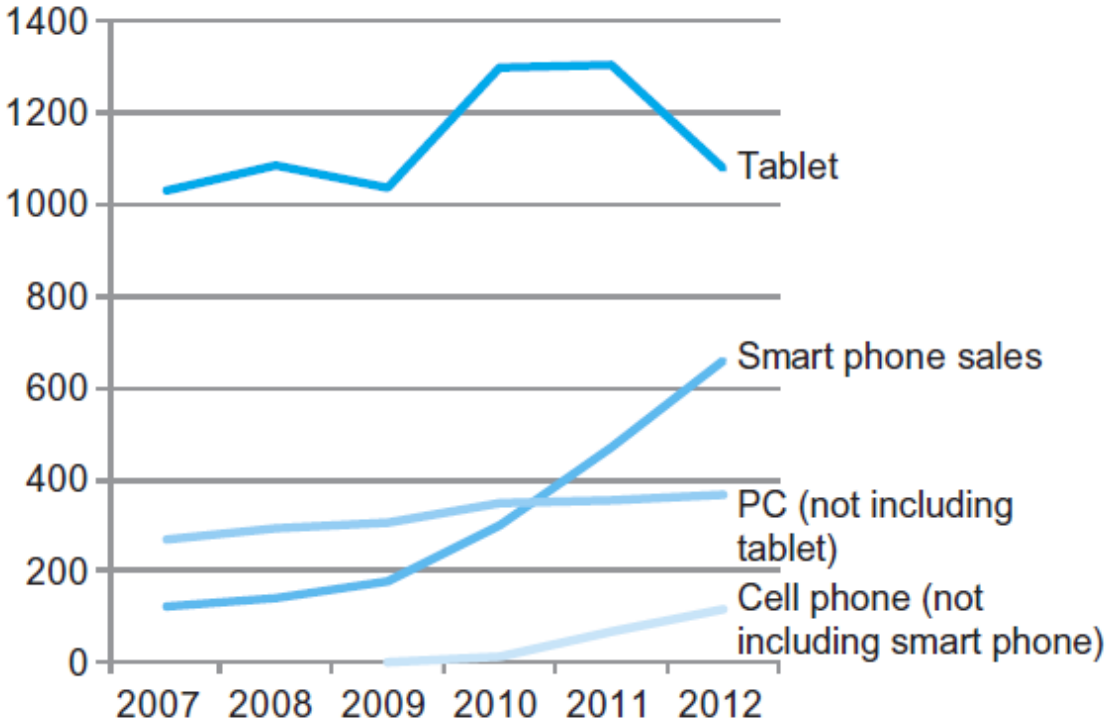
- Computers in automobiles
- Cell phones
- Human genome project
- World Wide Web
- Search Engines

Traditional Classes of Computing Applications

Computers are used in three dissimilar classes of applications:

- **Personal computer (PC)** A computer designed for use by an individual, usually incorporating a graphics display, a keyboard, and a mouse.
- **Server** A computer used for running larger programs for multiple users, often simultaneously, and typically accessed only via a network. Have high capacity, performance, reliability. Range from small servers to building sized.
- *Supercomputer* A class of computers with the highest performance and cost; they are configured as servers and typically cost tens to hundreds of millions of dollars. They represent a small fraction of the overall computer market.
- **Embedded computer** A computer inside another device used for running one predetermined application or collection of software.

The PostPC Era



The PostPC Era

- **Personal mobile devices (PMDs)** are small wireless devices to connect to the Internet; they rely on batteries for power, and software is installed by downloading apps. Conventional examples are smart phones and tablets.
- **Cloud Computing** refers to large collections of servers (in giant datacenters known as Warehouse Scale Computers (WSCs)) that provide services over the Internet; some providers rent dynamically varying numbers of servers as a utility.
- **Software as a Service (SaaS)** delivers software and data as a service over the Internet, usually via a thin program such as a browser that runs on local client devices, instead of binary code that must be installed, and runs wholly on that device. Examples include web search and social networking.

Definitions

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%

What You Will Learn

- **How are programs** written in a high-level language, such as C or Java, **translated into** the **machine language**, and how does the hardware execute the resulting program.
- What is the **interface between** the **software** and the **hardware**, 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.
- Since the first commercial computer in 1951, what great ideas did computer architects come up with that lay the **foundation of modern computing**.

Understanding Performance

The performance of a program depends on a combination of :

- **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**
  - Moore's Law states that integrated circuit resources double every 18–24 months.
  - As computer designs can take years, the resources available per chip can easily double or quadruple between the start and finish of the project.
  - Computer architects must anticipate where the technology will be when the design finishes rather than design for where it starts.

- **Use abstraction to simplify design**
  - A major productivity technique for hardware and software is to use abstractions to characterize the design at different levels of representation; lower-level details are hidden to offer a simpler model at higher levels.

- **Make the common case fast**
  - Making the common case fast will tend to enhance performance better than optimizing the rare case.

Eight Great Ideas

- **Performance via parallelism**
  - Since the dawn of computing, computer architects have offered designs that get more performance by computing operations in parallel.

