Computer Organization & Architecture

Cybersecurity Department

Course Code: ***CBS219***

Lecture 3: Data Representation

Halal Abdulrahman Ahmed

Lecture Outline

- Introduction to Data Representation
- Binary, Octal, Decimal, Hexadecimal Systems
- Number System Conversions
- Signed vs Unsigned Numbers
- 1’s & 2’s Complement (Negative Numbers)
- Floating-Point Representation (Real Numbers)
- Parity Bit & Error Detection
- Bits, Bytes & Word Size
- Text Code

Learning Outcomes

By the end of this lecture, students will be able to:

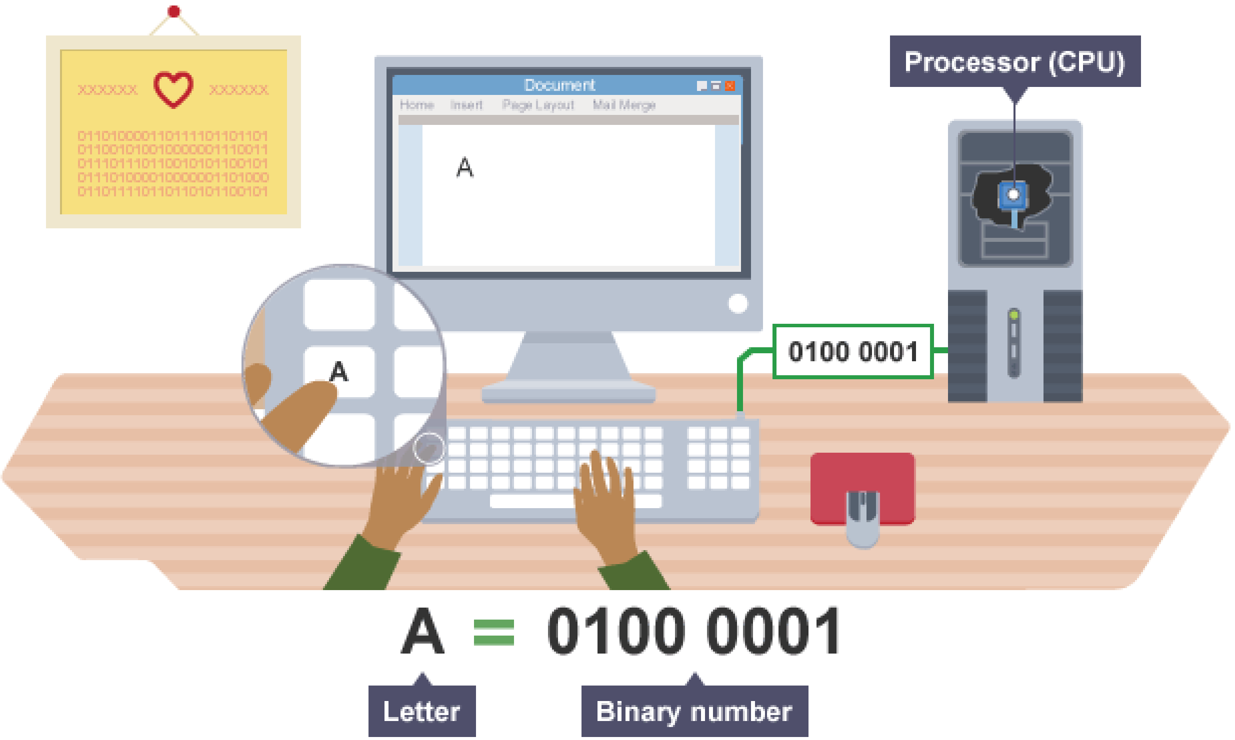
- Explain why computers use binary to represent data
- Convert numbers between binary, octal, decimal & hexadecimal
- Distinguish between signed and unsigned integers
- Represent negative numbers using 1's and 2's complement
- Understand floating-point (IEEE-754) representation basics
- Explain parity bit and its role in error detection
- Define bit, byte, word-size, ASCII, and Unicode

Data Representation

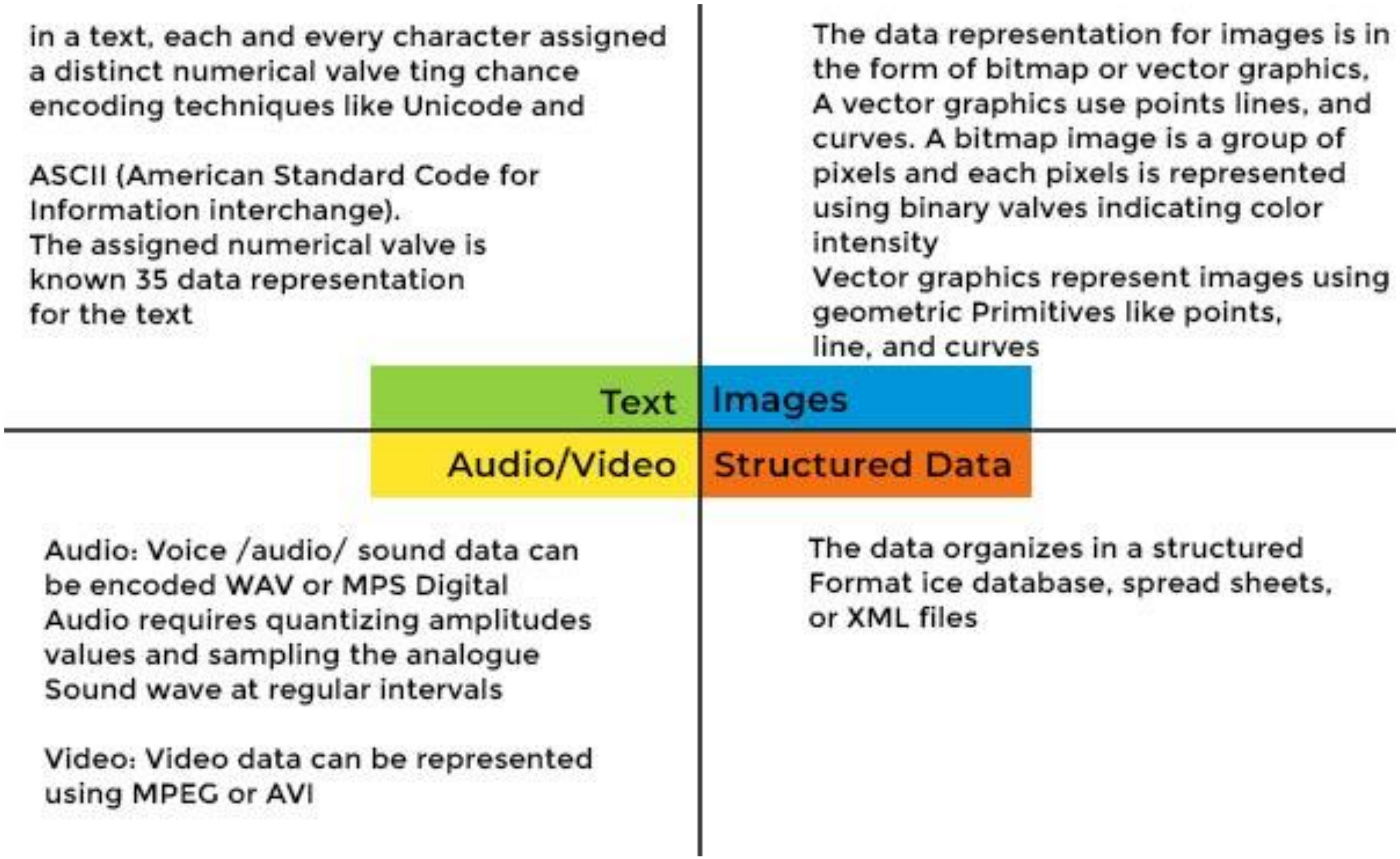
- Computers do not understand human language; they understand data within the prescribed form. Data representation is a method to represent data and encode it in a computer system.
- They understand only 0 and 1 and represent numbers, text, and other information using binary digits (bits).

