UNIT 2

Number Systems

Student Learning Outcomes

Understand Number System:

- The different numbering systems, including decimal, binary, hexadecimal, and octal, and their respective base values and digits.
- Why computers primarily use the binary number system for data representation.
- Machine-level representation of data, including how data is stored and processed within the computer's architecture.
- The representation of whole and real numbers in a computer, including binary encoding methods for both.
- How various arithmetic operations, such as addition, subtraction, multiplication, and division, are performed on binary representations of numbers?
- The concept of common text encoding schemes, such as ASCII and Unicode, and How they represent characters.
- How digital data representations work for various forms of multimedia, such as images, audios, videos, and other multimedia resources.
- Different file formats and their variations for specific applications.
- The concept of file extensions and their significance in identifying file types and applications.
- Key terms related to data representation, including ASCII, Unicode, binary, signed and unsigned numbers, bits, bytes, hexadecimal number systems, negatives in binary, two's complement, binary arithmetic, overflow, and underflow.
- The concept of Boolean functions, to represent logic operations and relationships between binary variables.
- How to construct Boolean expressions using variables and Boolean operators.
- Common Boolean identities and simplification techniques.
- The concept of duality in Boolean algebra, where OR becomes AND, and 0 becomes 1.
- The fundamentals of digital logic, which involves using binary digits (0 and 1) to process and store information.
- Difference between analog and digital signals and understanding their key differences.
- Various logic gates (AND, OR, NOT, NAND, XOR) and their functions in processing binary data.
- The purpose and construction of truth tables for evaluating the output of logic expressions based on input combinations.
- The concept of switches and their role in digital systems, often used to represent binary input.
- Karnaugh maps as a visual tool for simplifying Boolean expressions.
- Truth table, Boolean expression, circuit diagram of Half-adder and Full-adder
- Half-adder and Full-adder as digital systems with specific objectives, components and interaction among those components

Introduction

Understanding number systems is fundamental in computer science and digital electronics. This chapter will delve into various numbering systems, their applications, and how they are used in computers. We will cover the following topics:

1. Different numbering systems: decimal, binary, hexadecimal, and octal.
2. Binary number system in computers.
3. Machine-level data representation.
4. Representation of whole and real numbers.
5. Binary arithmetic operations.
6. Common text encoding schemes: ASCII and Unicode.
7. File formats and extensions.
8. Key terms in data representation.
9. Binary data manipulation and conversion.
10. Encoding schemes.
11. Differences between file formats.
12. Storing images, audio, and video in computers.

2.1 Numbering Systems

Numbering systems are essential in computing because they form the basis for representing, storing, and processing data. Different numbering systems help computers perform tasks like calculations, data storage, and data transfer. These systems allow computers to represent various kinds of information, such as text, colors, and memory locations. Here is a description of a few numbering systems:

2.1.1 Decimal System

The decimal number system is a base-10 number system that consists of digit from 0 to 9 and we use it in everyday life. That's why each digit of the number represents a power of 10. In the decimal system the place values starting from the rightmost digits are 10^0, 10^1, 10^2, and so on. For example, the decimal number 523 means:

$5 \times 10^2 + 2 \times 10^1 + 3 \times 10^0 = 500 + 20 + 3 = 523$

2.1.2 Binary System

In binary, the place values are arranged from the right to left, starting with 2^0, and ending at 2^n,where each position represents a power of 2. For example, the binary number 1011 can be converted to decimal as follows:

$$1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 8 + 0 + 2 + 1 = 11_{10}$$

Computers work in binary system especially because this method fits well with electronics. Digital circuits have two states: They can be either on or off. These states are easily represented by the binary digits: 1 represent ON, and 0 represnets OFF. When typing on the keyboard, the computer translates every

letter to a binary. Similarly, number, text, images, and sound are all, at their lowest level, reduced to binary. We shall discuss this is details later in this chapter. When you type a letter on your keyboard, the computer converts it into a binary code. Similarly, all types of data, including numbers, text, images, and sounds, are ultimately broken down into binary code. We will explore it further later in this chapter.

