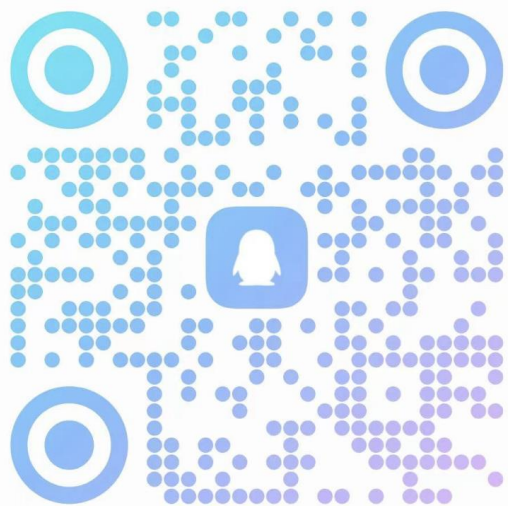


计算机组成原理

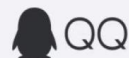


2025春计算机...

群号: 892932264



扫一扫二维码，入群聊



花忠云

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

<http://faculty.hitsz.edu.cn/huazhongyun>

计算机科学与技术学院

第二章 计算机的运算方法

2.1 计算机中数的表示

2.2 定点运算

2.3 浮点运算

2.1 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

- IEEE 754标准

- 算数移位与逻辑移位

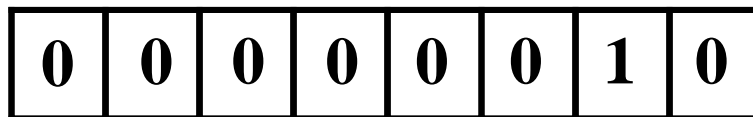
无符号数

- 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

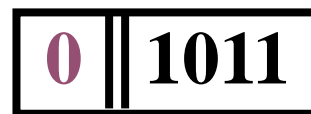
+ 0.1011或0.1011

- 0.1011

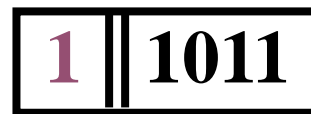
+ 1100或1100

- 1100

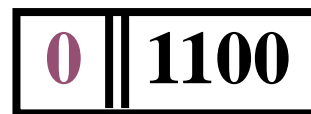
机器数：符号数字化的数



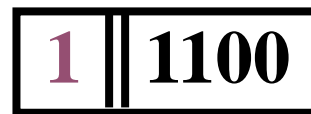
小数点的位置



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。

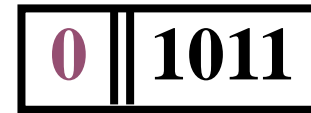
有符号数：真值与机器数

真值：带符号的数

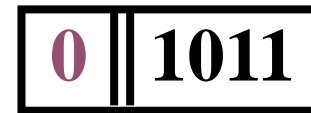
+ 0.1011

+ 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

$x = +1110$ $[x]_{\text{原}} = 0, 1110$ 用逗号将符号位和数值部分隔开

$x = -1110$ $[x]_{\text{原}} = 1, 1110$

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

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

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

$x = -0.0000$ 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

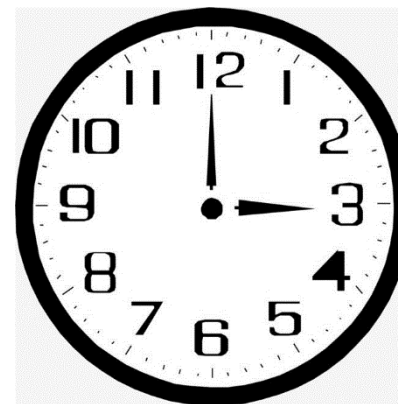
十进制		原码									
	3		<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table>	0	0	0	0	0	0	1	1
0	0	0	0	0	0	1	1				
+	-3	+	<table border="1"><tr><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table>	1	0	0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



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 → 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

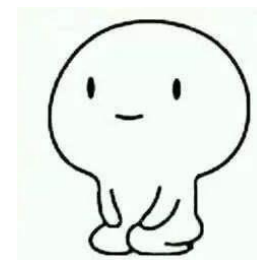
记作 $-1011 \equiv +0101 \pmod{2^4}$

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： $-1011 \equiv -1011 + 2^4 \equiv +0101 \pmod{2^4}$

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

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

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

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：

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

$$x = + 0.1110$$

$$[x]_{\text{补}} = 0.1110$$

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：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：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：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 $[y]_{\text{补}} = y_0.y_1y_2 \cdots y_n$ ，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 $[-y]_{\text{补}}$

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
		$-10101 + 100000 = 001011$		正确
		$+11111 + 100000 = 111111$		大
		$-11111 + 100000 = 000001$		正确

移码表示法：二进制整数

- 移码定义

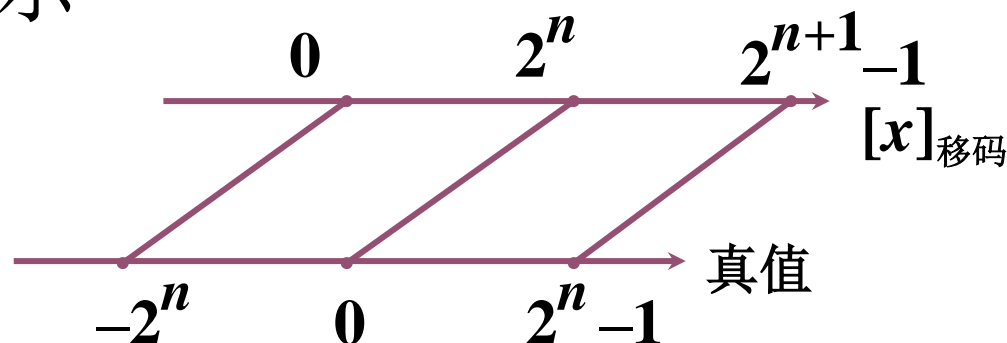
$$[x]_{\text{移}} = 2^n + x \quad (2^n > x \geq -2^n)$$

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

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	1 0 0 0 0	0 0 0 0 0	0
- 1 1 1 1 1	1 0 0 0 1	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

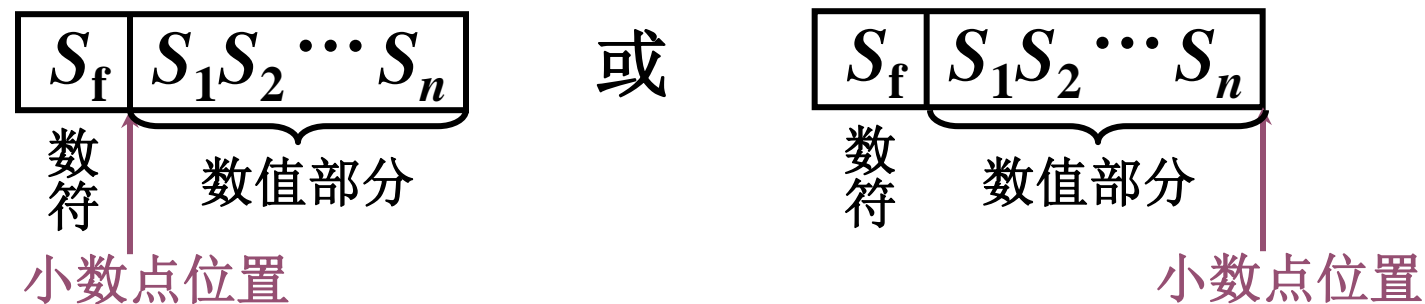
可用移码思想表示浮点数的阶码, 便于判断浮点数阶码大小

2.1 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
 - IEEE 754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