Data Representation

- Everything, numbers, letters, images, and sounds must be converted into binary form. Understanding data representation helps in memory design, error detection, and cybersecurity.

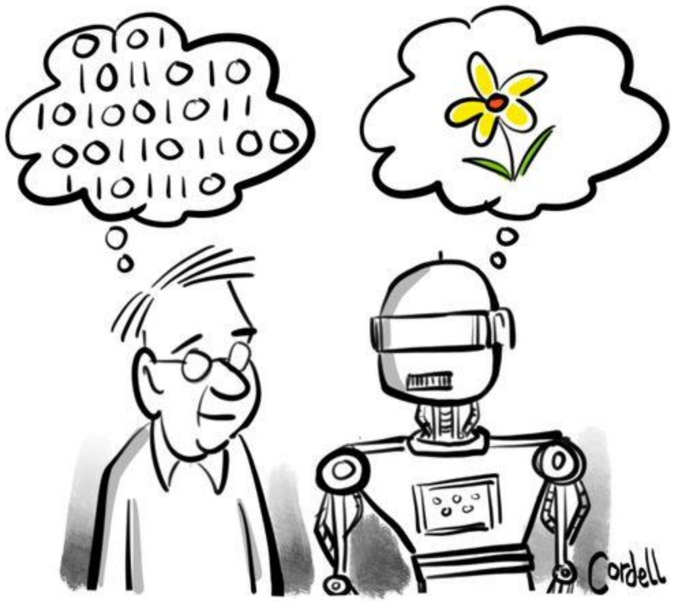


Some Common Data Representation Methods

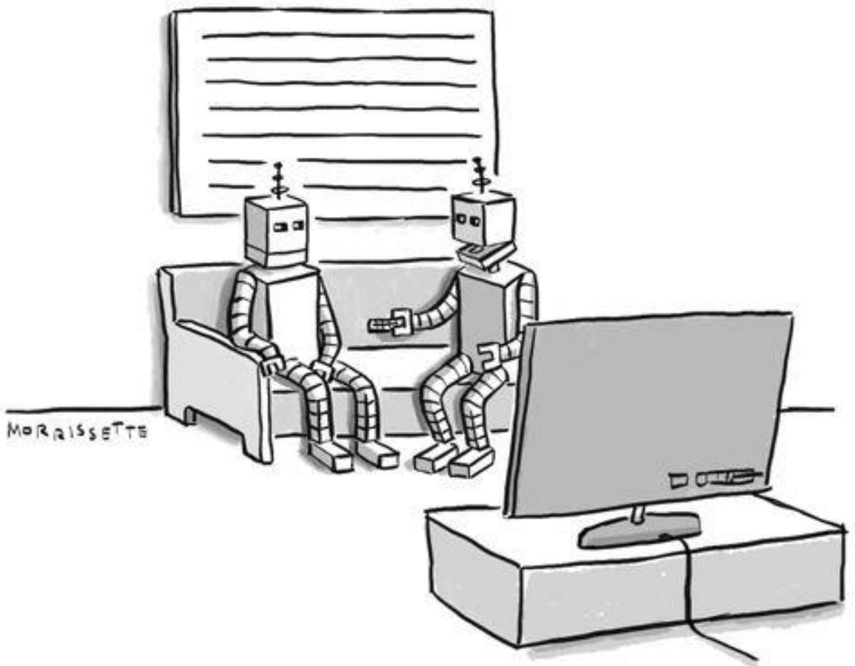


Computers represent data in the following forms:

- Number System
- Bits and Bytes
- Text Code



Number System



"Should we watch 'Zero' or 'One?'"

Number System

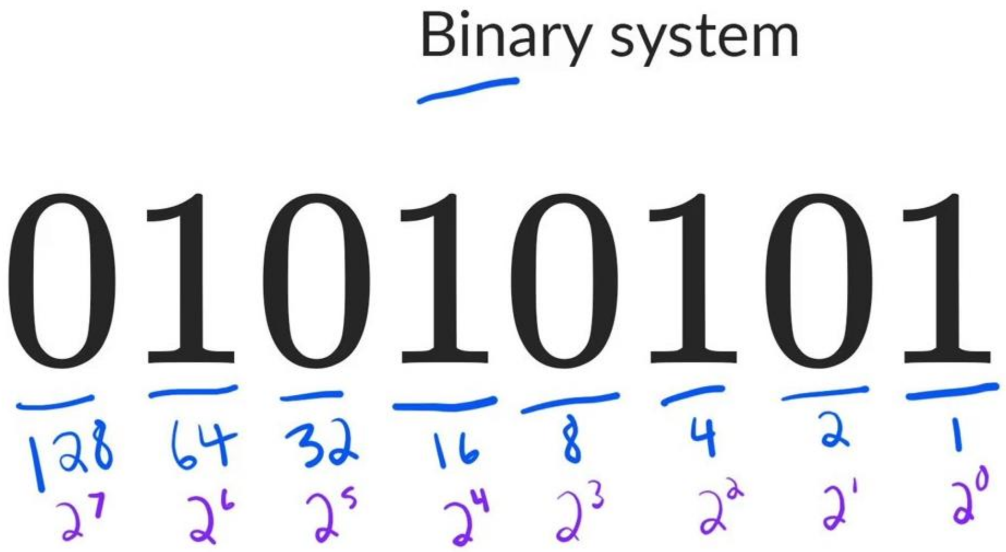
A computer system considers numbers as data; it includes integers, decimals, and complex numbers. All the inputted numbers are represented in binary formats like 0 and 1. A number system is categorized into four types:

1. **Binary**
2. **Octal**
3. **Decimal**
4. **Hexadecimal number**

The DaVinci Code

Binary Number System

A binary number system is a base of all the numbers considered for data representation in the digital system. A binary number system consists of only two values, either 0 or 1; so its base is 2. It can be represented to the external world as $(10110010)_2$. A computer system uses binary digits (0s and 1s) to represent data internally.

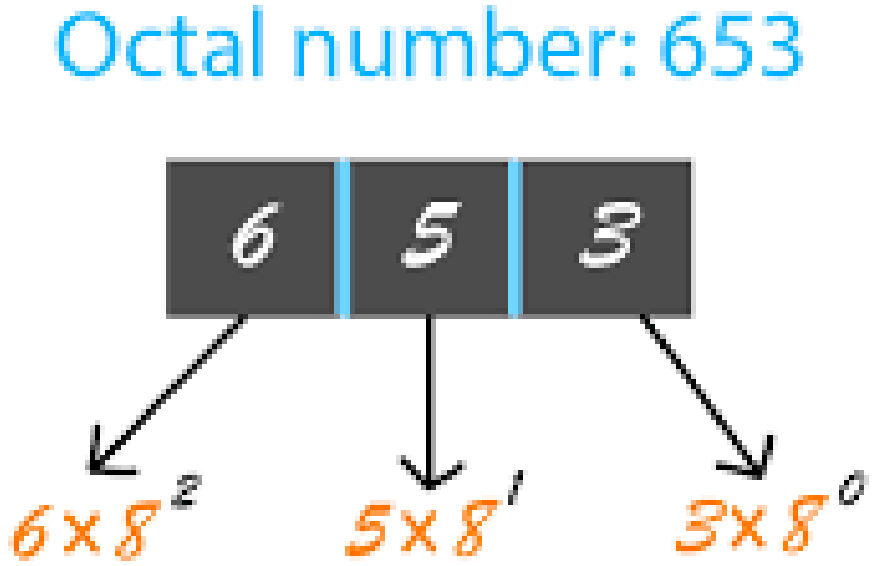


Binary (Base 2)