2.1.2.1 Conversion from Decimal to Binary

The following algorithm translate a decimal number to binary.

1. To convert 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.

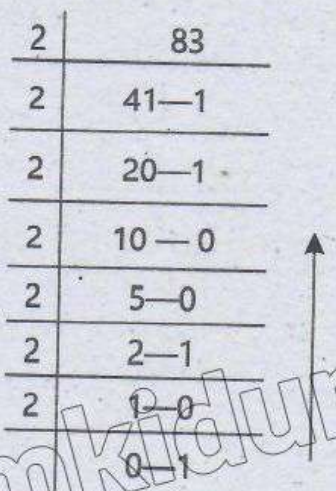


Figure 2.1 Decimal to Binary conversion

Class activity

1. **Marks Conversion:** Each student will take his or her marks from 8th grade for each subject and convert them from decimal to binary. For example, if a student score 85 in Math, he/she will convert 85 to binary (which is 1010101).
2. **Clock Time Conversion:** Students will be given various times of the day and asked to convert them into binary. For instance, 3:45 PM would be converted as follows:
   Hours (15) = 1111
   Minutes (45) = 101101
3. Write your sleeping time in binary.

2.1.3 Octal System

Octal is a positional numeral system with base eight, which implies that a digit to be used ranges from 0 to 7. The last digit is a single digit power of 8 while the other digits are the coefficients. In the decimal system, the place values starting from the 8^0, 8^1, 8^2 and so on. For example, the octal number 157 means, $1 \times 8^2 + 5 \times 8^1 + 7 \times 8^0 = 64 + 40 + 7 = 111_{10}$.

Each octal digit represents three binary digits (bits) because the octal system is base-8, and the binary system is base-2. This relationship arises from the fact that 8 is a power of 2 ($8 = 2^3$). So, each octal digit can be precisely represented by three binary digits (bits). This means that any value from 0 to 7 in octal can be converted into a 3-bit binary number. This relationship makes conversion between binary and octal straight forward. Table 2.1 shows the correspondence between octal and binary digits:

Example:

Consider the 9-bit binary number 110101011. This number can be divided into groups of three

Bits from right to left:

110 101 011

Each group of three bits corresponds to a single octal digit:

110 = 6
101 = 5
011 = 3

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

Table 2.1: Correspondence between Octal and Binary Digits

So, the binary number 110101011 is equal to 653 in octal.

Note that the octal number system isn't actually used in modern computers to do their work. Therefore, we can say that the binary number 110101011 is equal to 653 in octal. Whenever you have a binary number that cannot be divided into groups of a three, you'll have to add zero up to the left end of it to make it appropriate.

2.1.3.1 Conversion from Decimal to Octal

The algorithm below translates a decimal number into an octal.

1. To convert the decimal number to an equivalent octal number, divide the number by 8.
2. Write down the remainder.
3. After that divide the obtained quotient by 8.
4. Continue the divisions until one of the numbers results in 0.
5. Octal is a base eight number and the octal number is the remainder read from the bottom up to the top.

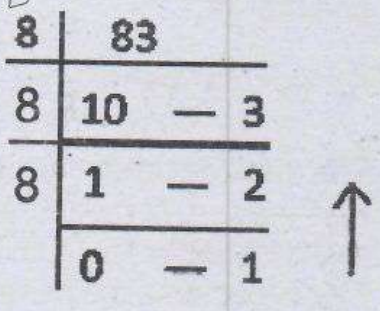


Figure 2.2: Conversion from Decimal to Octal.

