Number Systems

CSI-140 Introduction to Programming

Instructor: Dr. Vikas Thammanna Gowda Semester: Fall 2025

Overview: *This chapter introduces students to the concept of number systems, which are the foundation for representing, storing, and processing numerical information in both human and computer contexts. Students will explore three primary positional number systems—**Decimal (Base 10)**, **Binary (Base 2)**, and **Hexadecimal (Base 16)**—learning how each system works, their advantages, and their real-world applications. The chapter also cover **conversion techniques** between these systems, the direct relationship between binary and hexadecimal, and the **representation of data using bits and bytes**. By understanding these concepts, students will develop the skills to interpret and manipulate numeric data in various forms, a critical competency for programming, digital electronics, and computer science.*

Learning Objectives *After studying these notes, students should be able to:*

- *Explain what a number system is and why different bases are used.*
- *Describe the role of number systems in everyday life, science, engineering, and computing.*
- *Identify the base, allowed digits, and positional notation rules for decimal, binary, and hexadecimal systems.*
- *Convert numbers between decimal, binary, and hexadecimal using systematic methods.*
- *Apply the direct 4-bit mapping between binary and hexadecimal for quick conversions.*
- *Compare the advantages and limitations of decimal, binary, and hexadecimal systems.*
- *Recognize practical applications of each system, including memory addressing, data representation, and color coding.*
- *Define bits and bytes, and explain their role in digital data storage and processing.*
- *Relate number system concepts to practical computing tasks such as debugging, network addressing, and low-level programming.*

Contents

1 Introduction to Number Systems and Bases

Number systems are essential because they provide a structured and universally understood way to represent, store, and manipulate numerical information. Without them, it would be nearly impossible to communicate quantities, perform calculations, or store data in a consistent and efficient manner. Number systems act as the "language" of mathematics, science, and computing.

1.1 Why We Need Number Systems

- **Representation of Quantities:** Number systems allow us to represent both small and extremely large quantities in a compact and precise way. For example, the distance from Earth to the Sun is about 149,600,000 km in decimal, which is easier to understand than writing the same number as a string of tally marks.
- **Communication of Information:** A standardized number system ensures that numerical information can be shared and understood across different regions, cultures, and disciplines. For instance, the decimal number `98.6` (body temperature in Fahrenheit) means the same worldwide.
- **Mathematical Operations:** Structured number systems make it possible to apply consistent rules for arithmetic, algebra, and higher-level mathematics. For example, $2 + 3 = 5$ works the same way in any decimal context.
- **Data Storage and Processing:** Computers and digital devices rely on binary representation to store and process all kinds of information efficiently. For instance, the letter "A" is stored as 01000001_2 in ASCII.
- **Simplifying Complex Concepts:** Positional notation breaks down large values into smaller components based on powers of the base. For example, $345_{10} = 3 \times 10^2 + 4 \times 10^1 + 5 \times 10^0$.

1.2 Where We Use Number Systems

- **Daily Life (Decimal):** Counting objects, measuring distances, telling time, managing money, and recording statistics all rely on the decimal number system. For example, buying an item for $12.99 uses decimal currency representation.
- **Science and Engineering (Decimal and Scientific Notation):** The speed of light is approximately 3.00×10^8 m/s in scientific notation. Engineers may also use binary and hex for microcontroller programming.
- **Computing and Digital Electronics (Binary and Hexadecimal):** A byte with the binary value 11001100_2 can be written as CC_{16} in hexadecimal for readability.
- **Networking and Communication (Hexadecimal):** A MAC address like `AC:DE:48:00:11:22` is written in hexadecimal to compactly represent 48 bits of data.
- **Programming and Software Development (Hexadecimal in Web Colors):** The color red in web design is `#FF0000`, where `FF` (255 in decimal) represents maximum red intensity, and green and blue are set to 0.
- **Education and Research (Various Bases):** Students may convert 42_{10} to binary (101010_2) and to hexadecimal ($2A_{16}$) to understand base relationships.

1.3 Bases

A number system is a way to represent numbers using a specific base (or radix). The base determines how many unique digits the system uses and the value each digit's position represents. All positional number systems (including binary, decimal, and hexadecimal) work on the same principle: each position in a number has a place value that is a power of the base. For example, in base-10 (decimal) the rightmost digit is in the 10^0 (ones) place, the next is 10^1 (tens), then 10^2 (hundreds), and so on.

