# Lesson 1 - What Is A Computer?

Carlos Luna

2024-2025

## Contents

# 1 Overview

Congratulations on taking your first step on learning computer science! Before we begin, have you ever considered what a computer is? What the most important components are? In this introductory lesson, I describe the major components of a computer (that are important to introductory computer science).

# 2 CPU

The component that traditionally does most of the heavy lifting is the **Central Processing Unit**, almost always referred to as just the "**CPU**" unless you want to be different in what is likely a bad way. This chip is what does it all: read values from memory and storage, performs all calculations the computer has to do [1], and is what communicates with every other device/component to make it all work.

Modern CPUs also contain a very important component, called the **cache**. The CPU cache is extraordinarily helpful piece of silicon, effectively giving the CPU an exceptionally fast short term memory, and provides a great performance benefit to the currently running program. While only the CPU itself can really access this component, we can sometimes write code in a way that encourages the CPU to use it[2]. However, in order to be as fast as it can be, the cache also has to be very small. As a result, at best it can really only store enough data for small calculations at best.

The CPU also has **registers**, which can only hold a single value. However, registers are also the fastest thing we can get, and is what the CPU will store values in that are currently in use. For example, if the computer is calculating "3 * 10", then it will place "3", "10", and the result "30" into a register [3].

However, since this kind of value storage is fairly small and tied directly to the CPU (if the CPU is turned off, we lose all of the values), we can't use it for storing any permanent or "large" (e.g., a mouse click sound effect) data.

# 3 Storage

Since we obviously need a way to store large/permanent data, we have developed such storage methods, ranging from punch cards to modern SSDs (which are now found on the latest generation of gaming consoles and most new computers). From a beginner comp sci standpoint though, we don't really care how fast this method of storage is, just that we have some method of permanently storing large amounts of data. We shall refer to this kind of computer memory as **storage**.

However, relative to the CPU cache and registers, even the fastest SSDs are still pretty slow and would cause a significant amount of slow down. So, it would be really convenient if something existed that could store that is kind of in between the CPU cache/registers and "storage".

# 4 Memory

Luckily, such a data storage type exists. **Random Access Memory (RAM)** is a storage component that fulfills this requirement. RAM is used by applications to store "temporary"-ish data, but it can really store anything that can be stored in storage, if there's enough RAM available. For example, a messaging application might store user avatar images and the currently active thread in RAM, as none of the data is actually "permanent", as the images and messages can just be retrived from the application's server.

[1] Technically, all of the components do calculations (such as the Graphics Processing Unit for displaying anything, or the RAM calculating memory offsets)

[2] "Writing Cache Friendly Code" lecture slides

[3] That is, if enough registers are available, but that's a senior level college class kind of problem.

## 5 How are values represented?

At this point, you might be wondering, "how a computer can store data"? Fundamentally, all a computer knows is if a wire/piece of metal has +5V or -5V of electricity. What we can do with this, is use the base 2 number system, binary, and assign a 1 or a 0. From there, we can do pretty much everything, although it would make it easy for us to let a computer convert binary to decimal, since we already know the decimal number system. As a result, people have developed ways to represent a wide variety of things, like integers, floating point numbers (like 3.14), images, and audio. For now, we focus on *unsigned* integers, which start at 0 and are never negative (they have the range $[0, \infty)$).

We begin looking at binary by first looking at how a decimal number actually works. I imagine you all have a pretty intuitive understanding of what "100" is, but what about the way it's written gives it its meaning? In decimal, we only have the digits {0, 1, 2, 3, 4, 5, 6, 8, 9}, so how is it really that we can express larger quantities? We know that if we want to go past 9, we reset the digit to zero and add a one in front of it, but have you ever thought about what this is actually doing? You might be thinking about how the leftmost digit carries the most "weight" in the number, and that's part of the way there. But what exactly is this "weight"? In a number base system of base $b$ ($b = 2$ for binary, $b = 10$ for decimal), each digit in a number of base $b$ ($n_b$) carries a weight of $b^i$, where $i$ is the "index" of the digit $(n_i)_b$: [4]

$$n_b = (...n_3 n_2 n_1 n_0)_b$$

For example, let's use the decimal number 35014, so $b = 10$, and $n_b = 35014_{10}$. Then, we would have that

$$n_b = 35014_{10} = (n_4 n_3 n_2 n_1 n_0)_{10}$$

and each digit $n_i$ corresponds with the following digit:

| | |
|---|---|
| $n_4$ | 3 |
| $n_3$ | 5 |
| $n_2$ | 0 |
| $n_1$ | 1 |
| $n_0$ | 4 |