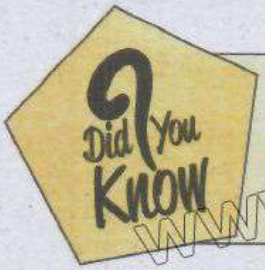
Example: Convert 83 to octal

- 83 / 8 = 10 remainder 3
- 10 / 8 = 1 remainder 2
- 1 / 8 = 0 remainder 1

The above steps are graphically shown in Figure 2.2. Going up from bottom, the remainder reading will give the desired result, that is 123 in the octal system.

Class activity

1. Work in pairs to convert the following decimal numbers to octal: 45, 128, 64.
2. Convert these octal numbers to decimal: 57, 124, 301.
3. Share your answers with the class and discuss any differences.



The octal system was used in early computing systems like PDP-8. It was used because it is easier to convert between octal and binary than between decimal and binary.

Tidbits

When converting between number systems, double-check your remainders and sums to ensure accuracy. Practice with different numbers to become more comfortable with the conversion process.

2.1.4 Hexadecimal System

The hexadecimal is a base 16 number system with digit number from 0 to 9 and alphabets from A to F; each digit represents 16 to the power of the position of the digit. The letter A to F stand for the numeric value of 10 to 15, The digits in hexadecimal move from right to left in place value that are 16^0, 16^1, 16^2... anothers. For example, the hexadecimal number 1A3 can be represented in decimal as:

$$1 \times 16^2 + A \times 16^1 + 3 \times 16^0 = 1 \times 256 + 10 \times 16 + 3 \times 1 = 256 + 160 + 3 = 419_{10}$$

The hexadecimal number system is not directly used by computers either. However, it provides an even more compact representation than octal. This makes it easier for us to read and write large binary numbers.

This is because the hexadecimal system is base-16 and the binary system is base-2, therefore every single hexadecimal digit equals four binary bits. This relationship stems from the fact that 16 is a power of 2 ($16 = 2^4$). This means that any hexadecimal number between 0 and 15 then it can be converted into 4-bit binary number.

Table 2. 2 illustrates conversion of hexadecimal to binary digits. Each group of four bits corresponds to a single hexadecimal digit.

Example:

Therefore, the binary number 1101011010110010 equals to the hexadecimal number D6B2. In case a binary number cannot be grouped as four bits add zero(s) to the left of the number to make it fit.

1101 0110 1011 0010

Hexadecimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Table 2.2: Correspondence between Hexadecimal and Binary Digits

1101 = *D*

0110 = 6

1011 = *B*

0010 = 2

2.1.4.1 Converting Decimal to Hexadecimal

The following algorithm converts a decimal number to hexadecimal:

1. Convert the decimal number to an absolute value by dividing it by 16.
2. Record the quotient and the remainder.
3. Continue dividing the quotient by 16 and write down the remainder until the quotient is zero.
4. The hexadecimal number, as you might have guessed, is the remainder read from bottom to top.

Example: Convert 2297 to hexadecimal

2297 / 16 = 143 remainder 9
143 / 16 = 8 remainder F
8 / 16 = 0 remainder 8

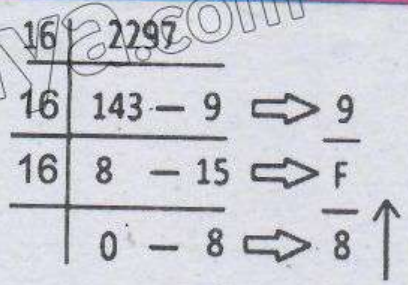


Figure 2.3: Decimal to Hexadecimal

The above steps are graphically shown in Figure 2.3. Reading the remainders from bottom to top gives the required result, i.e., 8F9 in hexadecimal.

Class activity

Find the following values and express them in hexadecimal. Discuss your findings with your classmates:

- Minimum Age to Cast Vote
- Length of the Indus River
- Total Districts in Pakistan
- Height of K2 (the second-highest mountain in the world)
- Area of Pakistan

2.2 Data Representation in Computing Systems

Computers can process and store a lot of information. In the following section we will discuss numeric data representation.

