Number Systems for Information Systems

Understanding How Computers Store and Process Data

MIS 2201: Management Information Systems

September 2, 2025

Contents

1 Course Information

As future information systems professionals, understanding how computers store and process data is fundamental to your success in the field. This material covers essential concepts about number systems, data storage, and digital representation that underpin all modern information systems. Whether you're working with databases, networks, or business applications, these foundational concepts will help you make informed decisions about technology solutions. The only prerequisite for this material is a solid foundation in basic arithmetic operations.

2 Introduction: Why Do Computers Use Different Number Systems?

2.1 Computer Components Overview

To understand why computers use different number systems, we must first examine the fundamental components that make up a computer system. Every computer consists of four essential components that work together to process information.

Input devices such as keyboards, mice, cameras, and microphones allow us to provide data and instructions to the computer. **Output devices** including monitors, speakers, and printers enable the computer to communicate results back to us. The **processor (CPU)** serves as the "brain" of the computer, performing all calculations and logical operations. Finally, **memory** systems provide both temporary storage through RAM and permanent storage through hard drives, allowing the computer to store and retrieve information as needed.

2.2 The Electronic Reality

Key Information

The fundamental reason computers use binary numbers lies in their electronic design. At the most basic level, computer circuits can only exist in two distinct states: an **ON state** with +5 volts (represented as 1) and an **OFF state** with 0 volts (represented as 0). This binary nature of electronic circuits makes it natural for computers to work with binary numbers.

This electronic limitation explains why computers use **Binary Numbers** consisting only of 0s and 1s internally, even though humans naturally prefer working with **Decimal Numbers** that use digits 0 through 9. Understanding this fundamental difference is crucial for information systems professionals, as it forms the foundation for how all digital information is stored and processed in the systems you'll design, manage, and optimize throughout your career.

3 Understanding Decimal Numbers (Base 10)

3.1 Why is 12 larger than 3?

The answer lies in **positional notation** - the position of each digit determines its value.

Example

Decimal Number Breakdown

$$324 = 3 \times 100 + 2 \times 10 + 4 \times 1 \quad (1)$$

$$324 = 3 \times 10^2 + 2 \times 10^1 + 4 \times 10^0 \quad (2)$$

3.2 Key Insight: Powers of 10

The decimal system uses powers of 10 because there are **exactly 10 possible symbols** available for each digit position. These symbols are the familiar digits `0, 1, 2, 3, 4, 5, 6, 7, 8, 9` that we use in everyday mathematics. This relationship between the number of available symbols and the base of the number system is fundamental to understanding how different number systems work.

4 Binary Numbers (Base 2)

4.1 The Binary System

Since computers can only represent two distinct electronic states (ON/OFF), the binary number system naturally uses **powers of 2** as its foundation. In binary, we have only two possible symbols for each digit position: `0` and `1`. Each digit position in a binary number represents a specific power of 2, starting from 2^0 on the rightmost position and increasing as we move left. This elegant system allows computers to represent any number using just these two electronic states.

Example

Binary to Decimal Conversion Examples

$$101_2 = 1 \times 4 + 0 \times 2 + 1 \times 1 = 5_{10} \quad (3)$$

$$111_2 = 1 \times 4 + 1 \times 2 + 1 \times 1 = 7_{10} \quad (4)$$

$$10001_2 = 1 \times 16 + 0 \times 8 + 0 \times 4 + 0 \times 2 + 1 \times 1 = 17_{10} \quad (5)$$

Important Note

Important Note: Leading Zeros
Adding zeros to the left doesn't change the value: $111_2 = 0111_2 = 00111_2 = 7_{10}$

5 Other Number Systems

5.1 Octal (Base 8)

The octal number system uses 8 possible symbols: `0, 1, 2, 3, 4, 5, 6, 7`, with each position representing a power of 8. This system was historically popular in early computing because it provides a convenient way to group binary digits, since $8 = 2^3$, meaning each octal digit corresponds exactly to three binary digits.

