1 Basic Theories of Information

Chapter Objectives

Understanding the computer mechanism of data representation and basic theories.
In particular, the binary system is an important subject to learn, indispensable for computer data representation. However, for people who are used to the decimal system, it is hard to become familiar with this representation, so it should be carefully studied.

① Understanding a computer's basic data units such as binary numbers, bits, bytes, words, etc. and their conversion from and to decimal and hexadecimal digits.
② Understanding basic concepts of computer internal data representation, focusing on numeric data, character codes, etc.
③ Understanding proposition calculus and logical operations.

Introduction

In order to make a computer work, it is necessary to convert the information we use in daily life into a format that can be understood by the computer. For that reason, the way information is actually represented inside a computer as well as the way it is processed will be learned here.

1.1 Data representation

1.1.1 Numeric conversion

For a computer to do processing it is first necessary to input into the memory the programs which are contents of a task or processing procedures. The binary system is what is used to represent this information. While the binary system represents information by means of the combination of "0" and "1," we ordinarily use the decimal system. Therefore, an important fundamental knowledge required by information processing engineers is to understand the relationship between binary and decimal numbers. This is the basic difference between computers and human beings as well as the point of contact between them.
Since the mechanism in which the computer operates is completely based on binary numbers, the relationship between binary and decimal numbers, as well as hexadecimal numbers combining binary numbers will be explained here.

(1) Data representation unit and processing unit

① Binary numbers

The internal structure of a computer is composed of an enormous number of electronic circuits. Binary numbers represent two levels of status in the electronic circuits, as in:

- Whether the electric current passes through it or not
- Whether the voltage is high or low

For example, setting the status where the electric current passes through (the power is on) to "1" and the status where the electric current does not pass through (the power is off) to "0," then by replacing the computer status or data with numerical values their representation can be easily performed in an extremely convenient way.
The representation of decimal numbers from "0" to "10" using binary numbers is shown in Figure 1-1-1.

Figure 1-1-1
Decimal numbers and binary numbers

	Decimal numbers	Binary numbers	
	0	0	
	1	1	
	2	10	A carry occurs
	3	11	
	4	100	A carry occurs
	5	101	
	6	110	
	7	111	
	8	1000	A carry occurs
	9	1001	
A carry occurs	10	1010	

As can be seen in this Figure, compared to the decimal system, a carry occurs more frequently in the binary system, but since besides "0" and "1," no other figure is used, it is the most powerful tool for the computer.

② Bits

A bit (binary digit) is 1 digit of the binary system represented by "0" or "1." A bit is the smallest unit that represents data inside the computer. 1 bit can represent only 2 values of data, "0" or "1," but 2 bits can represent 4 different values:

- 00
- 01
- 10
- 11

However, in practice, the amount of information processed by a computer is so immense (there are 26 values in the English alphabet alone) that the two bits, 0 and 1, are insufficient for an information representation method.

③ Bytes

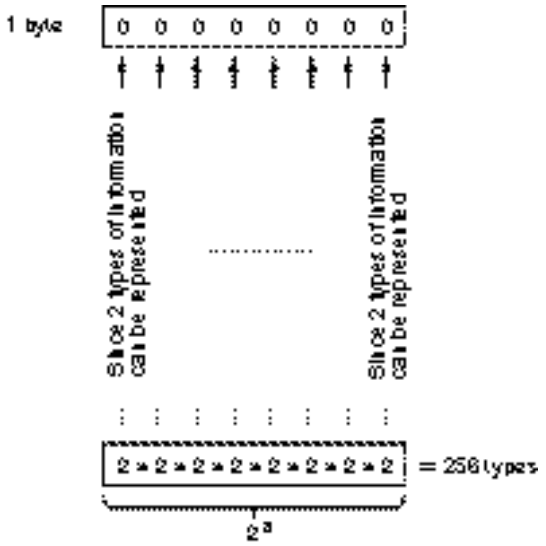
Compared to a bit, which is the smallest unit that represents data inside the computer, a byte is a unit that represents with 8 bits 1 character or number. Considering that a byte is equal to 8 bits, the following is the information which can be represented with one byte, by the combination of "0" and "1."

- 00000000
- 00000001
- 00000010

  →
- 11111101
- 11111110
- 11111111

The information represented by a string of "1's" and "0's" is called a bit pattern. Since 1 bit can be represented in two ways, the combination of 8 bit patterns into 1 byte enables the representation of 2^8=256 types of information. In other words, besides characters and figures, symbols such as "+" and "-" or special symbols such as "<" and ">" can also be represented with one byte.

Figure 1-1-2
Types of information that can be represented with one byte



However, since the number of kanji (Chinese characters) amounts to thousands, they cannot be represented with one byte. Therefore, two bytes are connected to get 16 bits, and one kanji is represented with two bytes. With 16 bits, 216 = 65,536 kanji can be represented.

④ Words

A bit is the smallest unit that represents data inside a computer and a byte is a unit that represents 1 character. However, if the computers' internal operations were performed on the bit basis, the operation speed would be too low. For that reason the idea of processing using a unit called word was born.

Over 10 years ago, personal computers operated on words each consisting of 16 bits. Currently mainstream PGs use words each consisting of 32 bits.

⑤ Binary system and hexadecimal system

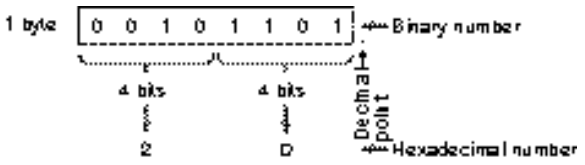
In information processing, the binary system is used to simplify the structure of the electronic circuits that make up a computer. However, for us, the meaning of string of "0's" and "1's" is difficult to understand. In the decimal system, the numeric value "255" has 3 digits, but in the binary system the number of digits becomes 8. Therefore, in order to solve the problem of difficulty in identification and of a large number of digits hexadecimal system is used.
A hexadecimal number is a numeric value represented by 16 numerals, from "0" to "15." When it becomes 16, a carry occurs. However, since it cannot distinguish between the "10" before a carry has been generated, and the "10" after a carry has been generated, for purposes of convenience, in the hexadecimal system "10" is represented by the letter "A," "11" by "B," "12" by "C," "13" by "D," "14" by "E" and "15" by "F."

Figure 1-1-3
Decimal numbers, binary numbers, and hexadecimal numbers

Decimal numbers	Binary numbers	Hexa-decimal numbers
0	0	0
1	1	1
2	10	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F
16	10000	10
17	10001	11
18	10010	12
19	10011	13
20	10100	14

Figure 1-1-3 shows the notation of the numbers "0" to "20" of the decimal system in the binary system and the hexadecimal system.
Focusing on the relationship between the hexadecimal numbers and binary numbers in this table, it can be noted that 4 digits in the binary system correspond to 1 digit in the hexadecimal system. Therefore, binary numbers can be converted to hexadecimal numbers, by replacing each group of 4 bits with a hexadecimal digit, starting from the decimal point. (Figure 1-1-4)

Figure 1-1-4
Binary, and hexadecimal counting systems



(2) Representation of numeric data

By means of the combinations of "0's" and "1's," characters are represented as codes. However, a different representation method is used to process numeric data. Here, the radix and radix conversion, the addition and subtraction of binary numbers and hexadecimal numbers, the representation of negative numbers, among other points considered basic for the representation of numeric data, will be explained.

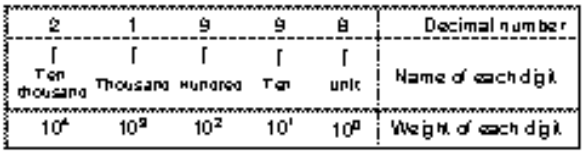
① Radix and "weight"

a. Decimal numbers' "weight" and its meaning

When quantities are represented using decimal numbers, 10 types of numerals from "0" to "9" are combined. Each of them, from the lower digit in the ascendant order has a "weight" as 10^0, 10^1, 10^2, 10^3... (Figure 1-1-5).
For example, using the weight, a decimal number 1,234 would be represented as follows:

$$1{,}234 = 1 \times 10^3 + 2 \times 10^2 + 3 \times 10^1 + 4 \times 10^0$$

Figure 1-1-5
Weight of each digit of the decimal number 21998

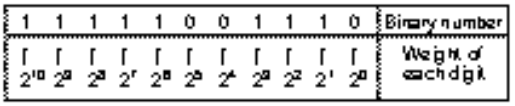


In Figure 1-1-5 the weight of each digit is represented as 10^0, 10^1, 10^2, 10^3,... this "10" is called "Radix" and the value placed at the upper right of 10 is called the "Exponent." The notation and meaning of the weight in the decimal system is explained below.
In 10^0, the radix 10 is multiplied 0 times by 1, so it becomes 1, in 10^1, the radix 10 is multiplied 0 times by itself, so it becomes 10.
Likewise, in 10^2, 10 is multiplied 2 times by itself, so it becomes 100; in 10^3, 10 is multiplied 3 times by itself, so it becomes 1,000.
In this way, even when the number of digits increases, it can be easily represented by writing down in small numbers, to the upper right of 10, the numeric value that indicates the number of times the radix 10 is multiplied (exponent).

b. Binary digits "weight" and its meaning

The radix of the decimal system is 10, and the radix of the binary system is 2. As in the decimal system, the weight of each digit in the binary system is shown in Figure 1-1-6.

Figure 1-1-6
Weight of each digit of the binary number 11111001110



The notation and meaning of the weight in the binary system is explained below.
In 2^0, the radix 2 is multiplied 0 times by itself, so it becomes 1, in 2^1, the radix 2 is multiplied only 1 time by itself, so it becomes 2. Likewise, in 2^2, 2 is multiplied 2 times by itself, so it becomes 4.
To verify that the decimal number 1,988 is represented as "11111001110" in the binary system, the weight of each of the digits represented by 1 in the binary representation should be added, as is shown below:

$$\begin{array}{ccccccccccccccccccc}
1 & & 1 & & 1 & & 1 & & 1 & 0 & 0 & & 1 & & 1 & & 1 & 0 \\
\downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow & & & & \downarrow & & \downarrow & & \downarrow & \\
2^{10} & + & 2^9 & + & 2^8 & + & 2^7 & + & 2^6 & & & + & 2^3 & + & 2^2 & + & 2^1 & \\
\downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow & & & & \downarrow & & \downarrow & & \downarrow & \\
= 1{,}024 & + & 512 & + & 256 & + & 128 & + & 64 & & & + & 8 & + & 4 & + & 2 & \\
= 1{,}998 & & & & & & & & & & & & & & & & & &
\end{array}$$

② Auxiliary units and power representation

Since the amount of information processed by computers is immense, auxiliary units that represent big amounts are also used.
Likewise, since computers operate at high speeds, auxiliary units that represent extremely small amounts are also needed to represent the performance.
Figure 1-1-7 shows the auxiliary units that represent large and small amounts as well as the exponent to which the radix is raised.

