



جامعة المستقبل
AL MUSTAQBAL UNIVERSITY
كلية الهندسة والتقنيات الهندسية

# Electrical Engineering Techniques Department

## 3rd Stage

# Microprocessor

## Lecture 1 – Recall (Number Systems)

Dr. Mohammed Fadhil
PhD in Computer Networks
Email: mohammed.fadhil1@uomus.edu.iq

# Overview Objectives

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

Define digital logic and its applications

Understand numbering system and its type

Understand decimal, binary, octal and hexadecimal numbers.

Count in decimal, binary, octal and hexadecimal systems.

Understand Conversion and Arithmetic Operations

# Introduction

- **Digital Computers**
  - Digital computers process and store data as discrete binary digits (bits). These bits represent states like:
    - **Magnetic markers (present/absent)**
    - **Switches (on/off)**
    - **Relay states**
    - **All data—including numbers, text, and commands—is converted to digital form.**
- **Digital Logic**
  - **The foundation of modern electronics (computers, phones, etc.), digital logic uses binary code (0s and 1s) to:**
    - **Design circuits (e.g., logic gates like AND, OR, NOT)**
    - **Process input signals into specific outputs**
    - **Enable computing, robotics, and electronic systems**
- **Digital Logic Design**
  - A core discipline in electrical and computer engineering, it involves:
  - Creating hardware (circuit boards, microchips)
  - Integrating electrical and computational traits (power, logic functions, protocols)
  - Processing inputs for computers, phones, navigation systems, and other tech

# Number Systems

- **A numbering system** is a mathematical framework for representing and organizing numbers using a specific set of symbols and rules.
- It defines how numbers are expressed and manipulated within a given base or radix, determining the number of unique symbols representing values.
- Numbering systems are essential for various fields, including mathematics, engineering, and computer science, as they provide a structured way to handle numerical data.

# Number Systems

- For digital information to be processed by a circuit, it must be represented in a suitable format for that circuit.
- To achieve this, **a base B number system** (B a natural number ≥ 2) needs to be chosen.
- Several number systems are used in digital technology, with the most commonly used systems are:
  - **Decimal** (base **10**)
  - **Binary** (base **2**)
  - **Octal** (base **8**)
  - **Hexadecimal** (base **16**).

# Decimal number system

- **Key Features:**
  - **Base-10 system** using digits: 0,1,2,3,4,5,6,7,8,9
  - **Positional-value system:** Digit value depends on its place.
- **Examples:**
  - **453**
    - 4 (**MSD**): Hundreds ($10^2$)
    - 5: Tens ($10^1$)
    - 3 (**LSD**): Units ($10^0$)
- **Expanded Form:**
  - **9261** = $(9\times10^3) + (2\times10^2) + (6\times10^1) + (1\times10^0)$
  - **3267.317** = $(3\times10^3) + (2\times10^2) + (6\times10^1) + (7\times10^0) + (3\times10^{-1}) + (1\times10^{-2}) + (7\times10^{-3})$

# Decimal number system

- **Visual Representation: 5432.387**

| Position | $10^3$ | $10^2$ | $10^1$ | $10^0$ | . | $10^{-1}$ | $10^{-2}$ | $10^{-3}$ |
|---|---|---|---|---|---|---|---|---|
| Example | 5 | 4 | 3 | 2 | . | 3 | 8 | 7 |
| Weight | 1000 | 100 | 10 | 1 | . | 0.1 | 0.01 | 0.001 |

- **Key Terms:**
  - **MSD**: Most Significant Digit (leftmost)
  - **LSD**: Least Significant Digit (rightmost)
  - **Decimal point**: Separates integer (left) and fractional (right) parts.

# Binary number system

- **Key Features:**
  - **Base-2 system** using digits: 0, 1 (bits)
  - **Positional-value system**: Each bit's value depends on its **power-of-2 position.**

- **Example**
  - $1101.011_2$

| Position | $2^3$ | $2^2$ | $2^1$ | $2^0$ | . | $2^{-1}$ | $2^{-2}$ | $2^{-3}$ |
|---|---|---|---|---|---|---|---|---|
| Bit | 1 | 1 | 0 | 1 | . | 0 | 1 | 1 |
| Weight | 8 | 4 | 2 | 1 | . | 0.5 | 0.25 | 0.125 |

- **Decimal Conversion:**
  - $(1\times8) + (1\times4) + (0\times2) + (1\times1) + (0\times0.5) + (1\times0.25) + (1\times0.125)$
  - $= 8 + 4 + 0 + 1 + 0 + 0.25 + 0.125 = 13.375_{10}$

# Binary number system

- **Universal Rule:**
  - Any number (decimal, binary, octal, hexadecimal) equals the sum of each **digit ×** its positional value.
- **Key Terms & Notes**
  - **Bit**: Binary digit (0 or 1)
  - **Byte**: 8 bits
  - **MSB**: Most Significant Bit (leftmost)
  - **LSB**: Least Significant Bit (rightmost)
  - **Binary ↔ Decimal**:
    - Digital systems output binary → converted to decimal for display.
    - User inputs (decimal) → converted to binary for processing.

