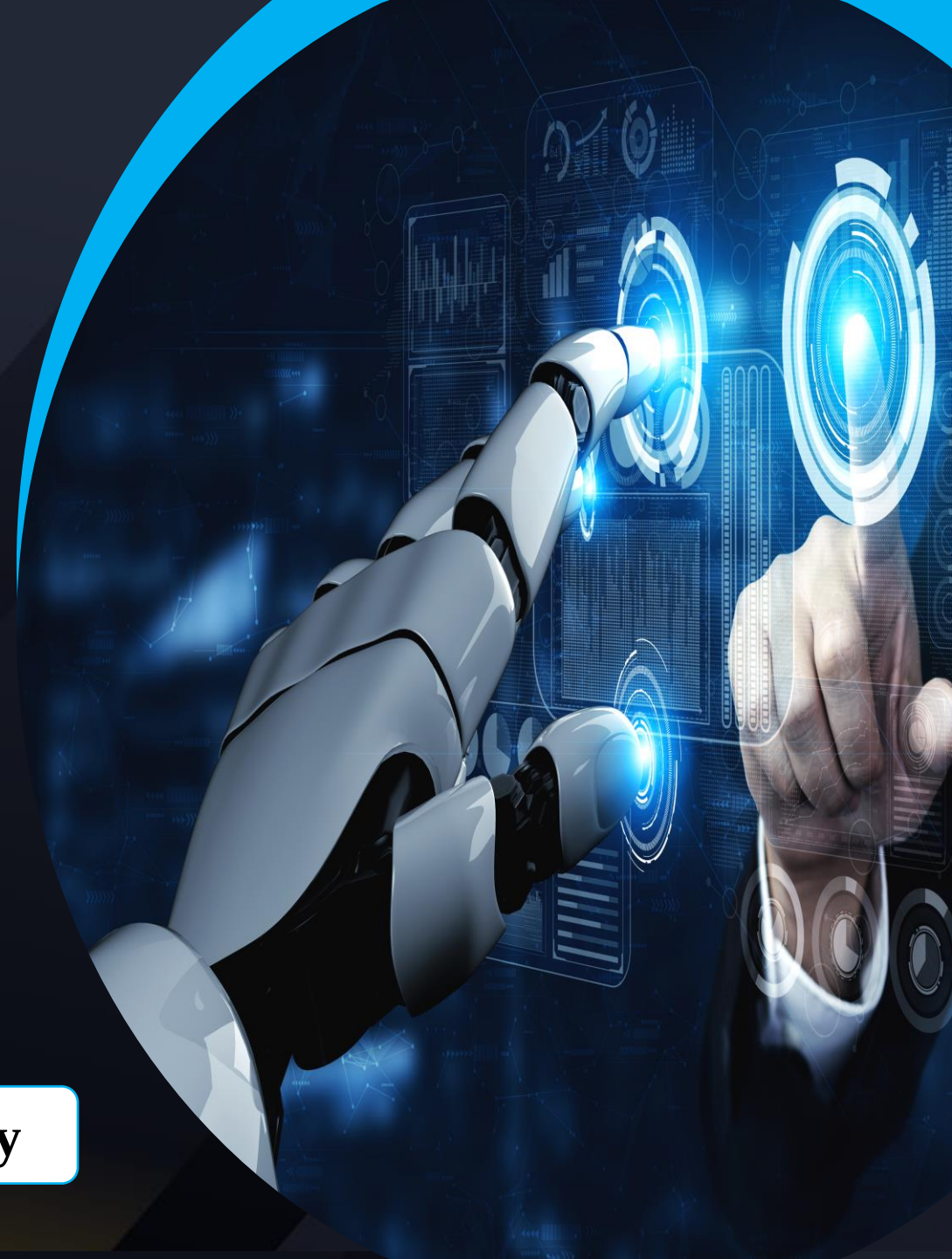


Computer Organization and Architecture

Prepared by Dr. Eng. Gamal Selim

Lecture 01 - Computer Abstractions and Technology



In this lecture, we will discuss the following topics:

- **Classes of Computers**
- **The PostPC Era**
- **From a High-Level Language to the Language of Hardware**
- **Safe Place for Data**
- **Technologies for Building Processors and Memory**
- **Performance**
 - ✓ Defining Performance
 - ✓ Throughput and Response Time
 - ✓ Measuring Performance
 - ✓ CPU Performance and Its Factors
 - ✓ Understanding Program Performance

