Chapter 3

Number Representation

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.

3.1

DECIMAL AND BINARY

Figure 3-1

Decimal system

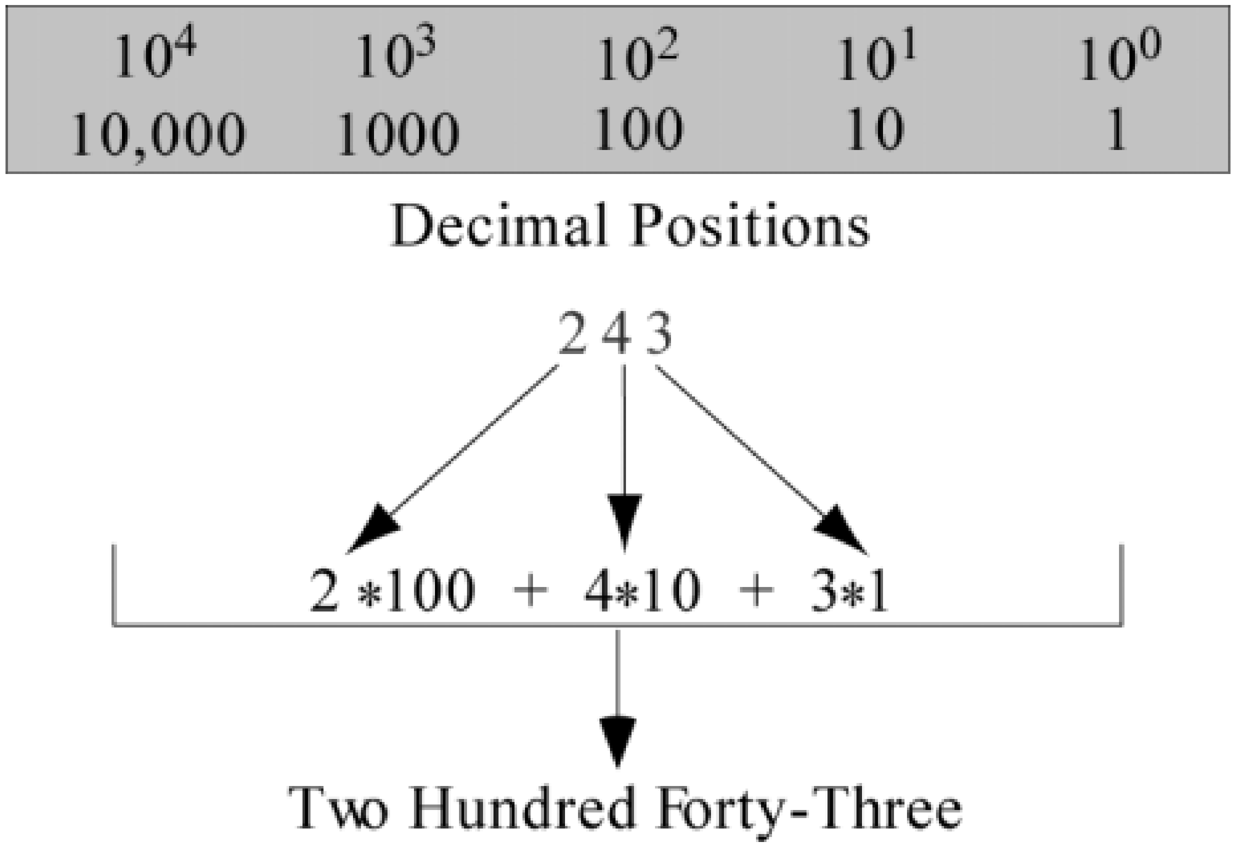
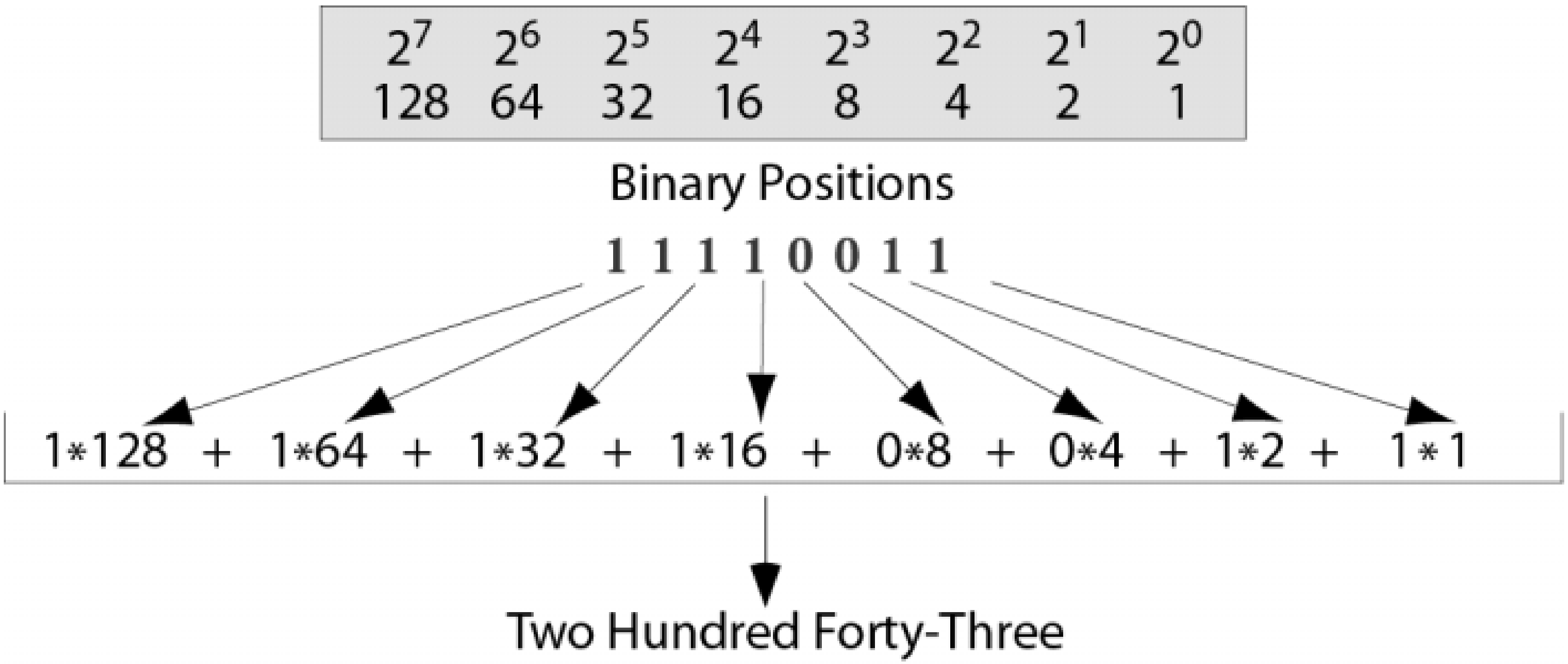