2.2.1 Binary Encoding of Integers (Z) and Real Numbers (R)

When we store data in computers, especially numbers, it's important to understand how they are represented and stored in memory. Let's explore how different sizes of integer values are stored in 1, 2, and 4 bytes, and how both positive and negative integers are handled.

2.2.2 Whole Numbers (W) and Integers (Z)

Integers, also known as whole numbers, are important elements in both mathematics and computer science. Knowledge of these concepts is important for primary computations, solving problems through programming, working with data and designing algorithms.

2.2.2.1 Whole Numbers (W)

Whole numbers are a set of non-negative integers. They include zero and all the positive integers. Mathematically, the set of whole numbers is:

$W = \{0,1,2,3,...\}$

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.

A 1-byte integer has 8 bits to store values. If all 8 bits are on, it represents the maximum value, 11111111_2, which is 255_{10}. If all bits are off, it represents the minimum value, 00000000_2, which is 0_{10}. Similarly, using 2 or 4 bytes, we get more bits to store data allowing us to store bigger values. If *n* is the number of bits, the maximum value that can be represented is 2^n -1 for examples:

- 1-Byte whole number (8 bits): Maximum value = 2^8-1 = 255
- 2-Byte whole number (16 bits): Maximum value = 2^{16} - 1 = 65,535
- 4-Byte whole number (32 bits): Maximum value = 2^{32}-1 = 4,294,967,295

2.2.2.2 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 stored 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 compliment, explained in the following section.

2.2.2.3 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_2 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 = $-2^{31}-1 = -2,147,483,648$

The reason we use binary and these ranges is that computers use transistors that have two states: ON (1) and OFF (0). This binary system forms the foundation of all digital computing!

Top Tip: When working with different integer types, always check whether the data type is signed or unsigned to avoid unexpected results, especially when dealing with large values or negative numbers.

Understanding how integers are stored in memory helps you appreciate the inner workings of computers and ensures you can effectively work with different data types in programming.

2.3 Storing Real Values in Computer Memory

In computers, real values, also known as floating-point numbers, are used to represent number with fractions and/or decimals.

2.3.1 Understanding Floating-Point Representation

Floating-point numbers (real values) are represented similarly to scientific notation as given below:

A floating-point number = sign x mantissa x 2^{exponent}. According to the above formula, 5.75 is represented as 1.4375 x 2. To convert the fractional part of a real (floating-point) number from decimal (base-10) to binary (base-2), multiply the fractional part by 2 and write down the integral part of the result. Repeat this process with the new fractional part until the value of the fractional part becomes zero or until the required precision is achieved.

Steps for Conversion:

1. Identify the Fractional Part: Get the fractional part of the decimal number. For instance, in the number 4.625, the integral part is 4 and the fractional part is 0.625.
2. Convert the Fractional Part to Binary: Multiply the fractional part by 2, and write down the integer that is obtained. Repeat this process with the new fractional part till it gets to 0 or until then required number of decimal places is achieved.

Example: Converting 0.375 to Binary

1. Identify the Fractional Part: Fractional part: 0.375
2. Convert the Fractional Part 0.375 to Binary:

$0.375 \times 2 = 0.75$ (Integer part: 0)

$0.75 \times 2 = 1.5$ (Integer part: 1)

$0.5 \times 2 = 1.0$ (Integer part: 1)

The integer parts recorded are 0, 1, 1.

3. Combine the Results: Combine the binary representations of the integer parts from top to bottom:

$$0.375_{10} = 0.011_{2}$$

In computing, it is critical to express real numbers in a binary form since it facilitates computing and storage. This process involves converting both the integer (decimal) and the fractional parts of a given number into binary. Two commonly use standards for this representation are "Single precision (32-bit)" and "Double Precision (64-bit)".

2.3.1.1 Single Precision (32-bit)

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.

Here the exponent can be ranged between -126 and +127.

The approximate range of values from 1.4×10^{-45} to 3.4×10^{38}.

