



中山大學 软件工程学院
SUN YAT-SEN UNIVERSITY SCHOOL OF SOFTWARE ENGINEERING

计算机组成原理

授课老师：吴炜滨

大纲



- 进制转换
- 数据表示
- 无符号数
- 有符号数
- 数的定点表示和浮点表示

大纲



➤ 进制转换

进制转换

■ 书写

- 十进制: $(123)_{+}$
- 二进制: 1001B 或 $(1001)_{-}$
- 十六进制: 17DBH 或 $(17\text{DB})_{+六}$

十进制数	二进制数	十六进制数	十进制数	二进制数	十六进制数
0	0 0 0 0 0	0	16	1 0 0 0 0	10
1	0 0 0 0 1	1	17	1 0 0 0 1	11
2	0 0 0 1 0	2	18	1 0 0 1 0	12
3	0 0 0 1 1	3	19	1 0 0 1 1	13
4	0 0 1 0 0	4	20	1 0 1 0 0	14
5	0 0 1 0 1	5	21	1 0 1 0 1	15
6	0 0 1 1 0	6	22	1 0 1 1 0	16
7	0 0 1 1 1	7	23	1 0 1 1 1	17
8	0 1 0 0 0	8	24	1 1 0 0 0	18
9	0 1 0 0 1	9	25	1 1 0 0 1	19
10	0 1 0 1 0	A	26	1 1 0 1 0	1A
11	0 1 0 1 1	B	27	1 1 0 1 1	1B
12	0 1 1 0 0	C	28	1 1 1 0 0	1C
13	0 1 1 0 1	D	29	1 1 1 0 1	1D
14	0 1 1 1 0	E	30	1 1 1 1 0	1E
15	0 1 1 1 1	F	31	1 1 1 1 1	1F

■ 任意一个数N可用下式表示

- r : 基值; n 、 m : 正整数, 分别代表整数位和小数位的位数
- d_i 为系数, 代表第 i 位的一个数码, 可以是 $0 \sim (r - 1)$ 数码中的任意一个
- r^i 为第 i 位的权数

$$\begin{aligned} N &= (d_{n-1}d_{n-2} \cdots d_1d_0.d_{-1}d_{-2} \cdots d_{-m})_r \\ &= d_{n-1}r^{n-1} + d_{n-2}r^{n-2} + \cdots + d_1r^1 + d_0r^0 + d_{-1}r^{-1} + \cdots + d_{-m}r^{-m} \end{aligned}$$

$$= \sum_{i=-m}^{n-1} d_i r^i$$

进制转换



■ 二进制数转为十进制数

- 按“权”展开法

- $(11011.1)_{\text{二}} = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} = (27.5)_{\text{十}}$

■ 十进制数转为二进制数

- 重复相除（乘）法

进制转换



■ 11.6 转二进制

$$(11.6)_{+} \approx (1011.1001)_{-}$$

整数部分除2取余数

$$11 \div 2 = 5 \dots\dots 1$$

$$5 \div 2 = 2 \dots\dots 1$$

$$2 \div 2 = 1 \dots\dots 0$$

$$1 \div 2 = 0 \dots\dots 1$$

高位

直到商为0止

1011

小数部分乘2取整数

$$0.6 * 2 = 1.2$$

$$0.2 * 2 = 0.4$$

$$0.4 * 2 = 0.8$$

$$0.8 * 2 = 1.6$$

高位

直到小数部分为0或满足精度要求的位数止

0.1001 1001...

大纲



➤ 数据表示

■ 数据表示

- 定义：能由计算机硬件直接识别和引用的数据类型，如定点数、浮点数等
- 在计算机系统中，有对这些数据类型进行操作的机器指令和功能部件
 - 这些数据类型及其运算需由硬件实现
 - 数据表示是所有数据类型中最常用、相对比较简单、用硬件实现比较容易的几种



➤ 无符号数

无符号数



- 无符号数：没有正负号的数
- 在计算机中如何表示和存储？
 - 只有数值部分，将其转变成二进制
 - 如放在寄存器中，寄存器的位数（机器字长）反映了无符号数的表示范围



8 位

0 (8个0) ~ 255 (8个1)



16 位

0 (16个0) ~ 65535 (16个1)