Figure 1-1-7
Auxiliary units

	unit symbol	Exponent notation	Remarks
units that represent large amounts	T (giga)	10^{12}	$\fallingdotseq 2^{40}$
	G (tera)	10^{9}	$\fallingdotseq 2^{30}$
	M (mega)	10^{6}	$\fallingdotseq 2^{20}$
	k (kilo)	10^{3}	$\fallingdotseq 2^{10}$
units that represent small amounts	m (milli)	10^{-3}	$\frac{1}{1,000}$
	μ (micro)	10^{-6}	$\frac{1}{1,000,000}$
	n (nano)	10^{-9}	$\frac{1}{1,000,000,000}$
	p (pico)	10^{-12}	$\frac{1}{1,000,000,000,000}$

It is important to note that, as indicated in the Remarks column in Figure 1-1-7, kilo is equal to 10^3, but it is also almost equal to 2^{10}. In other words, the kilo we ordinarily use is equal to 1,000, however, since the binary system is used in computing, 2^{10} (1,024) is a kilo. Furthermore, if 2^{10} and 10^3 are almost equal, 10^6 that is a mega, is almost equal to 2^{20} and 10^9 a giga, is almost equal to 2^{30}.
Therefore, when it is said that a computer memory capacity is 1 kilobyte, strictly speaking, 1 kilobyte does not mean 1,000 bytes, but 1,024 bytes.

③ Addition and subtraction of binary numbers

a. Addition

The following are the 4 basic additions of the binary system:

- 0 + 0 = 0 (0 in the decimal system)
- 0 + 1 = 1 (1 in the decimal system)
- 1 + 0 = 1 (1 in the decimal system)
- 1 + 1 = 10 (2 in the decimal system) ← Main characteristic of the binary system that differs from the decimal system

Among these additions, a carry is generated in 1 + 1 = 10.

```
 [1]   ← Carry
  1
+ 1
---
 1 0
```

Example $(11010)_2 + (1100)_2$

```
 [1 1]   ← Carry
  1 1 0 1 0
+     1 1 0 0
-------------
  1 0 0 1 1 0
```

The result is $(100110)_2$.

b. Subtraction

The following are the 4 basic subtractions of the binary system:

- 0 – 0 = 0
- 0 – 1 = –1
- 1 – 0 = 1
- 1 – 1 = 0

Among these subtractions, if the upper digit of 0 is 1 in 0 – 1 = –1, a "borrow" is performed.

```
 ♥      ← Borrow
 1 0
– 1
---
  1
```

Example $(10011)_2 - (1001)_2$

♥ ← Borrow

```
  1 0 0 1 1
– 1 0 0 1
-----------
  1 0 1 0
```

The result is $(1010)_2$.

④ Addition and subtraction of hexadecimal numbers

Basically, the addition and subtraction of hexadecimal numbers is similar to that of decimal and binary numbers.

a. Addition

Addition is performed starting at the lowest (first from the left) digit. When the addition result is higher than 16, a carry to the upper digit is performed.

Example $(A8D)_{16} + (B17)_{16}$

```
  1   1   ← Carry
    A 8 D
+   B 1 7
---------
  1 5 A 4
```

```
    10   8   13
+   11   1    7
-----------------
    21   9   20
```

- First digit: D + 7 = (In the decimal system: 13 + 7 = 20) = 16 (carried 1) + 4
  The sum of the first column is 4 and 1 is carried to the second column.
- Second digit: 1 + 8 + 1 = (In the decimal system: 10) = A
  ↑ Carried from the first column
- Third digit: A + B = (In the decimal system: 10 + 11 = 21) = 16 (carried 1) + 5
  The sum of the third column is 5 and 1 is carried to the fourth column.

The result is $(15A4)_{16}$.

b. Subtraction

Subtraction is performed starting from the first column, and when the subtraction result is negative (minus), a borrow from the upper order column is performed.

Example $(6D3)_{16} - (174)_{16}$

♥ ← Borrow

```
    6 D 3
–   1 7 4
---------
    5 5 F
```

♥ 16

```
    6   13   3
–   1    7   4
---------------
    5    5  15
```

- First digit: Since 3 – 4 = –1, a borrow is performed from D in the second digit (D becomes C). 16 (borrowed 1) + 3 – 4 = F (In the decimal system: 19 – 4 = 15)
- Second digit: C – 7 = 5 (In the decimal system: 12 – 7 = 5)
- Third digit: 6 – 1 = 5

The result is $(55F)_{16}$.

(3) Radix conversion

In order to process numeric values in a computer, decimal numbers are converted into binary or hexadecimal numbers. However, since we ordinarily use decimal numbers, it would be difficult to understand the meaning of the result of a process if it were represented by binary or hexadecimal numbers.

Therefore, the conversion amongst decimal, binary and hexadecimal numbers is necessary. This operation is called radix conversion.
A concrete explanation of the conversion amongst the radixes of decimal, binary and hexadecimal numbers, which are currently used the most, will be performed below. In order to avoid confusion, the respective radix will be written outside the parenthesis to distinguish them. For example:

- Notation of binary numbers: $(0101)_2$
- Notation of decimal numbers: $(123)_{10}$
- Notation of hexadecimal numbers: $(1A)_{16}$

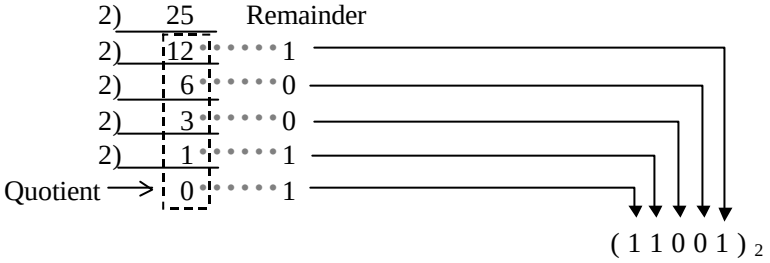
① Conversion of decimal numbers into binary numbers

The method of conversion for translating decimal numbers into binary numbers differs depending on whether the decimal number is an integer or a fraction.

a. Conversion of decimal numbers

The decimal integer is divided into 2, and the quotient and remainder are obtained. The resulting quotient is divided into 2 again, and the quotient and remainder are obtained. This operation is repeated until the quotient becomes 0.
Since a decimal integer is divided into 2, when the decimal integer is an even number the remainder will be "0," when it is an odd number the remainder will be "1." The binary digit is obtained by placing the remainder(s) in the reverse order.

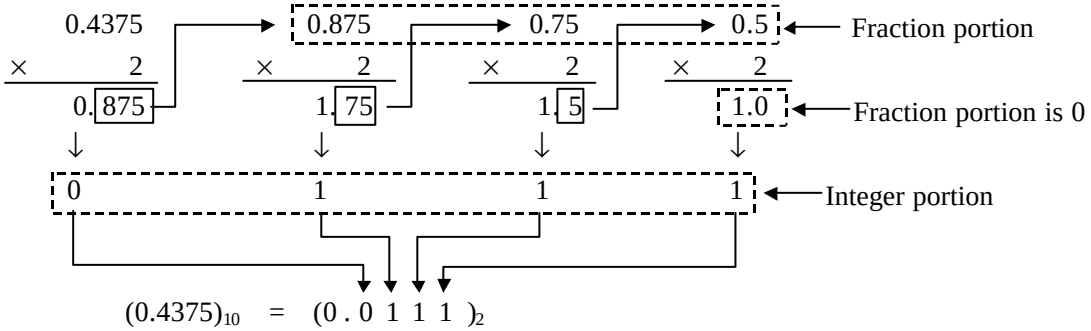
Example $(25)_{10}$



b. Conversion of decimal fractions

The decimal fraction is multiplied by 2, the integer and fraction portion of the product are separated, and the integer section is extracted. Since the integer portion is the product of the multiplication of the fraction portion by 2, it will always be "0" or "1." Next, setting aside the integer portion, only the fraction portion is multiplied by 2. This operation is repeated until the fraction portion becomes 0. The binary digit is obtained by placing the integer portions extracted in the order they were extracted.

Example $(0.4375)_{10}$



It should be noted that when decimal fractions are converted into binary fractions, most of the times, the conversion is not finished, since no matter how many times the fraction portion is multiplied by 2, it will not become 0. In other words, the above-mentioned example is that of a special decimal fraction, but most of the decimal fractions become infinite binary fractions.
The verification of the kind of numeric values which correspond to special decimal fractions is

performed below. For example, the result of the conversion of the binary fraction 0.11111 into a decimal fraction is as follows:

$$
\begin{array}{lccccc}
0. & 1 & 1 & 1 & 1 & 1 \quad \leftarrow \text{Binary fractions} \\
 & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
 & 2^{-1} & 2^{-2} & 2^{-3} & 2^{-4} & 2^{-5} \quad \leftarrow \text{Weight} \\
 & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
 & 0.5 & 0.25 & 0.125 & 0.0625 & 0.03125 = 0.96875 \quad \leftarrow \text{Decimal fractions}
\end{array}
$$

From this example it can be understood that besides the decimal fractions that are equal to the weight of each digit (0.5, 0.25, 0.125, ...etc.) or the decimal fractions that result from their combination, all other decimal fractions become infinite binary fractions.