# Octal Number System

- **Key Features:**
  - Uses digits (**Base-8 system**): 0, 1, 2, 3, 4, 5, 6, 7
  - Positional-value system: Each digit's value depends on its power-of-8 position.
- **Examples**:
  - Octal Number: $(1101.011)_8$
    - MSD (Most Significant Digit): Leftmost 1
    - LSD (Least Significant Digit): Rightmost 1
  - Expanded Form:
    - $(4372)_8 = (4\times8^3) + (3\times8^2) + (7\times8^1) + (2\times8^0)$
    - $(372.36)_8 = (3\times8^2) + (7\times8^1) + (2\times8^0) + (3\times8^{-1}) + (6\times8^{-2})$

# Octal Number System

- **Visual Representation: $(1101.01)_8$**

| Position | $8^3$ | $8^2$ | $8^1$ | $8^0$ | . | $8^{-1}$ | $8^{-2}$ |
|---|---|---|---|---|---|---|---|
| Example | 1 | 1 | 0 | 1 | . | 0 | 1 |
| Weight | 512 | 64 | 8 | 1 | . | 0.125 | 0.0156 |

- **Key Notes:**
  - **No digits 8 or 9 (only 0-7).**
  - **Common Uses:**
    - **Shorthand for binary (3 bits = 1 octal digit).**
    - **Legacy computing systems.**

# Hexadecimal Number System

- **Key Features:**
  - Uses digits (**Base-16 system**): 0, 1, 2, ..., 9, A(10), B(11), C(12), D(13), E(14), F(15)
  - Positional-value system: Each digit's value scales by powers of **16**.
- Examples:
  - Hexadecimal Expansion:
    - $(A2C9)_{16} = (10\times16^3) + (2\times16^2) + (12\times16^1) + (9\times16^0)$
    - $(27.38)_{16} = (2\times16^1) + (7\times16^0) + (3\times16^{-1}) + (8\times16^{-2})$
  - Decimal Equivalents:
    - **A → 10, F → 15**
    - $(1F)_{16} = 1\times16 + 15 = (31)_{10}$

# Hexadecimal Number System

- **Visual Representation: $(A2C9.38)_{16}$**

| Position | $16^3$ | $16^2$ | $16^1$ | $16^0$ | . | $16^{-1}$ | $16^{-2}$ |
|---|---|---|---|---|---|---|---|
| Example | A | 2 | C | 9 | . | 3 | 8 |
| Weight | 4096 | 256 | 16 | 1 | . | 0.0625 | 0.0039 |

- **Why Hexadecimal?**
  - Compact representation of binary (4 bits = 1 hex digit).
  - Commonly used in programming, memory addressing, and digital systems.

# Decimal Numbers System

- Decimal number system is a system which has a radix (r = 10) and ten digits (0, 1, 2, 3, 4, 5, 6, 7, 8, and 9). The value of the number is the sum of digits after each has been multiplied by its weight.
- **Example 1 : Decimal Number System (Weighted Positional Notation)**

  **Decimal Number: $(83)_{10}$**

  **Expanded Form:**

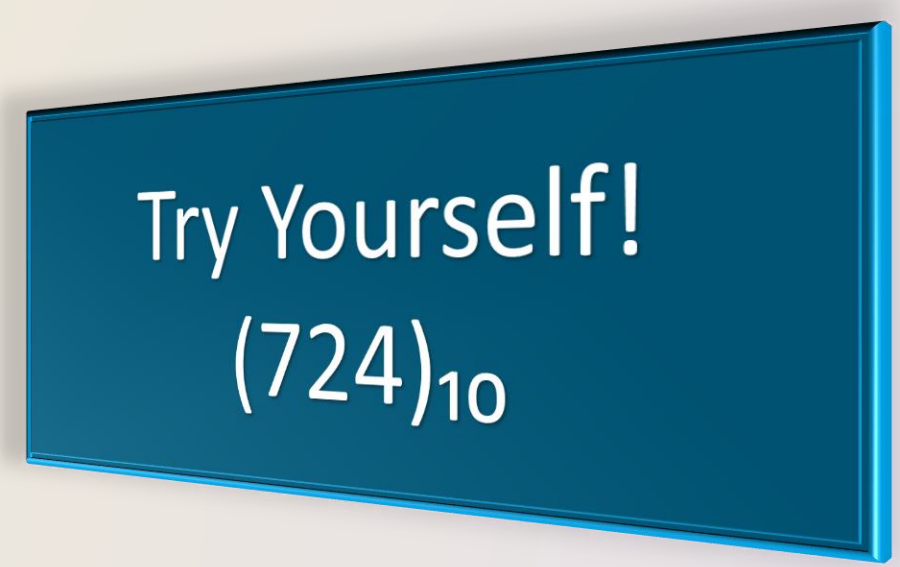
  $= 8 \times 10^1 + 3 \times 10^0$

  $= 8 \times 10 + 3 \times 1$

  $= 80 + 3$

  **Digit Weights:**

  - Digit "8" → Weight = 10 ($10^1$)
  - Digit "3" → Weight = 1 ($10^0$)



# Binary Numbers System

- Binary numbers system is another way to count which have a radix (r = 2) and two digits or bits (0 and 1)
  - Counting in Binary System
    - **In decimal numbers system** we count by starting at 0 and count up to 9. Then start another digit position to the left and continue counting 10 through 99. Then start a third digit position to the left and continue counting from 100 to 999, and so on.
    - **In binary system**, the same situation occurs when we count, except that in binary system we have only ***two bits 0 and 1***. Including another bit position and continue counting, 10, 11. With three bits, we can continue to count, 100, 101, 110, and 111. To continue counting, we need a fourth bit, and so on.