Changing the base changes the set of allowed digits and the place values:

- **Base 10:** Each place is a power of 10, and digits range from 0–9.
- **Base 2:** Each place is a power of 2, and digits can only be 0 or 1.
- **Base 16:** Each place is a power of 16, and digits range from 0–9 and A–F (where A–F represent 10–15).

In the following sections, we will explore the decimal (base-10), binary (base-2), and hexadecimal (base-16) systems in detail. We'll explain the digits they use, how positional notation works in each system, and provide step-by-step examples of converting between these bases. We'll also discuss the advantages of each system and where they are used in the real world.

2 Decimal (Base 10) Number System

There are ten possible digits, reflecting the base 10.

Digits Used:
`0, 1, 2, 3, 4, 5, 6, 7, 8, 9.`

2.1 How it Works (Positional Notation)

The decimal system is the standard system for denoting integer and non-integer numbers. It is a positional system with base 10, meaning each digit's place value is a power of 10. For example, in the decimal number 735, the digits represent:

$$\begin{aligned} & 7 \times 10^2 + 3 \times 10^1 + 5 \times 10^0 \\ &= 700 + 30 + 5 \\ &= 735. \end{aligned}$$

Each position (from rightmost to leftmost) is ones, tens, hundreds, thousands, etc., each 10 times the value of the previous position. The decimal system is the most commonly used number system in everyday life, largely because it aligns with human intuition and our tendency to count using ten fingers.

2.2 Converting Decimal to Other Bases

To convert a decimal number to another base (like binary or hex), a common method is *repeated division* by the target base:

1. Divide the decimal number by the new base (e.g. 2 for binary, 16 for hex).
2. Record the remainder – this is the least significant digit (LSD) in the new base.

3. Use the integer quotient for the next division and repeat until the quotient is 0.

4. The new-base number is the sequence of remainders read in reverse order (from last remainder to first).

We will see specific examples of this process for binary and hexadecimal conversions below.

2.3 Advantages of the Decimal System

- **Human-Friendly:** Decimal is easy for people to read, understand, and use in daily life. We naturally learn to count in tens (0–9) and perform arithmetic in base 10, partly because humans have ten fingers. The digit symbols 0–9 are familiar, and calculations like addition and multiplication tables are taught in base 10 from early on.

- **Widely Adopted:** Because it's intuitive, decimal has become a universal standard for most human activities. Almost all countries and cultures use decimal for commerce, education, and science. This universality makes communication of quantities straightforward (for example, everyone understands "100" in decimal as one hundred).

- **Ease of Calculation:** The decimal system's structure (powers of 10) makes it easy to perform mental math and aligns well with measurement systems. Many unit systems, like the metric system, are based on powers of 10, simplifying conversions (e.g. 1 meter = 100 centimeters). It's also convenient for currency (dollars and cents, where 100 cents = 1 dollar), so financial calculations are straightforward in decimal.

2.4 Applications and Uses of Decimal

- **Everyday Counting and Math:** Nearly all everyday counting, measuring, and basic arithmetic is done in decimal. When you count objects, tell time (aside from certain units like 60 minutes/hour, etc.), read a speed limit, or do math homework, you are using the decimal system. It's the default for writing numbers in almost every context (unless you're specifically working in computing or another specialized field).

- **Commerce and Finance:** Prices, money, and accounting use decimal. Most financial and commercial transactions are conducted in the decimal system. For example, if something costs \$19.99, that's a decimal representation. Interest rates, invoices, and budgets are all calculated in base 10, which helps ensure everyone understands the values.

- **Science and Engineering:** Scientific measurements (meters, liters, grams, etc.) commonly use decimal units and notation, especially with the metric system that scales by factors of ten. Engineering tolerances and measurements are often expressed in decimal. While computers work in binary internally, engineers and scientists generally interpret and present numerical results in decimal for clarity.

Overall, decimal's strength is its simplicity and intuitive use for humans, which is why it remains our everyday default number system.

3 Binary (Base 2) Number System

Only two digits are used in binary, reflecting its base of 2.

Digits Used:
`0, 1.`

3.1 How it Works (Positional Notation)

In the binary system, each digit (called a *bit*) represents a power of 2, with the rightmost bit being 2^0 (the ones place), the next being 2^1 (the twos place), then 2^2 (fours), 2^3 (eights), and so on. Because only two digits are available, place values grow by doubling at each step.

