数值运算

码的表示及范围

无符号数

依靠寄存器位数表达范围
0-2^n-1

正数

- 数值部分一样
- 整数前加**0，** ，小数前加**0.** 。
- [-x]补：[x]补码连同符号位在内，每位取反，末位加一

负数

原码

- 数值部分一样
- 整数前加 **1，** ，小数前加 **1.**

补码

- 原码除符号位以外，每位取反，末位加一
- [-x]补：[x]补码连同符号位在内，每位取反，末位加一

反码

- 原码除符号位以外，每位取反，末位**不加一**

移位

规则如下：

浮点数

- 表示形式

$N = S \times r^j$　　浮点数的一般形式

S 尾数　j 阶码　r 尾数的基值

计算机中 r 取 2、4、8、16 等

当 $r = 2$　　$N = 11.0101$

✓ $= 0.110101 \times 2^{10}$　二进制表示，10 规格化数

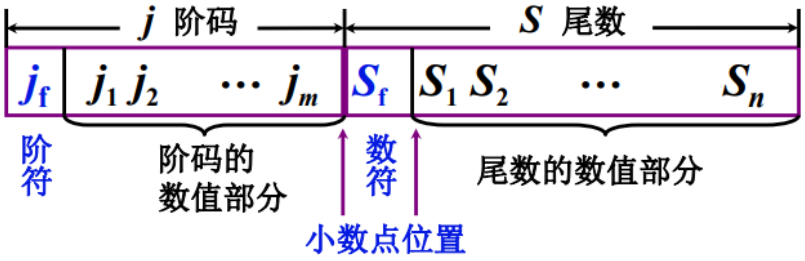
$= 1.10101 \times 2^{1}$

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

✓ $= 0.00110101 \times 2^{100}$

计算机中　S　小数、可正可负

- 各部分含义



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

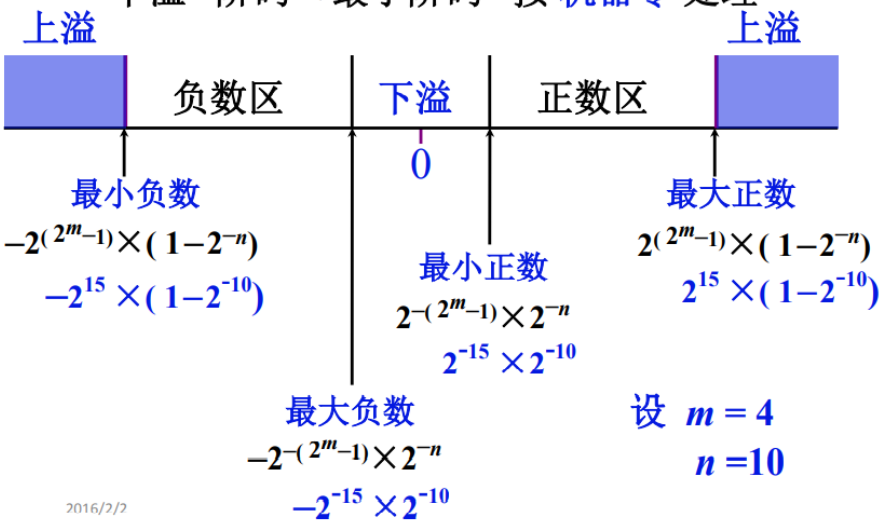
- 表示范围
  - 最小负数：[1-2^(-n)] × [-2^(2^m-1)]
  - 最大负数：[2^(-n)] × [-2^-(2^m-1)]
  - 最小正数：[2^(-n)] × [2^-(2^m-1)]

- 最大正数：[1-2^(-n)] × [2^(2^m-1)]

2. 浮点数的表示范围 6.2

上溢　阶码 > 最大阶码

下溢　阶码 < 最小阶码　按 机器零 处理



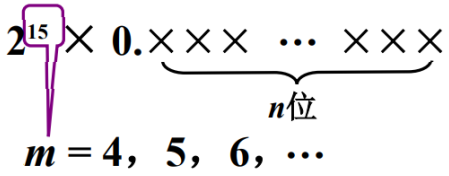
2016/2/2

- 实际样例

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：　∵　$2^{14}=16384$　　$2^{15}=32768$

∴ 如果是定点数15 位二进制数可反映 ±3 万之间的十进制数



满足 最大精度 可取 $m=4$，$n=18$

- 规格化形式及方法

保证数据精度，有效位数多

1为真值

2位为2进制两位，也就是4进制1位

3. 浮点数的规格化形式

6.2

$r=2$　　尾数最高位为 1

$r=4$　　尾数最高 2 位不全为 0　　基数不同，浮点数的

$r=8$　　尾数最高 3 位不全为 0　　规格化形式不同

4. 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

- 规格化样例

例如：设 $m=4$，$n=10$，$r=2$

6.2

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

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

乘法

原码二位乘

运算规则

表 6.11　原码两位乘的运算规则

乘数判断位 $y_{n-1}y_n$	标志位 C_j	操 作 内 容
0 0	0	$z \to 2$ 位，$y^* \to 2$ 位，C_j 保持“0”
0 1	0	$z+x^* \to 2$ 位，$y^* \to 2$ 位，C_j 保持“0”
1 0	0	$z+2x^* \to 2$ 位，$y^* \to 2$ 位，C_j 保持“0”
1 1	0	$z-x^* \to 2$ 位，$y^* \to 2$ 位，C_j 置“1”
0 0	1	$z+x^* \to 2$ 位，$y^* \to 2$ 位，C_j 置“0”
0 1	1	$z+2x^* \to 2$ 位，$y^* \to 2$ 位，C_j 置“0”
1 0	1	$z-x^* \to 2$ 位，$y^* \to 2$ 位，C_j 保持“1”
1 1	1	$z \to 2$ 位，$y^* \to 2$ 位，C_j 保持“1”

