计算机组成原理

2021春计算机组成原理19级456班

QQ群号：745575324

扫一扫二维码，加入群聊。

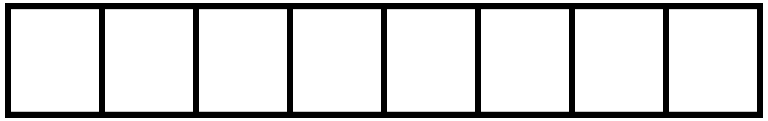
顾崇林
计算机科学与技术学院
guchonglin@hit.edu.cn

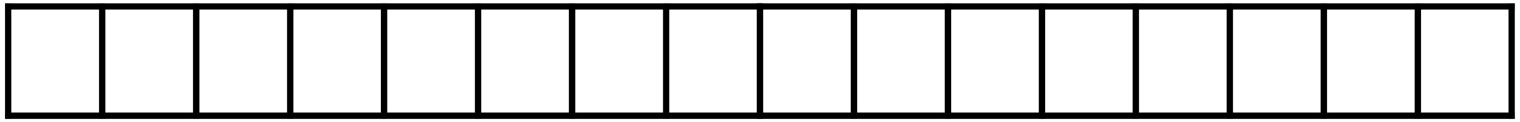
第二章 计算机的运算方法

- **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
- 定点运算
- 浮点运算

无符号数

- 寄存器的位数反映无符号数的表示范围。

8 位  **0 ~ 255**

16 位  **0 ~ 65535**

有符号数：真值与机器数

真值：带符号的数	机器数：符号数字化的数	
+ 0.1011或0.1001	0 ‖ 1011	小数点的位置
– 0.1011	1 ‖ 1011	小数点的位置
+ 1100或1100	0 ‖ 1100	小数点的位置
– 1100	1 ‖ 1100	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。

原码表示法：整数

带符号的绝对值表示

$x = +1110$　　$[x]_{原} = 0\,,1110$

用 逗号 将符号位和数值部分隔开

$x = -1110$　　$[x]_{原} = 1\,,1110$

$[x]_{原} = 2^4 + 1110 = 1\,,1110$

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

原码表示法：小数

$x = +0.1101$ $[x]_{原} = 0\,.\,1101$

用 小数点 将符号位和数值部分隔开

$x = -0.1101$ $[x]_{原} = 1-(-0.1101) = 1\,.\,1101$

$x = +0.1000000$ $[x]_{原} = 0\,.\,1000000$

用 小数点 将符号位和数值部分隔开

$x = -0.1000000$ $[x]_{原} = 1-(-0.1000000) = 1\,.\,1000000$

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

举例

- 例1. 已知 $[x]_{原} = 1.0011$，求 x

  解：由定义得

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

- 例2. 已知 $[x]_{原} = 1,1100$，求 x

  解：由定义得

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

举例

- 例3. 已知 $[x]_{原} = 0.1101$，求 x

  解：根据定义 $\because [x]_{原} = 0.1101$

  $\therefore \quad x = +0.1101$

- 例4. 求 $x = 0$ 的原码

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

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

  同理，对于整数 $[+0]_{原} = 0,0000$

  $[-0]_{原} = 1,0000$

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

  注意： $x = 0$ 也是要分成小数和整数分别讨论的

原码的优缺点

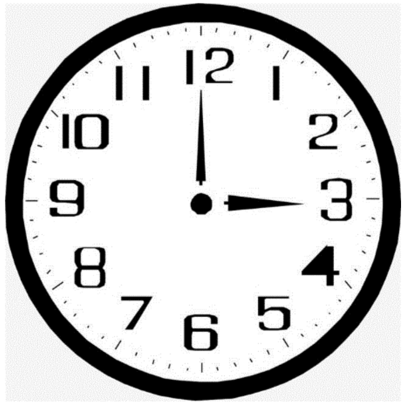
- 优点：简单、直观
- 缺点：做加减运算时，会出现如下问题：

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

- 能否只作加法?
  - 找到与负数等价的正数来代替这个负数，就可变减法为加法

补数表示法

- 小明从下午5点学习到凌晨3点，一共学了多少小时？
- 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？

$$\begin{array}{r} 1011 \\ -\ 1011 \\ \hline 0000 \end{array} \qquad \begin{array}{r} 1011 \\ +\ 0101 \\ \hline \boxed{1}0000 \end{array}$$

自然去掉

可见 -1011 与 $+0101$ 作用等价

记作 $-1011 \equiv +0101 \quad (\mathrm{mod}\ 2^4)$

同理 $-011 \equiv +101 \quad (\mathrm{mod}\ 2^3)$

$-0.1001 \equiv +1.0111 \quad (\mathrm{mod}\ 2^1)$

如何区分
正数和补数？

- 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
  - 一个正数和一个负数互为补数时，绝对值之和即为模数

补码表示法：二进制整数

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

其中：x 为真值，n 为二进制整数的位数

$x = +0101000$

$[x]_{补} = 0,0101000$

用 逗号 将符号位和数值部分隔开

$x = -1011000$

x的补数$= -1011000+2^7$

$= 0101000$

$[x]_{补} = 1,0101000$

2^7

检验上式为什么是2^{n+1}？

$[x]_{补} = 2^{7+1} +(-1011000)$

$= 100000000$

$-\quad 1011000$

$1,0101000$

补码表示法：二进制（纯）小数

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

其中：x 为真值

$x = +0.1110$

$[x]_{补} = 0.1110$

用 小数点 将符号位和数值部分隔开

$x = -0.1100000$

$[x]_{补} = -0.1100000+2$

$= 10.0000000$

$\quad - 0.1100000$

$= 1.0100000$

求补码的快捷方式

设 $x = -1010$ 时

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

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 $[x]_补 = 1.0001$，求 x。

解：由定义得 $x = [x]_补 - 2$

$= 1.0001 - 10.0000$

$= -0.1111$

或：$[x]_补 \to [x]_原$　　$[x]_原 = 1.1111$

$\therefore x = -0.1111$

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（需要记住）

补码除符号位外，末位减 1，再每位取反

练习：求下列真值的补码

真值	$[x]_{补}$	$[x]_{原}$
$x = -70 = -1000110$	$1,0111010$	$1,1000110$
$x = -0.1110$	1.0010	1.1110
$x = \boxed{+0.0000}$ $[+0]_{补} = [-0]_{补}$	$\boxed{0.0000}$	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 $[x]_{补} = \begin{cases} x & 1 > x \geq 0 \\ 2 + x & 0 > x \geq -1 \pmod 2 \end{cases}$

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

反码表示法：二进制整数

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

其中：x 为真值，n为二进制数位数

$x = +1101$

$[x]_{反} = 0,1101$

用 逗号 将符号位和数值部分隔开

$x = -1101$

$[x]_{反} = (2^{4+1}-1) - 1101$

$= 11111 - 1101$

$= 1,0010$

反码表示法：二进制小数

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

其中：x 为真值，n为二进制数位数

$$x = -0.1010$$

$$[x]_{反} = (2-2^{-4}) - 0.1010$$

$$= 1.1111 - 0.1010$$

$$= 1.0101$$

用 小数点 将符号位

和数值部分隔开

例子：已知反码求真值，0的反码

- 已知 $[x]_{反} = 1,1110$，求 x

  解：由定义得 $x = [x]_{反} - (2^{4+1} - 1)$

  $= 1,1110 – 11111$

  $= - 0001$

- 求 0 的反码

  解：设 $x = +0.0000$，$[+0.0000]_{反} = 0.0000$

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

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

  $[+ 0]_{反} \neq [-0]_{反}$

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
- 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$[y]_{补}$，求$[-y]_{补}$