➤ 有符号数

- 机器数与真值
- 原码表示法
- 补码表示法
- 反码表示法
- 移码表示法



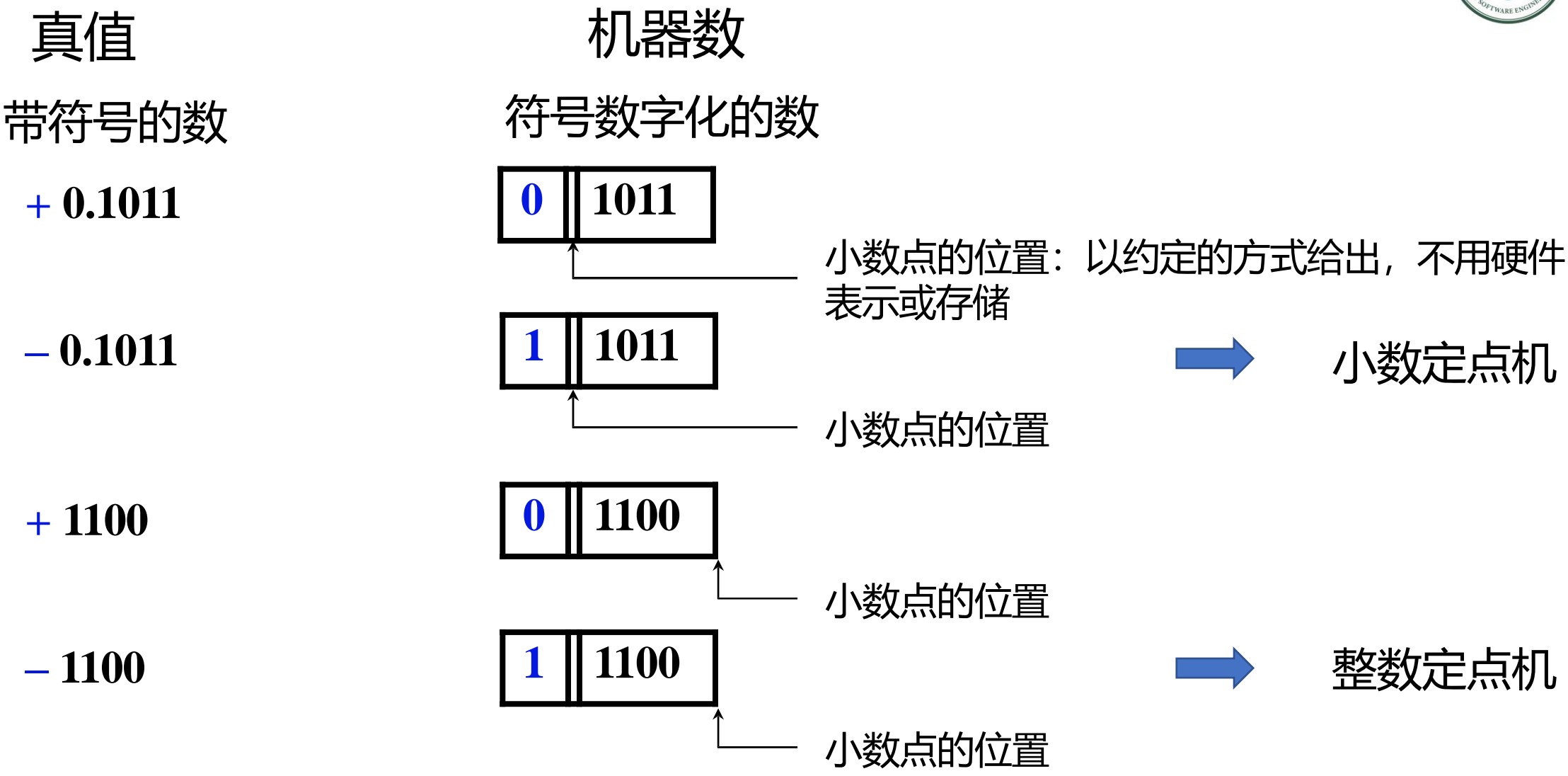
- 有符号数
 - 机器数与真值

机器数与真值



- 有符号数：有正负号的数
- 在计算机中如何表示和存储？
 - 数值部分、符号部分都保存到计算机中
- 机器数：保存在计算机当中的数
- 真值：带有正负号（ “+” ， “-” ） 的数
- 如何将真值转换为机器数？
 - 符号数字化

机器数与真值





- 有符号数
 - 原码表示法

原码定义



■ 整数

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

x 为真值

n 为整数数值位的位数

如 $x = +1110$ $[x]_{\text{原}} = 0, 1110$

用逗号将符号位和数值部分隔开，方便阅读，不用存储

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

原码：带符号的绝对值表示

原码定义



■ 小数

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

x 为真值

如 $x = +0.1101$ $[x]_{\text{原}} = 0.1101$ 用**小数点**将符号位和数值部分隔开, 方便阅读, 不需存储

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

$x = +0.1000000$ $[x]_{\text{原}} = 0.1000000$ 用**小数点**将符号位和数值部分隔开

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

■ 真值变成原码

- 整数：加一个符号位，正数加0负数加1，后面加一个逗号，数值部分照抄
- 小数：小数点前面的那一位就用于表示数据的符号，正数用0来表示，负数用1来表示，后面加一个小数点，然后把小数点后面数值部分照写