② Conversion of binary numbers into decimal numbers

The conversion into decimal numbers is performed by adding up the weights of each of the "1" digits of the binary bit string.

a. Conversion of binary integers

Example $(11011)_2$

$$
\begin{array}{l}
(1\ 1\ 0\ 1\ 1)_2 \\
2^4 + 2^3 + 2^1 + 2^0 \quad \leftarrow \text{Weight} \\
\downarrow\ \ \downarrow\ \ \downarrow\ \ \downarrow \\
16 + 8 + 2 + 1 = (27)_{10}
\end{array}
$$

b. Conversion of binary fractions

Example $(1.101)_2$

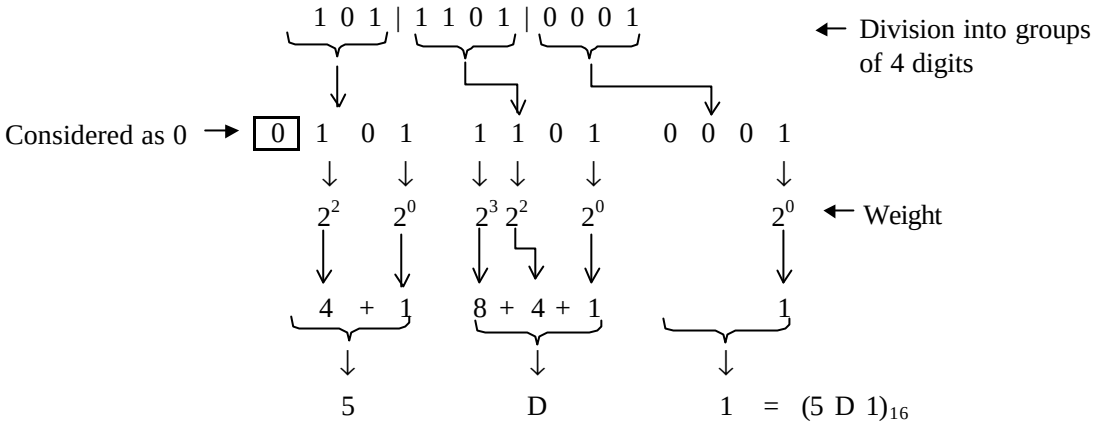
$$
\begin{array}{l}
(1\ .\ 1\ 0\ 1)_2 \\
2^0 + 2^{-1} + 2^{-3} \quad \leftarrow \text{Weight} \\
\downarrow\ \ \downarrow\ \ \downarrow \\
1 + 0.5 + 0.125 = (1.625)_{10}
\end{array}
$$

③ Conversion of binary numbers into hexadecimal numbers

Since 4-bit binary strings are equivalent to 1 hexadecimal digit, in binary integers, the binary number is divided into groups of 4 digits starting from the least significant digit. In binary fractions, the binary number is divided into groups of 4 digits starting from the decimal point. Then, the conversion is performed by adding up the weights of each of the binary digits displayed as "1," in each group of 4 bits. In the event that there is a bit string with less than 4 digits, the necessary number of "0's" is added and the string is considered as a 4-bit string.

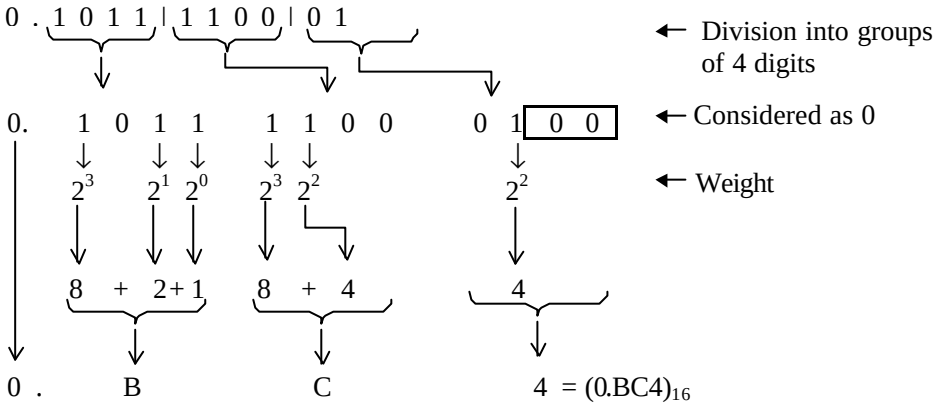
a. Conversion of binary integers

Example $(10111010001)_2$



b. Conversion of binary fractions

Example $(0.1011110001)_2$

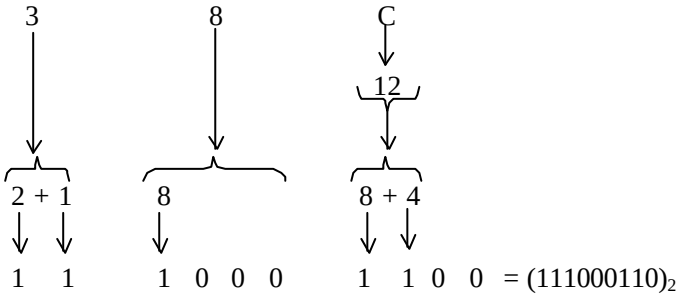


④ Conversion of hexadecimal numbers into binary numbers

Hexadecimal numbers are converted into binary numbers by performing the reverse procedure. In other words, 1 digit of the hexadecimal number is represented with a 4-digit binary number.

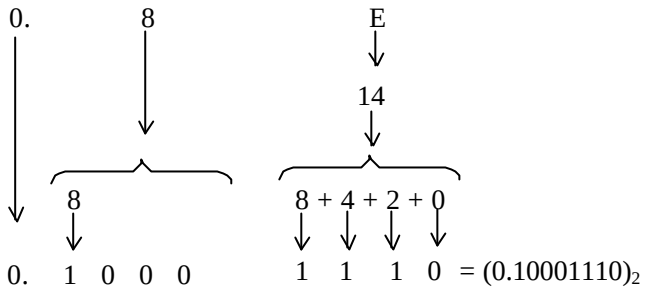
a. Conversion of hexadecimal integers

Example $(38C)_{16}$



b. Conversion of hexadecimal fractions

Example $(0.8E)_{16}$



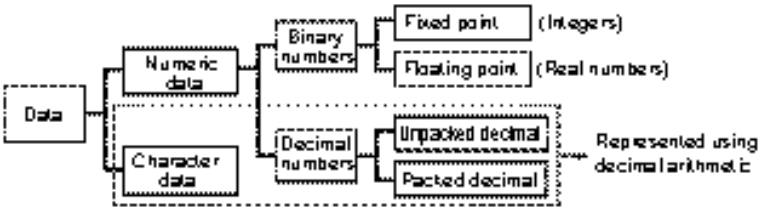
⑤ Conversion from decimal numbers into hexadecimal numbers and from hexadecimal numbers into decimal numbers

To convert them into binary numbers, decimal numbers are divided into 2, to convert them into hexadecimal numbers, and then they are divided into 16. Likewise, hexadecimal numbers are converted into decimal numbers by adding up the exponents whose radixes are 16.
It should be noted that due to the general unfamiliarity with the notation of hexadecimal numbers, ordinarily hexadecimal numbers are first converted into binary numbers to convert them into decimal numbers.

1.1.2 Numeric representation

In the computer, originally invented as a calculating machine, amongst other aspects involving the management of the data subject for processing, the precision and the easiness with which calculations can be performed have also been worked out. The representation format suitable for each type of data is explained here.

Figure 1-1-8
Data representation format



(1) Decimal digit representation

① Binary-coded decimal code

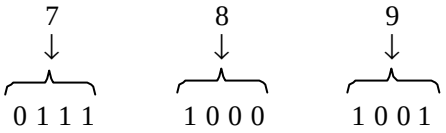
As a format of character data and decimal numbers, there is a representation method called binary-coded decimal code (BCD code) that, using 4-bit binary digits that correspond to the numbers 0 to 9 of the decimal system, represents the numeric value of each digit.

Figure 1-1-9 Binary-coded decimal code

Decimal number	Binary number	Binary-coded decimal code
0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1
2	0 0 1 0	0 0 1 0
3	0 0 1 1	0 0 1 1
4	0 1 0 0	0 1 0 0
5	0 1 0 1	0 1 0 1
6	0 1 1 0	0 1 1 0
7	0 1 1 1	0 1 1 1
8	1 0 0 0	1 0 0 0
9	1 0 0 1	1 0 0 1
10	1 0 1 0	0 0 0 1 0 0 0 0
11	1 0 1 1	0 0 0 1 0 0 0 1
⋮	⋮	⋮

Since the binary-coded decimal code is not a numeric value but a code, there are only 10 patterns, and the notation is performed by arranging the patterns for each digit.

For example, the representation of the decimal number "789" using the binary-coded decimal code would be as follows:



$(011110001001)_2$

In this way, as the number of digits of a decimal number increases, the length of the binary-coded decimal code increases as well (a group of 4 bits is added for each digit). This format is called variable-length format.

The same value as the binary-coded decimal code has also been set to the least significant 4 bits of the numeric characters of the EBCDIC, JISCII and other codes.

The binary-coded decimal code is mainly used for the numeric representation of office calculations, and according to the memory format of the computer, it is divided into unpacked decimal format and packed decimal format. And, since character codes as well as the unpacked decimal format or packed decimal format are represented by the use of the binary-coded decimal code, they are automatically processed using the decimal arithmetic system of the computer. It is not necessary for the user to be aware of this process.

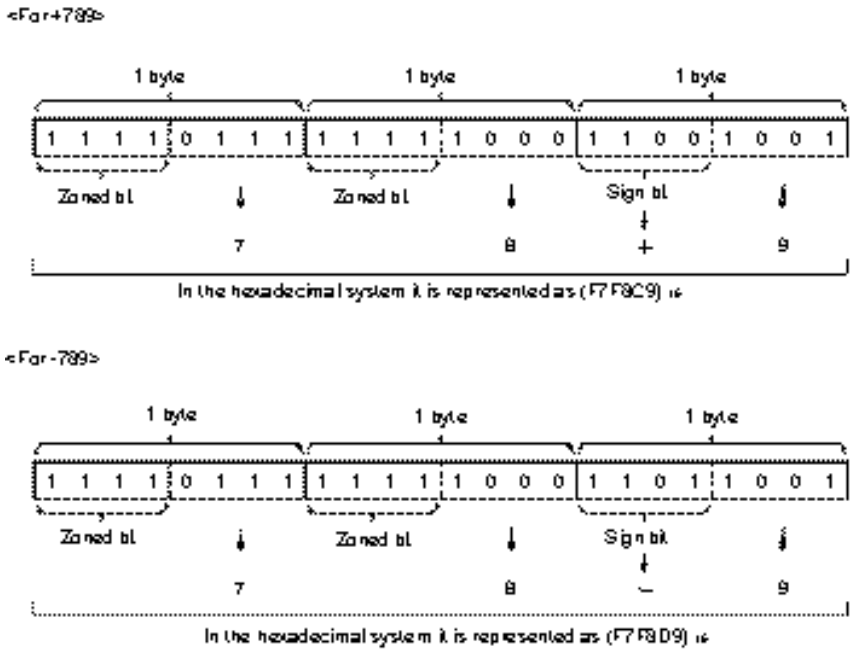
② Unpacked decimal format

When representing signed decimals, the unpacked decimal format uses 1 byte for each digit of the decimal number.

The unpacked decimal format represents the values from 0 to 9 in the least significant 4 bits of 1 byte, and in the most significant 4 bits, which are called zoned bits, in the case of the EBCDIC code used in high-end mainframe machines, where ordinarily $(1111)_2$ is stored. However, in the zoned bits of the least significant digit, the 4 bits that represent the sign are stored, in both the case of 0 and positive numbers, $(1100)_2$, and in the case of negative numbers, $(1101)_2$. In the JIS code used for data transmission as well as in the low-end machines, $(0011)_2$ is stored in the zoned bits. The unpacked decimal format is also called zoned decimal format.

The bit pattern of the representation of the decimal numbers +789 and –789 in the unpacked decimal format is shown in Figure 1-1-10.

