

Representing Information as Bit Patterns pp.35

Representing Text

- **Each character (letter, punctuation, etc.) is assigned a **unique** bit pattern.**
 - ASCII: Uses patterns of 7-bits to represent most symbols used in written English text
 - Unicode: Uses patterns of 16-bits to represent the major symbols used in languages world side

西文字符：ACSCII码

(American Standard Code for Information Interchange)

用7位二进制编码，最高位0

问题：为什么用7位？

0~127共可表示128个字符

‘A’~ ‘Z’ 26

‘a’~ ‘z’ 26

‘0’~ ‘9’ 10

其他键盘字符、控制键

≤ 128

0~32、127为非图形字符，其余94个图形字符

Printable ASCII Codes

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	space	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

❖ Examples:

✧ ASCII code for space character = 20 (hex) = 32 (decimal)

✧ ASCII code for 'L' = 4C (hex) = 76 (decimal)

✧ ASCII code for 'a' = 61 (hex) = 97 (decimal)

Figure 1.13 The message “Hello.” in ASCII

01001000	01100101	01101100	01101100	01101111	00101110
H	e	l	l	o	.

Exercise

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	space	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

- 01000011 01001000 01001001
01001110 01000001

汉字编码



(1) 输入码

音码类：全拼、双拼、微软拼音、自然码和智能ABC等

形码类：五笔字型法、郑码输入法、表形码等。

(2) 国标码 (GB2312)

每个汉字占两个字节，为什么？

一级汉字：3755个；二级汉字：3008个。

汉字 94×94 的矩阵，即94个区（7bit）和94个位（7bit），由区号和位号构成汉字的**区位码**。

区号	位号
----	----

汉字的国标码与区位码的关系：

每个汉字的区号和位号各加32 (20H) 就构成了国标码

加32的原因： 为了与ASCII码兼容，每个字节值大于32
(0~32为非图形字符码值)

(3) 机内码

汉字在设备或信息处理系统内部最基本的表达形式。

为了在计算机内部能够区分是汉字编码还是ASCII码，将国标码每个字节最高位设置为1 (80H 1000 0000B)。

国标码 “中” (56 50)H (0 1010110 0 1010000)B

机内码 (D6 D0)H (1 1010110 1 1010000)B

三种码之间关系：

机内码=国标码+80 80H=区位码+A0 A0H

(4) 汉字字形码

字库（宋体、楷体）

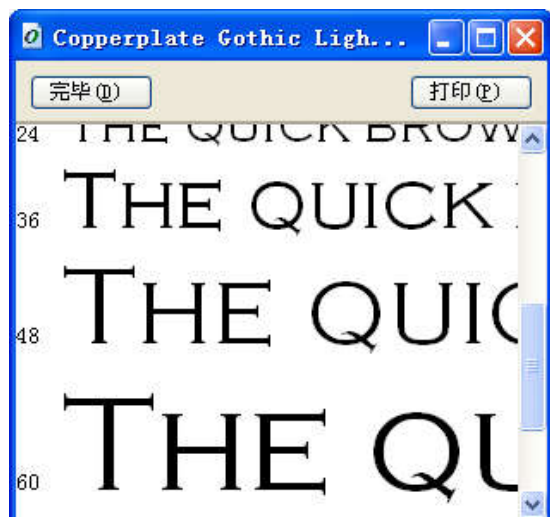
点阵：汉字字形点阵的代码

有 16×16 、 24×24 、 32×32 、 48×48 等
编码、存储方式简单、无需转换直接输出
放大后产生的效果差

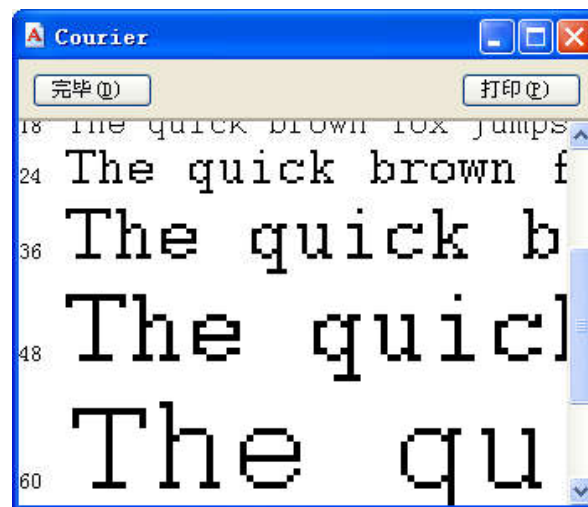
矢量：存储的是描述汉字字形的轮廓特征
矢量方式特点正好与点阵相反

Adobe Type1, True type

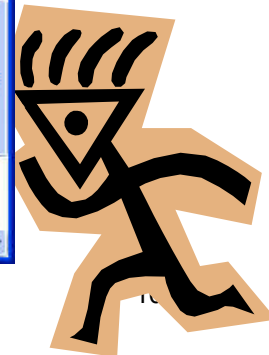
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	十六进制码
0							●	●									0 3 0 0
1							●	●									0 3 0 0
2							●	●									0 3 0 0
3							●	●						●			0 3 0 4
4	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		F F F E
5							●	●									0 3 0 0
6							●	●									0 3 0 0
7							●	●									0 3 0 0
8							●	●									0 3 0 0
9							●	●	●								0 3 8 0
10							●	●		●							0 6 4 0
11								●			●						0 C 2 0
12								●	●			●	●				1 8 3 0
13								●				●	●				1 0 1 8
14													●	●			2 0 0 C
15	●	●												●	●	●	C 0 0 7



矢量



点阵



不同字符编码的存储

ASCII

0xxxxxxx

GB

0xxxxxxx

0xxxxxxx

GB机
内码

1xxxxxxx

1xxxxxxx

因为计算机只能处理数字，如果要处理文本，就必须先把文本转换为数字才能处理。最早的计算机在设计时采用8个比特（bit）作为一个字节（byte），所以，一个字节能表示的最大的整数就是255（二进制11111111=十进制255），如果要表示更大的整数，就必须用更多的字节。比如两个字节可以表示的最大整数是 65535，4个字节可以表示的最大整数是 4294967295。

由于计算机是美国人发明的，因此，最早只有127个字符被编码到计算机里，也就是大小写英文字母、数字和一些符号，这个编码表被称为 ASCII 编码，比如大写字母 A 的编码是 65，小写字母 z 的编码是 122。

但是要处理中文显然一个字节是不够的，至少需要两个字节，而且还不能和ASCII编码冲突，所以，中国制定了 GB2312 编码，用来把中文编进去。

你可以想得到的是，全世界有上百种语言，日本把日文编到 Shift_JIS 里，韩国把韩文编到 Euc-kr 里，各国有各国的标准，就会不可避免地出现冲突，结果就是，在多语言混合的文本中，显示出来会有乱码。



因此，Unicode应运而生。Unicode把所有语言都统一到一套编码里，这样就不会再有乱码问题了。

Unicode标准也在不断发展，但最常用的是用两个字节表示一个字符（如果要用到非常偏僻的字符，就需要4个字节）。现代操作系统和大多数编程语言都直接支持Unicode。

现在，捋一捋ASCII编码和Unicode编码的区别：ASCII编码是1个字节，而Unicode编码通常是2个字节。

字母 **A** 用ASCII编码是十进制的 **65**，二进制的 **01000001**；

字符 **0** 用ASCII编码是十进制的 **48**，二进制的 **00110000**，注意字符 **'0'** 和整数 **0** 是不同的；

汉字 **中** 已经超出了ASCII编码的范围，用Unicode编码是十进制的 **20013**，二进制的 **01001110 00101101**。

你可以猜测，如果把ASCII编码的 **A** 用Unicode编码，只需要在前面补0就可以，因此，**A** 的Unicode编码是 **00000000 01000001**。