5.2 Hexadecimal (Base 16)

Hexadecimal is perhaps the most important alternative number system for information systems professionals, using 16 possible symbols: `0-9` and `A-F`. Since we only have 10 numerical digits, letters are used to represent the additional values needed. The letter A represents the decimal value 10, B represents 11, C represents 12, D represents 13, E represents 14, and F represents 15. Hexadecimal is extremely useful in information systems because $16 = 2^4$, meaning each hexadecimal digit corresponds exactly to four binary digits, making it an efficient way to represent binary data in a more human-readable form. You'll encounter hexadecimal in web development (color codes), database administration, and network configuration.

6 Converting Between Number Systems

6.1 Method 1: Decimal to Binary (Addition Method)

Example

To convert 19_{10} to binary:

1. Find powers of 2 that sum to 19: $16 + 2 + 1 = 19$
2. Place 1s in those positions, 0s elsewhere:

Binary:	0	0	0	1	0	0	1	1
Position:	128	64	32	16	8	4	2	1

Result: $19_{10} = 10011_2$

6.2 Method 2: Decimal to Binary (Division Method)

Example

Continuously divide by 2, tracking remainders:

$$19 \div 2 = 9 \text{ remainder } 1 \quad (6)$$
$$9 \div 2 = 4 \text{ remainder } 1 \quad (7)$$
$$4 \div 2 = 2 \text{ remainder } 0 \quad (8)$$
$$2 \div 2 = 1 \text{ remainder } 0 \quad (9)$$
$$1 \div 2 = 0 \text{ remainder } 1 \quad (10)$$

Reading remainders from bottom to top: $19_{10} = 10011_2$

7 Important Terminology and Storage Units

7.1 Bit and Byte

Understanding computer storage begins with two fundamental units. A **bit** is the smallest unit of data in computing, representing a single binary digit that can be either 0 or 1. The term "bit" comes from "binary digit" and represents the basic building block of all digital information. A **byte** consists of a group of 8 bits and serves as the standard unit for measuring computer memory and storage. This 8-bit grouping was chosen because it provides enough combinations (256 different values) to represent all characters in the English alphabet, numbers, and common symbols.

7.2 Storage Unit Hierarchy

Computer storage follows a hierarchical structure that builds upon the basic bit and byte units. Understanding this hierarchy is essential for working with digital systems and appreciating the scale of modern computing capabilities.

Unit	Size
1 Bit	Single binary digit (0 or 1)
1 Byte	8 bits
1 Kilobyte (KB)	1,024 bytes
1 Megabyte (MB)	1,024 KB = 1,048,576 bytes
1 Gigabyte (GB)	1,024 MB = 1,073,741,824 bytes
1 Terabyte (TB)	1,024 GB = 1,099,511,627,776 bytes

Important Note

Why 1,024 instead of 1,000?
Computers use **binary** (base 2), so storage is measured in powers of 2:

- 1 KB = 2^{10} = 1,024 bytes (not 1,000)
- This is why a "500 GB" hard drive shows as ∼465 GB in your computer

7.3 Storage Examples in Context

- **Text character**: 1 byte (8 bits)
- **Small image**: 100 KB - 1 MB
- **Song (MP3)**: 3-5 MB
- **HD Movie**: 1-4 GB
- **Modern video game**: 50-100 GB
- **Your entire computer**: 256 GB - 2 TB

7.4 Real-World Application: Internet Speeds

Internet companies advertise speeds in **bits** per second, not **bytes**:

$$100 \text{ Megabits per second} = 100 \div 8 = 12.5 \text{ Megabytes per second}$$

8 How Computers Store Images and Colors

8.1 Digital Images: Pixels and Binary

Every digital image is made up of tiny squares called **pixels** (picture elements). Each pixel stores color information using binary numbers.

