The main component of a Computer

- Computer is a programmable machine.
- Computer is a machine that manipulates data according to a list of instructions.
- Computer is any device which aids humans in performing various kinds of computations or calculations.

Three principle characteristics of computer

- It responds to a specific set of instructions in a well-defined manner.
- It can execute a pre-recorded list of instructions.
- It can quickly store and retrieve large amounts of data.

Structure of the Computer

Input Unit

- Computers accept coded information through input units.
- The most common input device is the keyboard. Whenever a key is pressed, the corresponding letter or digit is automatically translated into its corresponding binary code and transmitted to the processor.
- Many other kinds of input devices for human-computer interaction are available, including the touchpad, mouse, joystick, and trackball. These are often used as graphic input devices in conjunction with displays.
- Microphones can be used to capture audio input which is then sampled and converted into digital codes for storage and processing.
- Similarly, cameras can be used to capture video input.
- Digital communication facilities, such as the Internet, can also provide input to a computer from other computers and database servers.

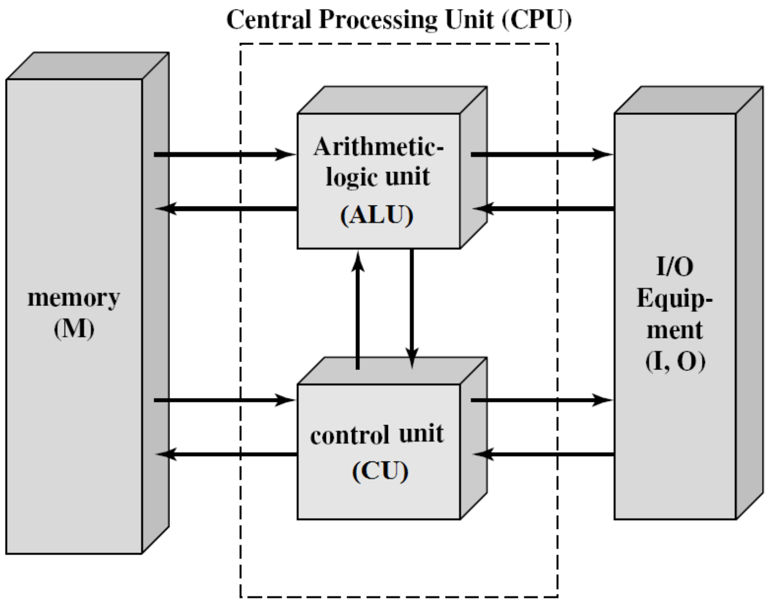


Fig. 1: Structure of the Computer

Output Unit

- The output unit is the counterpart of the input unit.
- Its function is to send processed results to the outside world. A familiar example of such a device is a *printer*.
- However, printers are mechanical devices, and as such are quite slow compared to the electronic speed of a processor.
- Some units, such as graphic displays, provide both an output function, showing text and graphics, and an input function, through touch-screen capability.
- The dual role of such units is the reason for using the single name *input/output* (I/O) unit in many cases.

Memory Unit

The function of the memory unit is to store programs and data. There are two classes of storage, called primary and secondary.

Primary Memory

- *Primary memory*, also called *main memory*, is a fast memory that operates at electronic speeds.
- Programs must be stored in this memory while they are being executed.
- The memory consists of a large number of semiconductor storage cells, each capable of storing one bit of information.
- Instructions and data can be written into or read from the memory under the control of the processor.

Secondary Storage

- Although primary memory is essential, it tends to be expensive and does not retain information when power is turned off.
- An additional, less expensive, permanent *secondary storage* is used when large amounts of data and many programs have to be stored, particularly for information that is accessed infrequently.
- Access times for secondary storage are longer than for primary memory.
- A wide selection of secondary storage devices is available, including *magnetic disks*, *optical disks* (DVD and CD), and *flash memory devices*.

Central Processing Unit (CPU)

Arithmetic and Logic Unit (ALU)

- Computer operations are executed in the *arithmetic and logic unit* (ALU) of the processor.
- Any arithmetic or logic operation, such as addition, subtraction, multiplication, division, or comparison of numbers, is initiated by bringing the required operands into the processor, where the operation is performed by the ALU.
- When operands are brought into the processor, they are stored in high-speed storage elements called *registers*.

- Each register can store one word of data. Access times to registers are even shorter than access times to the cache unit on the processor chip.

