子夜吳歌

～李白

長安一片月，
萬戶擣衣聲；
秋風吹不盡，
總是玉關情。
何日平胡虜，
良人罷遠征。

OBJECTIVES

After reading this chapter, the reader should be able to :

- Convert a number from decimal to binary notation and vice versa.
- Understand the different representations of an integer inside a computer: unsigned, sign-and-magnitude, one's complement, and two's complement.
- Understand the Excess system that is used to store the exponential part of a floating-point number.
- Understand how floating numbers are stored inside a computer using the exponent and the mantissa.

Number Representation

Bit & Byte

- Human are familiar with decimal digits.
  - 925, 16, 49
- Computers represent data in binary digit (bit, 位元).
  - 0101 1110
- Bits are a very small unit. For convenience, 8 bits are grouped as 1 byte (位元組), which are the basic memory unit in computers.
  - File Size: 30MBytes
  - Memory Size: 1GBytes
  - Hard Disk Size: 2TBytes

3.1

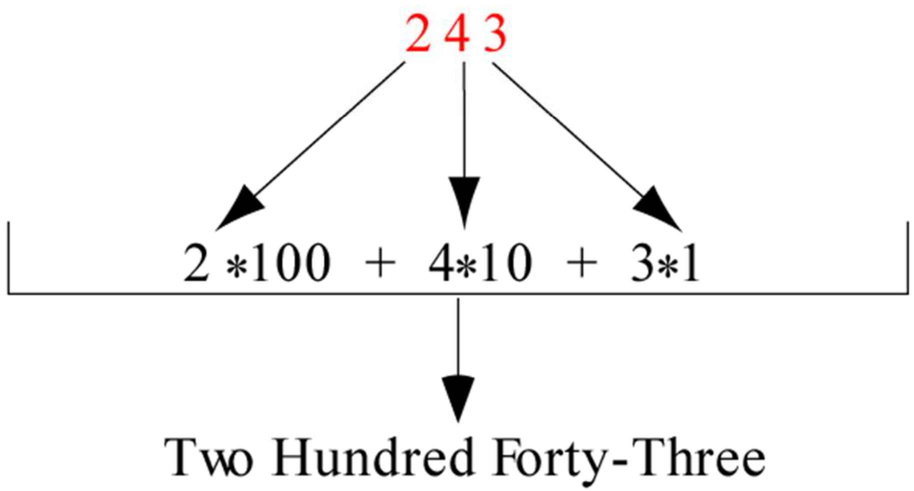
DECIMAL AND BINARY

Decimal system

The decimal system has 10 digits and is based on powers of 10

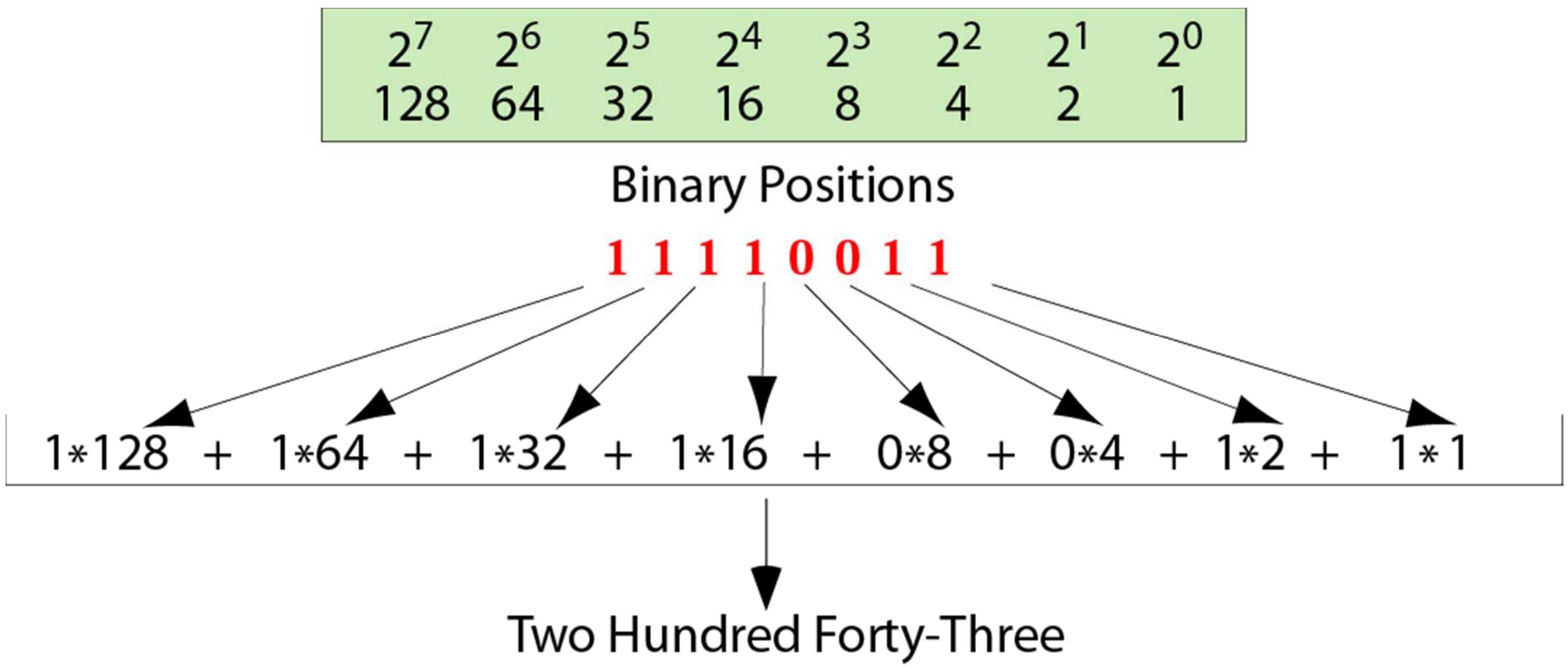
10^4	10^3	10^2	10^1	10^0
10,000	1000	100	10	1

Decimal Positions



Binary system

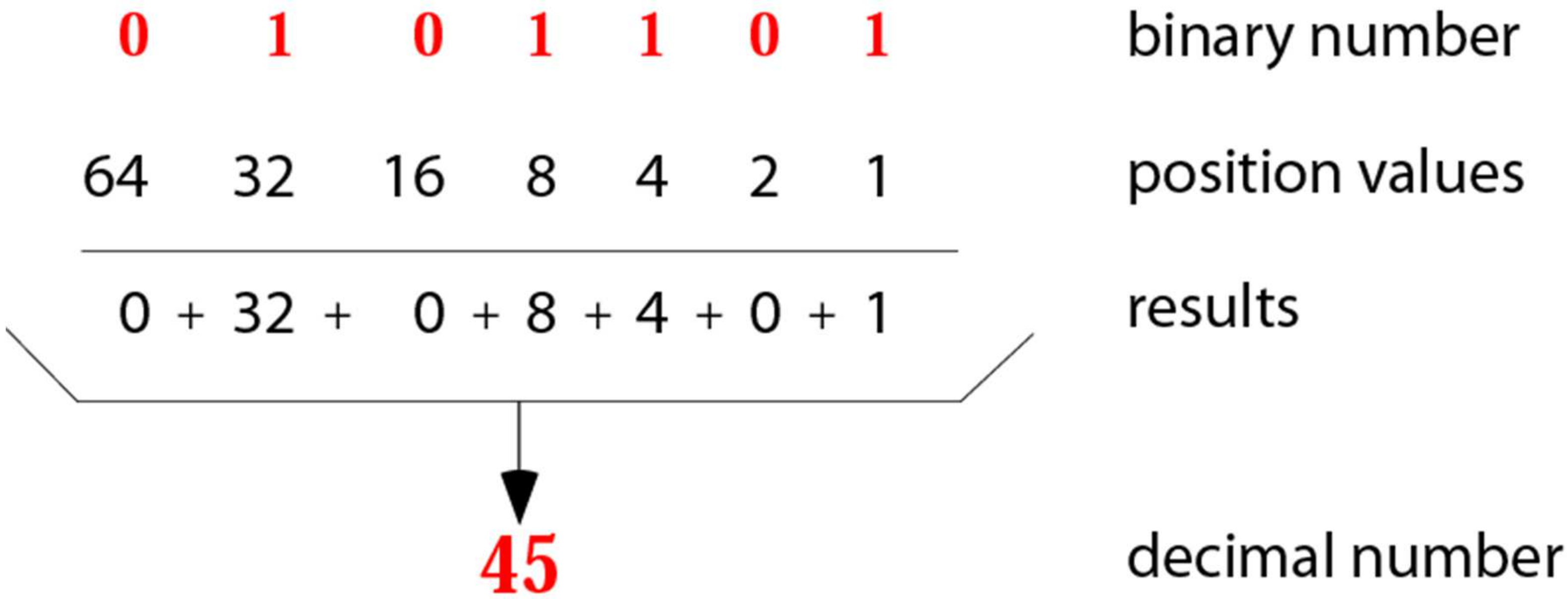
The binary system, used by computers to store numbers, has 2 digits, 0 and 1, and is based on powers of 2.