- **Performance via pipelining**
  - A particular pattern of parallelism.
  - Divide operations in small stages, e.g. fetch, decode, execute. While instruction i is executed, i+1 is decoded, i+2 is fetched.

- **Performance via prediction**
  - It can be faster on average to guess and start working rather than wait until you know for sure, assuming that the mechanism to recover from a misprediction is not too expensive and your prediction is relatively accurate.

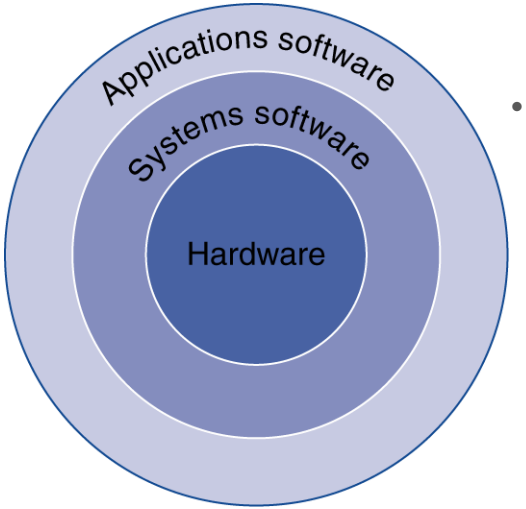
Eight Great Ideas

- **Hierarchy of memories**
  - Programmers want the memory to be fast, large, and cheap, as memory speed often shapes performance, capacity limits the size of problems that can be solved, and the cost of memory today is often the majority of computer cost.
  - We can address these conflicting demands with a hierarchy of memories, with the fastest, smallest, and the most expensive memory per bit at the top of the hierarchy and the slowest, largest, and cheapest per bit at the bottom.
  - Caches give the illusion that main memory is almost as fast as the top of the hierarchy and nearly as big and cheap as the bottom of the hierarchy

Affidabilità mediante ridondanza

- **Dependability via redundancy**
  - Computers not only need to be fast; they need to be dependable.
  - Since any physical device can fail, we make systems dependable by including redundant components that can take over when a failure occurs and to help detect failures.

Below Your Program

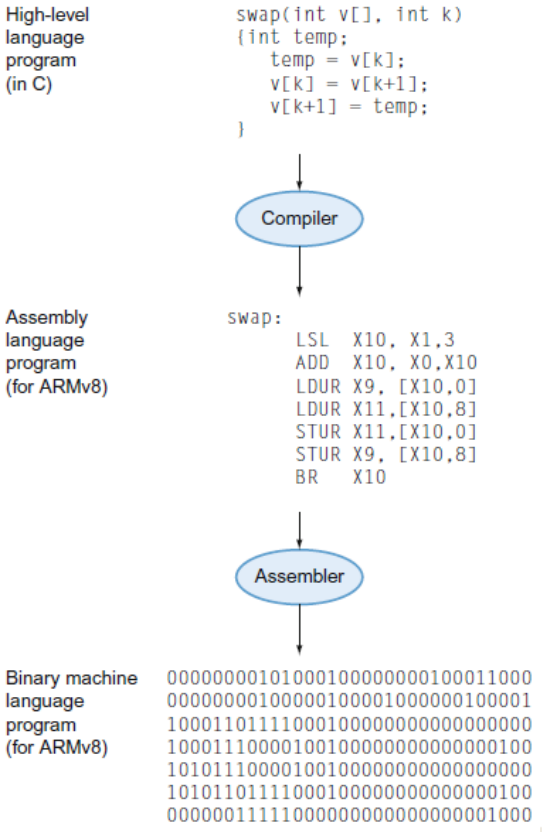


- **Application software**
  - Millions of code lines
- **System software**
  - Software that provides services that are commonly useful, including operating systems, compilers, loaders, and assemblers.
  - Compiler: translates HLL code to machine code.
  - Operating System: Supervising program that manages the resources of a computer for the benefit of the programs that run on that computer.
    - Handling input/output
    - Managing memory and storage
    - Scheduling tasks & sharing resources
- **Hardware**
  - Processor, memory, I/O controllers

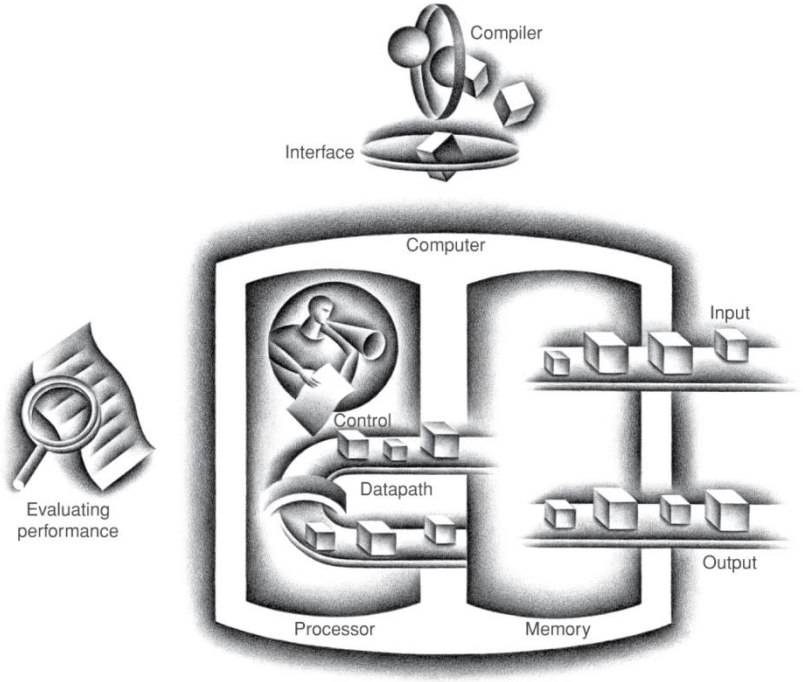
From a High-Level Language to the Language of Hardware

