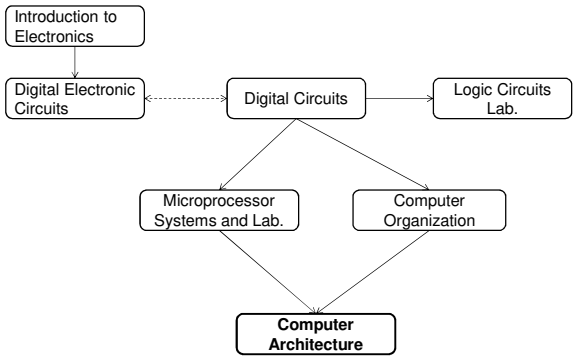
COMPUTER ARCHITECTURE

Feza BUZLUCA, Ph.D
Istanbul Technical University
Computer Engineering Department

http://akademi.itu.edu.tr/en/buzluca/
http://www.buzluca.info

1.0 Connections between the hardware-based courses in the İTÜ Computer Engineering Department



1.1. Why study Computer Architecture?

From: *Curriculum Guidelines for Undergraduate Degree Programs in Computer Science (2013)*
prepared by the Joint Task Force on Computing Curricula
of the IEEE (Institute of Electrical and Electronics Engineers) Computer Society and ACM (Association for Computing Machinery)

https://www.acm.org/education/curricula-recommendations

- Computing professionals should not regard the computer as just a black box that executes programs by magic.
- Computer architecture is a key component of computer engineering, and the practicing computer engineer should have (at least) a practical understanding of this topic.
- Students need to understand computer architecture **to develop programs** that can achieve high performance through a programmer's awareness of parallelism and latency.
- **In selecting a system** to use, students should be able to understand the tradeoff among various components, such as CPU clock speed, cycles per instruction, memory size, and average memory access time.

1.1. Why study Computer Architecture? (cont'd)

From: *IEEE/ACM Computer Engineering Curricula 2016:*

"One area of concern to the computer engineer is the software/hardware interface, where difficult trade-off decisions often provide engineering challenges.

*Considerations on this interface or boundary lead to an appreciation of and insights into **computer architecture** and the importance of a computer's machine code.*

At this boundary, difficult decisions regarding hardware/software trade-offs can occur, and they lead naturally to the design of special-purpose computers and systems.

For example, in the design of a safety-critical system, it is important to ensure that the system not harm the user or the public.

The computer engineer must thoroughly test, even with unlikely parameters, the hardware and software, and ultimately the system itself, to ensure that the system operates properly and reliably."

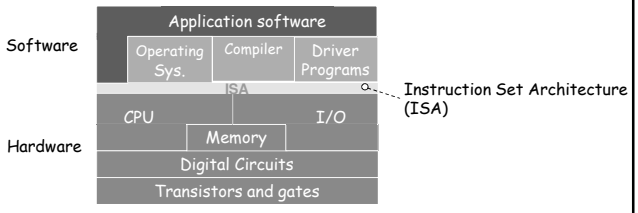
The Purpose of the Course

1. Learning how to provide **hardware-based solutions (**design computer systems) to engineering problems (considering speed, cost)
2. Learning how to **choose an appropriate computer system** to solve a problem (to realize a project) considering requirements such as processing and memory capabilities
3. Developing **high-quality software** for large and embedded systems

Topics:

- The Pipeline
  - Instruction Pipeline (Instruction-Level Parallelism)
  - Pipeline hazards and solutions
- Input/Output Organization
  - Handshaking
  - Data transfer between CPU and memory
- Exceptions and Interrupts
  - Vectors, multiple interrupts, priority, nested interrupts
- Direct Memory Access - DMA
- Memory Organization
  - Cache memory
  - Virtual Memory (*Operating systems*)
- RAID: (Redundant Array of Independent/Inexpensive Disks)
- Multiprocessor and multicore systems
  - Cache coherence

1.2. The layered logical model of a computer system:



Instruction Set Architecture (ISA) is the part of the computer architecture related to programming.

It is the interface between a computer's software and its hardware.

The ISA includes machine language instructions, registers, and native data types.

1.3 The Central Processing Unit – CPU

1.3.1 Categorization of CPUs

CPUs can be categorized based on various properties:

- Numbers of operands:
  - Zero operand/address machines (stack machines)
  - One operand/address machines (accumulator machines)
  - Two operand/address machines (register-register, register-memory, memory-memory)
  - Three operand/address machines
- Instruction sets and addressing modes
  - CISC (Complex Instruction Set Computer)
  - RISC (Reduced Instruction Set Computer)
- Instruction and data memories
  - Von Neumann architecture
  - Harvard architecture

1.3.1.1 According to instruction sets and addressing modes:

a) CISC (*Complex Instruction Set Computer*)

b) RISC (*Reduced Instruction Set Computer*)

CISC:

Motivation:

- A desire to simplify compilers. The machine language is made to look as much like a high-level language as possible.
- A desire to improve performance. Shorter programs written with powerful instructions.

Characteristics:

- Large number of instructions (100-250)
- Complex instructions and complex addressing modes (indirect memory access)
- Instructions that directly operate on memory locations
- Microprogrammed control unit

Consequences:

- Instructions have different lengths (difficult to decode and prefetch).
- Some instructions are used very rarely.
- The processor has a complex internal structure.

