Class 9 Computer Science Chapter 1: Introduction to Computational Systems Notes

Posted in **9 Computer Science**

These lecture notes are prepared by **Salman Ahmad**, who is both a **teacher** and a **web developer**. The purpose of this material is to make the topic clear and useful for students of Class 9 Computer Science.

Table of Contents

What is a System?

A **system** is a group of **connected parts that work together to perform a specific task or achieve a specific purpose**.
Example: A laptop is a computer system. Its **processor, RAM, storage, operating system, applications, and other components** work together to allow us to perform different tasks.

Information system

It is a **combination of people, technology, data, and procedures that work together to collect and turn data into useful information**.

Example

A **school information system** can collect student information such as names, marks, attendance, and fees. It processes this data and provides useful information such as **result cards, attendance reports, and student records**.

Objective of a System

The **objective of a system** is the **main purpose or goal that the system is designed to achieve**. A system is created to perform specific tasks and produce useful results.

Examples of System Objectives

Different systems are designed for different purposes. For example:

- **School Information System** → To manage student records, marks, attendance, and fees.
- **Banking System** → To manage customer accounts, deposits, withdrawals, and transactions.
- **Hospital Management System** → To manage patient records, appointments, treatments, and billing.
- **Library Management System** → To manage books, members, borrowing, and returning of books.
- **Online Shopping System** → To allow customers to search, order, and pay for products online.

Components of a System

The **components of a system** are the different parts that work together to achieve the system's objective. A system normally has the following main components:

1. **Input** → The data or resources entered into the system.
    - **Example:** Student names and marks entered into a school system.
2. **Processing** → The activities performed on the input to produce a useful result.
    - **Example:** Calculating a student's total and percentage.
3. **Output** → The useful information or result produced by the system.
    - **Example:** A student's result card.
4. **Feedback** → Information about the output that helps improve or control the system.
    - **Example:** A teacher checks the result and corrects an incorrect mark.

System Environment

The **system environment** includes everything **outside the system that interacts with or affects it**. These external factors can influence **how the system works and operates**.

Simple Example: Online Shopping System

An **online shopping system** interacts with:

- **Customers** – Buy products.
- **Delivery services** – Deliver orders.
- **Banks** – Process payments.
- **Internet** – Allows users to access the system.

> **Note:** These things are **outside the system but interact with it**, so they are part of its **environment**.

Communication also called System Communication

Communication between the different parts of a system is important for the system to work properly. It allows the components to **share information and work together smoothly** to achieve the system's goal.

Examples

- **Computing System:** The **CPU communicates with memory** to get and store data.
- **Biological System:** The **brain sends signals to muscles** to make the body move.

Software

Software is a set of **programs and instructions** that tells a computer **what to do and how to perform different tasks**. Unlike hardware, software cannot be physically touched.

Types of Software

There are two main types of software:

1. System Software

System software controls and manages the computer's hardware and provides a platform for other software to run.

Examples of System Software

- **Operating System (OS)** – Manages the computer's hardware and provides an interface for users and applications.
  Examples: Windows, Linux, macOS.
- **Device Drivers (DD)** – Help the operating system communicate with and control hardware devices.
  Examples: Printer driver, graphics driver, keyboard driver.
- **Utility Programs (UP)** – Help maintain, protect, and manage the computer.
  Examples: Disk Cleanup, antivirus software, backup tools.

2. Application Software

Application software is designed to help users **perform specific tasks**.

Examples of Application Software

- **Microsoft Word** → Writing documents
- **Microsoft Excel** → Working with data and calculations
- **Web Browsers** → Browsing the internet
- **Media Players** → Playing audio and video
- **Graphic Design Software** → Creating and editing images

> **Note: System software** manages the computer and its hardware, while **application software** helps users perform specific tasks.