新的问题又出现了：如果统一成Unicode编码，乱码问题从此消失了。但是，如果你写的文本基本上全部是英文的话，用Unicode编码比ASCII编码需要多一倍的存储空间，在存储和传输上就十分不划算。

所以，本着节约的精神，又出现了把Unicode编码转化为“可变长编码”的 **UTF-8** 编码。UTF-8编码把一个Unicode字符根据不同的数字大小编码成1-6个字节，常用的英文字母被编码成1个字节，汉字通常是3个字节，只有很生僻的字符才会被编码成4-6个字节。如果你要传输的文本包含大量英文字符，用UTF-8编码就能节省空间：

字符	ASCII	Unicode	UTF-8
A	01000001	00000000 01000001	01000001
中	x	01001110 00101101	11100100 10111000 10101101

在计算机内存中，统一使用Unicode编码，当需要保存到硬盘或者需要传输的时候，就转换为UTF-8编码。

用记事本编辑的时候，从文件读取的UTF-8字符被转换为Unicode字符到内存里，编辑完成后，保存的时候再把Unicode转换为UTF-8保存到文件：



所以你看很多网页的源码上会有类似 `<meta charset="UTF-8" />` 的信息，表示该网页正是用的UTF-8编码。

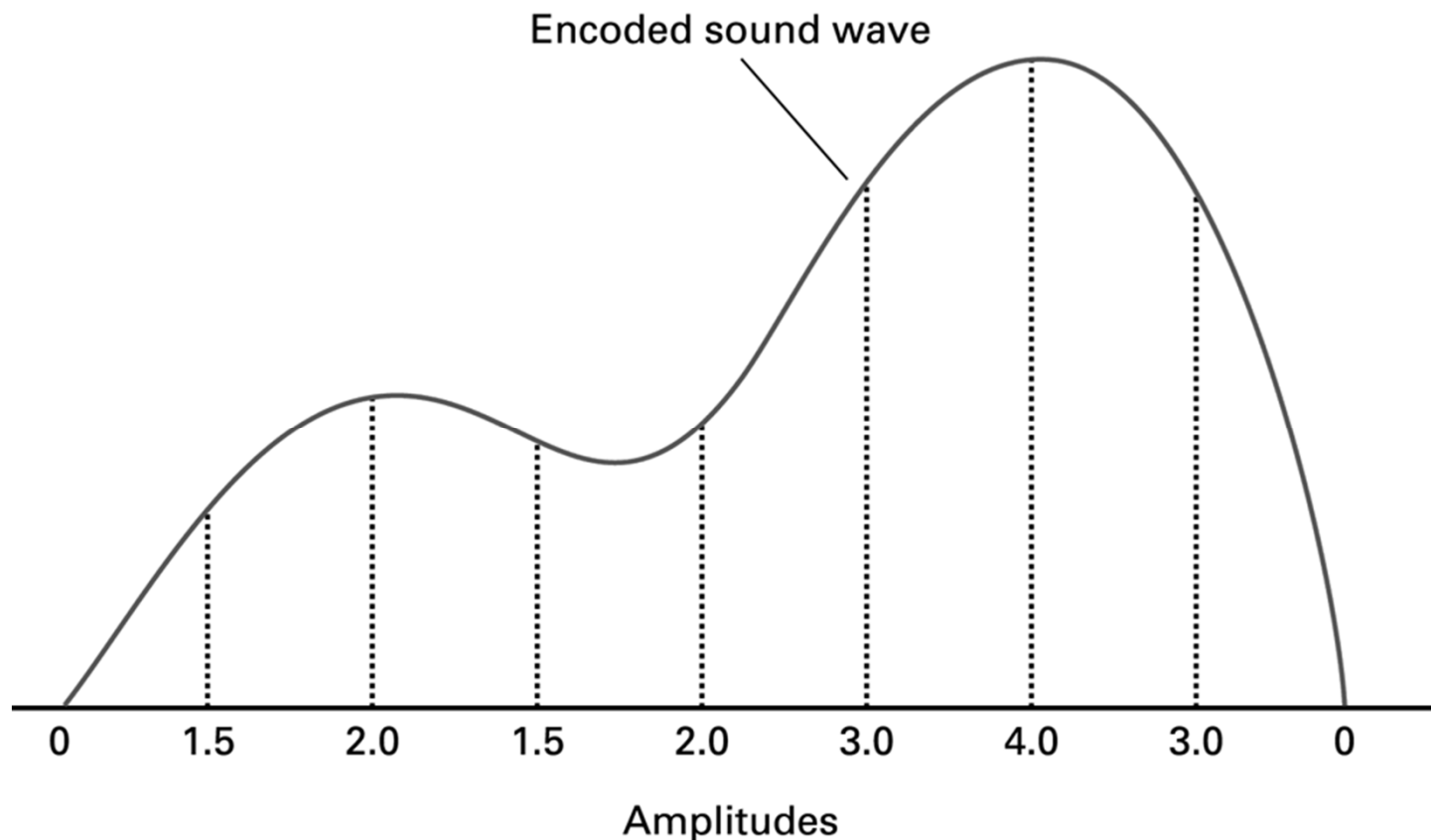
Representing Numeric Values

- **Binary notation:** Uses bits to represent a number in base two
- Limitations of computer representations of numeric values
 - Overflow (溢出) – occurs when a value is too big to be represented
 - Truncation (截断) – occurs when a value cannot be represented accurately

Representing Sound

- Sampling techniques
 - Used for high quality recordings
 - Records actual audio
- MIDI
 - Used in music synthesizers
 - Records “musical score”

Figure 1.14 The sound wave represented by the sequence 0, 1.5, 2.0, 1.5, 2.0, 3.0, 4.0, 3.0, 0



存储声音

- 每秒存储声音容量的公式

采样频率Hz × 采样精度bit / 8 × 声道数 = 每秒数据量Byte

- 例：用44.10KHz的采样频率，每个采样点用16位的精度存储，则录制1秒的立体声（双声道）节目，其文件所需存储量为：

$$44100 \times 16 / 8 \times 2 = 176.4 \text{KB}$$

Representing Images



Representing Images

- Bit map techniques
 - Pixel像素
 - RGB
 - 存储图像: 行 \times 列 \times 颜色深度=()B
- Vector techniques
 - Scalable