Lecture 01

Classes of Computers

Personal computers

General purpose, variety of software
Subject to cost/performance tradeoff

Server computers

Network based
High capacity, performance, reliability
Range from small servers to building sized

Lecture 01

Classes of Computers

Supercomputers

High-end scientific and engineering calculations

Highest capability but represent a small fraction of the overall computer market

Embedded computers

Hidden as components of systems

Stringent power/performance/cost constraints

Lecture 01

The PostPC Era

Personal Mobile Device (PMD)

Battery operated

Connects to the Internet

Hundreds of dollars

Smart phones, tablets, electronic glasses

Cloud computing

Warehouse Scale Computers (WSC)

Software as a Service (SaaS)

Portion of software run on a PMD and a portion run in the Cloud

Amazon and Google

Lecture 01

Below Your Program

Application software

Written in high-level language

System software

Compiler: translates HLL code to machine code

Operating System: service code

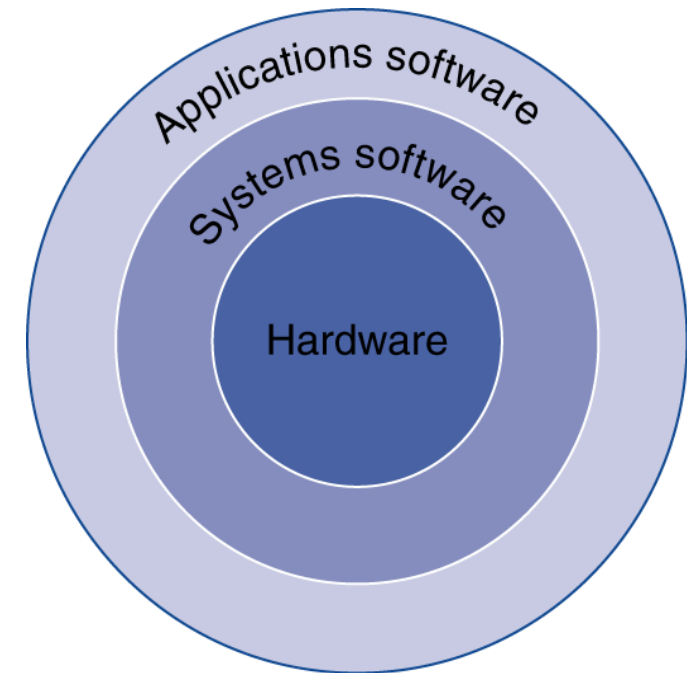
- Handling input/output

- Managing memory and storage

- Scheduling tasks & sharing resources

Hardware

Processor, memory, I/O controllers



Lecture 01

From a High-Level Language to the Language of Hardware

To actually speak to electronic hardware, you need to send electrical signals. The easiest signals for computers to understand are *on and off*, and so the computer alphabet is just two letters.

Just as the 26 letters of the English alphabet do not limit how much can be written, the two letters of the computer alphabet do not limit what computers can do. The two symbols for these two letters are the numbers 0 and 1, and we commonly think of the computer language as numbers in base 2, or *binary numbers*.

From a High-Level Language to the Language of Hardware

We refer to each “letter” as a **binary digit** or **bit**. Computers are slaves to our commands, which are called **instructions**. Instructions, which are just collections of bits that the computer understands and obeys, can be thought of as numbers.

For example, the bits **1000110010100000** tell one computer to add two numbers

Binary digit Also called a **bit**. One of the two numbers in base 2 (0 or 1) that are the components of information.

Instruction A command that computer hardware understands and obeys.

From a High-Level Language to the Language of Hardware

The first programmers communicated to computers in binary numbers, but this was so tedious that they quickly invented new notations that were closer to the way humans think. At first, these notations were translated to binary by hand, but this process was still tiresome. Using the computer to help program the computer, the pioneers invented programs to translate from symbolic notation to binary. The first of these programs was named an **assembler**.

From a High-Level Language to the Language of Hardware

This program translates a symbolic version of an instruction into the binary version. For example, the programmer would write

add A,B

and the assembler would translate this notation into

1000110010100000

This instruction tells the computer to add the two numbers A and B. The name coined for this symbolic language, still used today, is **assembly language**. In contrast, the binary language that the machine understands is the **machine language**.

Assembler A program that translates a symbolic version of instructions into the binary version.