System Software vs Application Software

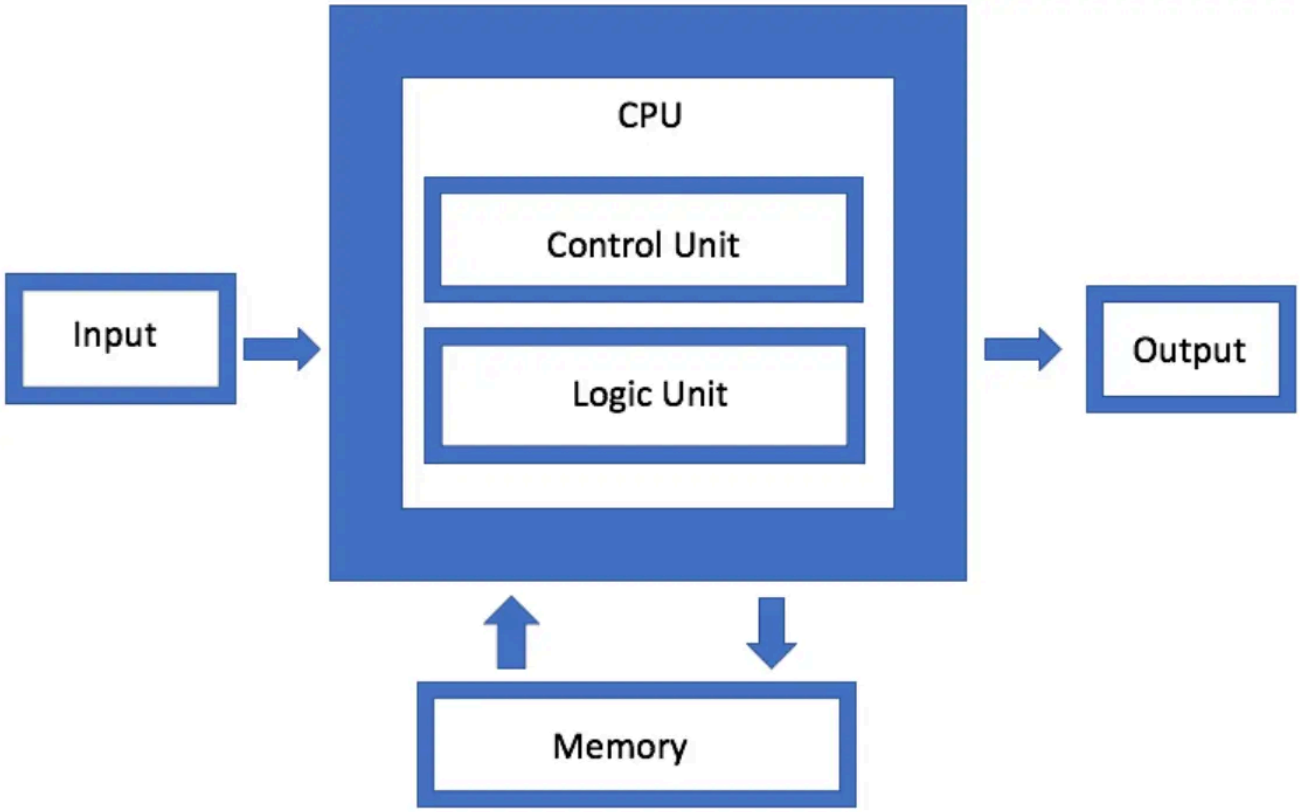
System Software	Application Software
It manages and controls the **computer's hardware**.	It helps users **perform specific tasks**.
It provides a platform for application software to run.	It runs on the system software.
It usually works in the **background**.	Users directly interact with it.
It is generally **pre-installed** with the computer or installed as part of the operating system.	It usually needs to be **installed by the user** according to their needs.
It is necessary for the basic operation of a computer.	It is used for specific user tasks.
Examples: Operating System, Device Drivers, Utility Programs.	**Examples:** MS Word, Excel, Web Browsers, Media Players.

The Architecture of von Neumann Computers

Introduction

The **von Neumann architecture** is a basic design or model of a computer. It explains how the main parts of a computer **work together to process data and instructions**. This model was developed in the **1940s** and is named after **John von Neumann**, a mathematician and physicist who contributed to its development.

Diagram or Struture



Main Components

A von Neumann computer mainly consists of:

- **Memory**
- **Central Processing Unit (CPU)**
- **Input Devices**
- **Output Devices**
- **System Bus**

1. Memory

Memory stores the **data and instructions** that the CPU needs to perform tasks. For example, when you open a program on your computer, it is loaded from the storage device into **RAM (Random Access Memory)**. The CPU can access the program and its data from RAM quickly, allowing the program to run efficiently.

2. Central Processing Unit (CPU)

The **CPU** is the main processing part of the computer. It **processes data and executes instructions** stored in memory.
The CPU has two important components:

Arithmetic Logic Unit (ALU)

The **ALU** performs:

- Mathematical calculations, such as **addition and subtraction**.
- Logical operations, such as **comparing values**.

Example: When you calculate **2 + 2** in a calculator application, the ALU performs the calculation.

Control Unit (CU)

The **Control Unit** controls and coordinates the activities of the computer. It tells other components **what to do and when to do it** according to the instructions of the program.

Example: When calculating **2 + 2**, the CU controls the process and makes sure the required data is obtained from memory and the ALU performs the calculation.

> **Note: ALU** → Performs calculations and logical operations.
> **CU** → Controls and coordinates the activities of the CPU.

3. Input Devices

Input devices allow users to **enter data and instructions into the computer**.

Examples include:

- **Keyboard**
- **Mouse**
- **Microphone**
- **Scanner**

Example: When you type your name using a keyboard, the keyboard sends the input to the computer for processing.

4. Output Devices

Output devices show or provide the **results produced by the computer**.

Examples include:

- **Monitor**
- **Printer**
- **Speakers**

Example: When the computer finishes processing some information, the result can be displayed on the monitor.

