第 6 章　复杂算术操作

(一) 单精度定点数乘法

1.原码定点一位乘法

(1) 人工算法与机器算法的同异性

■　在定点计算机中，两个原码表示的数相乘的运算规则是：乘积的符号位由两数的符号位按异或运算得到，而乘积的数值部分则是两个正数相乘之积。

■　设 n 位被乘数和乘数用定点小数表示(定点整数也同样适用)

被乘数　$[x]_{原} = x_f . x_{n-1} \ldots x_1 x_0$

乘数　$[y]_{原} = y_f . y_{n-1} \ldots y_1 y_0$

则乘积　$[z]_{原} = (x_f \oplus y_f) + (0. x_{n-1} \ldots x_1 x_0)(0. y_{n-1} \ldots y_1 y_0)$

式中，x_f 为被乘数符号，y_f 为乘数符号。

例.　0.1101×1.1011

```
手算           0.1101
        ×      0.1011
      ----------------
                 1101
                1101
部分积         0000
      +       1101
      ----------------
            0.10001111
上符号：    1.10001111
```

问题：

① 加数增多(由乘数位数决定)。

② 加数的位数增多（与被乘数、乘数位数有关）

改进：将一次相加改为分步累加。

(2)机器分步乘法

每次将一位乘数所对应的部分积与原部分积的累加和相加，并移位。

设置寄存器：

A：存放部分积累加和、乘积高位

B：存放被乘数

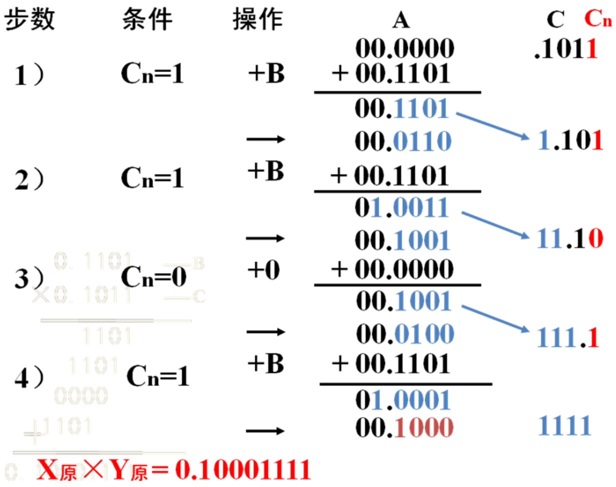
C：存放乘数、乘积低位

设置初值：

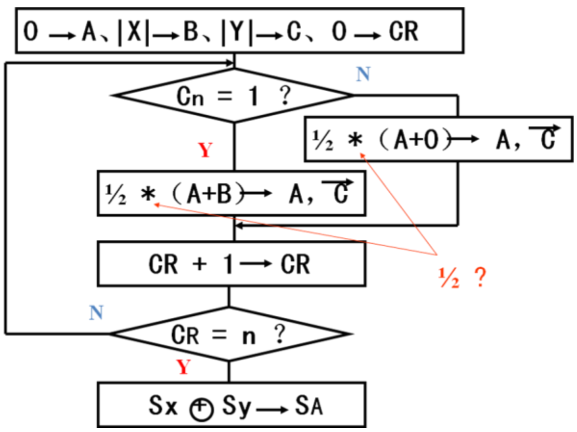
A = 00.0000

B = X = 00.1101

C = Y = 00.1011



(3)机器算法



(4)运算规则

i. 操作数、结果用原码表示；

ii. 绝对值运算，符号单独处理；

iii. 被乘数(B)、累加和(A)取双符号位；

iv. 乘数末位(Cn)为判断位，其状态决定下步操作；

v. 作 n 次循环（累加、右移）。

2.原码定点两位乘法

根据乘数每两位的取值情况，一次求出对应于该两位的部分积。此时，只要增加少量逻辑电路，就可使乘法速度提高一倍。

(1)算法分析

Yi+1(高位)	**Yi(低位)**	**部分积**	**累加、移位**
0	0	**0**	**¼ A**
0	1	x	**¼ (A+X)**
1	0	2x	**¼ (A+2X)**
1	1	3x	**¼ (A+3X)**

问题：如何实现+3X 操作？

(A+3X)= (A-X)+4X

解决问题的办法是：以(4X-X)来代替 3X 运算，在本次运算中只执行 -X，而 +4X 则归并到下一步执行，此时部分积已右移了两位，上一步欠下的+4X 已变成+X，在实际线路中要用一个触发器 C 来记录是否欠下+4X，若是，则 1→C。因此实际操作用 Yi-1、Yi、C 三位来控制。