3或1字节
彩色、灰度

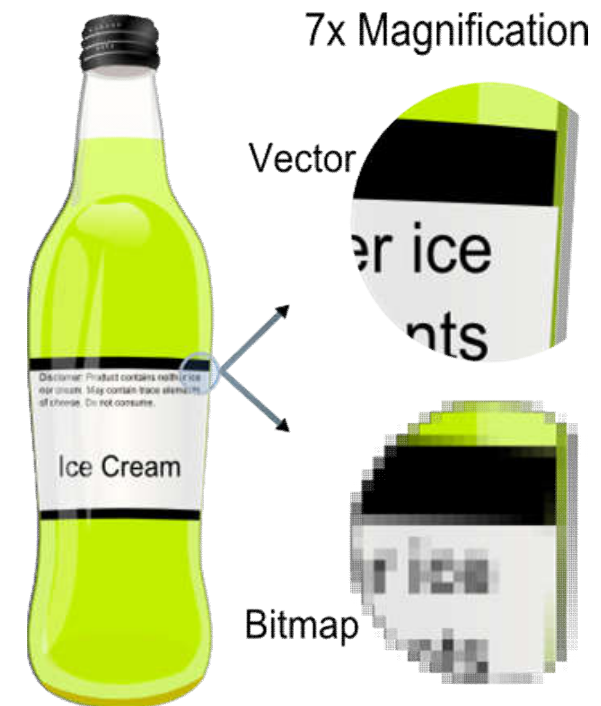
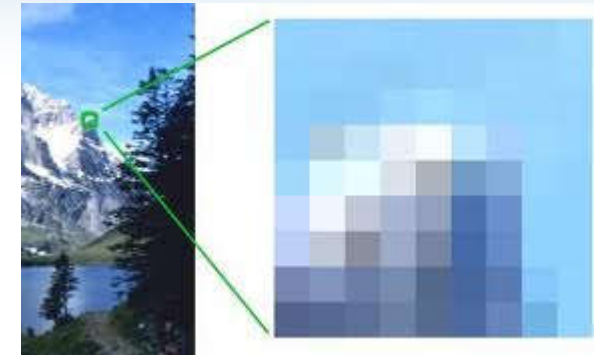
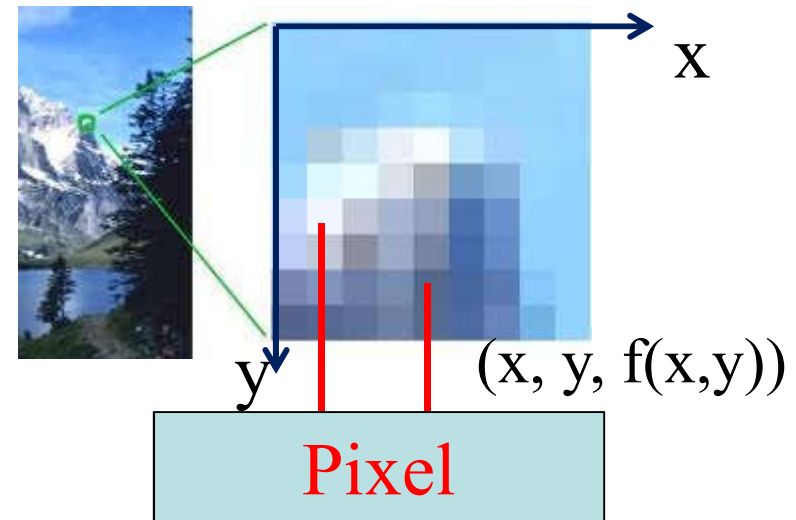


Image & pixel

Digital bitmap images are made up of pixels in a grid

Pixel:

- The smallest display element
- Associate with the position and color value



Color depth



Black & white



Gray scale



Color

Black & white

- Black & white images (**binary images**)



$$f(x, y) \in \{0, 1\}$$

Gray scale



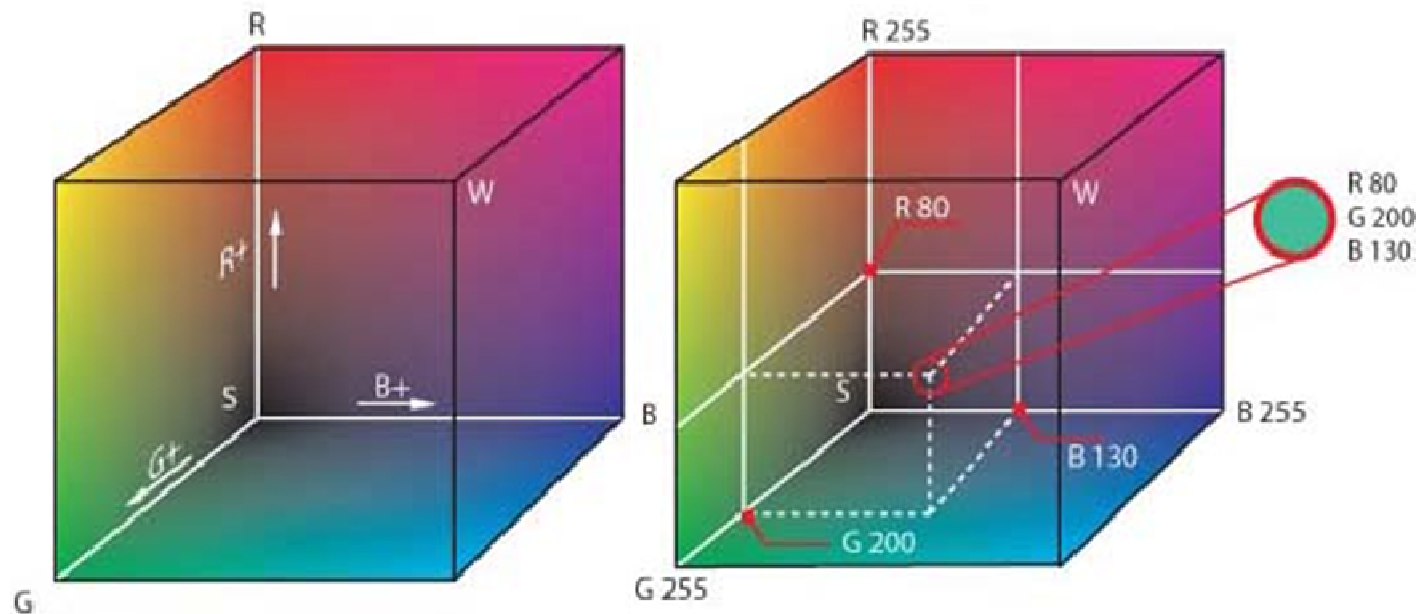
$$f(x, y) \in \{0, 1, \dots, 255\}$$



- Many shades of gray in between

Color

- Typical color representation: RGB model



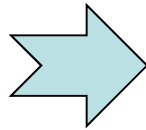
3 channels: **red, green, blue**

$R \in \{0,1,\dots,255\}$

$G \in \{0,1,\dots,255\}$

$B \in \{0,1,\dots,255\}$

Color



$$f(x, y) = \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad \text{with} \quad \begin{cases} R \in \{0, 1, \dots, 255\} \\ G \in \{0, 1, \dots, 255\} \\ B \in \{0, 1, \dots, 255\} \end{cases}$$

Required memory space

Color depth	Black & white	Gray	Color
Required bits	1	8	8*3=24

$(\text{Number of columns} * \text{number of rows} * \text{required bits}) / 8$
= required memory space (**Byte**)

Example:

For a 24-bit color image (1280 columns, 1024 rows), the required memory space is

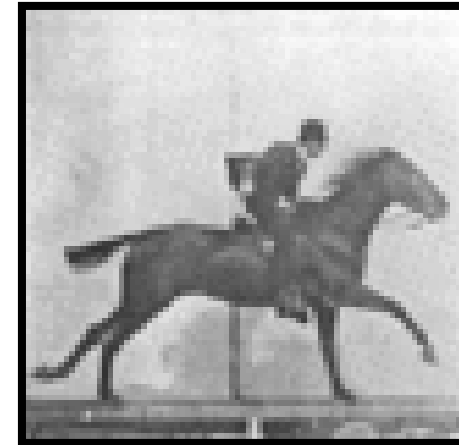
$$1280 * 1024 * 24 / 8 = 4 \text{ MB}$$

Video

- Video
 - Composed of a sequence of images, each image is called a **frame**
 - Played at a certain **frame rate**

Frame rate:

The number of images per unit of time of video



Required space for video

- If not compressed, the required memory space for a video (1 min)

Required memory space for a video = required space for each image * frame rate * 60

For a video (1280 columns, 1024 rows and 24bits for each frame) with a frame rate of 30, the required memory space is:

$$1280 * 1024 * 24 / 8 * 30 * 60 = 6.6 \text{GB}$$

Data Compression pp.58

- Lossy versus lossless
- Run-length encoding
- Frequency-dependent encoding
(Huffman codes)
- Relative encoding
- Dictionary encoding (Includes adaptive dictionary encoding such as LZW encoding.)