Bit Position	7	6	5	4	3	2	1	0
Value (2^n)	128	64	32	16	8	4	2	1

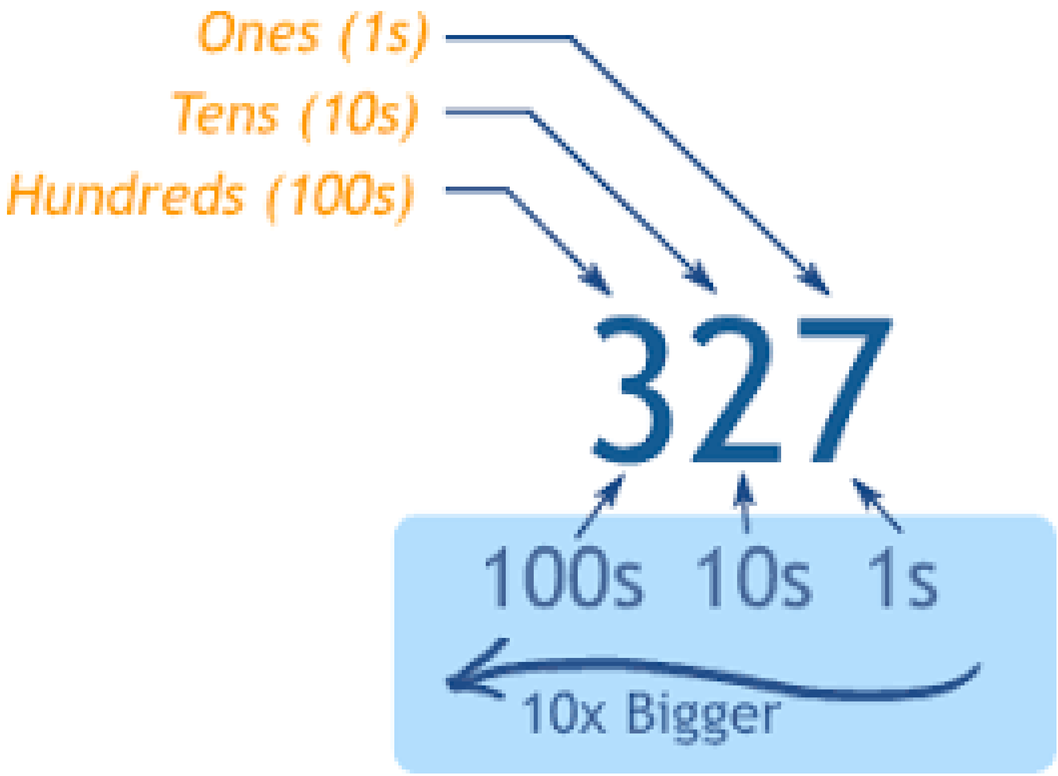
Octal Number System

The octal number system represents values in 8 digits. It consists of digits 0,12,3,4,5,6, and 7; so its base is 8. It can be represented to the external world as $(324017)_8$.



Decimal Number System

Decimal number system represents values in 10 digits. It consists of digits 0, 12, 3, 4, 5, 6, 7, 8, and 9; so its base is 10. It can be represented to the external world as $(875629)_{10}$.



Hexadecimal Number System

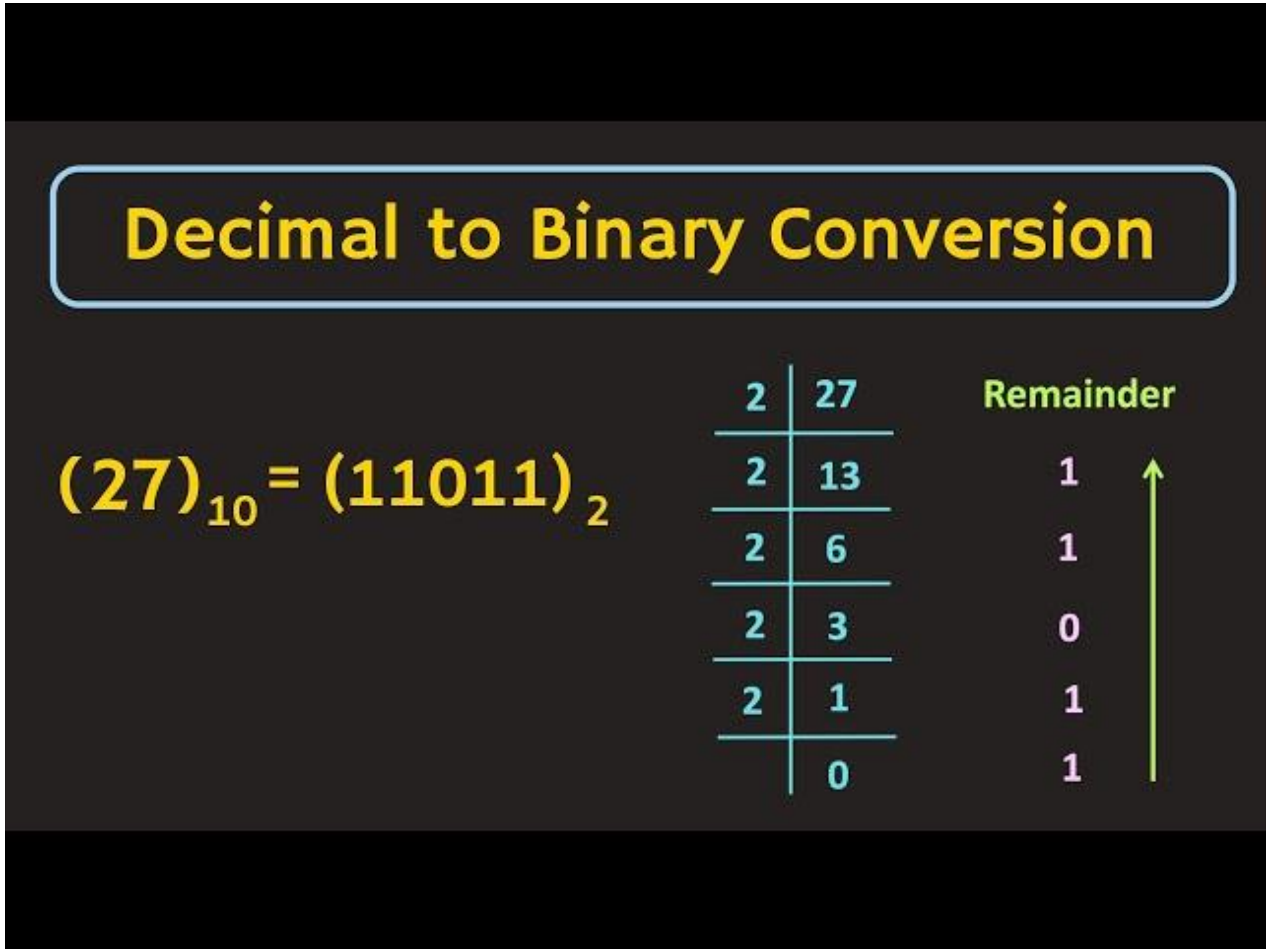
Hexadecimal number system represents values in 16 digits. It consists of digits 0, 12, 3, 4, 5, 6, 7, 8, and 9 then it includes alphabets A, B, C, D, E, and F; so its base is 16. Where A represents 10, B represents 11, C represents 12, D represents 13, E represents 14 and F represents 15.