■ 原码是保存在计算机的数据

- 位数有限，受限于计算机能保存的机器数的长度

举例



■ 已知 $[x]_{\text{原}} = 1.0011$ 求 x $- 0.0011$

解: 由定义得

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

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

解: 由定义得

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

■ 已知 $[x]_{\text{原}} = 0.1101$ 求 x

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

$$\therefore x = +0.1101$$

■ 求 $x = 0$ 的原码

$$\text{解： 设 } x = +0.0000 \qquad [+0.0000]_{\text{原}} = 0.0000$$

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

$$\text{同理，对于整数} \qquad [+0]_{\text{原}} = 0,0000$$

$$[-0]_{\text{原}} = 1,0000$$

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

原码的特点



■ 简单、直观

- 但是用原码作加法时，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	正
加法	正	负	减	可正可负
加法	负	正	减	可正可负
加法	负	负	加	负

■ 能否只作加法？

- 找到一个与负数等价的正数，来代替这个负数，就可使减 → 加



- 有符号数
 - 补码表示法

补数



■ 时钟：六点调整到三点

逆时针

$$\begin{array}{r} 6 \\ -3 \\ \hline 3 \end{array}$$

顺时针

$$\begin{array}{r} 6 \\ +9 \\ \hline 15 \\ -12 \\ \hline 3 \end{array}$$

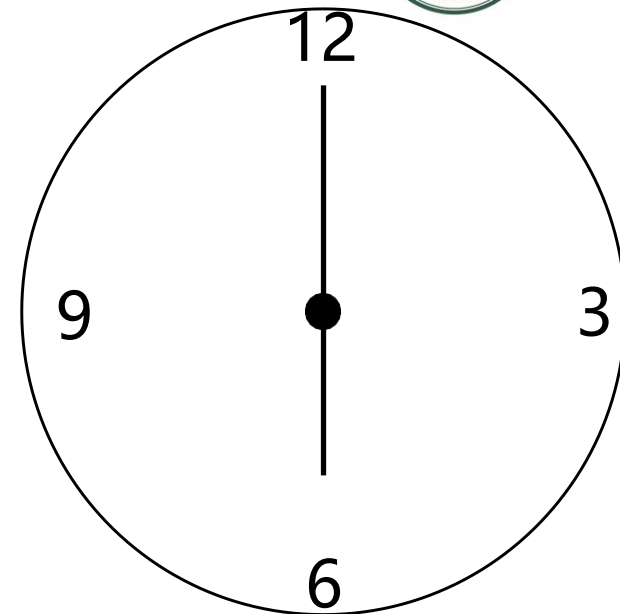
可见-3可用+9代替

减法 → 加法

称+9是-3以12为模的补数

记作 $-3 \equiv +9 \pmod{12}$

时钟以
12为模



补数



■ $-3 \equiv +9 \pmod{12}$

• 同理

• $-4 \equiv +8 \pmod{12}$

• $-5 \equiv +7 \pmod{12}$

• $3 \equiv 15 \equiv 27 \pmod{12}$

• 即: $3 \equiv 3 + 12 \equiv 3 + 24 \equiv 3 \pmod{12}$

■ 结论

- 一个负数加上 **模** 即得该负数的补数
- 一个正数和一个负数互为补数时它们绝对值之和即为 **模** 数
- 正数的补数即为其本身

机器中的补数



- 寄存器：位数是四位（模 16）

1011 \longrightarrow 0000 ?

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

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

自然去掉

可见 -1011 可用 $+ 0101$ 代替

记作 $-1011 \equiv + 0101 \pmod{2^4}$

同理 $-011 \equiv + 101 \pmod{2^3}$

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

机器数的补码表示



两个互为补数的数
分别加上模

结果仍互为补数

$$\begin{array}{rcl} -1011 & \equiv & +0101 \\ +10000 & & +10000 \\ \hline +0101 & \equiv & +10101 \end{array} \pmod{2^4}$$

$$\therefore +0101 \equiv +0101 \pmod{2^4}$$

丢掉

可见

$$\begin{array}{ccc} +0101 & \xrightarrow{\quad} & +0101 \\ & \xrightarrow{?} & -1011 \end{array}$$

?

$$0,0101 \longrightarrow +0101$$

?

$$1,0101 \longrightarrow -1011$$

$$2^{4+1} - 1011 = 100000$$

$$\begin{array}{r} -1011 \\ \hline 1,0101 \end{array}$$

用 逗号 将符号位和
数值部分隔开

$$\pmod{2^{4+1}}$$

补码定义



■ 整数

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

x 为真值

n 为整数数值位的位数

如

$$x = +1010$$

$$x = -1011000$$

$$[x]_{\text{补}} = 0,1010$$

用逗号将符号位
和数值部分隔开

$$\begin{aligned} [x]_{\text{补}} &= 2^{7+1} + (-1011000) \\ &= 100000000 \\ &\quad - 1011000 \\ \hline &1,0101000 \end{aligned}$$

