Chapter 2: Number Systems

LONG Question from the Chapter "Number System":

EXERCISE LONG Question with Answers

---

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

What is Unicode?
Unicode is a standard that assigns a unique number (called a *code point*) to every character used in writing systems around the world — letters, digits, symbols and even emoji. A code point is written as `U+` followed by a hexadecimal number (for example `U+0041`).

Why Unicode?
Older standards (like ASCII) handle only English characters. Unicode can represent characters from many languages (Arabic, Chinese, Hindi, Cyrillic, etc.), so computers can show and exchange text globally.

Code points — some examples:

- `A` (Latin capital A) → **U+0041**
- `π` (Greek small pi) → **U+03C0**
- ﻡ(Arabic letter Meem) → **U+0645**
- 你 (Chinese character "you") → **U+4F60**
- ह (Devanagari letter "ha") → **U+0939**
- 😀 (grinning face emoji) → **U+1F600**

How encoding works (short):

- Unicode assigns code points (e.g., `U+0041`).
- To store/send characters, Unicode uses encodings such as **UTF-8**, **UTF-16**, or **UTF-32**.
    - UTF-8 is most common: ASCII characters (U+0000 to U+007F) use 1 byte, other characters use 2–4 bytes.
    - Example: `A` (U+0041) in UTF-8 is a single byte `0x41`. 你 (U+4F60) requires 3 bytes in UTF-8.

Summary: Unicode gives every character a unique code point so text in any language can be represented and exchanged correctly.

---

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

Basic idea:
Integers are stored in binary (0 and 1). The number of bits used (8, 16, 32, 64) determines how large or small values can be.

Common integer sizes:

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

Unsigned vs Signed integers:

- **Unsigned** integers store only non-negative numbers: range = `0` to `2^n - 1` (where `n` is number of bits).
  - Example: 8-bit unsigned → `0` to `255` (`2^8 - 1 = 255`).
- **Signed** integers allow negative values. The most common method is **two's complement**. Range = `-2^(n-1)` to `2^(n-1) - 1`.
  - Example: 8-bit signed (two's complement) → `-128` to `+127`.

Two's complement representation (most used):

- To represent a positive number, write its normal binary.
- To represent a negative number `-X` in `n` bits:
  1. Write `X` in `n`-bit binary.
  2. Invert all bits (ones → zeros, zeros → ones).
  3. Add `1` to the inverted result.
- Example: store `+5` and `-5` in 8 bits:
  - `+5` = `0000 0101`.
  - `5` inverted = `1111 1010`; add `1` → `1111 1011` = `-5`.

Endianness (note): When integers use multiple bytes, the order of bytes in memory may be **little-endian** (least significant byte first) or **big-endian** (most significant byte first). This affects how bytes appear in memory but not the numeric value.

Summary: Integers are binary values stored in fixed bit lengths. Two's complement is widely used for signed integers because arithmetic (addition/subtraction) works naturally.

---

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

(A) Decimal → Binary (positive number) — repeated division method

Example: Convert **25** (decimal) to binary.

Steps:

1. Divide 25 by 2 → quotient 12, remainder **1**.
2. Divide 12 by 2 → quotient 6, remainder **0**.
3. Divide 6 by 2 → quotient 3, remainder **0**.
4. Divide 3 by 2 → quotient 1, remainder **1**.
5. Divide 1 by 2 → quotient 0, remainder **1**.
   Now read remainders from bottom to top: **11001**.
   So $\mathbf{25_{10} = 11001_2}$.

(B) Binary → Decimal (positive number) — positional method

Example: Convert $\mathbf{11001_2}$ to decimal.

Compute powers of 2:

- $1\times2^4 + 1\times2^3 + 0\times2^2 + 0\times2^1 + 1\times2^0 = 16 + 8 + 0 + 0 + 1 = \mathbf{25}$.

(C) Negative integers (two's complement) — example −13 using 8 bits

To encode −13 in 8-bit two's complement:

1. Write +13 in 8 bits: `0000 1101`.
2. Invert bits: `1111 0010`.
3. Add 1: `1111 0011`.
   So **−13** in 8-bit two's complement = $\mathbf{11110011_2}$.

To convert a two's complement binary back to decimal (if MSB = 1):
Example: 11110011_2 (8 bits).

1. Since MSB is 1, it is negative. Find magnitude: invert bits → `00001100`. Add 1 → `00001101` = 13 decimal. So value = **−13**.

Summary: Use repeated division for positive decimal → binary. For negative integers use two's complement steps (invert + add 1) to represent them in fixed bit width.

---

4. Perform the following binary arithmetic operations

(a) Multiplication: $\mathbf{101_2 \times 11_2}$

First, understand values: $101_2 = 5_{10}$, $11_2 = 3_{10}$. Result should be $15_{10} = 1111_2$.

Step-by-step (binary long multiplication):

```
101      (multiplicand)
```

```
  ×  11      (multiplier)
  -----
    101      (101 × 1    )   ← least significant bit of multiplier
   1010      (101 × 1 shifted left by one) ← next bit
  -----
   1111
```

Final result: $\mathbf{1111_2}$ (which equals 15 decimal).

(b) Division: $1100_2 \div 10_2$

Interpretation: $1100_2 = 12_{10}$; $10_2 = 2_{10}$. So quotient = $6_{10} = 110_2$, remainder = 0.

Simple reasoning: dividing by 10_2 (binary 2) is same as shifting right by 1 bit:

- 1100_2 >> 1 = 110_2 with remainder 0.

Step-by-step long division (short):

- 10 goes into 11 → 1, remainder 1 (bring next bit)
- Next we have 10 → 1, remainder 0
- Next 00 → 0 remainder 0
  → Quotient bits: 110, remainder 0.