表中 z 表示原有部分积，x^* 表示被乘数的绝对值，y^* 表示乘数的绝对值，$\to 2$ 表示右移两位，当进行 $-x^*$ 运算时，一般都采用加 $[-x^*]_{补}$ 来实现。这样，参与原码两位乘运算的操作数是绝对值的补码，因此运算中右移两位的操作也必须按补码右移规则完成。尤其应注意的是，乘法过程中可能要加 2 倍被乘数，即 $+[2x^*]_{补}$，使部分积的绝对值大于 2。为此，只有对部分积取 3 位符号位，且以最高符号位作为真正的符号位，才能保证运算过程正确无误。

注意事项

- 部分积三位符号位，且最高位为真正符号位；乘数两位符号位
- **乘数y默认取正，符号单独计算**
- **仅考虑乘数和被乘数的数值部分**
- 求2x时，如符号位要进位，需保留符号位进位
- **符号位不参与计算**
- 移位遵循补码规则
- 乘数数值部分偶数位，移n/2次，奇数位，高位前补1个0，最后一步移1位

具体算例

例 6.18 设 $x=0.111111, y=-0.111001$，用原码两位乘求 $[x\cdot y]_{原}$。

解：① 数值部分的计算如表 6.12 所示，其中

$$x^*=0.111111,[-x^*]_{补}=1.000001,2x^*=1.111110,y^*=0.111001$$

表 6.12 例 6.18 原码两位乘数值部分的运算过程

部分积	乘数 y^*	C_j	说明
000.000000 +000.111111	00111001	0	开始，部分积为 0，$C_j=0$ 根据 $y_{n-1}y_nC_j=010$，加 x^*，保持 $C_j=0$
000.111111 000.001111 +001.111110	11001110	0	→2 位，得新的部分积，乘数同时→2 位 根据“100”加 $2x^*$，保持 $C_j=0$
010.001101 000.100011 +111.000001	11 01110011	0	→2 位，得新的部分积，乘数同时→2 位 根据“110”减 x^*（即加 $[-x^*]_{补}$），C_j 置“1”
111.100100 111.111001 +000.111111	0111 00011100	1	→2 位，得新的部分积，乘数同时→2 位 根据“001”加 x^*，C_j 置“0”
000.111000	000111		形成最终结果

② 乘积符号的确定：

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

故 $[x\cdot y]_{原}=1.111000000111$

不难理解，当乘数为偶数时，需做 $n/2$ 次移位，最多做 $n/2+1$ 次加法。当乘数为奇数时，乘数高位前可只增加一个“0”，此时需做 $n/2+1$ 次移位（最后一步移一位），最多需做 $n/2+1$ 次加法。

补码一位乘

补一运算规则

y_iy_{i+1}	$y_{i+1}-y_i$	操作
00	0	部分积右移一位
01	1	部分积加 $[x]_{补}$，再右移一位
10	-1	部分积加 $[-x]_{补}$，再右移一位
11	0	部分积右移一位

应该注意的是，按比较法进行补码乘法时，像补码加、减法一样，符号位也一起参加运算。

补一注意事项

- 部分积两位符号位，且最高位为真正符号位；乘数一位符号位
- 求2x时，如符号位要进位，需保留符号位进位
- 最后一步**不需要移位**
- 符号位**参与计算**
- 移位遵循补码规则
- 求解结果为补码形式

补一具体算例

例 6.21 已知$[x]_{补}=0.1101$，$[y]_{补}=0.1011$，求$[x\cdot y]_{补}$。

解：表 6.16 列出了例 6.21 的求解过程。

表 6.16 例 6.21 求$[x\cdot y]_{补}$的过程

部分积	乘数y_n	附加位y_{n+1}	说明
00.0000 +11.0011	01011	0	初值$[z_0]_{补}=0$ $y_ny_{n+1}=10$，部分积加$[-x]_{补}$
11.0011 11.1001 11.1100 +00.1101	10101 11010	1 1	→1 位，得$[z_1]_{补}$ $y_ny_{n+1}=11$，部分积→1 位，得$[z_2]_{补}$ $y_ny_{n+1}=01$，部分积加$[x]_{补}$
00.1001 00.0100 +11.0011	11 11101	0	→1 位，得$[z_3]_{补}$ $y_ny_{n+1}=10$，部分积加$[-x]_{补}$
11.0111 11.1011 +00.1101	111 11110	1	→1 位，得$[z_4]_{补}$ $y_ny_{n+1}=01$，部分积加$[x]_{补}$
00.1000	1111		最后一步不移位，得$[x\cdot y]_{补}$

故 $[x\cdot y]_{补}=0.10001111$

由表 6.16 可见，与校正法（参见表 6.13 和表 6.14）相比，Booth 算法的部分积仍取双符号位，乘数因符号位参加运算，故多取 1 位。