

# 第三章 数据存储

计算机中5种不同的数据类型

不同的数据是如何存储的

整数的存储方式

无符号数

有符号数：原码、反码、补码

浮点数的存储格式：

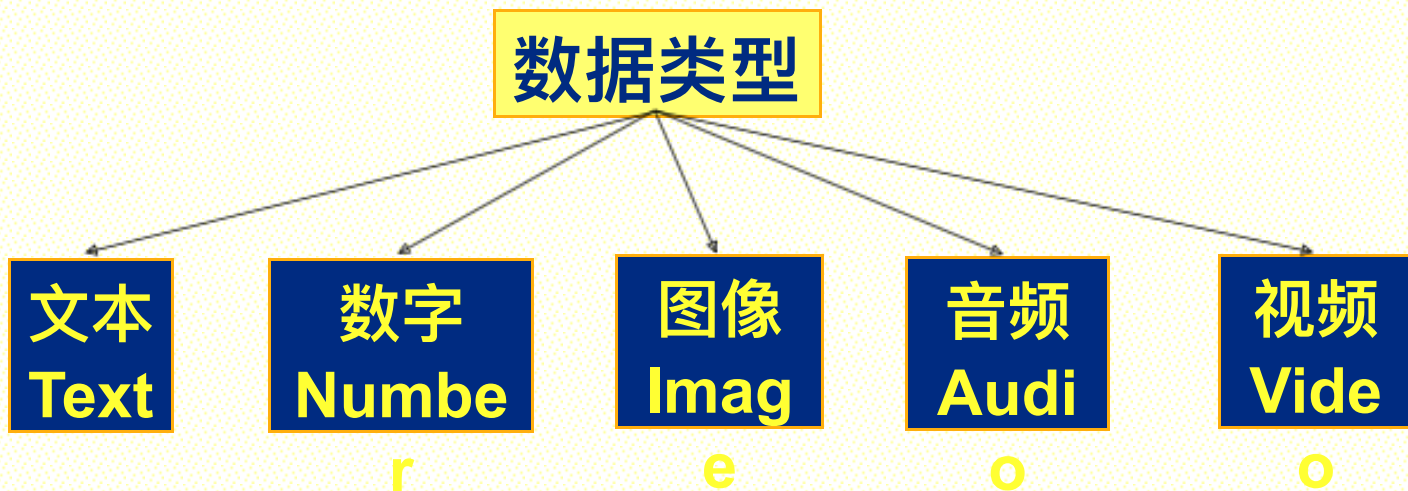
IEEE-754

单精度数

双精度数

- 文本的存储方式
- 音频数据
  - 采样、量化、编码
- 图像数据
  - 位图
  - 矢量图

## 3.1 数据类型



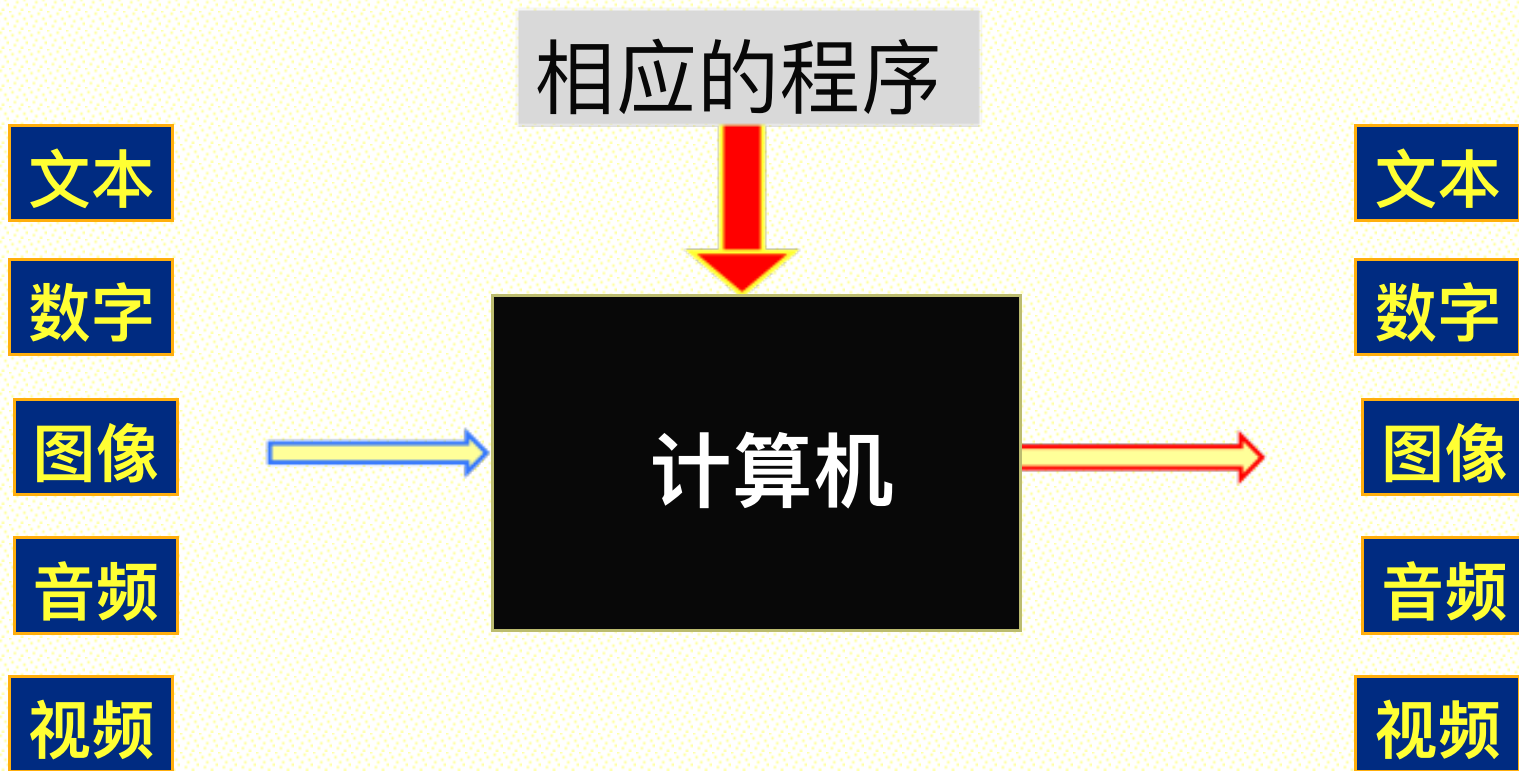
- 数据分类

- 每类数据的特点所决定，编码、存储和处理方式不同

- 多媒体

- 包含上述两种类型及以上的数据

# 冯.诺依曼模型的数据处理



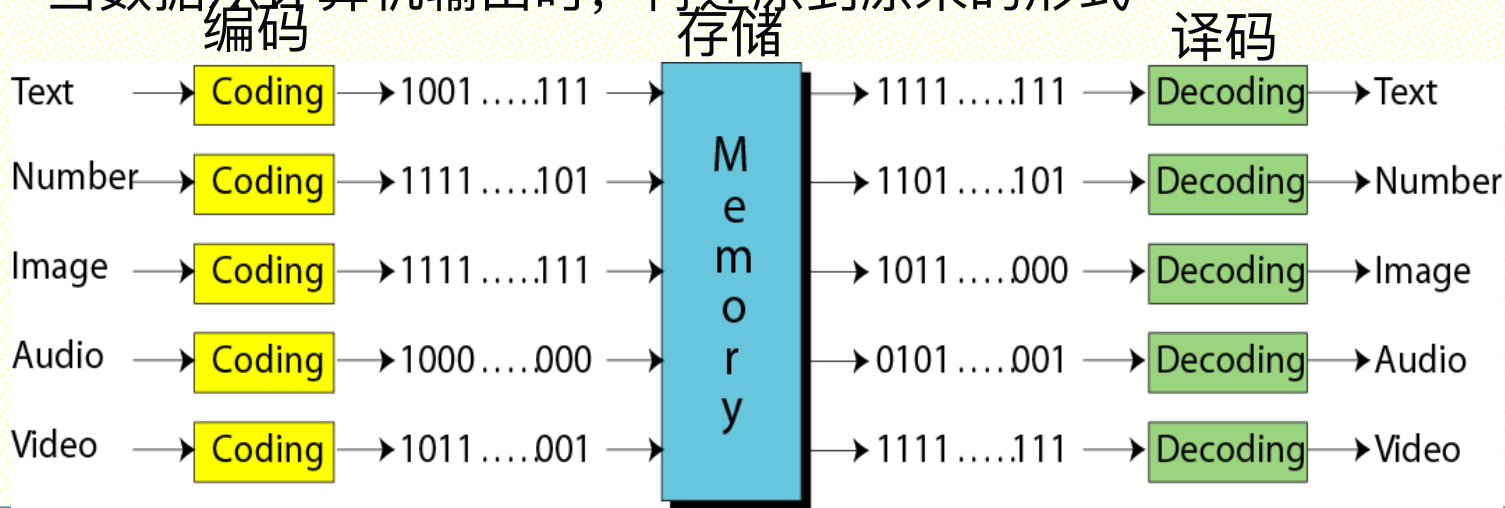
# 计算机内部的数据

- **位**, Bit (比特), 计算机中的最小单位, 0或1

- **位模式**: 0和1的组合

- 所有计算机外部的数据类型的数据都采用统一的数据表示法, 转换后存入计算机中。

- 当数据从计算机输出时, 再还原到原来的形式



# 位模式宽度与符号集

## 位模式宽度决定符号集的个数

2 bits :00, 01,10,11

3 bits: 000, 001,010,011  
, 100, 101,110,111

## 记住两个公式

**N:** 符号集数量

**n:** 二进制位数

$$N=2^n$$

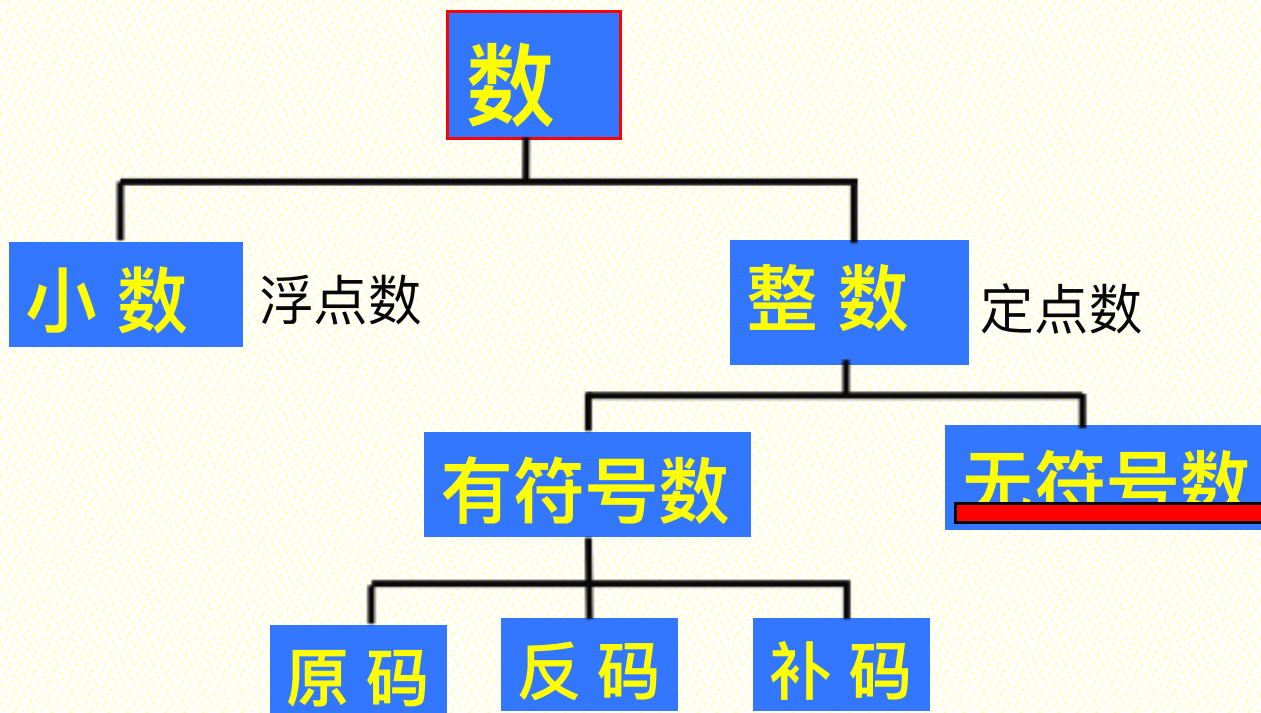
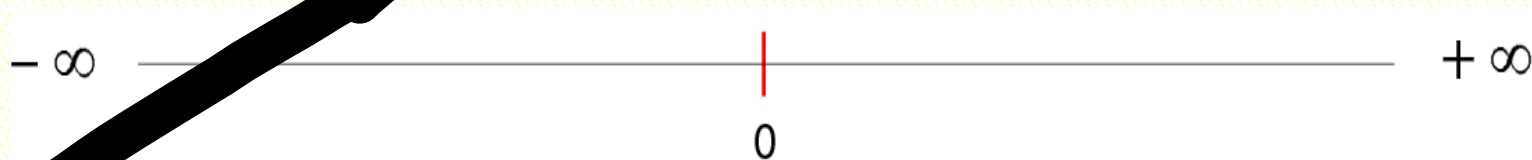
$$n= \log_2 N_{(\text{向上取整})}$$

计算机的字长 (word):

取决于计算机的类型,

8位, 16位, 32位, 64位

## 3.2 存储数字



## 存储整数（无符号数）

位模式宽度	符号个数	数的范围
1	2	0~1
2	4	0