3.2

CONVERSION

Binary to decimal conversion



Example 1

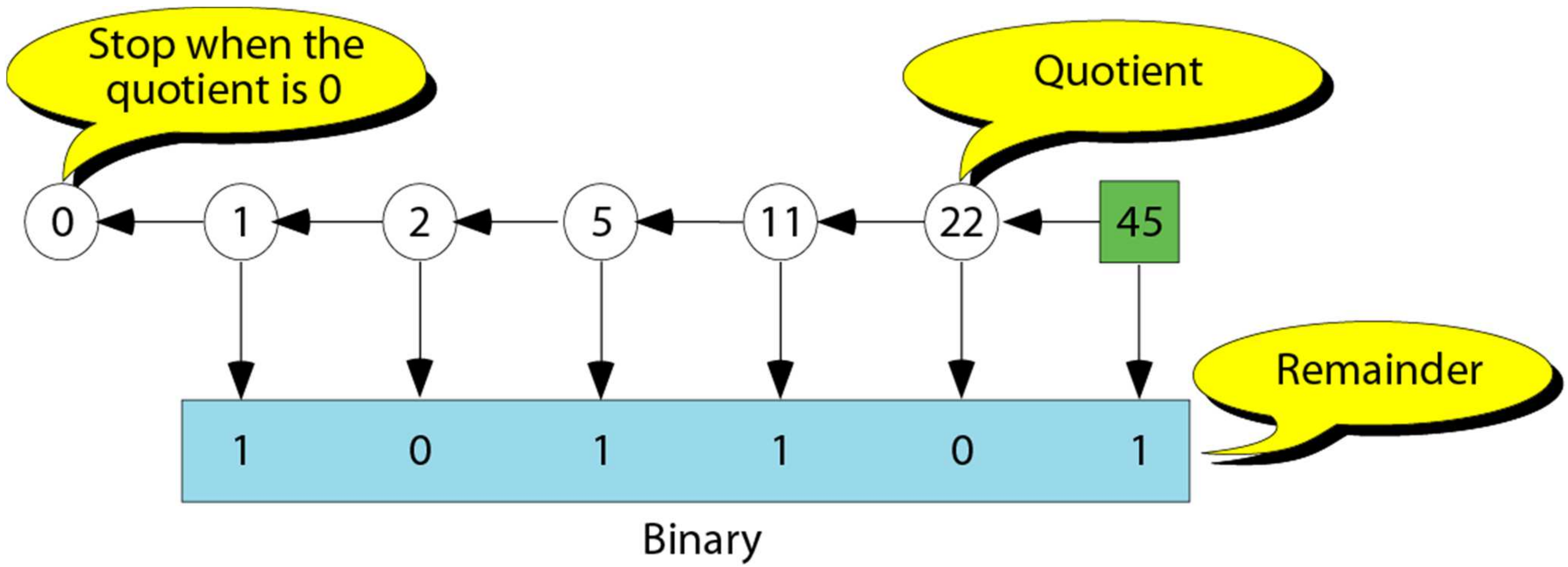
Convert the binary number 10011 to decimal.

Solution

Write out the bits and their weights. Multiply the bit by its corresponding weight and record the result. At the end, add the results to get the decimal number.

Binary	**1**	**0**	**0**	**1**	**1**
Weights	16	8	4	2	1
	-------	-------	-------	-------	-------
	16 +	0 +	0 +	2 +	1
Decimal					**19**

Decimal to binary conversion



Example 2

Convert the decimal number 35 to binary.

Solution

Write out the number at the right corner. Divide the number continuously by 2 and write the quotient and the remainder. The quotients move to the left, and the remainder is recorded under each quotient. Stop when the quotient is zero.

