

计算机组成原理

文杰

计算机科学与技术学院

wenjie@hit.edu.cn

个人主页: <http://faculty.hitsz.edu.cn/wenjie>

第二章 计算机的运算方法

第二章 计算机的运算方法

- 计算机中数的表示
- 定点运算与浮点运算

计算机中数的表示

- 计算机中数的表示

- 无符号数和有符号数

- 机器数与真值；原码/补码/反码/移码表示法

- 定点表示和浮点表示

- IEEE754标准

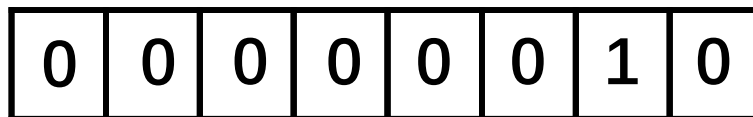
无符号数

- 以8位寄存器为例

十进制

二进制

无符号数2



8 位



0 ~ 255

16 位



0 ~ 65535

寄存器的位数反映无符号数的表示范围

有符号数：真值与机器数

真值：带符号的数

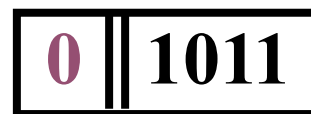
+ 0.1011或0.1011

- 0.1011

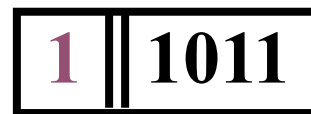
+ 1100或1100

- 1100

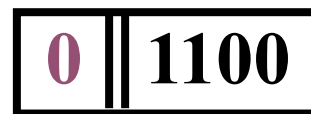
机器数：符号数字化的数



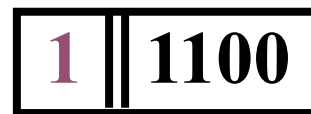
↑ 小数点的位置



↑ 小数点的位置



↑ 小数点的位置



↑ 小数点的位置

注：以后非特殊说明，默认二进制数表示；

二进制数位数不是8的倍数，只是为了讲解方便。

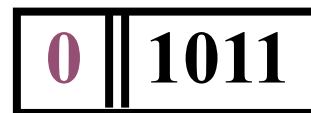
有符号数：真值与机器数

真值：带符号的数

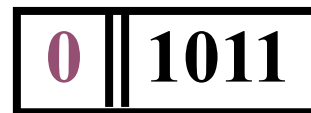
+ 0.1011

+ 1011

机器数：符号数字化的数



↑ 小数点的位置



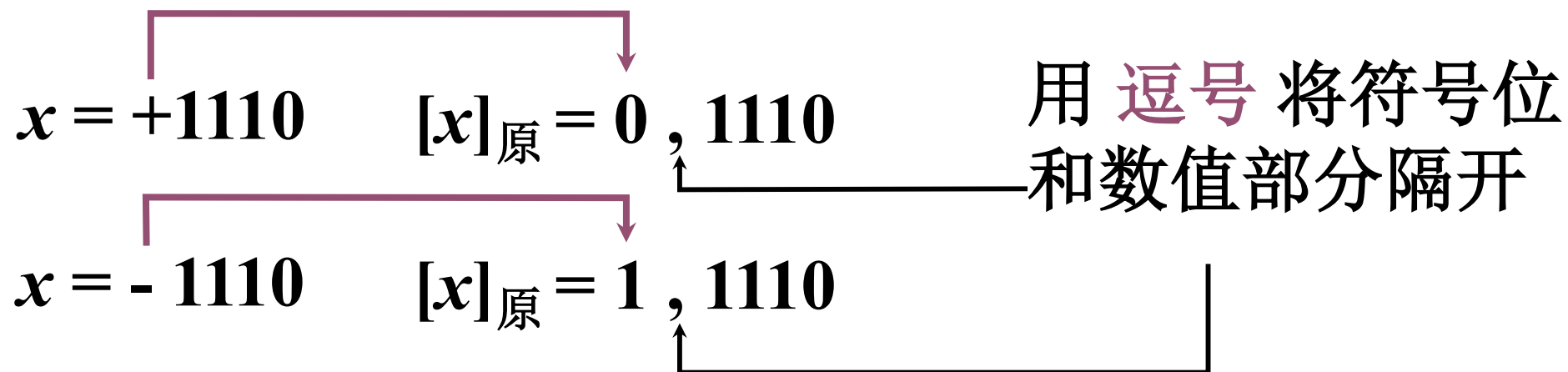
↑ 小数点的位置



- 小数和整数在计算机中的表示一模一样？
- 如何表示既有整数又有小数的数据，如+11.01？

原码表示法：整数

带符号的绝对值表示



$$[x]_{\text{原}} = 2^4 - (-1110) = 1, 1110$$

$$[x]_{\text{原}} = \begin{cases} 0, x & 2^n > x \geq 0 \\ 2^n - x & 0 \geq x > -2^n \end{cases}$$

x 是真值， n 是数值位数

原码表示法：小数

$$x = +0.1101 \quad [x]_{\text{原}} = 0.1101$$

用 **小数点** 将符号位和数值部分隔开

$$x = -0.1101 \quad [x]_{\text{原}} = 1.1101 = 1 - (-0.1101)$$

$$x = +0.1000000 \quad [x]_{\text{原}} = 0.1000000$$

用 **小数点** 将符号位和数值部分隔开

$$x = -0.1000000 \quad [x]_{\text{原}} = 1.1000000 = 1 - (-0.1000000)$$

$$[x]_{\text{原}} = \begin{cases} x & 1 > x \geq 0 \\ 1 - x & 0 \geq x > -1 \end{cases}$$