RISC:

Motivation: Programs written in high-level languages were compiled on CISC processors, and analysis of the generated code yielded the following results.

- There are many assignment operations (A = B) .
- Most of the instructions in a compiled program are the relatively simple ones. Complex machine instructions are often hard to exploit because the compiler must find those cases that exactly fit the construct.
- Accessed operands are mostly local and scalar (not array or vector).
- Function calls (subroutines) have a large overhead: saving the return address, transfer of parameters, local variables, stack (memory) access
- Most (98%) of the subroutines transfer 6 or fewer parameters. [1]
- Most (92%) of the subroutines use 6 or fewer local scalar variables.[1]
- Depth of nesting function calls is mostly (99%) less than 8. [2]

Based on these results, RISC processors with simple instructions which operate only on registers and access memory only for load/store operations were designed.

1. Andrew S. Tanenbaum, Implications of structured programming for machine architecture, Communications of the ACM, Vol.21 , No.3 (1978),pp. 237 - 246
2. Yuval Tamir and Carlo H. Sequin, "Strategies for Managing the Register File in RISC," IEEE Transactions on Computers Vol. C-32(11) pp. 977-989, 1983.

Characteristics of Reduced Instruction Set Architectures

Although a variety of different approaches to reduced instruction set architecture have been taken, certain characteristics are common to all of them:

- A small set of instructions (about 30)
- Simple instructions with simple format (fixed length, easy to decode)
- Simple addressing modes
- Register-to-register operations
- Memory access only for load/store instructions (*load-store architecture*).
- One instruction per clock cycle (owing to pipelining)
- *Hardwired* control unit

Other Characteristics:

Not all of the features listed below are included in all RISC processors, and CISC processors may also include some of these features:

- A large number of registers (128-256) (*Register File*)
- Overlapped register window to transfer parameters and to save local data
- Instruction pipeline
- Harvard architecture

Examples of CISC and RISC processors:

- CISC:
  VAX, PDP-11, Intel x86 until Pentium, Motorola 68K.
- RISC:
  MIPS, SPARC, Alpha, HP-PA, PowerPC, i860, i960, ARM, Atmel AVR
- Hybrid (Outer CISC shell with an inner RISC core):
  Pentium, AMD Athlon.

There is a growing realization that

- RISC designs may benefit from the inclusion of some CISC features, and
- CISC designs may benefit from the inclusion of some RISC features.

The result is that

- the more recent RISC designs, notably the PowerPC, are no longer "pure" RISC and
- the more recent CISC designs, notably the Pentium II and later Pentium models, do incorporate some RISC characteristics.

Examples of products where RISC processors are used:

- ARM:
  - Apple iPod, Apple iPhone, iPod Touch, Apple iPad.
  - Samsung mobile devices (Cortex-A)
  - RIM BlackBerry smartphone/email device
  - Microsoft Windows Mobile
  - Nintendo Game Boy Advance
- MIPS:
  - SGI computers, PlayStation, PlayStation 2
- Power (Performance Optimization With Enhanced RISC) Architecture by IBM:
  - IBM supercomputers, midrange servers, and workstations
  - Apple PowerPC-based Macintosh
  - Nintendo Gamecube, Wii
  - Microsoft Xbox 360
  - Sony PlayStation 3
- Atmel AVR:
  - BMW cars

1.3.1.2. Categorization of CPUs according to their instruction and data memories

a) Von Neumann Architecture:

Instructions and data are stored in the same memory.

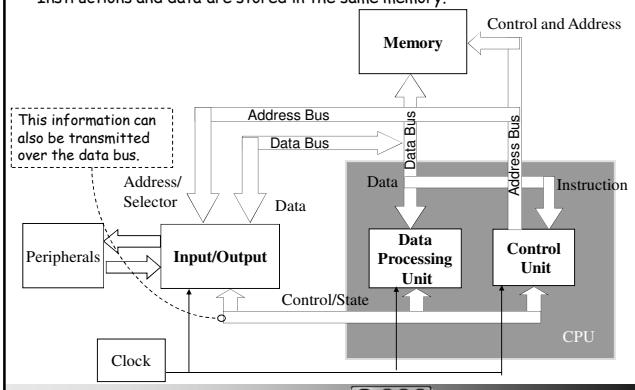
b) Harvard Architecture:

Instructions and data are stored in different memories.

The CPU can fetch instructions and data at the same time.

Von Neumann Architecture: John von Neumann (1903 – 1957)

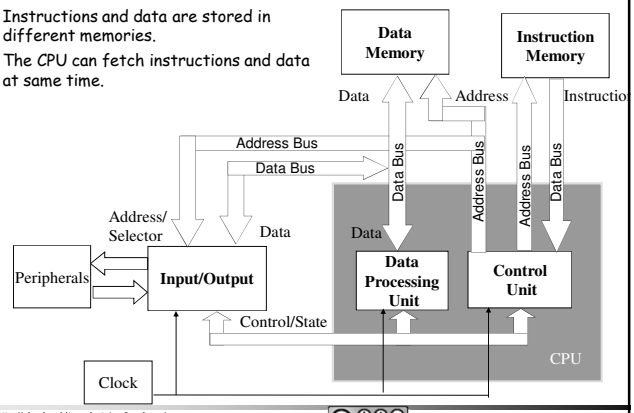
Instructions and data are stored in the same memory.