补码定义



■ 小数

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

x 为真值

如

$$x = +0.1110$$

$$x = -0.1100000$$

$$[x]_{\text{补}} = 0.1110$$

$$[x]_{\text{补}} = 2 + (-0.1100000)$$

$$= 10.0000000$$

$$- 0.1100000$$

$$\hline 1.0100000$$

用小数点将符号位
和数值部分隔开

求补码的快捷方式



■ 设 $x = -1010$ 时

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

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

■ 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（**求反加1**）

求补码的快捷方式



■ 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（**求反加1**）

■ 当真值为**正**时

- 原码 = 补码

■ 已知 $[x]_{\text{补}} = 1,1110$ 求 x

解： 由定义得

$$\begin{aligned}x &= [x]_{\text{补}} - 2^{4+1} \\&= 1,1110 - 100000 \\&= -0010\end{aligned}$$

$$[x]_{\text{补}} \xrightarrow{?} [x]_{\text{原}}$$

$$[x]_{\text{原}} = 1,0010$$

$$\therefore x = -0010$$

■ 当真值为**负**时

- 原码可用补码除符号位外，每位取反，末位加 1 求得 (**求反加1**)

■ 已知 $[x]_{\text{补}} = 0.0001$ 求 x

解：由定义得 $x = +0.0001$

■ 已知 $[x]_{\text{补}} = 1.0001$ 求 x

解：由定义得

$$\begin{aligned} x &= [x]_{\text{补}} - 2 \\ &= 1.0001 - 10.0000 \\ &= -0.1111 \end{aligned}$$

$$\begin{aligned} [x]_{\text{补}} &\rightarrow [x]_{\text{原}} \\ [x]_{\text{原}} &= 1.1111 \\ \therefore x &= -0.1111 \end{aligned}$$

■ 求下列真值的补码

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

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

$$[-1]_{\text{补}} = 2 + x = 10.0000 - 1.0000 = 1.0000$$



- 有符号数
 - 反码表示法

反码定义



■ 整数

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

x 为真值 n 为整数数值位的位数

如 $x = +1101$

$$[x]_{\text{反}} = 0,1101$$

用逗号将符号位
和数值部分隔开

$x = -1101$

$$\begin{aligned} [x]_{\text{反}} &= (2^{4+1} - 1) - 1101 \\ &= 11111 - 1101 \\ &= 1,0010 \end{aligned}$$

反码定义



■ 小数

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

x 为真值

n 为小数的位数

如 $x = +0.1101$

$$x = -0.1010$$

$$[x]_{\text{反}} = 0.1101$$

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

$$= 1.1111 - 0.1010$$

$$= 1.0101$$

用小数点将符号位和
数值部分隔开



反码定义



■ 已知 $[x]_{\text{反}} = 0,1110$ 求 x

解：由定义得 $x = +1110$

■ 已知 $[x]_{\text{反}} = 1,1110$ 求 x

解：由定义得

$$\begin{aligned} x &= [x]_{\text{反}} - (2^{4+1} - 1) \\ &= 1,1110 - 11111 \\ &= -0001 \end{aligned}$$

反码定义



■ 求0的反码

解： 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

$$[+0]_{\text{反}} = 0,0000$$

$$[- 0]_{\text{反}} = 1,1111$$

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

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，原码 = 补码 = 反码（考虑机器字长的限制）
- 对于负数，符号位为 1，其数值部分
 - 原码：数值部分和真值相同
 - 补码：原码除符号位外每位取反末位加 1
 - 反码：原码除符号位外每位取反

■ 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

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

解: 设 $[y]_{\text{补}} = y_0.y_1 y_2 \dots y_n$

$$(1) \quad [y]_{\text{补}} = 0.y_1 y_2 \dots y_n$$

$$y = 0.y_1 y_2 \dots y_n$$

$$-y = -0.y_1 y_2 \dots y_n$$

$$[-y]_{\text{补}} = 1.\bar{y}_1 \bar{y}_2 \dots \bar{y}_n + 2^{-n}$$

$[y]_{\text{补}}$ 连同符号位在内, 每位取反, 末位加1

即得 $[-y]_{\text{补}}$

$$(2) \quad [y]_{\text{补}} = 1.y_1 y_2 \dots y_n$$

$$[y]_{\text{原}} = 1.\bar{y}_1 \bar{y}_2 \dots \bar{y}_n + 2^{-n}$$

$$y = -(0.\bar{y}_1 \bar{y}_2 \dots \bar{y}_n + 2^{-n})$$

$$-y = 0.\bar{y}_1 \bar{y}_2 \dots \bar{y}_n + 2^{-n}$$

$$[-y]_{\text{补}} = 0.\bar{y}_1 \bar{y}_2 \dots \bar{y}_n + 2^{-n}$$

$[y]_{\text{补}}$ 连同符号位在内, 每位取反, 末位加 1

即得 $[-y]_{\text{补}}$



- 有符号数
 - 移码表示法

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

$x + 2^5$

+10101 + 100000	= 1,10101	大	正确
-10101 + 100000	= 0,01011	大	正确
+11111 + 100000	= 1,11111	大	正确
-11111 + 100000	= 0,00001	大	正确