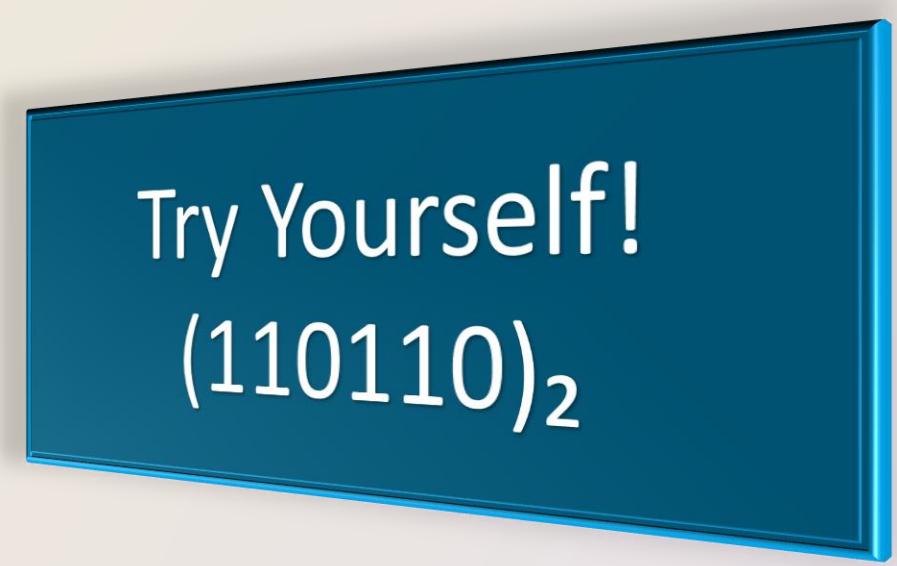
# Binary Numbers System

## Binary Numbers Table

| Number | Binary Number | Number | Binary Number | Number | Binary Number |
|---|---|---|---|---|---|
| 1 | 1 | 11 | 1011 | 21 | 10101 |
| 2 | 10 | 12 | 1100 | 22 | 10110 |
| 3 | 11 | 13 | 1101 | 23 | 10111 |
| 4 | 100 | 14 | 1110 | 24 | 11000 |
| 5 | 101 | 15 | 1111 | 25 | 11001 |
| 6 | 110 | 16 | 10000 | 26 | 11010 |
| 7 | 111 | 17 | 10001 | 27 | 11011 |
| 8 | 1000 | 18 | 10010 | 28 | 11100 |
| 9 | 1001 | 19 | 10011 | 29 | 11101 |
| 10 | 1010 | 20 | 10100 | 30 | 1111 |

# Conversion from Binary to Decimal

- The value of the binary number in decimal system can be computed as the sums of the bits after each have been multiplied by its weight as illustrated in the following examples:
  - **Binary Number: $(110)_2$**
    - **Step 1: Expand Using Positional Weights**
      - $= 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$
    - **Step 2: Calculate Powers of 2**
      - $= 1 \times 4 + 1 \times 2 + 0 \times 1$
    - **Step 3: Sum the Values**
      - $= 4 + 2 + 0$
        $= (6)_{10}$
- Starting from the left:
  - the first bit (1) has a weight of 4 ($2^2$),
  - the second bit (1) has a weight of 2 ($2^1$), and
  - the third bit (0) has a weight of 1 ($2^0$).



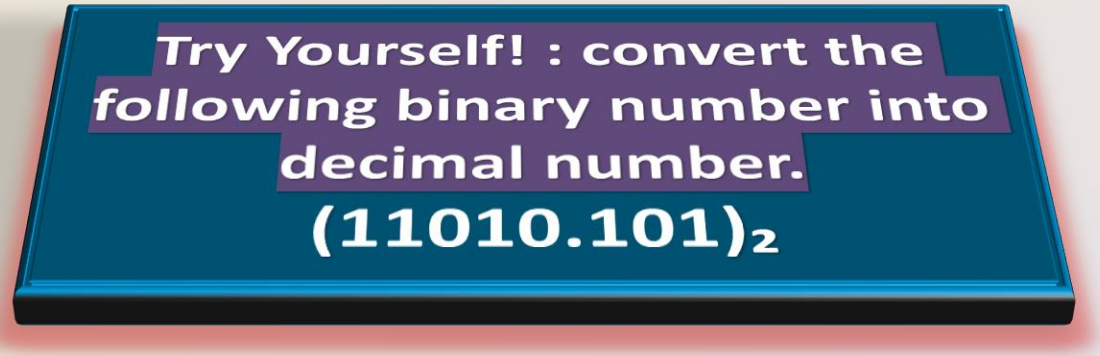
# Conversion from Binary to Decimal

- **Binary Number: $(10110.1)_2$**
- **Step-by-Step Conversion:**
  - **Integer Part (Left of Binary Point):**
    $= 1\times2^4 + 0\times2^3 + 1\times2^2 + 1\times2^1 + 0\times2^0$
    $= 1\times16 + 0\times8 + 1\times4 + 1\times2 + 0\times1$
  - **Fractional Part (Right of Binary Point):**
    $= 1\times2^{-1}$
    $= 1\times0.5$
  - **Summation:**
    $= 16 + 0 + 4 + 2 + 0 + 0.5$
    $= (22.5)_{10}$

- Starting from the left:
  - the first bit (1) has a weight of 16 ($2^4$),
  - the second bit (0) has a weight of 8 ($2^3$),
  - the third bit (1) has a weight of 4 ($2^2$),
  - the fourth bit (1) has a weight of 2 ($2^1$), and
  - the fifth bit (0) has a weight of 1 ($2^0$).
  - After the binary point, the following bit (1) has a weight of 0.5 ($2^{-1}$).

# Conversion from Binary to Decimal

- Binary Number: $(101.1011)_2$
- Step-by-Step Conversion:
  - Integer Part (Left of Binary Point):
    a) $1\times2^2 = 1\times4 = 4$
    b) $0\times2^1 = 0\times2 = 0$
    c) $1\times2^0 = 1\times1 = 1$
  - Fractional Part (Right of Binary Point):
    a) $1\times2^{-1} = 1\times0.5 = 0.5$
    b) $0\times2^{-2} = 0\times0.25 = 0$
    c) $1\times2^{-3} = 1\times0.125 = 0.125$
    d) $1\times2^{-4} = 1\times0.0625 = 0.0625$
  - Summation:
    = 4 (integer) + 0 + 1 + 0.5 (fraction) + 0 + 0.125 + 0.0625
    = $(5.6875)_{10}$



# Conversion from Decimal to Binary

There are two methods for converting decimal numbers to binary numbers:

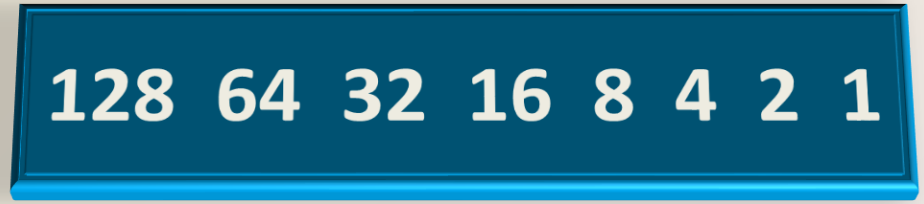
1. **Sum of Weights Method**
   - This method determines the binary equivalent by identifying which combination of binary weights sum to the decimal value.
   - The weight positions in a binary number are represented as:

$$2^{n-1} \ldots 2^5 \; 2^4 \; 2^3 \; 2^2 \; 2^1 \; 2^0 \bullet 2^{-1} \; 2^{-2} \ldots 2^{-n}$$

↑

**Binary Point**

Where **n** is the number of bits from the binary point.




- $(7)_{10} = 4 + 2 + 1$
  $= 1\times4 + 1\times2 + 1\times1$
  $= 1\times2^2 + 1\times2^1 + 1\times2^0$
  $= (111)_2$
- $(13)_{10} = 8 + 4 + 0 + 1$
  $= 1\times8 + 1\times4 + 0\times2 + 1\times1$
  $= 1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0$
  $= (1101)_2$

# Conversion from Decimal to Binary

## 2. Repeated Division by 2 Method

- To convert a decimal number N to binary:
  i. Divide N by 2
  ii. Divide each resulting quotient by 2 until the quotient becomes 0
  iii. The remainders form the binary number:
    1. First remainder = Least Significant Bit (LSB)
    2. Last remainder = Most Significant Bit (MSB)

- **$(11)_{10}$ to Binary:**

| Division | Quotient | Remainder | Bit Position |
|---|---|---|---|
| $11 \div 2$ | 5 | 1 | LSB ($2^0$) |
| $5 \div 2$ | 2 | 1 | $2^1$ |
| $2 \div 2$ | 1 | 0 | $2^2$ |
| $1 \div 2$ | 0 | 1 | MSB ($2^3$) |

- **Result**: write remainders from bottom to top → **$(1011)_2$**

# Conversion from Decimal to Binary

- $(34)_{10}$ to Binary

| Division | Quotient | Remainder | Bit Position |
|---|---|---|---|
| $34 \div 2$ | 17 | 0 | LSB ($2^0$) |
| $17 \div 2$ | 8 | 1 | $2^1$ |
| $8 \div 2$ | 4 | 0 | $2^2$ |
| $4 \div 2$ | 2 | 0 | $2^3$ |
| $2 \div 2$ | 1 | 0 | $2^4$ |
| $1 \div 2$ | 0 | 1 | MSB ($2^5$) |

- **Result**: write remainders from bottom to top → $(100010)_2$

- $(45)_{10}$ to Binary

| Division | Quotient | Remainder | Bit Position |
|---|---|---|---|
| $45 \div 2$ | 22 | 1 | LSB ($2^0$) |
| $22 \div 2$ | 11 | 0 | $2^1$ |
| $11 \div 2$ | 5 | 1 | $2^2$ |
| $5 \div 2$ | 2 | 1 | $2^3$ |
| $2 \div 2$ | 1 | 0 | $2^4$ |
| $1 \div 2$ | 0 | 1 | MSB ($2^5$) |

- **Result:** write remainders from bottom to top → $(101101)_2$

# Binary Weights Table

| Integer Part (Positive Powers) | Fractional Part (Negative Powers) |
|---|---|
| $2^7 = 128$ | $2^{-1} = 0.5$ |
| $2^6 = 64$ | $2^{-2} = 0.25$ |
| $2^5 = 32$ | $2^{-3} = 0.125$ |
| $2^4 = 16$ | $2^{-4} = 0.0625$ |
| $2^3 = 8$ | $2^{-5} = 0.03125$ |
| $2^2 = 4$ | $2^{-6} = 0.015625$ |
| $2^1 = 2$ | $2^{-7} = 0.0078125$ |
| $2^0 = 1$ | $2^{-8} = 0.00390625$ |

# Conversion of decimal fractions to binary fractions

There are two methods for converting decimal numbers to binary numbers:

1. **Sum of Weights Method**
   - The sum of weights method can be applied to fractional decimal numbers as shown before. The following example illustrates the method:

- **Fraction $(0.625)_{10}$ → Binary**
  - **Weights: 0.5, 0.25, 0.125**
  - **Steps:**
    1. **0.5 ≤ 0.625? Yes → 1 (Remainder: 0.625 - 0.5 = 0.125)**
    2. **0.25 ≤ 0.125? No → 0**
    3. **0.125 ≤ 0.125? Yes → 1 (Remainder: 0.125 - 0.125 = 0)**
  - **Result: $0.101_2$ → (0.5 + 0 + 0.125).**
  - **Table Format:**

| Weight | 0.5 | 0.25 | 0.125 |
|---|---|---|---|
| Bit | 1 | 0 | 1 |

# Conversion of decimal fractions to binary fractions

## 2. Repeated Multiplication by 2 Method

- To convert a decimal fraction to binary fraction by this method, we need to apply the following steps:
  1. **Multiply** the decimal fraction by 2.
  2. **Separate** the result into its integer part (carry) and fractional part.
  3. **Repeat** the multiplication with the fractional part until it becomes 0 (or reaches the desired precision).
  4. **Collect** the integer parts (carries) in order—the **first carry** is the **Most Significant Bit (MSB)**, and the **last carry** is the **Least Significant Bit (LSB)**.