For example, the binary number 10110_2 can be broken down by place values:

- Positions (from rightmost to leftmost) are 2^0, 2^1, 2^2, 2^3, 2^4, etc.
- 10110_2 has bits in the 2^4, 2^3, 2^2, 2^1, 2^0 places respectively (from left to right). We can label these: 1×2^4, 0×2^3, 1×2^2, 1×2^1, 0×2^0.
- Calculating each contribution: $1 \times 16 + 0 \times 8 + 1 \times 4 + 1 \times 2 + 0 \times 1$.
- Summing these values: $16 + 0 + 4 + 2 + 0 = 22$ (in decimal).

So, 10110_2 represents the decimal number 22. Each binary digit tells us whether a certain power of 2 is included (1) or not (0) in the total.

3.2 Binary to Decimal Conversion

Let's convert a binary number to its decimal equivalent step-by-step:
Example: Convert binary 10110_2 to decimal:

1. Write out place values for each bit position (from the rightmost bit to the leftmost bit). For 10110_2:
   - The rightmost bit 0 is in the 2^0 (ones) place.
   - Next bit 1 is in the 2^1 (twos) place.
   - Next bit 1 is in the 2^2 (fours) place.
   - Next bit 0 is in the 2^3 (eights) place.
   - Leftmost bit 1 is in the 2^4 (sixteens) place.
2. Multiply each bit by its place value:
   - $1 \times 2^4 = 16$
   - $0 \times 2^3 = 0$
   - $1 \times 2^2 = 4$
   - $1 \times 2^1 = 2$
   - $0 \times 2^0 = 0$
3. Add the results: $16 + 0 + 4 + 2 + 0 = 22$.

Result: $10110_2 = 22_{10}$ (binary 10110 equals 22 in decimal).

This method—multiplying bits by powers of 2 and summing—works for any binary-to-decimal conversion.

3.3 Decimal to Binary Conversion

Now let's convert a decimal number to binary using the repeated division method described earlier.
Example: Convert decimal 19_{10} to binary:

1. $19 \div 2 = 9$ remainder 1. (Divide by 2 and record the remainder.)
2. $9 \div 2 = 4$ remainder 1.
3. $4 \div 2 = 2$ remainder 0.
4. $2 \div 2 = 1$ remainder 0.
5. $1 \div 2 = 0$ remainder 1. (Stop here, as the quotient is now 0.)
6. Now read the remainders in reverse order (from last to first): 10011_2.

So, $19_{10} = 10011_2$. (We can check: $1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 16 + 0 + 0 + 2 + 1 = 19$, confirming the result.)

3.4 Advantages of the Binary System

- **Simplicity in Electronics:** Binary is ideal for digital systems because it only uses two states. Physical devices like transistors, switches, and logic gates can be simple and robust when they only need to distinguish between off (0) and on (1) states. An electrical component is either conducting current or not, a circuit is open or closed – this binary nature makes hardware design straightforward and reliable.
- **Noise Tolerance and Reliability:** With only two possible values, binary signals are unambiguous. A signal is either high or low, with a clear gap between the two. This gives binary data a resistance to noise – slight fluctuations or interference in a binary signal can be corrected or ignored as long as the signal is closer to one state than the other. As a result, copies of binary data can be made without loss of fidelity (a copied 0 is still 0, and 1 stays 1), and errors can be detected more easily. This reliability is crucial for computers, which move millions of bits per second without mistakes.
- **Universal Computation Basis:** Binary underpins all modern computing and digital electronics. Because of the two-state (0/1) nature of binary, it maps perfectly onto the on/off states of transistors in computer chips. Every piece of data in a computer—numbers, text characters, images, videos—is ultimately represented in binary code. This makes binary the foundational language of computers. The simplicity of binary allows computers to use logical sequences of 0s and 1s to perform complex tasks. In short, binary's big advantage is that it is machine-friendly—it's the only thing computers "speak." (Fun fact: the term *bit* is short for "binary digit" – a single 0 or 1 in a binary sequence.)

3.5 Applications and Uses of Binary

- **Computing and Digital Systems:** Virtually all electronic computing devices use binary. The programs we run are compiled into binary machine code, and data is stored in memory chips as binary patterns of bits (0s and 1s). Everything from your smartphone's processor to the memory in a USB flash drive relies on binary states (charged or not charged, current flowing or not) to represent information. Binary numbers are the foundation of computer

operations, data storage, and processing. (For example, the ASCII code for a letter is a binary number, and color values in an image file are binary numbers.)