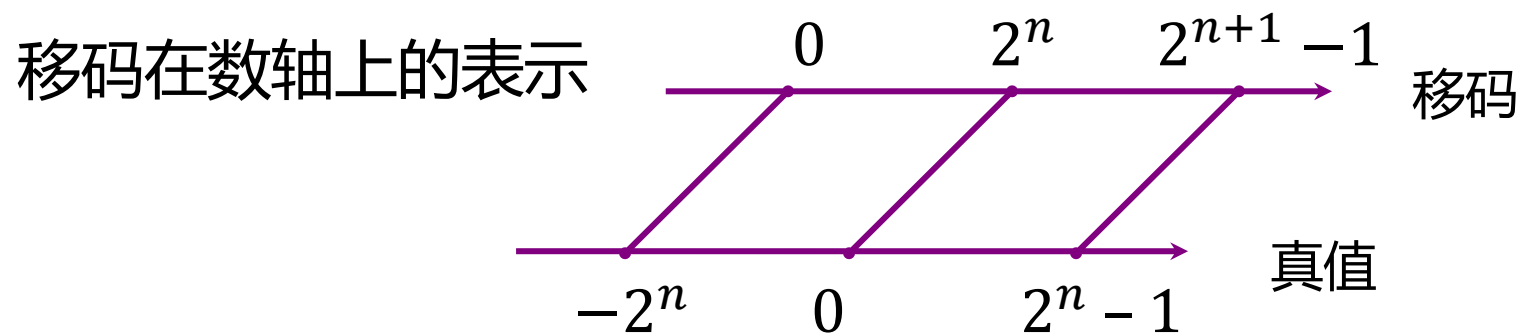
移码



■ 定义

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

x 为真值, n 为整数数值位的位数



如 $x = 10100$

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

$$x = -10100$$

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

用 逗号 将符号位
和数值部分隔开

■ 注意

- 真值转换成移码时，不区分对待正数和负数
- 只有整数形式的定义，没有小数形式的定义
 - 与移码在计算机的数据表示中的作用有关
 - 移码的大小很好判断，通常用来表示浮点数据表示的阶码部分，能方便地判断浮点数的阶码大小
 - 阶码都是整数

移码和补码的比较



■ 设 $x = +1100100$

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

$$[x]_{\text{补}} = 0,1100100$$

■ 设 $x = -1100100$

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

$$[x]_{\text{补}} = 1,0011100$$

■ 同一真值的补码与移码只差一个符号位，数值部分完全相同

真值、补码和移码的对照表



机器数数值部分
位数 $n=5$

真值 x	$[x]_{\text{补}}$	$[x]_{\text{移}}$	$[x]_{\text{移}}$ 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



- 当 $x = 0$, $n = 5$ 时

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

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

$$\therefore [+0]_{\text{移}} = [-0]_{\text{移}}$$

- 当 $n = 5$ 时

$$\text{最小的真值为 } -2^5 = -100000$$

$$[-100000]_{\text{移}} = 2^5 - 100000 = 000000$$

- 可见：最小真值的移码为全 0

➤ 数的定点表示和浮点表示

- 定点表示
- 浮点表示

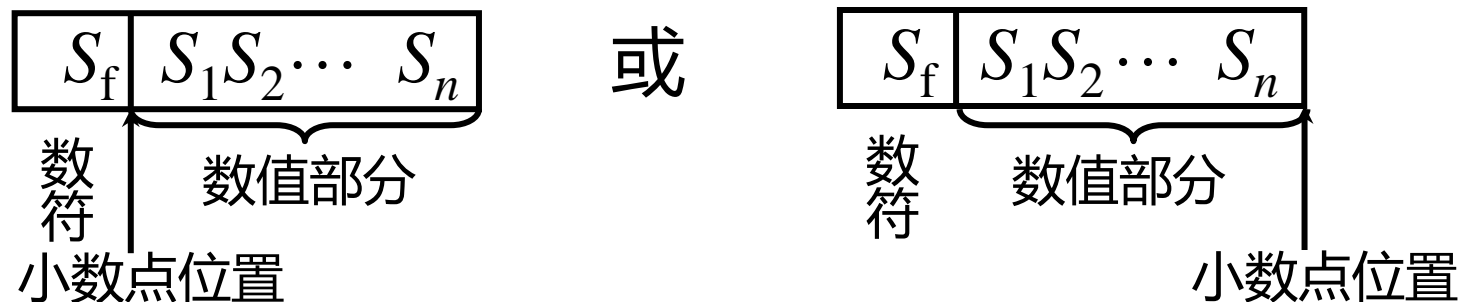


- 数的定点表示和浮点表示
 - 定点表示

定点表示



- 小数点按约定方式标出，计算机中不表示或存储小数点



	定点机	小数定点机	整数定点机
表示范围	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	$-(1 - 2^{-n}) \sim +(1 - 2^{-n})$	$-(2^n - 1) \sim +(2^n - 1)$