Number Systems Overview

System	Base (Radix)	Digits Used	Example	Usage
Decimal	10	0–9	845	Human counting
Binary	2	0, 1	101101	Computer processing
Octal	8	0–7	527_8	Old systems shorthand
Hexadecimal	16	0–9, A–F	$3AF_{16}$	Memory addressing

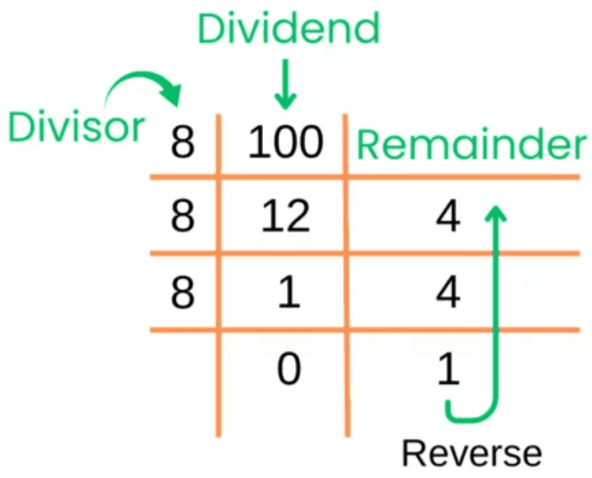
EXAMPLE

Example 1


Decimal to Binary Conversion
(27)10 = (11011) 2
2 | 27
2 | 13
2 | 6
2 | 3
2 | 1
0
Remainder
1
1
0
1
1

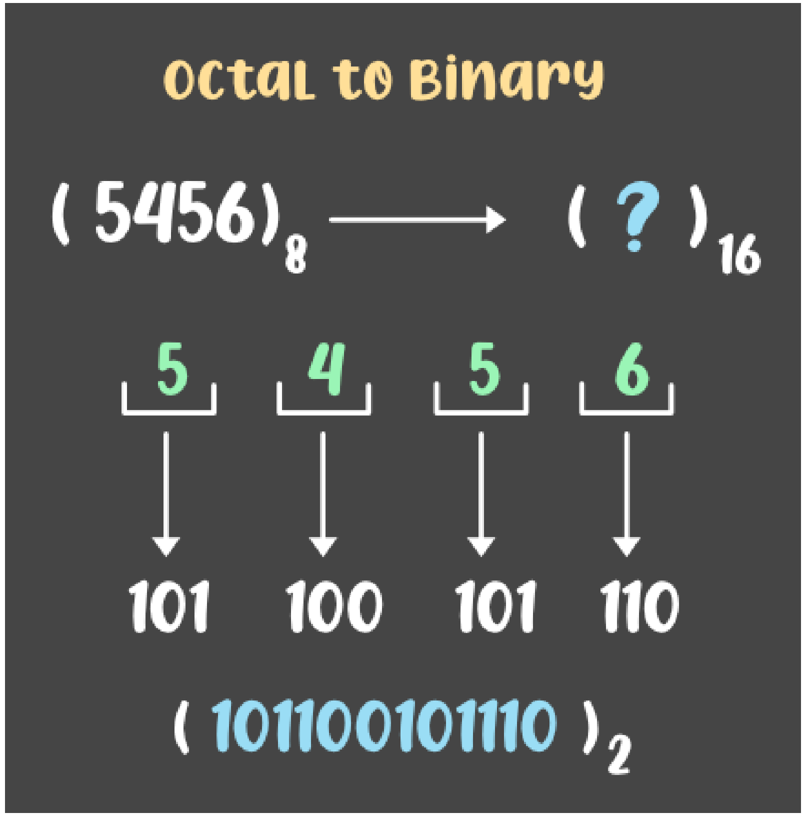
Example 2

Decimal= 100 ⟶ Octal ?