- **Digital Electronics:** Beyond just general-purpose computers, binary is used in all sorts of digital electronics—calculators, digital clocks, and any system with digital circuits. Logic circuits in these devices use binary signals (voltage high = 1, low = 0) to perform operations.
- **Networking and Communications:** Low-level network protocols and data transmission break information into binary. Even if at higher levels we might use hex or decimal notation, at the wire level it's all binary pulses. For instance, a byte (8 bits) is the basic unit of data in most modern networks, and it's essentially an 8-digit binary number.
- **Boolean Logic:** Binary's 0 and 1 can also represent false and true in Boolean logic. This is heavily used in programming and circuit design—conditions and decisions in code (if/else statements) ultimately boil down to true/false values (1 or 0) and are processed using binary logic.

In summary, binary's two-state nature makes it uniquely suited for the digital world. It provides a stable, clear, and efficient way to represent and manipulate data with hardware. This is why everything from simple calculators to complex supercomputers internally uses binary.

4 Hexadecimal (Base 16) Number System

4.1 Digits Used

There are 16 symbols in total. The digits 0–9 have their usual values (zero through nine), and the letters A–F represent the decimal values 10 through 15, respectively.

Digits Used:
`0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.`

4.2 How it Works (Positional Notation)

Hexadecimal is a base-16 system, so each digit position represents a power of 16. For example, in a hexadecimal number the rightmost position is 16^0 (ones), the next is 16^1 (sixteens), then 16^2 (256s), 16^3 (4096s), and so on. Just like in decimal or binary, you determine a hex number's value by multiplying each digit by its place value and summing the results. The main difference is that for digits A–F, you use 10–15 in the calculation.

To get a feel for hexadecimal counting: after 8, 9 in hex comes A (decimal 10), then B (11), C (12), D (13), E (14), F (15). When you increment 1 past F (15), you reach 10 in hex, which means $1 \times 16^1 + 0 \times 16^0 = 16$ in decimal. In the same way that 99 in decimal goes to 100 (adding a new place), or 11 in binary goes to 100, in hex F goes to 10. Thus, `0x10` (we often prefix hex numbers with "0x") equals 16_{10}, `0x11` equals 17, and so forth.

4.3 Hexadecimal to Decimal Conversion

Let's convert a hexadecimal number to its decimal equivalent step-by-step.
Example: Convert hexadecimal $1A3_{16}$ to decimal:

1. Identify the digits and their positions. $1A3_{16}$ has three hex digits: 1, A, and 3. From rightmost to leftmost, they are in the 16^0 (ones), 16^1 (sixteens), and 16^2 (256s) places, respectively.
2. Convert each hex digit to its decimal value. In hex, $A = 10$ in decimal (the digit "1" is just 1, and "3" is 3 in decimal).
3. Multiply each digit by its place value (power of 16):
   - $1 \times 16^2 = 1 \times 256 = 256$
   - $A\,(10) \times 16^1 = 10 \times 16 = 160$
   - $3 \times 16^0 = 3 \times 1 = 3$
4. Add the results: $256 + 160 + 3 = 419$.

Result: $1A3_{16} = 419_{10}$. In other words, hex 1A3 represents the decimal number 419. (In general, for any hex-to-decimal conversion, multiply each hex digit by 16 raised to the power of its position index, and sum them up.)

4.4 Decimal to Hexadecimal Conversion

Now let's convert a decimal number to hexadecimal using repeated division by 16.
Example: Convert decimal 254_{10} to hexadecimal:

1. $254 \div 16 = 15$ remainder 14. The remainder 14 corresponds to the hex digit `E` (since $E_{16} = 14_{10}$). The quotient is 15.
2. $15 \div 16 = 0$ remainder 15. The remainder 15 corresponds to hex `F` ($F_{16} = 15_{10}$). The new quotient is 0, so we stop.
3. Now read the remainders in reverse order (from last division to first): F E.