x 是真值

举例：根据原码求真值

- 例1. 已知 $[x]_{\text{原}} = 1.0011$, 求 x

解：由小数原码定义得

$$x = 1 - [x]_{\text{原}} = 1 - 1.0011 = -0.0011$$

- 例2. 已知 $[x]_{\text{原}} = 1,1100$, 求 x

解：由整数原码定义得

$$x = 2^4 - [x]_{\text{原}} = 10000 - 1,1100 = -1100$$

举例

- 例3. 已知 $[x]_{\text{原}} = 0.1101$, 求 x

解: 根据定义 $\because [x]_{\text{原}} = 0.1101$

$$\therefore x = +0.1101$$

- 例4. 求 $x = 0$ 的原码

解: 对于小数, 设 $x = +0.0000$ 则 $[+0.0000]_{\text{原}} = 0.0000$

$$x = -0.0000 \text{ 则 } [-0.0000]_{\text{原}} = 1.0000$$

同理, 对于整数 $[+0]_{\text{原}} = 0,0000$ 假设整数的数值位是4位

$$[-0]_{\text{原}} = 1,0000$$

$$\therefore [+0]_{\text{原}} \neq [-0]_{\text{原}}$$

注意: $x = 0$ 的原码要分成小数和整数分别讨论

原码的优缺点

- 优点：简单、直观
- 缺点：1) +0和-0原码不一样
2) 作加法运算时，会出现如下问题：

要求	数1	数2	实际操作	结果符号
加法	正	正	加法	正
加法	正	负	减法	可正可负
加法	负	正	减法	可正可负
加法	负	负	加法	负

实际运算时能否只作加法运算？

原码的缺点

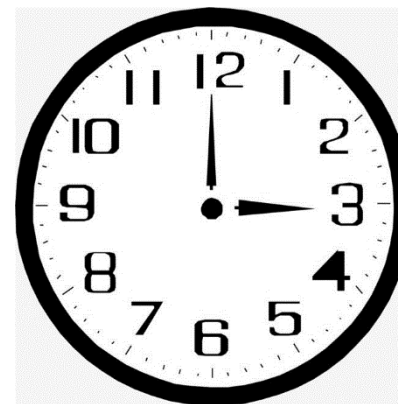
十进制		原码									
	3		<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table>	0	0	0	0	0	0	1	1
0	0	0	0	0	0	1	1				
+	-3	+	<table border="1"><tr><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table>	1	0	0	0	0	0	1	1
1	0	0	0	0	0	1	1				
<hr/>		<hr/>									
	0		<table border="1"><tr><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td></tr></table>	1	0	0	0	0	1	1	0
1	0	0	0	0	1	1	0				
		-6	<div>转十进制</div>								



原码运算 不等于 十进制运算?

补数表示法

- 例：将时钟从5点调到3点
- 补的概念：时钟以12为模
 - 逆时针： $5 - 2 = 3$
 - 顺时针： $5 + 10 = 3 + 12$
- 可见 -2 可用 $+10$ 代替
 - 称 $+10$ 是 -2 （以 12 为模）的**补数**
 - 记作 $-2 \equiv +10 \pmod{12}$
 - 同理 $-4 \equiv +8 \pmod{12}$
 - $-5 \equiv +7 \pmod{12}$



减法 \longrightarrow 加法

补数——续

计数器（模 16） $1011 \longrightarrow 0000 ?$

$$\begin{array}{r} 1011 \\ - 1011 \\ \hline 0000 \end{array}$$

$$\begin{array}{r} 1011 \\ + 0101 \\ \hline 10000 \end{array}$$

可见 -1011 与 $+0101$ 作用等价

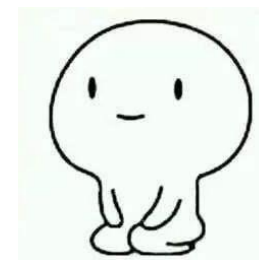
记作 $-1011 \equiv +0101 \pmod{2^4}$

同理 $-011 \equiv +101 \pmod{2^3}$

$-0.1001 \equiv +1.0111 \pmod{2^1}$

自然去掉

正数的补数？



- 结论（真值的绝对值小于模）
 - 一个负数加上 “模” 即得该负数的补数
 - 一个负数和一个正数互为补数时，绝对值之和即为模数

补数——续

正数的补数即为其本身

对于时钟：3点、15点、27点都是3点 \longrightarrow $3 \equiv 15 \equiv 27 \pmod{12}$
 $3 \equiv 3+12 \equiv 3+24 \equiv 3 \pmod{12}$

同理： $+0101 \equiv +0101 + 2^4 \equiv +0101 \pmod{2^4}$

前面已证明： $-1011 \equiv -1011 + 2^4 \equiv +0101 \pmod{2^4}$

$\boxed{+}0101 \longrightarrow 0, 0101$

$\boxed{-}1011 \longrightarrow 1, 0101$

$$\begin{array}{r} 2^{4+1} + (-1011) = 100000 \\ \quad \quad \quad -1011 \\ \hline 1,0101 \pmod{2^{4+1}} \end{array}$$

问题：+ 0101 究竟是-1011的补数还是+0101的补数呢？



补码表示法：二进制整数

整数补码定义：

$$[x]_{\text{补}} = \begin{cases} 0, x & 2^n > x \geq 0 \\ 2^{n+1} + x & 0 > x \geq -2^n \pmod{2^{n+1}} \end{cases}$$

其中： x 为真值， n 为二进制整数的位数

$$x = +0101000$$

$$[x]_{\text{补}} = 0,0101000$$

用 逗号 将符号位
和数值部分隔开

$$x = -1011000$$

$$[x]_{\text{补}} = 2^{7+1} + (-1011000)$$

$$= 100000000$$

$$- \quad 1011000$$

$$\hline 1,0101000$$

补码表示法：二进制（纯）小数

小数补码定义：

$$[x]_{\text{补}} = \begin{cases} x & 1 > x \geq 0 \\ 2 + x & 0 > x \geq -1 \pmod{2} \end{cases}$$

其中： x 为真值

$$x = +0.1110$$

$$[x]_{\text{补}} = 0.1110$$

用 小数点 将符号位
和数值部分隔开

$$x = -0.1100000$$

$$\begin{aligned} [x]_{\text{补}} &= -0.1100000 + 2 \\ &= 10.0000000 \\ &\quad - 0.1100000 \\ &= 1.0100000 \end{aligned}$$

求补码的快捷方式

设 $x = -1010$ 时

$$\begin{aligned} \text{则 } [x]_{\text{补}} &= 2^{4+1} - 1010 = 11111 + 1 - 1010 \\ &= 100000 = 11111 + 1 \\ &\quad - 1010 \\ \hline &= 1,0110 \end{aligned} \qquad \begin{aligned} &= 11111 + 1 \\ &\quad - 1010 \\ \hline &= \boxed{10101} + 1 \\ &= 1,0110 \end{aligned}$$

$$\text{又 } [x]_{\text{原}} = \boxed{1,1010}$$

当真值为负时，补码可用原码除符号位外
每位取反，末位加1求得

举例：已知小数补码求真值

已知 $[x]_{\text{补}} = 1.0001$ ，求 x 和 $[x]_{\text{原}}$ 。

解：由定义得 $x = [x]_{\text{补}} - 2$

$$= 1.0001 - 10.0000$$
$$= -0.1111$$

根据真值 x

$[x]_{\text{补}}$ 除符号位外，按位取反

$\rightarrow [x]_{\text{原}} = 1.1111 = 1.1110 + 0.0001$

结论：当真值为负时，已知补码求原码的快捷方法：

- 补码除符号位外，每位取反，末位加 1
- 补码除符号位外，末位减 1，再每位取反

练习：已知负数补码求真值

已知 $[x]_{\text{补}} = 1,1110$ ，求 x

解：由定义得 $x = [x]_{\text{补}} - 2^{4+1}$

$$= 1,1110 - 100000$$
$$= -0010$$

或： $[x]_{\text{补}} \rightarrow [x]_{\text{原}} \rightarrow x$

根据快捷求法： $[x]_{\text{原}} = 1, 0001 + 1 = 1,0010$

$$\therefore x = -0010$$

练习：求下列真值的补码

真值	$[x]_{\text{补}}$	$[x]_{\text{原}}$
$x = -70 = -1000110$	1, 0111010	1,1000110
$x = -0.1110$	1.0010	1.1110
$x = +0.0000$ $[+0]_{\text{补}} = [-0]_{\text{补}}$	0.0000	0.0000
$x = -0.0000$	0.0000	1.0000
$x = -1.0000$	1.0000	不能表示