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

High-level
language
program
(in C)

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

Compiler

Assembly
language
program
(for MIPS)

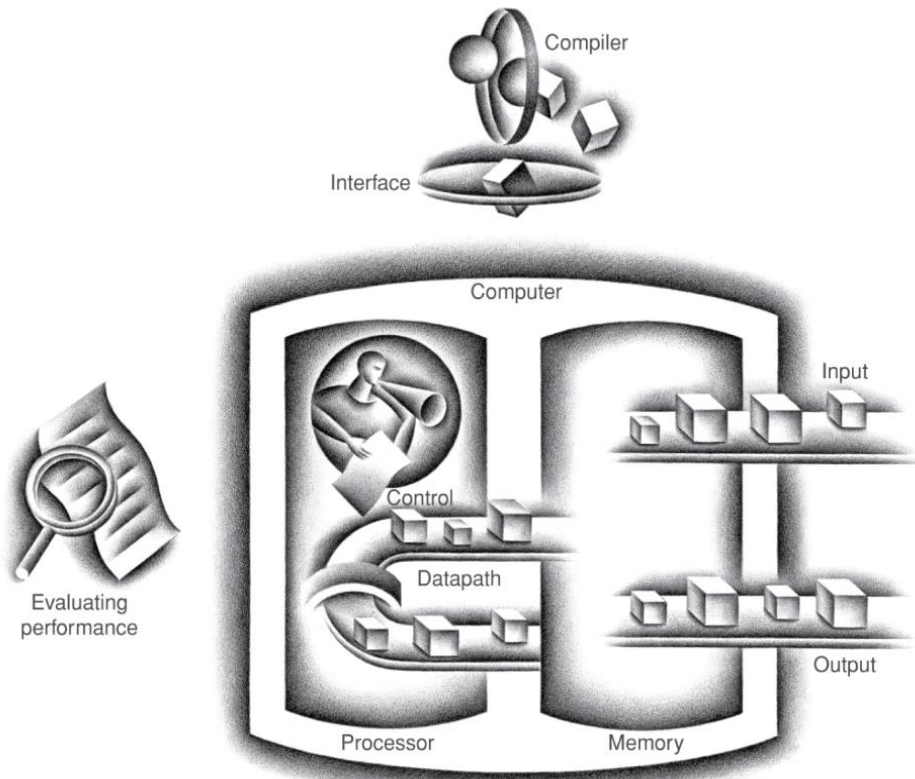
```
swap:
    muli $2, $5, 4
    add  $2, $4, $2
    lw   $15, 0($2)
    lw   $16, 4($2)
    sw   $16, 0($2)
    sw   $15, 4($2)
    jr   $31
```

Assembler

Binary machine
language
program
(for MIPS)

```
000000001010000100000000000011000
000000000000110000001100000100001
10001100011000100000000000000000
100011001111001000000000000000100
101011001111001000000000000000000
101011000110001000000000000000100
00000011111000000000000000001000
```

