第 6 章 计算机的运算方法

6.1 无符号数和有符号数

一、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

二、有符号数

1.机器数与真值

计算机没有硬件来表示小数点，小数点的位置只能通过约定的方式确定。

真值（带符号的数）	机器数（符号数字化的数）		
+0.1011	**0**	小数点	1011
-0.1011	**1**	小数点	1011
+1100	**0**	1100	小数点
-1100	**1**	1100	小数点

2.原码表示法

（1）定义：

整数：$[x]_{原}=0,x$（当 $0<=x<2^n$）2^n-x（$-2^n<x<=0$）

小数：$[x]_{原}=x$（当 $0<=x<1$）$1-x$（$-1<x<=0$）

例：求 x=0 的原码：

设 x=+0.0000，则$[+0.0000]_{原}=0.0000$；设 x=-0.0000，则$[-0.0000]_{原}=1.0000$

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

故$\mathbf{[+0]_{原}\neq[-0]_{原}}$

3.原码的特点：简单、直观

但是用原码作加法，会出现如下问题：

要求	数 1	数 2	实际操作	结果符号
加法	+	+	加	+
加法	+	-	减	-/+
加法	-	+	减	-/+
加法	-	-	加	负

能否只作加法？**找到一个与负数等价的整数来代替这个负数**！

3.补码表示法

结论：

（1）一个负数加上“模”即得该负数的补数；

（2）一个整数和一个负数互为补数时，它们绝对值之和即为模数；

（3）正数的补数即为其本身；

（4）负数的补码=负数的反码+1。

但是补码的定义如下：

$[x]_{补}=2^{n+1}+x$（负整数），$2+x$（$-1<=x<0(\mathrm{mod}\ 2)$）

4.反码表示法

不再累述

5.三种机器数的小结

（1）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；

（2）对于正数，原码=补码=反码；
（3）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末位加 1 得补码，原码除符号位外每位取反得反码。

6.移码表示

补码很难直接判断其真值大小。

（1）移码的定义：

$[x]_{移}=2^n+x$

只有整数形式，没有小数形式。

（2）移码和补码的比较：

补码与移码只差一个符号位。

（3）真值、补码和移码的对照表

如图 1.1 所示。

图 1.1 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
-100000	100000	000000	0
- 11111	100001	000001	1
- 11110	100010	000010	2
⋮	⋮	⋮	⋮
- 00001	111111	011111	31
± 00000	000000	100000	32
+ 00001	000001	100001	33
+ 00010	000010	100010	34
⋮	⋮	⋮	⋮
+ 11110	011110	111110	62
+ 11111	011111	111111	63

（4）移码的特点

①若 x=0，则$[+0]_{移}=2^5+0=1,00000$（最高位为符号位），$[-0]_{移}=2^5-0=1,00000$。所以$\mathbf{[+0]_{移}=[-0]_{移}}$。

②当 n=5 时，最小的真值为$-2^5=-100000$，$[-100000]_{移}=2^5-100000=000000$。

可见，最小真值的移码为全 0。

用移码表示浮点数的阶码，能方便地判断浮点数阶码的大小。

6.2 数的定点表示和浮点表示

一、定点表示

是指小数点按照约定方式给出。无论是软件还是硬件开发人员，都必须遵循按照约定方式。

有以下两种形式：

1.

数符	小数点	数值部分

这说明表示的全为小数，若为补码，则能表示的唯一整数是-1。

数符	数值部分	小数点

这说明表示的全为整数。

由此我们可以把定点机分为两类，一类是小数定点机（形式 1）和整数定点机（形式 2）。下面来说一下他们用原码、反码和补码表示的时候的范围。

表 2.1 定点机的表示范围

	小数定点机	整数定点机
原码	-（$1-2^{-n}$）~（$1-2^{-n}$）有 2^{n+1} 个	$-(2^n-1)$~ (2^n-1)
补码	-1~（$1-2^{-n}$）	$-(2^n)$~(2^n-1)
反码	-（$1-2^{-n}$）~（$1-2^{-n}$）	$-(2^n-1)$~ (2^n-1)