Value	Representation	Sign Bit	Exponent (8 bits)	Mantissa (23 bits)
Grouping		1 bit	8 bits	23 bits
5.75	1.4375×2^{2}	0	10000001	10111000000000000000
-5.75	-1.4375×2^{2}	1	10000001	10111000000000000000
0.15625	1.25×2^{-3}	0	01111101	01000000000000000000
-0.15625	-1.25×2^{-3}	1	01111101	01000000000000000000

Table 2.3: 32-bit Floating Point Representation

Explanation:

Table 2.3 illustrates how 32-bit floating point values are represented in binary form. Each floating point value is broken down into three main components: the sign bit, the exponent, and the mantissa.

1. **Grouping:** This row explains the bit allocation for the 32-bit floating point format: 1 bit for the sign, 8 bits for the exponent, and 23 bits for the mantissa.

1. **5.75:** Representation: 1.4375×2^2 - Sign Bit: 0 (positive) - Exponent: 2 +127 = 129, which is 10000001_2 - Mantissa: The binary representation of 0.4375 is $10111000000000000000000_2$
2. **-5.75:** Representation: -1.4375×2^2 - Sign Bit: 1 (negative) - Exponent: 2 + 127 = 129, which is 10000001_2 - Mantissa: The binary representation of 0.4375 is $10111000000000000000000_2$
3. **0.15625:** Representation: 1.25×2^{-3} - Sign Bit: 0 (positive) - Exponent: -3 +127 = 124, which is 01111101_2 - Mantissa: The binary representation of 0.25 is $01000000000000000000000_2$
4. **-0.15625:** Representation: -1.25×2^{-3} - Sign Bit: 1 (negative) - Exponent: -3 +127 = 124, which is 01111101_2 - Mantissa: The binary representation of 0.25 is $01000000000000000000000_2$

This breakdown helps illustrate how floating point values are stored and manipulated in computer systems

2.3.1.2 Double Precision (64-bit)

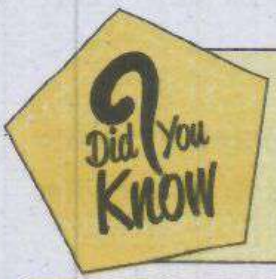
In double precision, the exponent is represented using 11 bits. The exponent is stored in a biased form, with a bias of 1023. The range of the actual exponent values can be determined as follows:

- **Bias:** 1023
- **Exponent range:** The actual exponent values range from -1022 to +1023.

Therefore, the smallest and largest possible exponent values in double-precision are:

- **Minimum exponent:** -1022
- **Maximum exponent:** +1023

We can perform the same steps given for the single-precision, except the difference of the abovementioned values.



The smallest positive number representable in single precision is approximately 1.4×10^{45} and in double precision is approximately 4.9×10^{-324}!

Tidbits

When performing computation with floating point values one should also consider possible round off errors. In scientific computing, it is necessary to monitor these errors to maintain the accuracy.

Class activity

1. Write down the binary representation of the following decimal numbers: 2. 5,7. 25, and 10.5.
2. Then, convert these binary representations to the format single precision format.
3. After completing this operation, discuss with classmates and yourself how the degree of accuracy of the representation differs based on the size of the number?

The information about how real values is stored in computer memory help us understand the precision and limitations of digital computation. With this understanding of floating-point representation, it becomes possible to control and manipulate these numbers in different ways.

2.4 Binary Arithmetic Operations

Arithmetic operations include addition, subtraction, multiplication and division, and are performed on two numbers at a time. Binary arithmetic operations are similar to decimal operations but follow binary rules. Here's a brief overview of the basic operations:

2.4.1 Addition

Binary addition uses only two digits: 0 and 1. Here, we will learn how to add binary numbers and how to handle the addition of negative binary numbers.