Control Unit (CU)

- The memory, arithmetic and logic unit, and I/O units store and process information and perform input and output operations.
- The operation of these units must be coordinated in some way. This is the responsibility of the control unit.
- The control unit sends control signals to other units and senses their states.
- Control circuits are responsible for generating the *timing signals* that govern the transfers and determine when a given action is to take place.
- Data transfers between the processor and the memory are also managed by the control unit through timing signals.
- A large set of control lines (wires) carries the signals used for timing and synchronization of events in all units.

Standards Organizations

Manufacturers know that by establishing common guidelines for a particular type of equipment, they can market their products to a wider audience than if they came up with separate—and perhaps incompatible—specifications. Some standards organizations have formal charters and are recognized internationally as the definitive authority in certain areas of electronics and computers.

Institute of Electrical and Electronic Engineers (IEEE)

- The Institute of Electrical and Electronic Engineers (IEEE) is an organization dedicated to the advancement of the professions of electronic and computer engineering.
- The IEEE actively promotes the interests of the worldwide engineering community by publishing an array of technical literature.
- The IEEE also sets standards for various computer components, signaling protocols, and data representation.

International Telecommunications Union (ITU)

- The International Telecommunications Union (ITU) is based in Geneva, Switzerland.
- The ITU was formerly known as the International Consultative Committee on Telephony and Telegraphy.
- As its name implies, the ITU concerns itself with the interoperability of telecommunications systems, including telephone, telegraph, and data communication systems.
- The telecommunications arm of the ITU, the ITU-T, has established a number of standards.

ISO

- The International Organization for Standardization (ISO) is the entity that coordinates worldwide standards development.
- ISO is not an acronym, but derives from the Greek word, isos, meaning “equal.”
- The ISO consists of over 2,800 technical committees, each of which is charged with some global standardization issue.
- Its interests range from the behavior of photographic film to the pitch of screw threads to the complex world of computer engineering.
- The proliferation of global trade has been facilitated by the ISO.
- Today, the ISO touches virtually every aspect of our lives.

Others

- Many countries, including the European Community, have commissioned umbrella organizations to represent their interests within various international groups.
- The group representing the United States is the American National Standards Institute (ANSI).
- Great Britain has its British Standards Institution (BSI) in addition to having a voice on CEN (Comite Europeen de Normalisation), the European committee for standardization.

Computer Generations

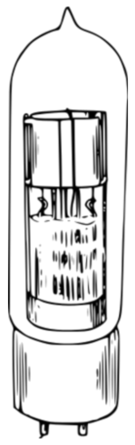
Originally calculations were computed by humans, whose job title was computers. These human computers were typically engaged in the calculation of a mathematical expression. The calculations of this period were specialized and expensive, requiring years of training in mathematics. The first use of the word "computer" was recorded in 1613, referring to a person who carried out calculations, or computations. The historical development of computer is divided in various generations as discussed below:

The Zero Generation [1642–1945]

- In this duration mechanical calculating machines are invented that could add, subtract, multiply, and divide.
- None of these devices could be programmed or had memory.
- They required manual intervention throughout each step of their calculations.
- For example; prior to the 1500s, a typical European businessperson used an abacus for calculations.

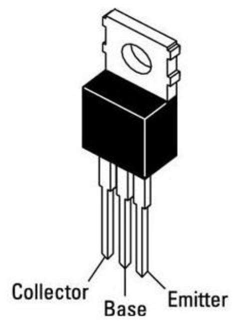
The First Generation [1946 -1958]

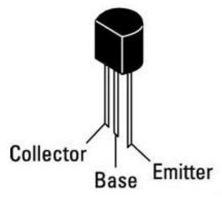
- The first computers used vacuum tubes for circuitry and magnetic drums for memory, and were often enormous, taking up entire rooms.
- They were very expensive to operate and in addition to using a great deal of electricity, generated a lot of heat, which was often the cause of malfunctions.
- First generation computers relied on machine language, the lowest-level programming language understood by computers, to perform operations, and they could only solve one problem at a time.
- Input was based on punched cards and paper tape, and output was displayed on printouts.


Vacuum Tube

- The first computers used vacuum tubes for circuitry and magnetic drums for memory, and were often enormous, taking up entire rooms.
- They were very expensive to operate and in addition to using a great deal of electricity, generated a lot of heat, which was often the cause of malfunctions.
- First generation computers relied on machine language, the lowest-level programming language understood by computers, to perform operations, and they could only solve one problem at a time.
- Input was based on punched cards and paper tape, and output was displayed on printouts.