Figure 1-1-10 Unpacked decimal format

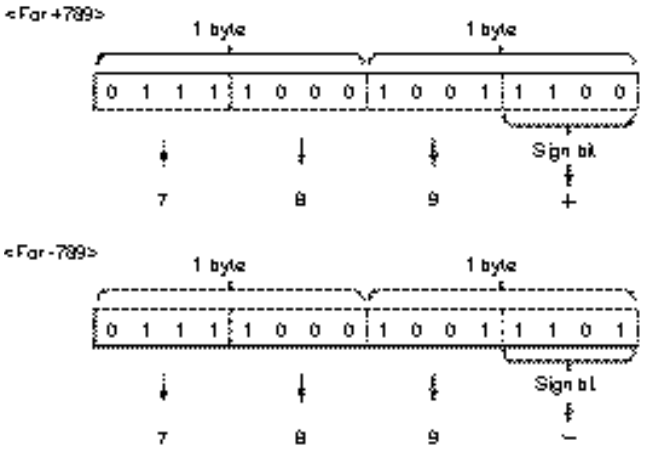


In the unpacked decimal format, excepting the least significant byte, only half of a byte is used. This was considered a waste of resources. This defect was eliminated by the packed decimal format.

③ Packed decimal format

In the packed decimal format, 1 byte represents a numeric value of 2 digits and the least significant 4 bits represent the sign. The bit pattern of the sign is the same as that of the unpacked decimal format, $(1100)_2$ for 0 and positive numbers, and $(1101)_2$ for negative numbers.
Figure 1-1-11 shows the bit pattern of the packed decimal format.

Figure 1-1-11 Packed decimal format



Compared to the unpacked decimal format, the packed decimal format has the following advantages:

- A numeric value can be represented by fewer bytes.
- The conversion into the binary system is easy.

(2) Binary representation

① Representation of negative integers

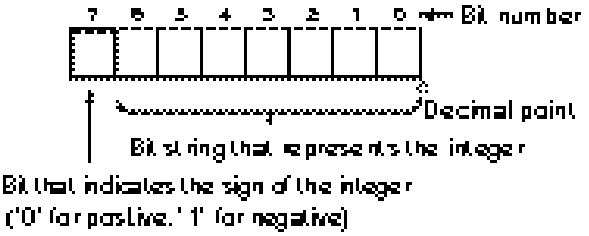
The typical example of methods to represent negative integers are mentioned below:

- Absolute value representation
- Complement representation

a. Absolute value representation of negative integers

As is shown in Figure 1-1-12, in the absolute value representation of negative integers, the first bit represents the sign and the 7 other bits represent the numeric value (absolute value).

Figure 1-1-12
Absolute value representation of negative integers



For example, in $(00001100)_2$, since the sign bit at the top is 0, it is a positive number. Likewise, since the 7 other bits, which are the absolute value of the numeric value, are $(0001100)_2 = 2^2 + 2^3 = (12)_{10}$, the decimal number 12 (positive number) is represented.
On the other hand, since in $(10001100)_2$ the sign bit at the top is 1, it is a negative number. The decimal number –12 = (negative number) is represented.
However, since in this representation method the numeric value 0 can be represented in two ways, as 00000000 (positive zero) or as 10000000 (negative zero), the operation becomes complicated, and for that reason it is not widely used.
It should be noted that when the absolute value representation of negative numbers is performed with 8 bits, the range of numeric values that can be represented is as follows (in decimal digits):

–127 to 127

b. Complement representation of negative integers

The complement is the number that indicates the quantity by which a numeric value falls short of a specific numeric value. There are 2 types of radix complements, the radix complement and the reduced radix complement.

- Decimal complement

There are 2 types of decimal complements, "10's complement" and "9's complement." For example, the 9's complement of a given numeric value will be the result of the subtraction of each of the digits of this numeric value from 9. Likewise, the 10's complement of a given numeric value will be the result of the subtraction of each of the digits of this numeric value from 10. As a result, the 10's complement is the result of the addition of 1 to the 9's complement.

Example "9's complement" of $(123)_{10}$

$$\begin{array}{r} 999 \\ -\ 123 \\ \hline 876 \end{array}$$

Example "10's complement" of $(123)_{10}$

$$\begin{array}{rl} 1000 & (= 999 + 1) \\ -\ 123 & \\ \hline 877 & \end{array}$$

- Binary complement

There are 2 types of binary complements, "1's complement" and "2's complement."

- 1's complement

The "1's complement" of a given numeric value will be the result of the subtraction of each of the digits of this numeric value from 1, as a result, all the "0" and "1" bits of the original bit string are switched.
For example, the "1's complement" of the bit string $(10110011)_2$ is shown below:

1 0 1 1 0 0 1 1
↓ ← All the "0" and "1" bits of the original bit string are switched
0 1 0 0 1 1 0 0 ←"1's complement"

- 2's complement
  "2's complement" is the "1's complement" bit string plus 1. Therefore, the "2's complement" of the bit string $(10110011)_2$ is obtained as follows:

1 0 1 1 0 0 1 1
↓ ← All the "0" and "1" bits of the original bit string are switched
0 1 0 0 1 1 0 0 ←"1's complement"
\+ 1 ←1 is added
0 1 0 0 1 1 0 1 ←"2's complement"

- "1's complement" and "2's complement" representation of negative integers
  - "1's complement" representation of negative integers
    - Sign bit: 0 for positive, 1 for negative, and both, +0 and -0, for 0
    - Numeric value: "1's complement"

For example, the "1's complement" representation of the decimal number -126 will be as follows:

0 1 1 1 1 1 1 0 ← + 126
Sign → ↓ ↓ ← All the "0" and "1" bits of the original bit string are switched
1 0 0 0 0 0 0 1 ← − 126

- "2's complement" representation of negative integers
  - Sign bit: 0 for positive and 0, 1 for negative
  - Numeric value: "2's complement"

For example, the "2's complement" representation of the decimal number -126 will be as follows:

0 1 1 1 1 1 1 0 ← + 126
Sign → ↓ ↓ ← All the "0" and "1" bits of the original bit string are switched
1 0 0 0 0 0 0 1
\+ 1 ←1 is added
1 0 0 0 0 0 1 0 ← − 126

As can be observed, even for the same number the bit strings of the "1's complement" and the "2's complement" differ.
Figure 1-1-13 shows a comparison of the range of numeric values which can be represented with 3 bits in the "1's complement" and the "2's complement" representation. From this Figure it can be noted that the range of representable numeric values with the "2's complement" is wider. Likewise, as in the absolute value representation of negative integers, and in the representation of negative numbers using the "1's complement," 0 can be represented both, as +0 and as -0, so the operation becomes complicated. For that reason, a great number of today's computers have adopted the 2's complement method.
Figure 1-1-14 shows the range of representable numeric values when an n-bit binary number is represented with the "1's complement" and the "2's complement."

Figure 1-1-13
"1's complement"
and "2's complement"

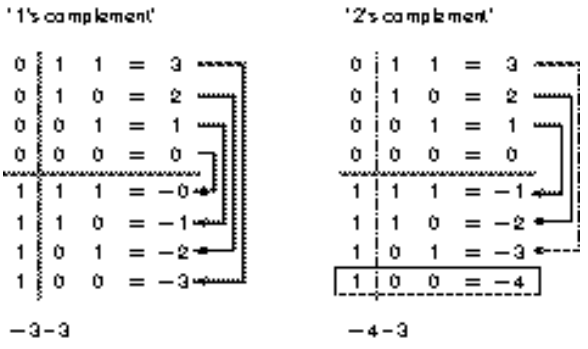
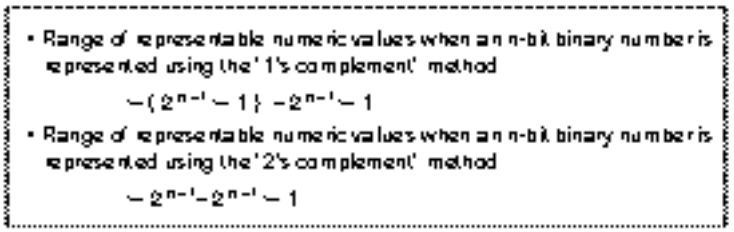


Figure 1-1-14
Range of representable numeric values with "1's complement" and "2's complement"



Another important reason for the adoption of the 2's complement method is illustrated by the following example:

Example

When a decimal calculation of 100 - 90 is performed in a computer, the decimal numbers 100 and -90 are first converted into binary numbers. At this time, if –90 is represented using the "2's complement" representation, the minus sign will not be necessary, and the representation will be as follows:

$(100)_{10} = (01100100)_2$

$(-90)_{10} = (10100110)_2$

Therefore, the subtraction 100-90 can be replaced by the addition 100+(-90).

```
  0 1 1 0 0 1 0 0
+ 1 0 1 0 0 1 1 0
-----------------
1 0 0 0 0 1 0 1 0     (Decimal digit 10)
↑
```

Since the bit number is 8, the 9th digit resulting from the carry is ignored.

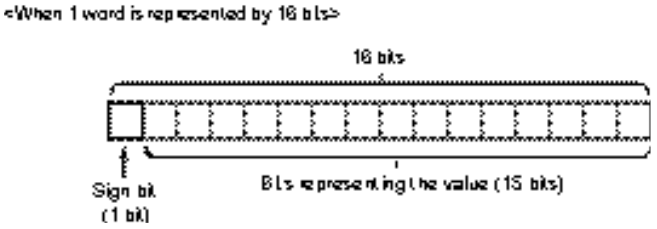
Therefore, the reason why negative numbers are represented using the "2's complement" method in computing is that subtractions can be performed as additions. In other words, since subtractions can be performed with the addition circuits, the subtraction circuits are unnecessary, simplifying the hardware structure.

② Fixed point

a. Integer representation

The fixed point is a data representation format used mainly when integer type data is processed (Figure 1-1-15). Therefore, the fixed point format is also called an integer type.
In the unpacked decimal format or packed decimal format, depending on the digit number of the decimal number, the number of bytes changes, but in the fixed point format one word is represented in a fixed length such as 16 bits and 32 bits.
For that reason, if there is an attempt to represent a numeric value that exceeds the fixed length a problem called overflow will occur.