$N = S \times r^j$ 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

✓ $= 0.110101 \times 2^{10}$ 规格化数

$= 1.10101 \times 2^1$

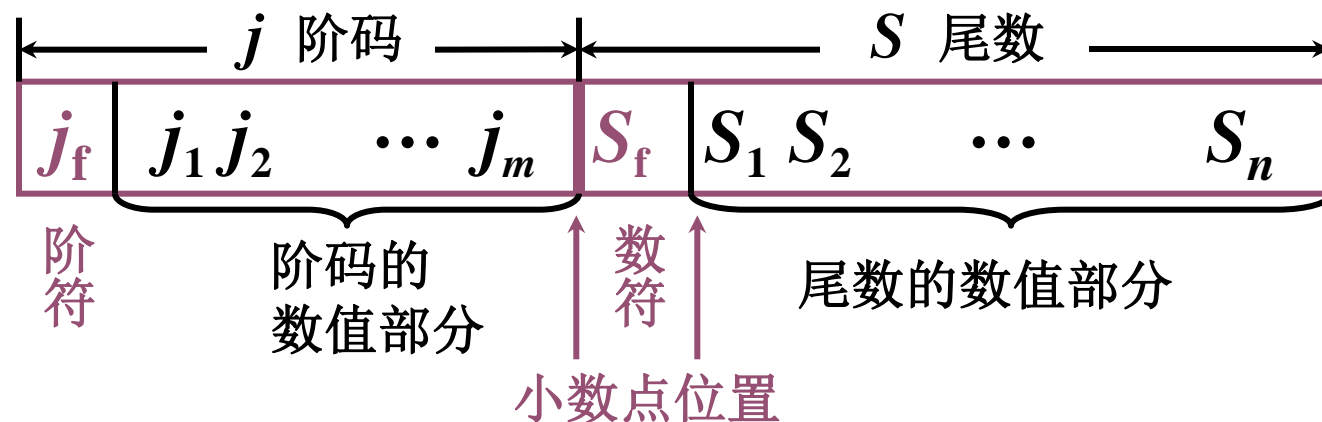
$= 1101.01 \times 2^{-10}$

✓ $= 0.00110101 \times 2^{100}$

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



S_f 代表浮点数的符号

n 其位数反映浮点数的精度

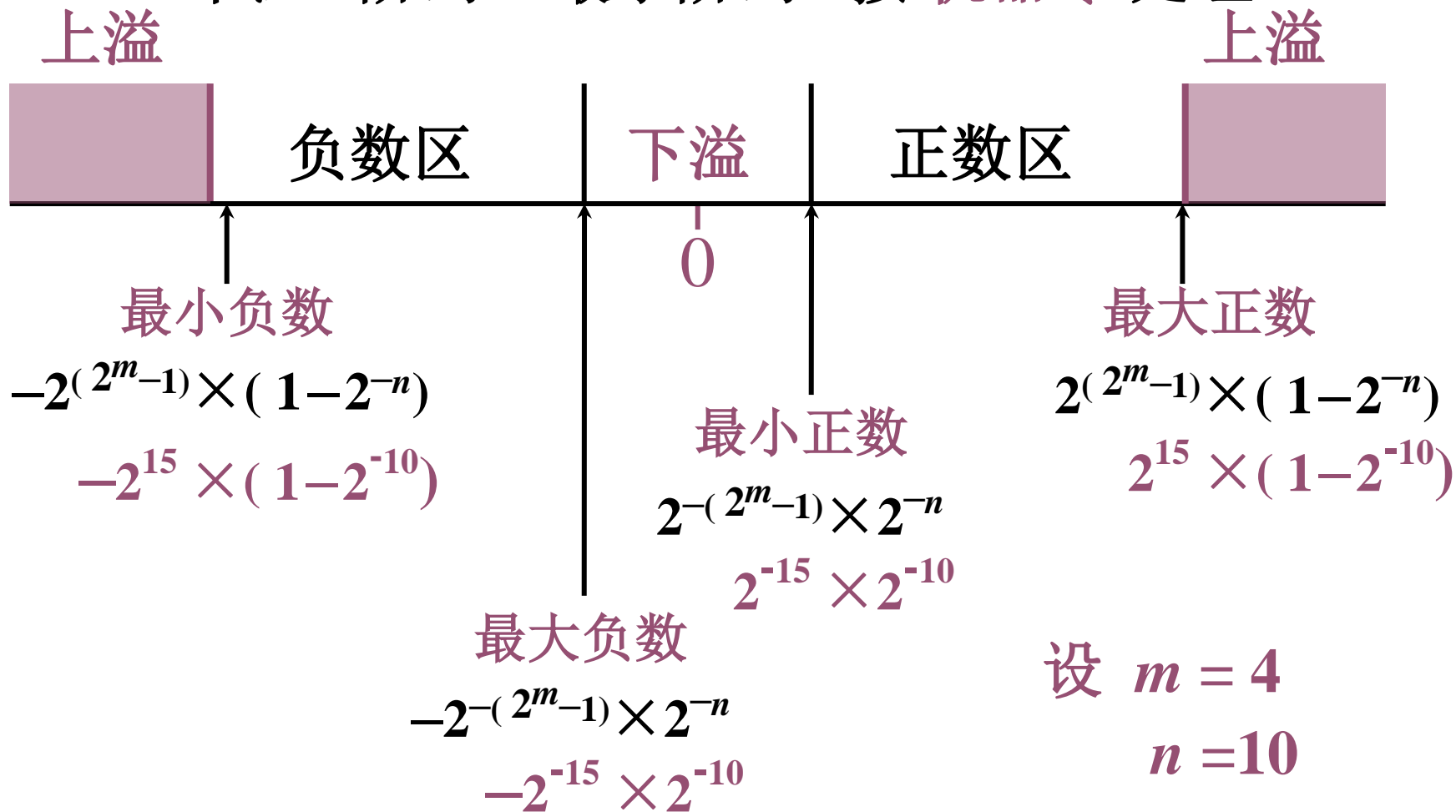
m 其位数反映浮点数的表示范围

j_f 和 m 共同表示小数点的实际位置

浮点数的表示范围

上溢 阶码 > 最大阶码

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

基数 r 越大，可表示的浮点数的范围越大

基数 r 越大，浮点数的精度降低

- 规格化数的定义

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

- 规格化数的判断

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

原码 不论正数、负数，第一数位为1

补码和反码 符号位和第一数位不同

- 特例

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

$$[S]_{\text{原}} = 1.100 \dots 0$$

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

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

$$S = -1$$

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

$\therefore [-1]_{\text{补}}$ 是规格化的数

- 例5. 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6. 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 $[x]_{\text{原}} = 1, 0010; 0. 1001100000$

$[x]_{\text{补}} = 1, 1110; 0. 1001100000$

$[x]_{\text{反}} = 1, 1101; 0. 1001100000$

- **例7.** 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（其他要求同上例）。

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

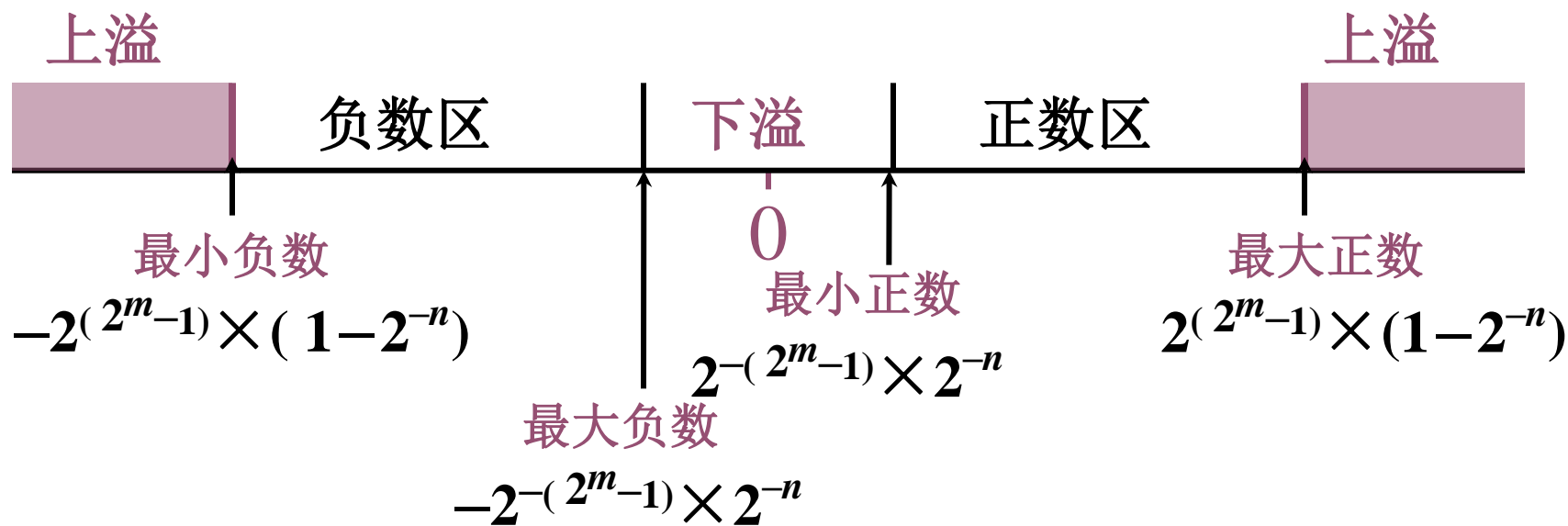
$[x]_{\text{原}} = 0, 0110; 1. 1110100000$

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

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

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

- 例8. 写出下列原码数轴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:

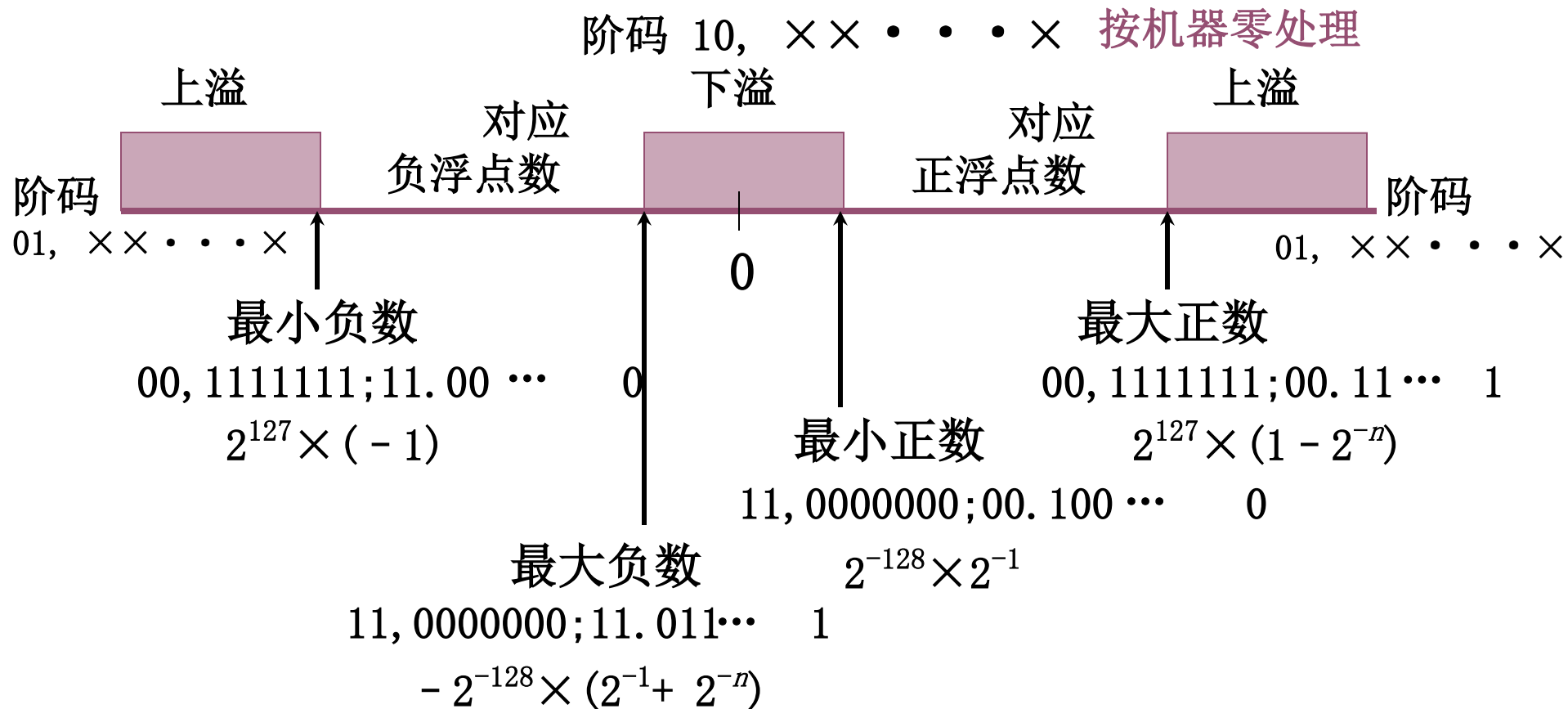
真值

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

溢出判断

设机器数为补码，尾数为 **规格化形式**，并假设阶符取 2 位，阶码取 7 位，数符取 2 位，尾数取 n 位，则该 **补码** 在数轴上的表示为



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

$\times, \times \times \times \times; \quad 0.00 \dots 0$

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

$0, 0000; \quad 0.00 \dots 0$

有利于机器中“判 0”电路的实现

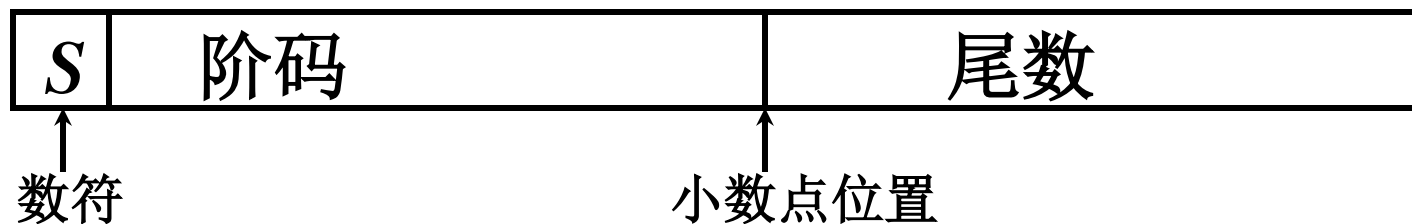
16位长的浮点数，其中阶码7位（含1位阶符），尾数9位（含1位数符），当浮点数采用原码表示时，表示的数的范围是 [填空1]，当浮点数采用补码表示时，数的表示的范围是 [填空2]

- A. $-2^{64} \sim 2^{64} (1-2^{-8})$
- B. $-2^{63} \sim 2^{63} (1-2^{-8})$
- C. $-2^{63} \sim 2^{63} (1-2^{-9})$
- D. $-2^{63} (1-2^{-8}) \sim 2^{63} (1-2^{-8})$

2.1 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
 - **IEEE 754标准**
 - 算数移位与逻辑移位

IEEE 754 标准



尾数为规格化表示

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）											

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 = $(-1)^s * (\text{尾数}_2) * 2^{-126}$ (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？
 - 因为规格化的二进制数小数点后是1。为了打包更多的位到数中，就在二进制表示中把这个1移到小数点前面并省略了这个数。这个1称为**前导数**。
 - 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 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 = $(-1)^S * (\text{尾数}_2) * 2^{-126}$ (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**。二进制表示中的指数部分是无符号数，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，实际“指数”范围： $[1-127: 254-127] = [-126: 127]$
 - 如果-128，实际“指数”范围： $[1-128: 254-128] = [-127: 126]$
- 相邻两个单精度浮点数相差？

IEEE 754浮点数：正负非规格化数

- 正负非规格化数 = $(-1)^S \times (\text{尾数}_2) \times 2^{-126}$
- 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 = $(-1)^S * (\text{尾数}_2) * 2^{-126}$ (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 = $(-1)^S \times (1 + \text{尾数}_2) \times 2^{(\text{指数} - 127)}$ (指数：1—254)
- 正负非规格化数 = $(-1)^S \times (\text{尾数}_2) \times 2^{(0 - 126)}$

最小正浮点数： $N_1 = (1 + 0_2) * 2^{(1 - 127)} = 2^{-126}$

第二小正浮点数： $N_2 = (1 + 0.0...01_2) * 2^{(1-127)} = 2^{-126} + \underline{2^{-149}}$



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

.....

最小正非规格化数： $UN_1 = 0.0...01_2 * 2^{(0 - 126)} = 2^{-23} * 2^{-126} = \underline{2^{-149}}$ ₅₆

IEEE 754浮点数：真值转二进制

- 例9

- 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

IEEE 754浮点数：二进制转真值

- 例10
- 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

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

IEEE 754浮点数转换工具

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).

[illegible]

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

2.1 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
 - IEEE 754标准
 - 算数移位与逻辑移位

移位运算

- 移位的意义

$$15\text{ m} = 1500\text{ cm}$$

小数点右移 2 位

机器用语 15 相对于小数点 左移 2 位
(小数点不动)

左移 绝对值扩大

右移 绝对值缩小

- 在计算机中，移位与加减配合，能够实现乘除运算

算术移位规则

符号位不变

真值	码 制	添补代码
正数	原码、补码、反码	0
负数	原 码	0
	补 码	左移 添 0
		右移 添 1
	反 码	1

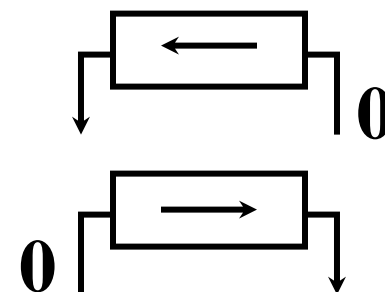
算术移位和逻辑移位的区别

算术移位 有符号数的移位

逻辑移位 无符号数的移位

逻辑左移 低位添 0，高位移丢

逻辑右移 高位添 0，低位移丢



例如 01010011

10110010

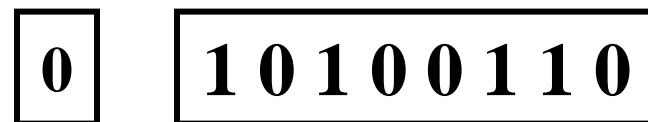
逻辑左移 10100110

逻辑右移 01011001

算术左移 00100110

算术右移 11011001 (补码)

高位 1 移丢



- 例11. 设机器数字长为 8 位（含 1 位符号位），写出 $A = +26$ 时，三种机器数左、右移一位和两位后的表示形式及对应的真值，并分析结果的正确性。

解： $A = +26 = +11010$

则 $[A]_{\text{原}} = [A]_{\text{补}} = [A]_{\text{反}} = 0,0011010$

移位操作	机 器 数	对应的真值
	$[A]_{\text{原}} = [A]_{\text{补}} = [A]_{\text{反}}$	
移位前	0,0011010	+26
左移一位	0,0110100	+52
左移两位	0,1101000	+104
右移一位	0,0001101	+13
右移两位	0,0000110	+6

- 例12. 设机器数字长为 8 位（含 1 位符号位），写出 $A = -26$ 时，三种机器数左、右移一位和两位后的表示形式及对应的真值，并分析结果的正确性。

解： $A = -26 = -11010$

原码	移位操作	机 器 数	对应的真值
	移位前	1,0011010	- 26
	左移一位	1,0110100	- 52
	左移两位	1,1101000	- 104
	右移一位	1,0001101	- 13
	右移两位	1,0000110	- 6

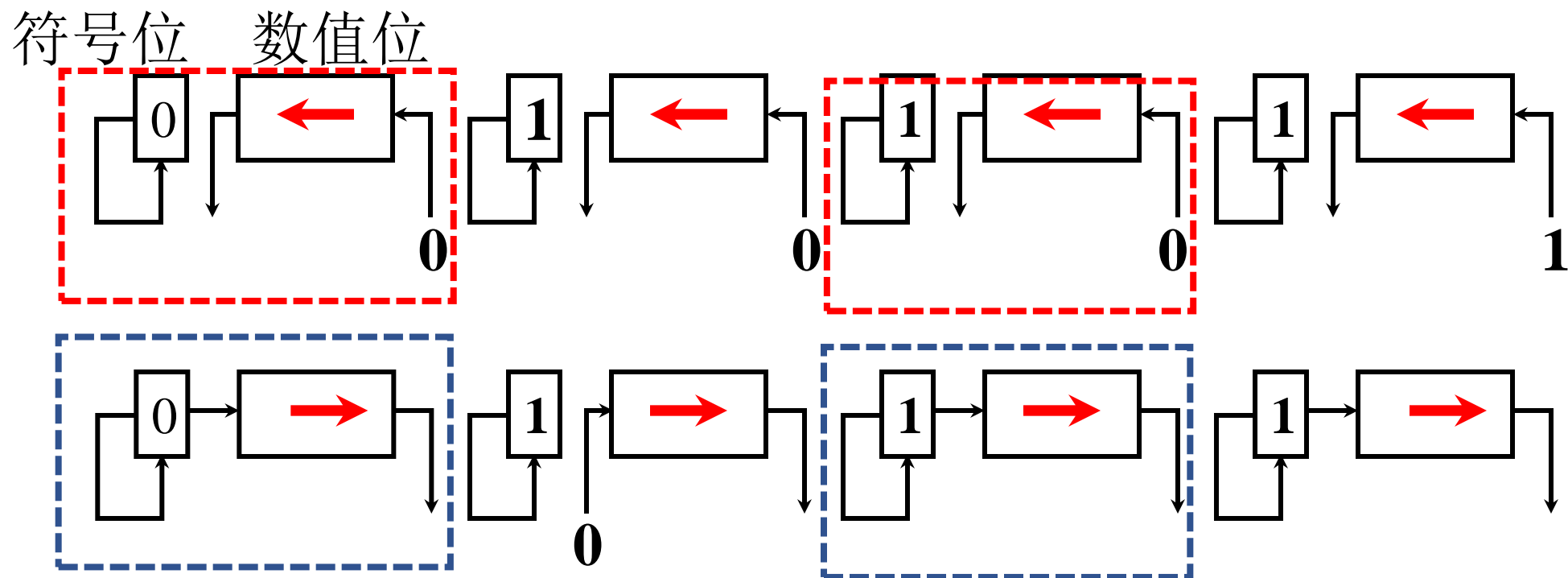
补码

移位操作	机 器 数	对应的真值
移位前	1,1100110	– 26
左移一位	1,1001100	– 52
左移两位	1,0011000	– 104
右移一位	1,1110011	– 13
右移两位	1,1111001	– 7

反码

移位操作	机 器 数	对应的真值
移位前	1,1100101	– 26
左移一位	1,1001011	– 52
左移两位	1,0010111	– 104
右移一位	1,1110010	– 13
右移两位	1,1111001	– 6

算术移位的硬件实现



(a) 真值为正

(b) 负数的原码

(c) 负数的补码

(d) 负数的反码

← 丢 1 出错

出错

正确

正确

→ 丢 1 影响精度

影响精度

影响精度

正确

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
- **IEEE 754标准**
- 算数移位与逻辑移位