	0 ←	1 ←	2 ←	4 ←	8 ←	17 ←	35 Dec.
Binary	**1**	**0**	**0**	**0**	**1**	**1**	

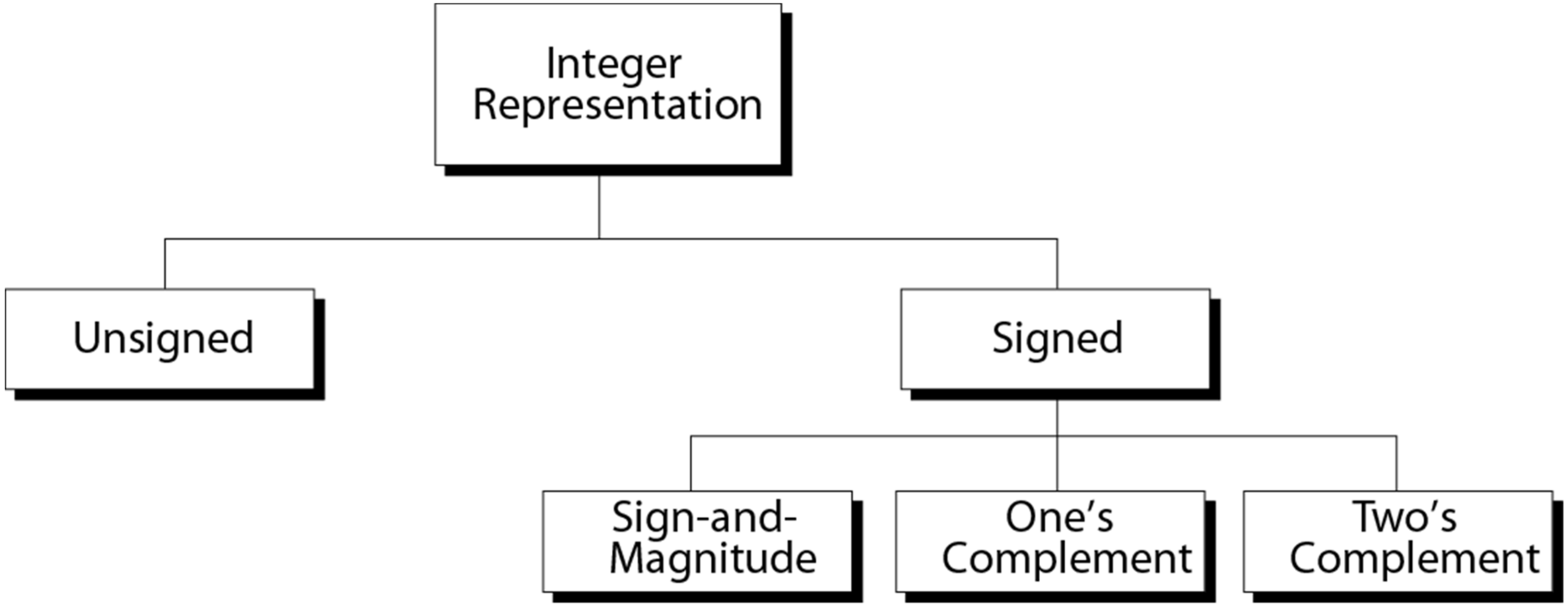
3.4

INTEGER REPRESENTATION

Range of integers

- An integer can be positive or negative
- To use computer memory more efficiently, Integers can be represented as unsigned or signed numbers
- There are three major methods of signed number representation:
  - Sign-and-magnitude
  - One’s complement
  - Two’s complement

Taxonomy of integers



8 = 00001000   -8   10001000   11110111   11111000

Unsigned integer

- Unsigned integer range: 0 … (2^N-1)

# of Bits	Range		
---------	-------------------------------------		
8	0	~	255
16	0	~	65,535

- Storing unsigned integers process:

1. The number is changed to binary
2. If the number of bits is less than N, 0s are added to the left of the binary number so that there is a total of N bits

Example 3

Store 7 in an 8-bit memory location.

Solution

First change the number to binary 111. Add five 0s to make a total of N (8) bits, 00000111. The number is stored in the memory location.

Example 4

Store 258 in a 16-bit memory location.