The Second Generation [1959 -1964]

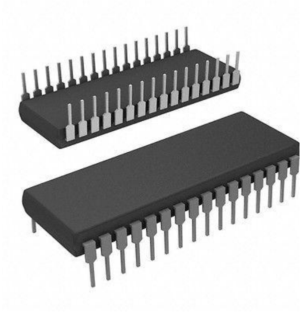




Transistors

- Transistors replaced vacuum tubes and ushered in the second generation of computers.
- One transistor replaced the equivalent of 40 vacuum tubes.
- Allowing computers to become smaller, faster, cheaper, more energy-efficient and more reliable.
- Still generated a great deal of heat that can damage the computer.
- Second-generation computers moved from cryptic binary machine language to symbolic, or assembly, languages, which allowed programmers to specify instructions in words.
- Second-generation computers still relied on punched cards for input and printouts for output.
- These were also the first computers that stored their instructions in their memory, which moved from a magnetic drum to magnetic core technology.

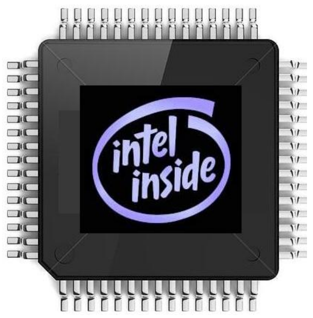
The Third Generation [1965 -1970]



Integrated Circuits (IC)

- The development of the integrated circuit was the hallmark of the third generation of computers.
- Transistors were miniaturized and placed on silicon chips, called semiconductors, which drastically increased the speed and efficiency of computers.
- Much smaller and cheaper compare to the second generation computers.
- It could carry out instructions in billionths of a second.
- Users interacted through keyboards and monitors and interfaced with an operating system, which allowed many different applications at same time with a central program that monitored the memory.
- Computers for the first time became accessible to a mass audience because they were smaller and cheaper than their predecessors.

The Fourth Generation [1971 -today]



Microprocessor

- The microprocessor brought the fourth generation of computers, as thousands of integrated circuits were built onto a single chip.
- These more powerful computers could be linked together to form networks.
- Fourth generation computers also saw the development of GUIs, the mouse and handheld devices.

The Fifth Generation [Today to future]

- Based on Artificial Intelligence (AI).
- Still in development.
- The use of parallel processing and superconductors is helping to make artificial intelligence a reality.
- The goal is to develop devices that respond to natural language input and are capable of learning and self-organization.
- There are some applications, such as voice recognition, that are being used today.

Moore's Law

- How small can we make transistors? How densely can we pack chips?
- Every year, scientists continue to thwart prognosticators' attempts to define the limits of integration.
- In 1965, Intel founder Gordon Moore stated, "The density of transistors in an integrated circuit will double every year."
- The current version of this prediction is usually conveyed as "the density of silicon chips doubles every 18 months."
- This assertion has become known as Moore's Law. Moore intended this postulate to hold for only 10 years. However, advances in chip manufacturing processes have allowed this law to hold for almost 40 years.
- Using current technology, Moore's Law cannot hold forever. There are physical and financial limitations that must ultimately come into play.
- At the current rate of miniaturization, it would take about 500 years to put the entire solar system on a chip! Clearly, the limit lies somewhere between here and there. Cost may be the ultimate constraint.

Rock's Law

- Rock's Law, proposed by early Intel capitalist Arthur Rock, is a corollary to Moore's law: "The cost of capital equipment to build semiconductors will double every four years."
- Rock's Law arises from the observations of a financier who has seen the price tag of new chip facilities escalate from about $12,000 in 1968 to $12 million in the late 1990s.
- At this rate, by the year 2035, not only will the size of a memory element be smaller than an atom, but it would also require the entire wealth of the world to build a single chip! So even if we continue to make chips smaller and faster, the ultimate question may be whether we can afford to build them.
- Certainly, if Moore's Law is to hold, Rock's Law must fall.

The Computer Level Hierarchy

Computer system organization can be approached in a several levels, in which each level has a specific function and exists as a distinct hypothetical machine. Each level's virtual machine executes its own particular set of instructions. Figure shows the commonly accepted layers representing the abstract virtual machines.