第二章 计算机的运算方法

2.1 计算机中数的表示

2.2 定点运算

- 加减法运算
- 一位乘法运算
- Booth算法

2.3 浮点运算

定点加减法运算（补码）

●加法

整数 $[A]_{\text{补}} + [B]_{\text{补}} = [A+B]_{\text{补}} \pmod{2^{n+1}}$

小数 $[A]_{\text{补}} + [B]_{\text{补}} = [A+B]_{\text{补}} \pmod{2}$

●减法

$$A - B = A + (-B)$$

整数 $[A - B]_{\text{补}} = [A + (-B)]_{\text{补}} = [A]_{\text{补}} + [-B]_{\text{补}} \pmod{2^{n+1}}$

小数 $[A - B]_{\text{补}} = [A + (-B)]_{\text{补}} = [A]_{\text{补}} + [-B]_{\text{补}} \pmod{2}$

连同符号位一起相加，符号位产生的进位自然丢掉

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

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

例：用补码运算求 $A+B$

- 设 $A = 0.1011$, $B = -0.0101$, 用补码运算求 $A+B = ?$

解： $[A]_{\text{补}} = 0.1011$

$$+[B]_{\text{补}} = 1.1011$$

$$[A]_{\text{补}} + [B]_{\text{补}} = \cancel{1}0.0110 = [A+B]_{\text{补}}$$

$$\therefore A+B = 0.0110$$

验证

$$\begin{array}{r} 0.1011 \\ - 0.0101 \\ \hline 0.0110 \end{array}$$

- 设 $A = -9$, $B = -5$ (设 A 和 B 数值位数为4), 求 $A+B$

解： $[A]_{\text{补}} = 1, 0111$

$$+[B]_{\text{补}} = 1, 1011$$

$$[A]_{\text{补}} + [B]_{\text{补}} = \cancel{1}1, 0010 = [A+B]_{\text{补}}$$

$$\therefore A+B = -1110$$

验证

$$\begin{array}{r} -1001 \\ + -0101 \\ \hline -1110 \end{array}$$

已知 $[X]_{\text{补}}$ ，以下哪个方法可以求出 $[-X]_{\text{补}}$ ？

- ☐ A 包括符号为在内按位取反
- ☒ B 包括符号位在内，每位取反，末位加 1
- ☐ C 除符号位外，每位取反，末位加 1
- ☐ D 符号为取反，其他保持不变

提交

关于 $[X]_{\text{补}}$ 和 $[-X]_{\text{补}}$ ，哪个表述正确？

- ☒ A $-[X]_{\text{补}} = [-X]_{\text{补}}$
- ☐ B $[X]_{\text{补}} = [-X]_{\text{补}}$
- ☐ C $[X]_{\text{补}}$ 和 $[-X]_{\text{补}}$ 没有任何关系

提交

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

设 $[y]_{\text{补}} = y_0.y_1y_2 \dots y_n$ ，根据符号位 y_0 为0或1分开讨论

<I>

$$[y]_{\text{补}} = 0.y_1y_2 \dots y_n$$

$$[y]_{\text{原}} = 0.y_1y_2 \dots y_n$$

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

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

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

<II>

$$[y]_{\text{补}} = 1.y_1y_2 \dots y_n$$

$$[y]_{\text{原}} = 1 + (0.\overline{y_1}\overline{y_2} \dots \overline{y_n} + 2^{-n})$$

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

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

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

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

观察发现 $[-y]_{\text{补}} + [y]_{\text{补}} = 2 \rightarrow 0 \pmod{2}$ $[-y]_{\text{补}} = -[y]_{\text{补}}$

*例：已知整数 $[X]_{\text{补}}$ ，求 $[-X]_{\text{补}}$

设 $[X]_{\text{补}} = X_n, X_{n-1} \dots X_1 X_0$ ，根据符号位 X_n 为0或1分开讨论

<I>

$$[X]_{\text{补}} = 0, X_{n-1} X_{n-2} \dots X_1 X_0$$

$$X_{\text{原}} = 0, X_{n-1} X_{n-2} \dots X_1 X_0$$

$$[-X]_{\text{原}} = 1, X_{n-1} X_{n-2} \dots X_1 X_0$$

$$[-X]_{\text{补}} = 1, (\bar{X}_{n-1} \bar{X}_{n-2} \dots \bar{X}_1 \bar{X}_0 + 1)$$

<II>

$$[X]_{\text{补}} = 1, X_{n-1} X_{n-2} \dots X_1 X_0$$

$$X_{\text{原}} = 1, (\bar{X}_{n-1} \bar{X}_{n-2} \dots \bar{X}_1 \bar{X}_0 + 1)$$

$$[-X]_{\text{原}} = 0, (\bar{X}_{n-1} \bar{X}_{n-2} \dots \bar{X}_1 \bar{X}_0 + 1)$$

$$[-X]_{\text{补}} = 0, (\bar{X}_{n-1} \bar{X}_{n-2} \dots \bar{X}_1 \bar{X}_0 + 1)$$

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

观察发现 $[-X]_{\text{补}} + [X]_{\text{补}} = 2^{n+1} \rightarrow 0 \pmod{2^{n+1}}$ $[-X]_{\text{补}} = -[X]_{\text{补}}$

例：用补码运算求 $A-B$

设机器数字长为 8 位（含 1 位符号位）且 $A = 15$, $B = 24$, 用补码求 $A - B$ 。

$$\text{解： } A = 15 = 0001111$$

$$B = 24 = 0011000$$

$$[A]_{\text{补}} = 0, 0001111 \quad [B]_{\text{补}} = 0, 0011000$$

$$+[-B]_{\text{补}} = 1, 1101000$$

$$[A]_{\text{补}} + [-B]_{\text{补}} = 1, 1110111 = [A - B]_{\text{补}}$$

$$\therefore A - B = -1001 = -9$$

设机器数字长为 8 位（含 1 位符号位）且 $A = -97$ ， $B = 41$ ，用补码求 $A - B$ （8 位补码表示）。（ $97 = 01100001$ $41 = 00101001$ ）

- ☐ A 11110110
- ☐ B 01110110
- ☐ C 11001000
- ☐ D 其他

例：用补码运算求 $A-B$

- 设机器数字长为 8 位（含 1 位符号位）且 $A = -97$, $B = 41$, 用补码求 $A - B$ 。（ $97 = 01100001$ $41 = 00101001$ ）

解： $A = -97 = -1100001$

$$[A]_{\text{补}} = 1, 0011111$$

$$+[-B]_{\text{补}} = 1, 1010111$$

$$[A]_{\text{补}} + [-B]_{\text{补}} = \mathbf{10}, 1110110 = [A - B]_{\text{补}}$$

$$\therefore A - B = +118 \quad (\text{溢出})$$

一位符号位判溢出

- 参加**加法运算**的两个数（减法时+减数相反数的补码）
 - 如果符号不同，不会溢出。
 - 如果符号相同（同0或同1），其结果的符号与原操作数的符号不同，即为溢出
- 最高有效位的进位 \oplus 符号位的进位 = 1，溢出
- 最高有效位的进位 \oplus 符号位的进位 = 0，无溢出
- 硬件实现——异或门

两位符号位判溢出

- 小数变形补码 $[x]_{\text{补}'} = \begin{cases} x & 1 > x \geq 0 \\ 4 + x & 0 > x \geq -1 \end{cases} \pmod{4}$
 - 整数变形补码 $[X]_{\text{补}'} = \begin{cases} X & 2^n > X \geq 0 \\ 2^{n+2} + X & 0 > x \geq -2^n \end{cases} \pmod{2^{n+2}}$
 - 最高符号位 代表其 真正的符号
 - 结果的双符号位 相同 未溢出
 - 结果的双符号位 不同 溢出
- | | | | |
|-----|--------|-----|--------|
| 00, | xxxxxx | 00. | xxxxxx |
| 11, | xxxxxx | 11. | xxxxxx |
| 10, | xxxxxx | 10. | xxxxxx |
| 01, | xxxxxx | 01. | xxxxxx |

双符号位判断补码运算溢出

- 设机器数字长为 **9** 位（含 2 位符号位）且 $A = -97$, $B = 41$, 用补码求 $A - B$ 。 ($97 = 01100001$ $41 = 00101001$)

解: $A = -97 = -1100001$

$$B = 41 = 0101001$$

$$[A]_{\text{补}} = 11, 0011111$$

$$+[-B]_{\text{补}} = 11, 1010111$$

$$[A]_{\text{补}} + [-B]_{\text{补}} = 1\mathbf{10}, 1110110 = [A - B]_{\text{补}}$$

$$\therefore A - B = +118 \quad (\text{溢出})$$

乘法运算——笔算乘法分析

$$A = -0.1101 \quad B = 0.1011$$

$$A \times B = -0.10001111$$

乘积的符号心算求得

$$\begin{array}{r} 0.1101 \\ \times 0.\textcolor{red}{1}0\textcolor{blue}{1}1 \\ \hline 1101 \\ \textcolor{blue}{1101} \\ 0000 \\ \textcolor{red}{1101} \\ \hline 0.10001111 \end{array}$$

✓ 符号位单独处理

✓ 乘数的某一位决定是否加被乘数

? 4位的积一起相加

✓ 乘积的位数扩大一倍

笔算乘法的改进

$$A \cdot B = A \cdot 0.1011$$

$$= 0.1A + 0.00A + \underline{0.001A} + 0.0001A$$

$$= 0.1A + 0.00A + \underline{0.001}(1 \cdot A + 0.1A)$$

$$= 0.1A + \underline{0.01}[0 \cdot A + 0.1(1 \cdot A + 0.1A)]$$

$$= \underline{0.1}\{1 \cdot A + 0.1[0 \cdot A + 0.1(1 \cdot A + \underline{0.1A})]\}$$

右移一位

$$= \underline{2^{-1}}\{1 \cdot A + 2^{-1}[0 \cdot A + 2^{-1}(1 \cdot A + \underline{2^{-1}(1 \cdot A + 0)})]\}$$

1 被乘数 $A + 0$

2 右移一位，得新的部分积

3 部分积 + 被乘数

...