Binary Addition Rules

Binary addition follows these simple rules:

1. 0 + 0 = 0
2. 0 + 1 = 1
3. 1 + 0 = 1
4. 1 + 1 = 0 (with a carry of 1 to the next higher bit)

Example of Binary Addition

Example 1:

```
  1101
 +1011
------
 11000
```

In this example:

- 1 + 1 = 0 (carry 1)
- 0 + 1 + 1 (carry) = 0 (carry 1)
- 1 + 0 + 1 (carry) = 0 (carry 1)
- 1 + 1 + 1 (carry) = 1 (carry 1)

2.4.2 Subtraction

In binary arithmetic, subtraction can also be carried out by adding the two's complement or the value of the subtrahend to the minuend.

Example: Subtract 6 from 9 in Binary

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

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

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

- Invert the bits of 0110_2:
  Inversion: 1001_2
- Add 1 to the inverted number:
  $1001_2 + 1_2 = 1010_2 = -6_{10}$

Step 2: Add the Minuend and the Two's Complement of the Subtrahend

$1001_2 + 1010_2 = 10011_2$

Step 3: Discard the Carry Bit

10011_2 Discard carry $0011_2 = 3_{10}$

So, 9 - 6 = 3.

2.4.3 Multiplication

Binary numbers are base-2 numbers, consisting of only 0s and 1s. Multiplying binary numbers follows similar principles to multiplying decimal numbers, but with simpler rules. Here, we will learn how to multiply binary numbers with example.

Steps to Multiply Binary Numbers

1. Write down the binary numbers, aligning them by the least significant bit (rightmost bit).
2. Multiply each bit of the second number by each bit of the first number, similar to the long multiplication method in decimal.
3. Shift the partial results one place to the left for each new row, starting from the second row.
4. Add all the partial results to get the final product.

Example

Let's multiply two binary numbers: 101_2 and 11_2.

```
  101
x 11
-----
  101        (This is 101₂ x 1₂)
 101x    0   (This is 101₂ X 1₂, shifted left)
-----
 1111
```

So, $101_2 X 11_2 = 1111_2$.

Did You Know

The Central Processing Unit (CPU) of a computer performs millions of binary multiplications every second to execute complex instructions and run programs!

2.4.3 Division

Binary division is similar to decimal division but only involves two digits: 0 and 1. It follows steps like comparing, subtracting, and shifting, akin to long division in the decimal system.

Steps of Binary Division

1. **Compare:** Compare the divisor with the current portion of the devidend.
2. **Subtract:** Subtract the divisor from the dividend portion if the divisor is less than or equal to the dividend.
3. **Shift:** Shift the next binary digit from the dividend down to the remainder.
4. **Repeat:** Repeat the process until all digits of the dividend have been used.

Example

Divide 1100_2 by 10_2

```
10 | 1100 | 110
   -10
   ----
    10
   -10
   ----
     0
```

(Step 1: Compare 10 with first two 11, subtract 10 from 11)

(Step 2: Bring down the next digit 0)

(Step 3: Compare 10 with 10, subtract 10 from 10)

(Step 4: Bring down the next digit 0, no more digits left)

Result: $1100_2 / 10_2 = 110_2$

Class activity

Practicing Binary Division

Objective: To practice and understand binary division through hands-on examples.

Instructions:

1. Form groups of three to four students.
2. Each group will solve the following binary division problems:
   (a) $10101_2 \div 10_2$
   (b) $11100_2 \div 11_2$
   (c) $100110_2 \div 101_2$
3. Write down each step of your division process clearly.
4. Present your solutions to the class, explaining each step and the reasoning behind it.

2.5 Common Text Encoding Schemes

Text encoding schemes are essential for representing characters from various languages and symbols in a format that computers can understand and process. Here are some of the most common text encoding schemes used in computers:

2.5.1 ASCII

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 as shown in Table 2.4.

ASCII enables different computers and devices to exchange text information reliably. Let's encode the name of our country using ASCII.