The BIG Picture



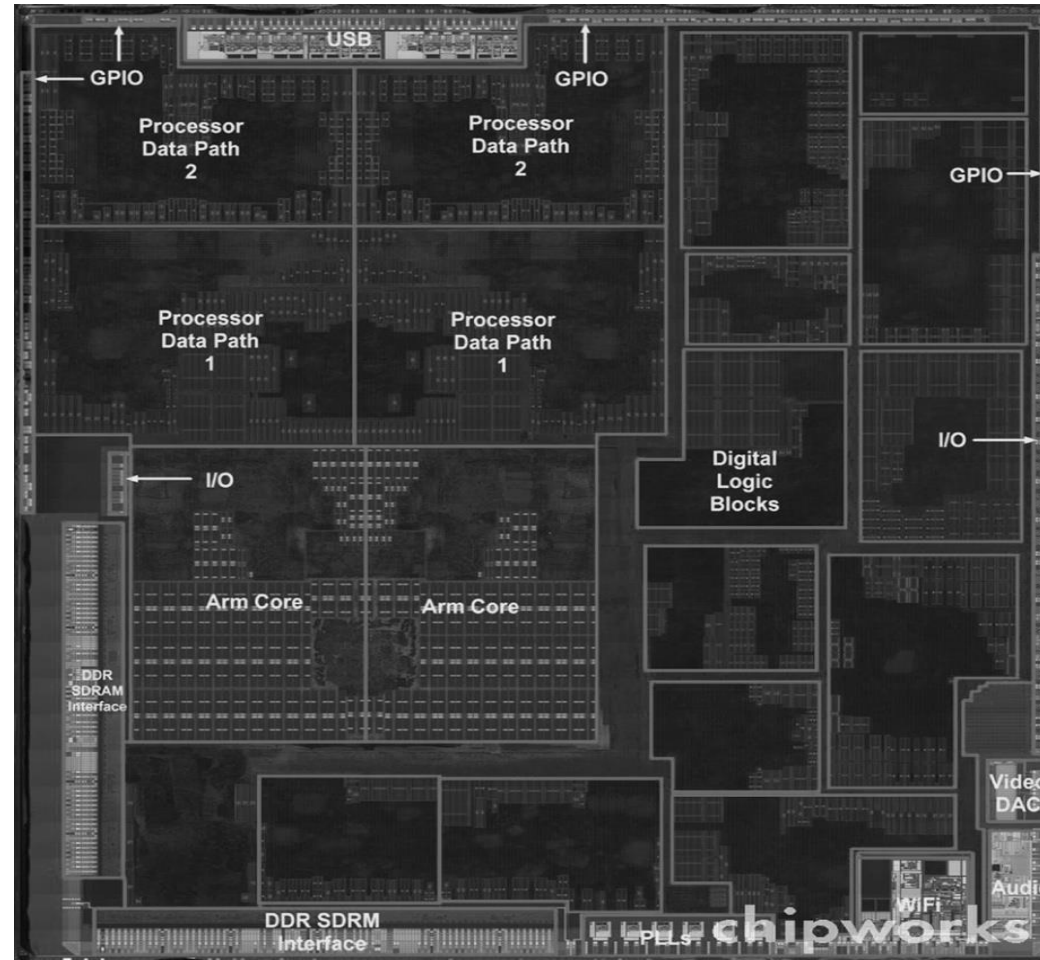
- Same components for all kinds of computer
 - Desktop, server, embedded
- **Input/output includes**
 - User-interface devices
 - Display, keyboard, mouse
 - Storage devices
 - Hard disk, CD/DVD, flash
 - Network adapters
 - For communicating with other computers

Data path: performs operations on data

Control: sequences data path, memory, ...

Cache memory: Small fast SRAM memory for immediate access to data

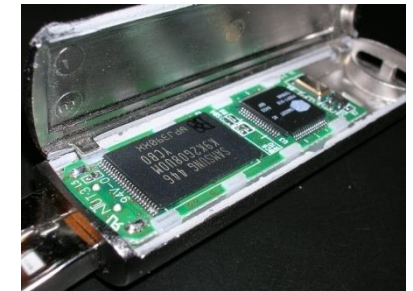
- Apple A5



Lecture 01

A Safe Place for Data

- **Volatile main memory**
 - Loses instructions and data when power off
- **Non-volatile secondary memory**
 - Magnetic disk
 - Flash memory
 - Optical disk (CDROM, DVD)



Lecture 01

A Safe Place for Data

Thus far, we have seen how to input data, compute using the data, and display data. If we were to lose power to the computer, however, everything would be lost because the memory inside the computer is **volatile**—that is, when it loses power, it forgets. In contrast, a DVD disk doesn't forget the movie when you turn off the power to the DVD player, and is thus a **nonvolatile memory** technology.

To distinguish between the volatile memory used to hold data and programs while they are running and this nonvolatile memory used to store data and programs between runs, the term **main memory** or **primary memory** is used for the former, and **secondary memory** for the latter. Secondary memory forms the next lower layer of the **memory hierarchy**. DRAMs have dominated main memory since 1975, but **magnetic disks** dominated secondary memory starting even earlier. Because of their size and form factor, personal Mobile Devices use **flash memory**, a nonvolatile semiconductor memory, instead of disks.

We've explained how we can input, compute, display, and save data, but there is still one missing item found in today's computers: computer networks. **Networked computers have several major advantages:**

- **Communication:** Information is exchanged between computers at high speeds.
- **Resource sharing:** Rather than each computer having its own I/O devices, computers on the network can share I/O devices.
- **Nonlocal access:** By connecting computers over long distances, users need not be near the computer they are using.