设置欠帐触发器CJ= $\begin{cases} \text{0 不欠帐} \\ \text{1 欠帐，下次补作+X操作} \end{cases}$

(2)机器算法：实际操作用 Yi，Yi+1，C 三位来控制

Yi(高位)	**Yi+1(低位)**	**Cj**	**操作**	
0	0	**0**	**¼ A**	0→**Cj**
0	0	1	**¼ (A+X)**	0→**Cj**
0	1	0	**¼ (A+X)**	0→**Cj**
0	1	1	**¼ (A+2X)**	0→**Cj**
1	0	**0**	**¼ (A+2X)**	0→**Cj**
1	0	1	**¼ (A-X)**	1→**Cj**
1	1	0	**¼ (A-X)**	1→**Cj**
1	1	1	**¼ A**	1→**Cj**

(3)运算实例

例 1. **$X_{原}$=1.111111，$Y_{原}$=0.111001，求 $(XY)_{原}$。**

初值： **A = 000.000000 ;部分积**

B = X = 000.111111 ;被乘数

2B = 001.111110 ;

-B = 111.000001 ;

C = Y = 00.111001 ;乘数

CJ = 0 ;欠账触发器

步数	条件 $C_{n-1}C_nC_J$	操作	A	C	$C_{n-1}C_nC_J$
			000.000000	00.111001	0
1）	0 1 0	+B	+000.111111		
			000.111111		
		$\xrightarrow{2}$	000.001111	1100.1110	0
2）	1 0 0	+2B	+001.111110		
			010.001101		
		$\xrightarrow{2}$	000.100011	011100.11	0
3）	1 1 0	-B	+111.000001		
			111.100100		
		$\xrightarrow{2}$	111.111001	000111 00.	1
4）	0 0 1	+B 还帐	+000.111111		
			000.111000	000111	

(XY)原=1.111000000111

（4）运算规则

i. 绝对值相乘，符号单独处理。

ii. A、B 取三符号位。

iii. C 取双符号位，参加移位；C 尾数凑足偶数位。

iv. CJ 初值为 0，根据每步操作决定其状态，不参加移位。

v. 作 1/2n 步循环；若需增加一步，则该步只还帐，不移位。

3.原码两位乘法和原码一位乘法比较

	原码一位乘	原码两位乘
符号位	$x_0 \oplus y_0$	$x_0 \oplus y_0$
操作数	绝对值	绝对值的补码
移位	逻辑右移	算术右移
移位次数	n	$\frac{n}{2}$（n为偶数）
最多加法次数	n	$\frac{n}{2}+1$（n为偶数）

(二) 单精度定点数除法

单精度定点除法有恢复余数法和加减交替法两种方法，在计算机中常用的是加

减交替法，因为它的操作步骤少，而且也不复杂。

两个原码数相除，其商的符号为两数符号的异或值，数值则为两数绝对值相除后的结果。

1.原码定点一位除法

（1）恢复余数法

被除数(余数)减去除数，如果为 0 或者为正值时，上商为 1，不恢复余数；如果结果为负，上商为 0，再将除数加到余数中，恢复余数。余数左移 1 位。

这种方法的缺点是：当某一次-Y 的差值为负时，要多一次+Y 恢复余数的操作，降低了执行速度，又使控制线路变得复杂，因此在计算机中很少采用。计算机中普遍采用的是不恢复余数的除法方案，又称之为加减交替法。

（2）加减交替法

当余数为正时，商上 1，求下一位商的办法，余数左移一位，再减去除数；当余数为负时，商上 0，求下一位商的办法，余数左移一位，再加上除数。

①算法分析

第二步:2r1-B=r2’<0

第三步:r2’+B=r2(恢复余数)

第四步:2r2-B=r3

2r2-B=2(r2’+B)-B

=2r2’+B=r3

第二步:2r1-B=r2’<0

第三步:2r2’+B=r3(不恢复余数)

