UNIVERSITY OF BABYLON

COLLEGE OF SCIENCE FOR WOMEN

COMPUTER DEPARTMENT

2024/ 2025

FIRST CLASS

Computer Organization

LECTURES

PREPARED BY:
*LECTURER: **Dr. Ahmed Mohammed Hussein***

2024-2025

❖ LEARNING OBJECTIVES

After completion of this lecture, you should be able to:

- Describe the data representation
- Describe the number systems

1. DATA REPRESENTATION

The basic building block of personal computers is the transistor. A ***transistor*** is an electronic device for controlling the flow of electrons in an electrical circuit. If electrons are allowed to flow, the circuit is **ON**; conversely, if electrons are not allowed to flow, the circuit is **OFF**. Thinking of a transistorized circuit as a switch like a light switch at a home. The switch is either **on** or **off** and stays that way until it is flipped again. When a circuit is **on**, we say it is in the marking state and assign a **1** to it. Conversely, when it is **off,** we assign a **0** to it.

A modern digital computer is often said to be a binary computer because its most basic circuits can remember either one of ***two states: 0 and 1***. The binary digits 0 and 1 are called **bits**.

2. NUMBERS SYSTEMS

The number systems that we discuss here are based on positional number systems. The decimal number system that we are already familiar with is an example of a positional number system. In contrast, the Roman numeral system is not a positional number system. Every positional number system has a ***base***, and an ***alphabet***. The base is a positive number. For example, the decimal system is a base-10 system. The number of symbols in the alphabet is equal to the base of the number system. The alphabet of the decimal system is 0 through 9, a total of 10 symbols or digits. There are four number systems that are relevant in the context of computer systems and programming. These are the *decimal* (base-10), *binary* (base-2), *octal* (base-8), and *hexadecimal* (base-16) number systems.

System	Base	Possible digits (alphabet)
Binary	**2**	**0 1**
Octal	**8**	**0 1 2 3 4 5 6 7**
Decimal	**10**	**0 1 2 3 4 5 6 7 8 9**
Hexadecimal	**16**	**0 1 2 3 4 5 6 7 8 9 A B C D E F**

Digits	Binary
0	**0000**
1	**0001**
2	**0010**
3	**0011**
4	**0100**
5	**0101**
6	**0110**
7	**0111**
8	**1000**
9	**1001**
A	**1010**
B	**1011**
C	**1100**
D	**1101**
E	**1110**
F	**1111**

Note

Computers internally use the binary system. The remaining two number systems—octal and hexadecimal—are used mainly for convenience to write a binary number even though they are number systems on their own. We would have ended up using these number systems if we had 8 or 16 fingers instead of 10.

In a positional number system, a sequence of digits is used to represent a number. Each digit in this sequence should be a symbol in the alphabet. There is a weight associated with each position. If we count position numbers from right to left starting with zero, the weight of position n in a base b number system is bn. For example, the number 579 in the decimal system is actually interpreted as

$$5 \times (10^2) + 7 \times (10^1) + 9 \times (10^0)\ .$$

(Of course, 10^0=1). In other words, 9 is in unit's place, 7 in 10's place, and 5 in 100's place.

2.1. BINARY SYSTEM

The key to understanding computers is the binary number system. The binary number system has only two digits: **0 to 1**. Just as decimal notation is based on places that represent powers of ten, binary notation is based on power of two. For example, the decimal number **537** is really the sum of powers of ten:

$$537 = (5\times10^2) + (3\times10^1) + (7\times10^0)$$
$$= (5\times100) + (3\times10) + (7\times1)$$
$$= 500 + 30 + 7 = 537$$

Binary numbers are sums of powers of two in the same way that decimal numbers are sums of powers of ten. The following table shows the decimal numbers represented by some of more important powers of two in personal computing:

$2^{-2} = 0.25$	$2^{-1} = 0.5$	$2^0 = 1$	$2^1 = 2$	$2^2 = 4$
$2^3 = 8$	$2^4 = 16$	$2^5 = 32$	$2^6 = 64$	$2^7 = 128$
$2^8 = 256$	$2^{16} = 65.536$	$2^{20} = 1.048.576$	$2^{24} = 16.777.216$	