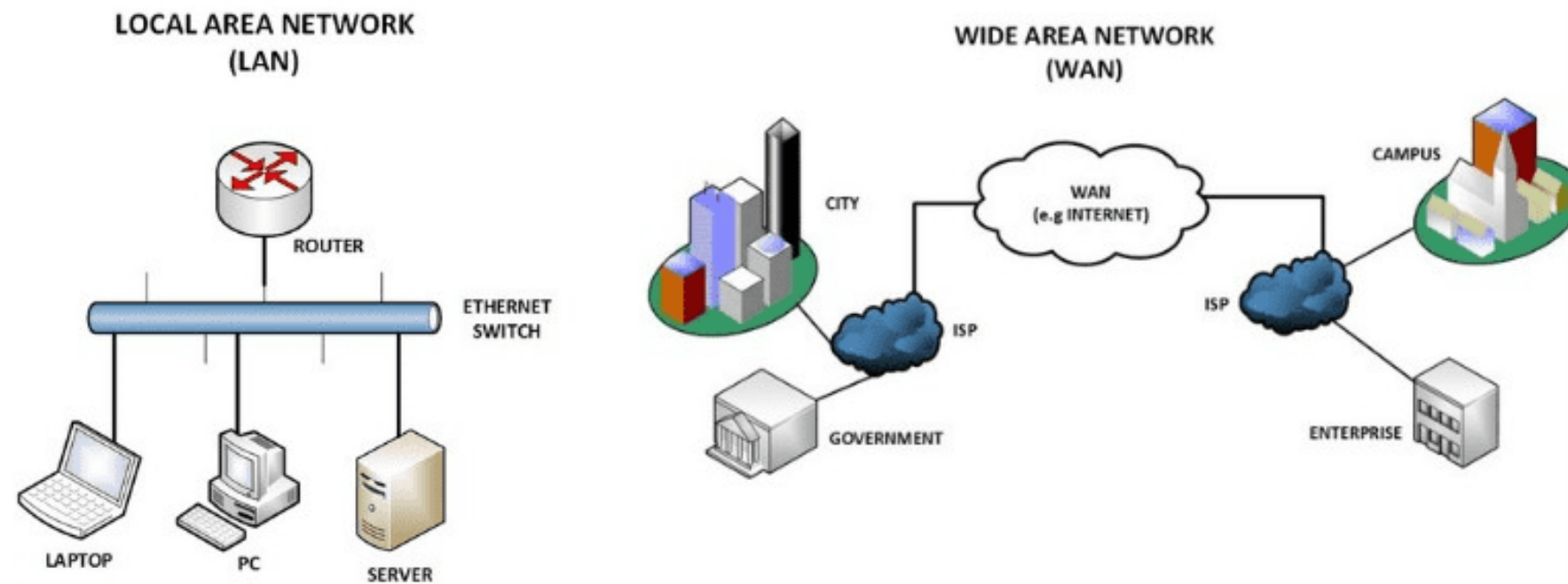
Lecture 01

Communicating with Other Computers

Local Area Network (LAN) A network designed to carry data within a geographically confined area, typically within a single building.

Wide Area Network (WAN) A network extended over hundreds of kilometers that can span a continent.





LAN vs WAN

- Assessing the performance of computers can be quite challenging
- When trying to choose among different computers, performance is an important attribute. Accurately measuring and comparing different computers is critical to purchasers and therefore to designers. The people selling computers know this as well.

As an individual computer user, you are interested in reducing **response time**—the time between the start and completion of a task—also referred to as **execution time**. Datacenter managers are often interested in increasing **throughput** or **bandwidth**—the total amount of work done in a given time. Hence, in most cases, we will need different performance metrics as well as different sets of applications to benchmark personal mobile devices, which are more focused on response time, versus servers, which are more focused on throughput.

Response time

How long it takes to do a task

Throughput

Total work done per unit time

e.g., tasks/transactions/... per hour

Do the following changes to a computer system increase throughput, decrease response time, or both?

1. Replacing the processor in a computer with a faster version
2. Adding additional processors to a system that uses multiple processors for separate tasks—for example, searching the web

Decreasing response time almost always improves throughput. Hence, **in case 1**, both response time and throughput are improved. **In case 2**, no one task gets work done faster, so only throughput increases.

If, however, the demand for processing in the second case was almost as large as the throughput, the system might force requests to queue up. In this case, increasing the throughput could also improve response time, since it would reduce the waiting time in the queue. Thus, in many real computer systems, changing either execution time or throughput often affects the other.

In discussing the performance of computers, we will be primarily concerned with response time for the first few chapters. 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

$$\begin{aligned} \text{Performance}_X &> \text{Performance}_Y \\ \frac{1}{\text{Execution time}_X} &> \frac{1}{\text{Execution time}_Y} \\ \text{Execution time}_Y &> \text{Execution time}_X \end{aligned}$$

That is, the execution time on Y is longer than that on X, if X is faster than Y.