有限个 (2^{n+1}) 离散
的数表示无穷个
实数



- 数的定点表示和浮点表示
 - 浮点表示

■ 为什么在计算机中要引入浮点数表示？

- 早期计算机因为硬件技术的限制，只有定点表示方式
- 科学计算过程当中经常会用到浮点数，在定点机中，需要程序员通过设定比例因子来调节小数点的位置，使其变为纯小数或纯整数后进行运算
 - 增加编程难度

$$101.01 + 0.101$$

$$= (0.101010 + 0.000101) \times 2^3 \quad (\text{缩小})$$

$$= (0.101111) \times 2^3 \quad (\text{运算})$$

$$= 101.111 \quad (\text{还原})$$

■ 为什么在计算机中要引入浮点数表示？

- 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 2×10^{33} 克，一个电子的质量大约为 9×10^{-28} 克，两者相差约 10^{61} 倍，若在定点机中表示： $2^x > 10^{61}$ ，大概需要 $x \geq 203$ 位
- 数据存储单元的利用率往往很低

浮点表示



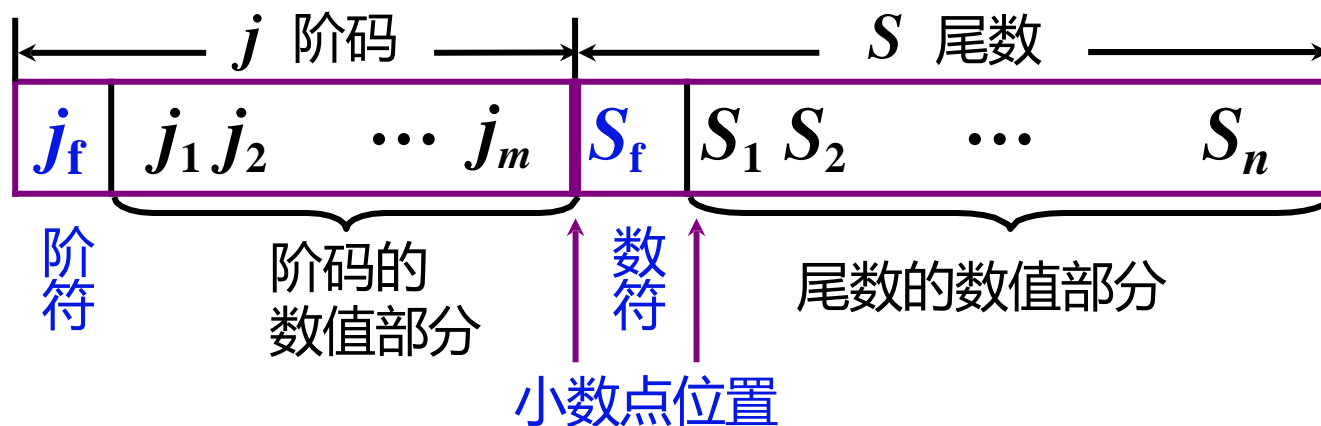
■ 浮点数的一般形式: $N = S \times r^j$

- S : 尾数, j : 阶码, r : 尾数的基值
- 计算机中: r 取 2、4、8、16 等; S : 小数定点形式表示, 绝对值小于等于1, 可正可负;
 j : 整数, 可正可负

■ 当 $r = 2$

$$\begin{aligned} N &= 11.0101 && \text{二进制表示} \\ \checkmark &= 0.110101 \times 2^{10} && \text{规格化数} \\ &= 1.10101 \times 2^1 \\ &= 1101.01 \times 2^{-10} \\ \checkmark &= 0.00110101 \times 2^{100} \end{aligned}$$

