

# 计算机组成原理

## (2) 定点数与浮点数的表示方法

王浩宇,教授

haoyuwang@hust.edu.cn

<https://howiepku.github.io/>

# 本节内容

- 定点数的表示方法
- 浮点数的表示方法

# 数据的类型

## ■ 按数据的表示范围分：

- 定点数：小数点位置固定，数据表示范围小
- 浮点数：小数点位置不固定，数据表示范围较大

## ■ 按能否表示负数分：

- 无符号数：所有均为表示数值，直接用二进制数表示
- 有符号数：有正负之分，最高位为符号位，其余表示数值。

## ■ 按数据格式分：

- 真值：没有经过编码的直观数据表示方式，其值可带正负号，任何数制均可
- 机器数：符号化后的数值(包括正负号的表示)，一般位数固定(8、16、32……)，不能随便忽略任何位置上的0或1

# 数据格式——定点数

■ 定点数：小数点固定在某一位置的数据； 设采用 $n+1$ 位数据

■ 定点小数：

▪ 表示形式

$$\boxed{X_0 \bullet X_{-1} X_{-2} X_{-3} \dots X_{-n}}$$

有符号数  $x = x_s x_{-1} x_{-2} \dots x_{-n} \quad 0 \leq |x| \leq 1 - 2^{-n}; \quad x_s \text{ 为符号位}$

无符号数  $x = x_0 x_{-1} x_{-2} \dots x_{-n} \quad 0 \leq x \leq 1 - 2^{-n};$

▪ 数据表示范围  $0.0\dots0 = 0 \leq |x| \leq 1 - 2^{-n} = 0.1\dots1$

■ 定点整数：

▪ 表示形式

$$\boxed{X_n X_{n-1} X_{n-2} \dots X_1 X_0 \bullet}$$

有符号数  $x = x_s x_{n-1} \dots x_1 x_0 \quad |x| \leq 2^n - 1; \quad x_s \text{ 为符号位}$

无符号数  $x = x_n x_{n-1} \dots x_1 x_0 \quad 0 \leq x \leq 2^{n+1} - 1; \quad x_n \text{ 为数值位}$

■ 注意：小数点的位置是机器约定好的，并没有实际的保存。

目前计算机中多采用定点纯整数表示，因此将定点数表示的运算简称为整数运算

# 数的机器码表示

## ■ 重点：

- 1、原码、补码、移码的表示形式
- 2、原码、补码、移码的表示范围

# 原码表示法

$$X_n X_{n-1} X_{n-2} \dots X_1 X_0$$

- **定义：**
- 定点小数：
 
$$[x]_{\text{原}} = \begin{cases} x & 1 > x \geq 0 \\ 1 - x = 1 + |x| & 0 \geq x > -1 \end{cases}$$
  - 定点整数：
 
$$[x]_{\text{原}} = \begin{cases} x & 2^n > x \geq 0 \\ 2^n - x = 2^n + |x| & 0 \geq x > -2^n \end{cases}$$

■ **举例：**

- $[+0.110]_{\text{原}} = 0.110$
- $[-0.110]_{\text{原}} = 1 - (-0.110) = 1.110$
- $[+110]_{\text{原}} = 0110$
- $[-110]_{\text{原}} = 2^3 - (-110) = 1000 + 110 = 1110$

实际机器中保存时  
并不保存小数点

# 原码表示法——特点

## ■ 0有两种表示法

- $[+0]_{\text{原}} = 0000$  ;  $[-0]_{\text{原}} = 1000$

## ■ 数据表示范围

- 定点小数:  $-1 < X < 1$
- 定点整数:  $-2^n < X < 2^n$  (若数值位 $n=3$ 即:  $-8 < X < 8$ )

## ■ 优点

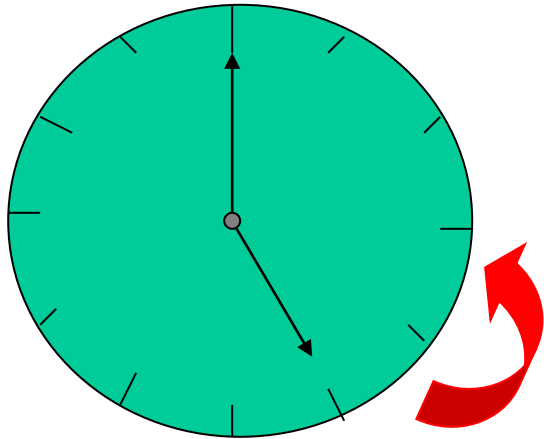
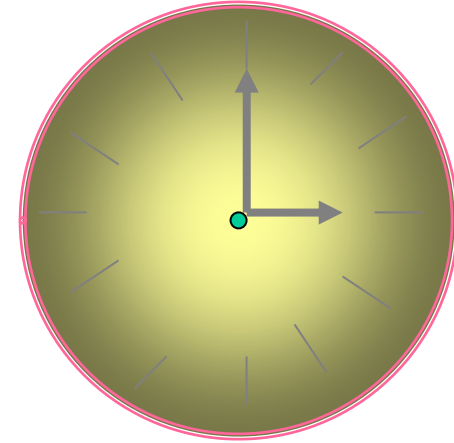
- 与真值对应关系简单;

## ■ 缺点

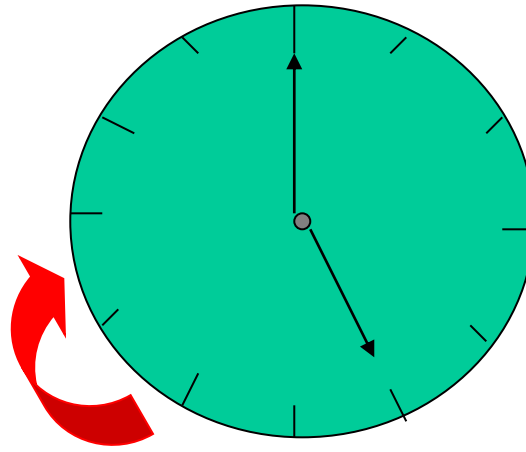
- 参与运算复杂, 需要将数值位与符号位分开考虑。

# 补码表示法的引入 (1/3)

要将指向5点的时钟调整到3点整，应如何处理？



$$5-2=3$$



$$5+10=3$$

(12自动丢失，12就是模)