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

<I>

$$\boxed{[y]_{补} = 0. y_1 y_2 \ldots y_n}$$

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

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

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

<II>

$$\boxed{[y]_{补} = 1. y_1 y_2 \cdots y_n}$$

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

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

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

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

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

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

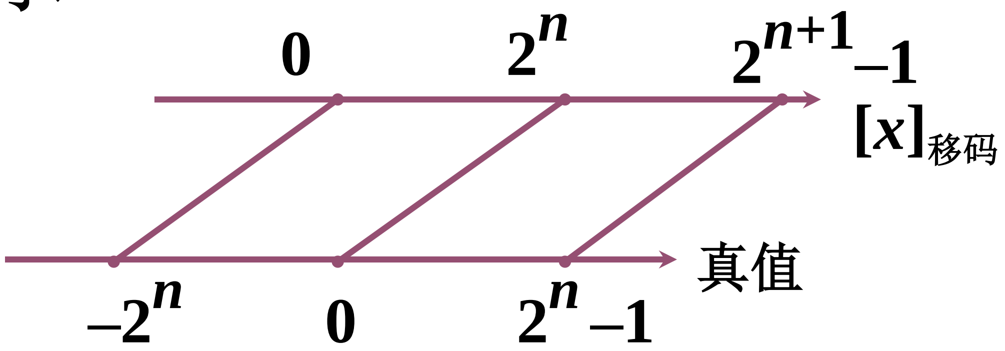
移码表示法：二进制整数

- 定义

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

其中：x 为真值，n 为 整数的位数

- 移码在数轴上的表示



- 例：

$x = 10100$

$[x]_{移} = 2^5 + 10100 = 1,10100$

$x = -10100$

$[x]_{移} = 2^5 - 10100 = 0,01100$

用 逗号 将符号位和数值位隔开

移码和补码的比较

设 $x = +1100100$

$[x]_{移} = 2^7 + 1100100 = 1,1100100$

$[x]_{补} = 0,1100100$

设 $x = -1100100$

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

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

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)	$[x]_{补}$	$[x]_{移}$	$[x]_{移}$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

