CHAPTER 2: NUMBER SYSTEMS

Solved Exercise

Tick (✓) the correct answer

1. What does ASCII stand for?

(a) American Standard Code for Information Interchange.

(b) Advanced Standard Code for Information Interchange

(c) American Standard Communication for Information Interchange

(d) Advanced Standard Communication for Information Interchange

2. Which of the following numbers is a valid binary number?

(a) 1101102 (b) 11011 (c) 110:11 (d) 1101A

3. How many bits are used in the standard ASCII encoding?

(a) 7 bits (b) 8 bits (c) 16 bits (d) 32 bits

4. Which of the following is a key advantage of Unicode over ASCII?

(a) It uses fewer bits per character

(b) It can represent characters from many different languages

(c) It is backward compatible with binary

(d) It is specific to the English language

5. How many bytes are used to store a typical integer?

(a) 1 byte (b) 2 bytes (c) 4 bytes (d) 8 bytes

6. What is primary difference between signed and unsigned integers?

(a) Unsigned integers cannot be negative

(b) Signed integers have a larger range

(c) Unsigned integers are stored in floating-point format

(d) Signed integers are only used for positive numbers

7. In the single precision, how many bits are used for the. exponent?

(a) 23 bits (b) 8 bits (c) 11 bits (d) 52 bits

8. What is the approximate range of values for single-precision floating-point numbers?

(a) 1.4 x 10'45to3.4x 1038 (b) 1.4x1038to3.4xW45

(c) 4.9x1 O'324 to 1.8 x W308 (d) 4.9x1 O'308 to 1.8 x 10324

9.What are the tiny dots that make up an image called?

(a) Pixels (b) Bits (c) Bytes (d) Nodes

10. In an RGB color model, what does RGB stand for?

(a) Red, Green, Blue (b) Red, Gray, Black

(c) Right, Green, Blue (d) Red, Green, Brown

Answer Key:

1	2	3	4	5	6	7	8	9	10
a	b	a	b	c	a	b	a	a	a

Short Answer Questions

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

Ans: ASCII is an acronym that stands for American Standard Code for Information Interchange. It is a character encoding standard adopted for representing in devices such as computers and similar systems that use text. Each alphabet, number or symbol is given a code number between 0 and 127.

2. Explain the difference between ASCII and Unicode.

Ans: ASCII (American Standard Code for Information Interchange) and Unicode are two-character encoding standards used to represent text in computers.

ASCII	Unicode
Developed in the 1960s	Developed in the 1990s
Uses 7-bit binary code (0-127) and Limited to 128 unique characters including: - Uppercase and lowercase letters - Digits (0-9) - Punctuation marks	Compatible with ASCII, as the first 128 Unicode characters match the ASCII character set.
Initially used for tele-printers and early computers	Uses variable-length encoding (UTF-8, UTF-16, UTF-32)
Does not support non-English languages	Designed to support languages from around the world

3. How does Unicode handle characters from different languages?

Ans: Unicode is an attempt at mapping all graphic characters used in any of the world's writing system. Unlike ASCII, which is limited to 7bits 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.

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

Ans: For an unsigned 2-byte integer, the range of values is: 0 to 65535
This is because:

- 2 bytes = 16 bits
- Unsigned integer: all 16 bits are used to represent the value
- Maximum value: $2^{16} - 1 = 65535$
- Minimum value: 0

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

Ans: Negative integers are typically represented in binary using a method

called two's complement.

Steps :

1. Determine the Bit Length (e.g., 8 bits, 16 bits) to represent the number.
2. Write the Positive Version in Binary
3. Invert all the bits (change 0 to 1 and 1 to 0). This step is called the one's complement.
4. Add 1 to the result of the one's complement to get the two's complement representation.

6. What is the benefit of using unsigned integers?

Ans: Unsigned integers do not need a sign bit; they can represent larger positive numbers compared to signed integers of the same bit size. Using unsigned integers can provide benefits in terms of range, precision, performance, efficiency, code clarity, and safety, making them a suitable choice for various applications.