## 补码表示法的引入 (2/3)

### ■ 继续推导：

- $5-2=5+10 \pmod{12}$
- $5+(-2)=5+10 \pmod{12}$
- $-2=10 \pmod{12}$

### ■ 结论：

在模为12的情况下，-2的补码就是10。一个负数用其补码代替，同样可以得到正确的运算结果。

## 补码表示法的引入 (3/3)

### ■ 进一步结论：

- 在计算机中，机器能表示的**数据位数是固定的**，其运算都是**有模运算**。
  - 若是 $n+1$ 位整数(包含符号位)，则其模为 $2^{n+1}$ ；
  - 若是小数，则其模为2。
- 若运算结果超出了计算机所能表示的数值范围，则只保留它的小于模的低 $n$ 位的数值，超过 $n$ 位的高位部分就自动舍弃了。
- 计算机里面，只有加法器，没有减法器，所有的减法运算，都必须用加法进行。
- 用补数代替原数，可把减法转变为加法。出现的进位就是模，此时的进位，就应该忽略不计。

# 补码表示法

$$X_n X_{n-1} X_{n-2} \dots X_1 X_0$$

## ■ 定义:

$$\begin{aligned}
 \text{■ 定点小数: } [x]_{\text{补}} &= \begin{cases} x & 1 > x \geq 0 \\ 2+x = 2 - |x| & 0 \geq x \geq -1 \end{cases} \quad (\text{mod } 2) \\
 \text{■ 定点整数: } [x]_{\text{补}} &= \begin{cases} x & 2^n > x \geq 0 \\ 2^{n+1}+x = 2^{n+1}-|x| & 0 \geq x \geq -2^n \end{cases} \quad (\text{mod } 2^{n+1})
 \end{aligned}$$

## ■ 举例:

$$\begin{aligned}
 \text{■ } [+0.110]_{\text{补}} &= 0.110 \\
 \text{■ } [-0.110]_{\text{补}} &= 10 + (-0.110) = 1.010 \\
 \text{■ } [+110]_{\text{补}} &= 0110 \\
 \text{■ } [-110]_{\text{补}} &= 2^4 + (-110) = 10000 - 110 = 1010
 \end{aligned}$$

实际机器中保存时并不保存小数点

# 由原码求补码

## ■由原码求补码的简便原则(负数)

- 除符号位以外, 其余各位按位取反, 末位加1;
- 从最低位开始, 遇到的第一个1以前的各位保持不变, 之后各位取反。

例:  $[X]_{\text{原}} = 1\ 1\ 0\ 1\ 1\ 0\ \mathbf{1}\ 0\ 0$

$[X]_{\text{补}} = 1\ 0\ 1\ 0\ 0\ 1\ 1\ 0\ 0$

# 求相反数的补码

- 由  $[X]_{\text{补}}$  求  $[-X]_{\text{补}}$ 
  - 连符号位一起各位求反，末位加1。
- 例：  $[X]_{\text{补}} = 1.1010101$
- 解：

$$\begin{array}{r}
 [X]_{\text{补}} = 1\ 1\ 0\ 1\ 0\ 1\ 0\ 1 \\
 \phantom{[X]_{\text{补}}} \phantom{=} \phantom{=} \phantom{=} \phantom{=} \phantom{=} \phantom{=} \phantom{=} \phantom{=} 0\ 0\ 1\ 0\ 1\ 0\ 1\ 0 \\
 \phantom{[X]_{\text{补}}} \phantom{=} \phantom{=} \phantom{=} \phantom{=} \phantom{=} \phantom{=} \phantom{=} \phantom{=} \phantom{=} \phantom{=} \phantom{=} \phantom{=} 1 \\
 + \\
 \hline
 [-X]_{\text{补}} = 0\ 0\ 1\ 0\ 1\ 0\ 1\ 1
 \end{array}$$

由  $[-X]_{\text{补}}$   
求  $[X]_{\text{补}}$ ，  
此规则同  
样适用。

# 移码表示法

$$X_n X_{n-1} X_{n-2} \dots X_1 X_0$$

## ■ 移码通常用于表示浮点数的阶码

- 用定点整数形式的移码

## ■ 定义：

$$[x]_{\text{移}} = 2^n + x \quad 2^n > x \geq -2^n$$

## ■ 与 $[x]_{\text{补}}$ 的区别：符号位相反

## ■ 优点：

- 可以比较直观地判断两个数据的大小；
  - 浮点数运算时，容易进行对阶操作；
- 表示浮点数阶码时，容易判断是否下溢；
  - 当阶码为全0时，浮点数下溢。

4位补码与移码

真值	补码	移码
-8	1000	0000
-7	1001	0001
-6	1010	0010
.....	.....	.....
0	0000	1000
+1	0001	1001
.....	.....	.....
+7	0111	1111