The hexadecimal representation is $\texttt{FE}_{16}$. We found that $254_{10} = \texttt{FE}_{16}$. As a check, convert $\texttt{FE}_{16}$ back to decimal: $F = 15$ and $E = 14$, so $F \times 16^1 + E \times 16^0 = 15 \times 16 + 14 \times 1 = 240 + 14 = 254$ – it matches. This divide-and-remainder method will work for any decimal-to-hex conversion, just as dividing by 2 works for decimal-to-binary.

4.5 Relation to Binary

One important aspect of hexadecimal is how it relates to binary. Sixteen is 2^4, meaning one hex digit is equivalent to four binary digits (also called a *nibble*). This makes hex extremely useful for representing binary in a shorter form. In fact, converting between binary and hex is very direct: to convert a binary number to hex, you can split the binary into groups of 4 bits (starting from the right) and convert each 4-bit group to the corresponding hex digit. Conversely, each hex digit expands to 4 binary bits.

4.5.1 Hex to Binary Conversion

To convert from hexadecimal to binary:

1. Write each hex digit as its 4-bit binary equivalent using the reference table below.
2. Combine the binary groups to form the complete binary number.

Example: Convert $\texttt{2F}_{16}$ to binary.

- $2_{16} = 0010_2$
- $F_{16} = 1111_2$
- Combine: $\texttt{2F}_{16} = 0010\,1111_2$

4.5.2 Binary to Hex Conversion

To convert from binary to hexadecimal:

1. Group the binary digits into sets of 4 bits, starting from the right (add leading zeros if necessary).
2. Replace each group with its corresponding hex digit from the table.

Example: Convert 11010110_2 to hexadecimal.

- Group into nibbles: 1101 0110
- $1101_2 = D_{16}$, $0110_2 = 6_{16}$
- Combine: $11010110_2 = \texttt{D6}_{16}$

4.5.3 4-bit Binary to Hexadecimal Table

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

4.5.4 Why This is Useful

This direct 4-bit mapping means:

- Large binary numbers can be written in a shorter hexadecimal form without losing precision.
- Switching between binary (machine-friendly) and hex (human-friendly) is quick and reduces transcription errors.

4.6 Advantages of the Hexadecimal System

- **Compact and Concise:** Hexadecimal can represent large binary numbers in a much shorter form. Because one hex digit covers 4 binary bits, it uses far fewer digits to represent the same value compared to binary. For example, an 8-bit value in binary might be 11010110_2, but in hex that's just $\texttt{D6}_{16}$ (2 digits). This conciseness means less space is needed to write numbers, and it's generally easier for humans to read or remember a hex string than a long binary string. In many cases, hex is also more concise than decimal for very large values (since base 16 grows quickly).

- **Easy to Convert To/From Binary:** Since $16 = 2^4$, converting between hex and binary is fast and simple. You don't need complex calculations—just a direct mapping of each nibble to a hex digit. This reduces the chance of error when transcribing or interpreting binary data. For example, it's easier to convert binary 110010111100_2 to hex by grouping bits (pad to 0011 0010 1111 00 and group: $0011 = 3$, $0010 = 2$, $1111 = F$, 00 (with padding) $= 0$) to get `0x32F0`, rather than converting that binary to a huge decimal or trying to keep track of the entire binary string. Grouping binary into hex makes the data more human-readable and less error-prone.

- **Human-Friendliness in Tech Contexts:** While binary is what the machine "sees," hex is a bit more user-friendly for people working with computers. It's easier to say or note a hex value (like `0xB4` or "B4 hex") than a long binary string (10110100_2) or an unwieldy decimal (180) when dealing with low-level data. In debugging, for instance, hex representations of memory addresses or error codes are shorter and more quickly understood by programmers, reducing cognitive load. Writing numbers with fewer digits (and avoiding long sequences of 0s and 1s) lowers the chance of mistakes.

4.7 Applications and Uses of Hexadecimal

- **Computer Memory and Addresses:** Hexadecimal is heavily used in programming and computer engineering as a convenient way to represent binary-coded values. For example, memory addresses in many systems are displayed in hex (you might see an address like `0x3F9A` instead of its decimal or binary form). Each byte of memory (8 bits) can be represented by two hex digits, which is far easier to read at a glance than 8 binary digits. Thus, hex is commonly used in low-level programming (such as assembly language) and debugging to show machine addresses or raw data.