右移一位，得结果

①

②

③

⑧

改进后的笔算乘法过程（竖式）

0.1101	部分积(乘数暂存)	丢弃	说明
$\times 0.1011$	0.0000	1011	初态，部分积高位=0
<u>???...????</u>	+ 0.1101		乘数为 1，加被乘数
	<u>0.1101</u>	1011	$\rightarrow 1$ ，（逻辑右移）
	<u>0.0110</u>	1101	形成新的部分积
	+ 0.1101		乘数为 1，加被乘数
	<u>1.0011</u>	1101	$\rightarrow 1$ ，
	<u>0.1001</u>	1110	形成新的部分积
	+ 0.0000		乘数为 0，加 0
	<u>0.1001</u>	1110	$\rightarrow 1$ ，
	<u>0.0100</u>	1111	形成新的部分积
	+ 0.1101		乘数为 1加 被乘数
	<u>1.0001</u>	111	$\rightarrow 1$ ，
	<u>0.1000</u>	1111	无乘数位，结束

改进的笔算乘法小结

- 两个 n 位数乘法运算可用 加 n 次和移位 n 次 实现
- 由乘数的末位决定 被乘数是否与原部分积相加
- 乘数右移1位（末位舍弃），空出高位存放部分积的低位，
同时部分积 右移 1 位 形成新的部分积。
- 被乘数只与部分积的高 n 位相加
- 硬件需求：移位寄存器, 全加器

计算机的运算方法

2.2 定点运算

- 加减法运算
- 一位乘法
- Booth算法
- 除法运算

原码一位乘运算规则

整数： 设 $[X]_{\text{原}} = \textcolor{red}{X}_n X_{n-1} \dots X_1 X_0$, $[Y]_{\text{原}} = \textcolor{red}{Y}_n Y_{n-1} \dots Y_1 Y_0$

$$\begin{aligned}[X \times Y]_{\text{原}} &= (X_n \oplus Y_n)(\textcolor{blue}{0} X_{n-1} \dots X_1 X_0 \times \textcolor{blue}{0} Y_{n-1} \dots Y_1 Y_0) \\ &= (X_n \oplus Y_n)(|X| \times |Y|)\end{aligned}$$

小数： 设 $[x]_{\text{原}} = \textcolor{red}{x}_0 . x_1 x_2 \dots x_n$, $[y]_{\text{原}} = \textcolor{red}{y}_0 . y_1 y_2 \dots y_n$

$$\begin{aligned}[x \times y]_{\text{原}} &= (x_0 \oplus y_0) . (0 . x_1 x_2 \dots x_n \times 0 . y_1 y_2 \dots y_n) \\ &= (x_0 \oplus y_0) . (|x| \times |y|)\end{aligned}$$

乘积的符号位单独处理, 数值部分为绝对值相乘

小数原码一位乘递推公式

$$\begin{aligned}
 |x| \times |y| &= |x|(0.y_1y_2 \dots y_n) \\
 &= |x| \times (y_1 \times 2^{-1} + y_2 \times 2^{-2} + \dots + y_n \times 2^{-n}) \\
 &= 2^{-1}(y_1 \times |x| + 2^{-1}y_2 \times |x| + \dots + 2^{-n+1}y_n \times |x|) \\
 &= 2^{-1}(y_1 \times |x| + \underbrace{2^{-1}(y_2 \times |x| + \dots + 2^{-1}(y_n \times |x| + \underbrace{0}_{z_0}) \dots))}_{z_1} \dots) \\
 &\quad \underbrace{\hspace{10em}}_{z_n}
 \end{aligned}$$

$$z_0 = 0$$

$$z_1 = 2^{-1}(y_n|x| + z_0)$$

$$z_2 = 2^{-1}(y_{n-1}|x| + z_1)$$

...

$$z_i = 2^{-1}(y_{n+1-i}|x| + z_{i-1})$$

...

$$z_n = 2^{-1}(y_1|x| + z_{n-1})$$

$x = -0.1110$, $y = 0.1101$, 求 $[x \times y]_{\text{原}}$

数值部分的运算

部分积(乘数暂存)	丢弃	说明
0.0000	110 <u>1</u>	部分积, 初态 $z_0 = 0$
+ 0.1110		乘数为 <u>1</u> , 加 $/x/$
0.1110	110 <u>1</u>	逻辑右移 $\rightarrow 1$,
0.0111	0 <u>110</u> 1	得 z_1
+ 0.0000		乘数为 <u>0</u> , 加 0
0.0111	0 <u>110</u>	逻辑右移 $\rightarrow 1$,
0.0011	1 <u>011</u> 01	得 z_2
+ 0.1110		乘数为 <u>1</u> , 加 $/x/$
1.0001	1 <u>011</u>	逻辑右移 $\rightarrow 1$,
0.1000	1 <u>101</u> 101	得 z_3
+ 0.1110		乘数为 <u>1</u> , 加 $/x/$
1.0110	1 <u>101</u>	逻辑右移 $\rightarrow 1$,
0.1011	0 <u>110</u> 1101	得 z_4 , 结束

整数原码一位乘递推公式

$$\begin{aligned}
 |X| \times |Y| &= |X|(\mathbf{0}Y_{n-1} \dots Y_1Y_0) = 2^{n-1}Y_{n-1}|X| + 2^{n-2}Y_{n-2}|X| + \dots + 2^1Y_1|X| + 2^0Y_0|X| \\
 &= \mathbf{2}^n * (2^{-1}Y_{n-1}|X| + 2^{-2}Y_{n-2}|X| + \dots + 2^{-n+1}Y_1|X| + 2^{-n}Y_0|X|) \\
 &= 2^n * (2^{-1} * (Y_{n-1}|X| + 2^{-1}Y_{n-2}|X| + \dots + 2^{-n+2}Y_1|X| + 2^{-n+1}Y_0|X|)) \\
 &= 2^n * (2^{-1} * (Y_{n-1}|X| + 2^{-1}(Y_{n-2}|X| + \dots + 2^{-n+3}Y_1|X| + 2^{-n+2}Y_0|X|))) \\
 &\quad \dots\dots\dots \dots\dots\dots \dots\dots\dots \dots\dots\dots \dots\dots\dots \dots\dots\dots \\
 &= \mathbf{2}^n * (2^{-1} * (Y_{n-1}|X| + 2^{-1}(Y_{n-2}|X| + 2^{-1}(\dots + 2^{-1}(Y_1|X| + 2^{-1}(Y_0|X| + \mathbf{0})))))) \dots)
 \end{aligned}$$

$$\underbrace{\hspace{15em}}_{Z_n} \qquad \underbrace{\hspace{15em}}_{Z_1}^{Z_0}$$

$$Z_0 = 0$$

$$Z_1 = 2^{-1}(Y_0|X| + Z_0)$$

$$Z_2 = 2^{-1}(Y_1|X| + Z_1)$$

...

$$Z_n = 2^{-1}(Y_{n-1}|X| + Z_{n-1})$$

$$X = -1110, Y = 1101, \text{求}[X \times Y]_{\text{原}}$$

数值部分的运算

部分积(乘数暂存)	丢弃	说明
0 0 0 0	1 1 0 1	高部分积置0 (* 2^n), $Z_0 = 0$
+ 1 1 1 0		乘数为 1, 加/X/
1 1 1 0	1 1 0 1	逻辑右移→1,
0 1 1 1	0 1 1 0	得 Z_1
+ 0 0 0 0		乘数为 0, 加 0
0 1 1 1	0 1 1 0	逻辑右移→1,
0 0 1 1	1 0 1 1	得 Z_2
+ 1 1 1 0		乘数为 1, 加/X/
1 0 0 0 1	1 0 1 1	逻辑右移→1,
0 1 0 0 0	1 1 0 1	得 Z_3
+ 0 1 1 1 0		乘数为 1, 加/X/
1 0 1 1 0	1 1 0 1	逻辑右移→1,
0 1 0 1 1	0 1 1 0	得 Z_4 , 结束

原码乘法小结

- 乘积的符号位 = 被乘数的符号位 \oplus 乘数的符号位
- 数值部分按绝对值相乘

特点：

- 绝对值运算
- 用移位的次数判断乘法是否结束
- 逻辑移位

补码一位乘法（小数）

- 补码一位乘运算规则

以小数为例 设被乘数 $[x]_{\text{补}} = x_0.x_1x_2 \cdots x_n$

乘数 $[y]_{\text{补}} = y_0.y_1y_2 \cdots y_n$

① 被乘数任意，乘数为正

同原码乘 但加和移位按补码规则运算
乘积的符号自然形成

② 被乘数任意，乘数为负

乘数 $[y]_{\text{补}}$ ，去掉符号位，操作同 ①

最后加 $[-x]_{\text{补}}$ ，校正

补码一位乘法（小数）

① 被乘数任意，乘数为正

$$[x]_{\text{补}} = x_0.x_1x_2 \dots x_n$$

$$[x]_{\text{补}} = x_0.x_1x_2 \dots x_n = 2 + x = 2^{n+1} + x \pmod{2}$$

$$[y]_{\text{补}} = y_0.y_1y_2 \dots y_n$$

$$[y]_{\text{补}} = 0.y_1y_2 \dots y_n$$

$$[x]_{\text{补}} \times [y]_{\text{补}} = [x]_{\text{补}} \times y = (2^{n+1} + x) \times y = 2^{n+1} \times y + x \times y$$

$$2^{n+1} \times y = 2 \sum_{i=1}^n y_i 2^{n-i} \quad \text{且} \quad \sum_{i=1}^n y_i 2^{n-i} \text{ 是一个大于等于 1 的正数。}$$

$$\text{则 } 2^{n+1} \times y = 2 \pmod{2}$$

$$[x]_{\text{补}} \times [y]_{\text{补}} = 2^{n+1} \times y + x \times y = 2 + x \times y = [x \times y]_{\text{补}}$$

$$\text{即 } [x \times y]_{\text{补}} = [x]_{\text{补}} \times y$$

同原码乘 但加和移位按补码规则运算
乘积的符号自然形成

补码一位乘法（小数）

② 被乘数任意，乘数为负

$$[x]_{\text{补}} = x_0.x_1x_2 \cdots x_n$$

$$[y]_{\text{补}} = 1.y_1y_2 \cdots y_n = 2 + y \pmod{2}$$

$$[y]_{\text{补}} = y_0.y_1y_2 \cdots y_n$$

$$y = [y]_{\text{补}} - 2 = 1.y_1y_2 \cdots y_n - 2 = 0.y_1y_2 \cdots y_n - 1$$

$$x \times y = x \times (0.y_1y_2 \cdots y_n - 1) = x \times (0.y_1y_2 \cdots y_n) - x$$

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

$$[x \times y]_{\text{补}} = [x \times (0.y_1y_2 \cdots y_n)]_{\text{补}} + [-x]_{\text{补}}$$

$$[x \times (0.y_1y_2 \cdots y_n)]_{\text{补}} = [x]_{\text{补}} \times (0.y_1y_2 \cdots y_n) \quad (\text{与①相同})$$

$$[x \times y]_{\text{补}} = [x]_{\text{补}} \times (0.y_1y_2 \cdots y_n) + [-x]_{\text{补}}$$