# 原、补、移码的编码形式

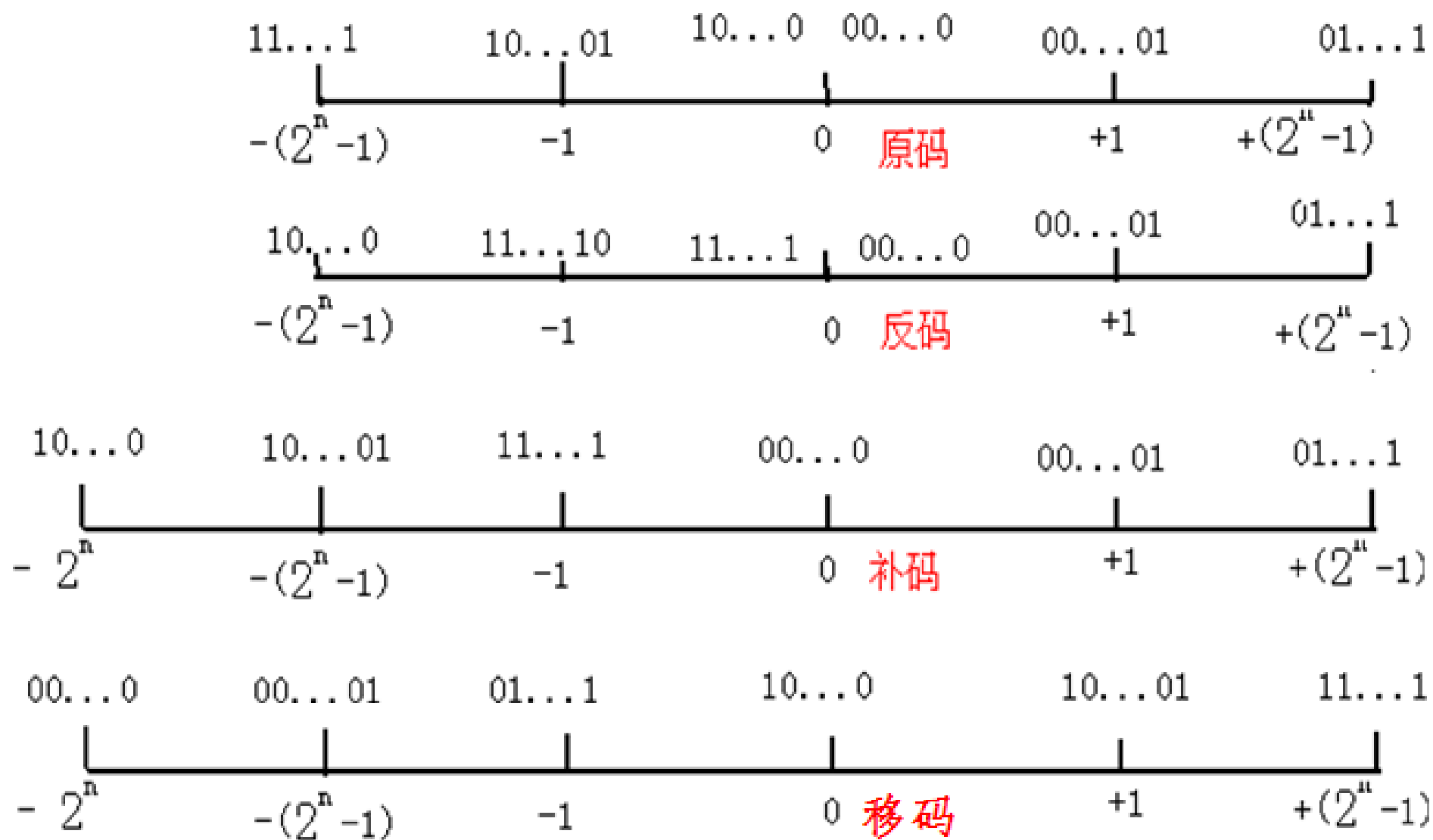
## ■ 正数：

- 原、补码的编码完全相同；
- 补码和移码的符号位相反，数值位相同；

## ■ 负数：

- 原码： 符号位为1  
数值部分与真值的绝对值相同
- 补码： 符号位为1  
数值部分与原码各位相反，且末位加1
- 移码： 符号位与补码相反，数值位与补码相同

以定点整数为例, 用数轴形式说明原码、反码、补码、移码表示范围和可能的数码组合情况





将十进制真值(−127, −1, 0, +1, +127)列表表示成二进制数及原码、反码、补码、移码值

十进制真值	二进制真值	原码表示	反码表示	补码表示	移码表示
-127	-111 1111	1111 1111	1000 0000	1000 0001	0000 0001
-1	-000 0001	1000 0001	1111 1110	1111 1111	0111 1111
0	+000 0000	0000 0000	0000 0000	0000 0000	1000 0000
	-000 0000	1000 0000	1111 1111		
+1	+000 0001	0000 0001	0000 0001	0000 0001	1000 0001
+127	+111 1111	0111 1111	0111 1111	0111 1111	1111 1111

负数时

符号位

+→0; -→1

数值位

各位取反

数值位

末位加1

符号位(正负数)

取反

设机器字长16位，定点表示，尾数15位，数符1位，问：

(1) 定点原码整数表示时，最大正数是多少？最小负数是多少？

(2) 定点原码小数表示时，最大正数是多少？最小负数是多少？

### ■ 定点原码整数

■ 最大正数

0	111 1111 1111 1111
---	--------------------

 $(2^{15}-1) = +32767$

■ 最小负数

1	111 1111 1111 1111
---	--------------------

 $-(2^{15}-1) = -32767$

### ■ 定点原码小数

■ 最大正数

0	111 1111 1111 1111
---	--------------------

 $(1-2^{-15}) = +(1-1/32768)$

■ 最小负数

1	111 1111 1111 1111
---	--------------------

 $-(1-2^{-15}) = -(1-1/32768)$

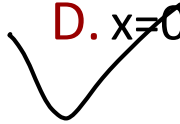
# 练习题

一个C语言程序在一台32位机器上运行。程序中定义了三个变量x，y和z，其中x和z是int型，y为short型。当x=127，y=-9时，执行赋值语句z=x+y后，x、y和z的值分别是：