8.1.1 Pixel Storage Basics

- **Black & White images**: 1 bit per pixel (0 = black, 1 = white)
- **Grayscale images**: 8 bits per pixel (0-255 shades of gray)
- **Color images**: 24 bits per pixel (8 bits each for Red, Green, Blue)

8.2 RGB Color System

Computers represent colors using the **RGB model** (Red, Green, Blue):

Key Information

Each color channel uses 8 bits = 256 possible values (0-255)

$$\text{Red: } 00000000 \text{ to } 11111111 \text{ (0 to 255 in decimal)} \tag{11}$$

$$\text{Green: } 00000000 \text{ to } 11111111 \text{ (0 to 255 in decimal)} \tag{12}$$

$$\text{Blue: } 00000000 \text{ to } 11111111 \text{ (0 to 255 in decimal)} \tag{13}$$

Example

Common Color Examples

Color	RGB Values	Binary Representation
Pure Red	RGB(255, 0, 0)	11111111 00000000 00000000
Pure Green	RGB(0, 255, 0)	00000000 11111111 00000000
Pure Blue	RGB(0, 0, 255)	00000000 00000000 11111111
White	RGB(255, 255, 255)	11111111 11111111 11111111
Black	RGB(0, 0, 0)	00000000 00000000 00000000
Yellow	RGB(255, 255, 0)	11111111 11111111 00000000

8.3 Image Transparency (Alpha Channel)

Modern images often include **transparency** information using an **Alpha channel**:

8.3.1 RGBA Color System

- **R**: Red (8 bits)
- **G**: Green (8 bits)
- **B**: Blue (8 bits)

- **A**: Alpha/Transparency (8 bits)

Alpha Value	Transparency Level
0	Completely transparent (invisible)
127	50% transparent (semi-transparent)
255	Completely opaque (solid)

Example

Example: Semi-transparent Red

$$\text{RGBA}(255, 0, 0, 127) = \text{Red color at 50\% transparency} \tag{14}$$

$$\text{Binary:} \quad 11111111\,00000000\,00000000\,01111111 \tag{15}$$

8.4 Image File Size Calculations

Understanding why image files can be large:

Example

Example: 1920×1080 HD Image

$$\text{Resolution:} \quad 1920 \times 1080 = 2,073,600 \text{ pixels} \tag{16}$$

$$\text{RGB (24 bits per pixel):} \quad 2,073,600 \times 24 = 49,766,400 \text{ bits} \tag{17}$$

$$\text{Convert to bytes:} \quad 49,766,400 \div 8 = 6,220,800 \text{ bytes} \tag{18}$$

$$\text{Convert to MB:} \quad 6,220,800 \div 1,048,576 \approx 5.9 \text{ MB (uncompressed)} \tag{19}$$

With Transparency (RGBA):

$$\text{RGBA (32 bits per pixel):} \quad 2,073,600 \times 32 = 66,355,200 \text{ bits} \tag{20}$$

$$\text{Convert to bytes:} \quad 66,355,200 \div 8 = 8,294,400 \text{ bytes} \approx 7.9 \text{ MB} \tag{21}$$

8.5 Image Compression

Why aren't all images huge files?

8.5.1 File Formats and Compression

- **BMP**: Uncompressed (large files, exact pixel data)
- **JPEG**: Compressed (smaller files, some quality loss)
- **PNG**: Compressed with transparency support (good quality, supports alpha)
- **GIF**: Limited colors (256 colors max), supports transparency

Example

Compression Example

- Uncompressed HD image: ∼6 MB
- JPEG compressed (high quality): ∼500 KB - 2 MB
- PNG compressed: ∼1-3 MB

8.6 Color Depth Variations

Different applications use different color depths:

Bit Depth	Application
1-bit	Black & white only (newspapers, simple graphics)
8-bit	256 colors or grayscale (older games, GIFs)
16-bit	65,536 colors (older displays, some mobile devices)
24-bit	16.7 million colors (standard RGB, most displays)
32-bit	16.7 million colors + transparency (RGBA, professional graphics)

