Unit no 02: Number Systems

Long Question Answers:

1. **Explain how characters are encoded using Unicode. Provide examples of characters from different languages and their corresponding Unicode code points.**

Unicode

Unicode is an attempt at mapping all graphic characters used in any of the world's writing systems. Unlike ASCII, which is limited to 7 bits and can represent only 128 characters, Unicode can represent over a million characters through different forms of encodings such as UTF-8, UTF-16, and UTF-32. UTF is an acronym that stands for Unicode Transformation Format.

UTF-8

It is a variable-length encoding scheme, meaning it can use a different number of bytes (from 1 to 4) to represent a character. UTF-8 is backward compatible with ASCII. It means it can understand and use the older ASCII encoding scheme without any problems. Therefore, if we have a text file written in ASCII, it will work perfectly fine with UTF-8, allowing it to read both old and new texts.

Example: The letter 'A' is Unicode, represented as, U+0041, is 01000001 in the binary format and occupies 8 bits or 1 byte.

Let's look at how Urdu letters are represented in UTF-8:

Example: The Urdu letter 'ب' is represented in Unicode as U+0628; its binary format is 11011000 10101000, means it takes 2 bytes.

UTF-16

UTF-16 is another variable character encoding mechanism, although it uses either 2 bytes or 4 bytes per character at most. Unlike UTF-8, it is not compatible with ASCII, meaning it cannot translate ASCII code.

Example: The letter A in UTF-16 is equal to 00000000 01000001 in binary or 65 in decimal (2 bytes).

For Urdu:
Example: The right Urdu letter 'ب' in UTF-16 is represented as is 00000110 00101000 in binary, which occupies 2 bytes of memory.

UTF-32

UTF-32 is a method of encoding that uses a fixed length, with all characters stored in 4 bytes per character. This makes it very simple but at the same time it may look a little complicated when it comes to space usage.

Example: Alphabet letter 'A' in UTF-32 is represented in binary as 00000000 00000000 00000000 01000001 which is 4 bytes.

2. **Describe in detail how integers are stored in computer memory.**

Integers (Z)

Integers extend the concept of whole numbers to include negative numbers. In computer programming, we call them signed integers. The set of integers is represented as:

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

To store both positive and negative values, one bit is reserved as the sign bit (the most significant bit). If the sign bit is ON (1), the value is negative; otherwise, it is positive. Using this system, the maximum positive value that can be stored in a 1-byte signed integer is $(01111111)_2$, which is 127_{10}. As the bits available to store a value is $n - 1$, hence the maximum value will be $2^{n-1} - 1$. We can use this formula to compute the maximum values for 2 and 4 bytes.
Negative values are stored using two's complement, explained in the following section.

Negative Values and Two's Complement

To store negative values, computers use a method called two's complement. To find the two's complement of a binary number, follow these steps:

1. Invert all the bits (change 0s to 1s and 1s to 0s).
2. Add 1 to the Least Significant Bit (LSB).

Example: Let's convert the decimal number -5 to an 8-bit binary number:

1. Start with the binary representation of 5: 00000101_2
2. Invert all the bits: 11111010_2
3. Add 1: $11111010_2 + 1_2 = 11111011_2$

So, -5 in 8-bit two's complement is **11111011_2**

Minimum Integer Value

For an 8-bit integer, we switch on the sign bit for the negative value and turn all bits ON, resulting in 11111111_2. Except the first bit, we take two's complement and get 10000000, which is 128_{10}. Thus, minimum value in 1-byte signed integer is -128, i.e., -2^7. The minimum value is computed using the formula -2^{n-1}, where n is the total number of bits.

- **2-Byte Integer (16 bits):** Minimum value = $-2^{15} = -32{,}768$
- **4-Byte Integer (32 bits):** Minimum value = $-2^{31} - 1 = -2{,}147{,}483{,}648$

3. Explain the process of converting a decimal integer to its binary representation and vice versa. Include examples of both positive and negative integers.

1. Conversion from Decimal to Binary (Positive Integer)

The following algorithm translates a decimal number to binary.

1. To convert a decimal number to binary form, divide the decimal number by 2.
2. Record the remainder.
3. Divide the number by 2 until the quotient which is left after division is 0.
4. Meaning it is represented by the remainders, and it's read from the bottom to the top of the binary number.

Example: Convert 83 to binary

83 / 2 = 41 remainder 1

41 / 2 = 20 remainder 1

20 / 2 = 10 remainder 0

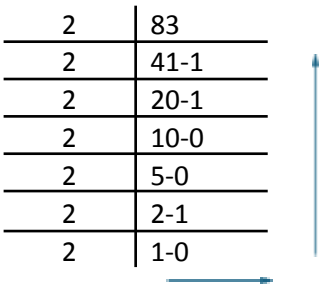
10 / 2 = 5 remainder 0

5 / 2 = 2 remainder 1

2 / 2 = 1 remainder 0

1 / 2 = 0 remainder 1

The above steps are graphically shown in Figure 2.1. If the remainders are read from bottom to top then it gives the required result in binary, which is **1010011**.