$$[+0]_{移} = 2^5 + 0 = 1,00000$$

$$[-0]_{移} = 2^5 - 0 = 1,00000$$

$$[+0]_{移} = [-0]_{移}$$

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

用移码表示浮点数的阶码，便于判断浮点数的阶码大小

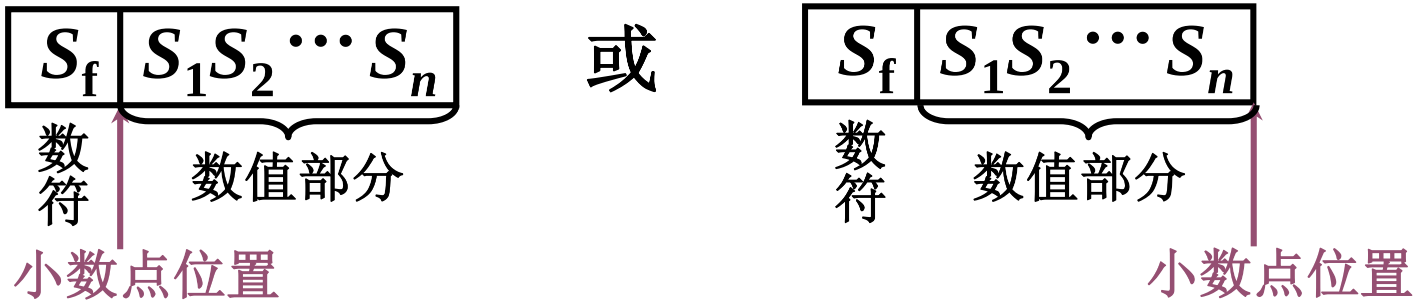
（接下来讲解）

第二章

- **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**
- 定点运算
- 浮点运算

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

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

$= 1.10101 \times 2^{1}$

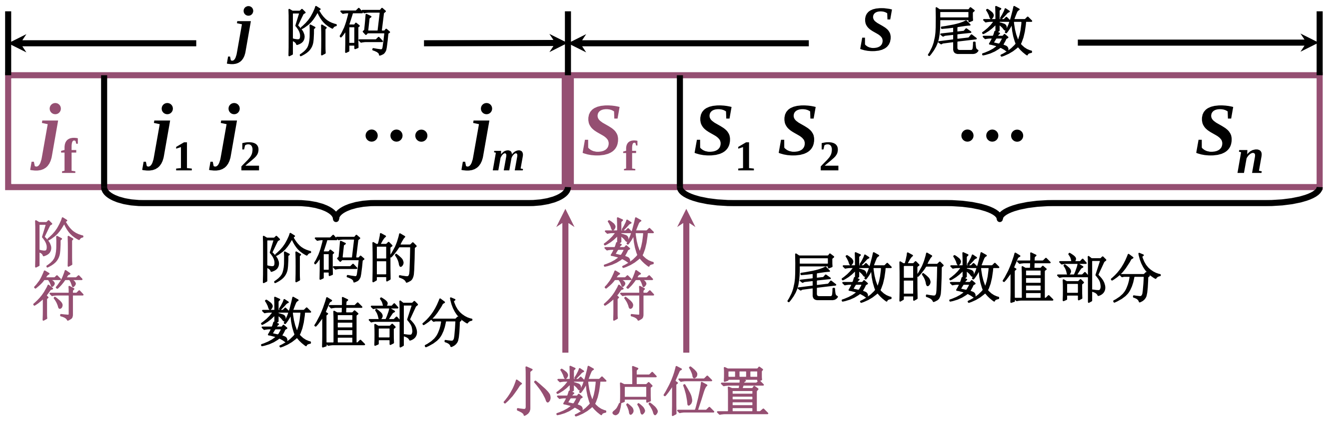
$= 1101.01 \times 2^{-10}$

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

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



S_f 代表浮点数的符号

n 其位数反映浮点数的精度

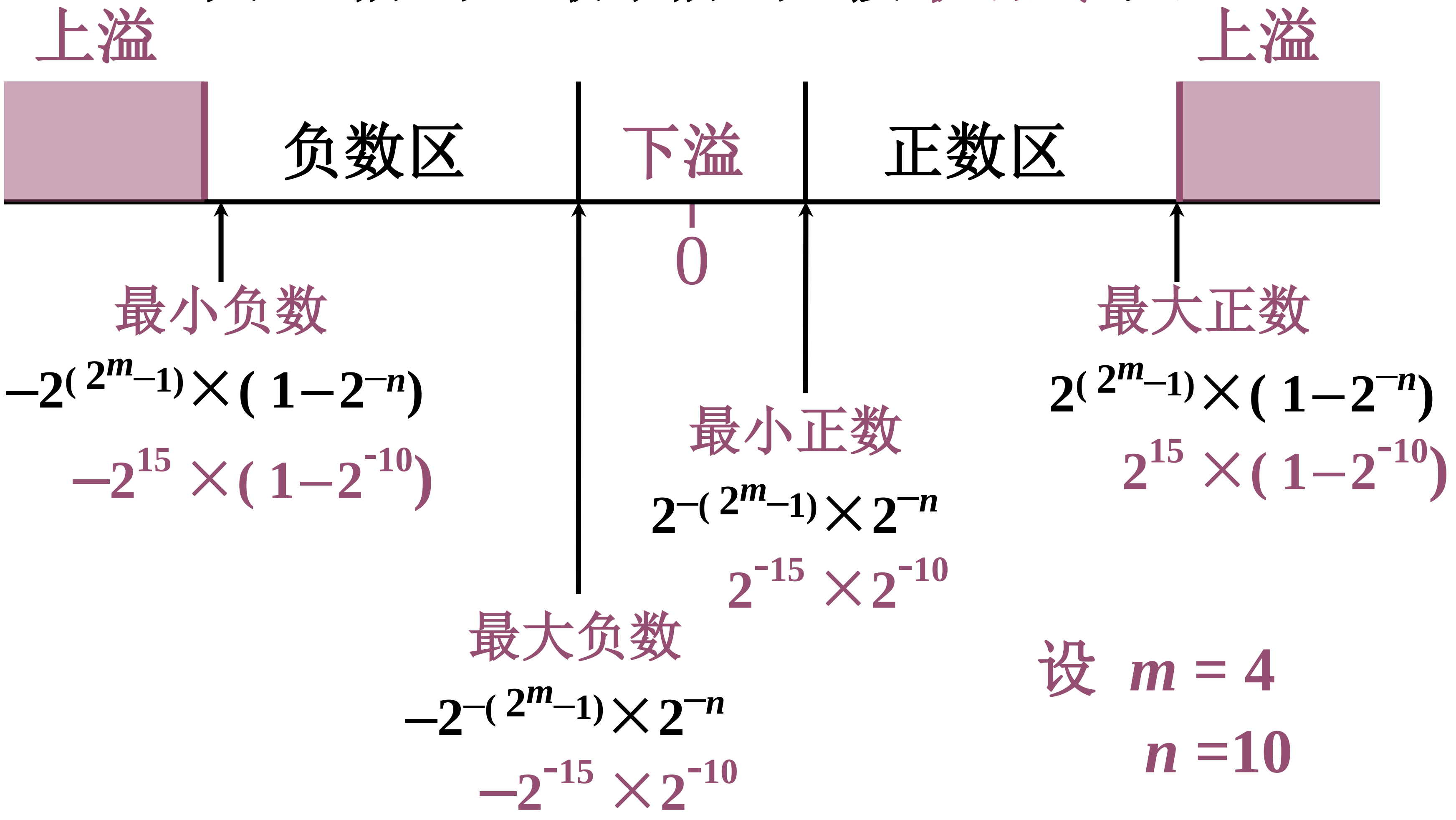
m 其位数反映浮点数的表示范围

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

$\therefore$ 15 位二进制数可反映 ±3 万之间的十进制数

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

$m = 4，5，6，\cdots$

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

- 例13. 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）

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

最小正数 $2^{-1111} \times 0.1\underbrace{000000000}_{9\text{个}0} = 2^{-15} \times 2^{-1} = 2^{-16}$

最大负数 $2^{-1111} \times (-0.1\underbrace{000000000}_{9\text{个}0}) = -2^{-15} \times 2^{-1} = -2^{-16}$

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

- 例14. 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 $[x]_{原} = [x]_{补} = [x]_{反} = 0.0010011000$

浮点机中 $[x]_{原} = 1, 0010;\ \ 0.1001100000$

$[x]_{补} = 1, 1110;\ \ 0.1001100000$

$[x]_{反} = 1, 1101;\ \ 0.1001100000$

- 例15. 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 $x = -58$

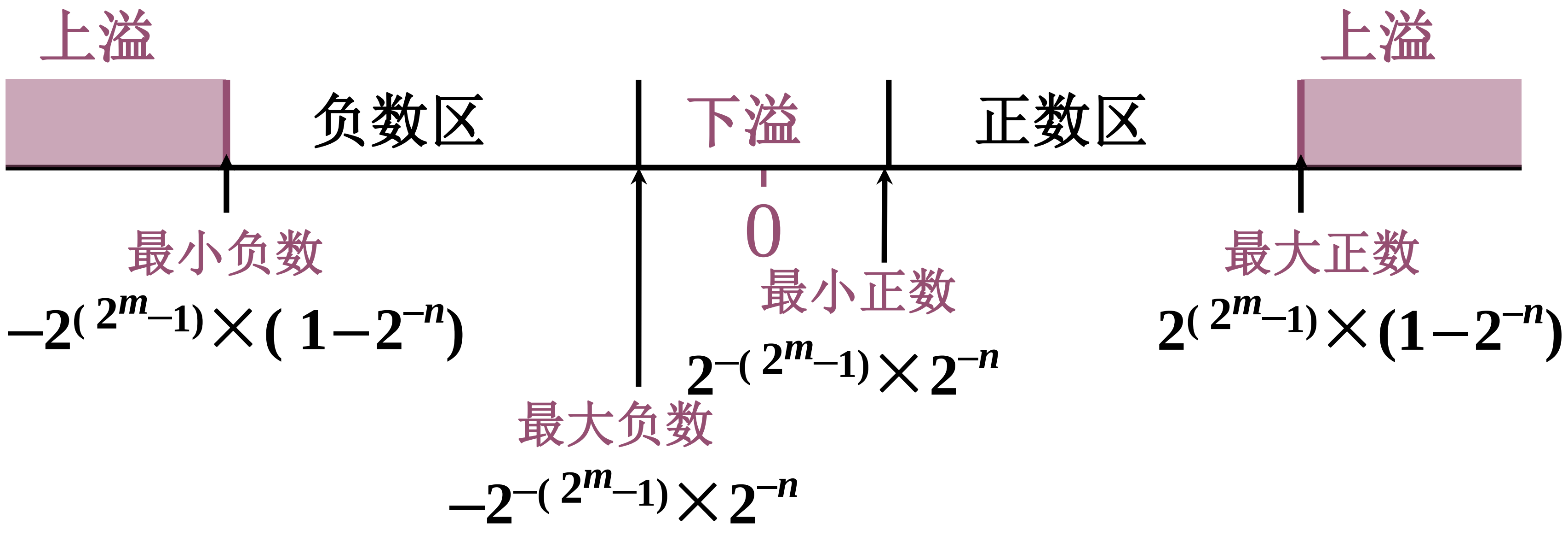
二进制形式 $x = -111010$

定点表示 $x = -0000111010$

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

定点机中	浮点机中
$[x]_{原} = 1, 0000111010$	$[x]_{原} = 0, 0110; 1.1110100000$
$[x]_{补} = 1, 1111000110$	$[x]_{补} = 0, 0110; 1.0001100000$
$[x]_{反} = 1, 1111000101$	$[x]_{反} = 0, 0110; 1.0001011111$
	$[x]_{阶移、尾补} = 1, 0110; 1.0001100000$

- 例16. 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 $m = 4 \qquad n = 10$

当阶码和尾数都用补码表示时，机器零为

×, × × × ×;　0. 0 0 ⋯ 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, 0 0 0 0;　0. 0 0 ⋯ 0

有利于机器中“判 0”电路的实现

IEEE 754 标准

S	阶码（含阶符）	尾　　数

数符　　　　　小数点位置

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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
s	8位指数（无符号数）								23位尾数（无符号数）						

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
23位尾数（无符号数）															

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 = $(-1)^S * (1 + 尾数_2) * 2^{(指数 - 127)}$
- 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 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 = $(-1)^S * (尾数_2) * 2^{(0 - 126)}$ （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 = $(-1)^S * (1 + 尾数_2) * 2^{(指数 - 127)}$
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 = $(-1)^S * (1 + 尾数_2) * 2^{(指数 - 127)}$
- 指数 - 127？
  - 使用**移码的思想**。二进制表示中的指数部分是**原码**，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 = $(-1)^S * (尾数_2) * 2^{(0 - 126)}$ （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 = $(-1)^S * (1 + 尾数_2) * 2^{(指数 - 127)}$
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 = $(-1)^S * (尾数_2) * 2^{(0 - 126)}$
- 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。

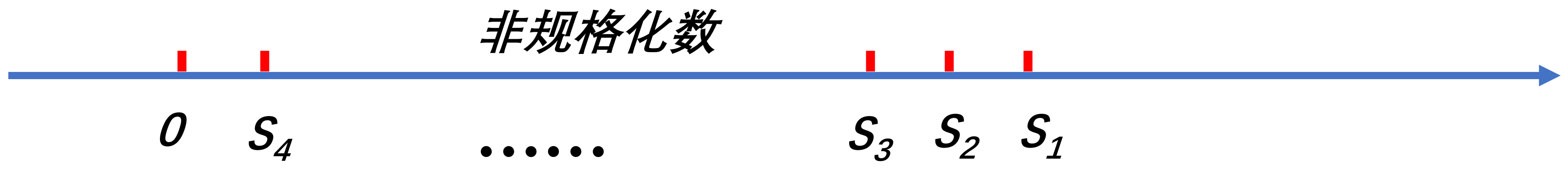
指数	尾数	表示对象	换算方法
0	0	0	**规定**
0	非0	正负非规格化数	正负非规格化数 = $(-1)^S * (尾数_2) * 2^{(0 - 126)}$ （S代表符号位，1为负数，0为正数）
[1: 254]	任意	正负浮点数	正负浮点数 = $(-1)^S * (1 + 尾数_2) * 2^{(指数 - 127)}$
255	0	正负无穷 (inf)	**规定**
255	非零	NaN	**规定**

IEEE 754浮点数：正负非规格化数

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

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

第二小正浮点数： $\boldsymbol{S_1} = (1 + 0.0\ldots01_2) * 2^{(1-127)} = \mathbf{2^{-126}} + \underline{\mathbf{2^{-149}}}$



最大非规格化数： $\boldsymbol{S_3}=0.1\ldots11_2*2^{(0-126)}=(1 - 2^{-23})*2^{-126} = \mathbf{2^{-126}} - \mathbf{2^{-149}}$

....