# Conversion of decimal fractions to binary fractions

- **Convert $(0.625)_{10}$ → Binary**

| Step | Multiply by 2 | Integer Part (Carry) | Fractional Part | Bit Position |
|---|---|---|---|---|
| 1 | 0.625 × 2 = 1.25 | 1 (MSB) | 0.25 | $2^{-1}$ |
| 2 | 0.25 × 2 = 0.5 | 0 | 0.5 | $2^{-2}$ |
| 3 | 0.5 × 2 = 1.0 | 1 (LSB) | 0.0 | $2^{-3}$ |

- **Result:**
  - write carries top to bottom → $0.101_2$

$= 1\times2^{-1} + 0\times2^{-2} + 1\times2^{-3} = 0.5 + 0 + 0.125$

$= 0.625_{10}$

- **Convert $(0.3125)_{10}$ → Binary**

| Step | Multiply by 2 | Integer Part (Carry) | Fractional Part | Bit Position |
|---|---|---|---|---|
| 1 | 0.3125 × 2 = 0.625 | 0 (MSB) | 0.625 | $2^{-1}$ |
| 2 | 0.625 × 2 = 1.25 | 1 | 0.25 | $2^{-2}$ |
| 3 | 0.25 × 2 = 0.5 | 0 | 0.5 | $2^{-3}$ |
| 4 | 0.5 × 2 = 1.0 | 1 (LSB) | 0.0 | $2^{-4}$ |

- **Result:**
  - write carries top to bottom → $0.0101_2$

$= 0\times2^{-1} + 1\times2^{-2} + 0\times2^{-3} + 1\times2^{-4} = 0 + 0.25 + 0 + 0.0625$

$= 0.3125_{10}$

# Homework!

- Convert the decimal number $(37.375)_{10}$ to binary number using:
  1. Sum-of-weights method.
  2. Repeated division-by-2 method.

**Answer $(100101.011)_2$**

# Quiz-1

- Convert the decimal number $(45.625)_{10}$ to binary number.

# Octal-to-Decimal Conversion

- Method:
  - **Multiply** each digit by its positional weight (**powers of 8**) and **sum** the results.
- Example:
  - Convert $372_8$ → Decimal
  - $\mathbf{372_8} = 3 \times 8^2 + 7 \times 8^1 + 2 \times 8^0$
  - $= 3 \times 64 + 7 \times 8 + 2 \times 1$
  - $= 192 + 56 + 2$
  - $= 250_{10}$

# Octal to Decimal Conversion (with Fraction)

**Note: Multiply each digit by 8^position**
**Integer part: left to right, powers start at 0**
**Fractional part: right to left, negative powers**

- **Example 1:**
  - Convert $24.6_8$ to its decimal equivalent
  - $24.6_8 = 2\times8^1 + 4\times8^0 + 6\times8^{-1}$
  - $= 2\times8 + 4\times1 + 6\times0.125$
  - $= 16 + 4 + 0.75$
  - $= 20.75_{10}$
- Result:
  - $24.6_8 = 20.75_{10}$

- **Example 2:**
  - Convert $15.2_8$ to its decimal equivalent
  - $15.2_8 = 1\times8^1 + 5\times8^0 + 2\times8^{-1}$
  - $= 1\times8 + 5\times1 + 2\times0.125$
  - $= 8 + 5 + 0.25$
  - $= 13.25_{10}$
- Result:
  - $15.2_8 = 13.25_{10}$

# Decimal-to-Octal Conversion

- **Integer Conversion (Repeated Division by 8)**
  - Method: **Divide** by 8, **track** remainders **bottom-up**.
  - Example: Convert $266_{10}$ → Octal

| Division | Quotient | Remainder |
|---|---|---|
| 266 ÷ 8 | 33 | **2** (LSB) |
| 33 ÷ 8 | 4 | **1** |
| 4 ÷ 8 | 0 | **4** (MSB) |

  - **Remainders** in **reverse** order: 4 1 2 → $\mathbf{412_8}$

# Decimal-to-Octal Conversion

- **Fraction Conversion (Repeated Multiplication by 8)**
  - Method: **Multiply** by 8, **track** carries (integer parts) **top-down**.
  - Example: Convert $\mathbf{0.23_{10}}$ → Octal (**3 places**)

| Multiplication | Product | Carry (Integer) | Fraction |
|---|---|---|---|
| 0.23 × 8 | 1.84 | **1** (MSD) | 0.84 |
| 0.84 × 8 | 6.72 | **6** | 0.72 |
| 0.72 × 8 | 5.76 | **5** (LSD) | 0.76 |

  - **Carries** in **order**: 1 6 5 → $\mathbf{0.165_8}$
  - The process is terminated after **three places**; if more accuracy were required, we continue multiplying to obtain more octal digit.

# Octal-to-Binary Conversion

- **Key Advantage:**
  - The primary advantage of the octal number system is the ease with which conversion can be made between binary and octal numbers.
  - Each octal digit maps directly to a 3-bit binary group, simplifying conversions.
  - Octal-Binary Reference Table

| Octal Digit | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| 3-Bit Binary | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |

# Octal-to-Binary Conversion

- **Integer Conversion**
- **Convert $472_8$ → Binary**
  - Step-by-Step:
    - **Split** each octal digit: **4 | 7 | 2**
    - **Map** to 3-bit binary using the table:
      - **4 → 100**
      - **7 → 111**
      - **2 → 010**
    - **Combine** results: **100 111 010 → $100111010_2$**
  - Visual Guide:
  - Result:
    - **$472_8 = 100111010_2$**

| Octal Digit | 4 | 7 | 2 |
|---|---|---|---|
| Binary | 100 | 111 | 010 |

# Octal-to-Binary Conversion

- **Fractional Conversion**
- **Convert $34.562_8$ → Binary**
  - Step-by-Step:
    - **Split integer and fractional parts**: 3 4 • 5 6 2
    - **Map** each to 3-bit binary using the table:
    - 3 → 011 | 4 → 100 | 5 → 101 | 6 → 110 | 2 → 010
    - **Combine** results (preserve the octal point): 011 100 • 101 110 010 → $011100.101110010_2$
  - Visual Guide:

| Octal Part | 3 | 4 | • | 5 | 6 | 2 |
|---|---|---|---|---|---|---|
| Binary | 011 | 100 | • | 101 | 110 | 010 |