- The ASCII code for an upper case letter "P" is 80.
- The code for letter 'a' in ASCII is 97.
- The ASCII code for the letter 'k' is 107.
- It is interesting to know that the ASCII code for the letter 'i' is 105.
- In the ASCII code system, the letter 's' has a code of 115.
- The code for 't' is 116 in ASCII.

The ASCII code is a numerical representation of characters in computer-based system, particularly for alphabetic characters.

For example, the ASCII code of the character 'n' is 110.

Class activity

1. Write down your name.
2. Find the ASCII code for each letter in your name. You can use the ASCII tableto for your help.
3. Convert each ASCII code to binary.
4. Write down your name in binary!

Character	ASCII Code	Character	ASCII Code
SP (space)	32	!	33
"	34	#	35
$	36	%	37
&	38	'	39
(	40	)	41
*	42	+	43
,	44	-	45
.	46	/	47
0	48	1	49
2	50	3	51
4	52	5	53
6	54	7	55
8	56	9	57
:	58	;	59
<	60	=	61
>	62	?	63
@	64	A	65
B	66	C	67
D	68	E	69
F	70	G	71
H	72	I	73
J	74	K	75
L	76	M	77
N	78	O	79

P	80	Q	81
R	82	S	83
T	84	U	85
V	86	W	87
X	88	Y	89
Z	90	[	91
\	92	]	93
^	94	-	95
?	96	a	97
b	98	c	99
d	100	e	101
f	102	g	103
h	104	i	105
j	106	k	107
l	108	m	109
n	110	o	111
p	112	q	113
r	114	s	115
t	116	u	117
v	118	w	119
x	120	y	121
z	122	{	123
|	124	}	125
~	126	DEL	127

Table:2.4

2.5.1 Extended ASCII

While the standard ASCII Table includes 128 characters, there is an extended version that includes 256 characters. This extended ASCII uses 8 bits and includes additional symbols, accented letters, and other characters. However, the original 128 characters are the most commonly used and serves as the basis for text representation in computers.

2.5.2 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.

2.5.2.1 UTF-8

It is a variable-length encoding scheme, meaning it can use a different numbers 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.

2.5.2.2 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.

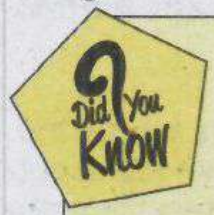
2.5.2.3 UTF-32

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.

2.6 Storing Images, Audio, and Video in Computers

Have you ever wondered how your favorite photos, songs, and movies are stored on your computer or phone? Let's dive into the fascinating world of digital storage to understand how computers manage these different types of files.



Data size is usually expressed in byte and its multiples.

- 1 Byte (B) = 8 Bits
- 1 Kilobyte (KB) = 1024 Bytes
- 1 Megabyte (MB) = 1024 Kilobytes
- 1 Gigabyte (GB) = 1024 Megabytes
- 1 Terabyte (TB) = 1024 Gigabytes
- 1 Petabyte (PB) = 1024 Terabytes
- 1 Exabyte (EB) = 1024 Petabytes
- 1 Zettabyte (ZB) = 1024 Exabytes
- 1 Yottabyte (YB) = 1024 Zettabytes

2.6.1 Storing Images

Images are made up of tiny dots called **pixels**. Each pixel has a color, and the combination of all these pixels forms the complete picture. Computers store images using numbers to represent these colors.

Color Representation: - In a color image, each pixel's color can be represented by three numbers: Red, Green, and Blue (RGB). Each of these numbers typically ranges from 0 to 255. - For example, a pixel with RGB values (255, 0, 0) will be bright red.

Image File Formats: The following are Commonly used image formats for photos - **JPEG** (Joint Photographic Expert Group). It compresses the image to save space but might lose some quality. - **PNG** (Portable Network Graphics): Supports transparency and maintains high quality without losing data. - **GIF** (Graphics Interchange Format): Used for simple animations and images with few colors.