Harvard Architecture : Harvard Mark I , Harvard University

Instructions and data are stored in different memories.

The CPU can fetch instructions and data at same time.



1.3.2 Internal Structure of a CPU

Data Processing Unit:

- Performs data processing and internal data storage functions.
- Includes registers, arithmetic-logic unit, floating point unit, data pipeline.

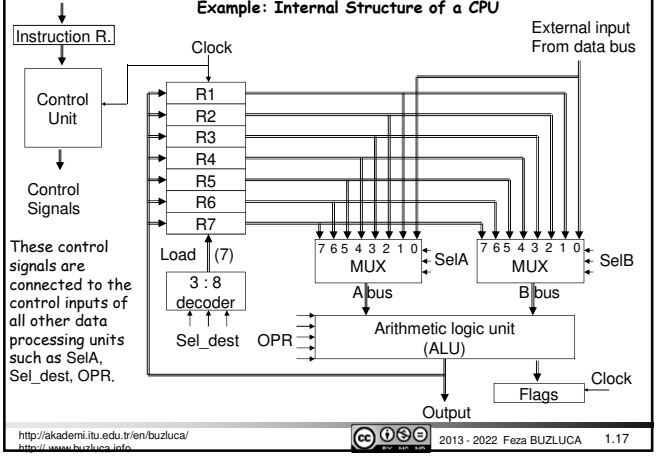
Control Unit:

- Decodes and interprets instructions; provides control signals to the data processing unit.
- Actually controls the operation of the CPU and hence the computer.
- Is designed as a finite state machine (refer to instruction cycles of the CPU for the states (instruction fetch, data fetch, execution, etc.))
- Can be implemented as a
  - synchronous sequential circuit (*hardwired*) or
  - microprogrammed machine.

Remember each instruction in the machine language of the processor is translated into a sequence of lower-level control unit instructions which are called microinstructions.

The internal structure of an exemplary CPU is shown on slide 1.17.

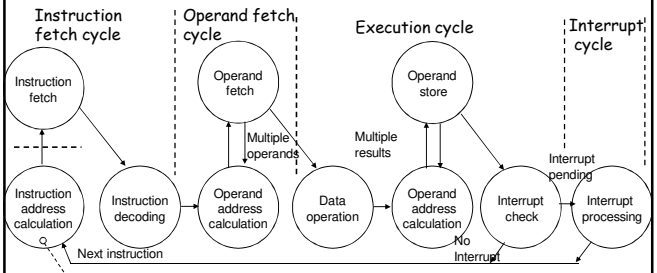
Example: Internal Structure of a CPU



Instruction Cycle of a CPU:

The control unit of a CPU designed as a finite state machine can be in one of 4 states (cycles):

1. Instruction fetch cycle, 2. Operand fetch c., 3. Execution c., 4. Interrupt cycl.



The address of the next instruction is calculated in the execution or interrupt cycle.

1.4 CPU Performance and Its Factors

Remember: All synchronous sequential digital circuits are constructed using a **clock signal** running at a constant rate (See BLG 231E lecture notes).

These discrete time events are called clock periods, clocks, cycles, or clock cycles.

We refer to the time of a clock period by its duration (e.g., 0.5 ns) or by its rate (e.g., 2 GHz).

Remember: Since the CPU is implemented as a synchronous sequential digital circuit, it is also triggered by a clock signal.

1.4.1 CPU execution time for a program:

CPU Time = CPU clock cycles for a program X Clock cycle time

or

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

$$\text{Clock rate} = \frac{1}{\text{Clock cycle time}}$$

The hardware designer can improve performance by reducing the number of clock cycles required for a program or the length of the clock cycle.

However, there is often a trade-off between the number of clock cycles needed for a program and the length of each cycle. Many techniques that decrease the number of clock cycles may also increase the clock cycle time.

1.4.2 Clock cycles per instruction (CPI):

The average number of clock cycles each instruction takes to execute.

$$\text{CPI} = \frac{\text{CPU clock cycles for a program}}{\text{Instruction count (IC)}}$$

Here, instruction count (IC) is the number of machine-language instructions that constitute the program.

These instructions are either generated by the compilers from high-level programs or written by the systems programmers using the assembly language of the processor.

Since different instructions may take different amounts of time depending on what they do (especially in CISC processors), CPI is **an average** of all the instructions executed in the program.

Example: MC68000 instructions

CLR.W D0;	*Clear Data register D0*	*8 cycles*
ADD.W (A1)+,D0;	*Add data pointed by A1 to D0 and increment A1*	*16 cycles*

Each execution of a given instruction may take a different number of clock cycles due to data and branch hazards (pipelining, DMA; we will discuss these topics later).

1.4.3 CPU Performance Equation:

Required clock cycles for a program:

CPU Clock Cycles = Instruction count (IC) X Cycles per instruction (CPI)

CPU time for the program:

CPU Time = CPU Clock Cycles X Clock cycle time

Basic performance equation:

CPU Time = IC X CPI X Clock cycle time

CPU Time = Instruction count (IC) X Cycles per instruction (CPI) X Clock cycle time

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

$$\text{CPU Time} = \frac{\text{Instruction count (IC) X Cycles per instruction (CPI)}}{\text{Clock rate}}$$

Technologies affecting performance characteristics:

- Clock cycle time: Hardware technology and organization
- CPI: Organization and instruction set architecture
- Instruction count: Instruction set architecture and compiler technology

Example:

Two implementations of the **same instruction set** architecture.