Figure 3-2

Binary system

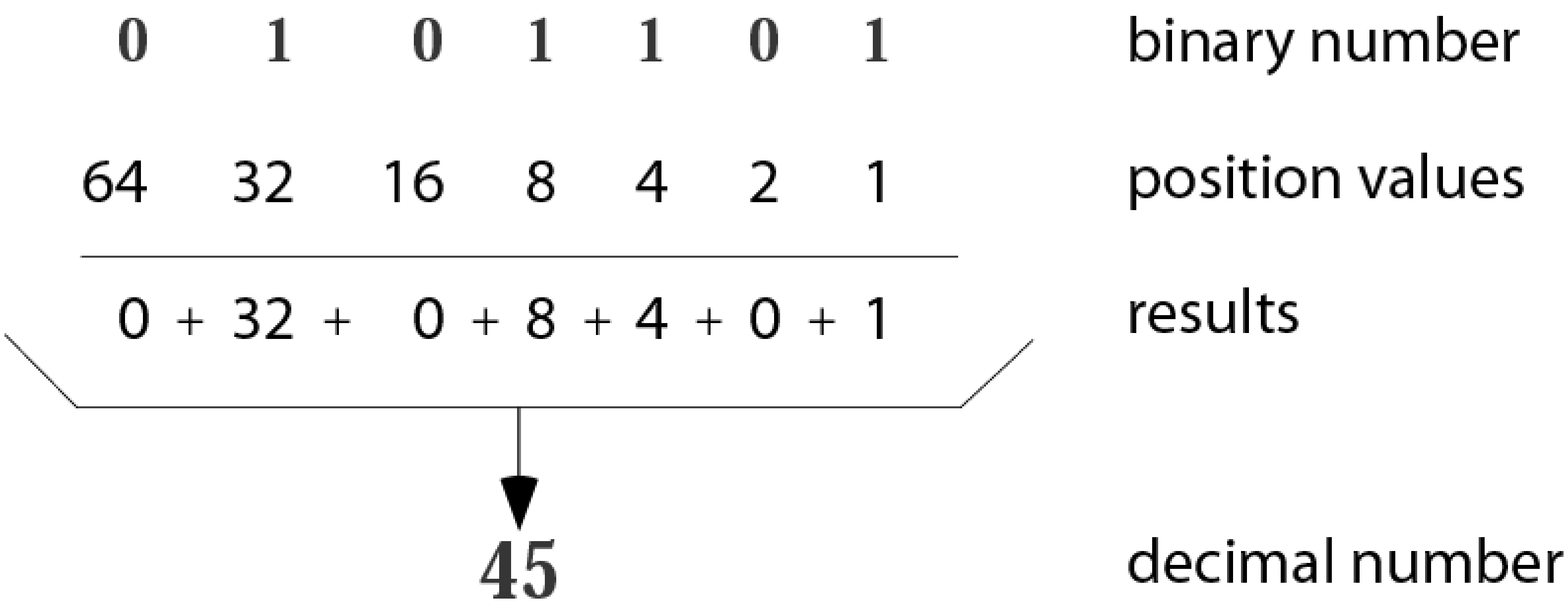


3.2

CONVERSION

Figure 3-3

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**

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**	