Compressing Images

- GIF: Good for cartoons
- JPEG: Good for photographs
- TIFF: Good for image archiving

Compressing Audio and Video

- MPEG / H.264
 - High definition television broadcast
 - Video conferencing
- 多视角视频编码格式 (MVC, Multiview Video Coding)
- MP3

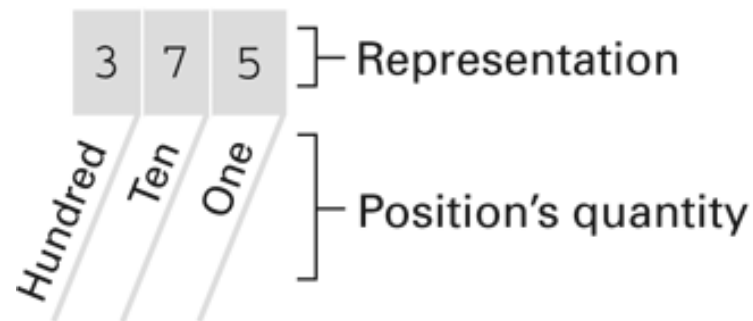
1.5 The Binary System pp.42

The traditional decimal system is based on powers of ten.

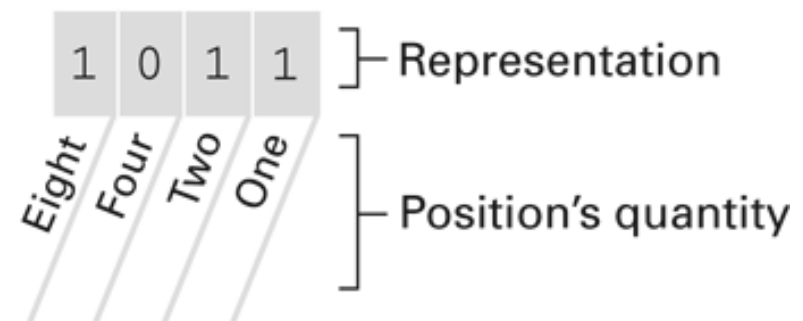
The Binary system is based on powers of two.

Figure 1.15 The base ten and binary systems

a. Base ten system



b. Base two system



$$\begin{aligned} 541.25_{10} &= 5 \times 10^2 + 4 \times 10^1 + 1 \times 10^0 + 2 \times 10^{-1} + 5 \times 10^{-2} \\ &= (500)_{10} + (40)_{10} + (1)_{10} + (2/10)_{10} + (5/100)_{10} \\ &= (541.25)_{10} \end{aligned}$$

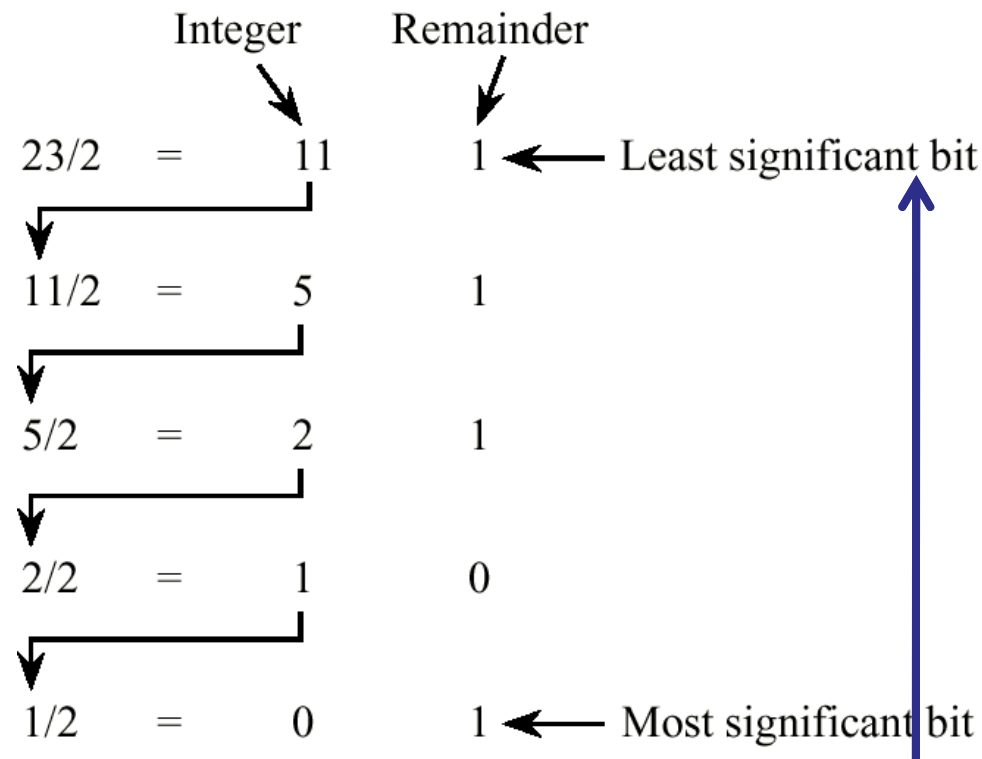
$$10101(\text{B}) = 2^4 + 2^2 + 2^0 = 16 + 4 + 1 = 21$$

$$101.11(\text{B}) = 2^2 + 2^0 + 2^{-1} + 2^{-2} = 5.75$$

Binary to decimal

Converting decimal to binary

Example: Convert 23.375_{10} to base 2. Start by converting the integer portion:



$$(23)_{10} = (10111)_2$$

stop when quotient is zero

- Now, convert the fraction:

$$\begin{array}{rclcl}
 .375 & \times & 2 & = & 0.75 \\
 \downarrow & & & & \uparrow \\
 .75 & \times & 2 & = & 1.5 \\
 \downarrow & & & & \uparrow \\
 .5 & \times & 2 & = & 1.0
 \end{array}$$

The diagram illustrates the conversion of the decimal fraction .375 to binary. It shows three steps of multiplying by 2. In each step, the integer part of the result is the next binary digit (1, 1, 1). The fractional part is carried over to the next step. A vertical blue box highlights the integer parts (0, 1, 1) and a light blue arrow points down from the top of the box to the bottom, indicating the sequence of bits.

$$(.375)_{10} = (.011)_2$$

- Putting it all together, $23.375_{10} = 10111.011_2$

Nonterminating Base 2 Fraction

- We can't always convert a terminating base 10 fraction into an equivalent terminating base 2 fraction:


$$\begin{array}{rclcl} .2 & \times & 2 & = & 0.4 \\ \hline \downarrow & & & & \\ .4 & \times & 2 & = & 0.8 \\ \hline \downarrow & & & & \\ .8 & \times & 2 & = & 1.6 \\ \hline \downarrow & & & & \\ .6 & \times & 2 & = & 1.2 \\ \hline \downarrow & & & & \\ .2 & \times & 2 & = & 0.4 \\ & & & & \vdots \end{array}$$

Hexadecimal

Decimal	Binary	Octal	Hexadecimal
00	0000	00	0
01	0001	01	1
02	0010	02	2
03	0011	03	3
04	0100	04	4
05	0101	05	5
06	0110	06	6
07	0111	07	7
08	1000	10	8
09	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Conversion between hexadecimal and binary numbers

❖ Each hexadecimal digit corresponds to 4 binary bits


11 0110 1111.1101 01 (B) = 36F.D4(H)
3 6 F D 4

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

Some Exercises

- Write down the computation steps
 - $(110111.01)_B = (\quad)_D$
 - $(01101101)_B = (\quad)_H$

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

Answers to Exercises

- $(110111.01)_B = 32 + 16 + 4 + 2 + 1 + 0.25 = (55.25)_D$
- $01101101_2 = (0110_2)(1101_2) = 6D_{16}$

Figure 1.19 The binary addition facts

$$\begin{array}{r} 0 \\ + 0 \\ \hline 0 \end{array}$$

$$\begin{array}{r} 1 \\ + 0 \\ \hline 1 \end{array}$$

$$\begin{array}{r} 0 \\ + 1 \\ \hline 1 \end{array}$$

$$\begin{array}{r} 1 \\ + 1 \\ \hline 10 \end{array}$$

Binary Addition

- Start with the rightmost bit
- Add each pair of bits
- Include the carry in the addition, if present

<u>carry</u>								
	1	1	1	1				
	0	0	1	1	0	1	1	0
								(54)
<u>+</u>	0	0	0	1	1	1	0	1
								(29)
<hr/>								
	0	1	0	1	0	0	1	1
								(83)
<u>bit position:</u>	7	6	5	4	3	2	1	0

Binary Subtraction

$$00001000 - 00000100 = ?$$

$$8 - 4 = 8 + (-4) = 4$$

How to represent signed numbers?