Computer A:

Clock cycle time: 250 ps (0.25 ns) and a CPI of 2.0 for some program

Computer B:

Clock cycle time: 500 ps (0.5 ns) and a CPI of 1.2 for the same program

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

Solution:

Each computer executes the same number of instructions (IC) for the program.

$\text{CPU clock cycles}_A = \text{IC X } 2.0$

$\text{CPU clock cycles}_B = \text{IC X } 1.2$

$\text{CPU time}_A = \text{CPU clock cycles}_A \text{ X Clock cycle time}_A$

$\text{CPU time}_A = \text{IC X } 2.0 \text{ X } 250 \text{ ps} = \mathbf{500 \text{ X IC ps}}$

$\text{CPU time}_B = \text{CPU clock cycles}_B \text{ X Clock cycle time}_B$

$\text{CPU time}_B = \text{IC X } 1.2 \text{ X } 500 \text{ ps} = \mathbf{600 \text{ X IC ps}}$

Computer A is faster.

The amount faster is given by the ratio of the execution times:

$$\frac{\text{CPU performance}_A}{\text{CPU performance}_B} = \frac{1 / \text{Execution time}_A}{1 / \text{Execution time}_B} = \frac{\text{Execution time}_B}{\text{Execution time}_A} = \frac{600 \text{ X IC ps}}{500 \text{ X IC ps}} = \mathbf{1.2}$$

1.4.3 CPU Performance Equation (cont'd):

Sometimes it is useful to consider instructions and their CPI values individually (NOT the overall average of all instructions) for the calculation of the total number of processor clock cycles.

Remember: Different instructions may take different amounts of time depending on what they do.

$$\text{CPU Clock cycles} = \sum_{i=1}^{n} (\text{IC}_i \times \text{CPI}_i)$$

n : Number of different instructions in the program

IC_i : Number of times instruction i is executed in a program

CPI_i: Average number of clocks per instruction for instruction i (for each execution of an instruction, CPI may be different due to some conflicts).

$$\text{CPU time} = \left(\sum_{i=1}^{n} (\text{IC}_i \times \text{CPI}_i) \right) \times \text{Clock cycle time}$$

Overall average CPI:

$$\text{CPI} = \frac{\sum_{i=1}^{n} (\text{IC}_i \times \text{CPI}_i)}{\text{Instruction count}} = \sum_{i=1}^{n} \left(\frac{\text{IC}_i}{\text{Instruction count}} \times \text{CPI}_i \right)$$

This form of the CPI calculation uses each individual CPI_i and the fraction of occurrences of that instruction in a program (i.e., IC_i / Instruction count).

Example:

(From Patterson, Hennessy, "Computer Organization and Design", MK, 2018)

A compiler designer is trying to decide between two code sequences for a high-level language statement (e.g., FOR, IF, Z = X + Y, etc.).

The hardware designers have supplied the following facts:

CPI for each instruction class:

Instruction i	**A**	**B**	**C**
CPI_i	1	2	3

For a particular high-level language statement, the compiler writer is considering two machine-language code sequences that require the following **instruction counts**:

Code sequence	A	B	C	Total
S1	2	1	2	5
S2	4	1	1	6

S1 executes fewer instructions.

Clock cycles for S1: (2 x 1) + (1 x 2) + (2 x 3) = 10 cycles

Clock cycles for S2: (4 x 1) + (1 x 2) + (1 x 3) = 9 cycles (fewer cycles)

Code sequence **S2 is faster**, even though it executes one extra instruction.

$\text{CPI}_{S1} = \text{CPU clock cycles}_{S1} / \text{Instruction count}_{S1} = 10 / 5 = 2.0$

$\text{CPI}_{S2} = \text{CPU clock cycles}_{S2} / \text{Instruction count}_{S2} = 9 / 6 = 1.5$

1.4.4 SUMMARY: CPU Performance and Its Factors

CPU Time = Instruction count (IC) X Cycles per instruction (CPI) X Clock cycle time

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

The only complete and reliable measure of computer performance is **time**.

Processor performance is dependent upon three characteristics:

- instruction count,
- clock cycles per instruction, and
- clock cycle (or rate).

In designing the processor, it is difficult to change one parameter in complete isolation from others because the basic technologies involved in changing each characteristic are interdependent.

1.4.4 SUMMARY: CPU Performance and Its Factors (cont'd)

CPU Time = Instruction count (IC) X Cycles per instruction (CPI) X Clock cycle time

Improving the CPU Performance:

Instruction count:

Designer can reduce the instruction count by adding more powerful and complex instructions to the instruction set.

However, this can increase either CPI or clock cycle, or both.

Clocks per instruction:

Computer architects can reduce the CPI using a pipeline to exploit instruction-level parallelism (section 2.4).

Adding more powerful and complex instructions often increases CPI.

Clock cycle time:

Clock cycle time depends on transistor speed, propagation delays in circuits, and the complexity of the operations performed in a single clock.

Clock time can be reduced when transistor sizes decrease. However, power consumption increases when clock time is reduced. This increases the amount of heat generated.

1.4.5 The Performance of a program:

The components that affect the factors in the CPU performance equation:

- **Algorithm:** Instruction count, CPI

  The algorithm determines the number of source program (high-level language) instructions, and consequently the number of processor instructions executed.