Figure 3-4

Decimal to binary conversion

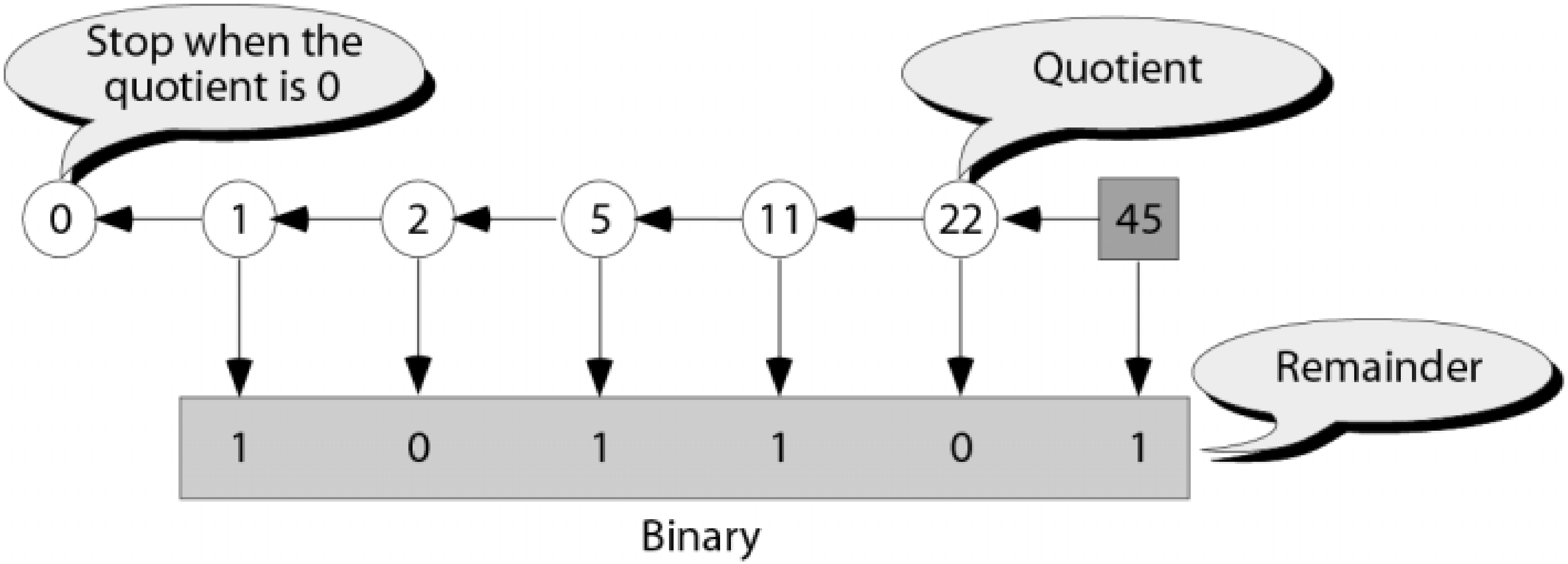
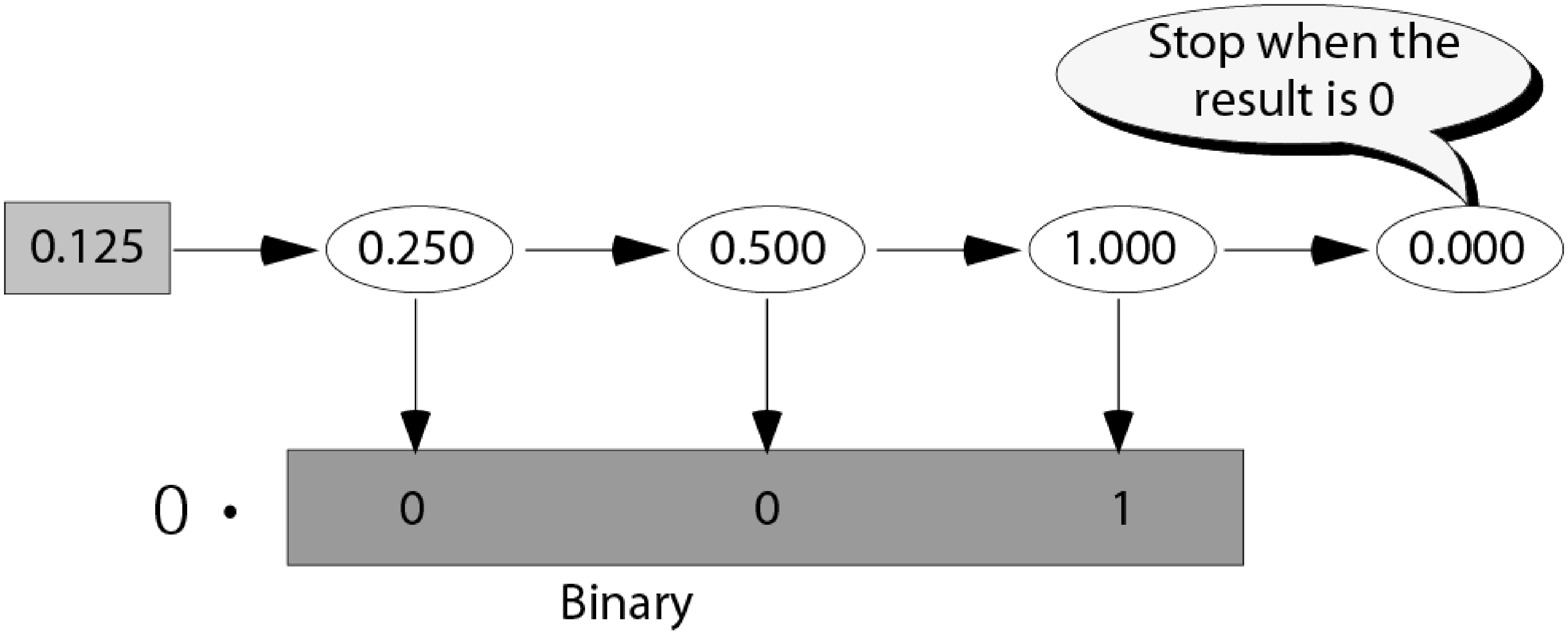


Figure 3-7

Changing fractions to binary



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.