最小非规格化正数： $\boldsymbol{S_4}=0.0\ldots01_2*2^{(0-126)} = 2^{-23} * 2^{-126} = \underline{\mathbf{2^{-149}}}$

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

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

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

符号位：1；指数部分：126；尾数部分：0.1_2

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：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

$$(-1)^S * (1 + \text{尾数}_2) * 2^{(\text{指数} - 127)} = (-1)^1 * (1 + 0.25) * 2^{(129 - 127)}$$

$$= -1 * 1.25 * 2^2$$

$$= -5.0$$

第二章

- 计算机中数的表示
- **定点运算**
- 浮点运算

移位运算

- 移位的意义

15.m = 1500.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（重要）

符号位不变

<table>
<tr><th>真值</th><th>码　制</th><th>添补代码</th></tr>
<tr><td>正数</td><td>原码、补码、反码</td><td>0</td></tr>
<tr><td rowspan="4">负数</td><td>原　码</td><td>0</td></tr>
<tr><td rowspan="2">补　码</td><td>左移 添 0</td></tr>
<tr><td>右移 添 1</td></tr>
<tr><td>反　码</td><td>1</td></tr>
</table>

- 例17. 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 $[A]_{原} = [A]_{补} = [A]_{反} = 0,0011010$

移位操作	机器数 $[A]_{原}=[A]_{补}=[A]_{反}$	对应的真值
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18. 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

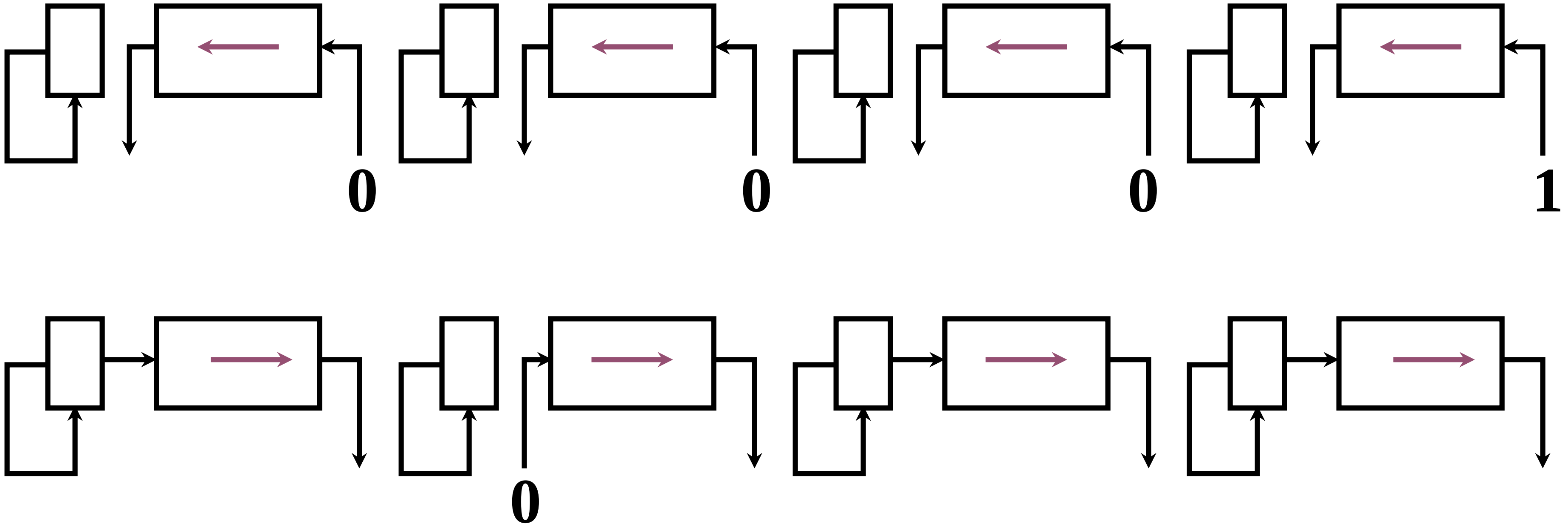
补码

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

3. 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

	(a)	(b)	(c)	(d)
←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

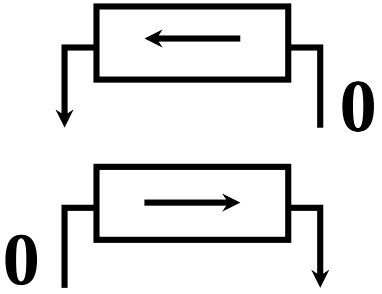
算术移位和逻辑移位的区别

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

加减法运算

- 补码加减运算公式

(1) 加法

整数 $[A]_{补} + [B]_{补} = [A+B]_{补}$（mod 2^{n+1}）

小数 $[A]_{补} + [B]_{补} = [A+B]_{补}$（mod 2）

(2) 减法

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

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

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

补码加减，连同符号位一起相加，符号位产生的进位自然丢掉

- 例19. 设 $A = 0.1011$，$B = -0.0101$，求 $[A+B]_{补}$

$$
\begin{array}{rcl}
\text{解：}\quad [A]_{补} & = & 0.1011 \\
+\,[B]_{补} & = & 1.1011 \\
\hline
[A]_{补}+[B]_{补} & = & \boxed{1}\,0.0110 = [A+B]_{补}
\end{array}
$$

$$\therefore A + B = 0.0110$$

验证

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

- 例20. 设 $A = -9$，$B = -5$，求 $[A+B]_{补}$

$$
\begin{array}{rcl}
\text{解：}\quad [A]_{补} & = & 1,0111 \\
+\,[B]_{补} & = & 1,1011 \\
\hline
[A]_{补}+[B]_{补} & = & \boxed{1}\,1,0010 = [A+B]_{补}
\end{array}
$$

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

验证

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

- 例21. 设机器数字长为 8 位（含 1 位符号位）且 A = 15，B = 24，用补码求 $A-B$

解：$A = 15 = 0001111$

$B = 24 = 0011000$

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

$$+\ [-B]_{补} = 1,1101000$$

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

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

练习 1　设 $x = \frac{9}{16}$ $y = \frac{11}{16}$，用补码求 $x+y$

$$x+y = -0.1100 = -\frac{12}{16}$$ 错

练习 2　设机器数字长为 8 位（含 1 位符号位）且 $A = -97$，$B = +41$，用补码求 $A-B$

$$A-B = +1110110 = +118$$ 错

一位符号位判溢出

- 一位符号位判溢出
  - 参加操作的**两个数**（减法时即为被减数和“求补”以后的减数）**符号相同，其结果的符号与原操作数的符号不同，即为溢出**
- 硬件实现
  - 最高有效位的进位⊕符号位的进位 ＝ 1，溢出

如

$$\left.\begin{array}{l} \mathbf{1 \oplus 0 = 1} \\ \mathbf{0 \oplus 1 = 1} \end{array}\right\} \text{有 溢出}$$

$$\left.\begin{array}{l} \mathbf{0 \oplus 0 = 0} \\ \mathbf{1 \oplus 1 = 0} \end{array}\right\} \text{无 溢出}$$

两位符号位判溢出

$$[x]_{补'} = \begin{cases} x & 1 > x \geq 0 \\ 4 + x & 0 > x \geq -1 \pmod 4 \end{cases}$$

$$[x]_{补'} + [y]_{补'} = [x + y]_{补'} \pmod 4$$

$$[x - y]_{补'} = [x]_{补'} + [-y]_{补'} \pmod 4$$

结果的双符号位 相同 未溢出 **00, ×××××**

11, ×××××

结果的双符号位 不同 溢出 **10, ×××××**

01, ×××××

最高符号位 代表其 真正的符号

乘法运算

- 分析笔算乘法

$A = -0.1101 \qquad B = 0.1011$

$A \times B = -0.10001111$ 乘积的符号心算求得

$$
\begin{array}{r}
0.1101 \\
\times 0.1011 \\
\hline
1101 \\
1101 \\
0000 \\
1101 \\
\hline
0.10001111
\end{array}
$$