A. x=0000007FH，y=FFF9H，z=00000076H

B. x=0000007FH，y=FFF9H，z=FFFF0076H

C. x=0000007FH，y=FFF7H，z=FFFF0076H

 D. x=0000007FH，y=FFF7H，z=00000076H

# 本节内容

- 定点数的表示方法
- 浮点数的表示方法

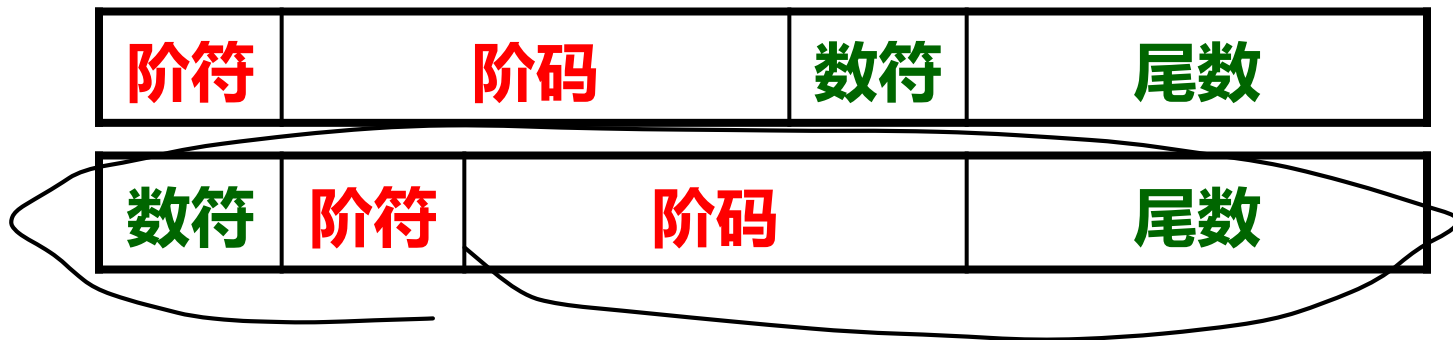
# 数据格式——浮点数

■浮点数：小数点位置可变，形如科学计数法中的数据表示。

■浮点数格式定义： $N = R^e \times M$

- M：尾数(mantissa)，是一个纯小数，表示数据的全部有效数位，决定着数值的精度；
- R：基数(radix)，可以取2、8、10、16，表示当前的数制；
  - 计算机中，一般默认为2，隐含表示。
- e：阶码(exponent)，是一个整数，用于指出小数点在该数中的位置，决定着数据数值的大小

■机器数的一般表示形式



# 科学计数法的表示

- 一个十进制数可以表示成不同的形式：

$$(N)_{10} = 123.456 = 123456 \times 10^{-3} = 0.123456 \times 10^{+3}$$

- 同理，一个二进制数也可以有多种表示：

$$(N)_2 = 1101.0011 = 11010011 \times 2^{-100} = 0.1101.0011 \times 2^{+100}$$

# 浮点数规格化

## ■ 浮点数的表示

- $1.11 \times 2^0 = 0.111 \times 2^1 = 11.1 \times 2^{-1}$
- 机器数的表示不同，不利于运算 ——→ 规格化

## ■ 规格化的目的

- 保证浮点数表示的唯一性
- 保留更多地有效数字，提高运算的精度

## ■ 规格化要求

- $1/R \leq |\text{尾数}| < 1$ ;

## ■ 规格化处理：

- 尾数向左移 $n$ 位 (小数点右移)，同时阶码减 $n$ ； ——→ 左规
- 尾数向右移 $n$ 位 (小数点左移)，同时阶码加 $n$ 。 ——→ 右规

# 浮点数的规格化

## ■ 尾数用原码表示时

- 尾数最高数值位为1；
  - 尾数形如  $0.1 \times \times \dots \times$  (正)；或  $1.1 \times \times \dots \times$  (负)；
  - 例如， $0.011 \times 2^5$  要规格化则变为  $0.11 \times 2^4$ ；  
 $-0.011 \times 2^5$  要规格化则变为  $1.11 \times 2^4$ ；
- 表示范围  $\frac{1}{2} \leq M \leq (1-2^{-n})$   
 $-(1-2^{-n}) \leq M \leq -\frac{1}{2}$

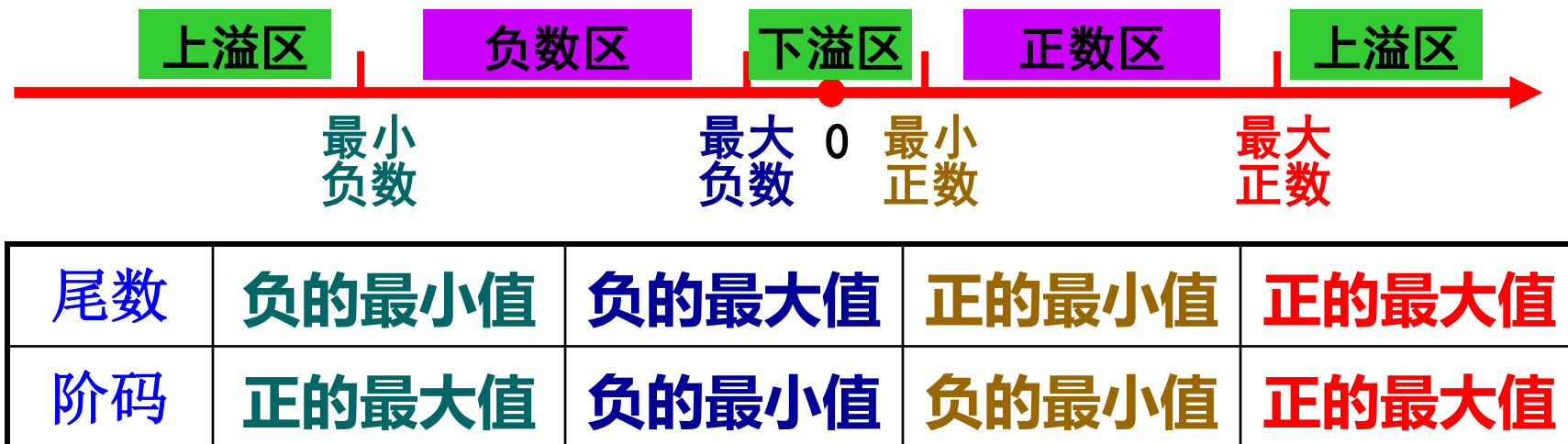
## ■ 尾数用补码表示时

- 尾数最高数值位和尾数符号位相反；  
 $\rightarrow$  为了机器方便判断
  - 尾数形如  $0.1 \times \times \dots \times$  (正)；或  $1.0 \times \times \dots \times$  (负)；  
 $\rightarrow$  范围  $\frac{1}{2} \leq M \leq (1-2^{-n})$
  - 例如， $0.011 \times 2^5$  要规格化，则变为  $0.11 \times 2^4$ ；  
 $-0.011 \times 2^5$  要规格化，则变为  $1.01 \times 2^4$ ；
- $-1 \leq M \leq -(\frac{1}{2} + 2^{-n})$