二、浮点表示（重点）

0.为什么要在计算机引入浮点数表示？

①编程困难，程序员要调节小数点的位置；

②数的表示范围小，为了能表示两个大小相差很大的数据，需要很长的机器字长；

③数据存储单元的利用率往往很低。

如图 2.1 所示。

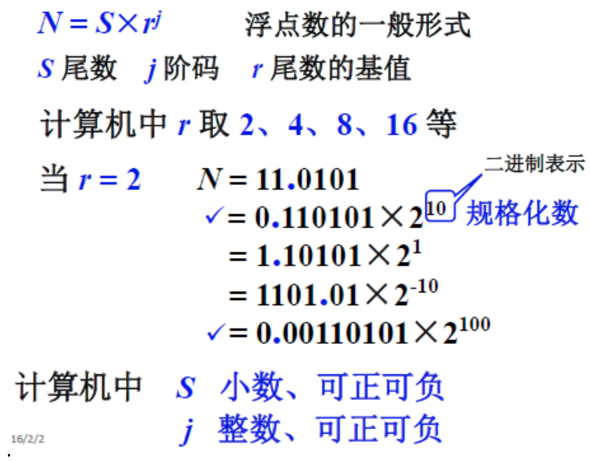


图 2.1 浮点表示

注意：**尾数是小数、可正可负，阶码是整数、可正可负**。当 r=2 尾数中的每一位二进制数就表示了尾数的一位；当 r=4，尾数当中的 2 位二进制数表示了 1 位四进制尾数。如图中所示，原来数是 11.0101，当我们固定小数点，将数字右移两位的时候，表示数字变为了原来的四分之一，所以阶码为 10（10 是十进制的 2,2^2=4）；当我们只右移 1 位时，表示数字变为了原来的二分之一，所以阶码为 1（1 是十进制的 1,2^1=2）；同样，当左移 2 位的时候，数字变为了原来的四倍，所以阶码为-10,（10 是十进制的 2,2^{-2}=1/4），剩余一个同理。在计算机中，只有打对号的两个数才能存储，其中第一种小数点后面第 1 位为 1，称为**规格化数**。

1.浮点数的表示形式

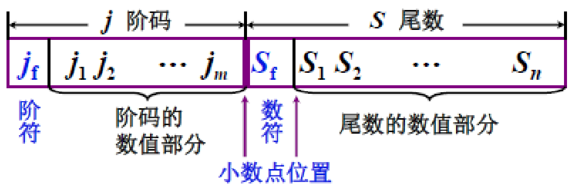


图 2.2 浮点数的表示形式

S_f 表示浮点数的符号；n 表示了浮点数的精度，m 表示浮点数的表示范围，j_f 和 m（实际是阶符和阶码）共同标识小数点的书记位置。

2.浮点数的表示范围

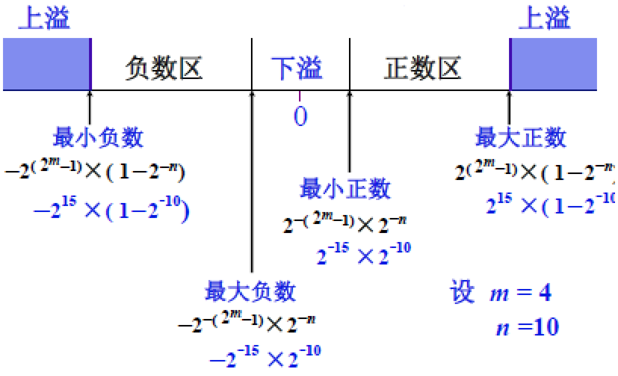


图 2.3 浮点数的表示范围

上溢：阶码>最大阶码。下溢：阶码<最小阶码，按**机器零**处理。阶码都用原码来表示。

最小负数和最大正数的绝对值其实是一样的。最大正数的尾数最大为全 1，阶码也为全 1。这样的情况下，尾数为（$1-2^n$），阶码为 2^{m-1}，故如图 2.3 所示。最大负数，最小正数同理。