- **instruction** A command that computer hardware understands and obeys.
- **assembler** A program that translates a symbolic version of instructions into the binary version.
- **assembly language** A symbolic representation of machine instructions.
- **machine language** A binary representation of machine instructions.

- **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 five classic components of a computer are **input**, **output**, **memory**, **datapath**, and **control**, with the last two sometimes combined and called the **processor**, or central processing unit CPU.
- This organization is independent of hardware technology: you can place every piece of every computer, past and present, into one of these five categories.

- **central processor unit (CPU)** Also called processor. The active part of the computer, which contains the datapath and control and which adds numbers, tests numbers, signals I/O devices to activate, and so on.

Inside the Processor (CPU)

- **datapath**
  - The component of the processor that performs arithmetic operations.

- **control**
  - The component of the processor that commands the datapath, memory, and I/O devices according to the instructions of the program.

Memory

- **memory**
  - The storage area in which programs are kept when they are running and that contains the data needed by the running programs.
- **dynamic random access memory (DRAM)**
  - Memory built as an integrated circuit; it provides random access to any location. Access times are 50 nanoseconds
- **cache memory**
  - A small, fast memory that acts as a buffer for a slower, larger memory. Typically SRAM.
- **static random access memory (SRAM)**
  - Also memory built as an integrated circuit, but faster and less dense than DRAM.

- SRAM and DRAM are volatile memories: they are used to hold data and programs while they are running; but we need nonvolatile memory used to store programs and data between runs.

Primary and Secondary memories

- We will distinguish between
  - **main memory** (also called **primary memory**) memory used to hold programs while they are running; typically consists of DRAM in today's computers.
  - **secondary memory** Nonvolatile memory used to store programs and data between runs; typically consists of flash memory in PMDs and SSDs and magnetic disks in servers.

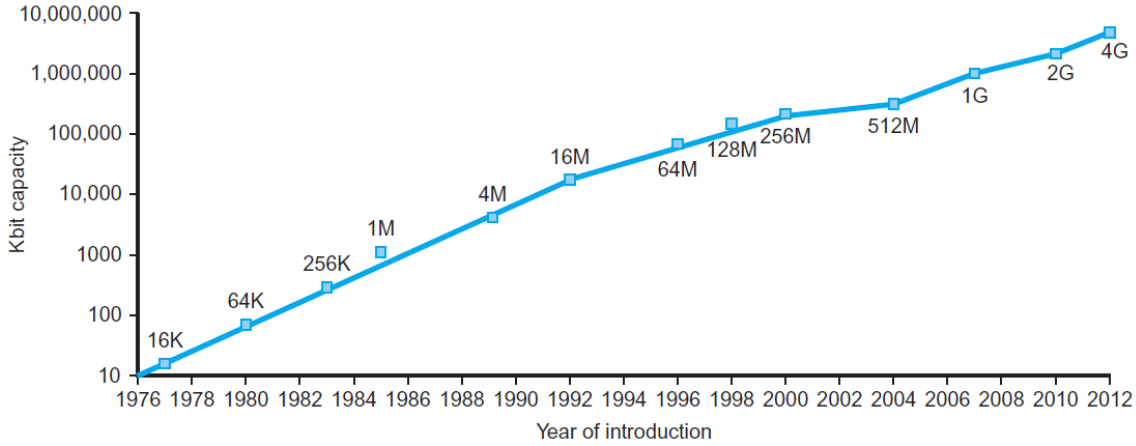
- **magnetic disk** Also called hard disk. A form of nonvolatile secondary memory composed of rotating platters coated with a magnetic recording material. Because they are rotating mechanical devices, access times are about 5 to 20 milliseconds.
- **flash memory** A nonvolatile semiconductor memory. It is cheaper and slower than DRAM but more expensive per bit and faster than magnetic disks. Access times are about 5 to 50 microseconds.

Abstractions

- One of the most important abstractions is the interface between the hardware and the lowest-level software: the **instruction set architecture (ISA)**, or simply **architecture**, of a computer.
- The instruction set architecture includes anything programmers need to know to make a binary machine language program work correctly, including instructions, I/O devices, and so on.
- Typically, the operating system will encapsulate the details of doing I/O, allocating memory, and other low-level system functions.
- **application binary interface (ABI)** The user portion of the instruction set plus the operating system interfaces used by application programmers. (ISA + system SW interface). It defines a standard for binary portability across computers.
- Note that we distinguish the instruction set architecture from an **implementation** of the architecture: an implementation is hardware that obeys the architecture abstraction.
- This abstract interface enables **many implementations** of varying cost and performance to **run identical software**.