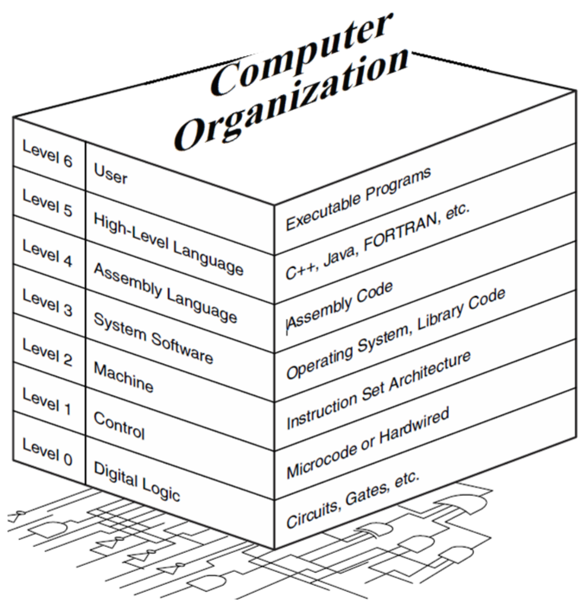


Figure: The Abstract Levels of Modern Computing Systems

Level 6

- Level 6, the User Level, is composed of applications and is the level with which everyone is most familiar.
- At this level, we run programs such as word processors, graphics packages, or games.
- The lower levels are nearly invisible from the User Level.

Level 5

- Level 5, the High-Level Language Level, consists of languages such as C, C++, FORTRAN, Lisp, Pascal, and Prolog.
- These languages must be translated (using either a compiler or an interpreter) to a language the machine can understand.
- Compiled languages are translated into assembly language.

Level 4

- Level 4, the Assembly Language Level, encompasses some type of assembly language.
- This is a one-to-one translation, meaning that one assembly language instruction is translated to exactly one machine language instruction.

Level 3

- Level 3, the System Software Level, deals with operating system instructions.
- This level is responsible for multiprogramming, protecting memory, synchronizing processes, and various other important functions.
- Instructions translated from assembly language to machine language are passed through this level unmodified.

Level 2

- Level 2, the Instruction Set Architecture (ISA), or Machine Level, consists of the machine language recognized by the particular architecture of the computer system.
- Programs written in a computer's true machine language on a hardwired computer can be executed directly by the electronic circuits without any interpreters, translators, or compilers.

Level 1

- Level 1, the Control Level, is where a *control unit* makes sure that instructions are decoded and executed properly and that data is moved where and when it should be.
- The control unit interprets the machine instructions passed to it, one at a time, from the level above, causing the required actions to take place.
- Control units can be designed in one of two ways: They can be *hardwired* or they can be *microprogrammed*.

Hardwired Control Units

- In hardwired control units, control signals emanate from blocks of digital logic components. These signals transmitted to all appropriate parts of the system.
- Hardwired control units are typically very fast because they are actually physical components. However, once implemented, they are very difficult to modify for the same reason.

Microprogrammed

- A microprogram is a program written in a low-level language that is implemented directly by the hardware.
- Machine instructions produced in Level 2 are fed into this microprogram, which then interprets the instructions by activating hardware suited to execute the original instruction.
- One machine-level instruction is often translated into several microcode instructions.
- Microprograms are popular because they can be modified relatively easily.
- The disadvantage of microprogramming is, of course, that the additional layer of translation typically results in slower instruction execution.

Level 0

- Level 0, the Digital Logic Level, is where we find the physical components of the computer system: the gates and wires.
- These are the fundamental building blocks, the implementations of the mathematical logic that are common to all computer systems.

The Von Neumann Model

In the earliest electronic computing machines, programming was synonymous with connecting wires to plugs. No layered architecture existed, so programming a computer was like play with wires and plugs. John W. Mauchly and J. Presper Eckert conceived of an easier way to change the behavior of their calculating machine. They reckoned that memory devices, in the form of mercury delay lines, could provide a way to store program instructions. This would forever end the tedium of rewiring the system each time it had a new problem to solve, or an old one to debug. Mauchly and Eckert could not immediately publish their insight due to World War II. After reading Mauchly and Eckert's proposal for the EDVAC, Von Neumann published and publicized the idea.