3	8	0~3
4	16	0~15
.....		
7	128	0~127
8	256	0~255
.....		
16	65536	0~65536
$n$	$N=2^n$	$0 \sim 2^n - 1$

只有：  
0和正整数

字长为  $n$

# 无符号数

## 存储无符号整数：

1. 将十进制数转换为二进制数
2. 如果二进制位数

不足字长 $n$ 位，在左边补0，达到 $n$ 位

大于 $n$ 位，**溢出**，无法表示



# 存储无符号数

例1：将 7 以无符号数的形式存储在8位存储单元中

解：

$$1: 7 = 111B$$

$$2: 7 = 0000\ 0111B$$

若存储在16位存储单元呢？

# 存储无符号数

例2: 将 258 存储在 8 位、16 位存储单元中

解:

1:  $258 = 1\ 0000\ 0010\ B$

2: 存储在 8 位存储单元: 只能存储 **右** 8 位  
结果: **溢出** (overflow)

3: 存储在 16 位存储单元:

$$258 = \underline{0000}\ \underline{0001}\ 0000\ 0010B$$

# 无符号数的范围

十进制数	8位存储器	16位存储器
-----	-----	-----
7	00000111	0000000000000111
234	11101010	0000000011101010
258	overflow	0000000100000010
24,760	overflow	0110000010111000
1,245,678	overflow	overflow

$0 \sim 2^n$

# 无符号数的真值

机器数 **00101011** 以无符号数的形式存储在计算机中，求其真值。

**解：**

**方法：** 直接将其转化为10进制即可

$$0010\ 1011\text{B} = 43$$

**无符号数的用途**

- 1.
- 2.
- 3.