Signed Integers 带符号整数

Sign bit:
0-positive

- How to represent a signed number?
 - Signed magnitude (原码)
 - 2's complement (二进制补码)
- Divide the range of values into 2 equal parts
 - First part corresponds to the positive numbers (≥ 0)
 - Second part correspond to the negative numbers (< 0)

8-bit Binary value	Unsigned value	Signed value
00000000	0	0
00000001	1	+1
00000010	2	+2
...
01111110	126	+126
01111111	127	+127
10000000	128	-128
10000001	129	-127
...
11111110	254	-2
11111111	255	-1

Sign bit:
1-negative

Signed Magnitude

- ❖ Also known as “sign and magnitude,” the leftmost bit is the sign (0 = positive, 1 = negative) and the remaining bits are the magnitude.
- ❖ Example: $+25_{10} = 00011001_2$ $-25_{10} = 10011001_2$
- ❖ **Two representations for zero:** $+0 = 00000000_2$, $-0 = 10000000_2$.
- ❖ Example: $(-5)+4 \longrightarrow 10000101+00000100=10001001 \longrightarrow (-9)D ?$

Solution: 2's complement representation

Two's Complement 二进制补码

- One representation for zero:
 $+0 = 00000000_2$, $-0 = 00000000_2$.
- Positive number:
2's complement = original number
- Negative number

starting value (negative numbers)	10100100 = -36
step1: keep the sign bit and reverse other bits	11011011
step 2: add 1 to the value from step 1	+ 1
sum = 2's complement representation	11011100 = -92

Binary Subtraction

- When subtracting $A - B$, convert B to its 2's complement
- Add A to $(-B)$, if the result is negative (leftmost bit=1), compute 2's complement again
- $-5+4$?

- 在计算机系统中，数值一律用补码来表示和存储。原因在于，使用补码，可以将符号位和数值位统一处理；同时，加法和减法也可以统一处理。此外，补码与原码的相互转换，其运算过程是相同的，不需要额外的硬件电路。

Overflow

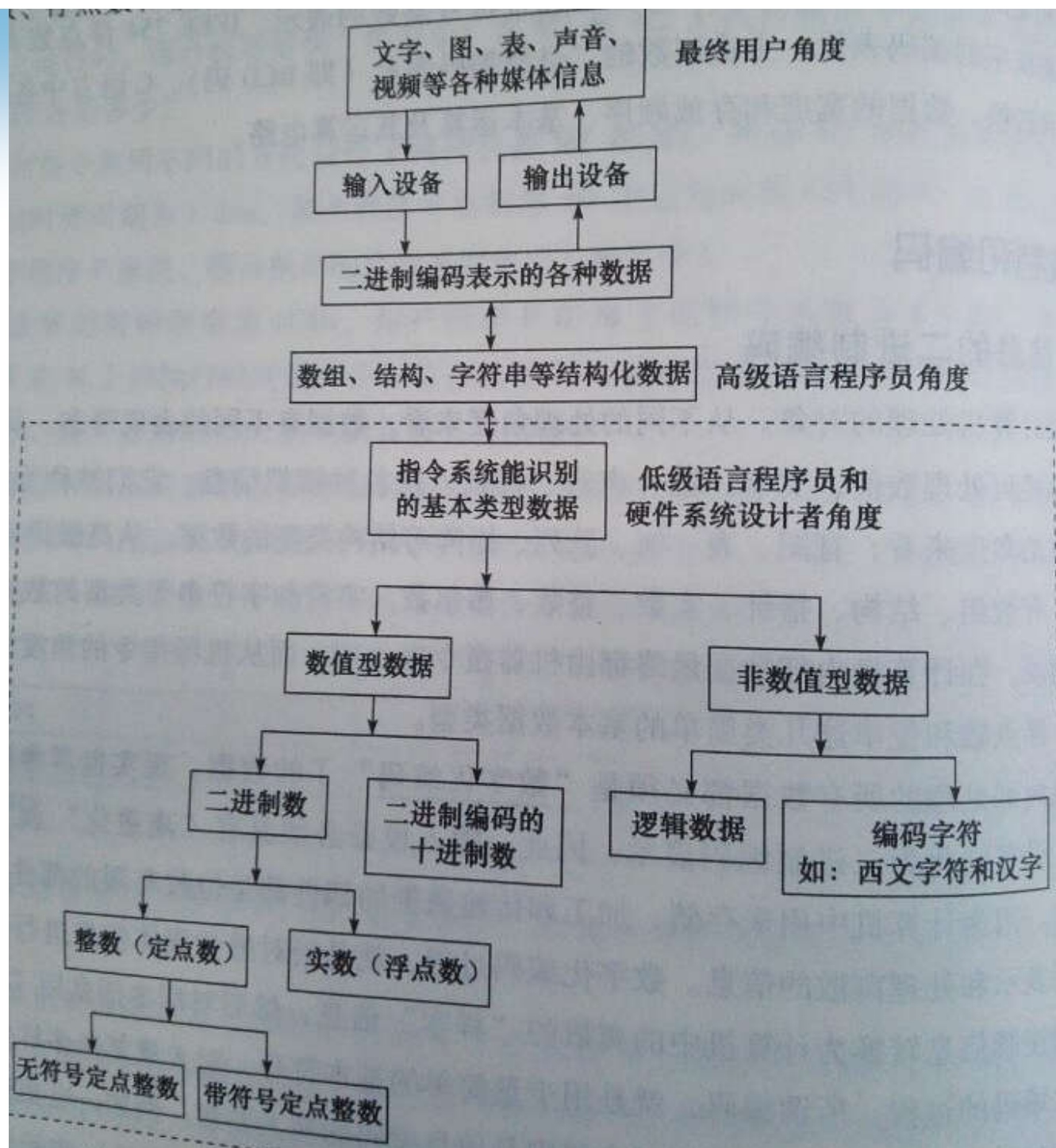
- When two numbers are added that have large magnitudes and the same sign, an **overflow** will occur if the result is too large to fit in the number of bits used in the representation

$$\begin{array}{r} 01010000 \\ + 00110010 \\ \hline 10000010 \end{array} \quad \begin{array}{l} (+80)_{10} \\ (+50)_{10} \\ \\ (-126)_{10} \end{array}$$

- The result should be $(+130)_{10}$, which is bigger than $+127$ (maximum for 8-bit)

Overflow

- Overflow occurs when
 - Adding two positive numbers and the sum is negative
 - Adding two negative numbers and the sum is positive
- If the numbers being added are of opposite signs, then an overflow will never occur