Solution

First change the number to binary 100000010. Add seven 0s to make a total of N (16) bits, 0000000100000010. The number is stored in the memory location.

Example of storing unsigned integers in two different computers

Decimal	*8-bit allocation*	*16-bit allocation*
7	00000111	0000000000000111
234	11101010	0000000011101010
258	overflow	0000000100000010
24,760	overflow	0110000010111000
1,245,678	overflow	overflow

- Unsigned numbers are commonly used for counting and addressing

Example 5

Interpret 00101011 in decimal if the number was stored as an unsigned integer.

Solution

32+8+2+1, the answer is 43.

Note:

In sign-and-magnitude representation, the leftmost bit defines the sign of the number. If it is 0, the number is positive.If it is 1, the number is negative.

Sign-and-magnitude integers

- Range: $-(2^{N-1}-1) \ldots +(2^{N-1}-1)$

# of Bits	Range			
8	–127	–0	+0	+127
16	–32767	–0	+0	+32767
32	–2,147,483,647	–0	+0	+2,147,483,647

- Storing sign-and-magnitude integers process:
  1. The number is changed to binary; the sign is ignored
  2. If the number of bits is less than N-1, 0s are add to the left of the binary number so that there is a total of N-1 bits
  3. If the number is positive, 0 is added to the left (to make it N bits). If the number is negative, 1 is added to the left

Example 6

Store +7 in an 8-bit memory location using sign-and-magnitude representation.

Solution

First change the number to binary 111. Add four 0s to make a total of N-1 (7) bits, 0000111. Add an extra zero because the number is positive. The result is:

00000111

Example 7

Store –258 in a 16-bit memory location using sign-and-magnitude representation.

Solution

First change the number to binary 100000010. Add six 0s to make a total of N-1 (15) bits, 000000100000010. Add an extra 1 because the number is negative. The result is:

1000000100000010

Example of storing sign-and-magnitude integers in two computers

Decimal	8-bit allocation	16-bit allocation
+7	00000111	0000000000000111
-124	11111100	1000000001111100
+258	overflow	0000000100000010
-24,760	overflow	1110000010111000

Example 8

Interpret 10111011 in decimal if the number was stored as a sign-and-magnitude integer.

Solution

Ignoring the leftmost bit, the remaining bits are 0111011. This number in decimal is 59. The leftmost bit is 1, so the number is ***–59****.*

Note:

There are two 0s in sign-and-magnitude representation: positive and negative.

In an 8-bit allocation:

+0 → 00000000

-0 → 10000000

Note:

In sign-and-magnitude representation,

You can add two positive integers:

0000 0110 = 6
0000 0101 = 5
0000 1011 = 11

But you cannot add a positive number and negative number:

0000 0110 = 6
1000 0101 = -5
1000 1011 = -11

One's complement integers

- Range: $-(2^{N-1}-1) \ldots +(2^{N-1}-1)$

# of Bits	Range		
---------	------------------------------------		
8	−127	−0 +0	+127
16	−32767	−0 +0	+32767
32	−2,147,483,647	−0 +0	+2,147,483,647

- Storing one's complement integers process:
  1. The number is changed to binary; the sign is ignored
  2. 0s are added to the left of the number to make a total of N bits
  3. If the sign is positive, no more action is needed. If the sign is negative, every bit is complemented.

Note:

In one's complement representation, the leftmost bit defines the sign of the number. If it is 0, the number is positive.If it is 1, the number is negative.

Example 9

Store +7 in an 8-bit memory location using one's complement representation.

Solution

First change the number to binary 111. Add five 0s to make a total of N (8) bits, 00000111. The sign is positive, so no more action is needed. The result is:

00000111

Example 10

Store –258 in a 16-bit memory location using one's complement representation.

Solution

First change the number to binary 100000010. Add seven 0s to make a total of N (16) bits, 0000000100000010. The sign is negative, so each bit is complemented. The result is:

1111111011111101

Example of storing one's complement integers in two different computers

Decimal	8-bit allocation	16-bit allocation
+7	00000111	0000000000000111
−7	11111000	1111111111111000
+124	01111100	0000000001111100
−124	10000011	1111111110000011
+24,760	overflow	0110000010111000
−24,760	overflow	1001111101000111

Example 11

Interpret 11110110 in decimal if the number was stored as a one's complement integer.