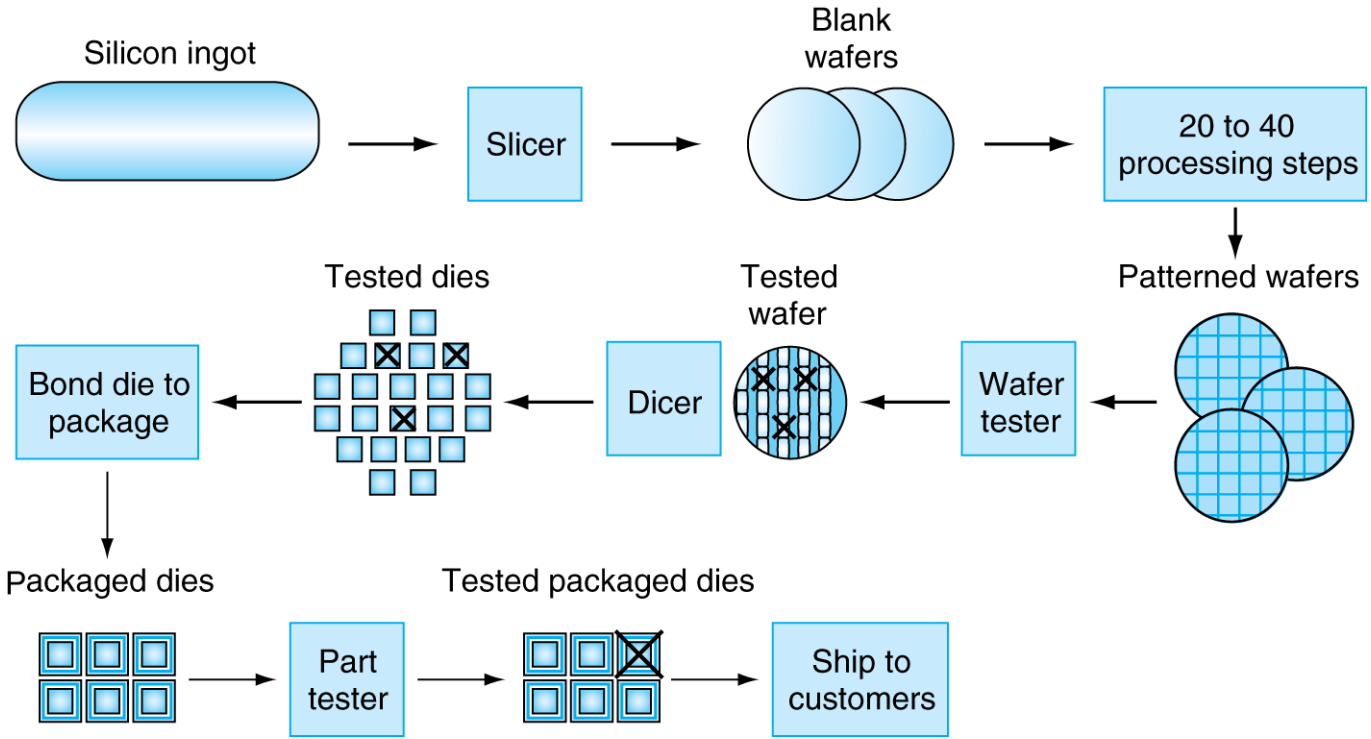
Technologies for building processors and memories

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



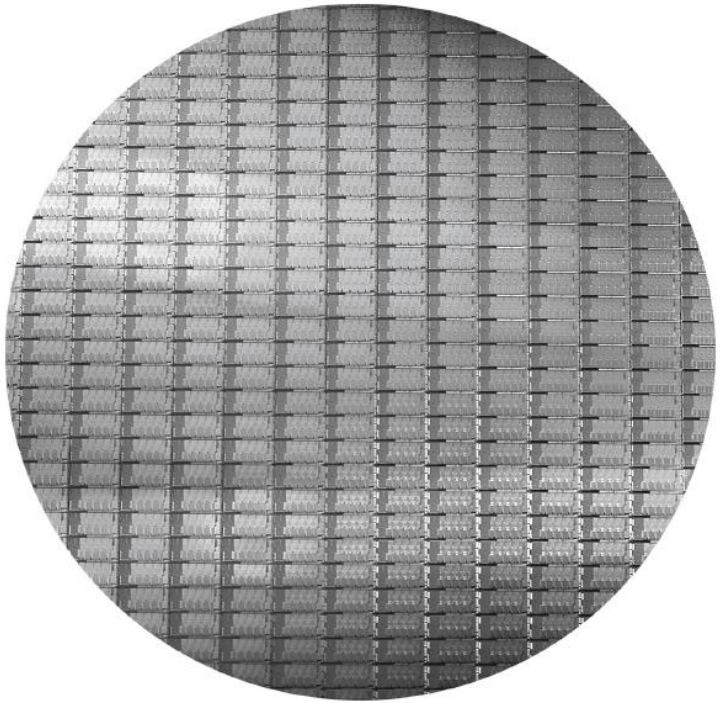
DRAM capacity

Technologies for building processors and memories



- **Yeld** the percentage of good dies from the total number of dies on the wafer.