负数补码右移，左边加1，则如果有  $1.1\dots 10xxx$  它一定是  $1.0xxx$  右移的结果 故不是规格化  
 $\rightarrow$  还能左规



# 浮点数的数据表示范围



- 浮点数的溢出：阶码溢出
  - 上溢：阶码大于所能表示的最大值；
  - 下溢：阶码小于所能表示的最小值；
- 机器零：
  - 尾数为 0，或阶码小于所能表示的最小值；

# 浮点数的最大、小值

移码表示  $[-2^m, +(2^m-1)]$     补码表示  $[-1, +(1-2^{-n})]$



	非规格化数据		规格化数据	
	真值	机器数	机器数	真值
最小负数	$-1 \times 2^{+(2^m-1)}$	1 1...11; 1 00...00	同左	同左
最大负数	$-2^{-n} \times 2^{-2^m}$	0 0...00; 1 11...11	0 0...00; 1 01...11	$-(2^{-1}+2^{-n}) \times 2^{-2^m}$
最小正数	$+2^{-n} \times 2^{-2^m}$	0 0...00; 0 00...01	0 0...00; 0 10...00	$+2^{-1} \times 2^{-2^m}$
最大正数	$+(1-2^{-n}) \times 2^{+(2^m-1)}$	1 1...11; 0 11...11	同左	同左

**【例】** 设浮点数的阶码6位（含符号位），尾数为10位（含符号位），阶码采用补码表示，尾数采用原码表示，分析其浮点数表示范围。

## ■ 最大正数

- 最大正数为  $0.11\dots1 \times 2^{011\dots1}$
- 即  $(1 - 2^{-9}) \times 2^{31}$
- 该浮点数即为规格化数形式；

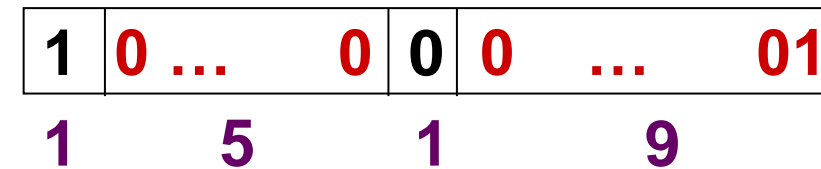
0	1 ...	1	0	1 ...	1
1	5	1	9		

【例】设浮点数的阶码6位（含符号位），尾数为10位（含符号位），阶码采用补码表示，尾数采用原码表示，分析其浮点数表示范围。

## ■ 最小正数

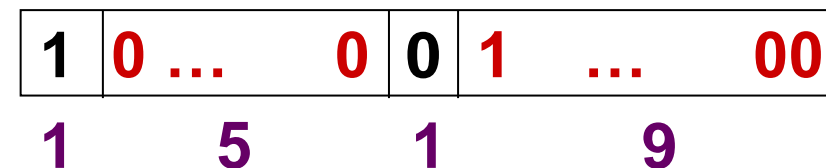
### ① 非规格化数形式

- 最小正数为  $0.0\dots01 \times 2^{10\dots0}$
- 即  $2^{-9} \times 2^{-(2^5)} = 2^{-9} \times 2^{-32}$



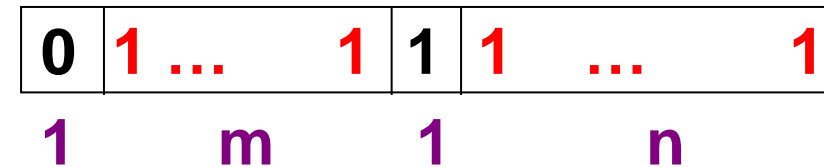
### ② 规格化数形式

- 最小正数为  $0.1 \times 2^{10\dots0}$
- $2^{-1} \times 2^{-(2^5)} = 2^{-33}$



【例】设浮点数的阶码6位（含符号位），尾数为10位（含符号位），阶码采用补码表示，尾数采用原码表示，分析其浮点数表示范围。

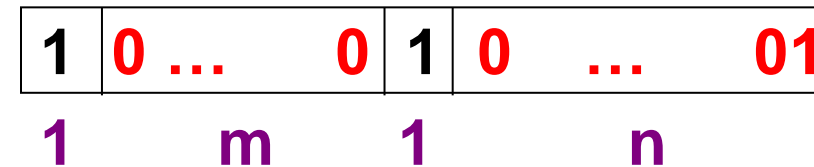
### ■最小负数



- 最小负数为  $-0.1\dots1 \times 2^{01\dots1}$
- 即  $-(1-2^{-9}) \times 2^{(2^5-1)} = -(1-2^{-9}) \times 2^{31}$
- 该浮点数即为规格化数形式；

【例】设浮点数的阶码6位（含符号位），尾数为10位（含符号位），阶码采用补码表示，尾数采用原码表示，分析其浮点数表示范围。

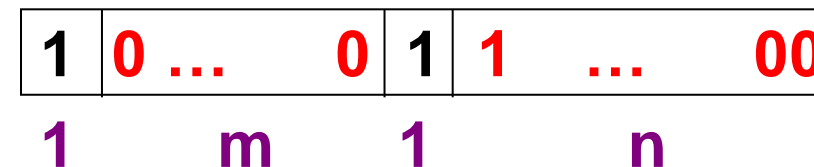
## ■ 最大负数



### ① 非规格化数形式

- 最大负数为  $-0.0\dots01 \times 2^{10\dots0}$
- 即  $-2^{-9} \times 2^{-(2^5)} = -2^{-9} \times 2^{-32}$