计数器  
地址  
其它数

# 数



数

小数

浮点数

整数

定点数

有符号数

无符号数

原码

反码

补码

# 存储有符号数

需要一位做为符号位，符号位在最高位

0 表示正数，1 表示负数

注意：表示数的个数未变，但其范围发生了变化

$$-(2^{N-1}-1) \sim +(2^{N-1}-1)$$

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

-0

# 存储有符号数：原码

## 表示方法：

1. 不考虑符号，将数转换为二进制
2. 数的位数小于  $N-1$  时，在左边补0，至  $N-1$  位
3. 如果原数为正，则最高为补 0

如果原数为负，则最高为补 1

问题：

+0   ☐   0000 0000

-0   ☐   1000 0000

# 存储有符号数：原码

例1：将 7 以原码形式存储在 8 位存储单元中

解：

$$1: 7 = 111 \text{ B}$$

$$2: 7 = \underline{000} 0111 \text{ B}$$

$$3: 7 = \underline{0} 000 0111 \text{ 原}$$



# 存储有符号数：原码

例1：将 7，-7 以原码形式存储在 16 位存储单元中

解：

1:  $7 = 111 \text{ B}$

2:  $7 = \underline{000\ 0000\ 0000\ 0}111 \text{ B}$

3:

$7 = \underline{0\ 000\ 0000\ 0000\ 0}111 \text{ 原}$

$-7 = \underline{1\ 000\ 0000\ 0000\ 0}111 \text{ 原}$

# 存储有符号数：原码

例1：将 258 和  $-258$  以原码形式存储在 16 位存储单元中

解：

$$1: 258 = 1\ 0000\ 0010\ \text{B}$$

$$2: 258 = \underline{000}\ \underline{000}1\ 0000\ 0010\ \text{B}$$

$$3: 258 = \underline{0}\underline{000}\ \underline{000}1\ 0000\ 0010\ \text{原}$$

$$-258 = \underline{1}\underline{000}\ \underline{000}1\ 0000\ 0010\ \text{原}$$

# 无符号数（原码）的真值

方法：

1. 忽略最高位的符号位
2. 将剩余的N-1位转化为十进制数
3. 若最高位为：

0，该数为正

1，该数为负

## 无符号数（原码）的真值

机器数 10111011 以原码的形式存储在计算机中，求其真值。

解：

方法：011 1011B=59

最高位为1，该数为负

10111011原 = -59

机器数 10111011 以无符号数的形式存储在计算机中，求其真值？

# 原码的问题及用途

问题：

存在两个0，在计算时容易混乱

用途：

1. 存储部分实数
2. 对模拟信号采样

# 实 练

1. 将下列数 (+265、-132、63、-31) 以原码的形式存储在16位的存储器中

2. 以下原码形式存储的机器数，求其真值

(1) 1010 1011

(2) 1000 0100 1101 0010

(3) 1111 1111 0111 1101

# 存储有符号数：反码

## 表示方法：

1. 不考虑符号，将数转换为二进制
2. 数的位数小于 N 时，在左边补 0，至 N 位
3. 如果原数为正，则保持不变

如果原数为负，则按位取反

问题：

+0   ☐   0000 0000

-0   ☐   1111 1111

# 存储有符号数：反码

例1：将 7，-7 以反码形式存储在 8 位存储单元中

解：

$$1: 7 = 111 \text{ B}$$

$$2: 7 = \underline{0000} 0111 \text{ B}$$

$$7 = 0000 0111 \text{ 反}$$

$$-7 = 1111 1000 \text{ 反}$$



# 存储有符号数：反码

例1：将 7， - 7以反码形式存储在16位存储单元中

**解：**

1:  $7 = 111 \text{ B}$

2:  $7 = \underline{000\ 0000\ 0000\ 0}111 \text{ B}$

3:

$$7 = \underline{0\ 000\ 0000\ 0000\ 0}111 \text{ 反}$$

$$-7 = \underline{1\ 111\ 1111\ 1111\ 1}000 \text{ 反}$$

## 存储有符号数：反码

例1：将 258 和 -258 以反码形式存储在 16 位存储单元中

解：

$$1: 258 = 1\ 0000\ 0010\ \text{B}$$

$$2: 258 = \underline{0000}\ \underline{000}1\ 0000\ 0010\ \text{B}$$

$$3: 258 = \underline{0\ 000\ 000}1\ 0000\ 0010\ \text{反}$$

$$-258 = \underline{1\ 111\ 111}0\ 1111\ 1101\ \text{反}$$

# 无符号数（反码）的真值

方法：

1. 若最高位为0，直接将其十进制数
2. 若最高位为 1

将该数按位取反后转化为十进制数

该数为负

# 反码的真值

例1：机器数 10111011 以原码的形式存储在计算机中，求其真值。

解：

方法：最高位为1，该数为负

$$1011\ 1011 \rightarrow 0100\ 0100B=68$$

$$10111011反=-68$$

例2：机器数 1111 1111 1111 0110 以反码的形式存储在，求其真值？

# 反码的问题及用途

问题：

存在两个0，在计算时容易混乱

用途：

数据通信中用于校验和检测错误

特点：

反码的反码即为原数

# 存储有符号数：补码

## 表示方法：

1. 不考虑符号，将数转换为二进制
2. 数的位数小于  $N$  时，在左边补0，至  $N$  位
3. 如果原数为正，则保持不变

如果原数为负，

方法一：按位取反 +1 （反码+1）

方法二：从左到右，按位取反 最后一个1及后面的0都不变

# 补码

例1：将 7， -7以补码形式存储在8位存储单元中

解：

$$1: 7 = 111 \text{ B}$$

$$2: 7 = \underline{0000\ 0}111 \text{ B}$$

$$7 = 0000\ 0111 \text{ 补}$$

$$\text{方法一: } 0000\ 0111$$

$$\text{方法二: } 0000\ 0111$$

$$\text{按位取反: } 1111\ 1000$$

$$1111\ 1001$$

$$+1 \quad \underline{1111\ 1001}$$

$$-7 = 1111\ 1001 \text{ 补}$$

# 补码

例2：将 7， - 7以补码形式存储在16位存储单元中

解：

1:  $7 = 111 \text{ B}$

2:  $7 = \underline{000\ 0000\ 0000\ 0}111 \text{ B}$

3:

$$7 = \underline{0000\ 0000\ 0000\ 0}111 \text{ 补}$$

$$-7 = \underline{1111\ 1111\ 1111\ 1}001 \text{ 补}$$



# 存储有符号数：反码

例1：将 258 和 -258 以反码形式存储在 16 位存储单元中

解：

$$1: 258 = 1\ 0000\ 0010\ B$$

$$2: 258 = \underline{0000}\ \underline{000}1\ 0000\ 0010\ B$$

$$3: 258 = 0000\ 0001\ 0000\ 00\ 10\text{补}$$

$$-258 = 1111\ 1110\ 1111\ 11\ 10\text{补}$$

# 补码的真值

方法：

1. 若最高位为0，直接将其十进制数
2. 若最高位为 1

求该数的补码，忽略符号位，将其转换为十进制数 *即忽略符号位求补码*

该数为**负**

# 补码的真值

例1：机器数 10111011 以补码的形式存储在计算机中，求其真值。

解：

方法：最高位为1，该数为负

$$1100\ 0101 - \rangle 0100\ 0101B=69$$

$$10111011\text{补} = -69$$

例2：机器数 1111 1111 1111 0110 以反码的形式存储在，求其真值？

# 补码的用途

补码中只有一个0

0000 0000

用途：用于存储整数和算术运算

补码的补码，即为原数

特殊：

最高位为1，其余位为0时，表示负的最小值

例：1000 0000 表示 -128

1000 表示 -8

# 整数的表示方式比较

## 机器数

0000  
0001  
0010  
0011  
0100  
0101  
0110  
0111  
1000  
1001  
1010  
1011  
1100  
1101  
1110  
1111

## 无符号

0  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15

## 原码

+0  
+1  
+2  
+3  
+4  
+5  
+6  
+7  
-0  
-1  
-2  
-3  
-4  
-5  
-6  
-7

## 反码

+0  
+1  
+2  
+3  
+4  
+5  
+6  
+7  
-7  
-6  
-5  
-4  
-3  
-2  
-1  
-0

## 补码

+0  
+1  
+2  
+3  
+4  
+5  
+6  
+7  
-8  
-7  
-6  
-5  
-4  
-3  
-2  
-1

# 总结1：真值的机器数表示

## 1、正数：

- 将正数转化为二进制，在左边补0至N位即可。
- 正数的原码、反码、补码相同。

## 2、负数：(注意:其最高位必须为1)

将其绝对值转化为二进制，在左边补0至N位。

- 原码：将最高位置1即可。
- 反码：将各位按位取反（0变1，1变0,注意符号位）

补码：

从右边起，第一个1左边的各位按位取反。

或：将其反码的值+1，舍弃溢出部分

## 结论2：机器数的真值

### 符号位 0

- 该数为正
- 则不管该数为原码、反码、补码，直接将二进制数转化为10进制数即可。

### 符号位 1

- 该数为负数
- 该数为原码表示：  
将除符号位以外的二进制数转化为10进制数即可。
- 该数为反码表示：  
各位按位取反后把二进制数转化为10进制数即可。
- 该数为补码表示：  
从右边起，第1个1左边的各位取反后把二进制数转化为10进制数即可（或求其补码）。

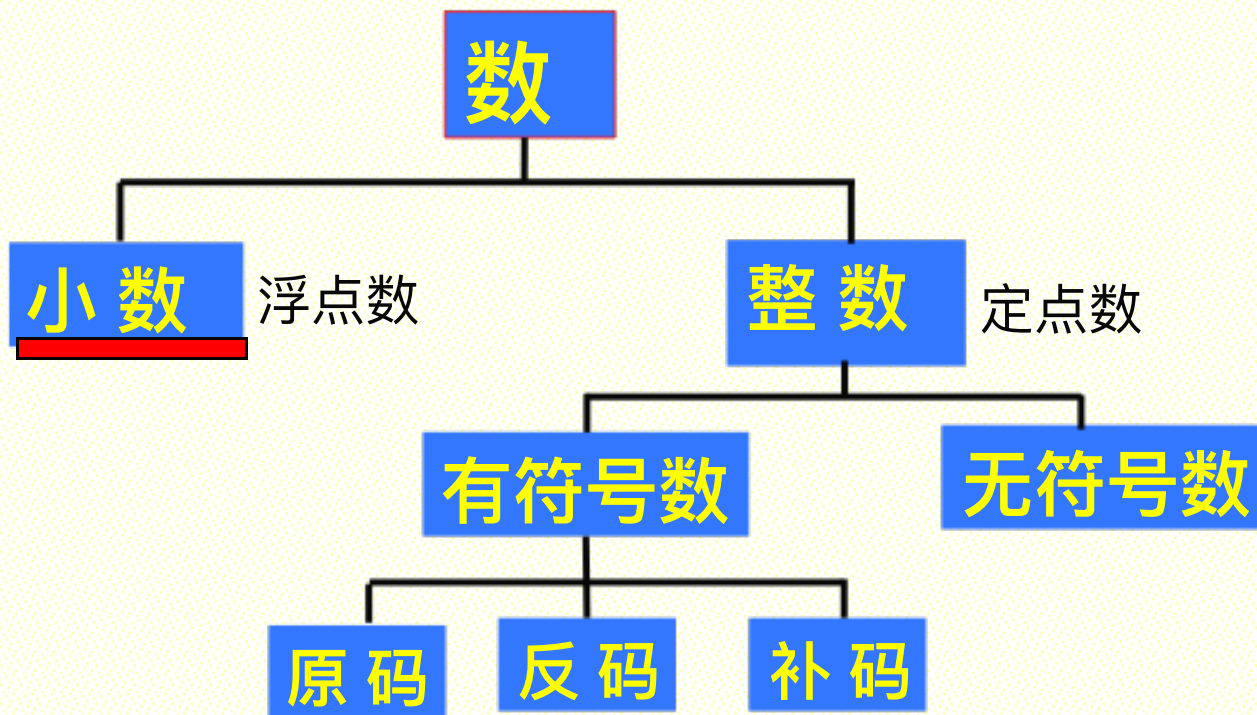
## 练习

计算机的字长为8位， 按样式填写下表

十进制数	原码	反码	补码
-125			
			xFF
		xC7	
	x8F		



## 3.2 存储数字



# 存储浮点数

## 浮点数的表示

- 整数、小数、小数点
- 用科学记数法

## 浮点数组成

- **S**ign: 数的符号
- **E**xponent: 指数
- **M**antissa: 尾数

## 存储浮点数格式:

- IEEE 754标准, 消除表达的不一致性

- IEEE

Institute of Electrical  
and Electronics  
Engineers

电气电子工程师协会

- 单精度: 32 bits

- 双精度: 64bits

# IEEE 754 浮点数标准格式

$$X = (-1)^S \times (1 + \text{尾数}) \times 2^{(\text{真实的指数} + \text{偏移})}$$

S	阶码	尾数
---	----	----

S: 符号位 (0 □ 非负数, 1 □ 负数)

有效位的规格化:  $1.0 \leq |\text{有效位}| < 2.0$

数前有一个隐含前导位1, 有效位是 “1. 尾数”