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 may 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

3.4

INTEGER REPRESENTATION

Figure 3-5

Range of integers

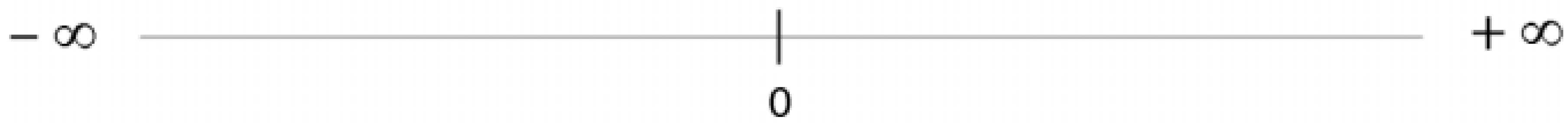


Figure 3-6

Taxonomy of integers

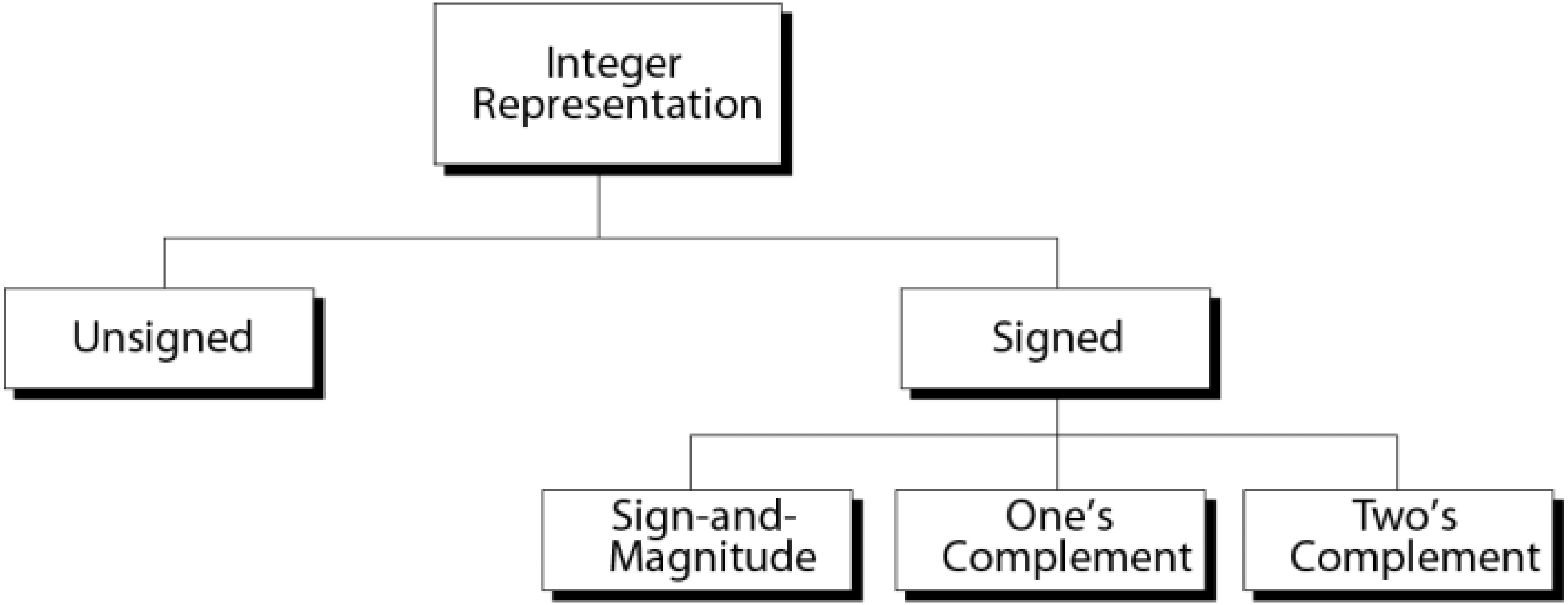


Table 3.1 Range of unsigned integers

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

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.