### ② 规格化数形式

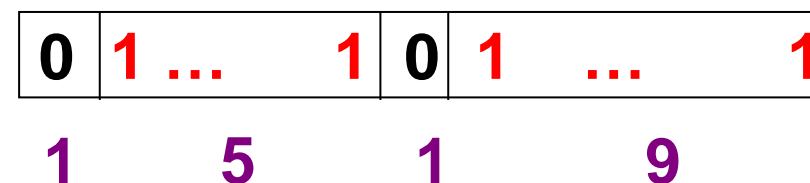


- 最大负数为  $-0.1 \times 2^{10\dots0}$
- 即  $-2^{-1} \times 2^{-(2^5)} = -2^{-1} \times 2^{-32}$

**【例】** 设浮点数的阶码6位（含符号位），尾数为10位（含符号位），阶码和尾数均采用补码表示，分析其规格化浮点数表示范围。

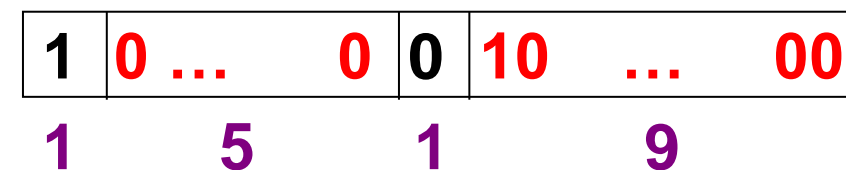
### ■ 最大正数

- 阶码最大、尾数最大
- 最大正数为  $0.11\dots1 \times 2^{11\dots1}$
- $(1-2^{-9}) \times 2^{31}$



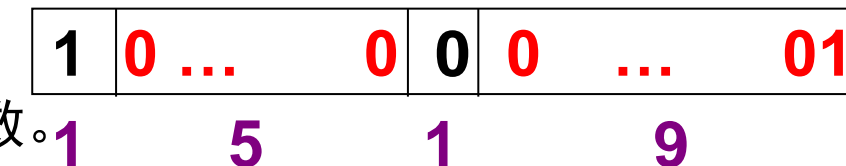
### ■ 最小正数

- 最小正数为  $0.10\dots00 \times 2^{-32}$
- 即  $2^{-32} \times 2^{-1} = 2^{-33}$



### ■ 注意：不是

- 因为  $0.0\dots1 \times 2^{-32}$  不是规格化数。



**【例】** 设浮点数的阶码6位（含符号位），尾数为10位（含符号位），阶码和尾数均采用补码表示，分析其规格化浮点数表示范围。

### ■ 最小负数

- 最小负数为  $-1.00\dots0 \times 2^{31}$
- 即  $2^{31} \times (-1) = -2^{31}$

0	1 ...	1	1	0 ...	0
1	5	1	9		

### ■ 最大负数

- 最大负数为  $-0.10\dots01 \times 2^{-32}$
- 即  $-(2^{-9} + 2^{-1}) \times 2^{-32}$
- 注意：因有规格化要求，不是

1	0 ...	0	1	0 1 ...	1
1	5	1	9		

1	0 ...	0	1	1 1 ...	1
1	5	1	9		

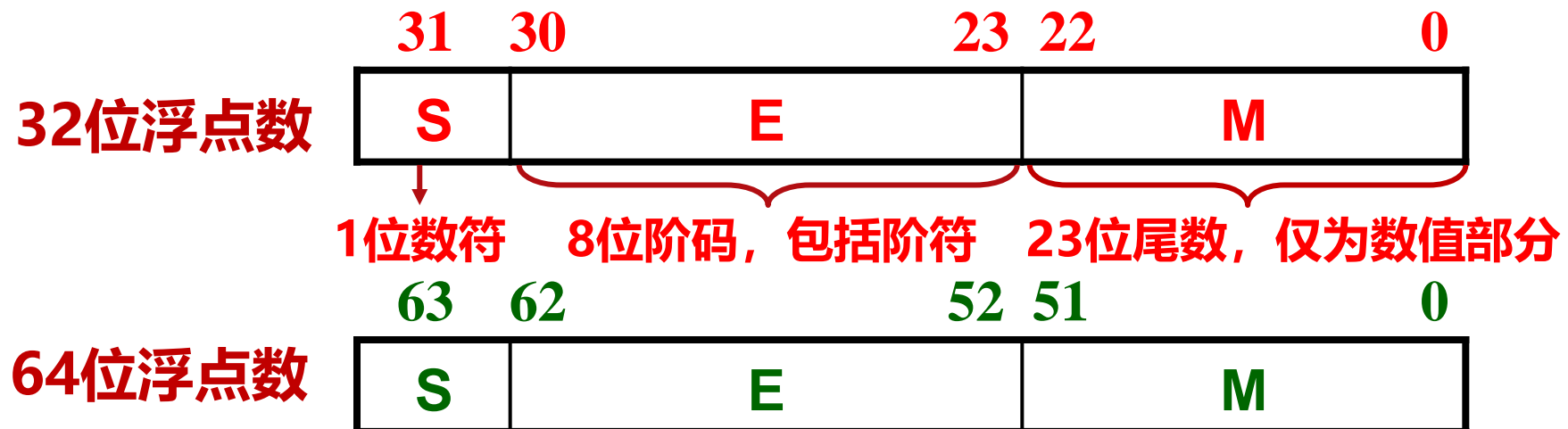


# 浮点数的IEEE754标准表示

## ■ IEEE (Institute of Electrical and Electronics Engineers)

- 美国电气及电子工程师学会
- IEEE是一家总部在美国的工程技术和电子专家的组织；
- IEEE致力于电气、电子、计算机工程和与科学有关的领域的开发和研究，也是计算机网络标准的主要制定者。

## ■ 为便于软件移植，按照 IEEE754 标准，实际机器内32位浮点数和64位浮点数的标准格式如下：



# 32位浮点数的IEEE754 标准表示

数符S	阶码E	尾数M
-----	-----	-----

- 数符S：表示浮点数的符号，占1位，0—正数、1—负数；
- 尾数M：23位，原码纯小数表示，小数点在尾数域的最前面；
  - 由于原码表示的规格化浮点数要求，最高数值位始终为1，因此该标准中隐藏最高数值位(1)，尾数的实际值为1.M；
- 阶码E：8 位，采用有偏移值的移码表示；
  - 移127码，即 $E=e+127$ ，E的8位二进制数即为移127码的编码；
- 浮点数的真值： $N = (-1)^S \times (1.M) \times 2^{E-127}$