✓ 符号位单独处理

✓ 乘数的某一位决定是否加被乘数

? 4个位积一起相加

✓ 乘积的位数扩大一倍

- 笔算乘法改进

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

$$= 0.1A + 0.00A + 0.001A + 0.0001A$$

$$= 0.1A + 0.00A + 0.001(A + 0.1A)$$

$$= 0.1A + 0.01[0 \cdot A + 0.1(A + 0.1A)]$$

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

右移一位

$$= \underbrace{\boxed{2^{-1}}\{A + 2^{-1}[0 \cdot A + \underbrace{2^{-1}(A + \underbrace{2^{-1}(\underbrace{A+0}_{①})}_{②})}_{③}]\}}_{⑧}$$

第一步　被乘数$A + 0$

第二步　右移 一 位，得新的部分积

第三步　部分积 + 被乘数

⋮

第八步　右移 一 位，得结果

- 改进后的笔算乘法过程（竖式）

部分积	乘数	说明
0.0000 +0.1101	1011	初态，部分积 = 0 乘数为 1，加被乘数
0.1101 0.0110 +0.1101	1101	→1，形成新的部分积 乘数为 1，加被乘数
1.0011 0.1001 +0.0000	1 1110	→ 1，形成新的部分积 乘数为 0，加 0
0.1001 0.0100 +0.1101	11 1111	→ 1，形成新的部分积 乘数为 1，加 被乘数
1.0001 0.1000	111 1111	→1，得结果

小结

- 乘法运算可用 **加和移位** 实现
  - $\boldsymbol{n}$ = 4，加 4 次，移 4 次

- 由乘数的末位决定被乘数是否与原部分积相加，然后->1**位形成新的部分积**，同时**乘数->1位**（末位移丢），空出高位存放部分积的低位。

- **被乘数只与部分积的高位相加**

  硬件： 3个寄存器，具有移位功能

  1个全加器

原码乘法

- 原码一位乘运算规则

以小数为例　　设$[x]_{原} = x_0. x_1x_2 \cdots x_n$

$$[y]_{原} = y_0. y_1y_2 \cdots y_n$$

$$[x \cdot y]_{原} = (x_0 \oplus y_0).(0. x_1x_2 \cdots x_n)(0.y_1y_2 \cdots y_n)$$

$$= (x_0 \oplus y_0). x^*y^*$$

式中　$x^* = 0. x_1x_2 \cdots x_n$　　为 x 的绝对值

$y^* = 0. y_1y_2 \cdots y_n$　　为 y 的绝对值

乘积的符号位单独处理 $x_0 \oplus y_0$

数值部分为绝对值相乘 $x^* \cdot y^*$

原码一位乘递推公式

$$x^* \cdot y^* = x^*(0.y_1 y_2 \ \dots \ y_n)$$

$$= x^*(y_1 2^{-1} + y_2 2^{-2} + \ \cdots \ + y_n 2^{-n})$$

$$= \underbrace{2^{-1}(y_1 x^* + 2^{-1}(y_2 x^* + \ \cdots \ \underbrace{2^{-1}(\overbrace{y_n x^* + 0}^{z_0})}_{z_1} \ \cdots))}_{z_n}$$

$$z_0 = 0$$

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

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

$$\vdots$$

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

- 例21. 已知 $\boldsymbol{x} = -0.1110$，$y = 0.1101$，求$[x \times y]_{原}$

解：第1步：写出原码、绝对值和符号位

$[x]_{原} = 1.1110$，$x^* = 0.1110$（为绝对值），$x_0 = 1$

$[y]_{原} = 0.1101$，$y^* = 0.1101$（为绝对值），$y_0 = 0$

第2步：$x^* \cdot y^*$计算过程，见下页PPT

第2步：$x^* \cdot y^*$计算过程

	部分积	乘数	说明
	0 . 0 0 0 0	1 1 0 $\underline{\underline{1}}$	部分积 初态 $z_0 = 0$
	+ 0 . 1 1 1 0		+ x^*
	0 . 1 1 1 0		
逻辑右移	0 . 0 1 1 1	0 1 1 $\underline{\underline{0}}$	$\longrightarrow$ 1，得 z_1
	+ 0 . 0 0 0 0		+ 0
	0 . 0 1 1 1	0	
逻辑右移	0 . 0 0 1 1	1 0 1 $\underline{\underline{1}}$	$\longrightarrow$ 1，得 z_2
	+ 0 . 1 1 1 0		+ x^*
	1 . 0 0 0 1	1 0	
逻辑右移	0 . 1 0 0 0	1 1 0 $\underline{\underline{1}}$	$\longrightarrow$ 1，得 z_3
	+ 0 . 1 1 1 0		+ x^*
	1 . 0 1 1 0	1 1 0	
逻辑右移	0 . 1 0 1 1	0 1 1 0	$\longrightarrow$ 1，得 z_4

- **第3步：计算结果**

  ① 乘积的符号位 $x_0 \oplus y_0 = 1 \oplus 0 = 1$

  ② 数值部分按绝对值相乘

$$x^* \cdot y^* = 0.\,1\,0\,1\,1\,0\,1\,1\,0$$

$$\text{则 } [x \cdot y]_{\text{原}} = 1.\,1\,0\,1\,1\,0\,1\,1\,0$$

- 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

补码乘法

- 补码一位乘运算规则

以小数为例　设 被乘数 $[x]_补 = x_0. x_1x_2 \cdots x_n$

乘数 $[y]_补 = y_0. y_1y_2 \cdots y_n$

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$[y]_补$，去掉符号位，操作同 ①

最后 加$[-x]_补$，校正

Booth 算法（被乘数、乘数符号任意）

设$[x]_补 = x_0.x_1x_2 \cdots x_n$　$[y]_补 = y_0.y_1y_2 \cdots y_n$

$[x \cdot y]_补$

$-[x]_补 = +[-x]_补$

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

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

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

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

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

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

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

附加位 y_{n+1}

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

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

Booth 算法递推公式

$[z_0]_补 = 0$

$[z_1]_补 = 2^{-1}\{(y_{n+1} - y_n)[x]_补 + [z_0]_补\}$　　$y_{n+1} = 0$

$\vdots$

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

$[x \cdot y]_补 = [z_n]_补 + (y_1 - y_0)[x]_补$　　最后一步不移位

如何实现

$y_{i+1} - y_i$ ？

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

- 例22. 已知 $x = +0.0011$，$y = -0.1011$，求 $[x \times y]_{补}$

解:	00.0000	1.0101	0	
	+11.1101			$+[-x]_{补}$
	11.1101			
补码右移	11.1110	1 1010	1	→1
	+00.0011			$+[x]_{补}$
	00.0001	1		
补码右移	00.0000	11 101	0	→1
	+11.1101			$+[-x]_{补}$
	11.1101	11		
补码右移	11.1110	111 10	1	→1
	+00.0011			$+[x]_{补}$
	00.0001	111		
补码右移	00.0000	1111 1	0	→1
	+11.1101			$+[-x]_{补}$
	11.1101	1111		最后一步不移位

$[x]_{补} = 0.0011$

$[y]_{补} = 1.0101$

$[-x]_{补} = 1.1101$

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

- 例23. 已知 $[x]_{补} = 1.0101$， $[y]_{补} = 1.0011$， 求$[x \times y]_{补}$

解：

	0 0 . 0 0 0 0	1 . 0 0 1 1	0	
	+ 0 0 . 1 0 1 1			$+[-x]_{补}$
	0 0 . 1 0 1 1			
补码右移	0 0 . 0 1 0 1	1 1 0 0 1	1	→1
补码右移	0 0 . 0 0 1 0	1 1 1 0 0	1	→1
	+ 1 1 . 0 1 0 1			$+[x]_{补}$
	1 1 . 0 1 1 1	1 1		
补码右移	1 1 . 1 0 1 1	1 1 1 1 0	0	→1
补码右移	1 1 . 1 1 0 1	1 1 1 1 1	0	→1
	+ 0 0 . 1 0 1 1			$+[-x]_{补}$
	0 0 . 1 0 0 0	1 1 1 1		最后一步不移位