实际上我们是由有限的个数表示无限多的实数。

练习：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？

因为 2 的 14 次方=16384,2 的 15 次方是 32768，满足要求。阶码应该在±15。所以定点数 15 位二进制数可以反映±3 万之间的十进制数。因为要保证数的最大精度，所以尾数尽可能的长，所以阶码位数尽可能短，所以阶码是 4 位。24-4-1（阶符）-1（数符）=18，为尾数的长度。

3.浮点数的规格化形式

为了保证尽可能保证浮点数的精度，所以要对浮点数进行规格化。

若 r=2，尾数最高位为 1；

若 r=4，尾数最高 2 位不全为 0；

若 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。

4.浮点数的规格化

表 2.1 浮点数的规格化形式

基值	规格化形式	规格化方式
r=2	左规	尾数左移 1 位，阶码减 1
	右规	尾数右移 1 位，阶码加 1
r=4	左规	尾数左移 2 位，阶码减 1
	右规	尾数右移 2 位，阶码加 2
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 3

基数 r 越大，可表示的浮点数**范围越大**，基数 r 越大，浮点数的**精度越大**。

例如：设 m=4，n=10，r=2。尾数规格化的浮点数表示范围：

最大正数：2^{+1111}*0.1111111111=2^{15}*($1-2^{-10}$)；

最小正数：2^{-1111}*0.1000000000=2^{-15}*2^{-1}=2^{16}；

最小负数：-2^{15}*($1-2^{-10}$)；

最大负数：-2^{-15}*2^{-1}=-2^{16}。

三、举例

例 6.13：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。其中树枝部分均取 10 位，数符取 1 位，浮点数阶码取 5 位（含 1 位阶符），尾数规格化。

解：设 x=$+\frac{19}{128}$，则二进制形式为 0.0010011，定点表示为 **0.0010011000**，因为要求 10 位，所以补 3 个零。

浮点规格化表示，因为规格化，所以小数点后第一位必须是 1，所以表示为 $x=0.1001100000\times2^{-10}$。

定点机中，原码=补码=反码=0.0010011000

浮点机中，[x]原=1, 0010;0.1001100000（第一位的 1 是阶码，0010 是阶符，0.1001100000 是尾数）

[x]补=1,1110;0.1001100000

[x]反=1,1101;0.1001100000

例 6.14：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，则写为二进制表示为 x=-111010，定点表示为-0000111010，浮点规格化形势为-(0.1110100000*2^{110})。定点机中，[x]原=1,0000111010，[x]反=1,1111000101，[x]反=1,1111000110。