由小数补码定义
$$[x]_{\text{补}} = \begin{cases} x & 1 > x \geq 0 \\ 2 + x & 0 > x \geq -1 \pmod{2} \end{cases}$$

$$[-1.0000]_{\text{补}} = 2 + x = 10.0000 - 1.0000 = 1.0000$$

反码表示法：二进制整数

整数反码定义：

$$[x]_{\text{反}} = \begin{cases} 0, x & 2^n > x \geq 0 \\ (2^{n+1}-1) + x & 0 \geq x > -2^n \quad (\text{mod } 2^{n+1}-1) \end{cases}$$

其中： x 为真值， n 为数值位数

$$x = +1101$$

$$[x]_{\text{反}} = 0,1101$$



用逗号将符号位
和数值部分隔开

$$x = -1101$$

$$[x]_{\text{反}} = (2^{4+1}-1) - 1101$$

$$= 11111 - 1101$$

$$= 1,0010$$

$$[x]_{\text{原}} = 1,1101$$

数值位按位取反

反码表示法：二进制小数

小数反码定义：

$$[x]_{\text{反}} = \begin{cases} x & 1 > x \geq 0 \\ (2-2^{-n}) + x & 0 \geq x > -1 \quad (\text{mod } 2-2^{-n}) \end{cases}$$

其中： x 为真值， n 为数值位数

$$x = -0.1010$$

$$[x]_{\text{反}} = (2-2^{-4}) - 0.1010$$

$$= 1.1111 - 0.1010$$

$$= 1.0101$$

用 小数点 将符号位

和数值部分隔开

$$[x]_{\text{原}} = 1.1010$$

数值位按位取反

例子：已知反码求真值，0的反码

- 已知 $[x]_{\text{反}} = 1,1110$ ，求 x

解：由定义得 $x = [x]_{\text{反}} - (2^{4+1} - 1)$
$$= 1,1110 - 11111$$
$$= -0001$$

- 求 0 的反码

解：对于小数：

设 $x = +0.0000$ ， $[+0.0000]_{\text{反}} = 0.0000$

$x = -0.0000$ ， $[-0.0000]_{\text{反}} = 1.1111$

同理，对于整数 $[+0]_{\text{反}} = 0,0000$ ， $[-0]_{\text{反}} = 1,1111$

$$[+0]_{\text{反}} \neq [-0]_{\text{反}}$$

三种机器数的小结

- 最高位为符号位，**书写上**用 “,” （整数）或 “.” （小数）将数值部分和符号位隔开
- **对于正数，符号位为0，原码 = 补码 = 反码**
- 对于负数，符号位为 1，其**数值**部分
 - 原码除符号位外每位取反（**反码**） 末位加 1 **→ 补码**
- 当真值为 **负** 时，**已知补码求原码**的方法：
 - 补码除符号位外，每位取反，末位加 1
 - 补码除符号位外，末位减 1，再每位取反

例子：机器数的真值

- 设机器数字长为8位(其中1位为符号位);对于整数,当其分别代表无符号数、原码、补码和反码时,对应的真值范围各为多少?

二进制代码	无符号数 对应的真值	原码对应 的真值	补码对应 的真值	反码对应 的真值
00000000	0	+0	± 0	+0
00000001	1	+1	+1	+1
00000010	2	+2	+2	+2
⋮	⋮	⋮	⋮	⋮
01111111	127	+127	+127	+127
10000000	128	-0	-128	-127
10000001	129	-1	-127	-126
⋮	⋮	⋮	⋮	⋮
11111101	253	-125	-3	-2
11111110	254	-126	-2	-1
11111111	255	-127	-1	-0

例：已知 $[y]_{\text{补}}$ ，求 $[-y]_{\text{补}}$

例：设 $[y]_{\text{补}} = y_0.y_1y_2 \cdots y_n$ ，其中 $y \neq -1.0 \cdots 0$ ，求 $[-y]_{\text{补}}$ 。

解： <I> $[y]_{\text{补}} = 0.y_1y_2 \cdots y_n$

$$y = 0.y_1y_2 \cdots y_n$$

$$-y = -0.y_1y_2 \cdots y_n$$

$$[-y]_{\text{补}} = 1.\overline{y_1}\overline{y_2} \cdots \overline{y_n} + 2^{-n}$$

<II> $[y]_{\text{补}} = 1.y_1y_2 \cdots y_n$

$$[y]_{\text{原}} = 1.\overline{y_1}\overline{y_2} \cdots \overline{y_n} + 2^{-n}$$

$$y = -(0.\overline{y_1}\overline{y_2} \cdots \overline{y_n} + 2^{-n})$$

$$-y = 0.\overline{y_1}\overline{y_2} \cdots \overline{y_n} + 2^{-n}$$

$$[-y]_{\text{补}} = 0.\overline{y_1}\overline{y_2} \cdots \overline{y_n} + 2^{-n}$$

$[y]_{\text{补}}$ 连同符号位在内，
每位取反，末位加 1，
即得 $[-y]_{\text{补}}$

补码加减

计算器 程序员

记忆 127 + 1 = -128 存储器中未保存数据

HEX 80
DEC -128
OCT 200
BIN 1000 0000

BYTE MS

按位 位移位

A	<<	>>	CE	⊗
B	()	%	÷
C	7	8	9	×
D	4	5	6	-
E	1	2	3	+
F	+/-	0	.	=

移码表示法

- 补码表示很难直接判断其真值大小

如	十进制	二进制	补码	
	$x = +21$	+10101	0,10101	错
	$x = -21$	-10101	1,01011	大
	$x = +31$	+11111	0,11111	错
	$x = -31$	-11111	1,00001	大
以上	$x + 2^5$	+10101 + 100000 = 110101		大 正确
		-10101 + 100000 = 001011		大 正确
		+11111 + 100000 = 111111		大 正确
		-11111 + 100000 = 000001		大 正确

移码表示法：二进制整数

- 移码定义

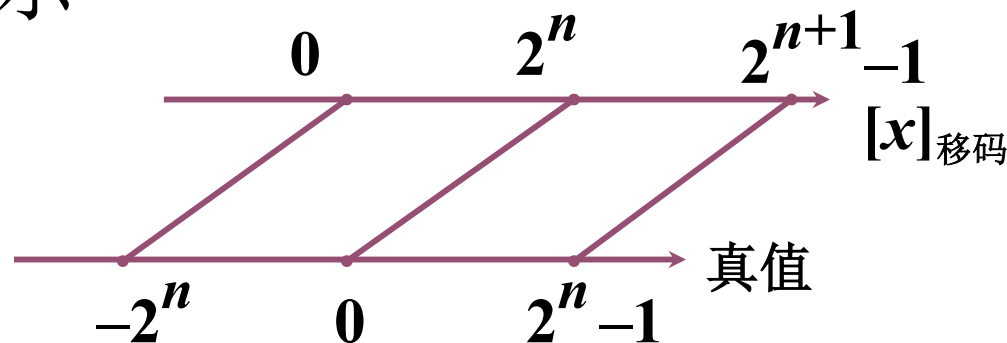
$$[x]_{\text{移}} = 2^n + x \quad (2^n > x \geq -2^n)$$