  - Result:
    - $34.562_8 = 011100.101110010_2$
    - **(Leading zero can be omitted: $11100.10111001_2$)**

# Binary-to-Octal Conversion

- **Conversion Rules:**
  - Group binary digits into **sets of 3** (right → left for integers, left → right for fractions)
  - Add leading/trailing zeros if needed to complete groups
  - Convert each 3-bit group to octal using reference table
- **Example: Simple Conversion**
  - **Convert $100111010_2$ → Octal**
  - **Steps**:
    - **Group from right (LSB): 100 | 111 | 010**
    - **Convert each group:**
      - **100 → 4**
      - **111 → 7**
      - **010 → 2**
  - **Result**:
    - **$100111010_2 = 472_8$**

# Binary-to-Octal Conversion

- **Padding with Leading/Trailing Zeros**
- **Key Rule:**
  - For integer parts (left of point):
    - Add zeros to the left of the MSB (most significant bit)
  - For fractional parts (right of point):
    - Add zeros to the right of the LSB (least significant bit)
- **Visual Examples:**
  - **Integer Padding (Left Side)**
    - **Original: 1 1 0 1 0 1 1 0 ← 8 bits (not divisible by 3)**
    - **Padded: 0 1 1 | 0 1 0 | 1 1 0 ← Added 1 leading zero**
    - **3 2 6 ← Octal equivalent**
  - **Fractional Padding (Right Side)**
    - **Original: .1 0 1 1 ← 4 bits (needs 2 more for 3-bit groups)**
    - **Padded: .1 0 1 | 1 0 0 ← Added 2 trailing zeros**
    - **5 4 ← Octal equivalent**

- **Step-by-Step Padding Guide**

| Binary Number | Action | Padded Version |
|---|---|---|
| 11010110 | Add 1 leading zero | 011 010 110 |
| 101.01101 | Integer: Leave as-is<br>+ 1 trailing zero (fraction) | 101 • 011 010 |
| .1101 | Add 2 trailing zeros | .110 100 |

# Binary-to-Octal Conversion

- **Example: With Padding Zeros**
- **Convert $11010110_2$ → Octal**
  - **Steps:**
    - Add **1 leading zero** to make complete groups:
      - **011 | 010 | 110**
    - Convert each group:
      - **011 → 3**
      - **010 → 2**
      - **110 → 6**
  - **Visual Guide:**

    **Original**: **1 1 0 1 0 1 1 0**

    **Padded**: **0 1 1 | 0 1 0 | 1 1 0**

    **3 2 6**
  - **Result:**
    - **$11010110_2 = 326_8$**

- **Example: Fractional Binary**
- **Convert $1011.01101_2$ → Octal**
  - **Steps:**
    - **Integer part (pad left):**
      - **Original: 1 0 1 1 → Needs 2 leading zeros**
      - **Padded: 001 | 011**
      - **Convert: 1 3**
    - **Fraction part (pad right):**
      - **Original: .01101 → Needs 1 trailing zero**
      - **Padded: .011 | 010**
      - **Convert: 3 2**
  - **Visual Guide:**
    - **Original: 1 0 1 1 • 0 1 1 0 1**
    - **Padded: 0 0 1 | 0 1 1 • 0 1 1 | 0 1 0**
    - **1 3 • 3 2**
  - **Result:**
    - **$1011.01101_2 = 13.32_8$**

# Hexadecimal Number System (Base-16)

- Key Advantages:
  - Compact representation of binary data (1 hex digit = 4 binary bits)
  - Direct mapping to memory addresses and machine code
  - Universal standard in web design (CSS colors), assembly language, and debugging
- Why Hexadecimal
  - Compact Representation:
    - **1 byte (8 bits) = 2 hex digits**
      - **Example: $11010011_2 = D3_{16}$**
    - **Easy Binary Conversion:**
      - **Binary: 1101 0011 → Split into nibbles**
      - **Hex: D 3 → $D3_{16}$**

| Hex | Decimal | Binary |
|---|---|---|
| 0 | 0 | 0000 |
| 1 | 1 | 0001 |
| 2 | 2 | 0010 |
| 3 | 3 | 0011 |
| 4 | 4 | 0100 |
| 5 | 5 | 0101 |
| 6 | 6 | 0110 |
| 7 | 7 | 0111 |
| 8 | 8 | 1000 |
| 9 | 9 | 1001 |
| A | 10 | 1010 |
| B | 11 | 1011 |
| C | 12 | 1100 |
| D | 13 | 1101 |
| E | 14 | 1110 |
| F | 15 | 1111 |

# Hexadecimal-to-Decimal Conversion

- **Key Principle:**
  - Every hex digit's value depends on its positional weight (powers of 16)
- **Method:**
  - **Multiply** each hex digit by its positional weight (powers of 16)
- **Formula:**
  - Decimal = $d_n \times 16^n + \ldots + d_1 \times 16^1 + d_0 \times 16^0$
  - (Where d = hex digit value, n = position from right starting at 0)
- **Example 1:**
  - **Convert $356_{16}$ to decimal**
  - 3 5 6
  - | | └ $6 \times 16^0 = 6 \times 1 = 6$
  - | └── $5 \times 16^1 = 5 \times 16 = 80$
  - └──── $3 \times 16^2 = 3 \times 256 = 768$
  - **Total = 768 + 80 + 6 = $854_{10}$**

- **Example 2:**
  - **Convert $2AF_{16}$ to decimal**
  - **2 A F**
  - **| | └ $15 \times 16^0 = 15 \times 1 = 15$**
  - **| └── $10 \times 16^1 = 10 \times 16 = 160$**
  - **└──── $2 \times 16^2 = 2 \times 256 = 512$**
  - **Total = 512 + 160 + 15 = $687_{10}$**