$[-x]_{补} = 0.1011$

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

乘法小结

- **整数乘法**与**小数乘法**完全相同
  - 可用 **逗号** 代替小数点

- 原码乘：符号位 **单独处理**
  补码乘：符号位 **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

除法运算

- 分析笔算除法

$$x = -0.1011 \quad y = 0.1101 \quad 求\ x \div y$$

```
           0.1101
0.1101 ) 0.10110
         0.01101
         0.010010
         0.001101
         0.00010100
         0.00001101
         0.00000111
```

✓商符单独处理

? 心算上商

? 余数不动低位补“0”
减右移一位的除数

? 上商位置不固定

商的真值符号取决于x_0与y_0 异或

余数的真值符号取决于被除数符号

$x \div y = -0.1101$ 商符心算求得

余数 -0.00000111

笔算除法和机器除法的比较

笔算除法	机器除法
商符单独处理	符号位异或形成
心算上商	$\|x\|-\|y\|>0$ 上商 1
	$\|x\|-\|y\|<0$ 上商 0
余数 不动 低位补 “0” 减右移一位 的除数	余数 左移一位 低位补 “0” 减 除数
2 倍字长加法器	1 倍字长加法器
上商位置 不固定	在寄存器 最末位上商

原码除法

- 以小数为例

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

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

$$[\frac{x}{y}]_{原} = (x_0 \oplus y_0).\ \frac{x^*}{y^*}$$

式中 $x^* = 0.\ x_1x_2 \cdots \ x_n$ 为 x 的绝对值

$y^* = 0.\ y_1y_2 \cdots \ y_n$ 为 y 的绝对值

商的符号位单独处理 $x_0 \oplus y_0$

数值部分为绝对值相除 $\frac{x^*}{y^*}$

约定

小数定点除法 $x^* < y^*$ 整数定点除法 $x^* > y^*$

被除数不等于 0

除数不能为 0

恢复余数法

- 例23. $x = -0.1011$，$y = -0.1101$，求$[\frac{x}{y}]_{原}$

$[x]_{原} = 1.1011$ $[y]_{原} = 1.1101$ $[y^*]_{补} = 0.1101$ $[-y^*]_{补} = 1.0011$

① $x_0 \oplus y_0 = 1 \oplus 1 = 0$

②

被除数（余数）	商	说 明
$\boxed{0.1011}$	0.0000	
+ 1.0011		$+[-y^*]_{补}$
1.1110	0	余数为负，上商 0
+ 0.1101		恢复余数 $+[y^*]_{补}$
$\boxed{0.1011}$	0	恢复后的余数
1.0110	0	$\leftarrow 1$
+ 1.0011		$+[-y^*]_{补}$
0.1001	01	余数为正，上商 1
1.0010	01	$\leftarrow 1$
+ 1.0011		$+[-y^*]_{补}$

逻辑左移

逻辑左移

被除数（余数）	商	说　明
0.0101	011	余数为正，上商 1
逻辑左移 0.1010	011	← 1
+ 1.0011		$+[-y^*]_{补}$
1.1101	0110	余数为负，上商 0
+ 0.1101		恢复余数 $+[y^*]_{补}$
0.1010	0110	恢复后的余数
逻辑左移 1.0100	0110	← 1
+ 1.0011		$+[-y^*]_{补}$
0.0111	01101	余数为正，上商 1

$$\frac{x^*}{y^*} = 0.1101$$

$$\therefore \left[\frac{x}{y}\right]_{原} = 0.1101$$

余数为正　上商 1

余数为负　上商 0，恢复余数

如果题目要求写出x/y的真值，则：

真值：$\frac{x}{y} = +0.1101$，余 -0.0111×2^{-4}

上商 5 次

第一次上商判溢出

移 4 次

在恢复余数法中，每当余数为负时，都需恢复余数，这就延长了机器除法的时间，操作也很不规则，对线路结构不利。加减交替法可克服这些缺点。

不恢复余数法（加减交替法）

- 恢复余数法运算规则

余数 $R_i>0$ 上商 "1"， $2R_i-y^*$

余数 $R_i<0$ 上商 "0"， R_i+y^* 恢复余数

$$2(R_i+y^*)-y^*=2R_i+y^*$$

- 不恢复余数法运算规则

上商 "1" $2R_i-y^*$

上商 "0" $2R_i+y^*$

加减交替

• 例24. $x = -0.1011$，$y = -0.1101$，求 $\left[\frac{x}{y}\right]_{原}$

解：

	被除数（余数）	商	说明
	0.1011	0.0000	
	+1.0011		$+[-y^*]_{补}$
	1.1110	0	余数为负，上商 0
逻辑左移	1.1100	0	←1
	+0.1101		$+[y^*]_{补}$
	0.1001	01	余数为正，上商 1
逻辑左移	1.0010	01	←1
	+1.0011		$+[-y^*]_{补}$
	0.0101	011	余数为正，上商 1
逻辑左移	0.1010	011	←1
	+1.0011		$+[-y^*]_{补}$
	1.1101	0110	余数为负，上商 0
逻辑左移	1.1010	0110	←1
	+0.1101		$+[y^*]_{补}$
	0.0111	01101	余数为正，上商 1

$[x]_{原} = 1.1011$

$[y]_{原} = 1.1101$

$[x^*]_{补} = 0.1011$

$[y^*]_{补} = 0.1101$

$[-y^*]_{补} = 1.0011$

- 例24. 结果

① $x_0 \oplus y_0 = 1 \oplus 1 = 0$

② $\frac{x^*}{y^*} = 0.1101$

$\therefore\ [\frac{x}{y}]_{原} = 0.1101$

如果题目要求写出x/y的真值，则：

真值：$\frac{x}{y} = +0.1101$，余 -0.0111×2^{-4}

商的真值符号取决于x_0与y_0 异或

余数的真值符号取决于被除数符号

特点 上商 n+1 次

第一次上商判溢出

移位 n 次，加 法n+1 次

用移位的次数判断除法是否结束

原码一位除——加减交替法（n+2次加法情况）

作业题4：X=-0.10101 Y=0.11011，用原码加减交替法计算 X÷Y

解：

①

$[x]_{原} = 1.10101$

$[y]_{原} = 0.11011$

$[x^*]_{补} = 0.10101$

$[y^*]_{补} = 0.11011$

$[-y^*]_{补} = 1.00101$

②

	被除数（余数）	商	说明
	0.10101	0.00000	
	+1.00101		$+[-y^*]_{补}$
	1.11010	0	余数为负，上商 0
逻辑左移	1.10100	0	← 1
	+0.11011		$+[y^*]_{补}$
	0.01111	01	余数为正，上商 1
逻辑左移	0.11110	01	← 1
	+1.00101		$+[-y^*]_{补}$
	0.00011	011	余数为正，上商 1
逻辑左移	0.00110	011	← 1
	+1.00101		$+[-y^*]_{补}$
	1.01011	0110	余数为负，上商 0
逻辑左移	0.10110	0110	← 1
	+0.11011		$+[y^*]_{补}$
	1.10001	01100	余数为负，上商0
逻辑左移	1.00010	01100	← 1
	+0.11011		$+[y^*]_{补}$
	1.11101	011000	余数为负，上商 0
	+0.11011		$+[y^*]_{补}$
	0. 11000		