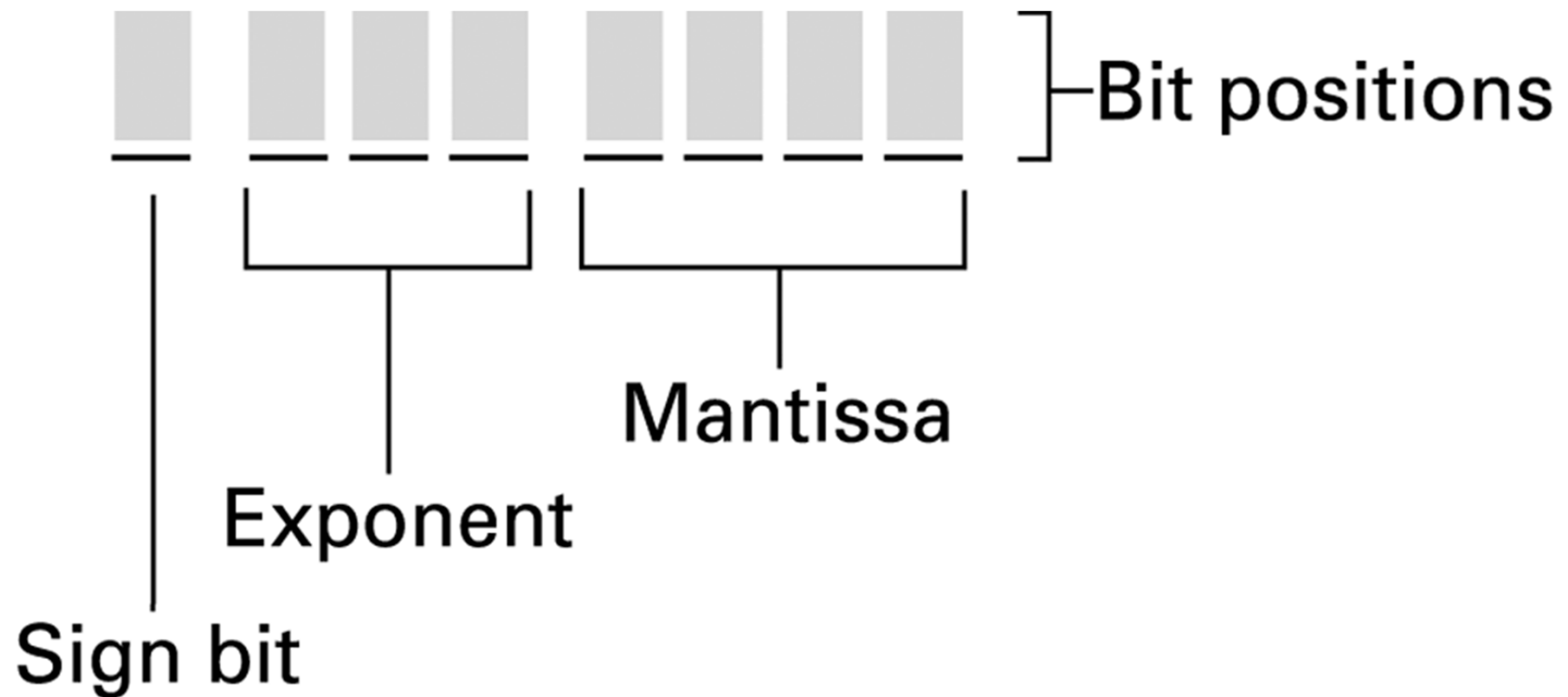


计算机外部信息与内部数据的转换

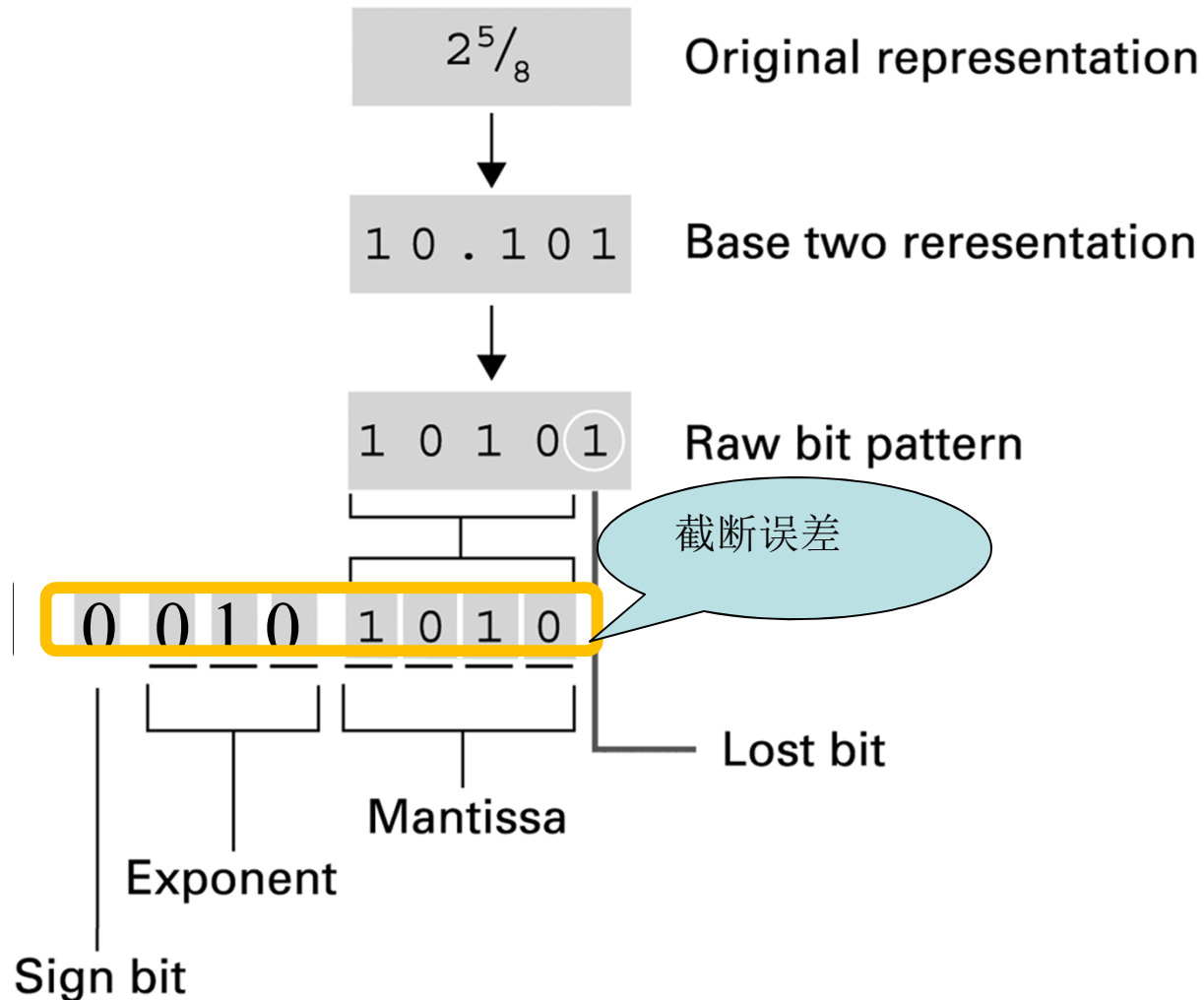
Storing Fractions 存储浮点数

- **Floating-point Notation:** Consists of a sign bit, a mantissa (尾数) field, and an exponent (指数) field.
- Related topics include
 - Normalized form
 - Truncation errors (截断误差)