Figure 1-1-15
Fixed point



Since in the fixed point format in Figure 1-1-15, where a value is represented with 15 bits, if a negative number is represented using the "2's complement" representation, the range of representable numeric values in the decimal system is as follows:

-2^{15} to $2^{15} - 1 = -32{,}768$ to $32{,}767$

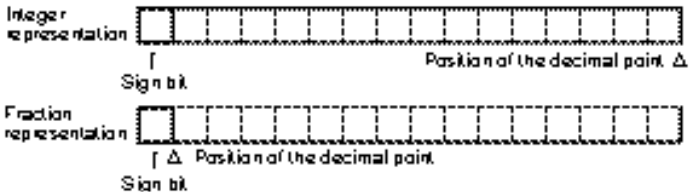
Likewise if one word is composed of n bits, and a negative number is represented using the "2's complement" representation, the range of representable numeric values in the decimal system is as follows:

-2^{n-1} to $2^{n-1} - 1$

b. Fraction representation

When integers are represented, the position of the decimal point is considered to be on the right-hand side of the least significant bit.
When fractions are represented, the position of the decimal point is considered to be immediately preceded by the sign bit.

Figure 1-1-16 Representation format of integers and fractions



③ Floating point

While the fixed point format represents integer-type data, the floating point format is used to represent real number type data. In ordinary mainframe computers, a maximum of 18-digit decimal numbers only can be represented. With 18 digits, there should be almost no problem in our daily life.
However, in a world where complicated calculations such as the ones mentioned below are required, correct results cannot be achieved with integer type data alone.

- Fluid mechanics calculations required for airplane design
- Calculations for weather forecasts
- Space flight planning and control
- Ballistic calculation
- CAD (Computer Aided Design)

For scientific and engineering fields requiring this kind of complicated calculation, the floating point format is used. Here, "complicated" means not only the calculation process itself is complicated, but also the either extremely large or small size of data is processed.
When we represent the number 1,500,000,000, instead of writing 8 zeros, we use the following exponent representation:

$$15 \times 10^{8}$$

In the floating point format it would be written as 0.15×10^{10}.

$$\underline{0.15} \times 10^{10}$$
$$\uparrow$$

This part is represented as smaller than 1

The name of each of the numbers is shown below.

$$0.15 \times 10^{10} \leftarrow \text{Exponent}$$
$$\uparrow \quad \uparrow$$

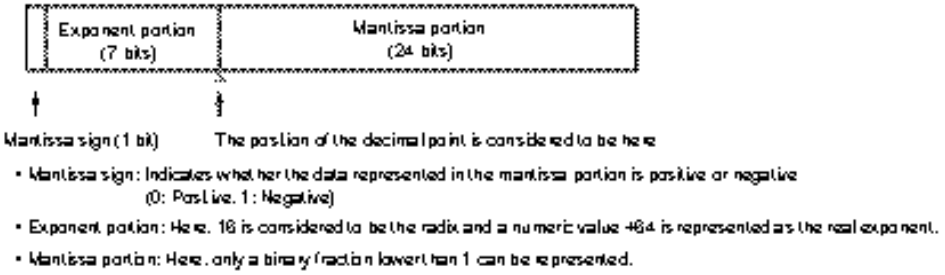
Mantissa Radix

Here, to make it easier to understand, the decimal system is used, but in the computer, the binary system is used.
The floating point representation format varies depending on the computer. This is roughly classified into the format used in mainframe computers and that defined by the IEEE (Institute of Electrical and Electronics Engineering).

a. Floating point representation format in mainframe computers

The floating point representation format used in general-purpose computers is shown in Figure 1-1-17. This format was adopted in the first general-purpose computer in the world the "IBM System/360" and it was called Excess 64.

Figure 1-1-17 Floating point representation format in general-purpose computers



- Exponent portion

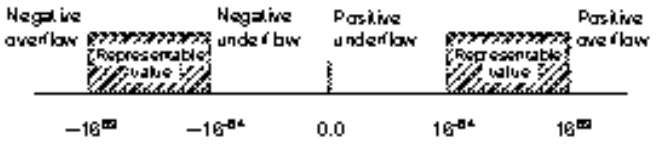
  The exponent portion has 7 bits, and the range of numeric values representable in the binary system is $(0000000)_2$ to $(1111111)_2$, which in the decimal system is 0 to 127. However a numeric value 64 times larger than the real exponent is represented. For that reason, the real exponent is equivalent to –64 to +63.

  Likewise, since the radix is considered to be 16, the numeric values that can be represented with the exponent portion range between

  16^{-64} to 16^{63}

  Then, including the sign bit, the range of numeric values that can be represented with the exponent portion is shown in Figure 1-1-18.

Figure 1-1-18 Range of numeric values that can be represented with the exponent portion



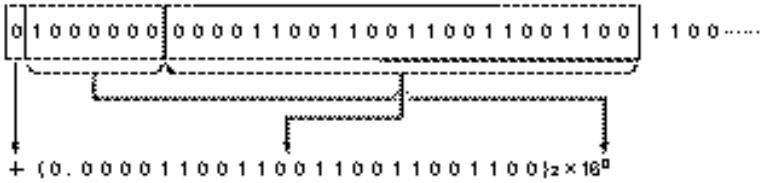
- Mantissa portion

  When the decimal fraction 0.05 is converted into a binary fraction, it becomes a repeating binary fraction.

  $(0.0000110011001100110011001100...)_2$

Figure 1-1-19 shows the representation of this fraction when the floating point format is used.

Figure 1-1-19 Representation of the binary fraction $(0.0000110011001100110011001100...)_2$

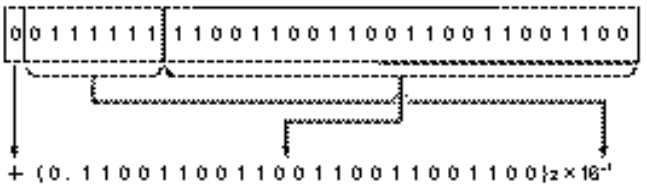


Since the mantissa portion has 24 bits, in this case, the decimal fraction 0.05 will not be represented correctly. (The error that occurs in this case is called a rounding error.)

However, if we look at the bit pattern of the mantissa portion, we can see that the 4 top bits are 0, if we then extract these 4 bits and shift the remaining bits to the left, 4 rounded bits can be represented. Here, as a result of shifting the mantissa portion 4 bits to the left, the original value of the mantissa portion was increased by $2^4 = 16$. In order to cancel this increase it is necessary to divide it into 16 ($\times 16^{-1}$). Since the radix is 16, the value of the exponent portion can be set to -1. The technique, used in this way, in order to reduce the rounding error to its minimum as well as to maximize precision is called normalization. Furthermore, as a result of this normalization technique, the bit strings that represent a value are standardized. This operation is performed automatically by the hardware.

The result of the normalization of the binary fraction representation in Figure 1-1-19 is shown in Figure 1-1-20.

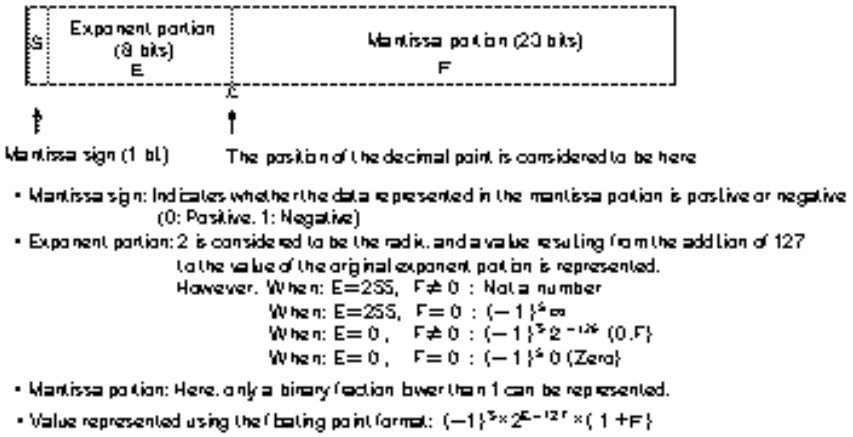
Figure 1-1-20
Normalization



b. IEEE Floating point representation format

The floating point representation format according to an IEEE standard is shown in Figure 1-1-21.

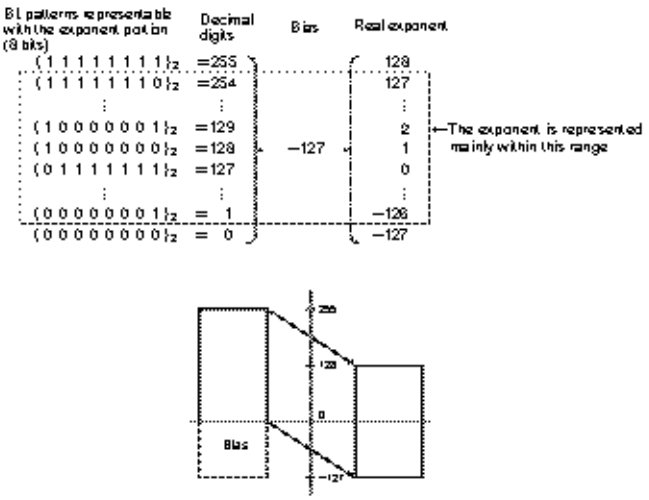
Figure 1-1-21 IEEE floating point representation format



The differences from the general-purpose computer floating point representation format are as follows:

- The exponent portion has 8 bits, and a value resulting from the addition of 127 to the value of the original exponent portion is represented. This addition to the original value is called bias (Figure 1-1-22).
- The mantissa portion has 23 bits and a binary fraction equivalent to the mantissa -1 is registered. In other words, 1 is considered to be omitted.
- The radix of the exponent portion is 2.

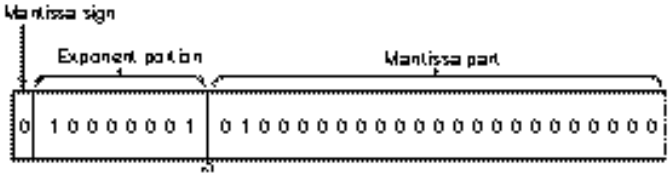
Figure 1-1-22 Representation of the exponent portion



For example, if the decimal number 5 is represented using this format, it will be represented as in Figure 1-1-23.

- Since it is positive, the mantissa sign will be 0.
- If $(5)_{10} = (101)_2 = (101)_2 \times 2^0 = (1.01)_2 \times 2^2$, then, the exponent portion will be $(1.01)_2$-$(1)_2 = (0.01)_2$
- If $(101)_2$ is shifted 2 bits to the right, it becomes $(1.01)_2$, 2^{-2} times the former value. In order to normalize it, 2 is added to the exponent 0, which becomes 2. As a result, since the exponent portion is 2 + 127 = 129, the representation will be $(10000001)_2$.

Figure 1-1-23
Representation of the decimal number 5



c. Shift operation

The mechanism of the shift operation performed at the normalization is explained below.
In the binary system, each digit has a weight which is a power of 2. This is called positional weight. Therefore, even though the same number 1 is represented, its meaning is different to that of the 1 positioned in the second digit and the 1 positioned in the third digit.