A *BINARY NUMBER* is a string of 1s and 0s, each indicating the presence or absence of a power of two. For example, consider the binary number **101**. This number is converted to its decimal equivalents as follows:

Binary number	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**
Power of two	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Decimal number	**0**	**0**	**0**	**0**	**0**	**4**	**0**	**1**

$$101 \text{ binary} = (1\times2^2) + (0\times2^1) + (1\times2^0)$$
$$= (1\times4) + (0\times2) + (1\times1)$$
$$= 4 + 0 + 1 = 5 \text{ decimal}$$

Thus, **101** is the binary number representation of the decimal number **5**. In a binary number such **101.1** the fractional number part is a sum of negative powers of **2.** For example, **101.1** binary is converted to its decimal equivalent as follows:

101.1 binary = $(1\times2^2) + (0\times2^1) + (1\times2^0) + (1\times2^{-1})$

$= (1\times4) + (0\times2) + (1\times1) + (1\times(1/2))$

$= 4 + 0 + 1 + 1/2 = 5.5$ decimal

Keep in mind that computers work **<u>exclusively with binary numbers</u>** because they can store only either a 1 or a 0. To do arithmetic and word processing they must convert from binary to decimal and back again.

In the binary system, using *n* bits, we can represent numbers from 0 through (2^n-1) for a total of 2^n different values. We need *m* bits to represent *X* different values, where

$$m = \lceil \log_2 X \rceil .$$

for example, 150 different values can be represented by using

$$\lceil \log_2 150 \rceil = \lceil 7.229 \rceil = 8 \text{ bits}.$$

In fact, using 8 bits, we can represent 2^8=256 different values (i.e., from 0 through 255).

2.2. BASIC TERMS AND NOTATION

The alphabet of computers, more precisely digital computers, consists of 0 and 1. Each is called a ***bit***, which stands for the binary digit. The term ***byte*** is used to represent a group of 8 bits. The term ***word*** is used to refer to a group of bytes that is processed simultaneously. The exact number of bytes that constitute a word depends on the system. For example, in the Pentium, a word refers to four bytes or 32 bits. We use the abbreviation "**b**" for bits, "**B**" for bytes, and "**W**" for words. Sometimes we also use ***doubleword*** and ***quadword***. A ***doubleword*** has twice the number of bits as the word and the ***quadword*** has four times the number of bits in a word.

2-bit → 2^2 = 4	**possible states**	**(00, 01, 10, 11)**
3-bit → 2^3 = 8	**possible states**	**(000, - - -, 111)**
8-bit →2^8 = 256	**possible states**	**(00000000, - - - , 11111111)**

Bits in a word are usually ordered from right to left, as you would write digits in a decimal number. The rightmost bit is called the *least significant bit* (**LSB**), and the leftmost bit is called the *most significant bit* (**MSB**).

We use standard terms such as ***kilo*** (K), ***mega*** (M), ***giga*** (G), and so on to represent large integers. Unfortunately, we use two different versions of each, depending on the number system, decimal or binary. Table 1 summarizes the differences between the two systems. Typically, computer-related attributes use the binary version. For example, when we say 128 megabyte (MB) memory, we mean 128×2^{20} bytes. Usually, communication-related quantities and time units are expressed using the decimal system. For example, when we say that the data transfer rate is 100 megabits/second (Mb/s), we mean 100×10^{6} Mb/s.

Table 1 Terms to represent large integer values

Term	Decimal (base 10)	Binary (base 2)
K (kilo)	10^3	2^{10}
M (mega)	10^6	2^{20}
G (giga)	10^9	2^{30}
T (tera)	10^{12}	2^{40}
P (peta)	10^{15}	2^{50}

3. UNSIGNED INTEGER REPRESENTATION

Now that you are familiar with different number systems, let us turn our attention to how integers (numbers with no fractional part) are represented internally in computers. Of course, we know that the binary number system is used internally. Still, there are a number of other details that need to be sorted out before we have a workable internal number representation scheme.