Solution

The leftmost bit is 1, so the number is negative. First complement it . The result is 00001001. The complement in decimal is 9. So the original number was −9. Note that complement of a complement is the original number.

Note:

One's complement means reversing all bits. If you one's complement a positive number, you get the corresponding negative number. If you one's complement a negative number, you get the corresponding positive number. If you one's complement a number twice, you get the original number.

$00001111_2 = 15$

$11110000_2 = -15$

$00001111_2 = 15$

Note:

There are two 0s in one's complement representation: positive and negative.

In an 8-bit allocation:

+0 → 00000000

-0 → 11111111

Note:

In one's complement representation,

You can add a positive number and a negative number :

0000 0111 = 7
1111 1000 = -7
1111 1111 = -0

But not always:

0000 1000 = 8
1111 1000 = -7
0000 0000 = 0

Two's complement integers

- Range: $-(2^{N-1}) \ldots +(2^{N-1}-1)$

# of Bits	Range		
8	–128	0	+127
16	–32,768	0	+32,767
32	–2,147,483,648	0	+2,147,483,647

- Storing two's complement integers process:

1. The number is changed to binary; the sign is ignored
2. If the number of bits is less than N, 0s are added to the left of the number so that there is a total of N bits.
3. If the sign is positive, no further action is needed. If the sign is negative, leave all the rightmost 0s and the first 1 unchanged. Complement the rest of the bits.

Example 12

Store +7 in an 8-bit memory location using two's complement representation.

Solution

First change the number to binary 111. Add five 0s to make a total of N (8) bits, 00000111. The sign is positive, so no more action is needed. The result is:

00000111

Example 13

Store –40 in a 16-bit memory location using two's complement representation.

Solution

First change the number to binary ***101000****. Add ten 0s to make a total of N (16) bits,* ***0000000000101000****. The sign is negative, so leave the rightmost 0s up to the first 1 (including the 1) unchanged and complement the rest. The result is:*

1111111111011000

Note:

In two's complement representation,
the leftmost bit defines the sign of the number.
If it is 0, the number is positive.
If it is 1, the number is negative.

Note:

Two's complement is the most common, the most important, and the most widely used representation of integers today.

Example of storing two's complement integers in two different computers

Decimal	*8-bit allocation*	*16-bit allocation*
+7	00000111	0000000000000111
−7	11111001	1111111111111001
+124	01111100	0000000001111100
−124	10000100	1111111110000100
+24,760	overflow	0110000010111000
−24,760	overflow	1001111101001000

# of Bits	*Range*		
8	−128	0	+127
16	−32,768	0	+32,767
32	−2,147,483,648	0	+2,147,483,647

Note:

In two's complement representation,

You can add a positive number and a negative number :

0000 0111 = 7
1111 1001 = -7
0000 0000= 0

0000 1000 = 8
1111 1001 = -7
0000 0001 = 1

Note:

There is only one 0 in two's complement:

In an 8-bit allocation:

0 → 00000000

Example 14

Interpret 11110110 in decimal if the number was stored as a two's complement integer.

Solution

The leftmost bit is 1. The number is negative. Leave 10 at the right alone and complement the rest. The result is 00001010. The two's complement number is 10. So the original number was –10.

Note:

Two's complement can be achieved by reversing all bits except the rightmost bits up to the first 1 (inclusive). If you two's complement a positive number, you get the corresponding negative number. If you two's complement a negative number, you get the corresponding positive number. If you two's complement a number twice, you get the original number.

Summary of integer representation

Contents of Memory	Unsigned	Sign-and-Magnitude	One's Complement	Two's Complement
0000	**0**	+0	+0	+0
0001	**1**	+1	+1	+1
0010	**2**	+2	+2	+2
0011	**3**	+3	+3	+3
0100	**4**	+4	+4	+4
0101	**5**	+5	+5	+5
0110	**6**	+6	+6	+6
0111	**7**	+7	+7	+7
1000	**8**	−0	−7	−8
1001	**9**	−1	−6	−7
1010	**10**	−2	−5	−6
1011	**11**	−3	−4	−5
1100	**12**	−4	−3	−4
1101	**13**	−5	−2	−3
1110	**14**	−6	−1	−2
1111	**15**	−7	−0	−1

3.5

EXCESS SYSTEM

Excess sysem

- Another representation that allows you to store both positive and negative numbers in a computer is call the Excess system
- A positive number, called the magic number, is used in the conversion process
- The magic number is normally (2^{N-1}) or $(2^{N-1}-1)$, where N is the bit allocation
- To represent a number in Excess,
  - Add the magic number to the integer
  - Change the result to binary and add 0s so that there is a total of N bits

Example 15

Represent –25 in Excess_127 using an 8-bit allocation.