浮点机中[x]原=0,0110;1.11101000000；[x]补 = 0, 0110; 1. 0001100000；

[x]反 = 0, 0110; 1. 0001011111　　　　[x]阶移、尾补 = 1, 0110; 1. 0001100000

机器零

浮点数尾数为 0 时，不论其阶码为何值按机器零处理。

当**浮点数阶码等于或小于它表示的最小数**时，不论尾数为何值，按机器零处理。

总的来说，是以下两种情况：

（1）x,xxxx;0.0000…000

（2）（阶码-16）1,0000;x.xx…xxx

当阶码用移码，尾数用补码表示时，机器零为 0,0000;0.00…0

这非常有利于机器中“判 0”电路的实现。

四、IEEE 754 标准

格式如图 2.4 所示

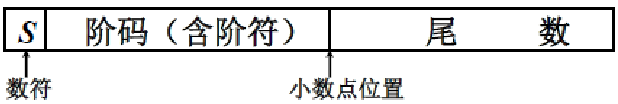


图 2.4 IEEE 754 标准浮点数示意图

由于尾数为规格化表示，那么最高位一定是 1，所以可以把这 1 位省略掉，这样可以提高精度。如表 2.2 所示

表 2.2 IEEE 754 数的类型

类型	符号位 S	阶码	尾数	总位数	表示范围
短实数	1	8	23	32	
长实数	1	11	52	64	
临时实数	1	15	64	80	

6.3 定点运算

一、移位运算

1.移位运算的数学意义

左移：绝对值扩大；右移：绝对值缩小。（都是数据相对于小数点移位）**在计算机中，移位与加减配合，能够实现乘除运算**。

2.算术移位规则

（1）符号位不变

（2）数据位规则如表 3.1 所示。

表 3.1 移位规则

	码制	添加代码
正数	原码、补码、反码	0
负数	原码	0
	补码	**左移右侧添 0**
		右移左侧添 1
	反码	1（代表原码里面是 0）

例：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

则原码、补码、反码均为 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 1 位	0,011010**0**（数值最高位丢弃）	+52
左移 2 位	0,11010**00**	+104
右移 1 位	0,**0**001101	+13
右移 2 位	0,**00**00110	+6

例：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

则原码为 1,0011010，反码为 1,1100101，补码为 1,1100110

原码结果如下：

移位操作	机器数	对应的真值
移位前	1,0011010	-26
左移 1 位	1,011010**0**（数值最高位丢弃）	-52
左移 2 位	1,11010**00**	-104
右移 1 位	1,**0**001101	-13
右移 2 位	1,**00**00110	-6

补码结果如下：

移位操作	机器数	对应的真值
移位前	1,1100110	-26

左移 1 位	1,10011**0**0（数值最高位丢弃）	-52
左移 2 位	1,00110**00**	-104
右移 1 位	1,**1**110011	-13
右移 2 位	1,**11**11001	-7

反码：

移位操作	机器数	对应的真值
移位前	1,1100101	-26
左移 1 位	1,100101**1**（数值最高位丢弃）	-52
左移 2 位	1,00101**11**	-104
右移 1 位	1,**1**110010	-13
右移 2 位	1,**11**11001	-6

3.算术移位的硬件实现

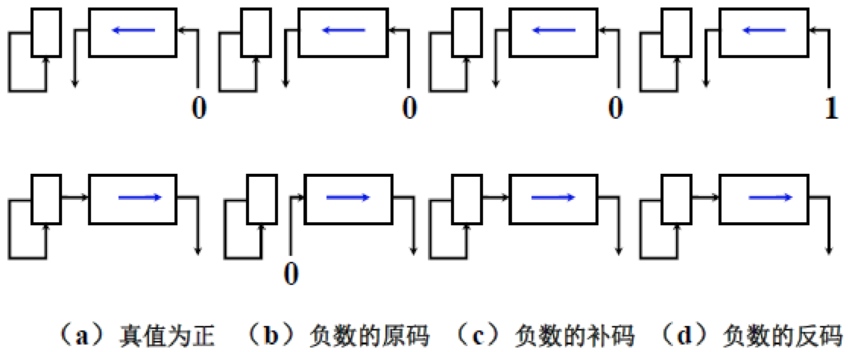


图 3.1 算法移位的硬件实现

在移位过程如果 1 被移掉会怎样：

左移：(a)出错(b)出错(c)正确（丢掉 0 出错）(d)正确

右移：(a)影响精度(b) 影响精度(c) 影响精度 (d)正确

4.算术移位与逻辑移位的区别

算术移位：有符号数的移位（符号位不动）；逻辑移位，无符号数的移位（符号位也参加移动）

逻辑左移，低位添 0，高位移丢；逻辑右移，高位添 0，低位移丢。

二、加减法运算

1.补码加减法运算的公式

如图 3.2 所示。

(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)