1 positioned in the second digit: $(10)_2 \rightarrow 2^1 = 2$

1 positioned in the third digit: $(100)_2 \rightarrow 2^2 = 4$

In the shift operation, by moving the position of 1 to the left (or to the right), the multiplication and division of numeric values can be easily performed.
The conversion into decimal numbers of $(100101)_2$ and $(1001010)_2$, resulting from shifting the first value 1 bit to the left, would be as follows:

Weight of each digit:		2^6	2^5	2^4	2^3	2^2	2^1	2^0	
$(100101)_2$	$\rightarrow$		1	0	0	1	0	1	$= 32 + 4 + 1 = (37)_{10}$
$(1001010)_2$	$\rightarrow$	1	0	0	1	0	1	0	$= 64 + 8 + 2 = (74)_{10}$

Through this conversion, it should be noted that the 1 that represented 2^5 before shifting, now represents 2^6, the 1 that represented 2^2 now represents 2^3, and the 1 that represented 2^0 now represents 2^1. In other words after the shift the value of each of the 1s was doubled, and the result of the conversion into the decimal system was also doubled from $(37)_{10}$, to $(74)_{10}$.
In short, the above-mentioned result shows that by shifting a binary digit 1 bit to the left, its value is doubled. Following this approach, the shift operation can be summarized by the following rules:
[Shift operation rules]

- When a binary number is shifted n bits to the left, its former value is increased 2^n times.
- When a binary number is shifted n bits to the right, its former value decreases 2^n times. (The former value is divided by 2^n)

The shift operation can be used to calculate numeric values, as in the above-mentioned example, as well as to simply change the position of a bit.

● Arithmetic shift

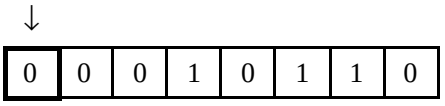
The arithmetic shift is the shift operation used to calculate numeric values. It is used in the fixed point format that represents negative numbers using the "2's complement" representation.
[Arithmetic shift rules]

- The sign bit is not shifted.
- The bit shifted out is lost.
- The bit to be filled into the bit position vacated as a result of the shift is:
  For left shifts: 0
  For right shifts: Same as the sign bit

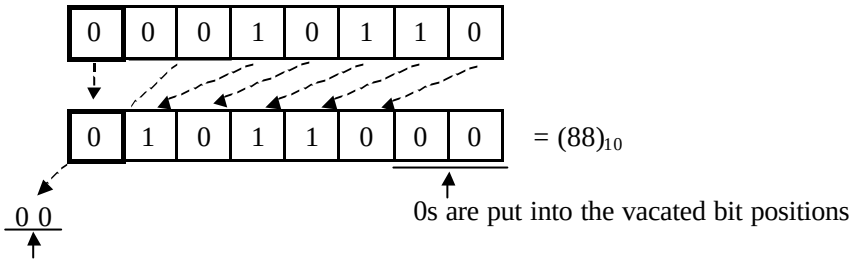
Example Calculation of $(22)_{10} \times 4$ using the arithmetic shift

① Represent $(22)_{10}$ using the fixed point format (8 bits)

$(22)_{10} = (10110)_2$



② Shift 2 bits to the left to increase it by 4 (= 2^2).



- Logical shift

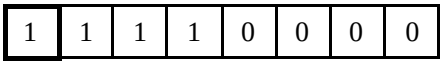
The logical shift is the shift operation used to change the bit position. The big difference from the arithmetic shift is that the sign bit is not treated differently.

[Logical shift rules]

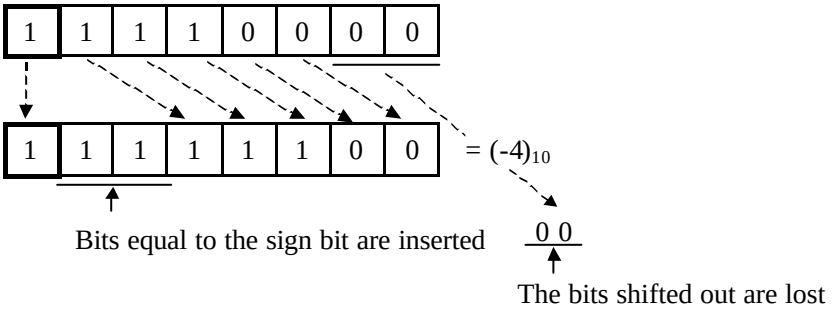
- The sign bit is also shifted (moved).
- The bit shifted out is lost.
- The bit to be filled into the bit position vacated as a result of the shift is 0.

Example After arithmetically and logically shifting $(-16)_{10}$ 2 bits to the right, convert each of the results into decimal digits.

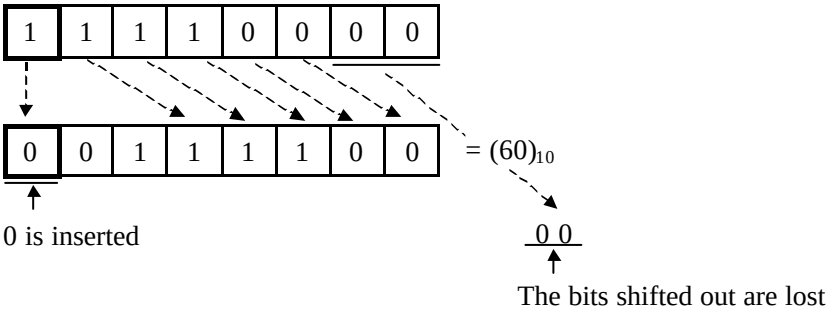
① Represent $(-16)_{10}$ using the fixed point format (8 bits).



② Arithmetically shift 2 bits to the right



③ Logically shift 2 bits to the right



1.1.3 Operation and precision

Since the storage capacity of a computer is limited, not all the numeric values we use can be represented correctly. In other words, a value represented in a computer is an approximate representation. As was mentioned above, in commercial data processing, operations are performed using decimal representation, while both the internal representation as well as the operation of scientific and engineering calculations, are performed using the binary representation. For that reason, the difference between the numeric value represented internally and the true value becomes a problem.

(1) Precision of the numeric value representation

The precision of a number is the range of its error, and so "high precision" means "small error". Focusing only on the integer part, if an enough number of digits to represent the conversion of decimal numbers into binary are available, no error occurs. However, the fraction part is not so simple; since many decimal fractions cannot be completely represented with the binary fractions containing a finite number of digits.

① Single precision

In scientific and engineering calculations, numerical values are represented with binary digits in word units. The word length depends on the hardware. For example, when 1 word = 16 bits, in general terms, the format in which 1 numeric value is represented with 1 word is called single precision. The range of numeric values representable with 16 bits, in case of an integer without a sign, is indicated below.

Minimum value = $(0000\ 0000\ 0000\ 0000)_2 = 0$

Maximum value = $(1111\ 1111\ 1111\ 1111)_2 = 65{,}535$

In other words, values higher than 65,535 cannot be represented.
Likewise, the range of numeric values representable with 16 bits, in the case of a fraction without a sign is indicated below.

Minimum value = $(0000\ 0000\ 0000\ 0000)_2 = 2^{-16} \leq 0.000015258789062500$

Maximum value = $(1111\ 1111\ 1111\ 1111)_2 = 1\text{-}2^{-16} \leq 0.999984741210937000$

In this case, values lower than 0.00001525878, and values higher than 0.999984741210937 can't be represented.

② Double precision

In order to widen the range of representable numeric values, the number of digits is increased. By representing 1 numeric value with 2 words, in comparison to the representation with 1 word, the range of representable numeric values becomes much wider. This format is called double precision. If 1 word = 16 bits, 1 numeric value is represented with twice as many bits, 32 bits.

Minimum value = $(0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000)_2 = 0$

Maximum value = $(1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111)_2 = 4{,}294{,}967{,}295$

In other words, values up to 4,294,967,295 can be represented.
Likewise, the range of numeric values representable with 32 bits, in case of a fraction without sign is indicated below.

Minimum value = $(0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000)_2 = 2^{-32}$
$\leq 0.00000000023283064365387$

Maximum value = $(1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111)_2 = 1 - 2^{-32}$
$\leq 0.99999999976716900000000$

(2) Operation precision

① Precision of fixed point representation

The range of representable numeric values with the fixed point representation depends on the computer hardware. Depending on the number of bits in one word, the range of representable numeric values differs. The step size of the integer part is always 1, regardless of the number of bits, and only the maximum value changes. However, in the step size of the fraction part the larger the number of bits assigned, the smaller the step size becomes and the error is also reduced.

② Precision of floating point representation

a. Overflow and underflow

When multiplication of extremely large values or extremely small values is performed, there are cases where the operation results exceed the range of numeric values that can be represented with the exponent portion. The condition that occurs when the product is higher than the maximum value that can be represented with the exponent portion, is called overflow (Maximum absolute value < Overflow). The condition that occurs when the product is lower than the minimum absolute value is called underflow (0 < Underflow < Minimum absolute value).

b. Cancellation

When subtraction of two floating point numbers of almost equal values is performed, since the result becomes extremely small, it is left out of the range of numeric values which can be represented. This condition is called cancellation.

c. Loss of information

When two values represented by using the floating point format are added, the exponents must coincide. Generally, exponents are adjusted to the largest value.
When an addition of an extremely small value and an extremely large value is performed, since exponents must be adjusted to the exponent of the largest value, the mantissa portion of the small value is shifted largely to the right. As a consequence of this shift, the information that should have been represented is lost. This condition is called loss of information.

In order to avoid this kind of error, it is necessary to think out strategies such as changing the order of the operations, etc. These strategies are worked out by the user.

1.1.4 Non-numeric value representation

When using a computer, in order to input numerals and characters (alphabetical characters, symbols, etc.) input devices such as keyboards, are used. Inside the computer, in order to represent characters using binary digits a concept called code is used.
Presently, different character codes are used depending on the computer. Here, the codes widely used around the world and in Japan will be explained.

(1) Character representation

The character codes widely used worldwide basically represent 1 character with 8 bits, that is, 1 byte.
The character codes used in the information processing field are sometimes called codes for information interchange.
By typing on a keyboard, these character codes are input in the computer as 1-byte codes.
The following keys are found in the keyboard of a personal computer, etc.

- Numeric keys: 10 types (0 to 9)
- Character keys: Alphabet: (Uppercase: A to Z and lowercase: a to z) 52 types
- Symbolic keys: 40 types
- Control character keys: 34 types (Space key, etc.)

To assign the unique bit pattern corresponding to these 136 types of characters and symbols, 256 types of bit patterns that can be represented with 8 bits are required.

(2) Character codes

The main character codes are listed below.

① ASCII (American Standard Code for Information Interchange) code

The ASCII code was established by the U.S. standards institution, ANSI (American National Standards Institute) in 1962. Character code of 8 bits composed by the code bit representing the alphabet, numeric characters, etc. (7 bits) and the parity bit used to detect errors. It is used in personal computers and in data transmission.