Class activity

Create a Pixel Art

1. Use graph paper to draw a simple image, such as a smiley face.
2. Color each square (pixel) and write down the RGB values for each color used.
3. Share your pixel art and RGB values with the class.

2.6.4 How Computers Store These Files

All these files (images, audio, and video)are stored as **binary data**, which means they are represented by sequences of 0s and 1s.

Storage Devices:

- **Hard Disk Drive (HDD)**: Uses spinning disks to read/write data. They offer large storage capacities.
- **Solid State Drive (SSD)**: Uses flash memory for faster access times and better performance.
- **Cloud Storage**: Stores files on remote servers accessible via the internet, providing flexibility and backup options.

IBM created the first hard drive in 1956 which weighed over a ton and could only store 5,000,000 bytes which is much less than the storage required for even one high-quality song today.

This leads us to an understanding and appreciation of how images, audio and videos are stored in the computers, allowing us to marvel at the underlying technology of our current digital age. Whether you're taking pictures, enjoying music, or watching films, it all stems how computers manage information!

Summary

- In computing, numbering systems are crucial as they form the foundation for representing, storing, and processing information.
- Decimal number system is a number system in which base is 10 and the digits involved are 0 to 9, which are commonly used in our daily lives.
- Binary is a base-2 number system that comprises of only the digits 0 and 1. Each digit represents a power of two.
- The Octal number system is another number system that has eight as its base; thus, it has eight digits 0 to7. Each digit represents a power of 8, this can be expressed as 8 digit.
- The Hexadecimal numbering system is another type of number system with base of 16, where the number 0 to 9 and alphabets A-F are used.
- Integers refers to the set of non-negative whole numbers, while whole numbers are the complete numbers. They include zero and all the positive integers, also positive zero.
- To store negative values, computers employ a technique commonly known as two's complement.
- In computers, real values, which are nicknamed as floating-point numbers are used to represent numbers with fraction or decimal point.
- Arithmetic operations mean addition, subtraction multiplication, and division performed on numbers in given base. Binary arithmetic involves performing these operations on numbers in binary form, or base 2.
- ASCII is an acronym for American Standard Code for Information Interchange. It is an industry standard used to encode text in computers and

6. What is the 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) 1.4×10^{-45} to 3.4×10^{38} (b) 1.4×10^{-38} to 3.4×10^{45}
   (c) 4.9×10^{-324} to 1.8×10^{308} (d) 4.9×10^{-308} to 1.8×10^{324}
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

Short Questions

1. What is the primary purpose of the ASCII encoding scheme?
2. Explain the difference between ASCII and Unicode.
3. How does Unicode handle characters from different languages?
4. What is the range of values for an unsigned 2-byte integer?
5. Explain how a negative integer is represented in binary.
6. What is the benefit of using unsigned integers?
7. How does the number of bits affect the range of integer values?
8. Why are whole numbers commonly used in computing for quantities that cannot be negative?
9. How is the range of floating-point numbers calculated for single precision?
10. Why is it important to understand the limitations of floating-point representation in scientific computing?

Long Questions

1. Explain how characters are encoded using Unicode. Provide examples of characters from different languages and their corresponding Unicode code points.
2. Describe in detail how integers are stored in computer memory.
3. Explain the process of converting a decimal integer to its binary representation and vice versa. Include examples of both positive and

negative integers.

4. Perform the following binary arithmetic operations:
   a. Multiplication of 101 by 11.
   b. Division of 1100 by 10.

6. Add the following binary numbers:

a)

$$\begin{array}{r} 101 \\ +\ 110 \\ \hline \end{array}$$

b)

$$\begin{array}{r} 1100 \\ +\ 1011 \\ \hline \end{array}$$

7. Convert the following numbers to 4-bit binary and add them:
   (a) 7 + (-4)
   (b) -5 + 3

8. Solve the following
   (a) $1101_2 - 0100_2$
   (b) $1010_2 - 0011_2$
   (c) $1000_2 - 0110_2$
   (d) $1110_2 - 100_2$