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

Ans: The number of bits determines the range of values. The range depends on whether the integer is signed or unsigned.

For signed integers, one bit is used to represent the sign (0 for positive, 1 for negative). The remaining bits represent the magnitude.

For unsigned integers, all bits represent the magnitude.

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

Ans: In computing, whole numbers are often used to represent quantities that can't be negative. Examples include the number of students in a school, a person's age in years, and grades, provided there are no negative figures such as credit point balances. Using whole numbers (unsigned integers) for quantities that cannot be negative is a practical, logical, and conventional choice in computing.

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

Ans: In this standard, 4 bytes (or 32 bits) are assigned where the 1st bit is the sign bit, and the next 8 bits are for the exponent and the remaining 23 bits are for the mantissa.

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

Ans: *Understanding the limitations of floating-point representation is essential in scientific computing because it can significantly impact the accuracy, reliability, and efficiency of numerical simulations and analyses. It becomes possible to control and manipulate the numbers in different ways.*

Long Questions

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

Ans: Unicode

Unicode is an attempt at mapping all graphic characters used in any of the world's writing system. Unlike ASCII, which is limited to 7bits 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.

1. 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 1101100010101000, means it takes 2 bytes.

2. UTF-16

UTF-16 is another variable character encoding mechanism, although it uses either 2 bytes or4 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 0000000001000001 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. .

3. 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.

Ans: Integers are represented in binary using a combination of bits i.e. 0s and 1s. The number of bits used to represent an integer depends on the size of the

integer.

Bits and Bytes: Computers operate using binary digits (bits), with each bit representing either 0 or 1. A group of 8 bits is called a byte.

Integer Size: The size of an integer determines how many bits are used to store it. Common sizes are:

8-bit integers (1 byte)
16-bit integers (2 bytes)
32-bit integers (4 bytes)
64-bit integers (8 bytes)

Unsigned Integers

Definition: Unsigned integers represent non-negative values only (0 and positive numbers).

Range: The range depends on the number of bits:

Range = 0 to $(2^n - 1)$ where n is the number of bits.

Signed Integers

Definition: Signed integers can represent both positive and negative values.

Sign Bit: The most significant bit (MSB) is reserved as the sign bit:

0: Positive number
1: Negative number

Range: The range of signed integers is:

Range= (-2^{n-1}) to $(2^{n-1} -1)$- where n is the number of bits.

2'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 O's to 1's and 1's to 0's).
2. Add 1to 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. Invert all the bits: 11111010
3. Add1:11111010 + 1 = 11111011

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

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. Except the first bit, we take two's complement| and get 10000000, which is 128. Thus, minimum value in 1-byte signed integer is. -128, i.e., -27. The minimum value is computed using the formula -2n-1, where n is the total number of bits.

- 2-Byte integer (16 bits): Minimum value = -216-1=215 =-32,768
- 4-ByteInteger (32 bits): Minimum value = -232-1-1=231-1 = -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.**

Ans: Conversion from Decimal to Binary:

The following algorithm translates a decimal number to binary.

- To convert decimal number to binary form, divide the decimal number by 2.
- Record the remainder.
- Divide the number by 2 until the quotient which is left after division is 0.
- 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 =1remainder 0
1/2 = 0 remainder 1
Hence $83_{10}=1010011_2$

Converting a Positive Binary Integer to Decimal:

This involves multiplying each bit in the binary number by 2^n, where n is the position of the bit from right to left, starting at 0.

Steps:

- Write down the binary number.
- Assign powers of 2 to each bit, starting from 202^020 on the far right.
- Multiply each bit by its corresponding power of 2.
- Sum the results.

Example: Convert 1010011_2 to Decimal.