All stored-program computers have come to be known as *von Neumann systems* using the *von Neumann architecture*. Today's version of the stored-program machine architecture satisfies at least the following characteristics:

- Consists of three hardware systems: A *central processing unit* (*CPU*) with a control unit, an *arithmetic logic unit* (*ALU*), *registers* and a program counter; a *main-memory system,* which holds programs that control the computer's operation; and an *I/O system.*
- Capacity to carry out sequential instruction processing
- Contains a single path, either physically or logically, between the main memory system and the control unit of the CPU, forcing alternation of instruction and execution cycles. This single path is often referred to as the *von Neumann bottleneck*.

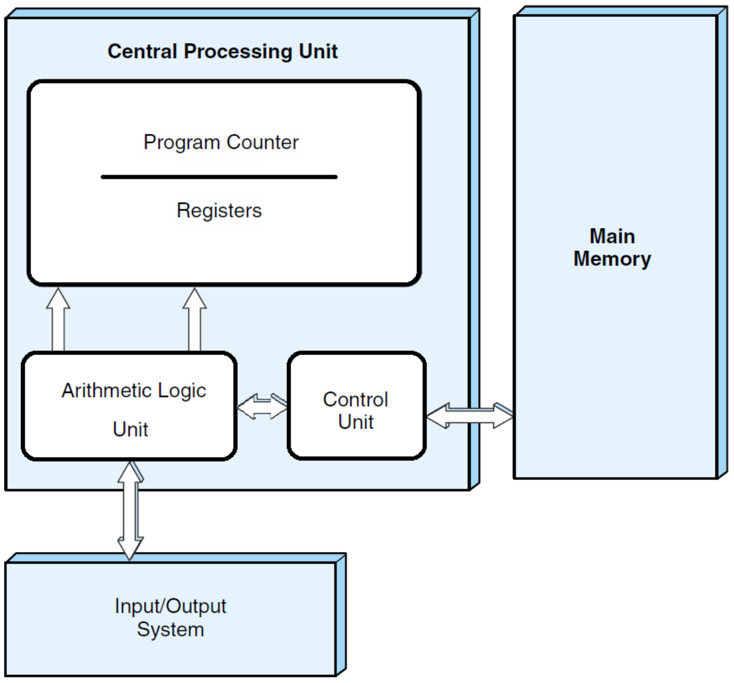


Fig. The von Neumann Architecture

Figure shows how these features work together in modern computer systems. The system shown in the figure passes all of its I/O through the arithmetic logic unit. This architecture runs programs in what is known as the *von Neumann execution cycle* (also called the *fetch-decode-execute cycle*), which describes how the machine works. One iteration of the cycle is as follows:

- The control unit fetches the next program instruction from the memory, using the program counter to determine where the instruction is located.
- The instruction is decoded into a language the ALU can understand.
- Any data operands required to execute the instruction are fetched from memory and placed into registers within the CPU.
- The ALU executes the instruction and places the results in registers or memory.

The ideas present in the von Neumann architecture have been extended so that programs and data stored in a slow-to-access storage medium, such as a hard disk, can be copied to a fast-access, volatile storage medium such as RAM prior to execution. This architecture has also been streamlined into what is currently called the *system bus model*, which is shown in Figure 1.5.

The data bus moves data from main memory to the CPU registers (and vice versa).

- The address bus holds the address of the data that the data bus is currently accessing.
- The control bus carries the necessary control signals that specify how the information transfer is to take place.