8.7 Real-World Applications in Information Systems

- **Web Development**: JPEG for photos, PNG for graphics with transparency in business websites
- **Enterprise Systems**: Image compression affects database storage costs and system performance
- **E-commerce**: Product image optimization impacts website loading speed and customer experience
- **Document Management**: Understanding file formats helps choose appropriate storage solutions
- **Digital Marketing**: Color depth and compression affect brand consistency across platforms
- **Business Intelligence**: Data visualization requires understanding of image formats for reports and dashboards

9 Character Encoding: ASCII and Unicode

9.1 How Computers Store Text

Just like images and numbers, computers must store text characters using binary numbers. Two main systems handle this: **ASCII** and **Unicode**.

9.2 ASCII (American Standard Code for Information Interchange)

9.2.1 ASCII Basics

- **Created**: 1963 for early computers and telegraphs
- **Size**: 7 bits per character (128 possible characters)
- **Extended ASCII**: 8 bits per character (256 possible characters)
- **Coverage**: English letters, numbers, punctuation, and control characters

9.2.2 ASCII Table (Key Characters)

Character	Decimal	Binary
'A'	65	1000001
'B'	66	1000010
'C'	67	1000011
...	...	...
'Z'	90	1011010
'a'	97	1100001
'b'	98	1100010
'c'	99	1100011
...	...	...
'z'	122	1111010
'0'	48	0110000
'1'	49	0110001
'2'	50	0110010
...	...	...
'9'	57	0111001
' '	32	0100000
'!'	33	0100001
'@'	64	1000000

9.2.3 ASCII Categories

- **Control Characters (0-31)**: Non-printable (like Enter, Tab, Backspace)
- **Digits (48-57)**: '0' through '9'
- **Uppercase Letters (65-90)**: 'A' through 'Z'
- **Lowercase Letters (97-122)**: 'a' through 'z'
- **Special Characters**: Punctuation, symbols, space

Example

ASCII Example: Storing "Hello"

$$\text{'H'} = 72 = 01001000_2 \tag{22}$$
$$\text{'e'} = 101 = 01100101_2 \tag{23}$$
$$\text{'l'} = 108 = 01101100_2 \tag{24}$$
$$\text{'l'} = 108 = 01101100_2 \tag{25}$$
$$\text{'o'} = 111 = 01101111_2 \tag{26}$$

"Hello" = 01001000 01100101 01101100 01101100 01101111
Total: 5 characters × 8 bits = 40 bits = 5 bytes

9.3 ASCII Limitations

- **Language barrier**: Only supports English characters
- **No accents**: Can't represent é, ñ, ü, etc.
- **No symbols**: Missing characters for other languages (Arabic, Chinese, Japanese, etc.)
- **Limited size**: Only 256 characters maximum

9.4 Unicode: The Global Solution

9.4.1 Unicode Overview

- **Purpose**: Represent every character from every language in the world
- **Coverage**: Over 1.1 million possible characters
- **Current use**: ∼150,000 characters defined (letters, symbols, emojis)
- **Backward compatible**: First 128 characters match ASCII exactly

Example

Emoji (UTF-8)

Grinning face emoji: (27)
Unicode: U+1F600 (28)
UTF-8: 11110000 10011111 10011000 10000000 (4 bytes) (29)

9.5 Character Encoding in File Sizes

Example

Text File Size Calculations

- ASCII text file (1,000 characters): 1,000 characters × 1 byte = 1,000 bytes ≈ 1 KB
- UTF-8 English text (1,000 characters): 1,000 characters × 1 byte = 1,000 bytes ≈ 1 KB
- UTF-8 Mixed international text (1,000 characters): ∼800 English × 1 byte + ∼200 accented × 2 bytes = 1,200 bytes ≈ 1.2 KB
- UTF-8 with emojis (1,000 characters): ∼900 regular × 1-2 bytes + ∼100 emojis × 4 bytes = ∼1,300 bytes ≈ 1.3 KB