连同符号位一起相加，符号位产生的进位自然丢掉

图 3.2 补码加减运算公式

2.溢出的判断

（1）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（异或）符号位的进位=1，溢出。

如：

$$\text{有溢出}\begin{cases}1\oplus 0=1\\0\oplus 1=1\end{cases}$$

$$\text{无溢出}\begin{cases}0\oplus 0=0\\1\oplus 1=0\end{cases}$$

简单说一下，假如原来两个数都是正数，那么符号位是 0，如果他们相加，进到符号位了一个 1，符号位变成了 1，但是符号位不会向上进位，所以符号位进位为 0。所以符合有溢出的第 1 种情况。其它可以如此解释。

（2）两位符号位判溢出

3.补码加减法的硬件配置

$$[x]_{\text{补}'}=\begin{cases}x & 1>x\geq 0\\4+x & 0>x\geq -1(\text{mod}\,4)\end{cases}$$

[x]补’+[y]补’=[x+y]补’

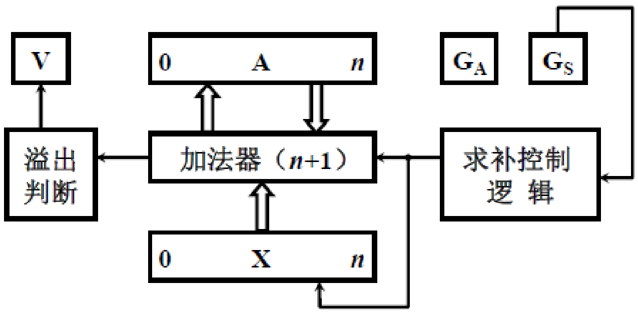
[x-y]补’=[x]补’+[-y]补’

结果的双符号位**相同**，**未溢出**：00,xxxxx/11,xxxxx

结果的双符号位**不同**，**溢出**：10,xxxxx/01,xxxxx

最高符号位代表其真正的符号！

3.补码加减法的硬件配置



A、X 均 n+1 位

用减法标记 G_S 控制求补逻辑

图 3.3 补码加减法的硬件配置

三、乘法运算

1.原码的乘法运算

一些认识：

（1）乘法运算可以用**加和移位**实现；

（2）由乘数的末位决定被乘数是否与元部分积相加，然后**右移 1 位形成新的部分积**，同时**乘数左移 1 位**（末位丢弃），空出高位存放部分积的低位；

（3）**被乘数只与部分积的高位相加**。

（1）原码一位乘运算规则

以小数为例

2.补码的乘法运算

（1）补码一位乘法运算

以小数为例：设备乘数[x]补=x0.x1x2…xn，乘数[y]补=y0.y1y2…yn

①被乘数任意，乘数为正

与原码乘相似，但加和移位按补码规则运算。乘积符号的自然形成

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。

③Booth 算法（**被乘数、乘数符号任意**）

如图 3.4 所示。

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

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

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

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

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

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

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

$$= [x]_{\text{补}}[(y_1-y_0)+\underbrace{(y_2-y_1)}_{y'_1}2^{-1}+\cdots+\underbrace{(y_{n+1}-y_n)}_{y'_n}2^{-n}] \qquad \boxed{\text{附加位 } y_{n+1}}$$

$$= y'_0[x]_{\text{补}}+2^{-1}(y'_1[x]_{\text{补}}+2^{-1}(y'_2[x]_{\text{补}}+\cdots 2^{-1}(y'_n[x]_{\text{补}}+0)\cdots))$$

图 3.4 Booth 算法

④Booth 算法递推公式

$$[z_0]_{\text{补}} = 0$$

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

$$\vdots$$

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

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

最后一步不移位

如何实现 $+(y_{i+1}-y_i)[x]_{\text{补}}$ ？

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	$\rightarrow 1$

图 3.5 Booth 算法递推公式

整数乘法和小数乘法过程完全相同，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。

四、除法运算

1.笔算除法

心算上商，余数不动低位补“0”，然后减右移一位的除数，上商的位置不固定。

2.笔算除法与机器除法的比较

笔算除法	机器除法

商符单独处理，心算上商	符号位异或形成，|x|-|y|>0，上商 1；否则上商 0
余数**不动**，低位补 0，**减右移一位**的除数。	余数**左移一位**，低位补零，**减**除数。
2 倍字长加法器，上商位置不固定	1 倍字长加法器，在寄存器最末尾上商。