  The algorithm may also affect the CPI, by favoring basic or complex instructions.

- **Programming language:** Instruction count, CPI

  Statements in the language are translated to processor instructions.

  It may also affect the CPI. For example, a language with heavy support for data abstraction will require indirect calls, which will use higher-CPI instructions.

- **Compiler:** Instruction count, CPI

  The compiler translates the source high-level language instructions in the source code into machine language instructions.

- **Instruction set architecture:** Instruction count, CPI, clock rate

  The CPU design affects all three factors of the performance.

1.4.6 Benchmarks

MIPS (Millions of instructions per second) and MFLOPS (floating-point operations per second) are inadequate in evaluating the performance of processors.

Because of differences in instruction sets, the instruction execution rate is not a valid means of comparing the performance of different architectures.

SPEC Benchmarks: (http://www.spec.org/)

The best known collection of benchmark suites is defined and maintained by the System Performance Evaluation Corporation (SPEC), an industry consortium.

A benchmark suite is a collection of programs, defined in a high-level language, that together can be used to test a computer in a particular application or system programming area.

The best known of the SPEC benchmark suites is **SPEC CPU2017**.This is the industry standard suite for processor-intensive applications.

Examples of other SPEC suites:

- **SPECjbb 2015 (Java Business Benchmark):** A benchmark for evaluating server-side Java-based electronic commerce applications
- **SPECweb2009:** Evaluates the performance of World Wide Web (WWW) servers
- **SPECmail2009:** Measures a system's performance acting as a mail server

1.5 The evolution of computers

Characteristics of evolution:

Increase in processor speed, increase in level of integration of circuits, decrease in component size, increase in memory size, and increase in I/O capacity and speed.

Reasons for the increase in processor speed:

- **Achievements in material:**

  Shrinking size of microprocessor components (this reduces the distance between components and hence increases speed)

- **Organizational improvements:**

  Heavy use of pipelining and parallel execution techniques, multiple ALUs multicore designs

  Cache memories

In this course, we will discuss the organizational improvements.

1.5.1 Improvements and Problems in Computer Organization and Architecture

There are three main approaches to achieving increased processor speed:

1. Increasing the hardware speed of the processor (clock speed) (But !)
2. Making changes to the processor organization and architecture that increase the effective speed of instruction execution (parallelism, pipeline)
3. Increasing the size and speed of cache memories

As clock speed and logic density increase, a number of problems arises (#1 above).

Power: It is difficult to dissipate the heat generated on high-density, high-speed chips (the power wall, slide 1.31).

RC delay: The speed at which electrons can flow on a chip between transistors is limited by the resistance and capacitance of the metal wires connecting them.

The wire interconnects become thinner, increasing resistance. Also, the wires are closer together, increasing capacitance.

Therefore, it is not possible to increase the clock speed.

Memory latency: In addition, the speed of memories does not increase at the same rate as that of processors.

Memory speeds lag processor speeds.

The Power Wall *:

Dynamic power per transistor (P) is proportional to frequency of operation (f) times the square of the operating voltage (V) (**$P \sim V^2 f$**).

To reduce the power increase, the operating voltage can be decreased, but it is limited by the transistors' operating threshold voltages.

The total dynamic power dissipated by an entire IC can be expressed as **$P \sim Nf$**, where N represents the total number of transistors operating simultaneously.

Increasing the number of transistors at Moore's law pace and increasing the operating frequency is bound to reach a thermal dissipation limit—the power wall.

By 2003, processors exceeded 200 W per chip. This milestone marked the crossing of a power threshold that requires far more expensive cooling technologies, which were outside the system-cost envelope of PC hardware at that time.

The industry had to choose which to slow down: the growth of the microprocessor's transistor number from one generation to the next, or the operational frequency rate.

They decided on the second option, which maintained Moore's law but sacrificed frequency growth.

*Source: T. M. Conte, E. P. DeBenedictis, P. A. Gargini, and E. Track, "Rebooting Computing: The Road Ahead," Computer, vol. 50, no. 1, pp. 20-29, Jan. 2017.

Integration of circuits

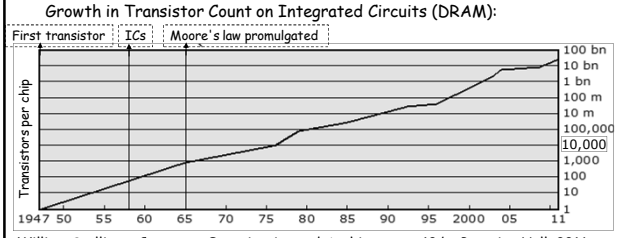
Moore's law (by Gordon Moore, cofounder of Intel): "The number of transistors that can be put on a single chip is doubling every year and this pace will continue into the near future". (1965)

Since the 1970s, the number of transistors on integrated circuits has been doubling approximately every 18 months.

Today, most visible with DRAM capacity. Its growth is slowing down.