**TRY YOUSELF! – CLASS ACTIVITY-**

**Convert $2AC_{16}$ to decimal**

# Decimal-to-Hexadecimal Conversion

- **Steps:**
  - **Divide** by 16, track remainders (convert ≥10 to A-F).
  - Read remainders bottom-up.
- **Example 1:**
  - **Convert $423_{10}$ to hex**

| Division | Quotient | Remainder | Hex Digit |
|---|---|---|---|
| 423 ÷ 16 | 26 | 7 | 7 |
| 26 ÷ 16 | 1 | 10 | A |
| 1 ÷ 16 | 0 | 1 | 1 |

  - **Result:** Read ↑ → $1A7_{16}$

- **Example 2:**
  - **Convert $214_{10}$ to hex**
  - **214 ÷ 16 = 13 R6 (6)**
  - **13 ÷ 16 = 0 R13 (D)**
  - **Result:** ↑ Read up → $D6_{16}$

**TRY YOUSELF! – CLASS ACTIVITY-**

**Convert $1024_{10}$ to Hexadecimal**

# Hexadecimal-to-Binary Conversion

- **Direct 4-Bit Grouping Method**
- **Key Rule:**
  - ▸ Each hexadecimal digit converts to exactly 4 binary bits (nibble)
  - ▸ Works for both integers and fractions
- **Conversion Steps:**
  - Separate each hex digit
  - Convert to 4-bit binary using the reference table
  - Combine all binary groups

- Example: Convert $9F2_{16}$ to Binary
  - 9 F 2
  - ↓ ↓ ↓
  - 1001 1111 0010
  - Result: $9F2_{16} = 100111110010_2$

  ---

- Example: Convert $A3.C5_{16}$ to Binary
  - A 3 . C 5
  - ↓ ↓ ↓ ↓
  - 1010 0011 . 1100 0101
  - Result: $A3.C5_{16} = 10100011.11000101_2$

# Binary-to-Hexadecimal Conversion

- 4-Bit Grouping Method (Reverse of Hex-to-Binary)
- Steps:
  - Group binary digits into 4-bit nibbles (start from right for integers, left for fractions)
  - Pad with leading/trailing zeros if needed
  - Convert each group to its hex equivalent

- **Example: Convert $101110100110_2$ to hex**
  - Binary: 1011 1010 0110
  - ↓ ↓ ↓
  - Hex: B A 6
  - **Result: $101110100110_2 = BA6_{16}$**

# Hexadecimal-to-Octal Conversion

- **Example: Convert $9F_{16}$ to octal**
  - Step 1: Hex → Binary (4-bit groups)
    - 9 F → 1001 1111
  - Step 2: Binary → Octal (3-bit groups, pad leading zeros)
    - Original: 1001 1111
    - Pad left: 010 011 111
    - Regroup: 010 011 111
  - Step 3: Binary → Octal
    - 010 = 2, 011 = 3, 111 = 7
  - **Result: $237_8$**

# Review Questions: Number System Conversions

1. Binary Conversion
   - Find the binary equivalent of decimal 363. Then, convert the resulting binary number to octal.
2. Maximum 8-Bit Value
   - What is the largest decimal number that can be represented using 8 bits in binary?
3. Binary-to-Decimal
   - Convert the binary number $1101011_2$ to its decimal equivalent.
4. Binary Counting Sequence
   - Determine the next binary number in the sequence after $10111_2$.
5. MSB Weight Calculation
   - Calculate the positional weight (decimal value) of the Most Significant Bit (MSB) in a 16-bit binary number.
6. Octal-to-Decimal
   - Convert the octal number $614_8$ to decimal.
7. Decimal-to-Octal
   - Convert the decimal number $146_{10}$ to octal.
8. Hexadecimal-to-Decimal
   - Convert the hexadecimal number $24CE_{16}$ to decimal.
9. Decimal-to-Hex-to-Binary
   - First, convert the decimal number $3117_{10}$ to hexadecimal. Then, convert the resulting hexadecimal number to binary.
10. Binary-to-Decimal (Fractional)
    - Solve for x in the equation: $1011.11_2 = x_{10}$
11. Octal-to-Decimal (Fractional)
    - Solve for x in the equation: $174.3_8 = x_{10}$
12. Decimal-to-Binary (Fractional)
    - Solve for x in the equation: $10949.8125_{10} = x_2$
13. Hexadecimal-to-Binary (Fractional)
    - Solve for x in the equation: $2C6B.F2_{16} = x_2$

# Binary Arithmetic Fundamentals

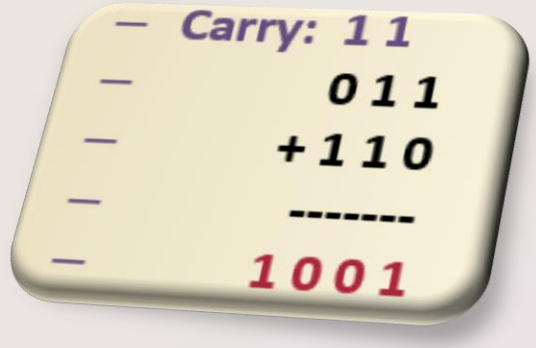
- Key Definitions
  - Binary Addition
    - Definition: Bitwise operation following four rules with possible carry propagation.
    - Carry: Overflow when the sum of bits exceeds 1 (similar to decimal "carrying the 1").
  - Binary Subtraction (Direct Method)
    - Definition: Bitwise operation requiring borrowing from higher positions when subtracting a larger digit from a smaller one.
  - 2’s Complement Method
    - 1’s Complement: Inverting all bits of a number (1→0, 0→1).
    - 2’s Complement: Adding 1 to the 1’s complement to represent negative numbers.
    - Purpose: Simplifies subtraction by converting it to addition (used in CPUs).