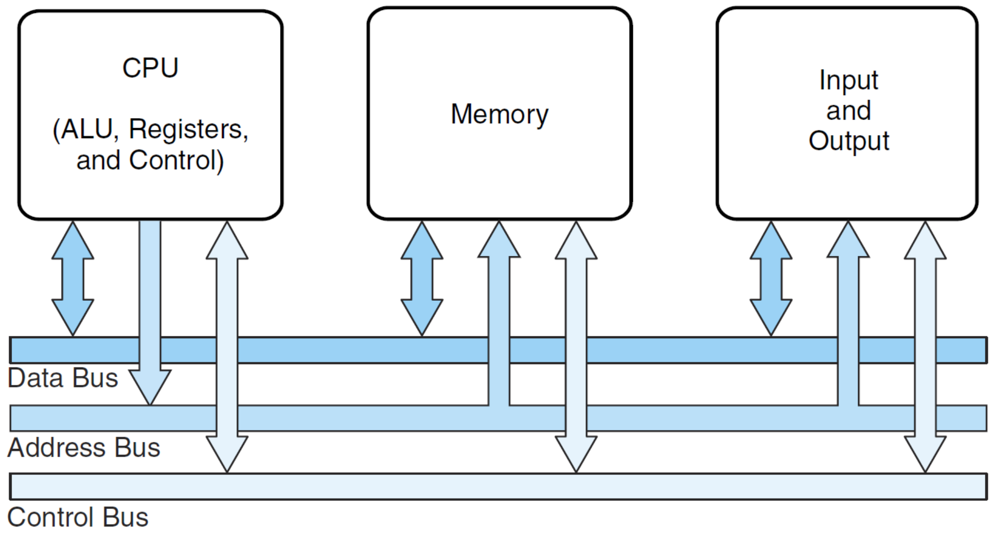


Fig. The Modified von Neumann Architecture, Adding a System Bus

Non–Von Neumann Models

Until recently, almost all general-purpose computers followed the ***von Neumann*** design. However, the von Neumann bottleneck continues to baffle engineers looking for ways to build fast systems that are inexpensive and compatible with the vast body of commercially available software.

Engineers who are not constrained by the need to maintain compatibility with von Neumann systems are free to use many different models of computing. A number of different subfields fall into the ***non-von Neumann*** category, including neural networks, genetic algorithms, quantum computation and parallel computers. Parallel computing is currently the most popular.

Even parallel computing has its limits, however. As the number of processors increases, so does the overhead of managing how tasks are distributed to those processors. Some parallel-processing systems require extra processors just to manage the rest of the processors and the resources assigned to them.

No matter how many processors are placed in a system, or how many resources are assigned to them, somehow, somewhere, a bottleneck is bound to develop. The best that we can do to remedy this is to make sure that the slowest parts of the system are the ones that are used the least. This is the idea behind *Amdahl's Law*.

Amdahl's Law

This law states that the performance enhancement possible with a given improvement is limited by the amount that the improved feature is used. The underlying premise is that every algorithm has a sequential part that ultimately limits the speedup that can be achieved by multiprocessor implementation.

The Evolution of the Intel x86 Architecture

The x86 architecture continues to dominate the processor market outside of embedded systems. Although the organization and technology of the x86 machines has changed dramatically over the decades, the instruction set architecture has evolved to remain backward compatible with earlier versions. The first 8086 was introduced with a clock speed of 5 MHz and had 29,000 transistors. In 2008 A quad-core Intel Core 2 introduced which operates at 3 GHz, a speedup of a factor of 600, and has 820 million transistors, about 28,000 times as many as the 8086. It is worthwhile to list some of the highlights of the evolution of the Intel x86 Architecture:

- **8080:** The world's first general-purpose microprocessor. This was an 8-bit machine, with an 8-bit data path to memory. The 8080 was used in the first personal computer.
- **8086:** A far more powerful, 16-bit machine. In addition to a wider data path and larger registers, the 8086 has an instruction cache, or queue, that prefetches a few instructions before they are executed. The 8086 is the first appearance of the x86 architecture.
- **80286:** This extension of the 8086 enabled addressing a 16-MByte memory instead of just 1 MByte.
- **80386:** Intel's first 32-bit machine, and a major overhaul of the product. With a 32-bit architecture, the 80386 rivaled the complexity and power of minicomputers and mainframes introduced just a few years earlier. This was the first Intel processor to support multitasking, meaning it could run multiple programs at the same time.
- **80486:** The 80486 introduced the use of much more sophisticated and powerful cache technology and sophisticated instruction pipelining. The 80486 also offered a built-in math coprocessor, offloading complex math operations from the main CPU.
- **Pentium:** With the Pentium, Intel introduced the use of superscalar techniques, which allow multiple instructions to execute in parallel.
- **Pentium Pro:** The Pentium Pro continued the move into superscalar organization begun with the Pentium, with aggressive use of register renaming, branch prediction, data flow analysis, and speculative execution.
- **Pentium II:** The Pentium II incorporated Intel MMX technology, which is designed specifically to process video, audio, and graphics data efficiently.
- **Pentium III:** The Pentium III incorporates additional floating-point instructions to support 3D graphics software.
- **Pentium 4:** The Pentium 4 includes additional floating-point and other enhancements for multimedia
- **Core:** This is the first Intel x86 microprocessor with a dual core, referring to the implementation of two processors on a single chip.
- **Core 2:** The Core 2 extends the architecture to 64 bits. The Core 2 Quad provides four processors on a single chip.