Solution

First add 127 to get 102. This number in binary is 1100110. Add one bit to make it 8 bits in length. The representation is 01100110.

Example 16

Interpret 11111110 if the representation is Excess_127.

Solution

First change the number to decimal. It is 254. Then subtract 127 from the number. The result is decimal 127.

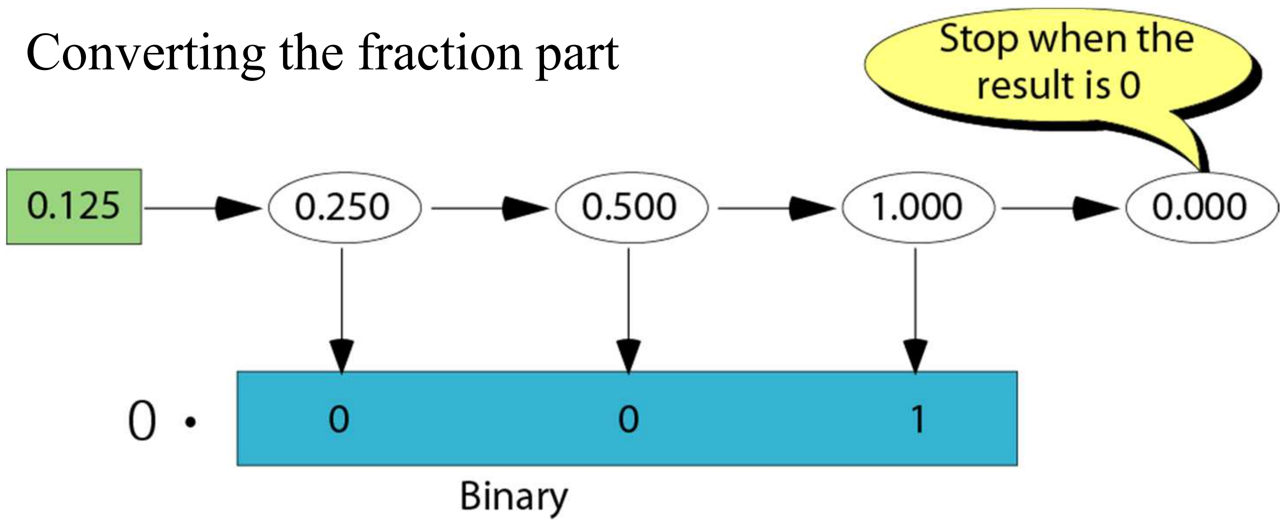
3.5

FLOATING-POINT REPRESENTATION

Changing fractions to binary

- A floating-point number is an integer and a fraction.
- Conversion floating-point number to binary
  - Convert the integer part to binary
  - Convert the fraction to binary
  - Put a decimal point between the two parts

Converting the fraction part



Example 17

Transform the fraction 0.875 to binary

Solution

Write the fraction at the left corner. Multiply the number continuously by 2 and extract the integer part as the binary digit. Stop when the number is 0.

0.875	→	1.750	→	1.5	→	1.0	→	0.0
0	.	1		1		1		

Example 18

Transform the fraction 0.4 to a binary of 6 bits.

Solution

Write the fraction at the left cornet. Multiply the number continuously by 2 and extract the integer part as the binary digit. You can never get the exact binary representation. Stop when you have 6 bits.