其中： x 为真值， n 为 整数的位数

小数的移码定义呢？



- 移码在数轴上的表示



- 例：

$$x = 10100$$

$$[x]_{\text{移}} = 2^5 + 10100 = 1,10100$$

$$x = -10100$$

$$[x]_{\text{移}} = 2^5 - 10100 = 0,01100$$

用 逗号 将符号位
和数值位隔开

移码和补码的比较

设 $x = +1100100$

$$[x]_{\text{移}} = 2^7 + 1100100 = \mathbf{1},1100100$$

$$[x]_{\text{补}} = \mathbf{0},1100100$$

设 $x = -1100100$

$$[x]_{\text{移}} = 2^7 + (-1100100) = \mathbf{0},0011100$$

$$[x]_{\text{补}} = 2^{7+1} - 1100100 = \mathbf{1},0011100$$

补码与移码只差一个符号位

真值、补码和移码的对照表

补码与移码只差一个符号位

真值 x ($n=5$)	$[x]_{\text{补}}$	$[x]_{\text{移}}$	$[x]_{\text{移}}$ 对应的 十进制整数
- 1 0 0 0 0 0	1 0 0 0 0 0	0 0 0 0 0 0	0
- 1 1 1 1 1	1 0 0 0 0 1	0 0 0 0 0 1	1
- 1 1 1 1 0	1 0 0 0 1 0	0 0 0 0 1 0	2
⋮	⋮	⋮	⋮
- 0 0 0 0 1	1 1 1 1 1 1	0 1 1 1 1 1	31
± 0 0 0 0 0	0 0 0 0 0 0	1 0 0 0 0 0	32
+ 0 0 0 0 1	0 0 0 0 0 1	1 0 0 0 0 1	33
+ 0 0 0 1 0	0 0 0 0 1 0	1 0 0 0 1 0	34
⋮	⋮	⋮	⋮
+ 1 1 1 1 0	0 1 1 1 1 0	1 1 1 1 1 0	62
+ 1 1 1 1 1	0 1 1 1 1 1	1 1 1 1 1 1	63

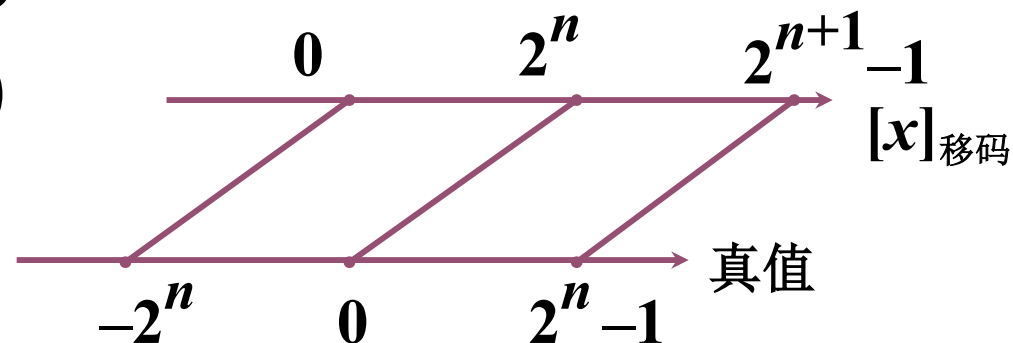
移码的特点

续前表, $n=5$

$$[+0]_{\text{移}} = 2^5 + 0 = 1,00000$$

$$[-0]_{\text{移}} = 2^5 - 0 = 1,00000$$

$$[+0]_{\text{移}} = [-0]_{\text{移}}$$



最小真值 $-2^5 = -100000$ 对应的移码为 $2^5 - 100000 = 000000$

最小真值的移码为全 0

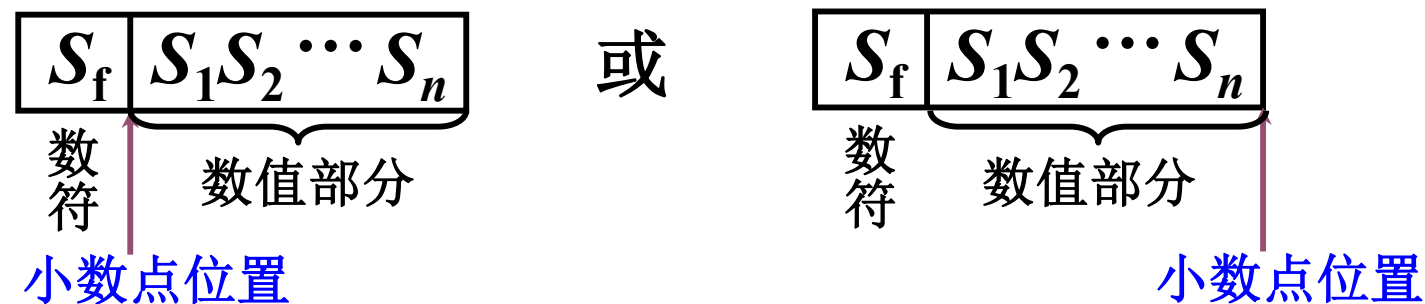
可用移码思想表示浮点数的阶码, 便于判断浮点数阶码大小

第二章 计算机中数的表示

- 计算机中数的表示
 - 无符号数和有符号数
 - 定点表示和浮点表示
 - IEEE754标准
 - 算数移位与逻辑移位

定点表示

- 小数点按约定方式标出
- 定点表示



定点机

小数定点机

整数定点机

原码

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

$$-(2^n - 1) \sim +(2^n - 1)$$

补码

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

$$-2^n \sim +(2^n - 1)$$

反码

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

$$-(2^n - 1) \sim +(2^n - 1)$$

浮点表示



为什么要引入浮点数表示？

- **编程困难**，程序员要调节小数点的位置；
- **数的表示范围小**，为了能表示两个大小相差很大的数据，需要很长的机器字长；

例如：太阳的质量是 0.2×10^{34} 克，一个电子的质量大约为 0.9×10^{-27} 克，两者的差距为 10^{61} 以上，若用定点数据表示： $2^x > 10^{61}$ ，解得： $x > 203$ 位。