Technologies for building processors and memories



- Intel Core i7 wafer (2012)
- 300mm wafer, 280 chips, 32nm technology
- Each chip is 20.7 x 10.5mm

Technologies for building processors and memories

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

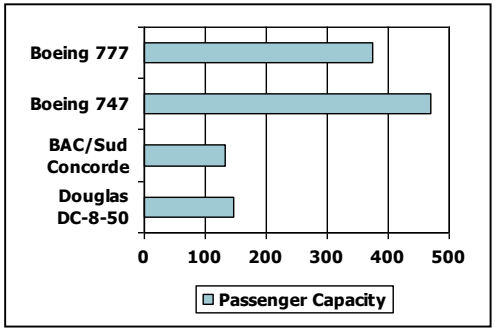
$$\text{Dies per wafer} \approx \text{Wafer area}/\text{Die area}$$

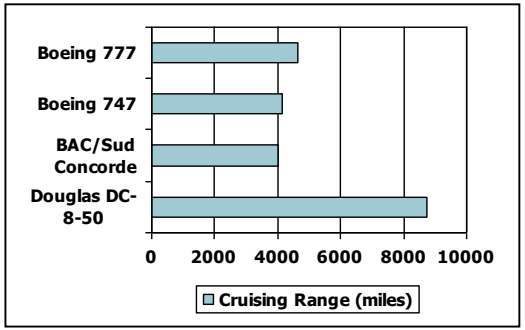
$$\text{Yield} = \frac{1}{(1+(\text{Defects per area} \times \text{Die area}/2))^2}$$

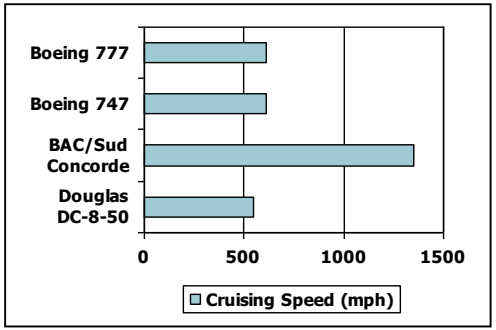
- Nonlinear relation to area and defect rate
  - Wafer cost and area are fixed
  - Defect rate determined by manufacturing process
  - Die area determined by architecture and circuit design

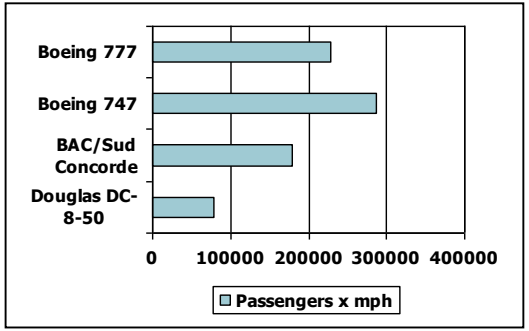
Defining performance

- When we say one computer has better performance than another, what do we mean?
- Which of the following airplanes has the best performance?









Response Time and Throughput

- If you were running a program on two different desktop computers, you'd say that the faster one is the desktop computer that gets the job done first.
- If you were running a datacenter that had several servers running jobs submitted by many users, you'd say that the faster computer was the one that completed the most jobs during a day.
- **response time** Also called **execution time**.
  - The total time required for the computer to complete a task, including disk accesses, memory accesses, I/O activities, operating system overhead, CPU execution time, and so on.
- **throughput** Also called **bandwidth**.
  - Another measure of performance, it is the number of tasks completed per unit time.

- 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 as $$\text{Performance}_X = \frac{1}{\text{Execution time}_X}$$
- With two computers
$$\text{Performance}_X > \text{Performance}_Y$$
$$\frac{1}{\text{Execution time}_X} > \frac{1}{\text{Execution time}_Y}$$
$$\text{Execution time}_Y > \text{Execution time}_X$$
- "X is *n* time faster than Y" if
$$\frac{\text{Performance}_X}{\text{Performance}_Y} = \frac{\text{Execution time}_Y}{\text{Execution time}_X} = n$$

Example of relative performance

- Time taken to run a program
  - 10s on A, 15s on B
  - Execution TimeB / Execution TimeA = 15s / 10s = 1.5
  - So A is 1.5 times faster than B

Measuring Performance

- Time is the measure of computer performance: the computer that performs the same amount of work in the least time is the fastest.
- However, time can be defined in different ways, depending on what we count.
- **wall clock time**, **response time**, or **elapsed time**.
  - total time to complete a task, including disk accesses, memory accesses, input/output (I/O) activities, operating system overhead—everything.
- **CPU execution time** Also called **CPU time**.
  - The actual time the CPU spends computing for a specific task.
  - Does not include time spent waiting for I/O or running other programs.
- CPU time can be further divided into
  - **user CPU time** The CPU time spent in a program itself.
  - **system CPU time** The CPU time spent in the operating system performing tasks on behalf of the program.
- Different programs are affected differently by CPU and system performance.