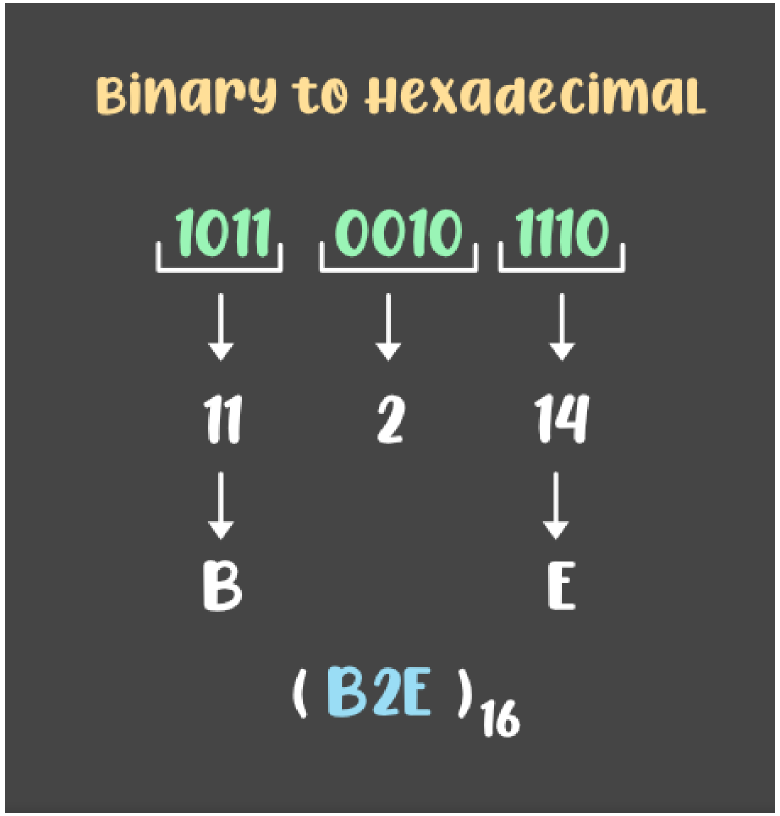


100 ⟶ 144

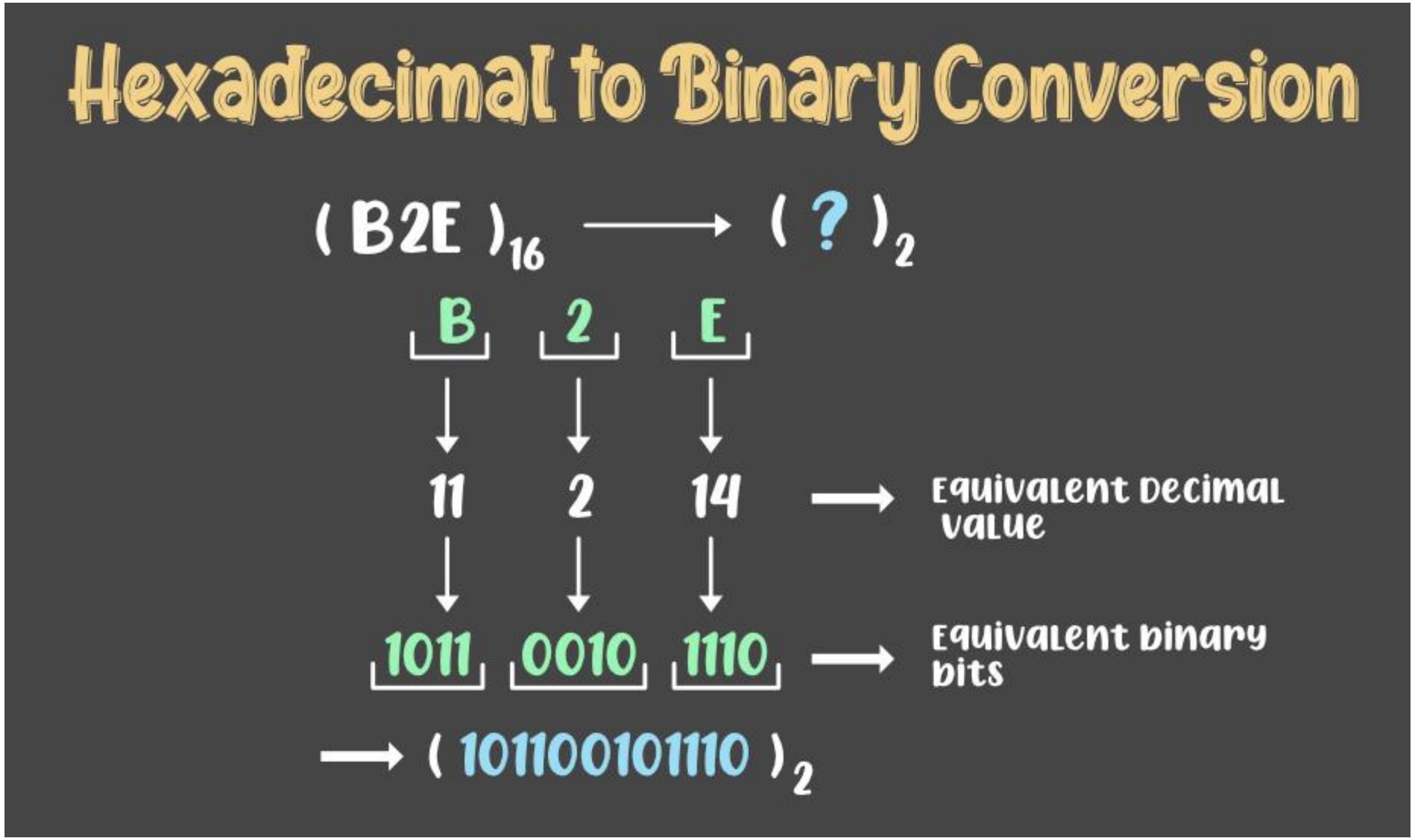
Example 3



Example 4

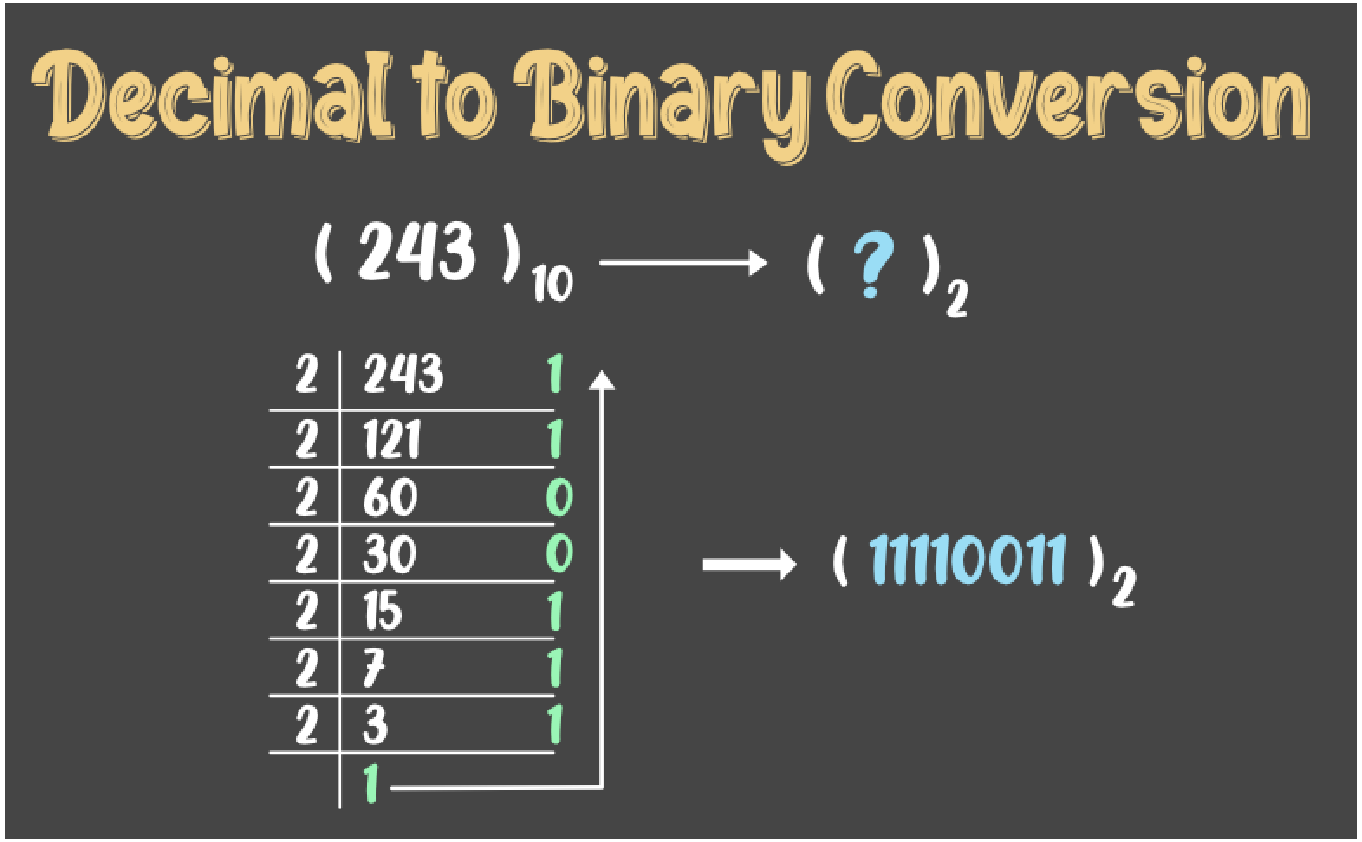


Example 5



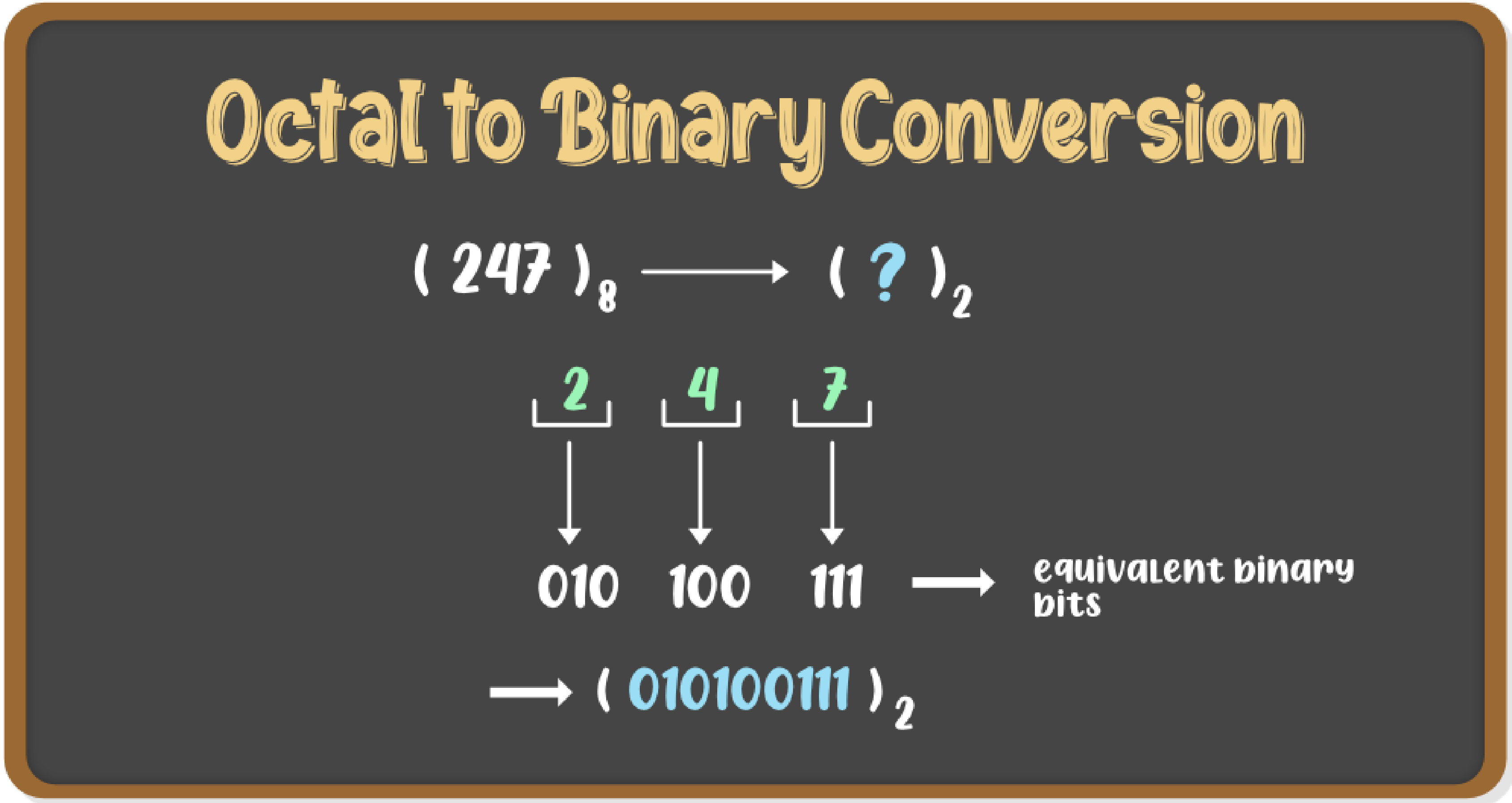
Hexadecimal to Binary Conversion
(B2E)16 → (?)2
B 2 E