# 32位浮点数的IEEE754 标准表示

## ■ 为什么32位浮点数阶码要有偏移量？

- 计算机为什么要用补码形式？
  - =》通过补码实现了加减法的统一
- 如果将浮点数用补码形式保存？
  - 如果仅仅采用补码作为阶码，由于阶码有正有负，整个数的符号位和阶数的符号位将导致不能进行简单的大小比较
  - 阶数采用了一个无符号的正整数存储。阶数的值直接进行二进制计算，于是阶数的值可以为0 到 255

# 32位浮点数的IEEE754 标准表示

- E为全0和全1用于表示特殊情况
- 对于规格化浮点数 E的范围变成1-254

# IEEE754 标准格式 （64位格式）



其真值表示为：

$$x = (-1)^S \times (1.M) \times 2^{E-1023} \quad e = E - 1023$$

# IEEE754 标准的数据表示

## ■ IEEE754 标准中的阶码E 0000 0000 ~ 1111 1111

### ■ 正零、负零 E=0000 0000, M=0000 ... 0000

- E与M均为零，正负之分由数据符号确定；

## ■ 注意：浮点数不能精确表示0！

$$x = (-1)^S \times (1.M) \times 2^e \quad \begin{array}{l} +0 \text{ 的机器码对} \\ \text{应的真值为} \end{array} \quad 1.0 \times 2^{-127}$$

$$e = E - 127 \quad \begin{array}{l} -0 \text{ 的机器码对} \\ \text{应的真值为} \end{array} \quad -1.0 \times 2^{-127}$$

# IEEE754 标准的数据表示

## ■ 正无穷、负无穷

- E为全1，M为全零，正负之分由数据符号确定；

## ■ 阶码E的其余值（0000 0001~1111 1110）为规格化数据；

- 真正的指数e的范围为 $-126 \sim +127$

**E=1111 1111, M=0000 ... 0000**

# IEEE754 标准对特殊数据的表示

符号位S	阶码E	尾数M	数值N
0/1	0	=0	0
0/1	0	≠0	$(-1)^S \times (0.M) \times 2^{-126}$
0/1	1~254	≠0	$(-1)^S \times (1.M) \times 2^{E-127}$
0/1	255	≠0	NaN (非数值)
0/1	255	=0	$(-1)^S \times \infty$ (无穷大)



# 非规范 (Denormalized) 浮点数

- $1.001 \times 2^{-125} - 1.0001 \times 2^{-125} = ?$
- $0.0001 \times 2^{-125}$ , 表达为规范浮点数则为  $1.0 \times 2^{-129}$

$$\begin{array}{|c|c|c|c|} \hline 0/1 & 1 \sim 254 & \neq 0 & (-1)^s \times (\underbrace{1.M}) \times 2^{\underbrace{E-127}} \\ \hline \end{array}$$

- 指数小于允许的最小指数值!

当浮点数的指数为允许的最小指数值, 即  $e_{\min}$  时, 尾数不必是规范化的。

$0.001 \times 2^{-126}$ , 其中指数-126 等于  $e_{\min}$

- [例] 若浮点数  $x$  的754标准存储格式为  $(41360000)_{16}$ ，求其浮点数的十进制数值。

■ 解：

$$(41360000)_{16} = \textcolor{red}{0}\textcolor{green}{100}\textcolor{green}{0001}\textcolor{magenta}{0011}\textcolor{magenta}{0110}\textcolor{magenta}{0000}\textcolor{magenta}{0000}\textcolor{magenta}{0000}\textcolor{magenta}{0000}$$

$\downarrow$   
**数符S**

$\underbrace{\hspace{10em}}$   
**阶码E**

$\underbrace{\hspace{15em}}$   
**尾数M**

- 指数  $e = E - 127 = \textcolor{green}{1000}\textcolor{green}{0010} - 0111\ 1111 = 0000\ 0011 = 3$
- 尾数  $1.M = \textcolor{green}{1}.\textcolor{magenta}{011}\textcolor{magenta}{0110}\textcolor{magenta}{0000}\textcolor{magenta}{0000}\textcolor{magenta}{0000}\textcolor{magenta}{0000} = 1.011011$
- 浮点数  $N = (-1)^S \times (1.M) \times 2^e$   
 $= (-1)^{\textcolor{red}{0}} \times (1.011011) \times 2^3$   
 $= (11.375)_{10}$

■ [例] 将  $(20.59375)_{10}$  转换成754标准的32位浮点数的二进制存储格式。

■ 解：

■  $(20.59375)_{10} = (10100.10011)_2$

■ 将尾数规范为1. M的形式：

$$10100.10011 = 1.010010011 \times 2^4 \quad e=4$$

■ 可得：  $M = 010010011$

$$S = 0$$


$$E = 4 + 127 = 131 = 1000\ 0011$$

■ 故，32位浮点数的754标准格式为：

$$0100\ 0001\ 1010\ 0100\ 1100\ 0000\ 0000\ 0000 = (41A4C000)_{16}$$

# 单精度浮点数与双精度浮点数


- 高级语言的float、double使用的即是IEEE754规定的格式。
- float：32位浮点值，也叫单精度浮点数（4字节保存）
- double：64位浮点值，也叫双精度浮点数（8字节保存）
- 单精度浮点数的例子：



-1100

单精度浮点数

十进制	规格化	符号	移阶码	尾数
-12	$-1.1 \times 2^3$	1	10000010	1000000 00000000 00000000
0.25	$1.0 \times 2^{-2}$	0	01111101	0000000 00000000 00000000



0.01

1位      8位      7位      8位      8位

# IEEE 754浮点数的表示范围

- 特殊情况下浮点数可以表示 $-\infty$ 到 $+\infty$

$$0/1 \quad | \quad 1 \sim 254 \quad | \quad \neq 0 \quad | \quad (-1)^S \times (\underbrace{1}_{\text{隐含}}.\underbrace{M}_{\text{尾数}}) \times 2^{\underbrace{E-127}_{\text{阶码}}}$$

最大正数?