阶码 (指数): 移码表示

阶码以无符号数的形式存储

单精度: 偏移 = 127

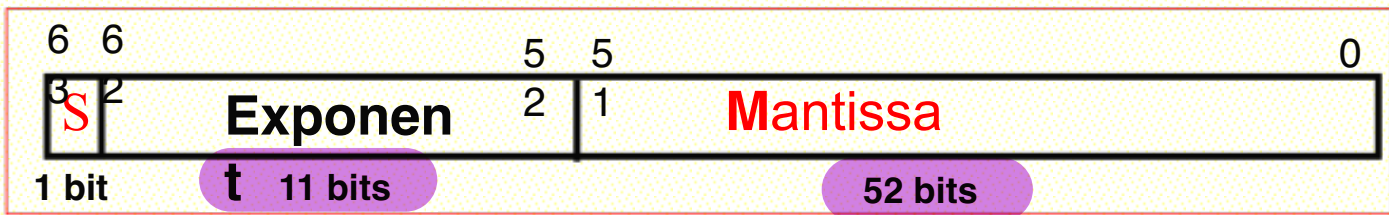
# 浮点数表示

规格化科学计数法:  $\pm 1.xxxx \times 2^{yyyy}$

单精度



双精度



有效数: IEEE 754隐藏了规格化的前导位

24 (单精度)  
53 (双精度)

# 浮点数的存储

1. 把十进制浮点数转换为二进制数:

2. 规范化

3. 符号位S: 正数为0, 负数为1

4. 指数E: 用移码表示

单精度: 真实指数 + 127

双精度: 真实指数 + 1023

5. 尾数M: 小数点后面的数,

单精度: 补0至23位

双精度: 补0至52位

# 实例 1

把 13.375 表示为IEEE单精度数形式

1、转换  $13.375 D = 1101.011B$

2、规范化:  $1101.011B = 2^3 \times 1.\underline{101011}$

3、S: 0

4、E:  $127+3=130 = 1000\ 0010B$

5、M: 101 011 0 0000 0000 0000 0000

6、

S	1000 0010	101 0110 0000 0000 0000 0000
---	-----------	------------------------------

结果:

十六进制不为正负

0100 0001 0101 0110 0000 0000 0000 0000 = **41560000H**

## 实例 2

把  $-161.875$  表示为IEEE单精度数形式

1、转换  $-161.875 \text{ D} = -1010\ 0001.111\text{B}$

2、规范化:  $1010\ 0001.111\text{B}$

3、S: 1  $= 2^7 \times 1.010\ 0001\ 111\text{B}$

4、E:  $127+7=134 = 1000\ 0110\ \text{B}$

5、M: 010 0001 111 0 0000 0000 0000

6、结果 1100 0011 0010 0001 1110 0000 0000 0000  
= C 3 2 1 E 0 0 0 H

## 双精度

$$1101.011\text{B} = 2^3 \times 1.\underline{101\ 011}$$

S: 0

E:  $1023+3=1026 = 100\ 0000\ 0010\ \text{B}$

M: 1010 1100 0000 0000 .....0000 0000

结果

0100 0000 0010 1010 1100 0000 0000. ... 0000

= 402AC00000000000H



# 浮点数的真值

1. 把浮点数分解为S、E、M
- 2. 根据符号位S的值确定数的正负
- 3. 指数转换为十进制，并减去移码
- 4. 尾数，小数点前面加1
- 5. 将规范化的数转化为非规范化
- 6. 把二进制浮点数的整数和小数分别转换为十进制数

## 实例

求下面 32-bit 浮点数的真值

1011 1110 0110 0110 0000 0000 0000 0000

解: 1 011 1110 0 110 0110 0000 0000 0000 0000

S: -

E:  $124 - 127 = -3$

M: 1.110011

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

结果:  $-0.001110011\text{B}$   
 $-(1 \times 2^{-3} + 1 \times 2^{-4} + 1 \times 2^{-5} + 1 \times 2^{-8} + 1 \times 2^{-9})$

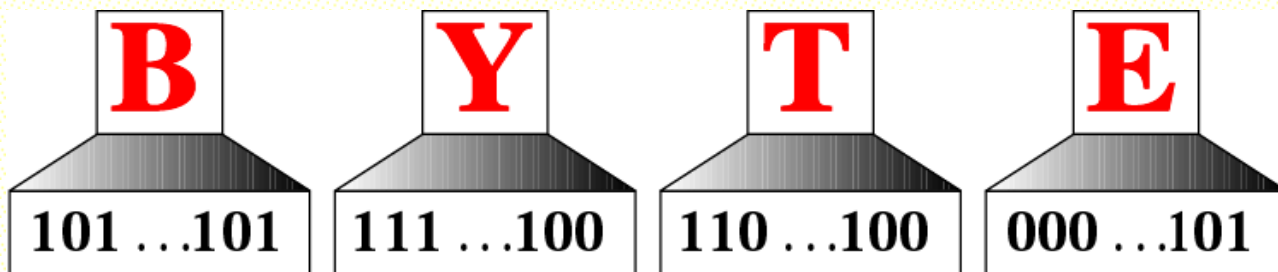
# 浮点数编码

单精度		双精度		表示的数
指数	尾数	指数	尾数	
0	000	0	000	0
0	非零	0	非零	$\pm$ 非规格化数
1~254	任意	1~2046	任意	$\pm$ 浮点数
255	0	2047	0	$\pm\infty$
255	非零	2047	非零	NaN (非数)

NaN 用于处理计算中出现的错误情况，比如 0.0 除以 0.0 或者求负数的平方根。

## 3.3 存储文本

用位模式来表示任何一个符号



编码





**N:** 符号集数量

**n:** 二进制位数

$$N=2^n$$

$$n = \log_2 N \text{ (向上取整)}$$

# ASCII 7 bits

高 3 位 \ 低 4 位	000	001	010	011	100	101	110	111
0000	NUL	DC0	 SP	 0	@	P	-	p
0001	SOH	DC1	!	1	 A	Q	 a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	\$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	,	7	G	W	g	w
1000	BS	CAN	)	8	H	X	h	x
1001	HT	EM	(	9	I	Y	I	y
1010	LF	SUB	*	:	J	Z	J	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	↑	n	~
1111	SI	US	/	?	O	←	o	DEL