② ISO (International Organization for Standardization) code

The ISO code is a 7-bit character code that was established by the International Organization for Standardization (ISO) in 1967 based on the ASCII code. It is the base of the character codes used in all countries of the world.

③ JIS (Japanese Industrial Standards) code

The JIS code was established as JIS X 0201 by adding the Romaji, hiragana and other characters peculiar to the Japanese language to the ISO code. The "JIS 7-bit code" used to represent Romaji and the "JIS 8-bit code" used to represent katakana as well as the "JIS kanji code," that represents 1 character with 2 bytes (16 bits), and is used to represent hiragana and kanji, exists.

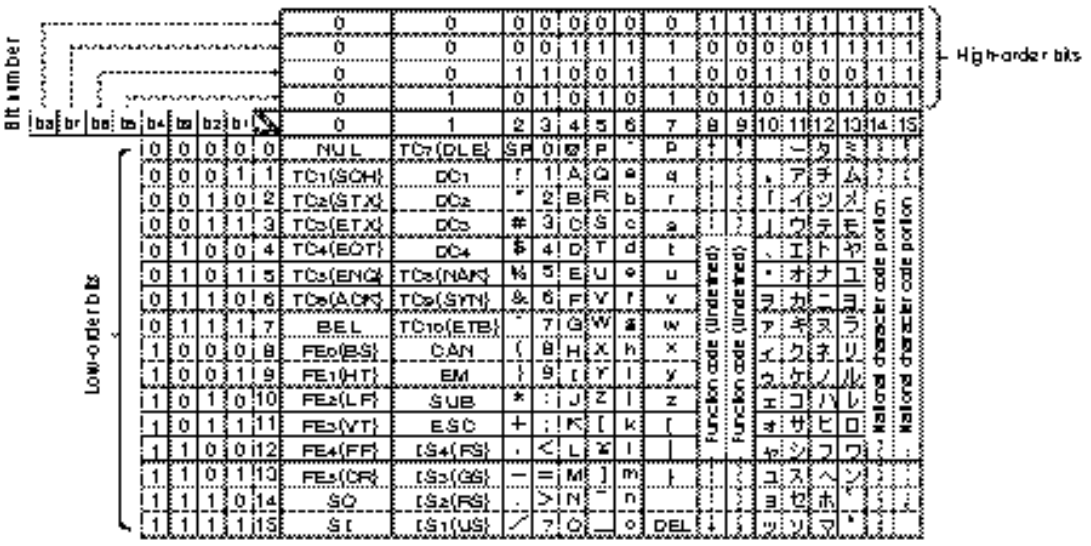
④ EBCDIC (Extended Binary Coded Decimal Interchange Code)

The EBCDIC is a character code developed by IBM. Compared to the above-mentioned character codes, which were established to be used as standards, the EBCDIC code was developed for IBM computers. Since IBM held the greatest share of the computer market when the third generation computers, in which this code was developed, were launched, other companies developed their computers according to this character code, and as a result it became a standard character code. Standards like this, resulting from the existence of a large number of users, are called *de facto* standards.

⑤ Shift JIS code

As was mentioned above, to represent kanji, the JIS kanji code represents 1 word with 2 bytes.

Figure 1-1-24 JIS X 0201 code table



There are times when the JIS code and the JIS kanji code are mixed. When 1-byte codes and 2-byte codes are mixed, their interpretation can't be performed. Therefore, a special escape code is added to the front and back of the kanji code string. For example, a bit string of 5 kanji becomes 10 bytes + 2 bytes. When data is missed during data transmission, etc. recovery is difficult.
In order to find a solution to this defect, the shift JIS code that converts the characters defined in the JIS Kanji code into another code system was created. The first byte of the shift JIS uses a code that is not used in the JIS (1 byte) code, and, at the same time, by avoiding control character codes in the second character, 1-byte codes and 2-byte codes can be mixed without using special escape codes.

⑥ Unicode

The unicode is a 2-byte code system unified to all the countries, which was proposed and designed by Apple Computer, IBM, Microsoft, and other U.S. companies in order to smooth the exchange of data amongst personal computers. This code was adopted by ISO as an international standard draft.

(3) Audio representation

As has been said, the current information technology provides multimedia support, and the data subject to processing is not limited to character data or numerical data. It also covers many kinds of information that are used in our daily life. One of the components which compose multimedia is audio.
The human voice is produced when the airflow generated in the lungs changes, vibrates and resonates due to a great number of organs such as the tongue, lips, teeth, jaw, nasal cavity and vocal cords. Since audio data has a complicated analog waveform, audio analysis is performed using a numeric formula and once it is converted into digital codes it is processed in the computer. Word processors that accept audio input and speaker recognition are examples of its recent applications.

(4) Image representation

In order to support current multimedia, not only audio but also image data must be processed.
Inside the computer, image data is processed as a set of dots. For that reason, the registration of the status of each of the dots that compose an image, is the registration of the image data itself. The easiest approach is to register two states, black and white, for each of the dots that compose an image. In this case, 1 bit is used to register the information of each dot. Today most of the image data is colored, so this method does not solve all the problems. Therefore, the representation method that combines the basic colors in each dot is used. Amongst computer screens, there are a great number of systems that combine the three primary colors (Red, green and blue) in 256 levels respectively and represent approximately 16,000,000 colors. In this case, since 8 bits are needed for 1 color, in order to register the information of 1 dot, 24 bits are used.

1.2 Information and logic

1.2.1 Proposition logic

Operations which can be processed in a computer are not limited to arithmetic formulas. By assigning a value to a sentence, sentence operations can be performed. For example, in logical mathematics, the sentences represented as "The wind is blowing," "It is raining," "x=5" and "y=2" are called propositions. Values of "truth" or "lie," in other words, "true" and "false" can be assigned to these propositions. However, one proposition will always be either "true" or "false." The same proposition can't be "true" at the same time it is "false." "The wind is blowing and it is raining" is possible, but "The wind is blowing, there is no wind" is impossible.
These propositions are represented by p, q, r, ... and other letters, and through the combination of their logical significance new synthetic propositions can be created. Each proposition relation is made clear,

through logical operation by proposition logic. Whether a synthetic proposition is true or false is determined by the truth table. An example of this table is shown in Figure 1-2-1.
This truth table shows:

- The proposition "The wind is not blowing" is false when the proposition 1, "The wind is blowing," is true, it is true when the proposition 1 is false.
- The proposition "The wind is blowing or it is raining" is true when both, the proposition 1, "The wind is blowing," and the proposition 2, "It is raining," are true or when either of the two is true. When both of them are false, it is false.

Figure 1-2-1 Truth table

Proposition 1 The wind is blowing	Proposition 2 It is raining	The wind is not blowing	The wind is blowing and it is raining	The wind is blowing or it is raining	If the wind blows it rains
True	True	False	True	True	True
Wind	Rain	Lie	Truth	Truth	Truth
True	False	False	False	True	False
Wind	No rain	Lie	Lie	Truth	Lie
False	True	True	False	True	False
No wind	Rain	Truth	Lie	Truth	Lie
False	False	True	False	False	False
No wind	No rain	Truth	Lie	Lie	Lie

1.2.2 Logical operation

Since the expression of the logical significance with words becomes lengthy and it is not suitable for computer operations, logical relations are represented with symbols. The symbols that represent these propositional operations (or logical operations) are called logical symbols or logical connectors. The main logical symbols used in information processing are NOT, AND, OR, exclusive OR, etc. Their meanings are explained below.
Each logical operation will be explained using the examples shown in Figure 1-2-1, and proposition 1 "The wind is blowing," which will be denoted as p, and proposition 2 "It is raining" as q.

(1) Negation

By negating the proposition "The wind is blowing," a new proposition, "The wind is not blowing" can be created. In this case, the logical symbol "¬ (NOT)" is used and it is represented as "¬ p."

Figure 1-2-2
Truth table for negation

p	¬p
T	F
F	T

T(rue) : True
F(alse) : False

(2) Logical product

When two propositions are connected with the conjunction "AND" as in "The wind is blowing and it is raining," both "The wind is blowing" and "It is raining" are expressed simultaneously.
The connection of the two propositions p and q with the conjunction "AND" is called logical product. In this case, the logical symbol "∧ (AND)" is used and it is represented as "p∧q." The truth table is shown in Figure 1-2-3, and the result is true only when p and q are both true.

Figure 1-2-3
Truth table for
the logical product

p	q	p∧q
T	T	T
T	F	F
F	T	F
F	F	F

(3) Logical sum

When two propositions are connected with the conjunction "OR" as in "The wind is blowing or it is raining," either "The wind is blowing" or "It is raining" is expressed.
The connection of the two propositions p and q with the conjunction "OR" is called logical sum. In this case, the logical symbol "∨ (OR)" is used and it is represented as "p∨q." The result is true only when p and q are both true.

Figure 1-2-4
Truth table for
the logical sum

p	q	p∨q
T	T	T
T	F	T
F	T	T
F	F	F

(4) Exclusive OR

In the logical sum mentioned above, "The wind is blowing or it is raining," either "The wind is blowing" or "It is raining" is expressed. This logical sum is true when "The wind is blowing and it is also raining," or in other words, when both propositions are true. Ordinarily, the word "or" is used in many cases to express exclusive meanings as "either of the two." The exclusive OR is used to support these cases.
In the case of the exclusive OR, the logical symbol "⊻ (EOR)" is used and it is represented as "p⊻q." The result is true only when p or q, either of the two, is true. Therefore, the result is false when p and q are both true or false. This logical operation is frequently used in programming.

Figure 1-2-5
Truth table for
the exclusive OR

p	q	p∨q
T	T	F
T	F	T
F	T	T
F	F	F

(5) Negative AND (NAND)

It is the negation of the above-mentioned logical product. It is represented as "¬ (p∧q)." This logical operation is frequently used in the design of digital circuits.

(6) Negative logical sum (NOR)

It is the negation of the above-mentioned logical sum. It is represented as "¬ (p∨q)."
Figure 1-2-6 puts the six logical operations mentioned above together.

Figure 1-2-6 Truth table for the logical operations NOT, AND, OR, EOR, NAND and NOR (Summary)

p	q	NOT p	p AND q	p OR q	p EOR q	p NAND q	p NOR q
T	T	F	T	T	F	F	F
T	F	F	F	T	T	T	F
F	T	T	F	T	T	T	F
F	F	T	F	F	F	T	T

(7) Logical expression laws

The representation using the above-mentioned logical symbols is called logical expression. Along with the logical symbols presented earlier, the symbols shown in Figure 1-2-7 are also used.