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

Lecture 01

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. Program *execution time* is measured in seconds per program. However, time can be defined in different ways, depending on what we count. The most straightforward definition of time is called *wall clock time*, *response time*, or *elapsed time*. These terms mean the total time to complete a task, including disk accesses, memory accesses, *input/output* (I/O) activities, operating system overhead—everything

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 elapsed time for one program. Hence, we often want to distinguish between the elapsed time and the time over which the processor is working on our behalf. **CPU execution time** or simply **CPU time**, which recognizes this distinction, is the time the CPU spends computing for this task and does not include time spent waiting for I/O or running other programs.

CPU execution time Also called **CPU time**. The actual time the CPU spends computing for a specific task.

CPU time can be further divided into the CPU time spent in the program, called **user CPU time**, and the CPU time spent in the operating system performing tasks on behalf of the program, called **system CPU time**. Differentiating between system and user CPU time is difficult to do accurately, because it is often hard to assign responsibility for operating system activities to one user program rather than another and because of the functionality differences among operating systems.

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

For consistency, we maintain a distinction between performance based on elapsed time and that based on CPU execution time. We will use the term *system performance* to refer to elapsed time on an unloaded system and ***CPU performance*** to refer to user ***CPU time***

Different applications are sensitive to different aspects of the performance of a computer system. Many applications, especially those running on servers, depend as much on **I/O performance**, which, in turn, relies on both hardware and software. Total elapsed time measured by a wall clock is the measurement of interest.

In some application environments, the user may care about throughput, response time, or a complex combination of the two (e.g., **maximum throughput with a worst-case response time**).

To improve the performance of a program, one must have a clear definition of what performance metric matters and then proceed to look for performance bottlenecks by measuring program execution and looking for the likely bottlenecks. **In the following chapters, we will describe how to search for bottlenecks and improve performance in various parts of the system.**

Designers refer to the length of a **clock period** both as the time for a complete *clock cycle* (e.g., 250 picoseconds, or 250 ps) and as the *clock rate* (e.g., 4 gigahertz, or 4 GHz), which is the inverse of the clock period. In the next subsection, we will formalize the relationship between the clock cycles of the hardware designer and the seconds of the computer user.

1. Suppose we know that an application that uses both personal mobile devices and the Cloud is limited by network performance. For the following changes, state whether only the throughput improves, both response time and throughput improve, or neither improves.
 - a. An extra network channel is added between the PMD and the Cloud, increasing the total network throughput and reducing the delay to obtain network access (since there are now two channels).

- b. The networking software is improved, thereby reducing the network communication delay, but not increasing throughput.
- c. More memory is added to the computer.

2. Computer C's performance is 4 times as fast as the performance of computer B, which runs a given application in 28 seconds. How long will computer C take to run that application?

Users and designers often examine performance using different metrics. If we could relate these different metrics, we could determine the effect of a design change on the performance as experienced by the user. Since we are confining ourselves to CPU performance at this point, the bottom-line performance measure is CPU execution time. A simple formula relates the most basic metrics (clock cycles and clock cycle time) to CPU time:

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

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

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

Lecture 01

Understanding Program Performance

The performance of a program depends on the algorithm, the language, the compiler, the architecture, and the actual hardware. The following table summarizes how these components affect the factors in the CPU performance equation.

The term **clock cycles per instruction**, which is the average number of clock cycles each instruction takes to execute, is often abbreviated as **CPI**

Hardware or software component	Affects what?	How?
Algorithm	Instruction count, possibly CPI	The algorithm determines the number of source program instructions executed and hence the number of processor instructions executed. The algorithm may also affect the CPI, by favoring slower or faster instructions. For example, if the algorithm uses more divides, it will tend to have a higher CPI.
Programming language	Instruction count, CPI	The programming language certainly affects the instruction count, since statements in the language are translated to processor instructions, which determine instruction count. The language may also affect the CPI because of its features; for example, a language with heavy support for data abstraction (e.g., Java) will require indirect calls, which will use higher CPI instructions.
Compiler	Instruction count, CPI	The efficiency of the compiler affects both the instruction count and average cycles per instruction, since the compiler determines the translation of the source language instructions into computer instructions. The compiler's role can be very complex and affect the CPI in complex ways.
Instruction set architecture	Instruction count, clock rate, CPI	The 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.



مجموعة جامعة
الجميع الذكية
للتعليم القابضة

MIDOCEAN
UNIVERSITY
BY EVERYONE'S SMART UNIVERSITY



Thank You