5. System Bus

A **system bus** is a communication pathway that allows the different components of a computer to **exchange data, addresses, and control signals**.

The system bus has three main parts:

Data Bus

The **Data Bus** carries the **actual data** between the CPU, memory, and other components.

Address Bus

The **Address Bus** carries information about **where the data should be sent or where it should be retrieved from**.

Control Bus

The **Control Bus** carries **control signals** that tell different components what actions to perform.

> **Note:** The **system bus** acts like a **communication pathway** that allows different parts of the computer to communicate with each other.

Working of von Neumann Architecture

The **von Neumann architecture** executes instructions using a step-by-step process called the **Fetch–Decode–Execute–Store cycle**.
The CPU mainly performs **four stages**:

1. **Fetching** → Gets the instruction from memory.
2. **Decoding** → Understands what the instruction means.
3. **Execution** → Performs the required operation.
4. **Storing** → Stores or displays the result.

Example: Adding Two Numbers

Suppose we use a calculator to calculate **2 + 2**. The computer follows the four stages to produce the answer.

1. Fetching

Fetching means getting the next instruction from **memory**.
The **Program Counter (PC)** contains the address of the next instruction. The CPU uses this address to get the instruction from memory and places it in the **Instruction Register (IR)**.

2. Decoding

Decoding means **understanding the instruction** that was fetched.

The **Control Unit (CU)** examines the instruction and determines what operation needs to be performed.

For example, the CPU understands that an **addition** operation is required.

3. Execution

Execution means **performing the required operation**.

The **Arithmetic Logic Unit (ALU)** performs mathematical and logical operations.

For example:

2 + 2 = 4

The ALU performs the addition and produces the result **4**.

4. Storing

After execution, the result is **stored in memory or sent to an output device**.

For example, the result **4** can be displayed on the calculator screen.

Complete Process

Fetch → Decode → Execute → Store

Stage	What Happens?
Fetch	Get the instruction from memory
Decode	Understand the instruction
Execute	Perform the required operation
Store	Store or display the result

Example: 2 + 2

Fetch: Get the addition instruction

↓

Decode: Understand that addition is required

↓

Execute: ALU calculates **2 + 2 = 4**

↓

Store: Result **4** is stored or displayed

Number System

A **number system** is a method of **representing numbers using a specific set of digits or symbols**. Different number systems use different numbers of digits, called the **base** or **radix**.

In computer science, the following four number systems are commonly used:

1. Decimal Number System

The **Decimal Number System** has a **base of 10**. It uses **10 digits from 0 to 9**.

This is the number system that we normally use in our daily life.

Example: 25, 100, 458

2. Binary Number System

The **Binary Number System** has a **base of 2**. It uses only **two digits: 0 and 1**.

Computers use binary because electronic circuits can represent two basic states, such as **ON and OFF**. **0 means OFF** and **1 means ON**.

Example: 1010, 1101, 10001

3. Octal Number System

The **Octal Number System** has a **base of 8**. It uses **eight digits from 0 to 7**.

Example: 25, 147, 706

4. Hexadecimal Number System

The **Hexadecimal Number System** has a **base of 16**. It uses **16 symbols: 0–9 and A–F**.

Here:

- **A = 10**
- **B = 11**
- **C = 12**
- **D = 13**
- **E = 14**
- **F = 15**

Hexadecimal is used in areas such as **memory addresses, color codes, MAC addresses, and IPv6 addresses**.

Example: 2A, FF, 1B

> **Note: 1 octal digit is equal to 3 binary bits**, while **1 hexadecimal digit is equal to 4 binary bits**.
> For example:
> - **Octal:** 0 = **000**, 1 = **001**, 2 = **010**, 3 = **011**
> - **Hexadecimal:** 0 = **0000**, 1 = **0001**, 2 = **0010**, 3 = **0011**
>
> This makes it easier to convert binary numbers into **octal and hexadecimal** forms.

Decimal to Binary Conversion and Vice Versa

To convert a decimal number into binary, we repeatedly **divide the number by 2** and record the remainders. The binary answer is obtained by reading the remainders **from bottom to top**.

Example: Convert 8_{10} to Binary

Divide **8 by 2** repeatedly and write the remainders on the right:

```
2  |  8
2  |  4 - 0
2  |  2 - 0
2  |  1 - 0
```

Now read the remainders from **bottom to top**:
Therefore: $\mathbf{8_{10} = 1000_2}$

Binary to Decimal

Now convert $\mathbf{1000_2}$ back to decimal:

```
1000₂

= (1 × 2³) + (0 × 2²) + (0 × 2¹) + (0 × 2⁰)
```

```
= 8 + 0 + 0 + 0

= 8₁₀
```

Therefore:
$1000_2 = 8_{10}$ ✅

> **Note:** When converting **decimal to binary**, divide by **2** repeatedly and read the remainders **from bottom to top**.