- **数据存储单元的利用率往往很低。**

浮点表示

$N = S \times r^j$ 浮点数的一般形式

S 尾数 j 阶码 r 基数（基值）

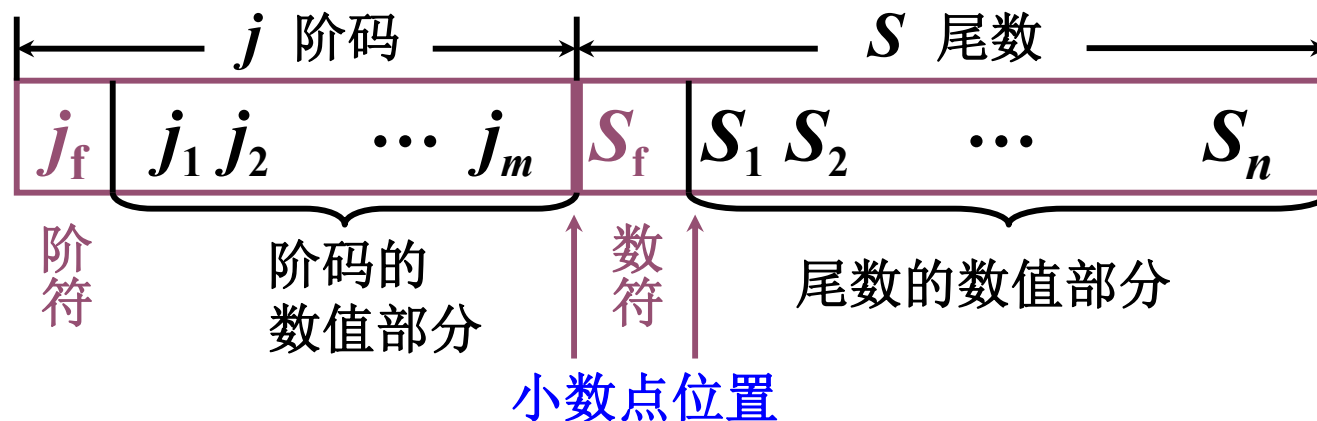
计算机中 r 取 2、4、8、16 等

当 $r = 2$ $N = 11.0101$ 二进制表示
 $\checkmark = 0.110101 \times 2^{10}$ 规格化数
 $= 1.10101 \times 2^1$
 $= 1101.01 \times 2^{-10}$
 $\checkmark = 0.00110101 \times 2^{100}$

计算机中 S 小数、可正可负
 j 整数、可正可负

浮点数的表示形式

$$N = S \times r^j$$



S_f 代表浮点数的符号

n 其位数反映浮点数的精度

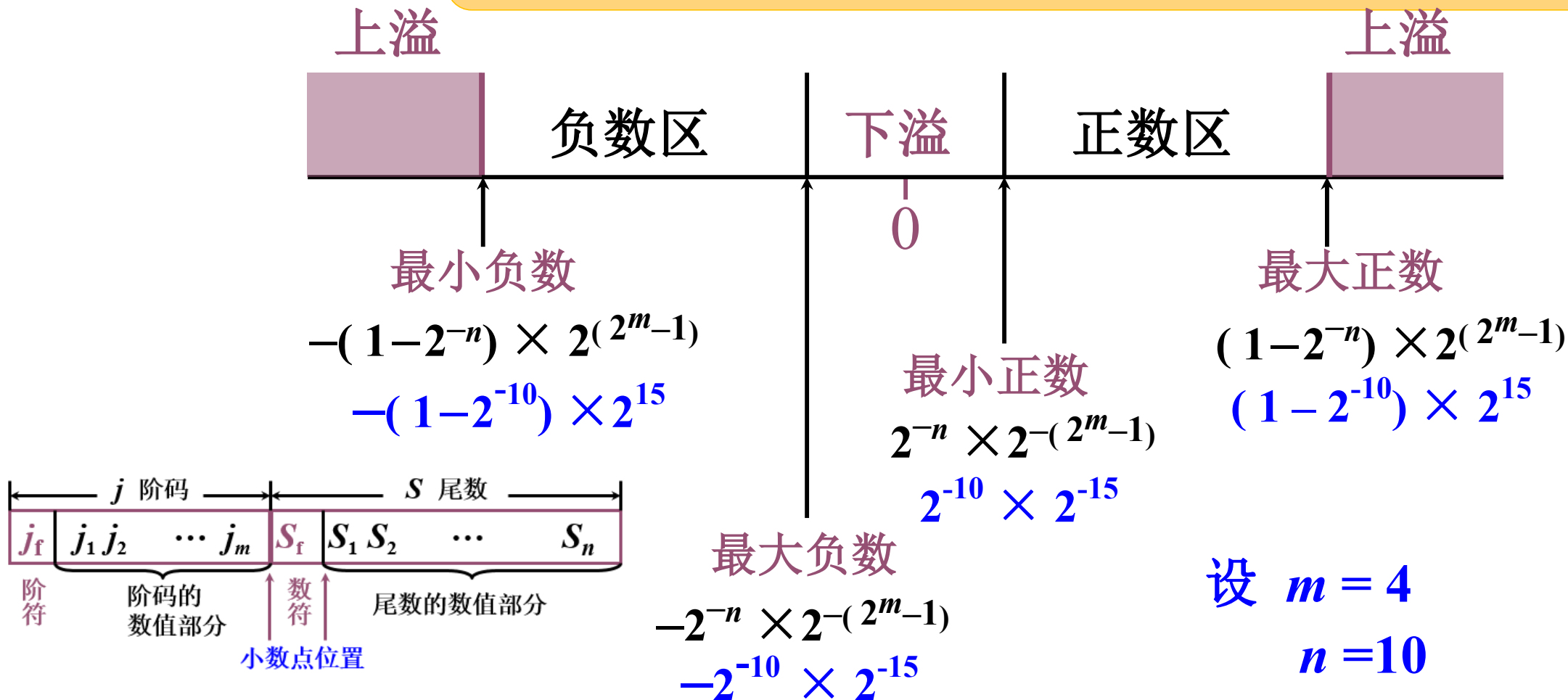
m 其位数反映浮点数的表示范围

j_f 和 m 共同表示小数点的实际位置

浮点数的表示范围

$$N = S \times r^j$$

上溢 阶码 > 最大阶码 机器停止运算，中断溢出处理
下溢 阶码 < 最小阶码 按机器零处理



练习

- 设机器数字长为 24 位，欲表示 ± 3 万的十进制数，试问在保证**数的最大精度**的前提下，除阶符、数符各取 1 位外，阶码、尾数各取几位？

解： $\because 2^{14} = 16384 \quad 2^{15} = 32768$

\therefore 15 位二进制数可反映 ± 3 万之间的十进制数

$$2^{15} \times 0.\underbrace{\times \times \times \dots \times \times \times}_{15\text{位}}$$