Table 3.2 Example of storing unsigned integers in two different computers

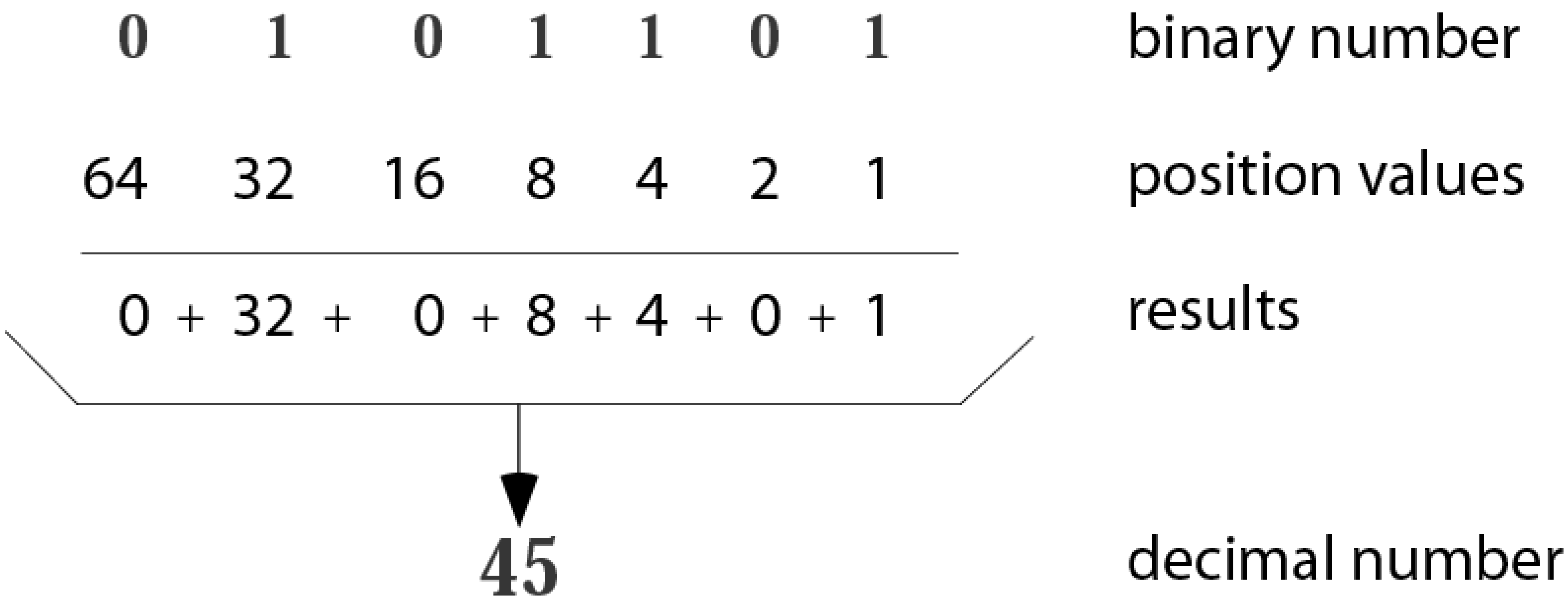
(一個為8-bit; 另一個為16-bit)

Decimal	*8-bit allocation*	*16-bit allocation*
	------------	--------------------------------
[redacted]	00000111	0000000000000111
[redacted]	11101010	0000000011101010
[redacted]	overflow	0000000100000010
[redacted]	overflow	0110000010111000
[redacted]	overflow	overflow

Example 5

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

Solution



Signed Number Representation

- Popular signed number representation:
  - Sign Magnitude (SM)
    - Easiest one
  - Diminished Radix Complement (DRC)
    - 1's complement
  - Radix Complement (RC)
    - 2's complement
    - The most popular in digital design
  - Positive numbers has the same representations in above mentioned system.

Sign Magnitude Number for 6-bit Word Size

SM number						Decimal number
d_5	d_4	d_3	d_2	d_1	d_0	
0	1	1	1	1	1	$+31 = +(2^5 - 1)$
0	1	1	1	1	0	+30
0	1	1	1	0	1	+29
0	1	1	1	0	0	+28
		⋮				
0	0	0	0	1	0	+2
0	0	0	0	0	1	+1
0	0	0	0	0	0	+0
1	0	0	0	0	0	−0
1	0	0	0	0	1	−1
1	0	0	0	1	0	−2
		⋮				
1	1	1	1	0	0	−28
1	1	1	1	0	1	−29
1	1	1	1	1	0	−30
1	1	1	1	1	1	$-31 = -(2^5 - 1)$

Figure 1.5.1 Sign magnitude (SM) numbers for a 6-bit word size.

2^n possibilities

2 for zeros:

0 00000

1 00000

$2^{n-1} - 1$ for positive and negative numbers

Table 3.3 Range of sign-and-magnitude integers

# of Bits	Range			
[redacted]	–127	–0	+0	+127
	–32767	–0	+0	+32767
	–2,147,483,647	–0	+0	+2,147,483,647

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.