乘数 $[y]_{\text{补}}$ ，去掉符号位，操作同①

最后加 $[-x]_{\text{补}}$ ，校正

补码一位乘法（小数）

- 补码一位乘运算规则

① 被乘数任意，乘数为正

$$[x \times y]_{\text{补}} = [x]_{\text{补}} \times y$$

② 被乘数任意，乘数为负

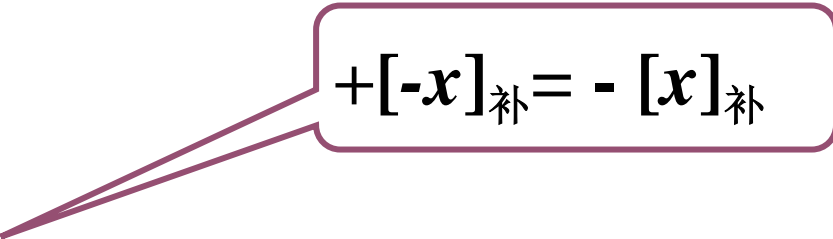
$$[x \times y]_{\text{补}} = [x]_{\text{补}} \times (0.y_1y_2 \dots y_n) + [-x]_{\text{补}}$$

$$[x]_{\text{补}} = x_0.x_1x_2 \dots x_n$$

$$[y]_{\text{补}} = y_0.y_1y_2 \dots y_n$$

统一算法

$$\begin{aligned}[x \times y]_{\text{补}} &= [x]_{\text{补}} \times (0.y_1y_2 \dots y_n) + [-x]_{\text{补}} \times y_0 \\ &= [x]_{\text{补}} \times (0.y_1y_2 \dots y_n) - [x]_{\text{补}} \times y_0\end{aligned}$$


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

• 例. 已知 $[x]_{\text{补}} = 0.1101$, $[y]_{\text{补}} = 1.0101$, 求 $[x \times y]_{\text{补}}$

解:		初值 $[z_0]_{\text{补}} = 0$	$[-x]_{\text{补}} = 1.0011$
$\begin{array}{r} 00.0000 \\ + 00.1101 \\ \hline 00.1101 \end{array}$		$y_4 = 1, +[x]_{\text{补}}$	
补码右移	$\begin{array}{r} 00.1101 \\ 00.0110 \\ \hline 00.0011 \end{array}$	$\begin{array}{r} 010\underline{1} \\ 101\underline{0} \\ \hline 010\underline{1} \end{array}$	$\rightarrow 1$ 得 $[z_1]_{\text{补}}$
补码右移	$\begin{array}{r} 00.0011 \\ + 00.1101 \\ \hline 01.0000 \end{array}$	$\begin{array}{r} 010\underline{1} \\ 001\underline{0} \\ \hline 0100 \end{array}$	$\rightarrow 1$ $y_3 = 0$, 直接右移得 $[z_2]_{\text{补}}$ $y_2 = 1, +[x]_{\text{补}}$
补码右移	$\begin{array}{r} 01.0000 \\ 00.1000 \\ \hline 00.0100 \end{array}$	$\begin{array}{r} 01 \\ 001\underline{0} \\ \hline 0001 \end{array}$	$\rightarrow 1$ 得 $[z_3]_{\text{补}}$
补码右移	$\begin{array}{r} 00.0100 \\ + 11.0011 \\ \hline 11.0111 \end{array}$	$\begin{array}{r} 0001 \\ 0001 \\ \hline 0001 \end{array}$	$\rightarrow 1$ $y_1 = 0$, 直接右移, 得 $[z_4]_{\text{补}}$ $+[-x]_{\text{补}}$ 进行校正
$\begin{array}{r} 11.0111 \\ 0001 \end{array}$		得最后结果	$\therefore [x \times y]_{\text{补}} = 1.01110001$

计算机的运算方法

2. 2定点运算

- 加减法运算
- 一位乘法
- Booth算法

Booth 算法（被乘数、乘数符号任意）

$$\text{设 } [x]_{\text{补}} = x_0 \cdot x_1 x_2 \cdots x_n \quad [y]_{\text{补}} = y_0 \cdot y_1 y_2 \cdots y_n$$

$$[x \times y]_{\text{补}}$$

$$= [x]_{\text{补}} (0 \cdot y_1 \cdots y_n) - [x]_{\text{补}} \cdot y_0$$

$$= [x]_{\text{补}} (y_1 2^{-1} + y_2 2^{-2} + \cdots + y_n 2^{-n}) - [x]_{\text{补}} \cdot y_0$$

$$2^{-1} = 2^0 - 2^{-1}$$

$$= [x]_{\text{补}} (-y_0 + y_1 2^{-1} + y_2 2^{-2} + \cdots + y_n 2^{-n})$$

$$2^{-2} = 2^{-1} - 2^{-2}$$

$$= [x]_{\text{补}} [-y_0 + (y_1 - y_1 2^{-1}) + (y_2 2^{-1} - y_2 2^{-2}) + \cdots + (y_n 2^{-(n-1)} - y_n 2^{-n})]$$

$$= [x]_{\text{补}} [(y_1 - y_0) + (y_2 - y_1) 2^{-1} + \cdots + (y_n - y_{n-1}) 2^{-(n-1)} + (0 - y_n) 2^{-n}]$$

$$= [x]_{\text{补}} [(y_1 - y_0) + (y_2 - y_1) 2^{-1} + \cdots + (y_{n+1} - y_n) 2^{-n}]$$

附加位 y_{n+1}

$$y'_1 2^{-1} + \cdots + y'_n 2^{-n}$$

Booth算法递推公式

$$[z_0]_{\text{补}} = 0 \quad y_{n+1} = 0$$

$$[z_1]_{\text{补}} = 2^{-1} \{ (y_{n+1} - y_n) [x]_{\text{补}} + [z_0]_{\text{补}} \}$$

$$[z_2]_{\text{补}} = 2^{-1} \{ (y_n - y_{n-1}) [x]_{\text{补}} + [z_1]_{\text{补}} \}$$

.....

$$[z_i]_{\text{补}} = 2^{-1} \{ (y_{n+2-i} - y_{n+1-i}) [x]_{\text{补}} + [z_{i-1}]_{\text{补}} \}$$

.....

$$[z_n]_{\text{补}} = 2^{-1} \{ (y_2 - y_1) [x]_{\text{补}} + [z_{n-1}]_{\text{补}} \}$$

$$[x \times y]_{\text{补}} = (y_1 - y_0) [x]_{\text{补}} + [z_n]_{\text{补}}$$

最后一步不移位

y_i	y_{i+1}	$y_{i+1} - y_i$	操作
0	0	0	$[z_{n-i}]_{\text{补}} \rightarrow 1$
0	1	1	$[z_{n-i}]_{\text{补}} + [x]_{\text{补}} \rightarrow 1$
1	0	-1	$[z_{n-i}]_{\text{补}} + [-x]_{\text{补}} \rightarrow 1$
1	1	0	$[z_{n-i}]_{\text{补}} \rightarrow 1$

已知 $x = +0.0011$, $y = -0.1011$, 求 $[x \times y]_{\text{补}}$

0.0000	1.01010	说明
+ 1.1101		$y_i y_{i+1} = 10$, 加 $[-x]_{\text{补}}$
1.1101	10101	算术右移 $\rightarrow 1$
1.1110	110101	
+ 0.0011		$y_i y_{i+1} = 01$, 加 $[x]_{\text{补}}$
0.0001	11010	算术右移 $\rightarrow 1$
0.0000	111010	$0 \neq 0$
+ 1.1101		$y_i y_{i+1} = 10$, 加 $[-x]_{\text{补}}$
1.1101	11101	算术右移 $\rightarrow 1$
1.1110	11110	$1 \neq 0$
+ 0.0011		$y_i y_{i+1} = 01$, 加 $[x]_{\text{补}}$
0.0001	11110	算术右移 $\rightarrow 1$
0.0000	11111	$0 \neq 0 \neq 0$
+ 1.1101		$y_i y_{i+1} = 10$, 加 $[-x]_{\text{补}}$
1.1101	1111	结束(最后一步不移位)

$[x]_{\text{补}} = 0.0011$
 $[y]_{\text{补}} = 1.0101$
 $[-x]_{\text{补}} = 1.1101$

$y_i \ y_{i+1}$	$y_{i+1} - y_i$	操作
0 0	0	$\rightarrow 1$
0 1	1	$+ [x]_{\text{补}} \rightarrow 1$
1 0	-1	$+ [-x]_{\text{补}} \rightarrow 1$
1 1	0	$\rightarrow 1$

$\therefore [x \times y]_{\text{补}}$
 $= 1.11011111$

已知 $x = +0.0011$, $y = -0.1011$, 求 $[x \times y]_{\text{补}}$

00.0000	1.01010	说明
$+11.1101$		$y_i y_{i+1} = 10$, 加 $[-x]_{\text{补}}$
11.1101	10101	算术右移 $\rightarrow 1$
11.1110	110101	
$+00.0011$		$y_i y_{i+1} = 01$, 加 $[x]_{\text{补}}$
00.0001	11010	算术右移 $\rightarrow 1$
00.0000	111010	$0 \neq 0$
$+11.1101$		$y_i y_{i+1} = 10$, 加 $[-x]_{\text{补}}$
11.1101	11101	算术右移 $\rightarrow 1$
11.1110	11110	$1 \neq 0$
$+00.0011$		$y_i y_{i+1} = 01$, 加 $[x]_{\text{补}}$
00.0001	11110	算术右移 $\rightarrow 1$
00.0000	11111	$0 \neq 0 \neq 0$
$+11.1101$		$y_i y_{i+1} = 10$, 加 $[-x]_{\text{补}}$
11.1101	1111	最后一步不移位

$[x]_{\text{补}} = 0.0011$
 $[y]_{\text{补}} = 1.0101$
 $[-x]_{\text{补}} = 1.1101$

$y_i y_{i+1}$	$y_{i+1} - y_i$	操作
0 0	0	$\rightarrow 1$
0 1	1	$+ [x]_{\text{补}} \rightarrow 1$
1 0	-1	$+ [-x]_{\text{补}} \rightarrow 1$
1 1	0	$\rightarrow 1$