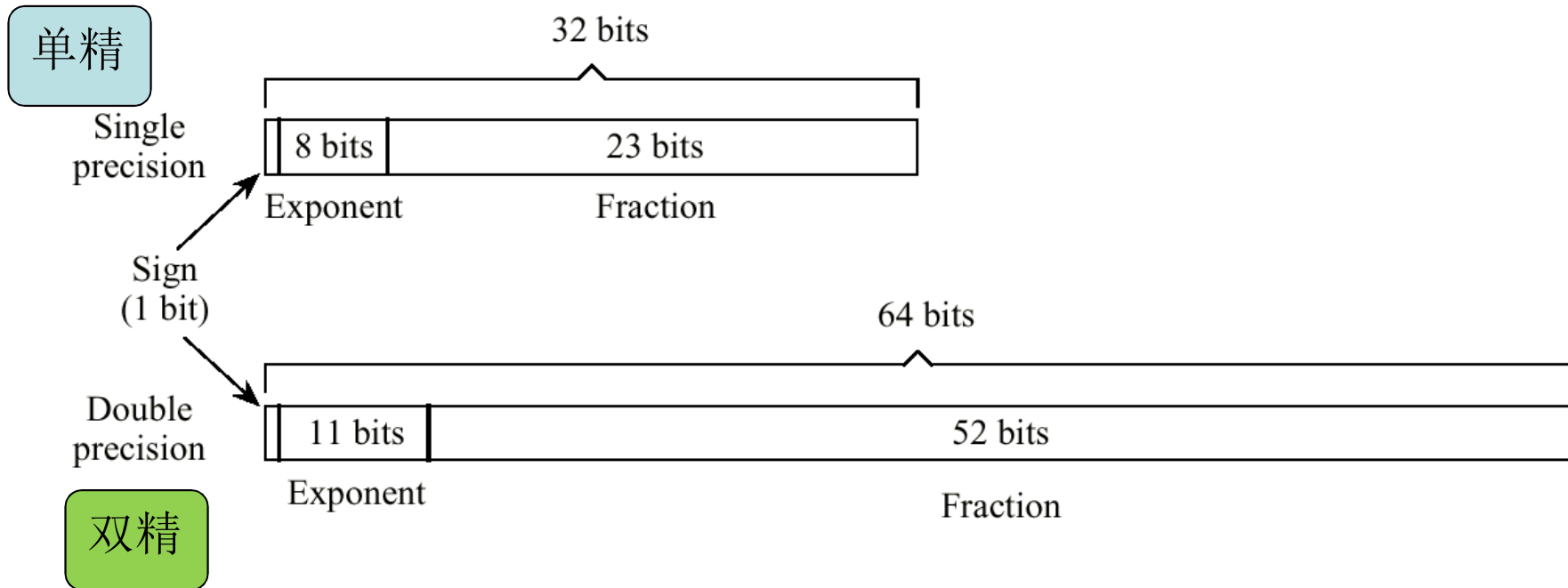
Figure 1.26 Floating-point notation components



Encoding the value $2\frac{5}{8}$



IEEE-754 Floating Point Formats



IEEE二进制浮点数算术标准（**IEEE 754**）是最广泛使用的浮点数运算标准，定义了表示浮点数的格式（包括负零-0）与反常值（denormal number），一些特殊数值（无穷（Inf）与非数值（NaN））



- IEEE 754 uses a bias (偏移值) of 127 for single precision (1023 for double precision)
- Biased exponent means that the value represented by a floating-point number is



p+127
P+1023

d

$$(-1)^S \times (1+d) \times 2^p$$

S: value of sign bit

d: value of fraction field

p: value of exponent field

- 指数有符号（补码），需将值调整到无符号数的范围内以便进行比较
- IEEE 754标准规定该固定值为 $2^{e-1}-1$ ，e为存储指数的比特的长度

$$\text{Number} = \pm \underbrace{1.\text{xx}...\text{xxx}}_{\text{d}} \times 2^{\pm p}$$

1 bit 8/11 bits 23/52 bits



p+Bias

d

Fraction

p+127
P+1023

E: data range 数据范围
F: data precision 数据精度

Examples: single precision

$$26.0D = 11010.0B = +1.10100 \times 2^{\textcircled{4}} + 127 = 131 = 10000011B$$

0	10000011	10100000000000000000000
---	----------	-------------------------

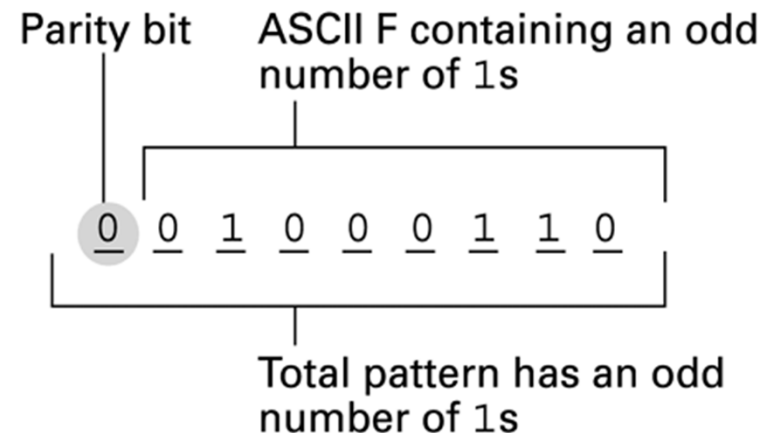
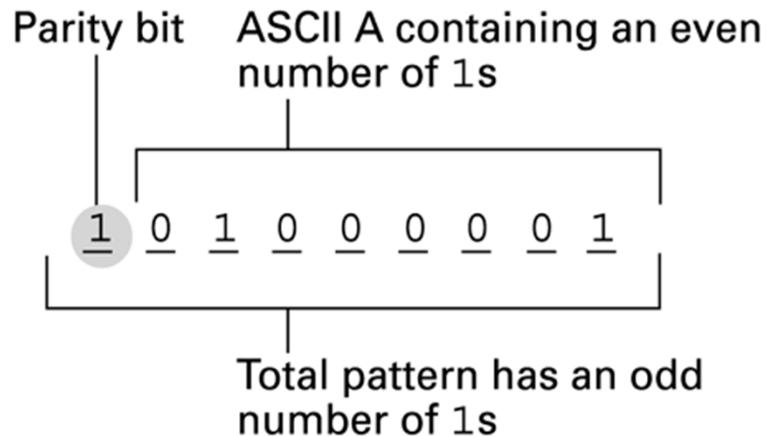
$$-2.5D = -10.1B = -1.01 \times 2^{\textcircled{1}} + 127 = 128 = 10000000B$$

1	10000000	01000000000000000000000
---	----------	-------------------------

1.9 Communication Errors pp.63

- During transmission, error could happen
 - For example, bit 0 \rightarrow 1 or bit 1 \rightarrow 0
- Parity bits (even versus odd) 奇偶校验位
- Checkbytes 校验字节
- Error correcting codes 纠错码

The ASCII codes for the letters A and F adjusted for **odd parity** (奇校验)



奇偶校验的缺点？

奇偶校验: 可发现奇数个错误, 无法纠正

An error-correcting code (纠错码)

Hamming distance = 3

- 汉明码
- 两个字码中不同位值的数目称为汉明距离 (Hamming distance)
- 任何一个模式要成为另一个模式，至少需要改变三个位

Symbol	Code
A	000000
B	001111
C	010011
D	011100
E	100110
F	101001
G	110101
H	111010

最多几位错时，利用这个编码可检测出错误？

Decoding the pattern 010100

Character	Code	Pattern received	Distance between received pattern and code
A	0 0 0 0 0 0	0 1 0 1 0 0	2
B	0 0 1 1 1 1	0 1 0 1 0 0	4
C	0 1 0 0 1 1	0 1 0 1 0 0	3
D	0 1 1 1 0 0	0 1 0 1 0 0	1
E	1 0 0 1 1 0	0 1 0 1 0 0	3
F	1 0 1 0 0 1	0 1 0 1 0 0	5
G	1 1 0 1 0 1	0 1 0 1 0 0	2
H	1 1 1 0 1 0	0 1 0 1 0 0	4

Hamming

Smallest distance

发现最多2位错，纠正1位

Exercises

- Using the error correction code table to decode the following message

001111 100100 001100 010001
000000 001011 011010 110110
100000 011100

Character	Code
A	0 0 0 0 0 0
B	0 0 1 1 1 1
C	0 1 0 0 1 1
D	0 1 1 1 0 0
E	1 0 0 1 1 0
F	1 0 1 0 0 1
G	1 1 0 1 0 1
H	1 1 1 0 1 0

- The following bytes are encoded using odd parity.