CPU Clocking

- All computers are constructed using a **clock** that determines when events take place in the hardware.
- **clock period** is the time for a complete clock cycle (e.g., 250 picoseconds, or 250 ps).
- **clock frequency** or **clock rate** is the inverse of the clock period (e.g., 4 gigahertz, or 4 GHz).
- A simple formula relates the clock cycles and clock cycle time to CPU time:

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

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

- This formula makes it clear that the hardware designer can improve performance by reducing the number of clock cycles required for a program or the length of the clock cycle.
- 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 × clock cycles
- How fast must Computer B clock be?

$$\text{CPU time}_A = \frac{\text{CPU clock cycles}_A}{\text{Clock rate}_A} \quad \Longrightarrow \quad 10\,\text{seconds} = \frac{\text{CPU clock cycles}_A}{2 \times 10^9 \,\frac{\text{cycles}}{\text{second}}}$$

$$\text{CPU clock cycles}_A = 10\,\text{seconds} \times 2 \times 10^9 \,\frac{\text{cycles}}{\text{second}} = 20 \times 10^9 \,\text{cycles}$$

$$\text{CPU time}_B = \frac{1.2 \times \text{CPU clock cycles}_A}{\text{Clock rate}_B} \quad \Longrightarrow \quad 6\ \text{seconds} = \frac{1.2 \times 20 \times 10^9\ \text{cycles}}{\text{Clock rate}_B}$$

$$\text{Clock rate}_B = \frac{1.2 \times 20 \times 10^9\ \text{cycles}}{6\ \text{seconds}} = \frac{0.2 \times 20 \times 10^9\ \text{cycles}}{\text{second}} = \frac{4 \times 10^9\ \text{cycles}}{\text{second}} = 4\ \text{GHz}$$

Instruction Performance

- The previous performance equations did not include any reference to the number of instructions needed for the program. The execution time must depend on the number of instructions in a program.
- One way to think about execution time is that it equals the number of instructions executed multiplied by the average time per instruction.

$$\text{CPU clock cycles} = \text{Instructions for a program} \times \frac{\text{Average clock cycles}}{\text{per instruction}}$$

- **clock cycles per instruction (CPI)**
  - Average number of clock cycles per instruction for a program or program fragment.
- Since different instructions may take different amounts of time depending on what they do, CPI is an average of all the instructions executed in the program.

CPI Example

- Computer A has a clock cycle time of 250 ps and a CPI of 2.0 for some program.
- 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?

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

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

Classic CPU Performance Equation

- We can now write the basic performance equation in terms of instruction count (the number of instructions executed by the program), CPI, and clock cycle time:

$$\text{CPU time} = \text{Instruction count} \times \text{CPI} \times \text{Clock cycle time}$$

- or, since the clock rate is the inverse of clock cycle time:

$$\text{CPU time} = \frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}}$$

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

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

Example: comparing code segments

- A compiler designer is trying to decide between two code sequences for a computer.

	CPI for each instruction class		
	A	B	C
CPI	1	2	3

Code sequence	Instruction counts for each instruction class			
	A	B	C	
1	2	1	2	→ 5 instructions
2	4	1	1	→ 6 instructions

$$\text{CPU clock cycles}_1 = (2 \times 1) + (1 \times 2) + (2 \times 3) = 2 + 2 + 6 = 10 \text{ cycles}$$
$$\text{CPU clock cycles}_2 = (4 \times 1) + (1 \times 2) + (1 \times 3) = 4 + 2 + 3 = 9 \text{ cycles}$$

$$\text{CPI}_1 = \frac{\text{CPU clock cycles}_1}{\text{Instruction count}_1} = \frac{10}{5} = 2.0$$

$$\text{CPI}_2 = \frac{\text{CPU clock cycles}_2}{\text{Instruction count}_2} = \frac{9}{6} = 1.5$$

Performance Summary

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

- Performance depends on
- **Algorithm**
  - Determines the number of source program instructions executed and hence the number of processor instructions executed. May also affect the CPI favoring slower or faster instructions.
- **Programming language**
  - Affects the instruction count, since statements in the language are translated to processor instructions, which determine instruction count. May also affect the CPI because of its features, e.g. heavy support data abstraction requires indirect calls, which use higher CPI instructions.
- **Compiler**
  - Affects both the instruction count and average cycles per instruction, since it determines the translation of the source language instructions into computer instructions.
- **Instruction set architecture**
  - Affects all three aspects of CPU performance, since it affects the instructions needed for a function, the cost in cycles of each instruction, and the overall clock rate of the processor.