9.6 Real-World Applications in Business Systems

9.6.1 Information Systems Management

- **Database Design**: Choose appropriate character encoding for international business data
- **Web Applications**: Ensure proper encoding for global customer bases
- **Data Integration**: Handle character encoding when merging systems from different regions
- **Business Intelligence**: Proper encoding ensures accurate reporting across international operations

9.6.2 Enterprise Technology Decisions

- **System Migration**: Understanding encoding prevents data corruption during system upgrades
- **Vendor Selection**: Evaluate software based on international character support
- **Cloud Storage**: Calculate storage costs considering character encoding overhead
- **User Experience**: Ensure applications display correctly for diverse user populations

9.7 Key Differences Summary

Aspect	ASCII	Unicode (UTF-8)
Character limit	128 (256)	1.1+ million
Bytes per char	1	1-4 (variable)
Languages	English only	All world languages
Emojis	No	Yes (supported)
File size	Smaller	Slightly larger
Compatibility	Limited	Universal

10 Practice Problems

10.1 Conversion Exercises

1. Convert 11 to binary
2. Convert 1101_2 to decimal

10.2 Challenge Storage and Image Problems

3. How many bytes are needed to store the RGB values for a single pixel?
4. What is the file size (in MB) of an uncompressed 800×600 RGB image?
5. How many different colors can be represented with 16-bit color depth?
6. If an image uses RGBA format, how many bits are needed per pixel?

10.3 Character Encoding Problems

7. What is the ASCII decimal value for the character 'M'?
8. Convert the ASCII character 'A' to binary (show all 8 bits).
9. How many bytes would the word "Computer" take in ASCII encoding?
10. How many bytes would a single emoji (grinning face) require in UTF-8 encoding?

10.4 Challenge Riddles

1. **When will 100 be less than 9?**
   - Answer: When 100 is in binary ($100_2 = 4_{10}$) and 9 is in decimal
2. **My name is Abd0, what is the value of Abd0 + 1?**
   - Answer: Abd1 (treating it as hexadecimal: $\text{Abd0}_{16} + 1_{16} = \text{Abd1}_{16}$)
3. **When will 1 + 1 = 10?**
   - Answer: In binary system ($1_2 + 1_2 = 10_2$)
4. **When will 1 + 1 = 11?**
   - Answer: When treating numbers as strings/text concatenation

11 Key Takeaways

1. **Position matters** in all number systems - each position represents a power of the base
2. **Binary is fundamental** to computer science because computers operate on electrical states
3. **Conversion skills** are essential for understanding how computers process information
4. **Different bases** serve different purposes in computer science and mathematics

5. **Storage units** follow binary progression (powers of 2) rather than decimal
6. **Digital images** are stored as collections of pixels, each containing color information in binary
7. **Color representation** uses binary numbers to define RGB values and transparency
8. **Text encoding** converts characters to binary using systems like ASCII and Unicode
9. **Unicode** enables global communication by supporting all world languages and symbols

12 Study Tips

12.1 Memory Aids

- Remember powers of 2: 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024...
- Practice converting small numbers first, then work up to larger ones
- For RGB colors: Remember $255 = 11111111_2$ (8 bits all set to 1)
- Storage calculations: Always convert to bytes first, then to larger units
- ASCII: 'A' = 65, 'a' = 97 (32 difference between upper/lowercase)
- UTF-8: English characters = 1 byte, accented characters = 2 bytes, emojis = 4 bytes

12.2 Common Mistakes to Avoid

- Don't confuse the direction when reading division remainders
- Remember that leading zeros don't change the value
- Be careful with positional notation - rightmost digit is always the 2^0 position
- Don't confuse bits and bytes when calculating file sizes
- Remember that 1 KB = 1,024 bytes, not 1,000 bytes
- ASCII vs Unicode: ASCII is limited to English, Unicode supports all languages

13 Additional Resources

13.1 Quick Reference Table

Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
15	1111