②算法

ri 为正，则 Qi 为 1，第 i+1 步作 2ri-Y；

ri 为负，则 Qi 为 0，第 i+1 步作 2ri+Y。

$r_{i+1}=2r_i+(1-2Q_i)Y$

③实例

X=0.10110，Y=-0.11111，求 X/Y，给出商 Q 和余数 R。

初值：　A＝ X = 00.10110

B＝ Y = 00.11111

-B＝11.00001

C＝ Q = 0.00000

步数	条件	操作	A	C (C_n)
	r		00.10110 r_0	0.00000
1）		←	01.01100 $2r_0$	
		-B	+11.00001	
	为正		00.01101 r_1	0.00001 Q_1
2）		←	00.11010 $2r_1$	
		-B	+11.00001	
	为负		11.11011 r_2	0.00010 Q_2
3）		←	11.10110 $2r_2$	
		+B	+00.11111	
	为正		00.10101 r_3	0.00101 Q_3
4）		←	01.01010 $2r_3$	
		-B	+11.00001	
	为正		00.01011 r_4	0.01011 Q_4
5）		←	00.10110 $2r_4$	
		-B	+11.00001	
	为负		11.10111 r_5'	0.10110 Q_5
6）		+B	+00.11111	
		恢复余数	00.10110 r_5	

Q= -0.10110

R= 0.10110×2^{-5}

$$X/Y=-0.10110+\frac{0.10110\times2^{-5}}{-0.11111}$$

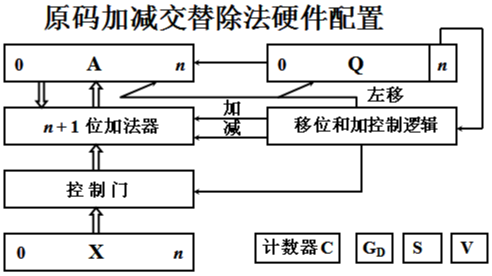
④运算规则

i. A、B 取双符号位，X、Y 取绝对值运算，X < Y 。

ii. 根据余数的正负决定商值及下一步操作。

iii. 求 n 位商，作 n 步操作；若第 n 步余数为负，则第 n+1 步恢复余数，不移位。

原码加减交替除法硬件配置



A、X、Q 均 n+1 位

用 Q_n 控制加减交替

除数不能为 0

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
	高位 1 移丢		（补码）

C_y ← 0 1 0 1 0 0 1 1　　0　1 0 1 0 0 1 1 0

(三) 浮点数的表示和运算

1. 浮点数的表示

(1)浮点数的表示范围

浮点数是指小数点位置可浮动的数据，通常以下式表示：

$$N=M\times R^{E}$$

其中，N 为浮点数，M**(Mantissa)**为尾数(可正可负)，E**(Exponent)**为阶码(可正可负)，R**(Radix)**称为“阶的基数(底)”，而且 R 为一常数，一般为 2，8 或 16。在一台计算机中，所有数据的 R 都是相同的，于是不需要在每个数据中表示出来。因此，浮点数的机内表示一般采用以下形式：

浮点数的机内表示一般采用以下形式：

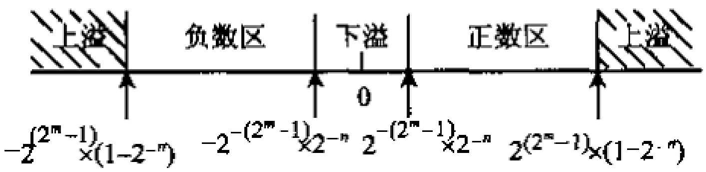
Ms	E	M
1 位	n+1 位	m 位

Ms 是尾数的符号位，设置在最高位上。

E 为阶码(移码)，有 n+1 位，一般为整数，其中有一位符号位，设置在 E 的最高位上，用来表正阶或负阶。

M 为尾数(原码)，有 m 位，由 Ms 和 M 组成一个定点小数。Ms=0，表示正号，Ms=1，表示负。为了保证数据精度属数通常用规格化形式表示：当 R＝2，且尾数值不为 0 时，其绝对值大于或等于$(0.5)_{10}$。对非规格化浮点数，通过将尾数左移或右移，并修改阶码值使之满足规格化要求。

浮点数的表示范围以通式 $N=M\times R^{E}$ 设浮点数阶码的数值位取 m 位，尾数的数值位取 n 位



浮点数在数轴上的表示范围

(2)IEEE754 标准<u>(Institute of Electrical and Electronics Engineers 美国电气和电子工程协会)</u>

S	阶码(含阶符)	尾　　数

数符　　　　小数点位置

根据 IEEE 754 国际标准，常用的浮点数有三种格式：

	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

单精度格式 32 位，阶码为 8 位，尾数为 23 位。另有一位符号位 S，处在最高位。

由于 IEEE754 标准约定在小数点左部有一位隐含位，从而实际有效位数为 24 位。这样使得尾数的有效值变为 1.M .