Performance

Time is the measure of computer performance: the computer that performs the same amount of work in the least time is the fastest. Program *response time or execution time* is measured in seconds per program. Computers are often shared, however, and a processor may work on several programs simultaneously. In such cases, the system may try to optimize throughput rather than attempt to minimize the response time for one program. Hence, we often want to distinguish between the response time and the time that the processor is working on our behalf.

Response 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

- Another measure of performance, it is the number of tasks completed per unit time.

Relative Performance

To maximize performance, we want to minimize response time or execution time for some task. Thus, we can relate performance and execution time for a computer X:

$$\text{Performance}_X = \frac{1}{\text{Execution Time}_X}$$

This means that for two computers X and Y, if the performance of X is greater than the performance of Y, we have

$$\text{Performance}_X > \text{Performance}_Y$$

$$\frac{1}{\text{Execution Time}_X} > \frac{1}{\text{Execution Time}_Y}$$

$$\text{Execution Time}_Y > \text{Execution Time}_X$$

That is, the execution time on Y is longer than that on X, if X is faster than Y. In discussing a computer design, we often want to relate the performance of two different computers quantitatively. We will use the phrase “X is *n* times faster than Y” or equivalently “X is *n* times as fast as Y”. So we can write

$$\frac{\text{Performance}_X}{\text{Performance}_Y} = n$$

If X is *n* times faster than Y, then the execution time on Y is *n* times longer than it is on X:

$$\frac{\text{Performance}_X}{\text{Performance}_Y} = \frac{\text{Execution Time}_Y}{\text{Execution Time}_X} = n$$

Prob. 1	If computer A runs a program in 10 seconds and computer B runs the same program in 15 seconds, how much faster is A than B?

We know that A is n times faster than B if

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

Thus the performance ratio is

$$\frac{\text{Performance}_A}{\text{Performance}_B} = \frac{15}{10} = 1.5$$

and A is therefore 1.5 times faster than B.

Measuring Performance

CPU Execution Time or **CPU Time**

This is the actual time spend by the CPU for computing a task and does not include time spent waiting for I/O or running other programs. CPU time can be further divided into two parts:

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.

The term ***system performance*** is used to refer ***Response Time*** and ***CPU performance*** to refer ***User CPU Time*** for unloaded system.

Clock Cycle

Almost all computers are constructed using a clock that determines when events take place in the hardware. These discrete time intervals are called **clock cycles** (or **ticks**, **clock ticks**, **clock periods**, **clocks**, **cycles**).

Clock Period

- Clock Period is the length of each clock cycle.

Designers refer to the length of a **clock period** as

- The time for a complete *clock cycle* (e.g., 250 picoseconds, or 250 ps).
- The *clock rate* (e.g., 4 gigahertz, or 4 GHz), which is the inverse of the clock period.

CPU Performance and Its Factors

A simple formula relates the clock cycles and clock cycle time to CPU time for a program:

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

Alternatively, because clock rate and clock cycle time are inverses,

$$\text{CPU execution time} = \frac{\text{CPU clock cycles}}{\text{Clock Rate}}$$

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.

Prob. 2 A program runs in 10 seconds on computer A, which has a 2 GHz clock. We are trying to help a computer designer build a computer, B, which will run this program in 6 seconds. The designer has determined that a substantial increase in the clock rate is possible, but this increase will affect the rest of the CPU design, causing computer B to require 1.2 times as many clock cycles as computer A for this program. What clock rate should we tell the designer to target?

Let's first find the number of clock cycles required for the program on A:

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

$$10\ Sec. = \frac{\text{CPU Clock Cycles}_A}{2 \times 10^9\ Cycles/Sec.}$$

$$\text{CPU Clock Cycles}_A = 10 \times 2 \times 10^9 = 20 \times 10^9\ cycles$$

CPU time for B can be found using this equation:

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

$$6\ sec = \frac{1.2 \times 20 \times 10^9 cycles}{Clock\ Rate\ _B}$$

$$Clock\ Rate\ _B = \frac{1.2 \times 20 \times 10^9 cycles}{6\ sec} = \frac{0.2 \times 20 \times 10^9 cycles}{sec}$$

$$Clock\ Rate\ _B = \frac{4 \times 10^9 cycles}{sec} = 4\ GHz$$

To run the program in 6 seconds, B must have twice the clock rate of A.