$m = 4, 5, 6, \dots$

满足 **最大精度** 可取 $m = 4, n = 18$

- 浮点数的规格化形式

$r = 2$ 尾数最高位为 1

$r = 4$ 尾数最高 2 位不全为 0

$r = 8$ 尾数最高 3 位不全为 0

基数不同，浮点数的
规格化形式不同

- 浮点数的规格化

$r = 2$ 左规 尾数左移 1 位，阶码减 1

右规 尾数右移 1 位，阶码加 1

$r = 4$ 左规 尾数左移 2 位，阶码减 1

右规 尾数右移 2 位，阶码加 1

$r = 8$ 左规 尾数左移 3 位，阶码减 1

右规 尾数右移 3 位，阶码加 1

$$N = 11.0101$$

$$\checkmark = 0.110101 \times 2^{10}$$

$$= 1.10101 \times 2^1$$

$$= 1101.01 \times 2^{-10}$$

$$= 0.00110101 \times 2^{100}$$

基数 r 越大，可表示的浮点数的范围越大

基数 r 越大，浮点数的精度越低

机器数规格化

(1) 规格化数的定义

$$r = 2^{-\frac{1}{2}} \leq |S| < 1$$

(2) 规格化数的判断

$S > 0$	规格化形式	$S < 0$	规格化形式
真值	$0.1 \times \times \cdots \times$	真值	$-0.1 \times \times \cdots \times$
原码	0.1 $\times \times \cdots \times$	原码	1.1 $\times \times \cdots \times$
补码	0.1 $\times \times \cdots \times$	补码	1.0 $\times \times \cdots \times$
反码	$0.1 \times \times \cdots \times$	反码	$1.0 \times \times \cdots \times$

原码 不论正数、负数，第一数位为1

补码 符号位和第1数位不同

特例

(1) 规格化数的定义

$$r = 2 \quad \frac{1}{2} \leq |S| < 1$$

$$S = -\frac{1}{2} = -0.100 \cdots 0$$

$$[S]_{\text{原}} = 1.100 \cdots 0$$

$$[S]_{\text{补}} = \boxed{1.1}00 \cdots 0$$

$\therefore [-\frac{1}{2}]_{\text{补}}$ 不是规格化的数

$$S = -1$$

$$[S]_{\text{补}} = \boxed{1.0}00 \cdots 0$$

$\therefore [-1]_{\text{补}}$ 是规格化的数

- 例1. 设 $m = 4$, $n = 10$, $r = 2$, 求尾数规格化后的浮点数表示范围（阶码与尾数都是原码表示）

$$\text{最大正数} \quad 2^{+1111} \times \underbrace{0.1111111111}_{10 \text{ 个 } 1} = 2^{15} \times (1 - 2^{-10})$$

$$\text{最小正数} \quad 2^{-1111} \times \underbrace{0.1000000000}_{9 \text{ 个 } 0} = 2^{-15} \times 2^{-1} = 2^{-16}$$

$$\text{最大负数} \quad 2^{-1111} \times (-\underbrace{0.1000000000}_{9 \text{ 个 } 0}) = -2^{-15} \times 2^{-1} = -2^{-16}$$

$$\text{最小负数} \quad 2^{+1111} \times (-\underbrace{0.1111111111}_{10 \text{ 个 } 1}) = -2^{15} \times (1 - 2^{-10})$$

- 例2. 将 $+\frac{19}{128}$ 写成二进制定点数、浮点数及在定点机和浮点机中的机器数（原/补/反码）形式。其中数值部分均取10位，数符取1位，浮点数阶码取5位（含1位阶符）。

解： 设 $x = +\frac{19}{128}$

二进制形式 $x = 0.0010011$

定点表示 $x = 0.0010011\ 000$

浮点规格化形式 $x = 0.1001100000 \times 2^{-10}$

定点机中 $[x]_{\text{原}} = [x]_{\text{补}} = [x]_{\text{反}} = 0.0010011000$

浮点机中

	阶码	尾数
$[x]_{\text{原}}$	$1, 0010$	0.1001100000

$[x]_{\text{补}} = 1, 1110; 0.1001100000$

$[x]_{\text{反}} = 1, 1101; 0.1001100000$

- 例3. 将 -58 表示成二进制定点数和浮点数，并写出它在定点机和浮点机中的**三种机器数**及**阶码为移码、尾数为补码**的形式
（其中数值部分 10 位，数符1位，阶码5 位（含1位阶符））。

解： 设 $x = -58$

二进制形式 $x = -111010$

定点表示 $x = -0000111010$

浮点规格化形式 $x = -(0.1110100000) \times 2^{110}$

定点机中

$[x]_{\text{原}} = 1, 0000111010$

$[x]_{\text{补}} = 1, 1111000110$

$[x]_{\text{反}} = 1, 1111000101$

浮点机中

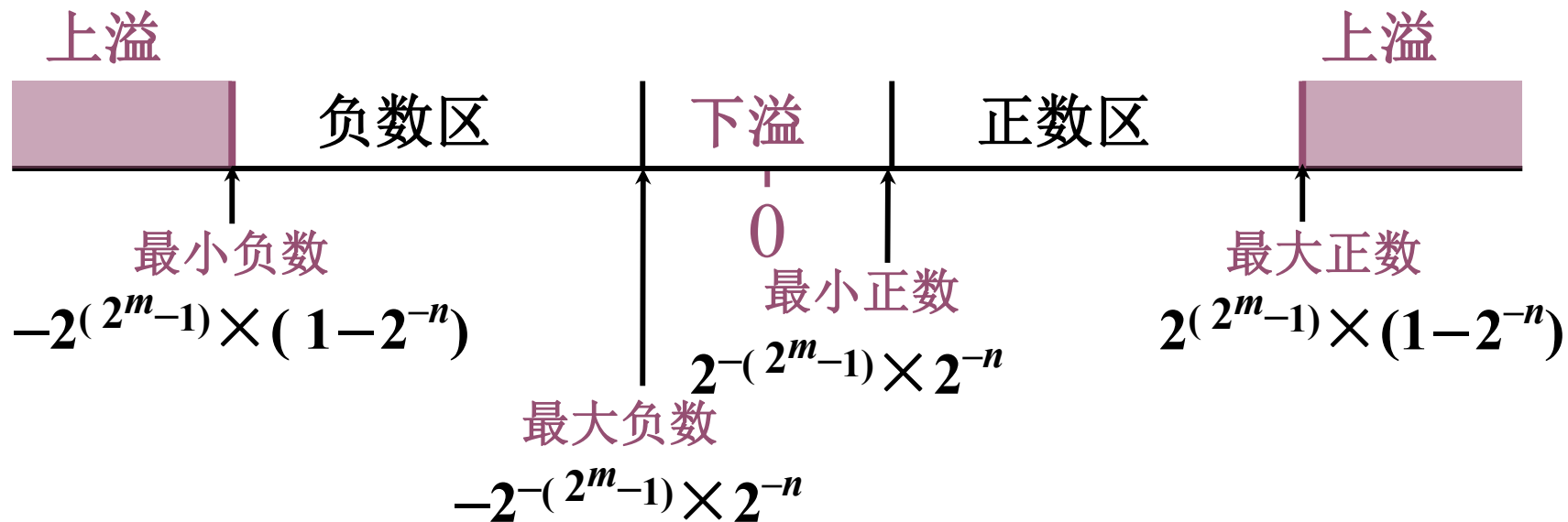
$[x]_{\text{原}} = 0, 0110; 1. 1110100000$