Which of them definitely has an error

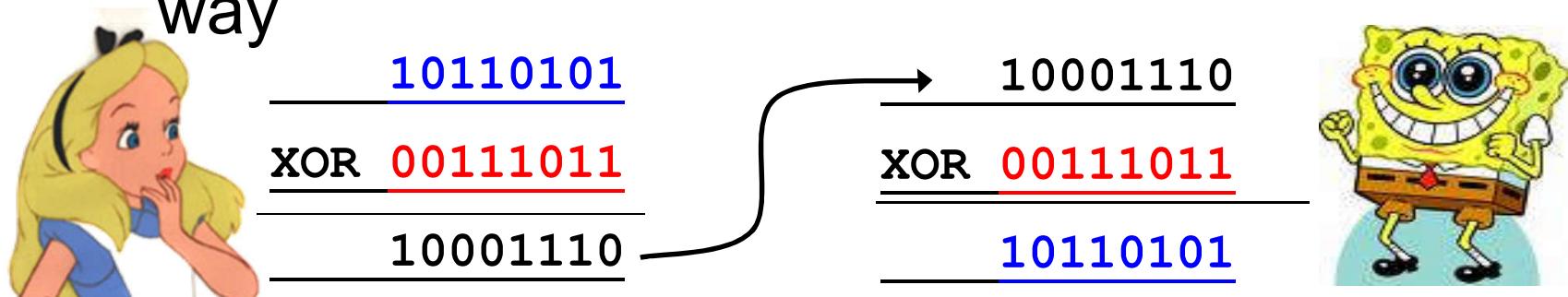
(a) 100101101 (b) 100000001 (c) 101100000 (d)
101111111

How to Share a Secrete?



Data Encryption 数据加密

- Suppose Alice wants to send a secret message, **10110101**, to Bob
 - If they both know a key, **00111011**, that no one else knows
 - Alice can send the encrypted message to Bob using XOR, and Bob can decrypt it the same way

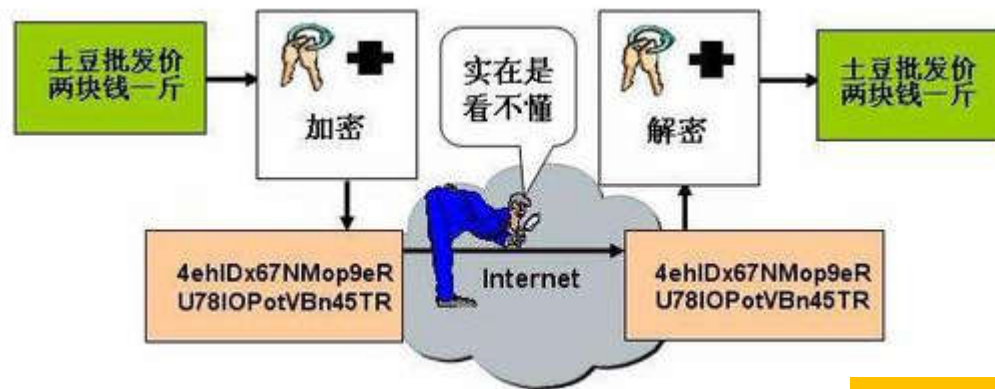


Secret Key Encryption (密钥加密)

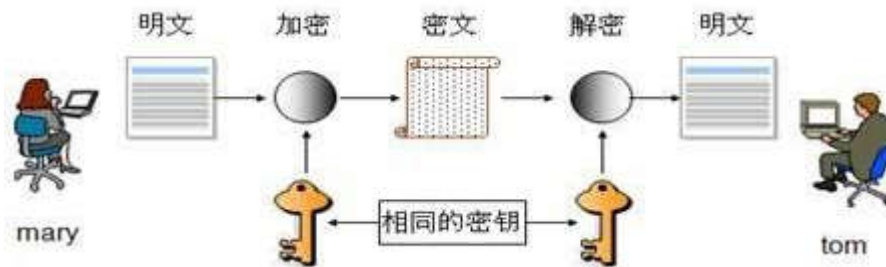
- This is called the *secret key encryption* (密钥加密)
- If no one else knows the secret key and the key is generated randomly and used only once, this is a very good encryption algorithm
- Problems:
 - The key can be used only once
 - Alice and Bob both need to know the key
 - Can be solved by **public-key encryption** (公钥加密)



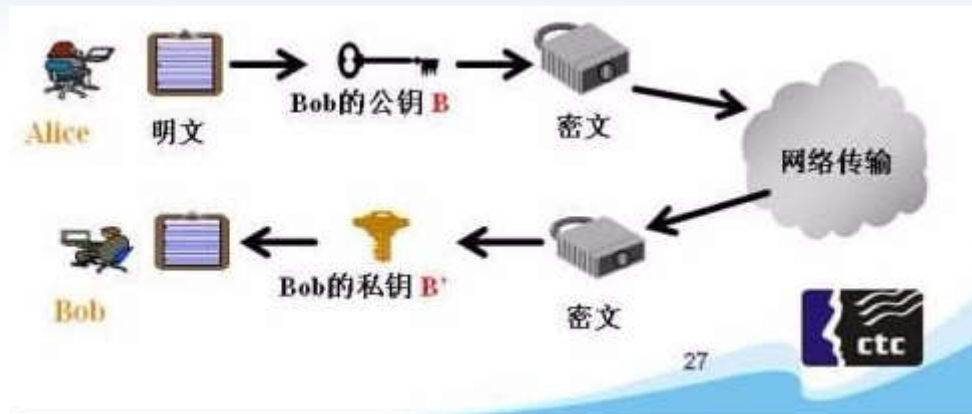
公钥和私钥是通过一种算法得到的一个密钥对(即一个公钥和一个私钥)，将其中的一个向外界公开，称为公钥；另一个自己保留，称为私钥



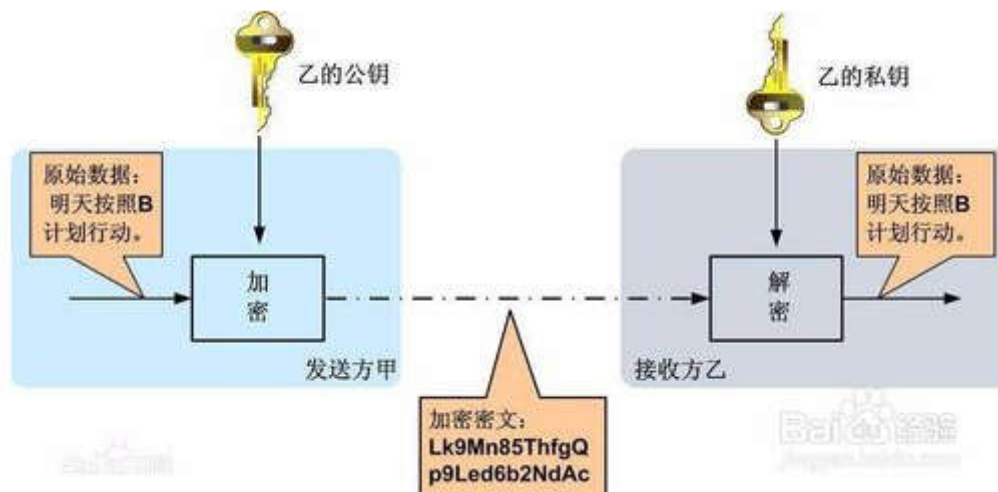
密钥加密（对称加密）



- public-key encryption (公匙加密) scheme
 - If U1 wants to share data with U2, U1 encrypts data using public key of U2 since only U2 knows the private key to decryption. Thus, information is transferred securely because the private key is unnecessary to be transmitted



公钥加密（非对称加密）



银行应用：你掌握着私钥，银行掌握着公钥