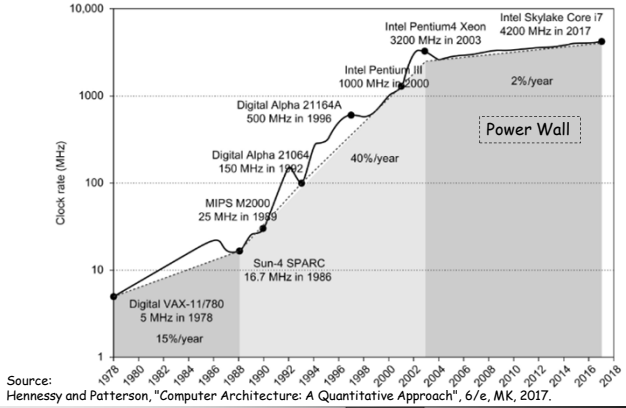
Gordon Moore expects Moore's law will end by around 2025.

Growth in Transistor Count on Integrated Circuits (DRAM):



William Stallings, Computer Organization and Architecture, 10/e, Prentice Hall, 2016

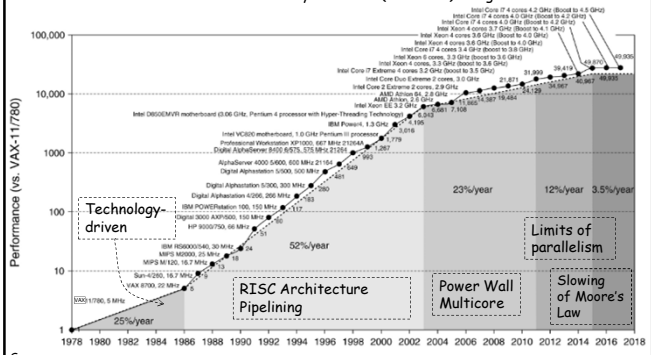
Increase in clock rate of microprocessors:



Source:
Hennessy and Patterson, "Computer Architecture: A Quantitative Approach", 6/e, MK, 2017.

Increase in processor performance :

Relative to the VAX 11/780 as measured by the SPEC (slide 1.28) integer benchmarks



Source:
Hennessy and Patterson, "Computer Architecture: A Quantitative Approach", 6/e, MK, 2017.

Limits to growth in processor performance:

End of Dennard scaling:

In 1974, Robert Dennard: "Power density is constant for a given area of silicon even as the number of transistors are increased by making them smaller".

"Transistors with smaller dimensions run faster but use less power."

Dennard scaling ended around 2004 because current and voltage could not keep dropping (one of the reasons of the power wall).

Limits to parallelism:

Idea: multiple efficient processors or cores instead of a single inefficient processor; data-level parallelism and thread-level parallelism.

However, it is not always possible to split a task into parts that can run in parallel.

Besides, dependencies and communication overhead among subtasks can decrease the performance.

Slowing of Moore's Law:

In 1965, Gordon Moore: "The number of transistors per chip would double every year" ; in 1975: "every two years".

That prediction lasted for about 50 years, but in the last few years, the rate of increase has been getting slower. Today, most visible with DRAM capacity.

It is expected that the Moore's law will end by around 2025.

Performance balance between processor and main memory

In computer system design, it is critical to balance the performance of different components so that gain in performance in one element is not handicapped by a lag in another element.

Processor speed has increased more rapidly than the speed of the main memory (memory access time).

For example, the Intel Core i7 6700 can generate two data memory references per core each clock cycle. With four cores and a 4.2 GHz clock rate, it can generate:

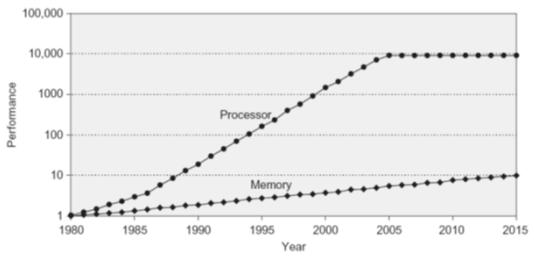
- A peak of 33.6 billion (=2 x 4 cores x 4.2 GHz) 64-bit data memory references per second.
- A peak instruction demand of about 16.8 billion (=4 cores x 4.2 GHz) 128-bit instruction references.

⇒ This is a total peak demand bandwidth of 33.6*64/8+16.8*128/8=537.6 GB/s = 501 GiB/s (GibiByte = 2^{30})!

In contrast, the peak bandwidth for DRAM main memory, using two memory channels, is only 6.3% of the demand bandwidth (2.133 GHz * 2 * 64 /8 = 34.1 GB/s).

A variety of techniques is used to compensate for this mismatch, including caches, wider data paths between memory and processor.

Performance gap between the CPU and the main memory



Single processor

The processor line: Increase in memory requests per second on average (i.e., the inverse of the latency between memory references) (single processor)

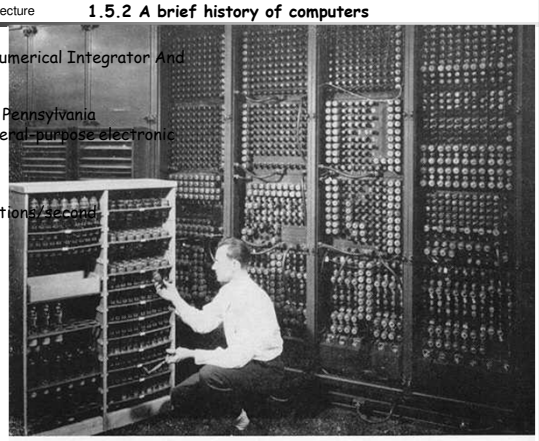
The memory line: Increase in DRAM accesses per second (i.e., the inverse of the DRAM access latency).