Decimal to Binary Conversion Practice Questions

Basic Questions

1. $5_{10} = ?_2$
2. $8_{10} = ?_2$
3. $10_{10} = ?_2$
4. $15_{10} = ?_2$
5. $20_{10} = ?_2$
6. $25_{10} = ?_2$

4-Digit Decimal Numbers

7. $1024_{10} = ?_2$
8. $1200_{10} = ?_2$
9. $1260_{10} = ?_2$
10. $1500_{10} = ?_2$
11. $2000_{10} = ?_2$
12. $4095_{10} = ?_2$

Challenge Questions

13. $345_{10} = ?_2$
14. $512_{10} = ?_2$
15. $999_{10} = ?_2$
16. $2048_{10} = ?_2$

> **Note:** You can do **1 to 2 examples** from each section. The extra examples are added only for **better understanding and extra practice**.

Octal to Binary and Binary to Octal

The **Octal Number System** has a base of **8** and uses the digits **0 to 7**.
The **Binary Number System** has a base of **2** and uses only **0 and 1**.
The conversion between octal and binary is easy because:

> **Note: 1 octal digit = 3 binary bits.**

To convert an octal number into binary, **replace each octal digit with its 3-bit binary equivalent**.

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

Example: Convert 5_8 to Binary

From the table:

$5_8 = 101_2$

```
2  |  5
2  |  2 - 1
   |  1 - 0
```

Example: Convert 25_8 to Binary

Convert each octal digit separately:

```
2 → 010
5 → 101
```

Therefore:

$25_8 = 010101_2$

Leading zeros can be removed:

$25_8 = 10101_2$

```
2  |  2
2  |  1 - 0
```

But because **1 octal digit = 3 binary bits**, we add a **leading zero**:

$2_8 = 010_2$

> **Note:** The extra zero is added **only to make the binary representation 3 bits**, because each octal digit corresponds to exactly **3 binary bits**.

```
2  |  5
2  |  2 - 1
   |  1 - 0
```

Convert 010101_2 to Octal

$$
\begin{aligned}
&010_2 \\
&= (0 \times 2^2) + (1 \times 2^1) + (0 \times 2^0) \\
&= (0 \times 4) + (1 \times 2) + (0 \times 1) \\
&= 0 + 2 + 0 \\
&= 2_8
\end{aligned}
$$

So:

$010_2 = 2_{10}$

Since **2** is already an octal digit:

$010_2 = 2_8$ ✅

> **Note:** For binary-to-octal conversion, remember that **1 octal digit = 3 binary bits**. Therefore, **010_2** represents the single octal digit **2_8**.

$$
\begin{aligned}
&101_2 \\
&= (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) \\
&= (1 \times 4) + (0 \times 2) + (1 \times 1) \\
&= 4 + 0 + 1 \\
&= 5_8
\end{aligned}
$$

Since **5** is already an octal digit:

$101_2 = 5_8$ ✅

Therefore: **$010101_2 = 25_8$** ✅

Practice Questions: Octal ↔ Binary

A. Convert Octal to Binary

Convert the following **octal numbers into binary**:

1. $3_8 = ?_2$
2. $5_8 = ?_2$
3. $7_8 = ?_2$
4. $12_8 = ?_2$
5. $25_8 = ?_2$
6. $34_8 = ?_2$
7. $47_8 = ?_2$
8. $105_8 = ?_2$
9. $246_8 = ?_2$
10. $725_8 = ?_2$

B. Convert Binary to Octal

Convert the following **binary numbers into octal**:

1. $011_2 = ?_8$
2. $101_2 = ?_8$
3. $110_2 = ?_8$
4. $010101_2 = ?_8$
5. $011100_2 = ?_8$
6. $100111_2 = ?_8$
7. $101010_2 = ?_8$
8. $001101011_2 = ?_8$
9. $010100110_2 = ?_8$
10. $111101101_2 = ?_8$

More Practice

11. $11_2 = ?_8$
12. $1001_2 = ?_8$
13. $10110_2 = ?_8$
14. $110101_2 = ?_8$
15. $1000111_2 = ?_8$
16. $10100110_2 = ?_8$
17. $111010011_2 = ?_8$

> **Note:** Always start grouping the binary digits **from the right**. If the first group has only **1 or 2 bits**, add leading zeros to make a group of **3 bits**.

Examples

1101_2 → 1 | 101 → 001 | 101

10101_2 → 10 | 101 → 010 | 101

Hexadecimal to Binary and Binary to Hexadecimal

The **Hexadecimal Number System** has a base of **16** and uses the digits **0–9** and the letters **A–F**.
The **Binary Number System** has a base of **2** and uses only **0 and 1**.
The conversion between hexadecimal and binary is easy because:

> **Note: 1 hexadecimal digit = 4 binary bits**. Therefore, each hexadecimal digit can be directly replaced by its **4-bit binary equivalent**.