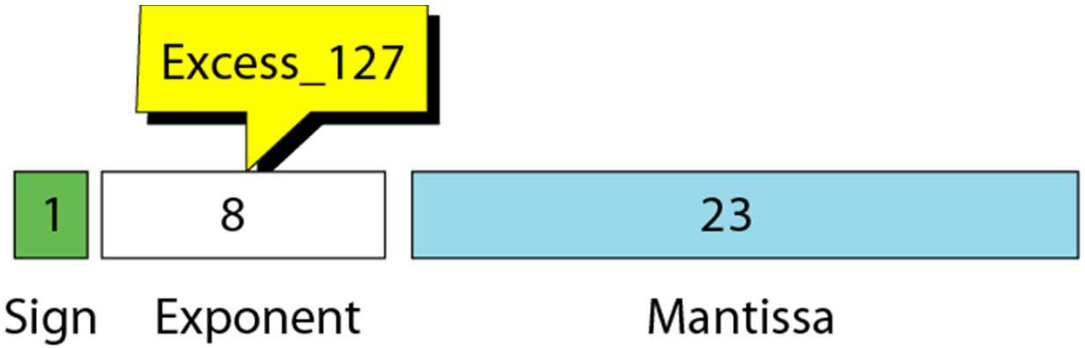
0.4	→	0.8	→	1.6	→	1.2	→	0.4	→	0.8	→	1.6
0	.	0		1		1		0		0		1

Normalization

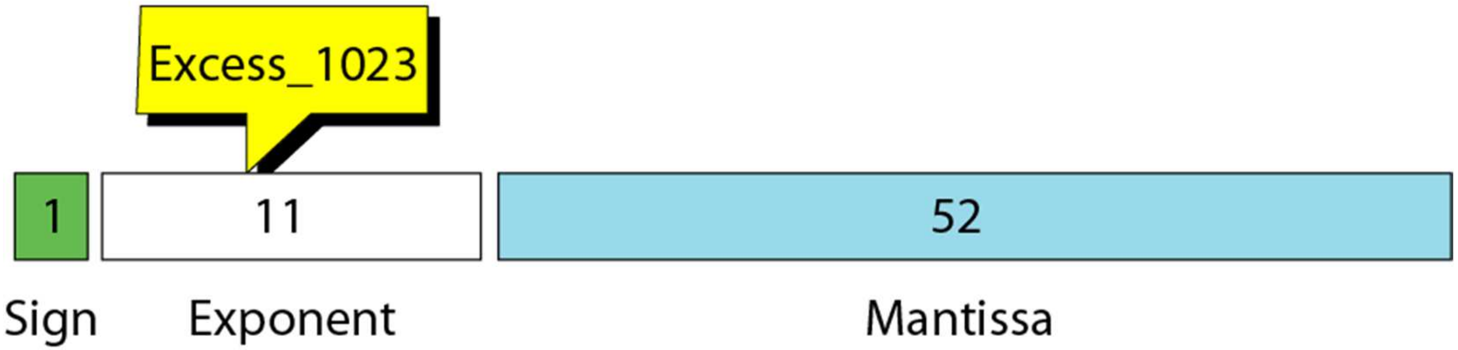
- A fraction is normalized so that operations are simpler
- Normalization: the moving of the decimal point so that there is only one 1 to the left of the decimal point.

Original Number	*Move*	*Normalized*
+1010001.1101	← 6	2^{6} x +1.01000111001
–111.000011	← 2	2^{2} x –1.11000011
+0.00000111001	6 →	2^{-6} x +1.11001
–0.001110011	3 →	2^{-3} x –1.110011

IEEE standards



a. Single Precision



b. Double Precision

Example 19

Show the representation of the normalized number $+ 2^6 \times 1.01000111001$

Solution

The sign is positive. The Excess_127 representation of the exponent is 133. You add extra 0s on the right to make it 23 bits. The number in memory is stored as:

0   10000101   01000111001000000000000

Example 20

Interpret the following 32-bit floating-point number

1 01111100 11001100000000000000000

Solution

The sign is negative. The exponent is –3 (124 – 127). The number after normalization is

$$-2^{-3} \times 1.110011$$

Example 21

Represent 81.5625 in IEEE standard

Solution

$81_{10} = 01010001_2$; $0.5625 = 0.1001_2$

$1010001.1001 = + 2^6 \times 1.0100011001$

Exponent 6 is expressed in Excess_127 as 133 = 10000101_2

0   10000101 01000110010000000000000

Example of floating-point representation

Number	*Sign*	*Exponent*	Mantissa
------------	----	-----------	-------------------------------
-2^{2} x 1.11000011	1	10000001	11000011000000000000000
$+2^{-6}$ x 1.11001	0	01111001	11001000000000000000000
-2^{-3} x 1.110011	1	01111100	11001100000000000000000