最后，
当余数是负数，
需要加$[y^*]_{补}$
此时不需要移位

注意

商的真值符号取决于x_0与y_0 异或

余数的真值符号取决于被除数符号

③

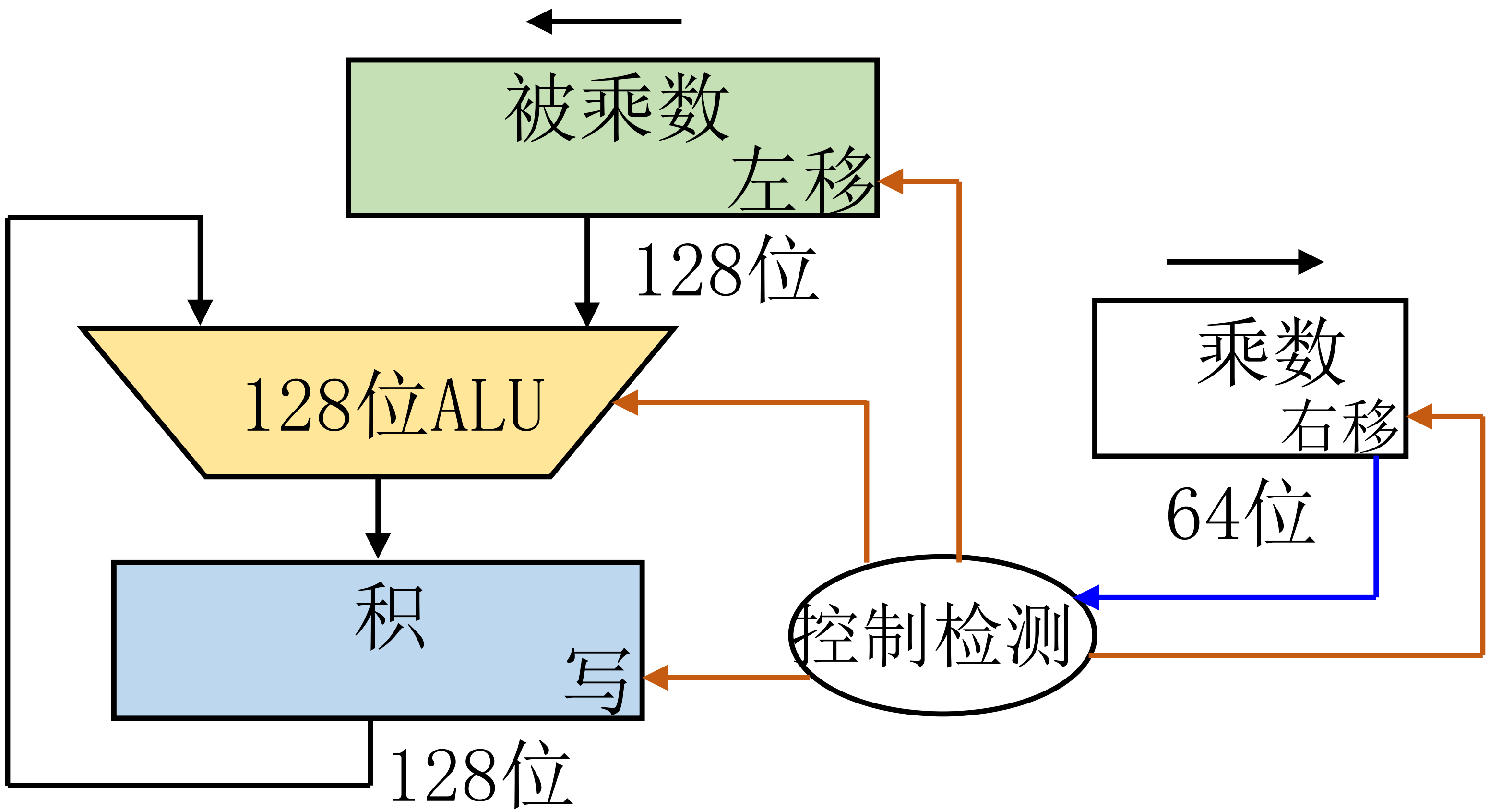
$x_0 \oplus y_0 = 1 \oplus 0 = 1$

$[\frac{x}{y}]_{原}$=1.11000

$\frac{x}{y} = -0.11000$

余 -0.11000×2^{-5}

乘法器硬件



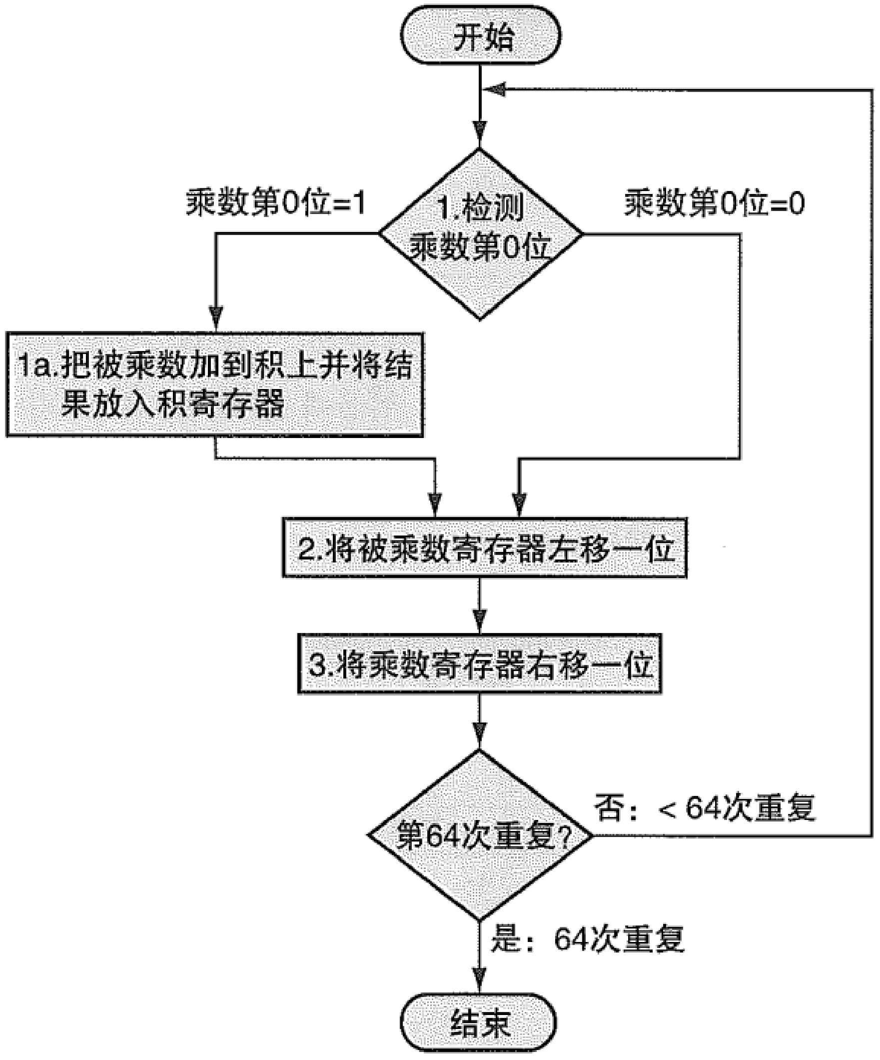
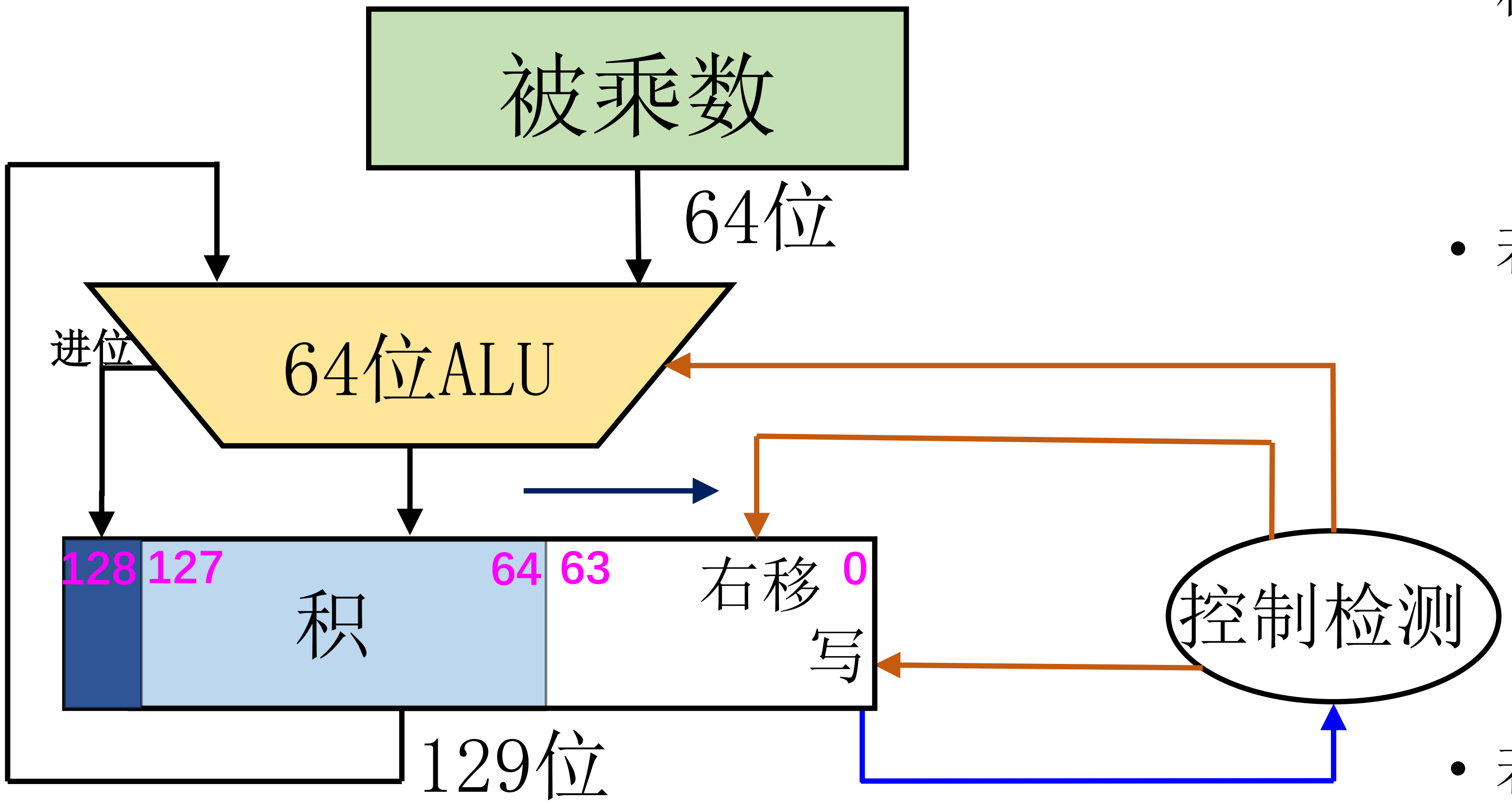