Hexadecimal to Binary

To convert a hexadecimal number into binary, **replace each hexadecimal digit with its 4-bit binary equivalent**.

Hexadecimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101

Hexadecimal	Binary
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Example: Convert 5_{16} to Binary

From the conversion table:

$5_{16} = 0101_2$

We can also verify this by converting 5 into decimal and then binary:

```
2  |  5
2  |  2 - 1
   |  1 - 0
```

Reading the remainders from bottom to top:

$5_{16} = 101_2$

But because **1 hexadecimal digit = 4 binary bits**, we add a leading zero:

$5_{16} = 0101_2$

> **Note:** The extra leading zero is added **only to make the binary representation 4 bits**, because each hexadecimal digit corresponds to exactly **4 binary bits**.

Example: Convert $2A_{16}$ to Binary

Convert each hexadecimal digit separately:

```
2 → 0010
A → 1010
```

Therefore:

$2A_{16} = 00101010_2$

Leading zeros can be removed when they are not required:

$2A_{16} = 101010_2$

Binary to Hexadecimal

To convert a binary number into hexadecimal, **group the binary digits into groups of 4 bits from the right**. Then replace each 4-bit group with its hexadecimal equivalent.

> **Note:** Always start grouping the binary digits **from the right**. If the first group has only **1, 2, or 3 bits**, add leading zeros to make a group of **4 bits**.

Convert 1010_2 to Hexadecimal

$$1010_2$$

$$= (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (0 \times 2^0)$$

$$= (1 \times 8) + (0 \times 4) + (1 \times 2) + (0 \times 1)$$

$$= 8 + 0 + 2 + 0$$

$$= 10_{16}$$

Since **10 in hexadecimal is represented by A**:

$1010_2 = A_{16}$ ✅

Convert 0010_2 to Hexadecimal

$$0010_2$$

$$= (0 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (0 \times 2^0)$$

$$= (0 \times 8) + (0 \times 4) + (1 \times 2) + (0 \times 1)$$

$$= 0 + 0 + 2 + 0$$

$$= 2_{16}$$

$0010_2 = 2_{16}$ ✅

Convert 00101010_2 to Hexadecimal

First, group the binary digits into groups of 4 from the right:

$$00101010_2$$

$$0010 \mid 1010$$

Now convert each group:

$$0010 \rightarrow 2$$
$$1010 \rightarrow A$$

Therefore:

$00101010_2 = 2A_{16}$ ✅

> **Note:** For binary-to-hexadecimal conversion, **1 hexadecimal digit = 4 binary bits**. Therefore, every group of 4 binary bits represents exactly one hexadecimal digit.

More Grouping Examples

1101_2 → 1101 → D

Therefore: **$1101_2 = D_{16}$**

10101_2 → 1 | 0101 → 0001 | 0101

Therefore: **$10101_2 = 15_{16}$**

110101_2 → 1101 | 0101

Therefore: **$110101_2 = D5_{16}$**

Practice Questions: Hexadecimal ↔ Binary

A. Convert Hexadecimal to Binary

Convert the following **hexadecimal numbers into binary**:

1. **$3_{16} = ?_2$**
2. **$5_{16} = ?_2$**
3. **$A_{16} = ?_2$**
4. **$F_{16} = ?_2$**
5. **$12_{16} = ?_2$**
6. **$2A_{16} = ?_2$**
7. **$3F_{16} = ?_2$**
8. **$4B_{16} = ?_2$**
9. **$7C_{16} = ?_2$**
10. **$AF_{16} = ?_2$**

B. Convert Binary to Hexadecimal

Convert the following **binary numbers into hexadecimal**:

1. **$0011_2 = ?_{16}$**
2. **$0101_2 = ?_{16}$**
3. **$1010_2 = ?_{16}$**
4. **$1111_2 = ?_{16}$**

5. $00101010_2 = ?_{16}$
6. $00111111_2 = ?_{16}$
7. $01001011_2 = ?_{16}$
8. $01111100_2 = ?_{16}$
9. $10101111_2 = ?_{16}$
10. $11011010_2 = ?_{16}$

More Practice

11. $1101_2 = ?_{16}$
12. $10101_2 = ?_{16}$
13. $110101_2 = ?_{16}$
14. $1000111_2 = ?_{16}$
15. $10100110_2 = ?_{16}$
16. $111010011_2 = ?_{16}$
17. $1011010110_2 = ?_{16}$

> **Note:** Always start grouping the binary digits **from the right**. If the first group has only **1, 2, or 3 bits**, add leading zeros to make a group of **4 bits**.

Examples of Binary Grouping

1101_2 → 1101 → D

10101_2 → 1 | 0101 → 0001 | 0101

110101_2 → 1101 | 0101

1000111_2 → 1 | 0001 | 11 → 0001 | 0001 | 11

Decimal to Octal Conversion and Vice versa

To convert a **decimal number into octal**, divide the decimal number repeatedly by **8**. Write the remainders and read them **from bottom to top**.