# 其它符号系统

- 扩展ASCII

- 在最高位加0，变成8位

- EBCDIC

- Extended Binary Coded Decimal Interchange Code

- IBM, 8-bit

- Unicode 16-bit

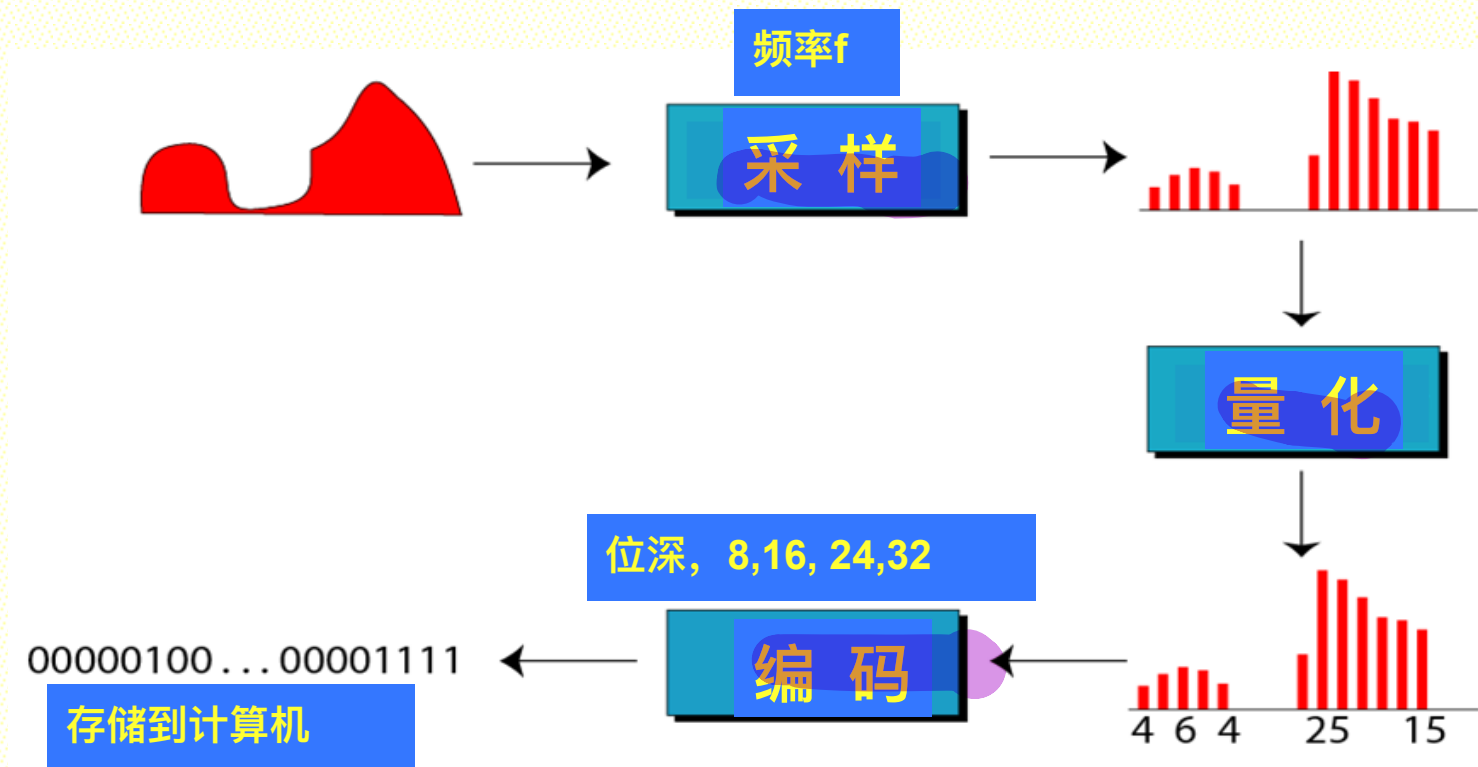
- JAVA

- ISO: 32-bit.

- International Organization for Standardization

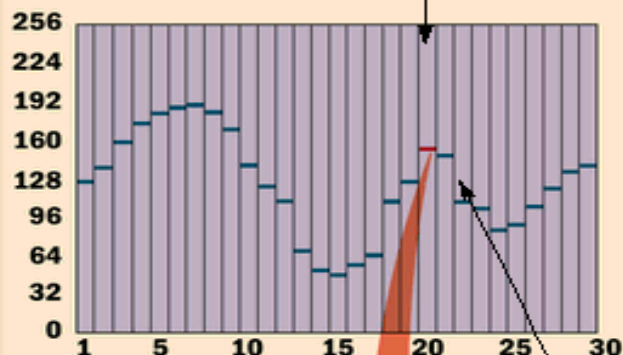
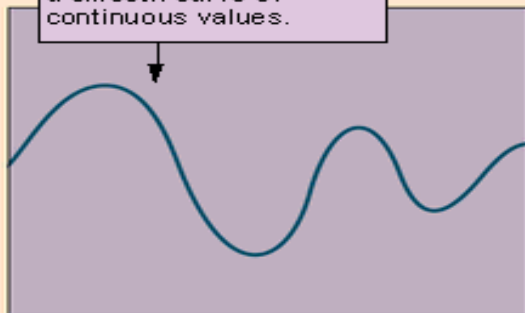
## 3.4 存储音频

模拟的音频信号，离散的数字信号。



# 音频

An analog sound wave is a smooth curve of continuous values.



Sample	Sample Height (Decimal)	Sample Height (Binary)
1	130	10000010
2	140	10001100
3	162	10100010
4	175	10101111
5	185	10111001
6	188	10111100
7	195	10111111
8	200	11000000
9	190	10111110
10	175	10101111
11	150	10011010
12	115	01110011
13	68	01000100
14	50	00110010
15	45	00101101
16	60	00111100
17	66	01000010
18	115	01110011
19	129	10000001
20	155	10011011
21	153	10011001

The height of each sample is converted into a binary number and stored. The height of sample 20 is 155 (decimal), so it is stored as its binary equivalent, 10011011.



# 音频文件

文件大小决定于采样频率和量化位深.

常见的采样频率

8,000 Hz、11,025 Hz-AM、22,050、

44,100 Hz (音频 CD) 48,000 Hz

96,000 或者 192,000 Hz - DVD-Audio、BD-ROM

量化位深: 8,16,24,32

位深 =  $\log_2$  每个样本有多少级不同  
表示

声道数

1: 单声道

2: 立体声

其它: 2.1, 5.1, 7.1

# 音频文件

计算方法.

$$\frac{\text{采样频率} \times \text{采样位深} \times \text{声道数} \times \text{时间}}{8 \times 1024 \times 1024} (\text{M})$$

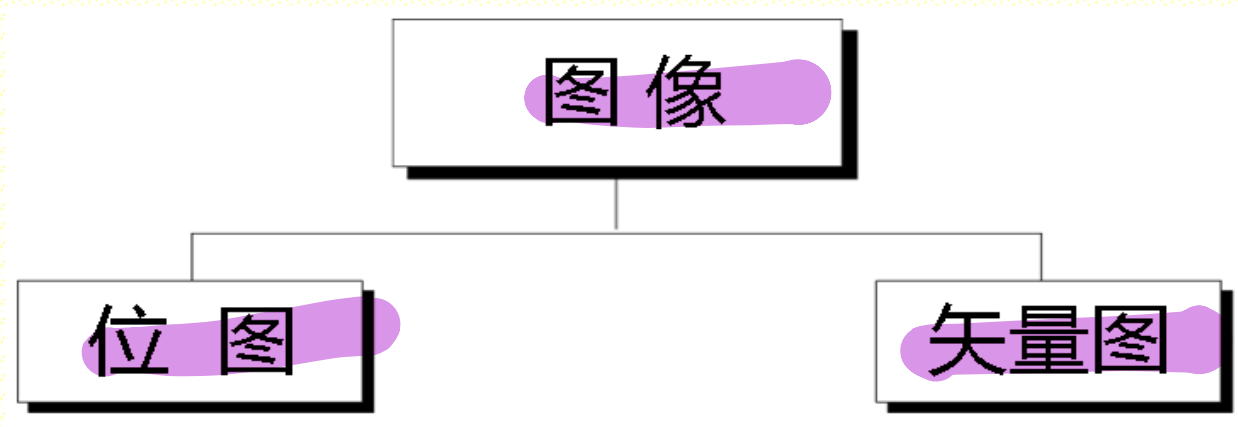
• 例，CD音频，每分钟数据.