2	83
2	41-1
2	20-1
2	10-0
2	5-0
2	2-1
2	1-0

2. Conversion from Decimal to Binary (Negative Integer)

Negative integers are stored in binary using **Two's Complement**.

Steps:

1. Convert the **absolute value** of the number to binary.
2. Invert the bits (0 → 1, 1 → 0).
3. Add **1** to the result.

Example: Convert –5 to 8-bit binary

1. 5 in binary (8-bit): 00000101
2. Invert bits: 11111010
3. Add 1: 11111010 + 1 = 11111011

Binary (–5) = 11111011_2

Binary to Decimal

Multiply each bit by 2 raised to its position (starting from 0, right to left), then sum them.

Example: Convert 1010_2 to decimal

$= 1\times2^3 + 0\times2^2 + 1\times2^1 + 0\times2^0$

$= 8 + 0 + 2 + 0 = 10$

Decimal = 10

---

4. Perform the following Binary Arithmetic operations:

a. Multiplication of 101 by 11.

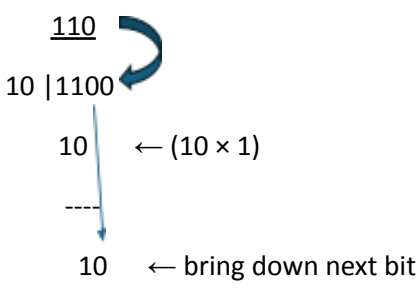
```
   101
  ×  11
 _______
   101     ← (101 × 1) [Multiply by rightmost bit]
+  101 ×   ← (101 × 1) shifted one position to the left
 _______
  1111
```

Binary Result: 1111
Decimal Check: 5 × 3 = 15 ⟶ Binary of 15 is 1111

b. Division of 1100 by 10

- 1100 in binary = 12 in decimal
- 10 in binary = 2 in decimal

Step-by-step Binary Division:



10 ← (10 × 1)

---

00 ← remainder

Final Answer:

- Quotient (binary): 110 → 6 in decimal
- Remainder: 0

So,
1100 ÷ 10 = 110 (binary)

---

5. Add the following binary numbers:

a. 101+110

101 (5 in decimal)

\+ 110 (6 in decimal)

1011 (11 in decimal)

Binary Result: 1011
Decimal Check: 5 + 6 = 11 → Binary of 11 is 1011

b. 1100 +1011

1100 (12 in decimal)
\+ 1011 (11 in decimal)

10111 (23 in decimal)

Binary Result: 10111
Decimal Check: 12 + 11 = 23 → Binary of 23 is 10111

---

6. Convert the following numbers to 4-bit binary and add them :

a. 7 + (- 4)

Step 1: Convert +7 to 4-bit binary
7 → 0111
Step 2: Convert -4 to 4-bit binary
1. Start with +4 → 0100
2. Invert the bits → 1011
3. Add 1 → 1011 + 1 = 1100

So, **-4 in 4-bit 2's complement = 1100**
Step 3: Add the two 4-bit binaries

0111 (7)
\+ 1100 (-4)

10011

This is a **5-bit result**, but we only keep the **lower 4 bits** (0011) in 4-bit arithmetic and **ignore the carry**.
Final Answer:
Result (4-bit binary): 0011
Decimal Check: 7 + (-4) = **3** ⟶ Binary: 0011

b. -5 + 3

Step 1: Convert -5 to 4-bit 2's complement

1. +5 ⟶ 0101
2. Invert bits ⟶ 1010
3. Add 1 ⟶ 1010 + 1 = 1011

So, **-5 in 4-bit = 1011**
Step 2: Convert +3 to 4-bit binary
3 ⟶ 0011
Step 3: Add the two binaries

```
 1011   (-5)
+ 0011   (+3)
 -------
  1110
```

Final Answer:
Result (4-bit binary): 1110
Decimal Check: -5 + 3 = **-2**, and 1110 in 4-bit 2's complement = **-2**

---

7. Solve the following:

a. 1101_2 -0100_2

1101_2 (which is 13 in decimal)
− 0100_2 (which is 4 in decimal)
Step-by-step Binary Subtraction:

```
  1101
− 0100
  -------
  1001
```

Final Answer:
Binary Result: 1001
Decimal Check: 13 − 4 = 9 ⟶ Binary: 1001

b. 1010_2-0011_2

1010_2 (which is 10 in decimal)
− 0011_2 (which is 3 in decimal)
Step-by-step Binary Subtraction:

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

Final Answer:
Binary Result: 0111

Decimal Check: 10 − 3 = 7 → Binary: 0111

c. $1000_2 - 0110_2$

1000_2 (which is 8 in decimal)
$- 0110_2$ (which is 6 in decimal)
Step-by-step Binary Subtraction:

```
  1000
- 0110
  -------
  0010
```

Final Answer:
Binary Result: 0010
Decimal Check: 8 − 6 = 2 → Binary: 0010

d. $1110_2 - 100_2$

1110_2 (which is 14 in decimal)
$- 0100_2$ (which is 4 in decimal)
Step-by-step Binary Subtraction:

```
  1110
- 0100
  -------
  1010
```

Final Answer:
Binary Result: 1010
Decimal Check: 14 − 4 = 10 → Binary: 1010