So, now that we know what each $n_i$ is, we can expand 35014 by using its digits and weight:

$$35014_{10} = (3 \times 10^4) + (5 \times 10^3) + (0 \times 10^2) + (1 \times 10^1) + (4 \times 10^0)$$

Now, expanding decimal numbers this way isn't really useful for our purposes, but what about other number bases? Well, as it turns out, this is how we convert from other number bases to decimal. For fun, lets convert $415_6$ (415 in base 6) to decimal:

$$415_6 = (4 \times 6^2) + (1 \times 6^1) + (5 \times 6^0) = 144 + 6 + 1 = 155_{10}$$

Using this method, we can now convert a binary string to a decimal number. However, you should note that we still can't represent every integer, since (at present) we have no way of indicating if the binary string represents a negative number. As an example, let's convert $110101_2$ to decimal:

$$\begin{aligned} 110101_2 &= (1 \times 2^5) + (1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) \\ &= 32 + 16 + 0 + 4 + 0 + 1 \\ &= 53_{10} \end{aligned}$$

[4] Since we're not necessarily working in base 10, we should denote somehow the base of the number we're working with. I know $(n_i)_b$ isn't the easiest thing to understand, but I wanted to make it clear that $n_i$ is of base $b$.

Now that we know how to convert binart to decimal, what about the other way around? Fortunately, all we need to do is divide any number by 2, and keep track if there is a remainder. Given a number $n$, we start by calculating $n/2$ with a remainder, which we write as $n = 2q + r$. Then, we write down $r$, which will be either one or zero. Then, we repeat with $q$ as $n$, until $q = 0$. After we meet this condition, we have all of the binary digits, and we just have to write down the digits in "reverse" order (the first binary digit is $r$ when $q = 0$).

| Step Number | $n$ | $q$ | $r$ |
|---|---|---|---|
| 1 | 5414 | 2707 | 0 |
| 2 | 2707 | 1353 | 1 |
| 3 | 1353 | 676 | 1 |
| 4 | 676 | 338 | 0 |
| 5 | 38 | 169 | 0 |
| 6 | 169 | 84 | 1 |
| 7 | 84 | 42 | 0 |
| 8 | 42 | 21 | 0 |
| 9 | 21 | 10 | 1 |
| 10 | 10 | 5 | 0 |
| 11 | 5 | 2 | 1 |
| 12 | 2 | 1 | 0 |
| 13 | 1 | 0 | 1 |

Table 1: Converting $5414_{10}$ to binary

Now that we have all of the $r$ values, we write them in "reverse", going up in the table starting from step 13, and we have the binary string $\mathbf{1010100100110}_2$, which is equivalent to $5414_{10}$.

### 5.1 Java Implementation For Converting Decimal To Binary List

Just for fun, the following is an implementation of the above method of converting decimal to binary. It's expected for you to not really understand it, but I figure you might want to see some code, since it'll be a while before we do any programming.

```java
import java.util.List;
import java.util.LinkedList;
import java.lang.StringBuilder;

public class Main {
    static List<Integer> decimalToBinary(int n) {
        LinkedList<Integer> binaryDigits = new LinkedList<Integer>();
        while(n != 0) {
            int remainder = n % 2;
            binaryDigits.addFirst(remainder);
            n = n / 2; // Java throws away remainder in integer division
        }
        return binaryDigits;
    }

    static <T> String ListJoin(List<T> list) {
        StringBuilder builder = new StringBuilder();
        list.forEach((x) -> builder.append(x.toString()));
        return builder.toString();
    }

    public static void main(String[] args) {
        int n = 5414;
        List<Integer> binaryDigits = decimalToBinary(n);
        System.out.println("Received: " + Integer.toString(n));
        System.out.println("Binary List: " + binaryDigits.toString());
        System.out.println("Binary String: " + ListJoin(binaryDigits));
    }
}
```

```
Received: 5414
Binary List: [1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0]
Binary String: 1010100100110
```

Figure 1: Output of above code