最小正数?

最大负数?

最小负数?

# 单精度浮点数与双精度浮点数

- 除0之外，IEEE754标准中单精度浮点数所能表示的绝对值最小的规格化浮点数的格式为：

S 0000 0001 000000000000000000000000

- $V = (-1)^S \times 2^{-126} \times (1.M) = (-1)^S \times 2^{-126} \times (1+0.00\dots0)$

- 除 $\pm\infty$ 之外，IEEE754标准中单精度浮点数所能表示的绝对值最大的规格化浮点数的格式为：

S 1111 1110 111111111111111111111111

- $V = (-1)^S \times 2^{+127} \times (1.M) = (-1)^S \times 2^{+127} \times (1+0.11\dots1)$

**例：将下列十进制数表示成IEEE754格式的32位浮点数二进制存储形式**

- 27/32
- 11/512

**解：**

- $27/32 = 27 * (1/32) = (0001\ 1011)_2 * 2^{-5}$ 
  - 尾数：1.1011；      阶码： $e = -5 + 4 = -1$ ， $E = e + 127 = 126$
  - IEEE754数据：0 0111 1110 1011 0000 0000 0000 0000 000
- $11/512 = (0000\ 1011)_2 * 2^{-9}$ 
  - 尾数：1.011；      阶码： $e = -9 + 3 = -6$ ， $E = e + 127 = 121$
  - IEEE754数据：0 0111 1001 0110 0000 0000 0000 0000 00

例：将十进制数-54表示成**二进制定点数**（16位）和**浮点数**（16位，其中数值部分10位，阶码部分4位，阶符和数符各取1位），并写出它在定点机和浮点机中的机器数形式。

■ 令  $x = -54$ ，则  $x = -110110$

■ 16位定点数真值表示：  $x = -000\ 0000\ 0011\ 0110$

■ 定点机器数形式

$[x]_{\text{原}}$ : 1 000 0000 0011 0110

$[x]_{\text{补}}$ : 1 111 1111 1100 1010

■ 浮点数规格化表示：  $x = -(0.1101100000) \times 2^{110}$

■ 浮点机器数形式

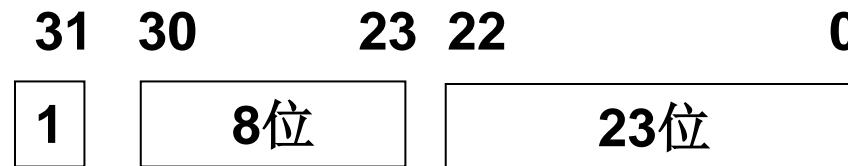
$[x]_{\text{原}}$ : 0 0110 ; 1 11 0110 0000

$[x]_{\text{补}}$ : 0 0110 ; 1 00 1010 0000

非IEEE754  
标准



计算机储存程序的特点之一是把数据和指令都作为二进制信号看待。今有一计算机字长32bit，数符位是第31bit；单精度浮点数格式如图所示。



对于二进制数 1000 1111 1110 1111 1100 0000 0000 0000

- ①表示一个补码整数，其十进制值是多少？
- ②表示一个无符号整数，其十进制值是多少？
- ③表示一个IEEE754标准的单精度浮点数，其值是多少？

二进制数 1000 1111 1110 1111 1100 0000 0000 0000

① 表示一个补码整数，其十进制值是多少？

■ 作为补码整数，其对应的原码是

1111 0000 0001 0000 0100 0000 0000 0000

■ 十进制值是  $-(2^{30} + 2^{29} + 2^{28} + 2^{20} + 2^{14})$

② 表示一个无符号整数，其十进制值是多少？

■ 作为无符号整数，其十进制值是

$2^{31} + 2^{27} + 2^{26} + 2^{25} + 2^{24} + 2^{23} + 2^{22} + 2^{21} + 2^{19} + 2^{18} + 2^{17} + 2^{16} + 2^{15} + 2^{14}$

二进制数 1000 1111 1110 1111 1100 0000 0000 0000

### ③ 作为IEEE754标准的单精度浮点数

■ 阶码E是0001 1111

■ 指数 $e = \text{阶码}E - 127 = 0001\ 1111 - 0111\ 1111$   
 $= -1100000B = -96D$

■ 尾数 $M=110\ 1111\ 1100\ 0000\ 0000\ 0000$

■ 则 $1.M = 1.110\ 1111\ 1100\ 0000\ 0000\ 0000$   
 $= 1.110\ 1111\ 11$

∴单精度浮点数值为：

$$\begin{aligned} X &= (-1)^s \times 1.M \times 2^e = -(1.110\ 1111\ 11) \times 2^{-96} \\ &= -(0.1110\ 1111\ 11) \times 2^{-95} \\ &= -(14 \times 16^{-1} + 15 \times 16^{-2} + 12 \times 16^{-3}) \times 2^{-95} \\ &= -0.3115 \times 2^{-95} \end{aligned}$$

## 练习题

假定变量  $i$ ,  $f$ ,  $d$  数据类型分别为  $\text{int}$ ,  $\text{float}$ ,  $\text{double}$  ( $\text{int}$  用补码表示,  $\text{float}$  和  $\text{double}$  用 IEEE754 单精度和双精度浮点数据格式表示), 已知  $i=785$ ,  $f=1.5678\text{e}3$ ,  $d=1.5\text{e}100$ , 若在 32 位机器中执行下列关系表达式, 则结果为真的是 ()

- (I)  $i==(int)(float)i$
- (II)  $f==(float)(int)f$
- (III)  $f==(float)(double)f$
- (IV)  $(d+f)-d==f$

- A. 仅 I 和 II
- B. 仅 I 和 III
- C. 仅 II 和 III
- D. 仅 III 和 IV

关键是 “==”  
两端的数据类  
型是否一致!