The most natural way to represent unsigned (i.e., nonnegative) numbers is to use the equivalent binary representation. A binary number with n bits can represent 2^n different values, and the range of the numbers is from 0 to 2^n-1. Padding of 0s on the left can be used to make the binary conversion of a decimal number equal exactly N bits. For example, to represent 16D we need [$\log_2 16$] = 5 bits. Therefore, 16D = 10000B. However, this can be extended to a byte (i.e., $N=8$) as

00010000B

A problem arises if the number of bits required to represent an integer in binary is more than the N bits we have. Clearly, such numbers are outside the range of numbers that can be represented using N bits. Recall that using N bits, we can represent any integer X such that

$$0 \le X \le 2^n - 1.$$

4. SIGNED NUMBER REPRESENTATION

One common way of handling negative numbers is to add one bit to the binary code of the number called sign bit. This is the frequently the most left bit, with a **0** indicating a positive number and a **1** a negative number.

There are three widely used techniques for representing both positive and negative numbers:-

1. **Sign and magnitude.**
2. **1's complement.**
3. **2's complement.**

In sign and magnitude method the first bit from the left used for a sign and the remaining for the magnitude of the number. For example, the numbers between **-127** and **+127** could be represented in **8 bits** , the ***first*** being used for a sign and the ***remaining* 7** for the magnitude.

For example, the negative number (-18) is represented using 6 bits, base 2 in the sign-magnitude format, as follows (110010), while a (+18) is represented as (010010). Although simple, the sign-magnitude representation is complicated when performing arithmetic operations. In particular, the sign bit has to be dealt with separately from the magnitude bits. Consider, for example, the addition of the two numbers +18 (010010) and -19 (110011) using the sign-magnitude representation. Since the two numbers carry different signs, then the result should carry the sign of the larger number in magnitude, in this case the (-19). The remaining 5-bit numbers are subtracted (10011-10010) to produce (00001), that is, (-1).

Note The positive values have the same representation systems, while variation occurs in the representation of negative values.

In the 1's complement representation system, negative values are obtained by complementing each bit of the representation of the corresponding positive value.

According to the 2's complement system, a positive number is represented the same way as in the sign-magnitude. However, a negative number is obtained by adding **1** to the 1' complement of value. Consider, for example, the representation of the number (-19) using 2's complement. In this case, the number 19 is first represented as (010011). Then each digit is complemented, hence the number will be (101100). Finally a "1" is added at the least significant bit position to result in (101101). Now, consider the 2's complement representation of the number (+18). Since the number is positive, then it is represented as (010010), the same as in the sign-magnitude case. Now, consider the addition of these two numbers. In this case, we add the corresponding bits without giving special treatment to the sign bit. The results of adding the two numbers produces (111111). This is the 2's complement representation of a (-1), as expected. The main advantage of the 2's complement representation is that no special treatment is needed for the sign of the numbers. Another characteristic of the 2's complement is the fact that a carry coming out of the most significant bit while performing arithmetic operations is ignored without affecting the correctness of the result. Consider, for example, adding -19 (101101) and +26 (011010). The result will be (1)(000111), which is correct (+7) if the carry bit is ignored.

EX: Represent the number +9 using the three methods

+9 →	**0,1001**	**signed – magnitude**
→	**0,1001**	**1`s complement**
→	**0,1001**	**2`s complement**

EX: what is the 1`s complement, 2`s complement and signed – magnitude of -9.

-9 $\rightarrow$	**1,1001**	**signed – magnitude**
$\rightarrow$	**1,0110**	**1`s complement**
$\rightarrow$	**1,0111**	**2`s complement**

<u>Note:-</u> *Special case in 2's complement representation is be that whenever a signed number has a* ***<u>1</u>*** *in the sign bit and a* ***<u>0</u>*** *for all the magnitude bits, its decimal equivalent is (-2^{N-1}) where* ***N*** *is the total number of bits including sign bit. For example:*

1,0000 $\longrightarrow -2^{5-1} = -2^{4} = -16$

The advantage of the sign 2's complement representation over the 1's complement and sign/magnitude is that **<u>it contains only one type of zero</u>** while the other representations have both a **+0** and **-0**. And the other is that **<u>addition of 2' complement numbers can be performed without regard for the sign</u>**.

STUDENT-ACTIVITY

1. What is the basic building block of personal computers?
2. List various number systems with examples.
3. Define the binary number system.