$[x]_{\text{补}} = 0, 0110; 1. 0001100000$

$[x]_{\text{反}} = 0, 0110; 1. 0001011111$

$[x]_{\text{阶移、尾补}} = 1, 0110; 1. 0001100000$

- 例4. 写出对应下图所示的浮点数的**补码**形式。设 $n = 10$, $m = 4$, 阶符、数符各取1位。



解:

真值

补码

最大正数	$2^{15} \times (1-2^{-10})$	0,1111; 0.1111111111
最小正数	$2^{-15} \times 2^{-10}$	1,0001; 0.0000000001
最大负数	$-2^{-15} \times 2^{-10}$	1,0001; 1.1111111111
最小负数	$-2^{15} \times (1-2^{-10})$	0,1111; 1.0000000001

机器零

- 当浮点数尾数为 0 时，不论其阶码为何值，按机器零处理
- 当浮点数阶码**小于**它所表示的最小数时，按机器零处理

如 $m = 4$ $n = 10$

当阶码和尾数都用**补码**表示时，机器零为

$\times, \times \times \times \times; \quad \mathbf{0.00 \dots 0}$

或者阶码 < -16 ，按照机器零处理

当阶码用**移码**，尾数用**补码**表示时，机器零为

$\mathbf{0,0000; 0.00 \dots 0}$

有利于机器中“判 0”电路的实现

16位长的浮点数，其中阶码7位（含1位阶符），尾数9位（含1位数符），当浮点数采用原码表示时，表示的数的范围是 [填空1]，当浮点数采用补码表示时，数的表示的范围是 [填空2]

- A. $-2^{64} \sim 2^{64} (1-2^{-8})$
- B. $-2^{63} \sim 2^{63} (1-2^{-8})$
- C. $-2^{63} \sim 2^{64} (1-2^{-8})$
- D. $-2^{63} (1-2^{-8}) \sim 2^{63} (1-2^{-8})$

正常使用填空题需3.0以上版本雨课堂

作答

16位长的浮点数，其中阶码7位（含1位阶符），尾数9位（含1位数符），当浮点数采用原码表示时，表示的数的范围是 [D]，当浮点数采用补码表示时，数的表示的范围是 [B]

- A. $-2^{64} \sim 2^{64} (1-2^{-8})$
- B. $-2^{63} \sim 2^{63} (1-2^{-8})$
- C. $-2^{63} \sim 2^{64} (1-2^{-8})$
- D. $-2^{63} (1-2^{-8}) \sim 2^{63} (1-2^{-8})$

原码的阶码范围是 $-63 \sim 63$
原码的尾数范围 $-(1-2^{-8}) \sim (1-2^{-8})$
原码的表示范围很明显，选D

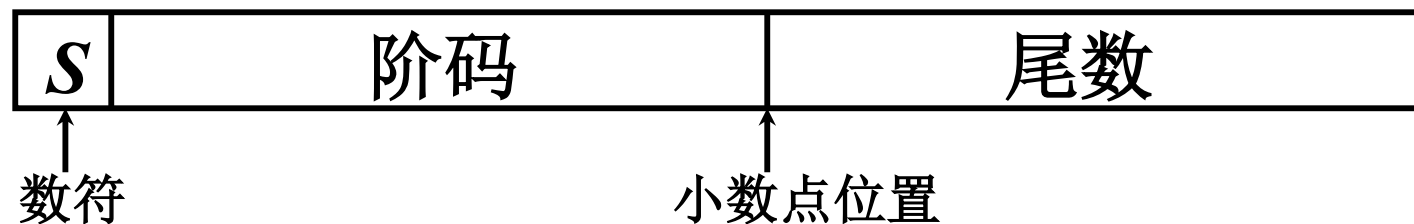
补码的尾数范围： $-1 \sim (1-2^{-8})$
补码的阶码范围是： $-64 \sim 63$

补码表示最小数的范围就是
阶码的最大 乘上 尾数的最小
所以选B

第二章 计算机中数的表示

- 计算机中数的表示
 - 无符号数和有符号数
 - 定点表示和浮点表示
 - IEEE754标准

IEEE 754 标准



尾数为规格化表示时:

非“0”的有效位最高位为“1”（隐含）

	符号位 S	阶码	尾数	总位数
短实数	1	8	23	32
长实数	1	11	52	64
临时实数	1	15	64	80

IEEE 754浮点数标准

- 单精度 (32-bit)

31	30	29	28	27	26	25	24	23	22 ~ 0												
S	8位指数 (无符号数)								23位尾数 (无符号数)												

- 双精度 (64-bit)

63	62	61	60	59	58	57	56	55	54	53	52	51~0									
S	11位指数 (无符号数)											52位尾数 (无符号数)									

	符号位 <i>S</i>	阶码	尾数	总位数
短实数	1	8	23	32
长实数	1	11	52	64
临时实数	1	15	64	80

IEEE 754浮点数：单精度为例



指数	尾数	表示对象	换算方法
0	0	0	规定（符号位不同，存在+0.0和-0.0）
0	非0	正负非规格化数	正负非规格化数 = $(-1)^S * (\text{尾数}_2) * 2^{(0 - 126)}$ (S代表符号位，1为负数，0为正数)
[1: 254]	任意	正负浮点数	正负浮点数 = $(-1)^S * (1 + \text{尾数}_2) * 2^{(\text{指数} - 127)}$
255	0	正负无穷 (inf)	规定
255	非零	NaN	规定

IEEE 754浮点数：正负浮点数

- 正负浮点数 = $(-1)^S \times (1 + \text{尾数}_2) \times 2^{(\text{指数} - 127)}$
- 尾数前加一？
 - 因为规格化二进制数，**小数点前要求是1**，这个1称为**前导数**。为了打包更多的位到数中，就在二进制表示中省略了前导数，默认小数点前有1。
 - 有效位数**：隐含的1加上尾数共有多少位。对单精度来说，有效位数是 24 位（隐含的1和 23 位尾数）；对双精度来说，是 53 位（1 + 52）。
 - 由于**尾数0和非规格化数**没有前导数，所以被赋予特殊的指数 0，硬件不会给它附加 1。

指数	尾数	表示对象	换算方法
0	0	0	规定
0	非0	正负 非 规格化数	正负非规格化数 = $(-1)^S * (\text{尾数}_2) * 2^{(0 - 126)}$ (S代表符号位, 1为负数, 0为正数)
[1: 254]	任意	正负浮点数	正负浮点数 = $(-1)^S * (1 + \text{尾数}_2) * 2^{(\text{指数} - 127)}$
255	0	正负无穷 (inf)	规定
255	非零	NaN	规定