$$\frac{44.1 \times 10^3 \times 16 \times 2 \times 60}{8 \times 1024 \times 1024} = 10\text{M}$$

用秒计算

↓      ↓ kHz      ↓ MB  
转换为 Byte

## 3.5 存储图像



存储图像的像素信息

放大易失真

文件大小决定于：

分辨率

颜色位深

适用于比较复杂的图像

存储生成图像的公式

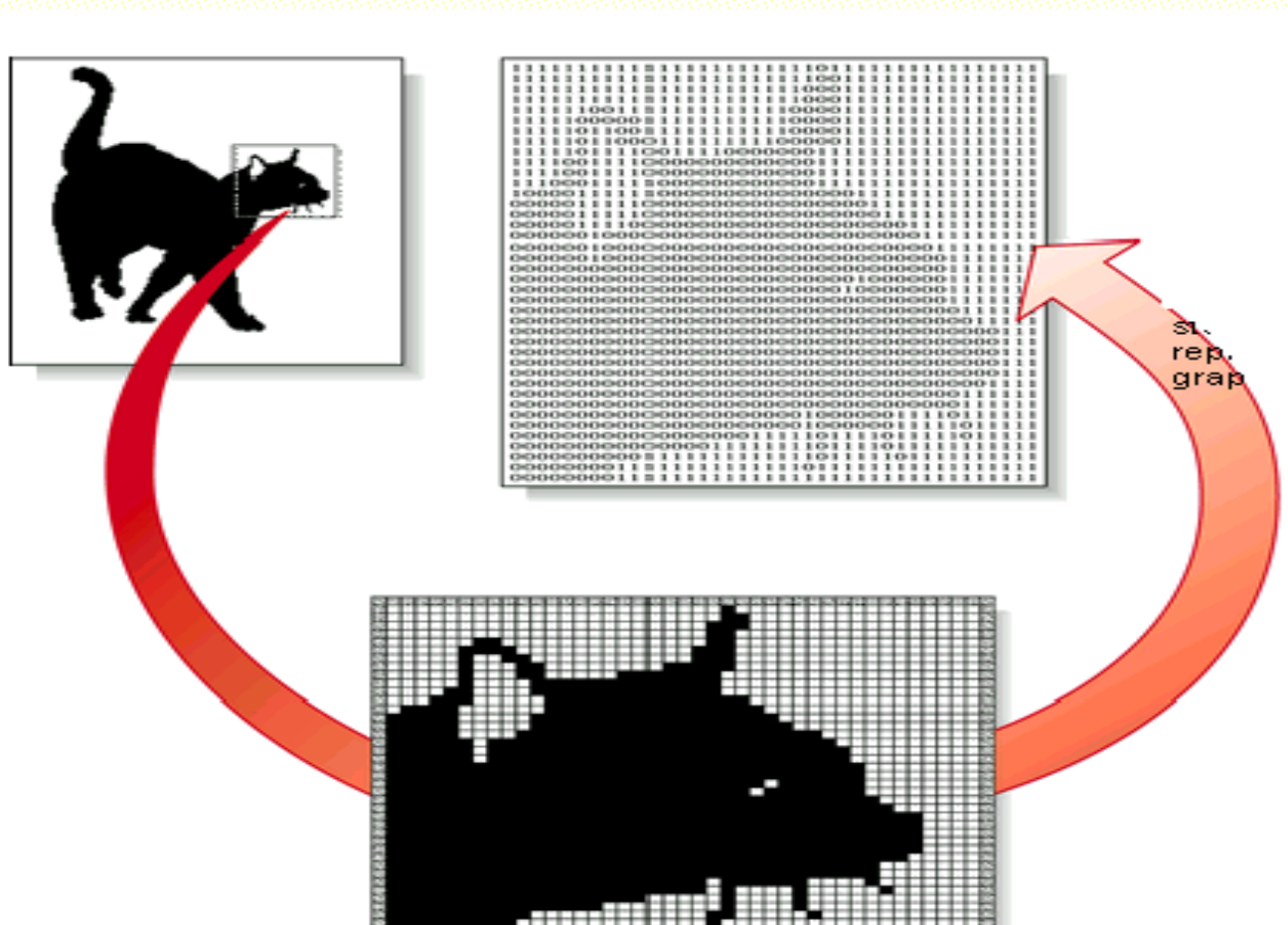
放大不失真

适用于比较规则的图像

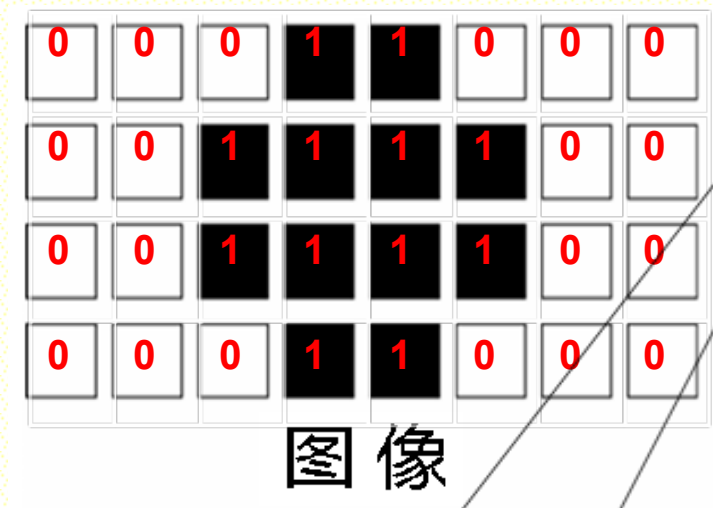
# 位图与矢量图效果



# 位图 (黑白或单色)



# 黑白位图的存储



线性表示

00011000

00011000

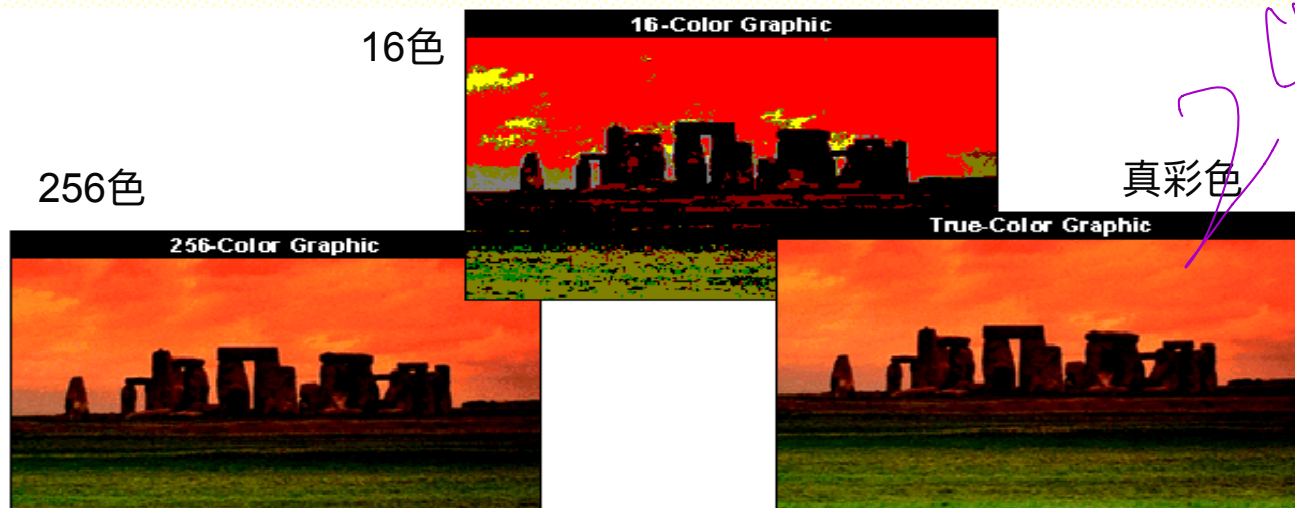
00011000

00011000

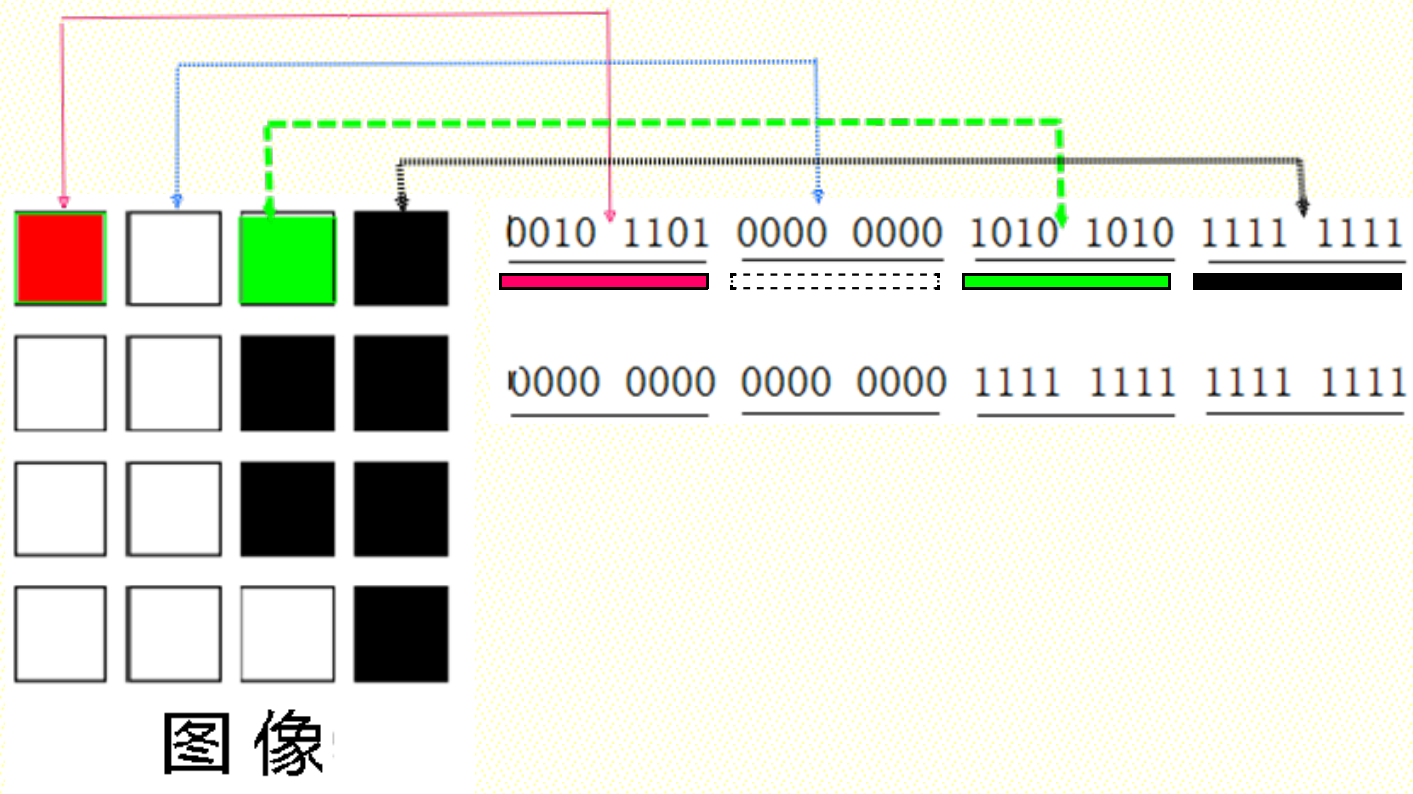
矩阵表示存储

# RGB三基色

		R	G	B
		↓	↓	↓
Red (with 100% intensity)	→	11111111	00000000	00000000
Green (with 100% intensity)	→	00000000	11111111	00000000
Blue (with 100% intensity)	→	00000000	00000000	11111111
White (with 100% intensity)	→	11111111	11111111	11111111

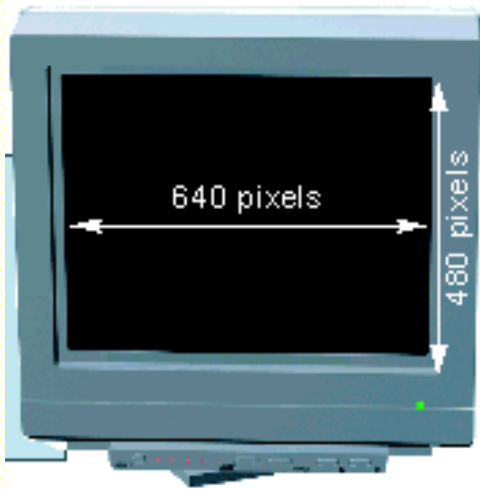