11 2 14
Equivalent Decimal value
1011 0010 1110
Equivalent binary bits
→ (101100101110)2

Example 6



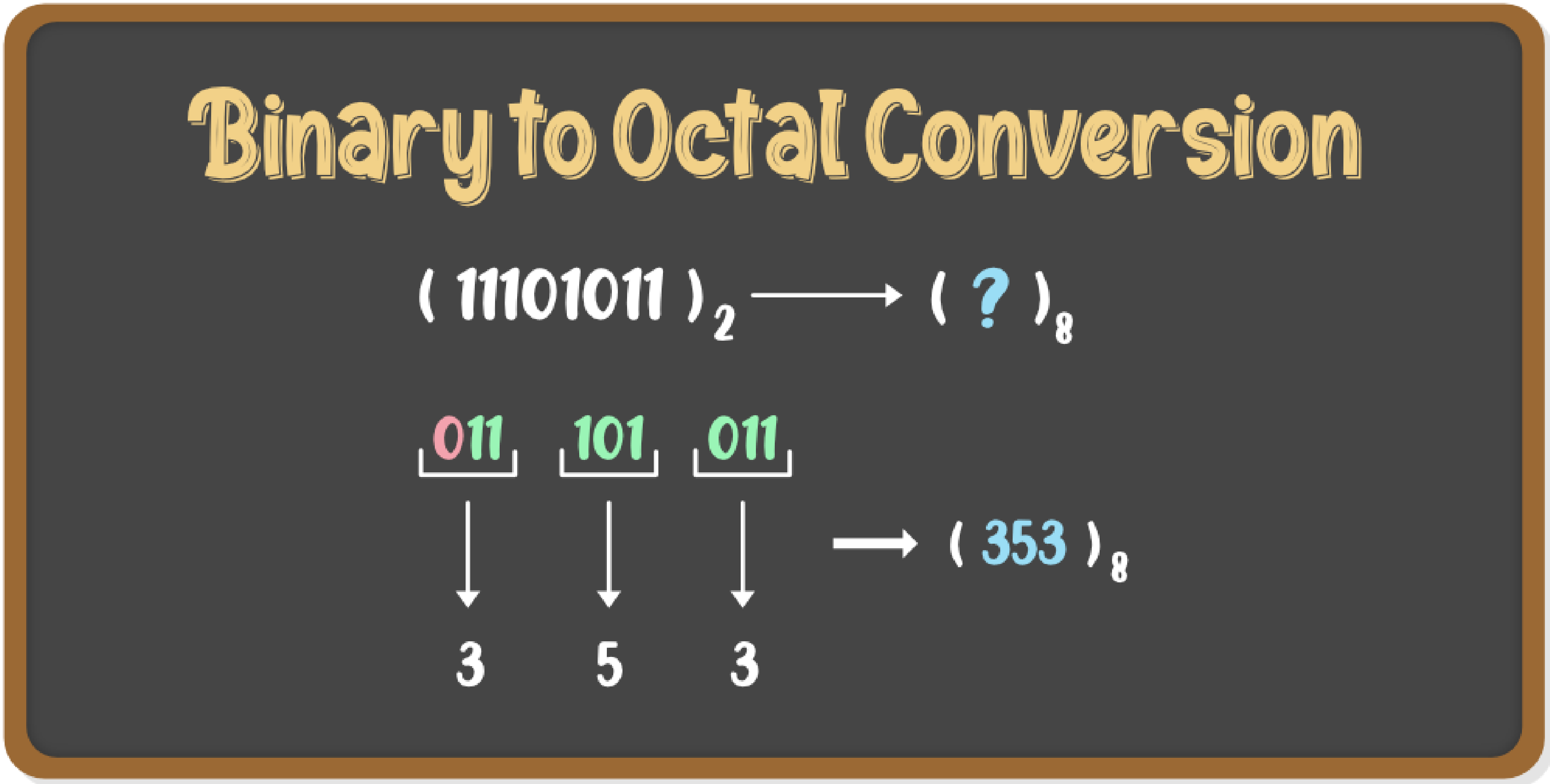
Decimal to Binary Conversion
(243)10 → (?)2
2 | 243 1
2 | 121 1
2 | 60 0
2 | 30 0
2 | 15 1
2 | 7 1
2 | 3 1
1
→ (11110011)2