IEEE 754浮点数：正负浮点数

- 正负浮点数 = $(-1)^S \times (1 + \text{尾数}_2) \times 2^{(\text{指数} - 127)}$
- 指数 - 127?
 - 使用移码的思想（偏移值是127）。二进制表示中的指数部分是移码，可以直接进行大小比较。如果两个数的符号相同，那么具有更大二进制指数的数就更大。
 - 对于真值而言，其实际的“指数”范围： $[1-127: 254-127] = [-126: 127]$

指数	尾数	表示对象	换算方法
0	0	0	规定
0	非0	正负非规格化数	正负非规格化数 = $(-1)^S * (\text{尾数}_2) * 2^{(0 - 126)}$ (S代表符号位, 1为负数, 0为正数)
[1: 254]	任意	正负浮点数	正负浮点数 = $(-1)^S * (1 + \text{尾数}_2) * 2^{(\text{指数} - 127)}$
255	0	正负无穷 (inf)	规定
255	非零	NaN	规定

IEEE 754浮点数：正负非规格化数

- 正负非规格化数 = $(-1)^S \times (\text{尾数}_2) \times 2^{(0 - 126)}$
- 什么是非规格化数？
 - 规格化数：科学计数法中整数部分没有前导 0 的数称为规格化数；
 - 非规格化数：整数部分前导为 0 的数
- 非规格化数的绝对值比浮点数绝对值更小
 - 对于正负浮点数来说，若二进制指数部分为1，则真值指数部分为 -126，和非规格化数相同。但浮点数尾数有前导1，导致浮点数绝对值更大。

指数	尾数	表示对象	换算方法
0	0	0	规定
0	非0	正负非规格化数	正负非规格化数 = $(-1)^S * (\text{尾数}_2) * 2^{(0 - 126)}$ (S代表符号位, 1为负数, 0为正数)
[1: 254]	任意	正负浮点数	正负浮点数 = $(-1)^S * (1 + \text{尾数}_2) * 2^{(\text{指数} - 127)}$
255	0	正负无穷 (inf)	规定
255	非零	NaN	规定

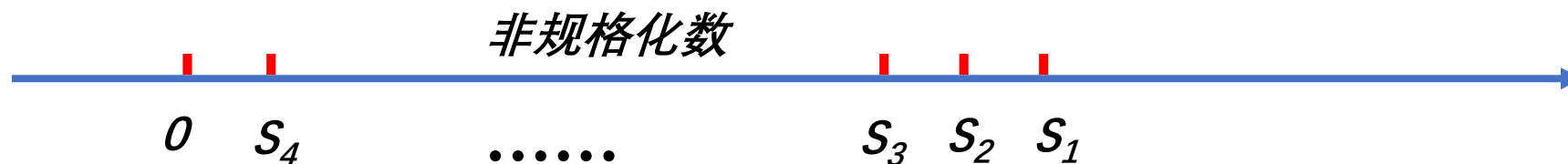
IEEE 754浮点数：正负非规格化数

- 正负浮点数 = $(-1)^S \times (1 + \text{尾数}_2) \times 2^{(\text{指数} - 127)}$
- 正负非规格化数 = $(-1)^S \times (\text{尾数}_2) \times 2^{(0 - 126)}$

尾数23位

最小正浮点数： $S_2 = (1 + 0_2) * 2^{(1 - 127)} = 2^{-126}$

第二小正浮点数： $S_1 = (1 + 0.0...01_2) * 2^{(1-127)} = 2^{-126} + \underline{2^{-149}}$



最大非规格化数： $S_3 = 0.1...11_2 * 2^{(0 - 126)} = (1 - 2^{-23}) * 2^{-126} = 2^{-126} - 2^{-149}$

最小非规格化正数： $S_4 = 0.0...01_2 * 2^{(0 - 126)} = 2^{-23} * 2^{-126} = \underline{2^{-149}}$

IEEE 754浮点数：真值转二进制

- 例题

- 将十进制 -0.75 转为单精度 IEEE 754格式二进制

- **解：** 根据十进制小数转二进制小数算法： $-0.75_{10} = -0.11_2$

规格化： $-0.11 = -1.1 * 2^{-1}$ ，能够规格化，说明是正负浮点数表示

$$-1.1 * 2^{-1} = (-1)^S \times (1 + \text{尾数}_2) \times 2^{(\text{指数} - 127)}$$

$$= (-1)^1 \times (1 + 0.1_2) \times 2^{(126 - 127)}$$

符号位：1；指数部分：126；尾数部分：0.1₂

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	0	1	1	1	1	1	1	0	1	0	0	0	0	0	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

IEEE754相关网址：<https://www.h-schmidt.net/FloatConverter/IEEE754.html>

IEEE 754浮点数：真值转二进制

IEEE754相关网址: <https://www.h-schmidt.net/FloatConverter/IEEE754.html>

Tools & Thoughts

IEEE-754 Floating Point Converter

Translations: [de](#)

This page allows you to convert between the decimal representation of numbers (like "1.02") and the binary format used by all modern CPUs (IEEE 754 floating point).

IEEE 754 Converter (JavaScript), V0.22

	Sign	Exponent	Mantissa
Value:	-1	2^{-1}	1.5
Encoded as:	1	126	4194304
Binary:	<input checked="" type="checkbox"/>	<input type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input type="checkbox"/>	<input checked="" type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
You entered	<input type="text" value="-0.75"/>		
Value actually stored in float:	<input type="text" value="-0.75"/> <div>+1</div>		
Error due to conversion:	<input type="text" value="0.00"/> <div>-1</div>		
Binary Representation	10111111010000000000000000000000		
Hexadecimal Representation	0xbf400000		

Update

There has been an update in the way the number is displayed. Previous version would give you the represented value as a possibly rounded decimal number and the same number with the increased pre float. Now the original number is shown (either as the number that was entered, or as a possibly rounded decimal string) as well as the actual full precision decimal number that the float value is representing. A nice example to see this behaviour. The difference between both values is shown as well, so you can easier tell the difference between what you entered and what you get in IEEE-754.

IEEE 754浮点数：二进制转真值

• 例题

- 将二进制IEEE754浮点数表示转换为十进制浮点数（空白为0）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
1	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

• 解

符号位为1，指数字段为129，尾数字段为 $2^{-2} = 0.25$ 。是浮点数

$$\begin{aligned}(-1)^S \times (1 + \text{尾数}_2) \times 2^{(\text{指数} - 127)} &= (-1)^1 \times (1 + 0.25) \times 2^{(129 - 127)} \\&= -1 \times 1.25 \times 2^2 \\&= -5.0\end{aligned}$$

0	非0	正负非规格化数	正负非规格化数 = $(-1)^S \times (\text{尾数}_2) \times 2^{(0 - 126)}$ (S代表符号位, 1为负数, 0为正数)
[1: 254]	任意	正负浮点数	正负浮点数 = $(-1)^S \times (1 + \text{尾数}_2) \times 2^{(\text{指数} - 127)}$

Translations: [de](#)

This page allows you to convert between the decimal representation of numbers (like "1.02") and the binary format used by all modern CPUs (IEEE 754 floating point).