Source: Hennessy and Patterson, "Computer Architecture: A Quantitative Approach", 6/e, MK, 2017.

1.5.2 A brief history of computers

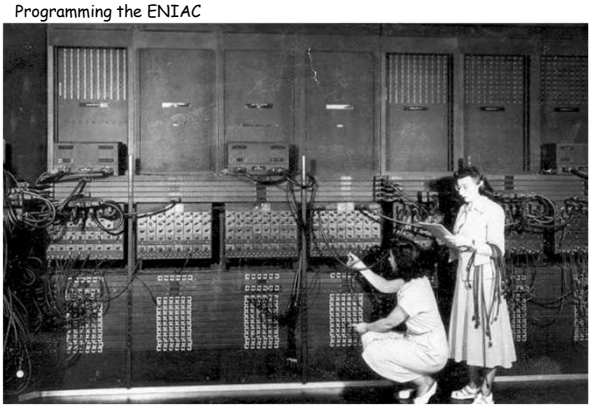
ENIAC 1946
(Electronic Numerical Integrator And Computer),

University of Pennsylvania
The first general-purpose electronic computer
- 30 tons
- 140 kW
- 5,000 additions/second



Replacing a bad tube meant checking among ENIAC's 19,000 possibilities.

Programming the ENIAC



Source: http://www.library.upenn.edu/exhibits/

Z3 (1941):

Konrad Zuse,
(1910-1995)

The first general-purpose computer.

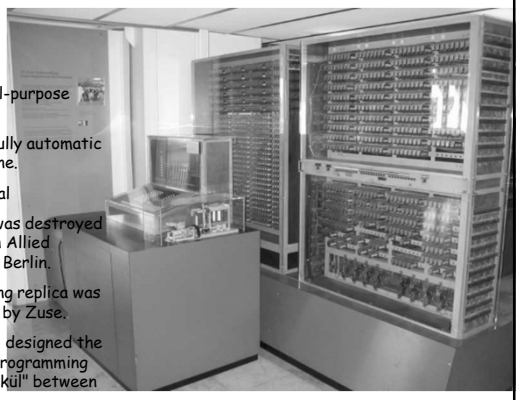
Programmable, fully automatic computing machine.

Electromechanical

The original Z3 was destroyed in 1943 during an Allied bombardment of Berlin.

A fully functioning replica was built in the 1962 by Zuse.

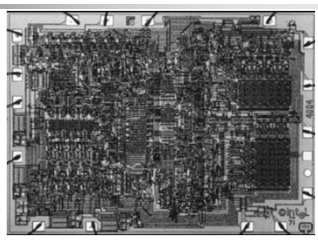
Konrad Zuse also designed the first high-level programming language "Plankalkül" between 1942 and 1946.

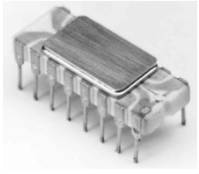


First microprocessor:

Intel 4004

- 1971
- 4-bit data
- 2,300 transistors
- 740 KHz
- Addressable Memory: 640 Bytes
- 12 V





Source: http://www.intel.com

First member of the x86 Family:

Intel 8086

- 1978
- 16-bit data
- 29,000 transistors
- 3-10 MHz (1 Mega = 10^6 = 1000^2)
- Addressable Memory: 1 MiB (1 **Mebi**Byte = 2^{20} = 1024^2) (for Mebi: Slides 1.45-1.48)
- 5 V

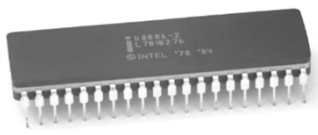
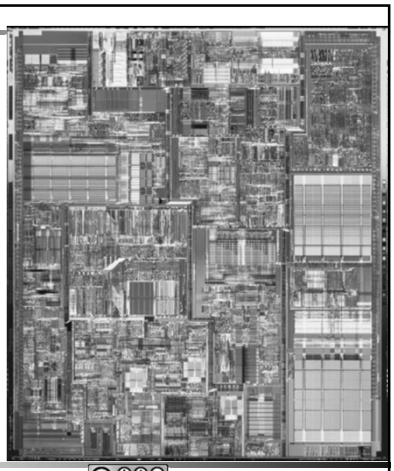


Image Source:
http://en.wikipedia.org/w/index.php?title=File:KL_Intel_D8086.jpg
License: GNU Free Documentation License
Contributors: Konstantin Lanzet

Multithreading (Hyper-threading)

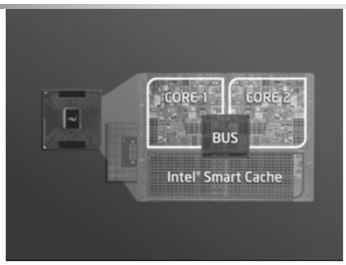
Intel Pentium4 + HT

- 2003
- 32/64-bit data
- 55 million transistors
- 3.4 GHz (Gigahertz)
- Addressable Memory: 64 GiBytes (Gibibyte)
- 1.2 V



Multicore Processors:
Intel® Core™ Duo

- 2006
- 64-bit data
- 100 million transistors
- 2.66 GHz
- 1.5 V



Intel® Core™ i9 - 9900

- 2019
- 3.10 GHz (Base) , 5.00 GHz (Max Turbo)
- 8 cores
- 64-bit data
- 128 GiB memory addressing capacity
- Cache: 512 KiB L1, 2 MiB L2, 16 MiB L3

1.6 Binary prefixes (Ki, Mi, Gi, …) IEEE 1541-2002 Standard

Problem:

In early days of the computer industry, it was common practice to use "**kilo**" as a prefix denoting multiplication by 1024 (= 2^{10}), although the real meaning of the decimal prefix kilo in SI (The International System of Units) was 1000 (= 10^3).