- **Color Codes in Web Design:** If you've ever worked with HTML or CSS, you have seen color codes like `#FF5733`. These are hexadecimal codes. In web design and graphics, colors are often represented as hex values—typically three bytes (six hex digits, with the format RRGGBB for the red, green, and blue components). For example, white is `#FFFFFF` (which in hex means R = FF = 255_{10}, G = FF, B = FF) and black is `#000000`. Hex makes these binary RGB values shorter and cleaner to work with. Designers and developers use hex codes because they map nicely to the binary values the computer uses for color intensities.

- **MAC Addresses and Identifiers:** Many technical identifiers are expressed in hex. A good example is a MAC address (Media Access Control address) for network devices. It's a 6-byte (48-bit) identifier usually written as 12 hex digits (often separated by colons or dashes, e.g. `AC:DE:48:00:11:22`). Writing that in decimal would be cumbersome, and in binary it would

be too long, so hex is a perfect compact representation. Similarly, other identifiers like GUIDs, serial numbers, or cryptographic keys are often displayed in hexadecimal for brevity.

- **Error Codes and Debugging:** When something goes wrong deep in a computer system, error messages (like stop codes in Windows or exception codes) are sometimes given in hex. For instance, you might see an error code `0xC0000005`. These codes correspond to specific binary patterns that the system uses internally. In hex, they're easier for a programmer to interpret or look up. Each portion of such a code might have particular meaning when translated from hex to binary bitfields. Using hex in these cases helps pinpoint issues at a binary level without writing out long binary strings.
- **Assembly and Machine Code:** Programmers who write in assembly language or examine raw machine code often use hexadecimal. The machine code bytes of a program are much easier to read in hex. For example, an instruction might be `3E A0` in hex, which is far more digestible than `00111110 10100000` in binary or some large decimal number. Tools like debuggers and disassemblers will display the machine instructions in hex alongside the assembly instructions. This is why one of the early skills in low-level programming is learning to quickly convert between hex and binary.

In summary, hexadecimal serves as a bridge between a human-friendly representation and binary. It's prevalent anywhere people need to closely examine or communicate binary data. By using base-16, we get a shorthand for binary that retains a clear mapping to it. Hexadecimal's advantage is evident in tech fields: it simplifies working with binary quantities by grouping bits into nibbles (4-bit groups), making them more readable and reducing mistakes. Yet, unlike decimal, hex is not typically used for everyday quantities – it shines in the niches where binary is under the hood.

5 Bit and Byte Representation

In digital systems, all information is ultimately stored and processed as sequences of binary digits (bits). Understanding how bits and bytes work is essential when working with number systems, since they form the building blocks of binary, hexadecimal, and other base representations.

5.1 What is a Bit?

A **bit** (short for *binary digit*) is the smallest unit of data in a computer and can have one of two possible values:

- 0 (off, low voltage, false)
- 1 (on, high voltage, true)

In binary notation, each bit corresponds to a power of 2. For example, in the 4-bit binary number 1011_2:

$$1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 8 + 0 + 2 + 1 = 11_{10}$$

5.2 What is a Byte?

A **byte** is a group of 8 bits. Bytes are the standard unit for measuring data storage and memory. One byte can represent $2^8 = 256$ distinct values, ranging from 0_{10} to 255_{10}.

5.3 Relationship Between Bits, Bytes, and Number Systems

- In **binary**, a byte is written as 8 bits, e.g., `11001010`.
- In **decimal**, the same byte might be represented as 202_{10}.
- In **hexadecimal**, a byte is represented with exactly 2 hex digits, e.g., $\texttt{CA}_{16}$, since each hex digit corresponds to 4 bits.

5.4 Examples

Consider the binary sequence `01101011`:

- **Binary:** 01101011_2
- **Decimal:** 107_{10} (calculated as $0 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$)
- **Hexadecimal:** $6B_{16}$ (split into $0110_2 = 6$, $1011_2 = \text{B}$)

5.5 Data Size Units

Data storage sizes are measured using multiples of bytes:

- 1 byte = 8 bits
- 1 kilobyte (KB) = 1024 bytes
- 1 megabyte (MB) = 1024^2 bytes
- 1 gigabyte (GB) = 1024^3 bytes

5.6 Why It Matters

Bits and bytes connect the abstract idea of number systems to the physical reality of how computers store and process information. Knowing how to convert between binary, decimal, and hexadecimal helps in:

- Reading and writing memory addresses
- Understanding file sizes and storage capacity
- Debugging and analyzing raw binary/hex data