例如，最小为 x1.0…0,，最大为 x1.1…1.规格化表示。故小数点左边的位恒为 1，可省去。

阶码部分采用移码表示，移码值 127，1 到 254 经移码为-126 到+127。

S(1 位)	E(8 位)	M(23 位)	N(共 32 位)
符号位	0	0	0
符号位	0	不等于 0	(-1)S 2-126 (0.M) 为非规格化数
符号位	1 到 254 之间	-	(-1)S 2E-127 (1.M) 为规格化数
符号位	255	不等于 0	NaN(非数值)
符号位	255	0	无穷大

0 有了精确的表示，无穷大也明确表示。对于绝对值较小的数，可以采用非规格化数表示，减少下溢精度损失。非规格化数的隐含位是 0，不是 1。

2. 浮点数的加/减运算

(1)加减法执行下述五步完成运算：

① “对阶”操作

比较两浮点数阶码的大小，求出其差 ΔE，保留其大值 E，E=max(Ex， Ey)。当 ΔE≠0 时，将阶码小的尾数右移 ΔE 位，并将其阶码加上 ΔE，使两数的阶码值相等。

② 尾数加减运算

执行对阶之后，两尾数进行加减操作。

③ 规格化操作

规格化的目的是使得尾数部分的绝对值尽可能以最大值的形式出现。

④舍入

在执行右规或者对阶时，尾数的低位会被移掉，使数值的精度受到影响，常用“0”舍“1”入法。

⑤再次规格化

当移掉的部分最高位为 1 时，在尾数的末尾加 1，如果加 1 后又使得尾数溢出，则要再进行一次右规。

⑥ 检查阶码是否溢出

阶码溢出表示浮点数溢出。在规格化和舍入时都可能发生溢出，若阶码正常，加/减运算正常结束。若阶码下溢，则设置机器运算结果为机器零，若上溢，则设置溢出标志。

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

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

(2) 对阶原则

小阶向大阶看齐

3. 规格化

(1) 规格化数的定义

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

(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

定点数和浮点数可从如下几个方面进行比较

①当浮点机和定点机中的位数相同时，浮点数的表示范围比定点数大得多

②当浮点数位规格化数时，其相对绝对远比定点数高

③浮点数运算要分阶码部分和尾数部分，而且运算结果都要求规格化，故浮点运算步骤比定点运算的步骤多，运算速度比定点运算的低，运算线路比定点运算的复杂

④在溢出的判断方法上，浮点数是对规格化的阶码进行判断，而定点数是对数值本身进行判断

总之，浮点数在数的表示范围，数的精度，溢出处理和程序编程方面(不取比例因子)均优于定点数。但在运算规则即硬件成本方面又不如定点数

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

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

1. 对阶

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

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

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

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

对阶后的$[S_x]_{补'}$

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

(3) 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

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 $x = 0.1101 \times 2^{10}$ $y = 0.1011 \times 2^{01}$

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

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

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

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

② 尾数求和

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

对阶后的 $[S_y]_{补'}$

尾数溢出需右规

③ 右规

$[x+y]_{补} = 00, 010; 01.001010$

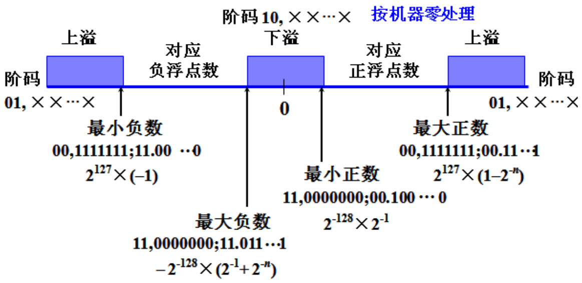
右规后

$[x+y]_{补} = 00, 011; 00.100101$

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

5. 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



二、浮点乘除运算

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

1. 乘法

$$x \cdot y = (S_x \cdot S_y) \times 2^{j_x + j_y}$$

2. 除法

$$\frac{x}{y} = \frac{S_x}{S_y} \times 2^{j_x - j_y}$$

3. 步骤

(1) 阶码采用补码定点加（乘法）减（除法）运算

(2) 尾数乘除同定点运算

(3) 规格化

4. 浮点运算部件

阶码运算部件，尾数运算部件