Figure 1-2-7
Logical symbols

Meaning		Symbols		Notation example
Negation	NOT	¬	‾	$\overline{X}$
Logical product	AND	∧	·	X·Y
Logical sum	OR	∨	+	X+Y
Exclusive OR	EOR	⊽	⊕	X⊕Y

As the logic becomes complicated, the logical expression also becomes extremely complicated. For that reason, in order to simplify the logical expressions, the following laws are used:

- Logical product law: $X \cdot X = X, \quad X \cdot \overline{X} = 0, \quad X \cdot 0 = 0, \quad X \cdot 1 = X$
- Logical sum law: $X + X = X, \quad X + \overline{X} = 1, \quad X + 0 = X, \quad X + 1 = 1$
- Exclusive OR law: $X \oplus X = 0, \quad X \oplus \overline{X} = 1, \quad X \oplus 0 = X, \quad X \oplus 1 = \overline{X}$
- Commutative law: $X + Y = Y + X, \quad X \cdot Y = Y \cdot X$
- Associative law: $X + (Y + Z) = (X + Y) + Z, \quad X \cdot (Y \cdot Z) = (X \cdot Y) \cdot Z$
- Distributive law: $X + (Y \cdot Z) = (X + Y) \cdot (X + Z)$
  $X \cdot (Y + Z) = (X \cdot Y) + (X \cdot Z)$
- Absorptive law: $X + (X \cdot Y) = X, \quad X \cdot (X + Y) = X$
- Restoring law: $\overline{\overline{X}} = X$
- De Morgan's law: $\overline{X + Y} = \overline{X} \cdot \overline{Y}, \quad \overline{X \cdot Y} = \overline{X} + \overline{Y}$

For example, the logical expression of the exclusive OR is represented as $X \oplus Y = (\overline{X} \cdot Y) + (X \cdot \overline{Y})$. By using the above-mentioned laws, this logical expression can be changed as follows:

$$
\begin{aligned}
X \oplus Y &= (\overline{X} \cdot Y) + (X \cdot \overline{Y}) \\
&= ((\overline{X} \cdot Y) + X) \cdot ((\overline{X} \cdot Y) + \overline{Y}) && \cdots\cdots \text{Distribution law} \\
&= ((X + \overline{X}) \cdot (X + Y)) \cdot ((\overline{Y} + \overline{X}) \cdot (\overline{Y} + Y)) && \cdots\cdots \text{Switching law and distribution law} \\
&= (\ 1\ \cdot (X + Y)) \cdot ((\overline{Y} + \overline{X}) \cdot\ 1\ ) && \cdots\cdots \text{Logical sum law} \\
&= (X + Y) \cdot (\overline{X} + \overline{Y}) && \cdots\cdots \text{Logical product law}
\end{aligned}
$$

Exercises

Q1 **Which of the following represents correctly the size relation amongst the following 4 prefix symbols that represent integer exponents of 10: G (giga), k (kilo), M (mega) and T (tera)?**

A. G < k < M < T
B. k < G < T < M
C. k < M < G < T
D. M < G < k < T
E. M < T < G < k

Q2 **Which of these values corresponds to 1 picosecond?**

A. 1 nanosecond × 1,000
B. 1 microsecond / 1,000,000
C. 2^{-12} seconds
D. 10^{-10} seconds
E. 10^{-11} seconds

Q3 **Given the binary digits A and B, where**

A = 01010101 B = 01100110

which of these values corresponds to the result of the operation A + B?

A. 01101010
B. 01111010
C. 10011010
D. 10111011
E. 11010101

Q4 **Which of these values is the correct result of the subtraction of the hexadecimal numbers DD and 1F "00-1F"?**

A. AF B. BE C. CE D. EC E. FC

Q5 **Which of these values represents in decimal numbers the result of the addition of the binary numbers 1.1011 and 1.1101?**

A. 3.1 B. 3.375 C. 3.5 D. 3.8 E. 3.9375

Q6 **Which is the decimal number that can be represented without error in the binary floating point representation?**

A. 0.2 B. 0.3 C. 0.4 D. 0.5

Q7 **Which of these values represent the hexadecimal fraction 0.248 in decimal fractions?**

A. $\frac{31}{32}$ B. $\frac{31}{125}$ C. $\frac{31}{512}$ D. $\frac{73}{512}$

Q8 **Which is the correct bit pattern of the decimal number +432 when it is represented in the packed decimal format? Note that the sign is represented by the last 4 bits, and that "1100" represents positive numbers while "1101" represents negative numbers.**

A. 0000 0001 1011 0000
B. 0000 0001 1011 1100
C. 0001 1011 0000 1100
D. 0100 0011 0010 1100
E. 0100 0011 0010 1101

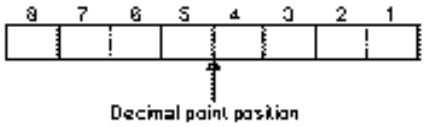
Q9 **What is the range of the integer representable with n bits using the fixed point format that represents a negative number with 2's complement representation? Here, the decimal point position is on the right side of the least significant bit (LSB).**

A. -2^{n} to 2^{n-1}
B. -2^{n-1} to 2^{n-1}
C. -2^{n-1} to $2^{n-1}-1$
D. $-2^{n-1}-1$ to 2^{n-1}

Q10 **Which of the following is the reason why in many computers the complement representation is used to simplify the operation circuit?**

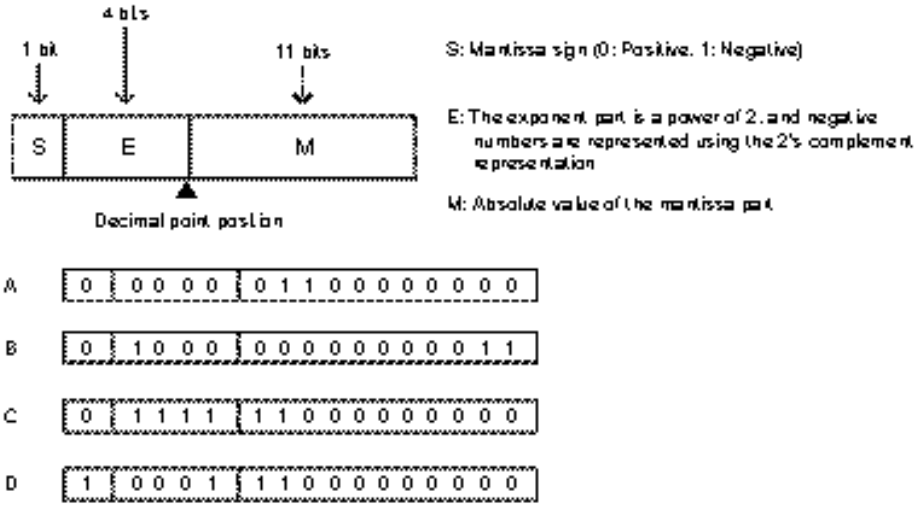
A. Additions can be processed as subtractions.
B. Subtractions can be processed as additions.
C. Multiplications can be processed by the combination of additions.
D. Divisions can be processed by the combination of subtraction.

Q11 **Which of these values corresponds to the representation of the decimal number -5.625 in binary number using the 8-bit fixed point format? Here, the position of the decimal point is between the fourth and fifth bits, and negative numbers are represented using the 2's complement representation.**



A. 01001100 B. 10100101 C. 10100110 D. 11010011

Q12 **Which is the normalized representation of the decimal number 0.375? Here, the numeric value is represented using the 16-bit floating point format and the format is indicated in the figure. The normalization performed is an operation that adjusts the exponent portion in order to eliminate the 0s of higher digit rather than the significant values of the mantissa p**



Q13 **Which is the value that corresponds to the result of logically shifting the hexadecimal number ABCD two bits to the right?**

A. 2AF3 B. 6AF3 C. AF34 D. EAF3

Q14 **The multiplication of binary number can be performed through the shift operation (digit shift), and addition. To increase the binary digit m by 2^n it is necessary to shift m n bits to the left.**
For example, m ´ 19 can be obtained through the following operation:
(Value of the result of shifting m one bit to the left) + (Value of the result of shifting m one bit to the left) + m
Which is the value of a?

A. 2 B. 3 C. 4 D. 5

Q15 **The decimal number -100 is registered using the 2's complement representation in a 8bit register. Which of these values represent, in decimal numbers, the result of arithmetically shifting this registration three bits to the right?**

A. –33 B. –13 C. –12 D. 19

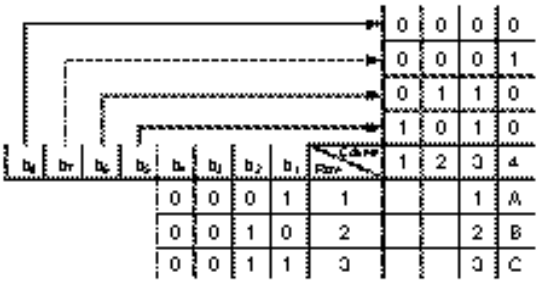
Q16 **Which of the following descriptions of the rounding error is correct?**

A. It is an error generated when an operation result exceeds the maximum value that can be processed by the computer.
B. Due to the limited number of digits in the number representation, it is an error generated as a result of rounding off, rounding up, or omitting portions smaller than the least significant digit.
C. It is an error generated due to the loss of the most significant values in the subtraction operation of numeric values, whose absolute values are almost equal.
D. It is an error generated due to the loss of the least significant values of the mantissa portion of the numeric value, with the lower exponent value in the subtraction operation of floating point numbers.

Q17 **What is the minimum number of digits required for representing uniquely with the same number of bits of the uppercase alphabetic characters (A to Z) and the numeric characters (0 to 9)?**

A. 5 B. 6 C. 7 D. 8

Q18 **The following table is a fraction of the JIS code table. Which of these values corresponds to the representation in the JIS code of the two characters "A" and "2" in this order?**



A. 00010100 00100011 B. 00110010 01000001
C. 01000001 00110010 D. 01000010 00110010

Q19 **The following truth table shows the operation results of the logical operation "x ★ y." Which of these expressions is equivalent to this operation?**

Truth table

x	y	x ★ y
True	True	False
True	False	False
False	True	True
False	False	False

A. x AND (NOT y)
B. x OR (NOT y)
C. (NOT x) AND y
D. (NOT x) AND (NOT y)
E. (NOT x) OR (NOT y)

Q20 **Which of the following expressions is equivalent to the logical expression $\overline{(A + B)\cdot C}$? Here, "·" represents the logical product (AND), "+," the logical sum (OR), and $\overline{A}$ is the negation of A (NOT):**

A. $(A\cdot B) + \overline{C}$
B. $A \cdot B \cdot \overline{C}$
C. $\overline{A} + \overline{B} + \overline{C}$
D. $(\overline{A} \cdot \overline{B}) + \overline{C}$