浮点数的存储和表示



S_f 代表浮点数的符号

n 其位数反映浮点数的精度

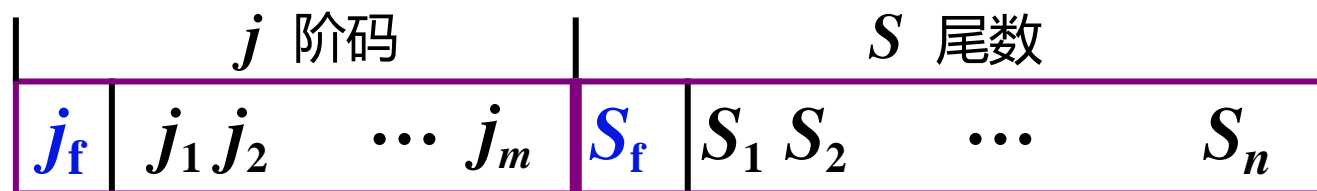
m 其位数反映浮点数的表示范围

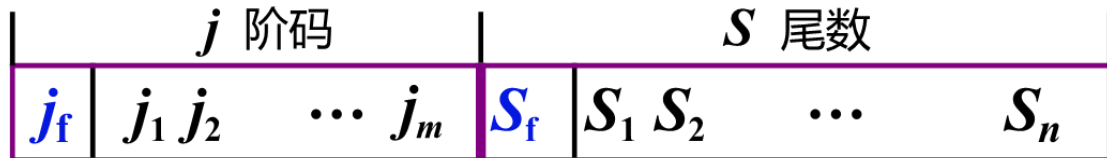
阶码 决定小数点的实际位置

浮点数的表示范围



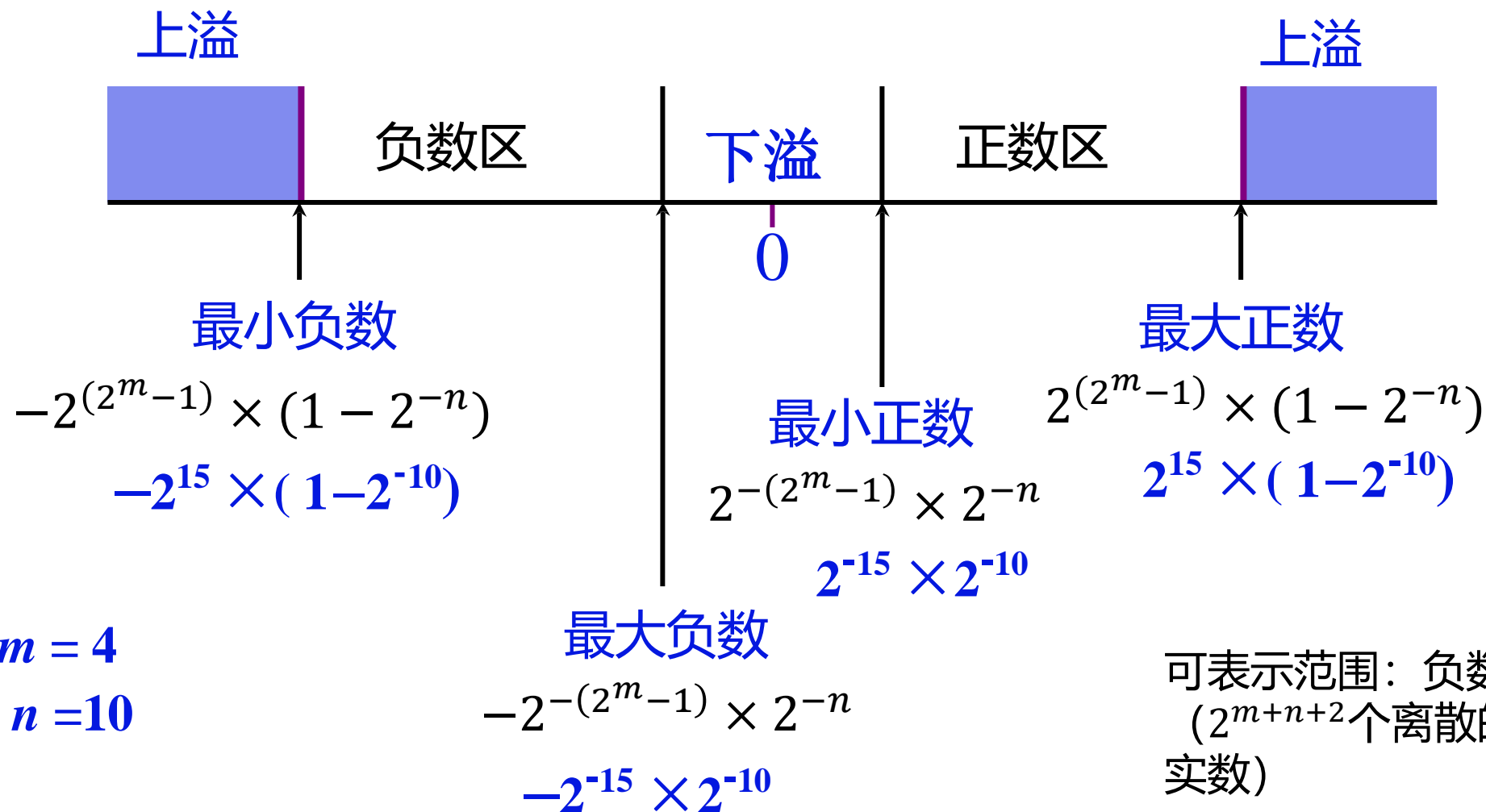
- 不考虑数据的规格化
- 无论是尾数还是阶码，都用原码形式进行表示
- 阶码的数值位取 m 位，尾数的数值位取 n 位





上溢：浮点数阶码大于最大阶码时，进行中断溢出处理

下溢：浮点数阶码小于最小阶码时，按机器零处理



设 $m = 4$
 $n = 10$

可表示范围：负数区+正数区+0
(2^{m+n+2} 个离散的数表示无穷个实数)

- 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，不论尾数为何值，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： $\because \quad 2^{14} = 16384 \quad \quad 2^{15} = 32768$

\therefore 如果是定点数，**15** 位二进制数可反映 ± 3 万之间的十进制数

$$\begin{array}{c} 2^{15} \times 0.\underbrace{\times \times \times \dots \times \times \times}_{n\text{位}} \\ \downarrow \\ m = 4, 5, 6, \dots \end{array}$$

为满足最大精度，可取 $m = 4, n = 18$

浮点数的规格化形式



■ 浮点数的一般形式: $N = S \times r^j$

- S : 尾数, j : 阶码, r : 尾数的基值
- 计算机中: r 取 2、4、8、16 等; S : 小数定点形式表示, 绝对值小于等于1, 可正可负;
 j : 整数, 可正可负

■ 为何要进行规格化?