Example: Convert 25_{10} to Octal

```
8  |  25
8  |  3 - 1
   |  0 - 3
```

Now read the remainders **from bottom to top**:

3 1

Therefore:

$25_{10} = 31_8$ ✅

> **Note:** Do not add a leading zero unless it is specifically needed to show a fixed number of digits. Therefore, **$25_{10} = 31_8$**, not **031_8**.

Convert 31_8 to Decimal

Multiply each octal digit by its corresponding power of **8**, starting from the right with 8^0:

```
31₈

= (3 × 8¹) + (1 × 8⁰)

= (3 × 8) + (1 × 1)

= 24 + 1

= 25₁₀
```

Therefore:

$31_8 = 25_{10}$ ✅

Practice Questions: Decimal ↔ Octal

A. Convert Decimal to Octal

Convert the following **decimal numbers into octal**:

1. $8_{10} = ?_8$
2. $15_{10} = ?_8$
3. $25_{10} = ?_8$
4. $40_{10} = ?_8$
5. $64_{10} = ?_8$
6. $100_{10} = ?_8$
7. $125_{10} = ?_8$
8. $250_{10} = ?_8$
9. $500_{10} = ?_8$
10. $1000_{10} = ?_8$

B. Convert Octal to Decimal

Convert the following **octal numbers into decimal** using the **multiplication method**:

1. $7_8 = ?_{10}$
2. $12_8 = ?_{10}$
3. $25_8 = ?_{10}$
4. $31_8 = ?_{10}$
5. $45_8 = ?_{10}$
6. $100_8 = ?_{10}$
7. $125_8 = ?_{10}$
8. $250_8 = ?_{10}$
9. $345_8 = ?_{10}$
10. $1000_8 = ?_{10}$

> **Note:** For decimal-to-octal conversion, repeatedly **divide by 8** and read the remainders **from bottom to top**. For octal-to-decimal conversion, multiply each digit by its corresponding power of **8**, starting from the right with 8^0.

Trick to Solve Bigger Questions

Example: Convert 4455_{10} to Octal

First divide **4455** by **8**:

4455 ÷ 8 = 556.875

The whole-number quotient is **556** and the decimal part is **0.875**.

Now find the remainder:

0.875 × 8 = 7

So the remainder is **7**.

Then continue with **556**:

556 ÷ 8 = 69.5

0.5 × 8 = 4

So the remainder is **4**.

Continue:

69 ÷ 8 = 8.625

0.625 × 8 = 5

Remainder = **5**

Continue:

8 ÷ 8 = 1

Remainder = **0**

Finally:

1 ÷ 8 = 0.125

0.125 × 8 = 1

Remainder = **1**

Now read the remainders **from bottom to top**:

1 → 0 → 5 → 4 → 7

Therefore:

$4455_{10} = 10547_8$ ✅

Data Representation in Computing Systems

Computers can **store and process different types of information**. To store this information, computers represent data using **binary digits (0 and 1)**.

One important type of data is **numeric data**, which includes whole numbers and integers.

1. Whole Numbers (W)

Whole numbers are numbers that include **zero and all positive numbers**. They do not include negative numbers or fractions.

Mathematically:

W = {0, 1, 2, 3, 4, ...}

Examples

- Number of students = **50**
- Age = **18**
- Number of books = **25**

These values cannot normally be negative.

Whole Number Storage

A **byte** consists of **8 bits**. The more bytes used, the more values a computer can store.

For **n bits**, the maximum whole number is:

Maximum value = $2^n - 1$

Storage	Bits	Maximum Value
1 Byte	8 bits	$2^8 - 1$ = **255**
2 Bytes	16 bits	$2^{16} - 1$ = **65,535**
4 Bytes	32 bits	$2^{32} - 1$ = **4,294,967,295**

For example, an 8-bit whole number can represent values from:

$00000000_2 = 0_{10}$

to

$11111111_2 = 255_{10}$

2. Integers (Z)

Integers include **positive numbers, negative numbers, and zero**.

Mathematically:

Z = {..., −3, −2, −1, 0, 1, 2, 3, ...}

In computing, these are called **signed integers** because they can represent both positive and negative values.

Sign Bit

To represent positive and negative numbers, one bit is used as the **sign bit**. It is usually the **most significant bit (MSB)**.

- **0** → Positive
- **1** → Negative

> **Note: MSB (Most Significant Bit)** is the leftmost bit in a binary number and usually has the highest place value. **LSB (Least Significant Bit)** is the rightmost bit and has the lowest place value.

Example:

```
Binary Number:  1 0 1 1 0 1 0 1
                ↑             ↑
               MSB           LSB
             (leftmost)    (rightmost)
```

In $\mathbf{10110101_2}$:

- **MSB = 1** → the leftmost bit.
- **LSB = 1** → the rightmost bit.