# Binary Addition

- **Rules Table**

| A + B | Sum | Carry |
|---|---|---|
| 0 + 0 | 0 | 0 |
| 0 + 1 | 1 | 0 |
| 1 + 1 | 0 | 1 |
| 1+1+1 | 1 | 1 |

- **Example: $011_2$ (3) + $110_2$ (6)**
  - 011
  - + 110
  - -----
  - 1001 (9)
  - **Step-by-Step:**
    - LSB (Right): 1 + 0 = 1
    - Middle: 1 + 1 = 0 (carry 1)
    - MSB (Left): 0 + 1 + 1 (carry) = 0 (carry 1)
    - Final Carry: Leading 1 → $1001_2$

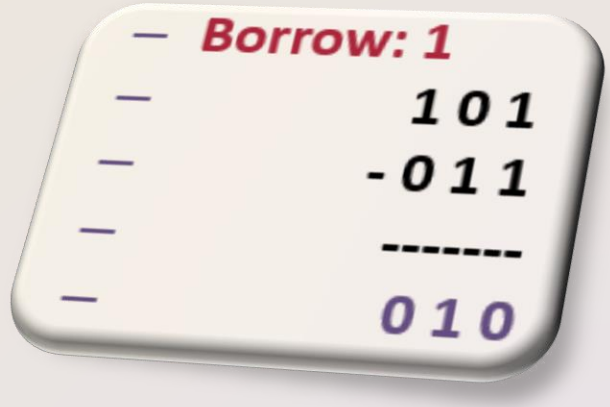


# Binary Subtraction (Direct Method)

- **Rules Table**

| A – B | Result | Borrow |
|---|---|---|
| 0 – 0 | 0 | 0 |
| 1 – 0 | 1 | 0 |
| 1 – 1 | 0 | 0 |
| 0 – 1 | 1 | 1 |

- **Example: $101_2$ (5) – $011_2$ (3)**
  - 101
  - - 011
  - -----
  - 010 (2)
  - Step-by-Step:
    - LSB: 1 – 1 = 0
    - Middle: 0 – 1 → Borrow 1 → 10 – 1 = 1
    - MSB: (0 after borrow) – 0 = 0
    - Final result: 010



# 1’s and 2’s Complement Conversion

- **Definitions**
  - **1’s Complement:**
    - Definition: Flip all bits of a binary number (0→1, 1→0).
    - Example: 0101 → 1010.
  - **2’s Complement:**
    - Definition: 1’s complement + 1. Represents negative numbers.
    - Example: 0101 → 1010 (1’s) → 1011 (2’s).
- **Key Advantages of 2’s Complement:**
  - Simplifies hardware design (uses same circuits for + and -)
  - Eliminates separate subtraction logic
  - Represents negative numbers in binary
  - The leftmost bit (MSB) is the sign bit (0 = positive, 1 = negative).

- **Example: Convert $1011\ 0101_2$ (Original)**

| Step | Binary Value | Visual Representation |
|---|---|---|
| Original Number | 1011 0101 | 1 0 1 1 0 1 0 1 |
| 1’s Complement | 0100 1010 | 0 1 0 0 1 0 1 0 |
| 2’s Complement | 0100 1010 + 1 = 0100 1011 | 0 1 0 0 1 0 1 1 |

# 1’s and 2’s Complement Conversion

- **Why Are They Used?**

| Feature | 1’s Complement | 2’s Complement |
|---|---|---|
| Negative Rep. | Two zeros (0000 and 1111) | Single zero (0000) |
| Hardware | Requires extra step for subtraction | Simplifies arithmetic (no carry logic) |
| Overflow | Detected by end-around carry | Detected by sign-bit mismatch |

# 1's Complement Examples

- **Example 1: Representing Negative Numbers**
  - **Number: -3 in 4-bit binary**
    - **Step 1: Write +3 → 0011**
    - **Step 2: Flip all bits → 1100 (This is -3 in 1's complement).**

    ---------- ---------- ---------- ----------

    ---------- ---------- ---------- ----------
- **Example 2: Subtraction Using 1's Complement**
  - **Calculate: 5 – 3**
    - **Step 1: Represent -3 → 1100 (1's complement of 0011).**
    - **Step 2: Add to 5 (0101):**
      - 0101 (5)
      - + 1100 (-3 in 1's)
      - -------
      - 10001
    - **Step 3: End-around carry → Add the overflow 1 back:**
      - 0001 (from 10001)
      - + 1 (carry)
      - -------
      - 0010 (2) → Correct!
    - **Problem: Without the extra carry step, you'd get 0001 (wrong!).**

# 2's Complement Examples

- **Example 1: Representing Negative Numbers**
  - **Number: -3 in 4-bit binary**
    - **Step 1: Write +3 → 0011**
    - **Step 2: Flip all bits → 1100 (This is -3 in 1's complement).**
    - **Step 3: Add 1 → 1101 (This is -3 in 2's complement).**

    ---------- ---------- ---------- ----------

    ---------- ---------- ---------- ----------
- **Example 2: Subtraction Using 2's Complement**
  - **Calculate: 5 – 3**
    - **Step 1: Represent -3 → 1101 (2's complement of 0011).**
    - **Step 2: Add to 5 (0101):**
  - **0101 (5)**
  - **+ 1101 (-3 in 2's)**
  - **-------**
  - **10010 → Discard overflow → `0010` (2) → Correct!**
- **No extra steps needed**

# Key Differences in Examples

| Operation | 1's Complement | 2's Complement |
| --- | --- | --- |
| Represent -3 | 1100 (flipped bits) | 1101 (flipped bits + 1) |
| Calculate 5-3 | Needs end-around carry (10001 → 0010) | Simple addition (10010 → discard → 0010) |

# Homework!

1. Convert -6 to 8-bit 2's complement.
2. Subtract 7 - 4 using 2's complement.

# THANK YOU ☺