Initially, this created no problem because there is not much difference between 1000 and 1024, and within the community of persons who used computers, everybody understood what was meant.

As the capacity of memories and disks has grown larger, industry has also used the prefixes **Mega** to denote 2^{20} and **Giga** to denote 2^{30}.

However, their meanings in SI are different; Mega: 10^6 , Giga: 10^9.

The disparity between binary and decimal multiples is larger with the larger prefixes.

For example, how many bytes can a terabyte (1 TB) of storage hold - 10^{12} bytes or 2^{40} bytes? The difference is roughly 10%.

In most other contexts, the industry uses the prefixes kilo, mega, giga, etc., in a manner consistent with their meanings in SI, namely as powers of 10.

For example, a 500 GB hard disk holds 500,000,000,000 bytes, and a 1 Gbit/s (gigabit per second) Ethernet connection transfers data at nominal speed of 1,000,000,000 bit/s.

Solution (Binary Prefixes):

The use of the same unit prefixes with different meanings has caused confusion.

Starting around 1998, the International Electrotechnical Commission (IEC) and several other standards organizations published standards and recommendations to remove the ambiguity.

From the SI Brochure of The International System of Units (SI):

"The SI prefixes refer strictly to powers of 10. They should not be used to indicate powers of 2 (for example, one kilobit represents 1000 bits and not 1024 bits)."

"The names and symbols for prefixes to be used with powers of 2 are recommended as follows:"

kibi (kilobinary)	Ki: 2^{10}	tebi (terabinary)	Ti: 2^{40}
mebi (megabinary)	Mi: 2^{20}	pebi (petabinary)	Pi: 2^{50}
gibi (gigabinary)	Gi: 2^{30}	exbi (exabinary)	Ei: 2^{60}

Decimal SI prefixes:

kilo k: 10^3 mega M: 10^6 giga G: 10^9 tera T: 10^{12}
peta P: 10^{15} exa E: 10^{18} zetta Z: 10^{21} yotta Y: 10^{24}

Solution (Binary Prefixes) (cont'd):

In 1998, the IEC defined a set of binary prefixes to denote powers of 2, and then the IEEE adopted these binary prefixes in the standard 1541-2002.

In 2005, 1541-2002 (IEEE Standard for Prefixes for Binary Multiples) was published as a full-use standard.

Prefix	Symbol	Value
kibi-	Ki	$1024^1 = 2^{10}$
mebi-	Mi	$1024^2 = 2^{20}$
gibi-	Gi	$1024^3 = 2^{30}$
tebi-	Ti	$1024^4 = 2^{40}$
pebi-	Pi	$1024^5 = 2^{50}$
exbi-	Ei	$1024^6 = 2^{60}$
zebi-	Zi	$1024^7 = 2^{70}$
yobi-	Yi	$1024^8 = 2^{80}$

Using decimal and binary prefixes:

Working with powers of 10 is easier; therefore, if the quantities we are dealing are not exact powers of 2, decimal powers should be preferred.

Using binary prefixes makes sense when the quantity we are dealing with is a power of 2.

Decimal prefixes:

- File size (bytes) such as MB, GB
- Disks and drives size (bytes)
- Transfer speed (bits/second)
- Processor speed (hertz)

Binary prefixes:

- RAM (bytes) such as MiB, GiB
- Cache (bytes)

In these lecture slides, all prefixes that refer to a memory capacity (RAM or cache) denote powers of 2! For example, 1 GB memory = 1 GiB memory = 2^{30} Bytes

Binary and Decimal prefixes:

Source: https://www.electropedia.org/iev/iev.nsf/

Binary prefixes:

Factor	Value	Prefix	
		Name	Symbol
$(2^{10})^1$	1 024	kibi	Ki
$(2^{10})^2$	1 048 576	mebi	Mi
$(2^{10})^3$	1 073 741 824	gibi	Gi
$(2^{10})^4$	1 099 511 627 776	tebi	Ti
$(2^{10})^5$	1 125 899 906 842 624	pebi	Pi
$(2^{10})^6$	1 152 921 504 606 846 976	exbi	Ei
$(2^{10})^7$	1 180 591 620 717 411 303 424	zebi	Zi
$(2^{10})^8$	1 208 925 819 614 629 174 706 176	yobi	Yi

Decimal prefixes:

Factor	Value	Prefix	
		Name	Symbol
$(10^3)^1$	1 000	kilo	k
$(10^3)^2$	1 000 000	mega	M
$(10^3)^3$	1 000 000 000	giga	G
$(10^3)^4$	1 000 000 000 000	tera	T
$(10^3)^5$	1 000 000 000 000 000	peta	P
$(10^3)^6$	1 000 000 000 000 000 000	exa	E
$(10^3)^7$	1 000 000 000 000 000 000 000	zetta	Z
$(10^3)^8$	1 000 000 000 000 000 000 000 000	yotta	Yi