For an **8-bit signed integer**, one bit is used for the sign, leaving **7 bits** for the value.

The maximum positive value is:

$\mathbf{2^7 - 1 = 127}$

So, the maximum positive value is:

$\mathbf{01111111_2 = 127_{10}}$

Negative values are commonly stored using **2's complement**.

1's Complement

1's complement is a method of representing signed binary numbers.

To find the **1's complement**, simply **invert every bit**:

- **0 → 1**
- **1 → 0**

Example

```
Original:        00000101
1's complement:  11111010
```

So, the 1's complement of $\mathbf{00000101_2}$ is $\mathbf{11111010_2}$.

2's Complement

2's complement is the common method used by computers to represent **negative integers**.

To find the 2's complement:

1. **Invert all the bits** to get the 1's complement.
2. **Add 1** to the result.

Example: Represent −5 in 8 Bits

First, write **+5** in binary:

```
00000101
```

Step 1: Invert all bits

```
11111010
```

Step 2: Add 1

```
  11111010
+ 00000001
-----------
  11111011
```

Therefore:

$-5_{10} = 11111011_2$ in 8-bit 2's complement.

5. Minimum Integer Value

For an **n-bit signed integer**, the minimum value is:

Minimum value = -2^{n-1}

For an **8-bit integer**:

$-2^7 = -128$

Therefore, an 8-bit signed integer can represent values from:

−128 to +127

Examples

Storage	Bits	Minimum Value	Maximum Value
1 Byte	8	**−128**	**127**
2 Bytes	16	**−32,768**	**32,767**
4 Bytes	32	**−2,147,483,648**	**2,147,483,647**

Note: Whole numbers can represent **0 and positive values**, while integers can represent **negative, zero, and positive values**. Computers commonly use **2's complement** to store negative integers.

Binary Arithmetic

Binary arithmetic is the process of performing **addition, subtraction, multiplication, and division** using only **0 and 1**.

1. Binary Addition

Rules

Operation	Result
0 + 0	0
0 + 1	1
1 + 0	1
1 + 1	10
1 + 1 + 1	11

Example 1

```
  101
+ 011
-----
 1000
```

$101_2 + 011_2 = 1000_2$

Example 2

```
  1101
+ 0011
------
 10000
```

$1101_2 + 0011_2 = 10000_2$

Example 3

```
  10110
+ 01101
-------
 100011
```

$10110_2 + 01101_2 = 100011_2$

2. Binary Subtraction

Rules

Operation	Result
0 − 0	0
1 − 0	1
1 − 1	0
10 − 1	1

When **0 − 1** occurs, we borrow from the next position.

Example 1

```
  1010
- 0011
------
  0111
```

$1010_2 - 0011_2 = 0111_2$

Example 2

```
  1101
- 0101
------
  1000
```

$1101_2 - 0101_2 = 1000_2$

Example 3

```
  10000
```

```
- 00111
-------
  01001
```

$10000_2 - 00111_2 = 01001_2$

3. Binary Multiplication

Rules

Operation	Result
0 × 0	0
0 × 1	0
1 × 0	0
1 × 1	1

Example 1

```
    101
  ×  10
  -----
    000
+  1010
  -----
   1010
```

$101_2 \times 10_2 = 1010_2$

Example 2

```
    101
  ×  11
  -----
    101
+  1010
  -----
   1111
```

$101_2 \times 11_2 = 1111_2$

Example 3

```
    110
  × 101
  -----
    110
   000
+ 11000
  -----
  11110
```

$110_2 \times 101_2 = 11110_2$

4. Binary Division

Binary division follows the same basic process as **decimal long division**.

Example 1

```
       10
     _____
10  )  100
       10
       --
        00
```

```
         0
        --
         0
```

$100_2 \div 10_2 = 10_2$

Example 2

```
          11
       ______
10  )  110
       10
       --
        10
        10
        --
         0
```

$110_2 \div 10_2 = 11_2$

Example 3

```
          101
       _______
10  )  1010
       10
       --
        01
         0
        --
         10
         10
         --
          0
```

$1010_2 \div 10_2 = 101_2$

Binary Subtraction Using 2's Complement

In binary arithmetic, **subtraction can also be performed by adding the 2's complement of the subtrahend to the minuend**.

> **Note: Minuend** is the number from which another number is subtracted.
> **Subtrahend** is the number that is subtracted from the minuend.
>
> **Example:** In **9 − 6**, **9** is the **minuend** and **6** is the **subtrahend**.

Example: Subtract 6 from 9

Minuend = $9_{10} = 1001_2$

Subtrahend = $6_{10} = 0110_2$

Step 1: Find the 2's Complement of the Subtrahend

Subtrahend:

```
0110
```

Invert all the bits:

```
1001
```

Add 1:

```
  1001
+ 0001
------
  1010
```

Therefore, the **2's complement of 0110_2 is 1010_2**.