$\therefore [x \times y]_{\text{补}}$
 $= 1.11011111$

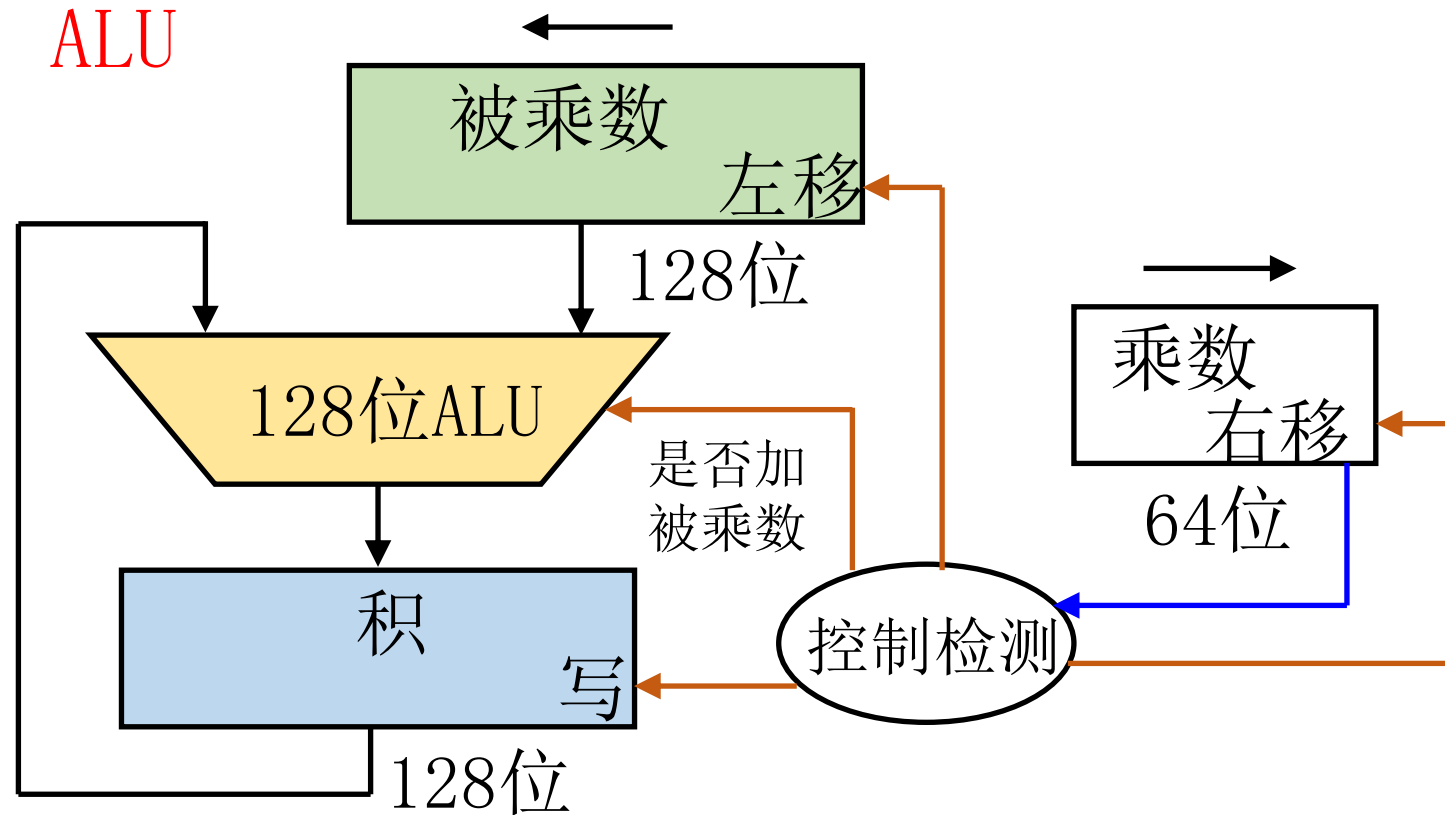
乘法小结

- 整数乘法与小数乘法基本相同
 - 用 逗号 代替小数点
 - 整数有整体移位操作，部分积最开始保存在积的高位
- 符号位：原码乘需要单独处理，补码乘 自然形成
- 原码乘去掉符号位运算，即为无符号数乘法
- 不同的乘法运算需有不同的硬件支持

乘法器硬件示意图

- 被乘数寄存器128位
 - 被乘数64位，要左移一位64次。
- 浪费：被乘数寄存器、ALU
 - 多数时间只用64位

$$\begin{array}{r} 1101 \\ \times 1011 \\ \hline 1101 \\ 1101 \\ 0000 \\ 1101 \\ \hline 10001111 \end{array}$$



黑书图3-3

改良版乘法器硬件

A. 积寄存器129位(初值为: 65个0 和 64位的乘数)

- 最高位用于保存加法器的进位

B. 若乘数最右端为1

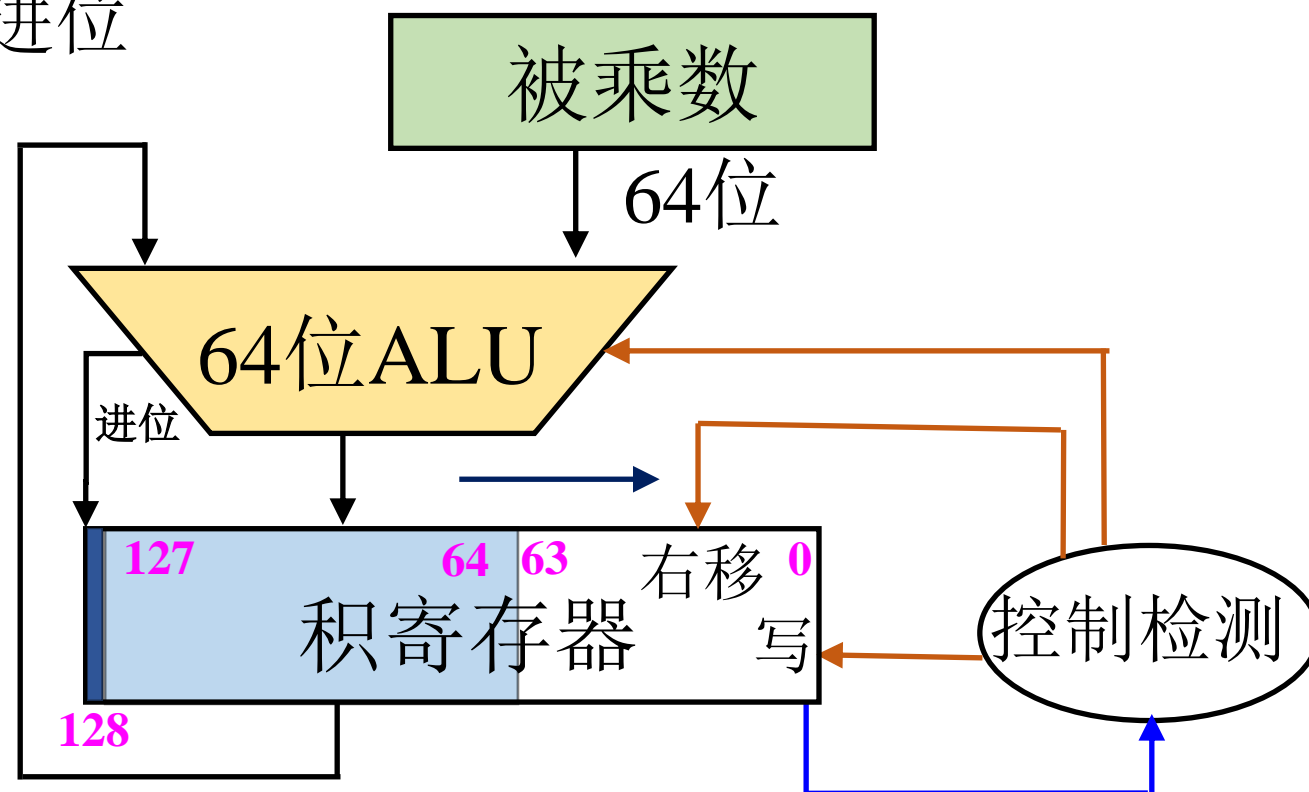
- 取积寄存器[127:64]
- 取出的值加上被乘数;
- **和**写入积寄存器[**128:64**]

C. 积寄存器整体右移一位

*B和C循环64次

*结果为积寄存器[127:0]

*快速乘法: 黑书3.3.3



黑书图3-5

计算机的运算方法

2.2 定点运算

- 加减法运算
- 一位乘法
- Booth算法

例. $[x]_{\text{补}} = 0.1101$, $[y]_{\text{补}} = 0.1011$, 求 $[x \cdot y]_{\text{补}}$

0 0 . 0 0 0 0	0.1 0 1 1 0	说明
+ 1 1 . 0 0 1 1		$y_i y_{i+1} = 10$, 加 $[-x]_{\text{补}}$
1 1 . 0 0 1 1	0 1 0 1 1	算术右移 $\rightarrow 1$
1 1 . 1 0 0 1	1 0 1 0 1 1	
1 1 . 1 1 0 0	1 1 0 1 0 1 $\neq 0$	$y_i y_{i+1} = 11$, 算术右移 $\rightarrow 1$
+ 0 0 . 1 1 0 1		$y_i y_{i+1} = 01$, 加 $[x]_{\text{补}}$
0 0 . 1 0 0 1	1 1 0 1 0	算术右移 $\rightarrow 1$
0 0 . 0 1 0 0	1 1 1 0 1 0 $\neq 0$	
+ 1 1 . 1 1 0 1		$y_i y_{i+1} = 10$, 加 $[-x]_{\text{补}}$
1 1 . 0 1 1 1	1 1 1 0 1	算术右移 $\rightarrow 1$
1 1 . 1 0 1 1	1 1 1 1 0 1 $\neq 0$	
+ 0 0 . 1 1 0 1		$y_i y_{i+1} = 01$, 加 $[x]_{\text{补}}$
0 0 . 1 0 0 0	1 1 1 1	最后一步不移位

$$[x]_{\text{补}} = 00.1101$$

$$[y]_{\text{补}} = 00.1011$$

$$[-x]_{\text{补}} = 11.0011$$

$y_i \ y_{i+1}$	$y_{i+1} - y_i$	操作
0 0	0	$\rightarrow 1$
0 1	1	$+ [x]_{\text{补}} \rightarrow 1$
1 0	-1	$+ [-x]_{\text{补}} \rightarrow 1$
1 1	0	$\rightarrow 1$

$$\therefore [x \times y]_{\text{补}} = 0.10001111$$

第二章 计算机的运算方法

2.1 计算机中数的表示

2.2 定点运算

2.3 浮点运算

浮点四则运算

- 一、浮点加减运算

$$x = S_x \cdot 2^{j_x} \quad y = S_y \cdot 2^{j_y}$$

1. 对阶：使两个浮点数的阶码相同

(1) 求阶差

$$\Delta j = j_x - j_y \quad \left\{ \begin{array}{ll} = 0 & j_x = j_y \quad \text{已对齐} \\ > 0 & j_x > j_y \quad \left\{ \begin{array}{ll} x \text{ 向 } y \text{ 看齐} & S_x \leftarrow 1, j_x - 1 \\ y \text{ 向 } x \text{ 看齐} & \checkmark S_y \rightarrow 1, j_y + 1 \end{array} \right. \\ < 0 & j_x < j_y \quad \left\{ \begin{array}{ll} x \text{ 向 } y \text{ 看齐} & \checkmark S_x \rightarrow 1, j_x + 1 \\ y \text{ 向 } x \text{ 看齐} & S_y \leftarrow 1, j_y - 1 \end{array} \right. \end{array} \right.$$