Example 7

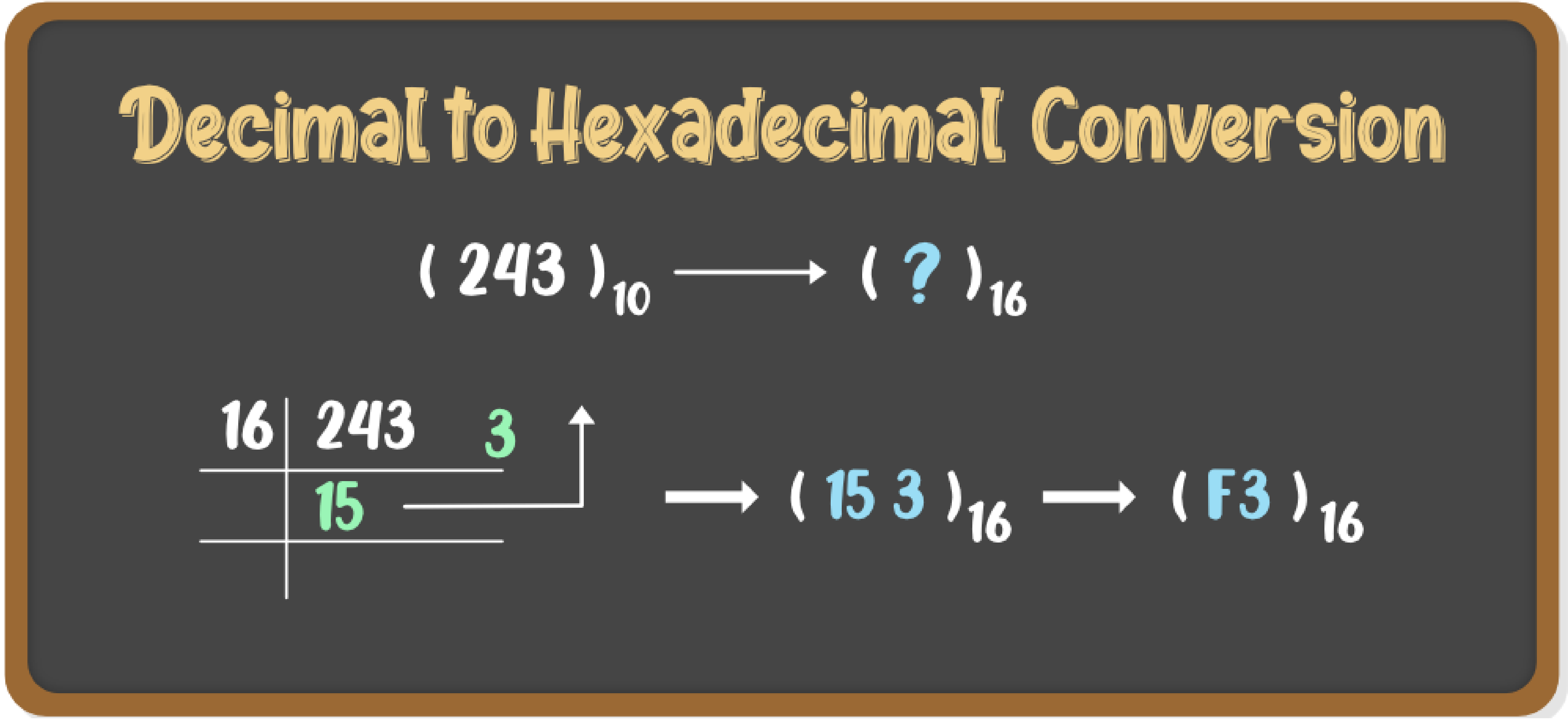


Octal to Binary Conversion
(247)8 → (?)2
2 4 7
010 100 111 → equivalent binary bits
→ (010100111)2

Example 8



Example 9



Decimal to Hexadecimal Conversion
(243)10 → (?)16
16 | 243 3
15
→ (15 3)16 → (F3)16

Example 10

$$670_{10} = 1236_8$$

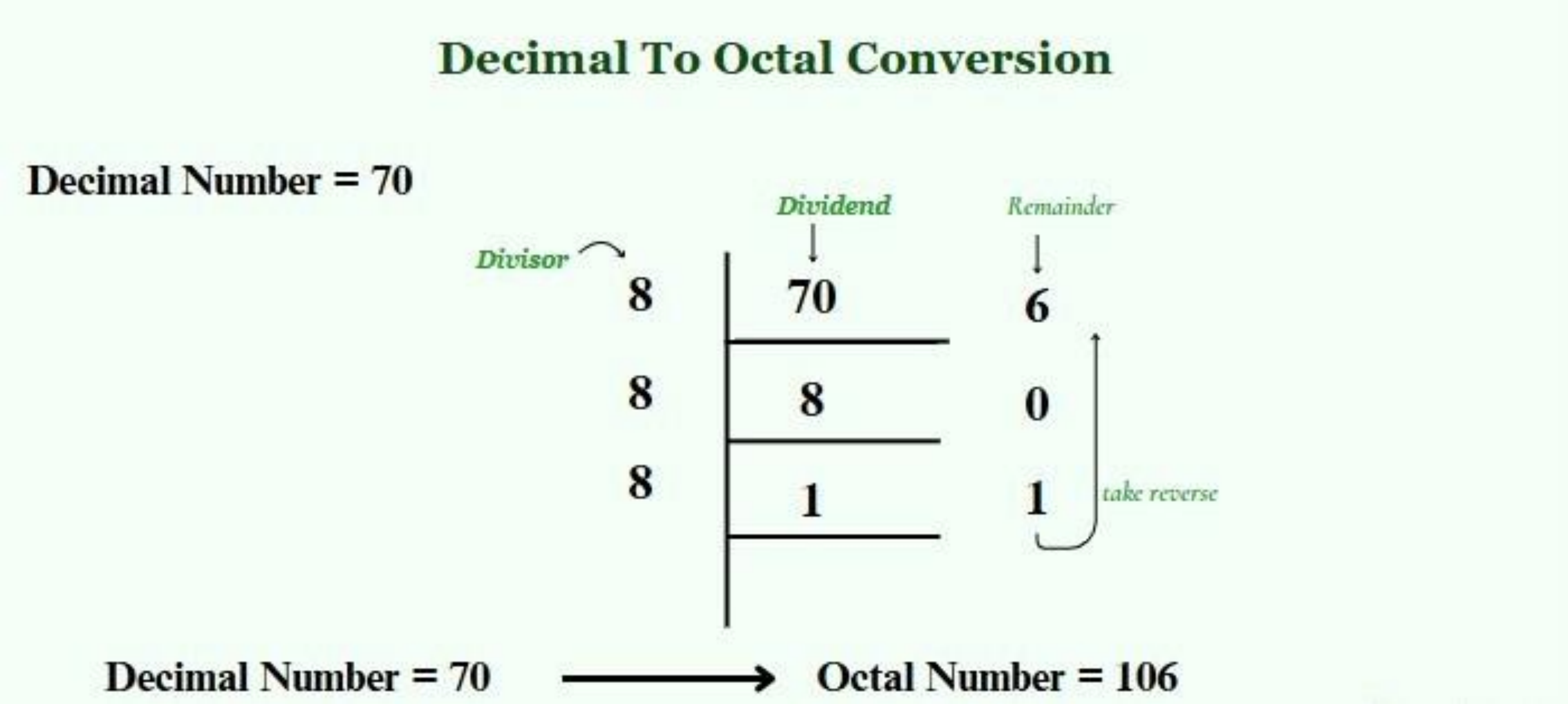
$$670 \div 8 = 83 \text{ r.} 6$$

$$83 \div 8 = 10 \text{ r.} 3$$

$$10 \div 8 = 1 \text{ r.} 2$$

$$1 \div 8 = 0 \text{ r.} 1$$

Example 11


Decimal To Octal Conversion
Decimal Number = 70
Dividend
Remainder
Divisor
8
70
6
8
8
0
8
1
1
take reverse
Decimal Number = 70
Octal Number = 106

Example 12

Hexadecimal to Decimal

Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Convert **2c9b** to decimal

2 c 9 b

$$(2 \times 16^3) + (12 \times 16^2) + (9 \times 16^1) + (11 \times 16^0)$$

$$8192 + 3072 + 144 + 11$$

$$11419$$

$$\therefore (2c9b)_{16} = (11419)_{10}$$

Unsigned vs Signed Numbers

Type	Range (8-bit)	Example
Unsigned	0 to 255	00001011 = 11
Signed (2’s Complement)	-128 to +127	11111011 = -5

Unsigned Numbers:

Only represent **positive values**.