The power wall

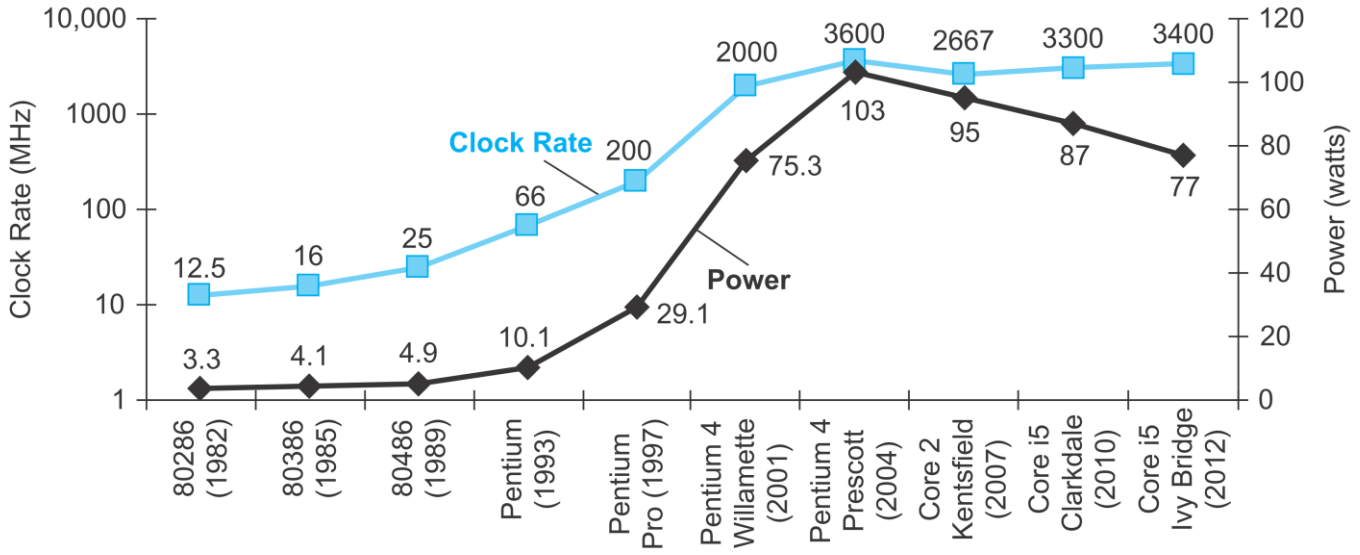


FIGURE 1.16 Clock rate and Power for Intel x86 microprocessors over eight generations and 30 years. The Pentium 4 made a dramatic jump in clock rate and power but less so in performance. The Prescott thermal problems led to the abandonment of the Pentium 4 line. The Core 2 line reverts to a simpler pipeline with lower clock rates and multiple processors per chip. The Core i5 pipelines follow in its footsteps.

Power dissipation in CMOS

- For CMOS, the primary source of energy consumption is so-called dynamic energy—that is, energy that is consumed when gates switch states from 0 to 1 and vice versa.
- For 0 -> 1 -> 0:

$$Energy \propto Capacitive\ load \times Voltage^2$$

- For a single transition:

$$Energy \propto 1/2 \times Capacitive\ load \times Voltage^2$$

- The power required per gate is just the product of energy of a transition and the frequency of transitions, which depends on the clock frequency:

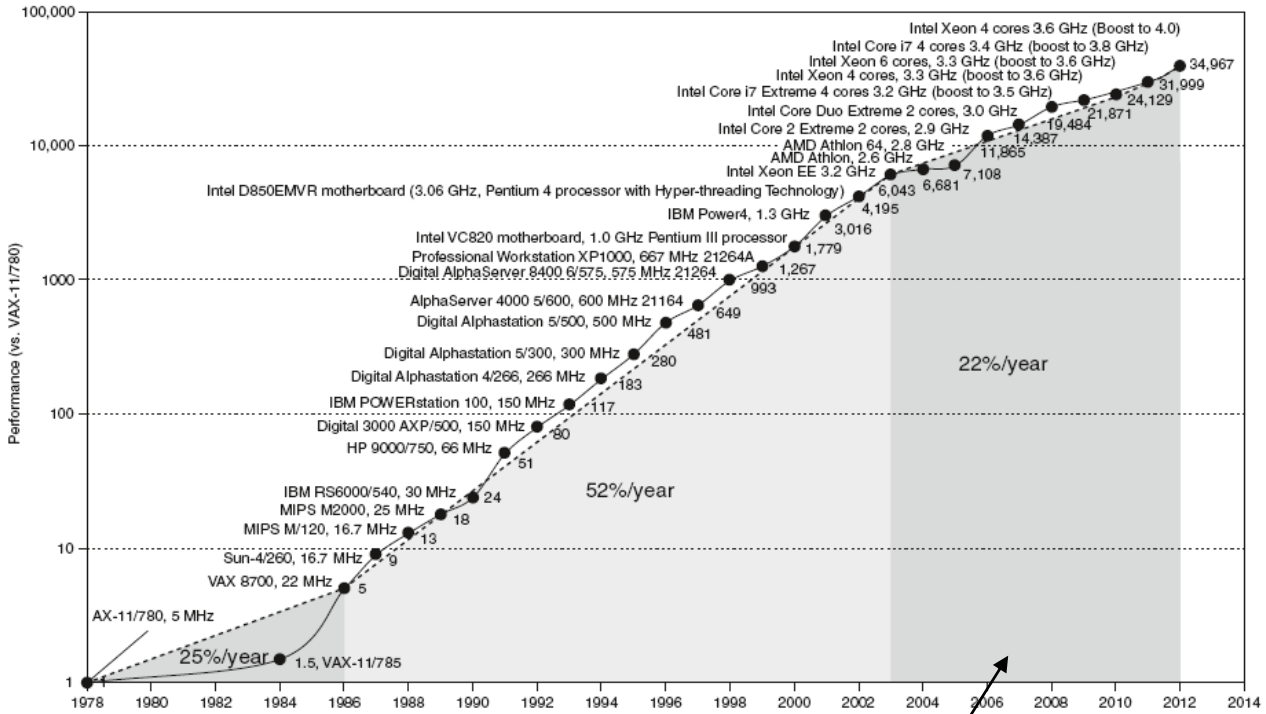
$$Power \propto \square\ Capacitive\,load \times Voltage^2 \times Frequency\,switched$$