图 3-4　第一种乘法算法，采用了图 3-3 所示的硬件。如果乘数的最低有效位为 1，则将乘数加到积上。如果不是，则执行下一步。在下两步中将被乘数左移和乘数右移。将这三个步骤重复 64 次

- 被乘数寄存器128位：
  - 被乘数64位，要进行64次左移一位
- 存在的问题：被乘数寄存器存储空间浪费、128位ALU浪费

改良版乘法器硬件



- 积寄存器129位
  - 右64位：乘数
  - 左65位：全0，多的一位用于保存加法器的进位
- 若乘数最右端为1
  - 将积寄存器[128:65]位取出（时间不计）
  - 取出的值和被乘数进行加法运算；同时，积寄存器进行右移一位运算（一个时钟周期）
  - 加法运算结果写入积寄存器[128:64]（时间不计）
- 若乘数最右端为0
  - 积寄存器整体右移一位
- 最后结果在积寄存器[128:1]

第二章

- 计算机中数的表示
- 定点运算
- **浮点运算**

6.4 浮点四则运算

- 一、浮点加减运算

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

1. 对阶

(1) 求阶差

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

(2) 对阶原则

小阶向大阶看齐

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

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

1. 对阶

① 求阶差　$[\Delta j]_补 = [j_x]_补 - [j_y]_补$

$$
\begin{array}{rr}
 & 00, 01 \\
+ & 11, 01 \\
\hline
 & 11, 10
\end{array}
$$

阶差为负（–2）　　$\therefore S_x \longrightarrow 2 \quad j_x + 2$

② 对阶　　$[x]_{补'} = 00, 11; 00.0011$

2. 尾数求和

$$
\begin{array}{rll}
 & [S_x]_{补'} & = 00.0011 \\
+ & [S_y]_补 & = 11.0110 \\
\hline
 & & 11.1001
\end{array}
$$

对阶后的$[S_x]_{补'}$

$\therefore [x+y]_补 = 00, 11; 11.1001$

3. 规格化

- (1) 规格化数的定义

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

- (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	$0.\boxed{1}\times\times\cdots\times$	原码	$1.\boxed{1}\times\times\cdots\times$
补码	$\boxed{0.1}\times\times\cdots\times$	补码	$\boxed{1.0}\times\times\cdots\times$
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 一数位不同

特例

$$S = -\frac{1}{2} = -0.100\cdots0$$

$$[S]_{原} = 1.100\cdots0$$

$$[S]_{补} = \boxed{1.1}00\cdots0$$

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

$$S = -1$$

$$[S]_{补} = \boxed{1.0}00\cdots0$$

$\therefore\ [-1]_{补}$ 是规格化的数

- (3)左规

尾数左移一位，阶码减 1，直到数符和第一数位不同为止

上例 $[x+y]_{补} = 00, 11; 11.1001$

左规后 $[x+y]_{补} = 00, 10; 11.0010$

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

- (4)右规

当 尾数溢出（ >1）时，需 右规

即尾数出现 01. ×× …×或 10. ×× …×时

尾数右移一位，阶码加 1

- 例27. $x = 0.1101 \times 2^{10}$，$y = 0.1011 \times 2^{01}$，求 $x+y$（除阶符、数符外，阶码取 3 位，尾数取 6 位）

解：
$[x]_补 = 00, 010; 00.110100$
$[y]_补 = 00, 001; 00.101100$

① 对阶

$$[\Delta j]_补 = [j_x]_补 - [j_y]_补 \quad \begin{array}{r} =\ 00, 010 \\ +\ 11, 111 \\ \hline \boxed{1}00, 001 \end{array}$$

阶差为 +1 $\qquad \therefore S_y \longrightarrow 1,\ j_y+1$

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

② 尾数求和

$$\begin{array}{rl} [S_x]_补 & = 00.110100 \\ +\ [S_y]_{补'} & = 00.010110 \\ \hline & \ \ \ 01.001010 \end{array}$$

对阶后的$[S_y]_{补'}$

尾数溢出需右规

③ 右规

$$[x+y]_{补} = 00, 010; 01.001010$$

右规后

$$[x+y]_{补} = 00, 011; 00.100101$$

$$\therefore\ x+y = 0.100101 \times 2^{11}$$

- 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

• 原例13延展. 设 $m=4$，$n=10$，$r=2$，求尾数规格化后的浮点数表示范围（阶码与尾数都是补码表示）

最大正数 $2^{01111}\times 0.\underbrace{1111111111}_{10\text{个}1} = 2^{15}\times(1-2^{-10})$

最小正数 $2^{10000}\times 0.1000000000 = 2^{-16}\times 2^{-1} = 2^{-17}$

最大负数 $2^{10000}\times(-0.1000000001) = -2^{-16}\times(2^{-1}+2^{-10})$

最小负数 $2^{01111}\times(-1.\underbrace{0000000000}_{10\text{个}0}) = -2^{15}\times 1$

- 例28. $x = (-\frac{5}{8}) \times 2^{-5}$，$y = (\frac{7}{8}) \times 2^{-4}$，求 $x - y$（除阶符、数符外，阶码取 3 位，尾数取 6 位）

解： $x = (-0.101000) \times 2^{-101}$ $y = (0.111000) \times 2^{-100}$

$[x]_补 = 11, 011; 11.011000$ $[y]_补 = 11, 100; 00.111000$

① 对阶

$$[\Delta j]_补 = [j_x]_补 - [j_y]_补 \begin{array}{r} = \ 11, 011 \\ + \ 00, 100 \\ \hline 11, 111 \end{array}$$

阶差为 -1 $\therefore S_x \longrightarrow 1, \quad j_x + 1$

$\therefore [x]_{补'} = 11, 100; 11.101100$

② 尾数求和

$$
\begin{array}{rr}
 & [S_x]_{补'} = 11.\ 101100 \\
+ & [-S_y]_{补} = 11.\ 001000 \\
\hline
 & \boxed{1}10.\ 110100
\end{array}
$$

③ 右规

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

右规后

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

$$\therefore\ x - y = (-0.100110) \times 2^{-11}$$

$$= (-\frac{19}{32}) \times 2^{-3}$$

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