- 尽可能保证数据的精度: 机器字长有限, 避免存储不必要的0, 使有效的位数尽可能多

$$N = 11.0101$$

$$\checkmark = 0.110101 \times 2^{10} \quad \text{规格化数}$$

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

浮点数的规格化形式



■ 规格化形式是什么？

- $r = 2$, 尾数最高位真值为 1
- $r = 4$, 尾数最高 2 位真值不全为 0
- $r = 8$, 尾数最高 3 位真值不全为 0

■ 基值的影响

- 基值不同，浮点数的规格化形式不同
- 基值越大，在同样浮点数表示形式下，可表示的浮点数的范围越大，浮点数的精度降低

浮点数的规格化



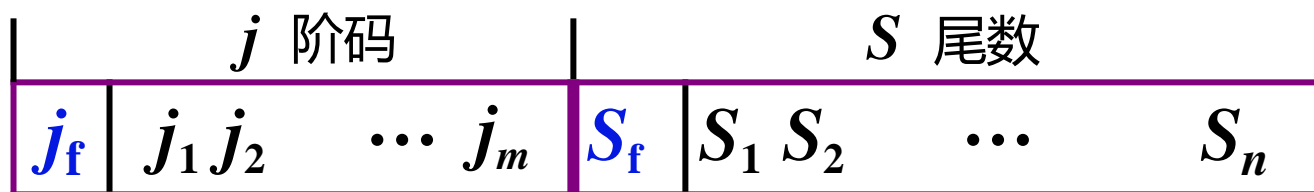
■ 如何进行规格化?

- 通过对尾数进行移动, 使其变为规格化形式
- $r = 2$
 - 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

浮点数的表示范围



- 无论是尾数还是阶码，都用原码形式进行表示
- 阶码的数值位取 m 位，尾数的数值位取 n 位
 - 设 $m = 4$, $n = 10$, $r = 2$, 尾数规格化后的浮点数表示范围



■ 设 $m = 4$, $n = 10$, $r = 2$, 尾数规格化后的浮点数表示范围

最大正数 $2^{+1111} \times 0.\underbrace{1111111111}_{10 \text{ 个 } 1} = 2^{15} \times (1 - 2^{-10})$

最小正数 $2^{-1111} \times 0.1\underbrace{0000000000}_{9 \text{ 个 } 0} = 2^{-15} \times 2^{-1} = 2^{-16}$

最大负数 $2^{-1111} \times (-0.\underbrace{1000000000}_{9 \text{ 个 } 0}) = -2^{-15} \times 2^{-1} = -2^{-16}$

最小负数 $2^{+1111} \times (-0.\underbrace{1111111111}_{10 \text{ 个 } 1}) = -2^{15} \times (1 - 2^{-10})$

- 将 $+\frac{19}{128}$ 写成在定点机和浮点机中的三种机器数形式。其中定点数和浮点数尾数数值部分均取10位，数符取1位，浮点数阶码取5位（含1位阶符），尾数规格化

解：设 $x = \frac{19}{128}$

二进制形式

$$x = 10011/10000000 = 0.0010011$$

定点表示

$$x = 0.0010011$$

浮点规格化形式

$$x = 0.10011 \times 2^{-10}$$

定点机中

$$[x]_{\text{原}} = [x]_{\text{补}} = [x]_{\text{反}} = 0.0010011\textcolor{red}{000}$$

浮点机中

$$[x]_{\text{原}} = 1, \textcolor{red}{00}10; \quad 0.10011\textcolor{red}{00000}$$

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

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

- 将 -58 写成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。其中定点数和浮点数尾数数值部分均取10位，数符取1位，浮点数阶码取5位（含1位阶符），尾数规格化

解：设 $x = -58$

二进制形式

$$x = -111010$$

定点表示

$$x = -111010$$

浮点规格化形式

$$x = -0.11101 \times 2^{110}$$

定点机中

$$[x]_{\text{原}} = 1, \textcolor{red}{0000}111010$$

$$[x]_{\text{补}} = 1, 1111000110$$

$$[x]_{\text{反}} = 1, 1111000101$$

浮点机中

$$[x]_{\text{原}} = 0, \textcolor{red}{0110}; 1.11101\textcolor{red}{00000}$$

$$[x]_{\text{补}} = 0, 0110; 1.0001100000$$

$$[x]_{\text{反}} = 0, 0110; 1.0001011111$$

$$[x]_{\text{阶移、尾补}} = 1, 0110; 1.0001100000$$



谢谢！