$1010011_2= 1x2^6+0x2^5+1x2^4+0x2^3+0x2^2+1x2^1+1x2^0$
$= 1x64+0+1x16+0+0+1x2+1x1$
$= 64+16+2+1$
$= 83$

Converting a Negative Decimal Integer to Binary

Negative integers are typically represented using **two's complement** in binary systems.

Steps for Two's Complement:

- Convert the absolute value of the decimal number to binary.
- Invert all bits (change 0 to 1 and 1 to 0).
- Add 1 to the inverted binary number.

Example (Decimal -5 to Binary in 8 bits):
Absolute value of −5= 5
25/2=12 remainder 1
12/2=6 remainder 0
1/2=0 remainder 1
Binary: 101 (3 bits, padded to 8 bits: 00000101)
Invert bits: 11111010
Add 1: 11111011
Result: Decimal −5 = Binary 11111011 (in 8-bit two's complement)

Converting a Negative Binary Integer to Decimal

To convert a two's complement binary number back to decimal:

- Check the most significant bit (MSB):
  - 0: Positive number (convert normally as in Step 2).
  - 1: Negative number (proceed to Step 2).
- Invert all bits and add 1 to get the absolute value of the number.
- Add a negative sign to the result.

Example (Binary 11111011 to Decimal):

- MSB is 1, so it's negative.
- Invert bits: 00000100
- Add 1: 00000101
- Convert to decimal: $1\times2^2+0\times2^1+1\times2^0=5$
- Add a negative sign: −5

Result: Binary 11111011 = Decimal = −5

4. Perform the following binary arithmetic operations:

a. Multiplication of 101 by 11.

b. Division of 1100 by 10.

a.

$$\begin{array}{r} 101 \\ \times\ 11 \\ \hline 101 \\ 101x \\ \hline 1111 \end{array}$$

b.

$$\begin{array}{r|l} 1100 & 10 \\ \hline -10 & 110 \\ \hline 10 & \\ -10 & \\ \hline 0 & \end{array}$$

5. Add the following binary numbers:

a)

101
+ 110
1011

b)

1100
+ 1011
10111

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

a: 7 +(-4)

- Convert the numbers to their 4-bit binary representation.
- Use two's complement for negative numbers.
- Perform binary addition.

1. Represent 7 in 4-bit binary:

- 7 in binary: 0111.

2. Represent 4 in 4-bit binary (Two's Complement):

- Start with 4 in binary: 0100
- Invert the bits: 1011
- Add 1: 1011+1=1100

Thus, −4 in 4-bit binary is 1100.

3. Add 0111 **and** 1100

The result is 10011. Since we're working with 4 bits, the overflow bit (leftmos
1) is discarded, leaving:

Result: 0011

4. Convert 0011 **back to decimal:**

0011 in decimal: 3.

Final Answer: 7+(−4)=3

(b) —5 + 3

1.Represent −5 in 4-bit binary (Two's Complement):

- Start with 5 in binary: 0101
- Invert the bits: 1010
- Add 1: 1010+1=1011.

Thus, −5 in 4-bit binary is 1011.

2. Represent 3 in 4-bit binary:

- 3 in binary: 0011
- **Add** 1011 **and** 0011
- The result is 1110

3. Convert 1110 back to decimal:

Since 1110 has the most significant bit (MSB) as 1, it represents a negative number in two's complement.

4. To find the decimal value:

- Invert the bits: 1110→0001
- Add 1: 0001+1=0010 =2
- Apply the negative sign: −2.

Final Answer: −5+3=−2

7. Solve the following:

(a) 1101_2-0100_2

```
  1101
 -010
    0
 ----
  1001
 ----
```

(b) 1010_2-0011_2

```
  1010
 -001
    1
 ----
  0011
 ----
```

(c) 1000_2-0110_2

```
  1000
 -011
    0
 ----
  0010
 ----
```

(d) 1110_2 -100_2

```
  1110
 - 10
    0
 ----
  1010
 ----
```