Unit no 02: Number Systems

Exercise:

Give Short answers to the questions.

1. What is the primary purpose of the ASCII encoding scheme?

Purpose of the ASCII encoding scheme:

ASCII (American Standard Code for Information Interchange) is used to represent text in computers by assigning a unique numeric code to each letter, digit, and symbol. This allows computers to store, process, and exchange text data using only numbers, which they understand. It includes 128 standard characters, covering English letters (A–Z, a–z), digits (0–9), punctuation marks, and control characters (like Enter and Tab).

2. Explain the difference between ASCII and Unicode?

Difference between ASCII and Unicode:

Feature	**ASCII**	**Unicode**
Full Form	American Standard Code for Information Interchange	Universal Character Encoding Standard
Characters	128 characters only (English letters, digits, symbols)	Over 140,000 characters (all languages, symbols, emojis)
Size	Uses **7 bits** per character	Uses **8, 16, or 32 bits** per character
Language Support	Supports only **English**	Supports **all major languages**
Use Case	Used in older or simple systems	Used in modern websites, apps, and systems

---

In Short:

ASCII is limited to English and uses 7-bit codes, while **Unicode** is a global standard that supports all languages and symbols with more bits.

3. How does Unicode handles characters from different languages?

How Unicode Handles Characters from Different Languages:

Unicode assigns a unique number (called a code point) to every character from every language, so computers around the world can display and process text in any script (like Arabic, Chinese, Urdu, Hindi, etc.).

🔹 **Simple Explanation:**

- Each character, symbol, or emoji has a **unique code** in Unicode.

- For example:
    - English **‘A’** = U+0041
    - Arabic **‘م’** = U+0645
    - Urdu **‘ش’** = U+0634

These codes work the **same way on all devices**, which makes Unicode **universal and reliable** for different languages.

In Short:

Unicode uses a unique code for every character from every language, allowing text from all scripts to be stored, displayed, and shared correctly on any system.

4. What is the range of vales for an unsigned 2-byte integer?

The range of values for an **unsigned 2-byte integer** (which uses 16 bits) is:

- **Minimum value**: **0**
- **Maximum value**: **65,535**

Explanation:

- An **unsigned** integer does not have a sign (negative), so it can only represent non-negative values.
- With 16 bits, you can represent values from **0** to **(2^16 - 1)**, which is **65,535**.

5. Explain how a negative integer is represented in binary?

How a Negative Integer is Represented in Binary:

Negative integers are typically represented in binary using a method called **Two's Complement**. This is the most common method used by computers.

🔹 **Two's Complement Representation Steps:**

1. **Start with the positive binary representation** of the number.
2. **Invert all the bits** (change 1s to 0s and 0s to 1s).
3. **Add 1** to the result from step 2.

🔹 **Example:**

Let's represent **-5** in 8-bit binary:

1. **Start with the positive binary** of 5:
   5 = 00000101 (in 8 bits)
2. **Invert the bits**:
   11111010
3. **Add 1**:
   11111010 + 1 = 11111011

So, **-5** in 8-bit two's complement is: **11111011**

6. What is the benefit of using unsigned integers?

Benefit of Using Unsigned Integers:

The main benefit of using **unsigned integers** is that they can **represent a larger range of positive numbers** compared to signed integers of the same bit length.

- **Unsigned integers** do not reserve any bits for representing negative values. All the bits are used for representing positive numbers.
- For example, with **8 bits**:
  - **Unsigned integer**: Can represent numbers from **0 to 255**.
  - **Signed integer**: Can represent numbers from **-128 to 127** (since 1 bit is used for the sign).

In Short:

Unsigned integers are useful when you only need to represent **positive values**, and they allow you to store **larger numbers** within the same bit size.

7. How does the number of bits affect the range of integer values?

The **more bits** you use, the **larger the range** of numbers you can represent.

◆ **For Unsigned Integers:**

- The range is from **0 to ($2^n - 1$)**, where **n** is the number of bits.
- Example:
  - **8 bits** → 0 to **255**
  - **16 bits** → 0 to **65,535**
  - **32 bits** → 0 to **4,294,967,295**

◆ **For Signed Integers (using Two's Complement):**

- The range is from **-2^{n-1} to ($2^{n-1} - 1$)**
- Example:
  - **8 bits** → −128 to **127**
  - **16 bits** → −32,768 to **32,767**
  - **32 bits** → −2,147,483,648 to **2,147,483,647**

In Short:

More bits = bigger number range.
Each extra bit **doubles** the range of values that can be stored.

8. Why are whole numbers commonly used in computing for quantities that cannot be negative?

Whole numbers (unsigned integers) are commonly used in computing when the values **cannot be negative**, because:

- **1. Saves Memory Space:**
  - No need to reserve a bit for the negative sign.
  - All bits are used to represent **positive values**, allowing a **larger range**.
- **2. Logical Accuracy:**
  - Some quantities **don’t make sense as negative**, like:
    - File sizes
    - Number of users
    - Age
    - Pixels
    - Items in a list

Using unsigned integers ensures the value **stays non-negative**.

In Short:

Whole numbers (unsigned) are used when values can't be negative because they give a **larger positive range**, **save memory**, and **prevent errors**.

9. How is the range of floating-point numbers calculated for single precision?

Single-precision floating-point numbers use **32 bits**, divided into three parts:

- **1 bit** for the **sign**
- **8 bits** for the **exponent**
- **23 bits** for the **fraction (mantissa)**

Range Calculation Formula:

The general formula is:

sign*mantissa*2^{exp}

- The exponent is **stored with a bias of 127**
- This allows both very **small** and very **large** numbers

Approximate Range of Single Precision:

- **Smallest positive number (near zero)**:
  $\approx 1.4 \times 10^{-45}$
- **Largest positive number**:
  $\approx 3.4 \times 10^{38}$

10. Why is it important to understand the limitations of floating-point representation in scientific computing?

It is important because of floating-point numbers:

1. **Can't store all decimal numbers exactly**, so small rounding errors can happen.
2. Have a **limited range**, so very big or very small numbers may not be stored correctly.
3. Can give **wrong results in repeated calculations or comparisons**.

In scientific computing, even small mistakes can cause big problems, so we must understand these limits to get **correct results**. Floating-point numbers are useful but not perfect. Knowing their **limits helps avoid mistakes** in scientific or technical work.