Step 2: Add the Minuend and 2's Complement

```
  1001
+ 1010
------
 10011
```

Step 3: Discard the Carry Bit

The result is:

```
10011
 ↑
Carry
```

Discard the leftmost carry:

```
0011
```

$0011_2 = 3_{10}$

Therefore:

$9_{10} - 6_{10} = 3_{10}$ ✅

or

$1001_2 - 0110_2 = 0011_2$ ✅

> **Note:** In 2's complement subtraction, first find the **2's complement of the subtrahend**, add it to the **minuend**, and **discard the final carry** if one is produced.

Common Text Encoding Schemes

Computers store text as **binary data (0s and 1s)**. Text encoding schemes convert **letters, numbers, and symbols** into a form that computers can understand and store.

1. ASCII

ASCII stands for **American Standard Code for Information Interchange**.
It uses **7 bits** and represents **128 characters**.

Example

When you type **A**, ASCII represents it as **65**.
A → 65

> **Note:** ASCII uses **7 bits**, so it can represent 2^7 **= 128 characters**.

2. Extended ASCII

Extended ASCII uses **8 bits** and can represent up to **256 characters**.
It includes additional symbols and **special characters** that are not included in standard ASCII.

Example

Some extended ASCII character sets can represent additional characters such as **β (Greek beta)** and other special symbols.

> **Note:** Extended ASCII is not one single universal character set. Different systems may use different 8-bit character sets for the additional 128 characters.

3. Unicode

Unicode is a character standard used to represent characters from **different languages and writing systems**.

Example

Unicode can represent English **A**, Urdu ب, Arabic ع, Chinese 中, and many other characters.
Common Unicode encoding formats are:

- **UTF-8**
- **UTF-16**
- **UTF-32**

UTF stands for **Unicode Transformation Format**.

4. UTF-8

UTF-8 is a variable-length encoding that uses **1 to 4 bytes** for a character.
It is **backward compatible with ASCII**. This means that standard ASCII characters use the same byte values in UTF-8.

Examples

A → 01000001 → 1 byte

An Urdu character such as ہ requires **2 bytes** in UTF-8.

> **Note:** UTF-8 uses fewer bytes for many common characters, especially English characters, which makes it efficient for storing and transmitting text.

5. UTF-16

UTF-16 is a variable-length encoding that uses **2 or 4 bytes** for a character.

Example

A → 00000000 01000001 → 2 bytes

Most commonly used characters require **2 bytes**, while some characters require **4 bytes**.

6. UTF-32

UTF-32 is a fixed-length encoding that uses exactly **4 bytes** for every character.

Example

A → 00000000 00000000 00000000 01000001

Therefore, the character **A** takes **4 bytes** in UTF-32.

Comparison of Text Encoding Schemes

Encoding	Size	Characters / Purpose
ASCII	7 bits	128 characters

Encoding	Size	Characters / Purpose
Extended ASCII	8 bits	Up to 256 characters
Unicode	Character standard	Characters from many languages
UTF-8	1–4 bytes	Variable-length Unicode encoding
UTF-16	2–4 bytes	Variable-length Unicode encoding
UTF-32	4 bytes	Fixed-length Unicode encoding

How Computers Store Files

Computers store **all types of files**, such as **images, audio, videos, and documents**, as **binary data (0s and 1s)**.

Storage Devices

1. Hard Disk Drive (HDD)

A **Hard Disk Drive (HDD)** uses **spinning disks** to read and write data. It usually provides a **large storage capacity**.

2. Solid State Drive (SSD)

A **Solid State Drive (SSD)** uses **flash memory** to store data. It provides **faster access and better performance** than an HDD.

Example

When you save a photo on your laptop, the photo is converted into **binary data** and stored on the computer's **HDD or SSD**.

3. Cloud Storage

Cloud storage stores files on **remote servers** that can be accessed through the **internet**.
It is useful for **backup** and for accessing files from **different devices**.

Examples of Cloud Storage

- **Google Drive**
- **OneDrive**
- **Dropbox**
- **iCloud**
- **Google Photos**

> **Note:** Regardless of the type of file, computers ultimately store and process digital data in the form of **0s and 1s**. Different file types use different formats to organize this binary data.

Multiple Choice Questions (MCQs) on Introduction To Computational Systems

1. What is a computational system?

a. A system used only for entertainment
b. A system that processes data to produce useful information
c. A system used only for communication
d. A system that only stores files

> **Answer: b. A system that processes data to produce useful information**

2. What is the main purpose of a computational system?

a. To process data
b. To play games
c. To display pictures only
d. To print documents only

> **Answer: a. To process data**

3. Which of the following is a component of a computational system?

a. Furniture
b. Electricity bill
c. Hardware
d. Office building