Final result: **Quotient = $\mathbf{110_2}$ (6 decimal), Remainder = 0.**

---

5. Add the following binary numbers

(a)

```
  101
+ 110
```

Work column-wise (right to left):

- Rightmost column: $1 + 0 = 1$ → write 1, carry 0.
- Middle column: $0 + 1 = 1$ → write 1, carry 0.
- Left column: $1 + 1 = 10_2$ → write 0, carry 1.
  Carry becomes new leftmost bit 1.

So result: $\mathbf{1011_2}$ (decimal 11).

(b)

```
 1100
+1011
```

Right to left:

- Column1 (rightmost): `0 + 1 = 1`, carry 0.
- Column2: `0 + 1 = 1`, carry 0.
- Column3: `1 + 0 = 1`, carry 0.
- Column4: `1 + 1 = 10`$_2$ → write `0`, carry `1`.
  Carry becomes next bit to left.

So result bits left-to-right: carry `1`, then `0 1 1 1` → **`10111`$_2$** (decimal 23).

---

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

(a) `7 + (-4)`
(b) `-5 + 3`

We use **4-bit two's complement** representation.

(a) `7 + (-4)`

- 4-bit representation:
  - `+7` = `0111`
  - `+4` = `0100` → `-4` in two's complement: invert `0100` → `1011`, add 1 → `1100`. So `-4` = `1100`.

Add:

```
  0111    (+7)
+ 1100    (-4)
-------
 10011   (carry out 1 plus 0011)
```

We ignore the final carry out for fixed 4-bit signed arithmetic. The 4 low bits are `0011`, which equals `+3`. Correct, since `7 + (-4) = 3`.

Final 4-bit result: **`0011` (which is +3)**.

(b) `-5 + 3`

- Representations:
  - `+5` = `0101` → `-5`: invert `0101` → `1010`, add 1 → `1011`. So `-5` = `1011`.
  - `+3` = `0011`.

Add:

```
  1011    (-5)
```

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

1110 in 4-bit two's complement is negative. To find its decimal value: invert 1110 → 0001, add 1 → 0010 = 2 decimal → so value is −2. That matches −5 + 3 = −2.

Final 4-bit result: **1110 (which is −2)**.

---

7. Solve the following (binary subtractions).

Problems (interpreting as base-2 subtraction):

(a) $1101_2 - 0100_2$
(b) $1010_2 - 0011_2$
(c) $1000_2 - 0110_2$
(d) $1110_2 - 100_2$ (note: $100_2 = 4_2$)

We do simple subtraction (all positive results here).

(a) $1101_2 - 0100_2$

- 1101_2 = 13 decimal, 0100_2 = 4 decimal. 13 − 4 = 9.
- Binary result: **1001_2**.

Step (column borrow method):

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

(b) $1010_2 - 0011_2$

- 1010_2 = 10, 0011_2 = 3. 10 − 3 = 7.
- Binary result: **0111_2** (or 111_2 with no leading zero).

Step:

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

(c) $1000_2 - 0110_2$

- 1000_2 = 8, 0110_2 = 6. 8 − 6 = 2.
- Binary result: **0010_2** (or 10_2).

Step:

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

(d) $1110_2 - 100_2$

- $1110_2 = 14$, $100_2 = 4$. $14 - 4 = 10$.
- Binary result: $\mathbf{1010_2}$.

Step:

```
 1110
-0100   (align 100 as 0100)
 ----
 1010
```

Additional LONG Question Only

---

1. Introduction to Number Systems

1. Define number system. Explain why different number systems are used in computer science.
2. Describe the difference between decimal, binary, octal, and hexadecimal systems with examples.
3. Explain the role of base (radix) in number systems with suitable examples.
4. Write in detail how computers use binary number system for processing.
5. Discuss the importance of positional notation in number systems.

---

2. Decimal Number System

1. Explain the structure and working of decimal number system with examples.
2. Differentiate between whole numbers, fractions, and decimal places in the decimal system.
3. Discuss the significance of place values in decimal system.
4. Write detailed steps to convert decimal to binary, octal, and hexadecimal.
5. Explain how decimal system is different from binary in terms of usage and applications.

---

3. Binary Number System

1. Define binary number system and explain why it is used in computers.
2. Explain with steps how to convert decimal numbers into binary and binary into decimal.
3. Differentiate between binary addition and decimal addition with examples.
4. Discuss the advantages and limitations of binary system.
5. Write detailed procedure of converting binary to octal and hexadecimal.

---

4. Octal Number System

1. Define octal number system and explain its importance.
2. Explain step-by-step method of converting octal numbers into binary and decimal.
3. Discuss the process of converting decimal to octal with an example.
4. Compare octal number system with decimal and binary.
5. Why is octal system considered as a shortcut for binary system? Explain with example.

---

5. Hexadecimal Number System

1. Define hexadecimal number system. Why is it widely used in computer science?
2. Explain conversion of hexadecimal numbers into decimal and binary with examples.
3. Write detailed procedure for converting decimal to hexadecimal.
4. Discuss how hexadecimal is used in computer memory addressing and programming.
5. Compare hexadecimal with decimal, binary, and octal systems.

---

6. Conversion between Number Systems

1. Write detailed steps for converting decimal to binary, octal, and hexadecimal with examples.
2. Explain the process of converting binary numbers into decimal, octal, and hexadecimal.
3. Discuss the step-by-step method of converting octal into decimal and binary.
4. Explain how to convert hexadecimal numbers into decimal, binary, and octal.
5. Why are conversions between number systems important in computer science? Explain with examples.