Note:

There are two 0s in sign-and-magnitude

representation: positive and negative.

In an 8-bit allocation:

+0 → 00000000

-0 → 10000000

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

Table 3.4 Example of storing sign-and-magnitude integers in two computers

Decimal	*8-bit allocation*	*16-bit allocation*
	------------	------------------------------
[redacted]	00000111	0000000000000111
[redacted]	11111100	1000000001111100
[redacted]	overflow	0000000100000010
[redacted]	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.

One's complement Number for 6-bit Word Size

DRC number						Decimal number
d_5	d_4	d_3	d_2	d_1	d_0	
0	1	1	1	1	1	$+31 = +(2^5 - 1)$
0	1	1	1	1	0	+30
0	1	1	1	0	1	+29
0	1	1	1	0	0	+28
		⋮				
0	0	0	0	1	0	+2
0	0	0	0	0	1	+1
0	0	0	0	0	0	+0
1	1	1	1	1	1	−0
1	1	1	1	1	0	−1
1	1	1	1	0	1	−2
		⋮				
1	0	0	0	1	1	−28
1	0	0	0	1	0	−29
1	0	0	0	0	1	−30
1	0	0	0	0	0	$-31 = -(2^5 - 1)$

Figure 1.5.2 Diminished radix complement (DRC) numbers for a 6-bit word size.

2^n possibilities

2 for zeros:

0 00000

1 11111

$2^{n-1} - 1$ for positive and negative numbers

Note:

There are two 0s in one's complement

representation: positive and negative.

In an 8-bit allocation:

+0 → 00000000

-0 → 11111111

One’s complement
(一位補數法)

- If the sign is positive (0), no more action is needed;
- If the sign is negative, every bit is complemented.

Table 3.5 Range of one's complement integers

# of Bits	Range			
[redacted]	−127	−0	+0	+127
[redacted]	−32767	−0	+0	+32767
[redacted]	−2,147,483,647	−0	+0	+2,147,483,647

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

Table 3.6 Example of storing one's complement integers in two different computers

Decimal	*8-bit allocation*	*16-bit allocation*
[redacted]	------------	------------------------------
[redacted]	00000111	0000000000000111
[redacted]	11111000	1111111111111000
[redacted]	01111100	0000000001111100
[redacted]	10000011	1111111110000011
[redacted]	overflow	0110000010111000
[redacted]	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.

Note:

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

Two's complement

- 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

e.g. ***0000000000101000***

↓ 變成負數

1111111111011000

Two’s complement的另類看法

0000000000101000

1111111111011000

1. 0000000000101000

One’s complement ↓

1111111111010111

+1 ↓

1111111111011000

2. 2^{16}-***0000000000101000***

```
    10000000000000000
-)   0000000000101000
---------------------
     1111111111011000
```

Table 3.7 Range of two's complement integers

# of Bits	Range		
[redacted]	–128	0	+127
	–32,768	0	+32,767
	–2,147,483,648	0	+2,147,483,647

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.

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

Table 3.8 Example of storing two's complement integers in two different computers

Decimal	*8-bit allocation*	*16-bit allocation*
[redacted]	00000111	0000000000000111
[redacted]	11111001	1111111111111001
[redacted]	01111100	0000000001111100
[redacted]	10000100	1111111110000100
[redacted]	overflow	0110000010111000
[redacted]	overflow	1001111101001000

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.

Table 3.9 Summary of integer representation

Contents of Memory	Unsigned	Sign-and-Magnitude	One's Complement	Two's Complement
[redacted]	0	+0	+0	+0
	1	+1	+1	+1
	2	+2	+2	+2
	3	+3	+3	+3
	4	+4	+4	+4
	5	+5	+5	+5
	6	+6	+6	+6
	7	+7	+7	+7
	8	–0	–7	–8
	9	–1	–6	–7
	10	–2	–5	–6
	11	–3	–4	–5
	12	–4	–3	–4
	13	–5	–2	–3
	14	–6	–1	–2
	15	–7	–0	–1

3.5

EXCESS SYSTEM

Usage

- It is used to store the exponential value of a fraction.
  - See later section: Floating number representation.