(2) 对阶原则

小阶向大阶看齐

例: $x = 0.1101 \times 2^{01}$, $y = (-0.1010) \times 2^{11}$, 求 $x + y$

解: $[x]_{\text{补}} = 00, 01; 00.1101$ $[y]_{\text{补}} = 00, 11; 11.0110$

1. 对阶

$$\begin{aligned} \text{① 求阶差 } [\Delta j]_{\text{补}} &= [j_x]_{\text{补}} - [j_y]_{\text{补}} = 00, 01 \\ &\quad + 11, 01 \\ &\quad \hline &\quad 11, 10 \end{aligned}$$

阶差为负 (- 2) $\therefore S_x \rightarrow 2 \quad j_x + 2$

$$\text{② 对阶 } [x]_{\text{补}}' = 00, 11; 00.0011$$

2. 尾数求和

$$\begin{array}{rcl} [S_x]_{\text{补}}' & = & 00.0011 \\ + [S_y]_{\text{补}} & = & 11.0110 \\ \hline & & 11.1001 \end{array} \quad \text{对阶后的 } [S_x]_{\text{补}}'$$

$$\therefore [x+y]_{\text{补}} = 00, 11; 11. 1001$$

左规和右规

- 原码规格化判断方法：不论正数、负数，第一数位为1
- 补码规格化判断方法：符号位和第一数位不同
- 补码左规：尾数每左移1位，阶码减1，直到数符与第一数位不同。

$$[x+y]_{\text{补}} = 00, 11; 11. 1001$$

$$\text{左规后 } [x+y]_{\text{补}} = 00, 10; 11. 0010$$

$$\therefore x + y = (-0.1110) \times 2^{10}$$

- 右规：当尾数溢出（>1）时，需右规

例如：双符号位的尾数出现 01. × ... × 或 10. × ... × 时，
尾数每右移一位，阶码加 1。

设 $x = 0.1101 \times 2^{10}$, $y = 0.1011 \times 2^{01}$, 求 $x + y$ (除阶符2位、数符2位外, 阶码取3位, 尾数取6位)

解: $[x]_{\text{补}} = 00, 010; 00.110100$

$[y]_{\text{补}} = 00, 001; 00.101100$

① 对阶 $[\Delta j]_{\text{补}} = [j_x]_{\text{补}} - [j_y]_{\text{补}} = 00, 010$

$$\begin{array}{r} + 11, 111 \\ \hline \textcolor{red}{1}00, 001 \end{array}$$

阶差为 +1 $\therefore S_y \rightarrow 1, j_y + 1$

$\therefore [y]_{\text{补}}' = 00, 010; 00.010110$

② 尾数求和: $[S_x]_{\text{补}} = 00.110100$

$$\begin{array}{r} + [S_y]_{\text{补}}' = \textcolor{violet}{00.010110} \\ \hline 01.001010 \end{array}$$

对阶后的 $[S_y]_{\text{补}}'$
尾数溢出需右规

③ 右规

$$[x+y]_{\text{补}} = 00, 010; 01. 001010$$

右规后

$$[x+y]_{\text{补}} = 00, 011; 00. 100101$$

$$\therefore x+y = 0. 100101 \times 2^{11}$$

• 4. 舍入

- 在 对阶 和 右规 过程中，可能出现尾数末位丢失引起误差，需考虑舍入

(1) 0 舍 1 入法

(2) 恒置 “1” 法

例 $x = (-\frac{5}{8}) \times 2^{-5}$, $y = (\frac{7}{8}) \times 2^{-4}$, 求 $x - y$ (除阶符、数符外, 阶码取 3 位, 尾数取 6 位)

正常使用主观题需2.0以上版本雨课堂

作答

- 例. $x = (-\frac{5}{8}) \times 2^{-5}$, $y = (\frac{7}{8}) \times 2^{-4}$, 求 $x - y$ (除阶符、数符外, 阶码取 3 位, 尾数取 6 位)

解: $x = (-0.101000) \times 2^{-101}$

$y = (0.111000) \times 2^{-100}$

$[x]_{\text{补}} = 11, 011; 11. 011000$

$[y]_{\text{补}} = 11, 100; 00. 111000$

① 对阶

$$[\Delta j]_{\text{补}} = [j_x]_{\text{补}} - [j_y]_{\text{补}} = 11, 011$$

$$\begin{array}{r} + 00, 100 \\ \hline 11, 111 \end{array}$$

阶差为 -1 $\therefore S_x \rightarrow 1, j_x + 1$

$\therefore [x]_{\text{补}'} = 11, 100; 11. 101100$

② 尾数求和

$$\begin{array}{r} [S_x]_{\text{补}'} = 11. 101100 \\ + [-S_y]_{\text{补}} = 11. 001000 \\ \hline 110. 110100 \end{array}$$

③ 右规

$$[x - y]_{\text{补}} = 11, 100; 10. 110100$$

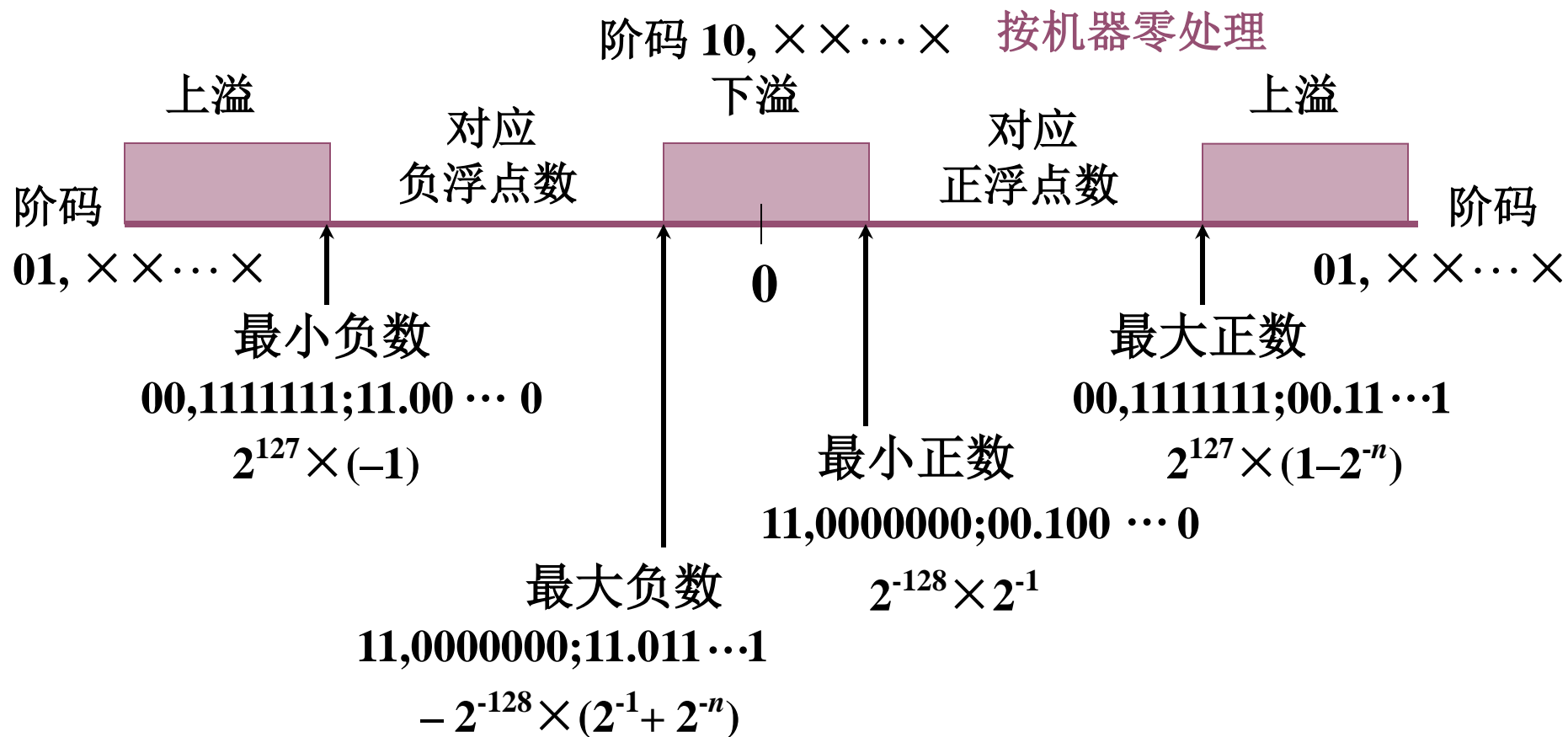
右规后

$$[x - y]_{\text{补}} = 11, 101; 11. 011010$$

$$\begin{aligned} \therefore x - y &= (-0.100110) \times 2^{-11} \\ &= \left(-\frac{19}{32}\right) \times 2^{-3} \end{aligned}$$

溢出判断

- 设机器数为补码，尾数为规格化形式，并假设阶符取 2 位，阶码的数值部分取 7 位，数符取 2 位，尾数取 n 位，则该补码在数轴上的表示为



计算机的运算方法

2.1 计算机中数的表示

2.2 定点运算

- 加减法运算
- 一位乘法运算
- Booth算法

2.3 浮点运算