Instruction Performance

The performance equations till know, not include any reference to the number of instructions needed for the program. However, since the compiler clearly generated instructions to execute, and the computer had to execute the instructions to run the program, the execution time must depend on the number of instructions in a program. In other words response time or execution time is that it equals the number of instructions executed multiplied by the average time per instruction. Therefore, the number of clock cycles required for a program can be written as

$$CPU\ clock\ cycles = Instructions\ for\ a\ program \times Average\ clock\ cycles\ per\ instruction$$

- The term ***clock cycles per instruction***, which is the average number of clock cycles each instruction takes to execute, is often abbreviated as **CPI**.
- 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 provides one way of comparing two different implementations of the same instruction set architecture, since the number of instructions executed for a program will, of course, be the same.

Prob. 3

If we have two implementations of the same instruction set architecture. Computer A has a clock cycle time of 250 ps and a CPI of 2.0 for some program, and computer B has a clock cycle time of 500 ps and a CPI of 1.2 for the same program. Which computer is faster for this program and by how much?

We know that each computer executes the same number of instructions for the program; let's call this number *I*. First, find the number of processor clock cycles for each computer:

$$CPU\ clock\ cycles_A = I \times 2.0$$

$$CPU\ clock\ cycles_B = I \times 1.2$$

Now we can compute the CPU time for each computer:

$$CPU\ time\ _A = CPU\ clock\ cycles_A \times Clock\ cycle\ time$$

$$CPU\ time\ _A = I \times 2 \times 250ps = 500 \times I\ ps$$

Likewise, for B:

$$CPU\ time\ _B = CPU\ clock\ cycles_B \times Clock\ cycle\ time$$

$$CPU\ time\ _B = I \times 1.2 \times 500\ ps = 600 \times I\ ps$$

Clearly, computer A is faster. The amount faster is given by the ratio of the execution times:

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

We can conclude that computer A is 1.2 times as fast as computer B for this program.

The Classic CPU Performance Equation

We can write the basic performance equation in terms of **instruction count** (the number of instructions executed by the program), CPI, and clock cycle time and known as *The Classic CPU Performance Equation*:

$$CPU\ time = Instructions\ count \times CPI \times clock\ cycle\ time$$

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

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

These formulas are particularly useful because they separate the three key factors that affect performance. We can use these formulas to compare two different implementations or to evaluate a design alternative if we know its impact on these three parameters.

We can see how these factors are combined to yield execution time measured in seconds per program:

$$Time = \frac{Seconds}{Program} = \frac{Instructions}{Program} \times \frac{Clock\ Cycles}{Instruction} \times \frac{Seconds}{Clock\ cycle}$$

Prob. 3 A compiler designer is trying to decide between two code sequences for a particular computer. The hardware designers have supplied the following facts:

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

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

	Instruction counts for each instruction class		
Code sequence	A	B	C
1	2	1	2
2	4	1	1

Which code sequence executes the most instructions? Which will be faster? What is the CPI for each sequence?

Sequence 1 executes 2 + 1 + 2 = 5 instructions. Sequence 2 executes 4 + 1 + 1 = 6 instructions. Therefore, sequence 1 executes fewer instructions.

We can use the equation for CPU clock cycles based on instruction count and CPI to find the total number of clock cycles for each sequence:

$$CPU\ clock\ cycle = \sum_{i=1}^{n} (CPI_i \times C_i)$$

This yields

$$CPU\ clock\ cycles_1 = (2 \times 1) + (1 \times 2) + (2 \times 3) = 2 + 2 + 6 = 10\ cycles$$

$$CPU\ clock\ cycles_2 = (4 \times 1) + (1 \times 2) + (1 \times 3) = 4 + 2 + 3 = 9\ cycles$$

So code sequence 2 is faster, even though it executes one extra instruction. Since code sequence 2 takes fewer overall clock cycles but has more instructions, it must have a lower CPI. The CPI values can be computed by

$$CPI = \frac{CPU\ clock\ cycles}{Instruction\ count}$$

$$CPI_1 = \frac{CPU\ clock\ cycles_1}{Instruction\ count_1} = \frac{10}{5} = 2.0$$

$$CPI_2 = \frac{CPU\ clock\ cycles_2}{Instruction\ count_2} = \frac{9}{6} = 1.5$$