# 256色 位图的存储





# 位图文件的大小



$$640 \times 480 = 307200 \text{ pixel}$$

$$640 \times 480 \times 8 = 307200 \times 8 \text{ bits}$$

$$640 \times 480 \times \frac{8}{8} = 307200 \text{ bytes}$$

$$1024 \times 768 = 786432 \text{ pixel}$$

$$1024 \times 768 \times 8 = 786432 \times 8 \text{ bits}$$

$$1024 \times 768 \times \frac{8}{8} = 786432 \text{ bytes}$$

屏幕分辨率的表示，你知道哪些？它们之间有什么关系？

## 计算方法

$$\frac{\text{长} \times \text{宽} \times \text{颜色所用位数}}{8} = \text{bytes}$$

颜色所用位数，若为256色 须  $2^8$  位来表示  
故乘8

若为65536色 则须16位表示65536

故颜色所用位数为16

结果除8是因为计算结果为bits 是位  
而结果需要用字节表示(byte)

1 byte = 8 bits 故除8

## 3.6 存储视频

• 视频是图像（帧）在时间轴上的表示。

• 其存储方式与图像类似

• 试计算一下，一部标清、高清、超清电视剧的文件大小(假定时长为60分钟)。

• 标清：720X576

• 高清：1280×720

• 超清：1920×1080

计算方法

一张图片大小(计算方法如上) × 时间(s) × 帧率(一秒几张图) × 一个像素多少字节

↓  
结果的单位为字节