- Usually use 2^n or $2^n - 1$

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

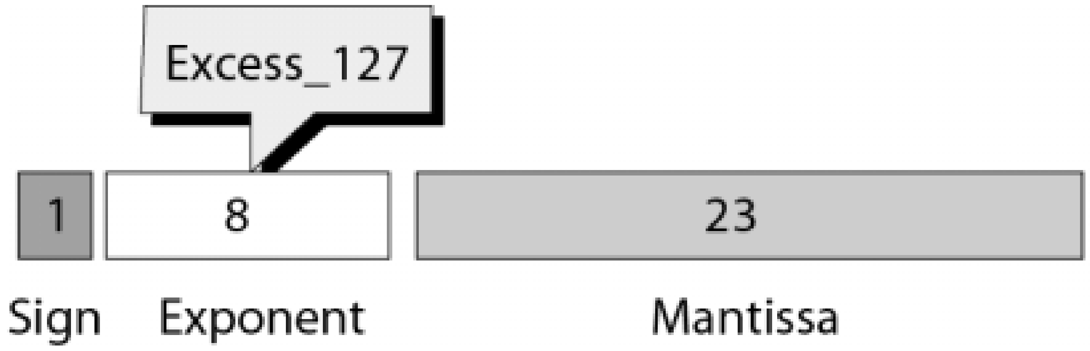
FLOATING-POINT REPRESENTATION

Table 3.10 Example of normalization

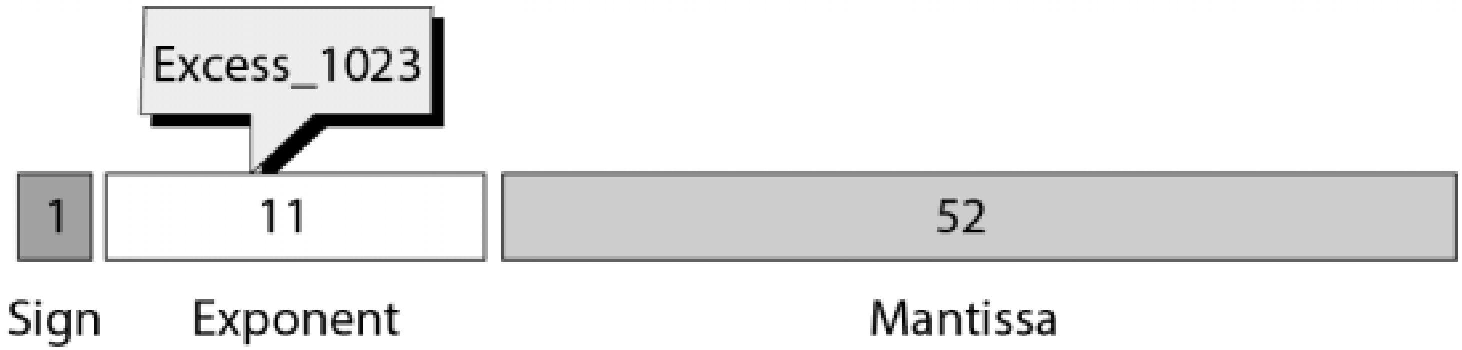
Original Number	*Move*	[redacted]
[redacted]	← 6	[redacted]
[redacted]	← 2	[redacted]
[redacted]	6 →	[redacted]
[redacted]	3 →	[redacted]

Figure 3-8

IEEE standards for floating-point representation



a. Single Precision



b. Double Precision

Example 19

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

the leftmost one is not stored.

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

Table 3.11 Example of floating-point representation

[redacted]	*Sign*	*Exponent*	[redacted]
[redacted]	----	-----------	[redacted]
[redacted]	1	10000001	[redacted]
[redacted]	0	01111001	[redacted]
[redacted]	1	01111100	[redacted]

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

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

3.6

HEXADECIMAL NOTATION

Hexadecimal

- Hexadecimal=> 16 number system (0~9,A~F)
- Conversion between binary and hexadecimal
- Hexadecimal:
  - Digit set:{0~9,A~F}

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

Binary to Hexadecimal

- An example:

1111	1110	0011	0001	1010	1011	0000	0001
F	E	3	1	A	B	0	1

- $(11111110001100011010101100000001)_2 =$

$(FE31AB01)_{16}$

Hexadecimal to Binary

- An example:

D	2	C	B	0
1101	0010	1100	1011	0000

- $(D2CB0)_{16}=(11010010110010110000)_2$