- How could clock rates grow by a factor of 1000 while power increased by only a factor of 30?
- Energy and power have been reduced by lowering the voltage, which occurred with each new generation of technology, passing from 5V till values below 1V.

Modern problems in power dissipation

- The modern problem is that further lowering of the voltage appears to make the transistors too leaky.
- Although dynamic energy is the primary source of energy consumption in CMOS, static energy consumption occurs because of leakage current that flows even when a transistor is off.
- In servers, leakage is typically responsible for 40% of the energy consumption.
- Increasing the number of transistors increases power dissipation, even if the transistors are always off.
- A variety of design techniques and technology innovations are being deployed to control leakage, but it’s hard to lower voltage further.

The Switch from Uniprocessors to Multiprocessors



Constrained by power, instruction-level parallelism, memory latency

Multiprocessors

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

SPEC CPU Benchmark

- Performance of processors is measured using benchmark programs supposedly typical of actual workload
- **SPEC (System Performance Evaluation Cooperative)** is an effort funded and supported by a number of computer vendors to create standard sets of benchmarks for modern computer systems.
  - Develops benchmarks for CPU, I/O, Web, …
- **SPEC CPU2006** consists of a set of 12 integer benchmarks (CINT2006) and 17 floating-point benchmarks (CFP2006).
  - 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

$$\sqrt[n]{\prod_{i=1}^{n} \text{Execution time ratio}_i}$$

CINT2006 for Intel Core i7 920

Description	Name	Instruction Count x 10^9	CPI	Clock cycle time (seconds x 10^{-9})	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

FIGURE 1.18 SPECINTC2006 benchmarks running on a 2.66 GHz Intel Core i7 920. As the equation on page 36 explains, execution time is the product of the three factors in this table: instruction count in billions, *clocks per instruction* (CPI), and clock cycle time in nanoseconds. SPECratio is simply the reference time, which is supplied by SPEC, divided by the measured execution time. The single number quoted as SPECINTC2006 is the geometric mean of the SPECratios.

SPEC Power Benchmark

- It reports power consumption of servers at different workload levels, divided into 10% increments, over a period of time.
- SPECpower started with another SPEC benchmark for Java business applications
  - exercises the processors, caches, main memory, Java virtual machine, compiler, garbage collector, and pieces of the operating system.
- Performance is measured in throughput, as business operations per second.
- Power is measured in Watts.
- **overall ssj_ops per watt**:

$$\text{overall ssj\_ops per watt} = \left(\sum_{i=0}^{10} \text{ssj\_ops}_i\right) \Big/ \left(\sum_{i=0}^{10} \text{power}_i\right)$$

SPECpower_ssj2008 for Xeon X5650

Target Load %	Performance (ssj_ops)	Average Power (watts)
100%	865,618	258
90%	786,688	242
80%	698,051	224
70%	607,826	204
60%	521,391	185
50%	436,757	170
40%	345,919	157
30%	262,071	146
20%	176,061	135
10%	86,784	121
0%	0	80
Overall Sum	4,787,166	1922
∑ssj_ops / ∑power =		2490

FIGURE 1.19 SPECpower_ssj2008 running on a dual socket 2.66 GHz Intel Xeon X5650 with 16 GB of DRAM and one 100 GB SSD disk.

Pitfall: Amdahl's Law

- A common pitfall: Improving an aspect of a computer and expecting a proportional improvement in overall performance.

- The **Amdahl's Law** states that execution time of the program after making the improvement is

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

- Example: multiply accounts for 80s of program running in 100s.
  - How much improvement in multiply performance to get 5× improvement overall?

$$20 = \frac{80}{n} + 20$$

Can't be done!

Fallacy: Low Power at Idle

- Look back at i7 power benchmark
  - At 100% load: 258W
  - At 50% load: 170W (66%)
  - At 10% load: 121W (47%)
- Google data center
  - Mostly operates at 10% – 50% load
  - At 100% load less than 1% of the time

- We should design hardware to achieve "**energy-proportional computing**."
- If future servers used, say, 10% of peak power at 10% workload, we could reduce the electricity bill of datacenters and CO2 emissions.

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

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

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

  - CPI varies between programs on a given CPU

References

- David A. Patterson and John L. Hennessy, “Computer organization and design ARM edition: the hardware software interface,” Morgan Kaufmann, 2016.
  - Chapter 1

Most of the text has been taken and adapted from “Computer Organization and Design ARM Edition: The Hardware Software Interface”.
If not differently indicated, all figures have been taken from the book or the material in the companion website of “Computer Organization and Design ARM Edition: The Hardware Software Interface”.