All bits are used for the number itself.

For 8 bits → range is **0 to 255**.

($00000000_2 = 0$, $11111111_2 = 255$)

Signed Numbers (2’s Complement):

Can represent **positive and negative** values.

The **leftmost bit** shows the sign:

- 0 → positive
- 1 → negative

For 8 bits → range is **–128 to +127**.

($10000000_2 = -128$, $01111111_2 = +127$)

1’s and 2’s Complement

Operation	Example	Result
1’s Complement	Invert bits of 00000101	11111010
2’s Complement	Invert + 1	11111011 (–5)

2’s Complement Arithmetic

Operation	Binary	Result
(+5) + (–5)	00000101 + 11111011	00000000
(–3) + (–2)	11111101 + 11111110	11111011 (–5)

Floating-Point Representation

Floating-point is how computers store **real numbers** (numbers with decimals). Computers use the **IEEE-754 standard** to store these numbers. Floating-point is how computers store decimal numbers. It breaks the number into 3 parts: sign (positive/negative), exponent (size), and mantissa (value).

Part	Bits	Purpose
Sign	1 bit	Tells if number is + or − (0 = +, 1 = −)
Exponent	8 bits	Shows how big or small the number is
Mantissa (Fraction)	23 bits	Stores the decimal part

Parity Bit

A **parity bit** is an extra bit added to a binary message to **detect errors** when data is sent from one device to another. It checks whether the number of **1s** is even or odd.

Type	Rule	Goal
Even Parity	Add 1 if number of 1s is odd	Make total 1s **even**
Odd Parity	Add 1 if number of 1s is even	Make total 1s **odd**

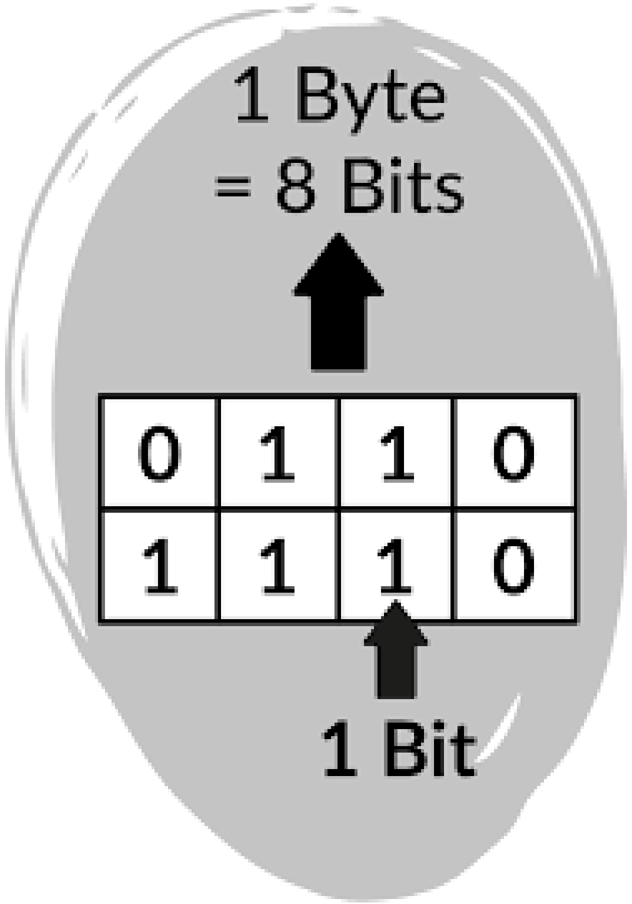
Example of Parity Bit

Original data: 1010

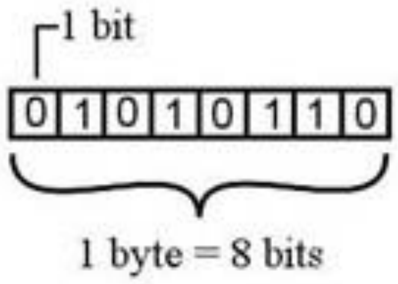
Number of 1s = 2 (even)

Type	Action	Result
Even Parity	Already even → add 0	10100
Odd Parity	Need odd → add 1	10101

Bits and Bytes



Bits and Bytes



Bits

- A bit is the smallest data unit that a computer uses in computation; all the computation tasks done by the computer systems are based on bits. A bit represents a binary digit in terms of 0 or 1. The computer usually uses bits in groups. It's the basic unit of information storage and communication in digital computing.

Bytes

- A group of eight bits is called a byte. Half of a byte is called a nibble; it means a group of four bits is called a nibble. A byte is a fundamental addressable unit of computer memory and storage. It can represent a single character, such as a letter, number, or symbol using encoding methods such as ASCII and Unicode.

Bits and Bytes

Unit	Size	Example	Use
Bit	1 binary digit	0 or 1	Smallest unit of data
Nibble	4 bits	1010	Used in hexadecimal
Byte	8 bits	01000001	Represents a single ASCII character
Word	16–64 bits	Varies by CPU	Represents data or instruction size

How many bits are 2 bytes?

16 bits.

What is a “word” in computer architecture?

- A **word** is the **natural unit of data** that a CPU can process or transfer at once.
- The CPU’s *word size* depends on how many bits it can handle in one operation.

CPU Type	Word Size	Equivalent in Bytes
32-bit CPU	32 bits	4 bytes
64-bit CPU	64 bits	8 bytes

- A 32-bit CPU means its *word size* is 32 bits (4 bytes). It processes 32 bits of data in a single operation.
- A 64-bit CPU has a *word size* of 64 bits (8 bytes). It can handle 64 bits of data in one operation.

Text Code

Text Code

A Text Code is a static code that allows a user to insert text that others will view when they scan it. It includes alphabets, punctuation marks and other symbols. Some of the most commonly used text code systems are:

- ASCII
- Unicode

ASCII

- ASCII stands for American Standard Code for Information Interchange. It is an 8-bit code that specifies character values from 0 to 127. ASCII is a standard for the Character Encoding of Numbers that assigns numerical values to represent characters, such as letters, numbers, exclamation marks and control characters used in computers and communication equipment that are using data.
- ASCII originally defined 128 characters, encoded with 7 bits, allowing for 2^7 (128) potential characters. The ASCII standard specifies characters for the English alphabet (uppercase and lowercase), numerals from 0 to 9, punctuation marks, and control characters for formatting and control tasks such as line feed, carriage return, and tab.

ASCII Code Examples

Character	Decimal	Binary
A	65	01000001
B	66	01000010
a	97	01100001
0	48	00110000

Unicode

It is a worldwide character standard that uses 4 to 32 bits to represent letters, numbers and symbols. Unicode is a standard character encoding which is specifically designed to provide a consistent way to represent text in nearly all of the world's writing systems. Every character is assigned a unique numeric code, program, or language. Unicode offers a wide variety of characters, including alphabets, ideographs, symbols, and emojis.

Data Representation and Cybersecurity

Concept	Application
Binary patterns	Used in encryption and hashing
ASCII/Unicode	Encoding attacks (XSS, SQL Injection)
Parity/ECC	Error detection and correction
Overflow	Buffer overflow exploits

Further Learning Resources

- **Number System Conversion Practice**

  WorkyBooks – Octal Number System

  Beginner-friendly explanation of the octal system with examples and quizzes.

- **Number System Conversions (Blog)**

  WordPress – Conversion of Number Systems

  Simple explanation of how to convert between binary, decimal, octal, and hexadecimal systems.


Any
Question