3.原码除法

以小数为例

[x]原=x0.x1x2…xb

[y]原=y0.y1y2…yn

[x/y]原=x0⊕y0 • x*/y*

式中，x*=0.x1x2…xn 为 x 的绝对值；y*=0.y1y2…yn 为 y 的绝对值。

商的符号位单独处理：x0⊕y0；数值部分为绝对值相除 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。

（1）恢复余数法

例 6.24 x=-0.1011，y=-0.1101，求[x/y]原，假设机器字长为 5 位，其中符号位 1 位，数值位 4 位。

解：[x]原=1.1011，[y]原=1.1101，[y*]补=0.1101，[-y*]补=1.0011

①x0⊕y0=1⊕1=0

②如下表：

被除数（余数）	商	说明
0.1011 +1.0011	0.0000	+[-y*]补
1.1110 +0.1101	0	余数为负，上商 0 恢复余数，+[y*]补
0.1011 1.0110 +1.0011	0 0	恢复后的余数 ←1 +[-y*]补
0.1001 1.0010 +1.0011	01 01	余数为正，上商 1 ←1 +[-y*]补
0.0101 0.1010 +1.0011	011 011	余数为正，上 1 ←1 +[-y*]补
1.1101 +0.1101	0110	余数为负，上商 0 恢复余数+[-y*]补
0.1010 1.0100 +1.0011	0110 0110	恢复后的余数 ←1 +[-y*]补
0.0111	01101	余数为正，上商 1

所以 x*/y*=0.1101

所以[x/y]原=0.1101

余数为正，上商 1；余数为负，上商 0，恢复余数。

上商 5 次（第一次是 0），第一次上商判溢出，移 4 次。

（2）加减交替法

运算规则：

设余数为 Ri，

上商“1”：2Ri-y*；上商“0”：2Ri+y*。仍然是余数为正，上商 1；余数为负，上商 0。

例 6.25 x=-0.1011，y=-0.1101，求[x/y]原，假设机器字长为 5 位，其中符号位 1 位，数值位 4 位。

解：[x]原=1.1011，[y]原=1.1101，[x*]补=0.1011，[y*]补=0.1101，[-y*]补=1.0011。

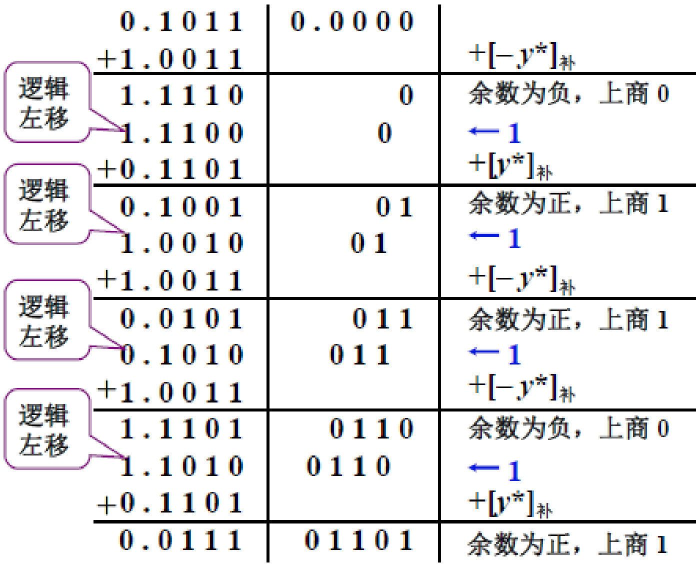


图 3.6 例 6.25 图解

特点：上商 **n+1** 次，**第一次上商**判溢出，移 **n** 次，加 **n+1** 次，用**移位的次数**判断除法是否结束。

（3）原码加减交替除法硬件配置

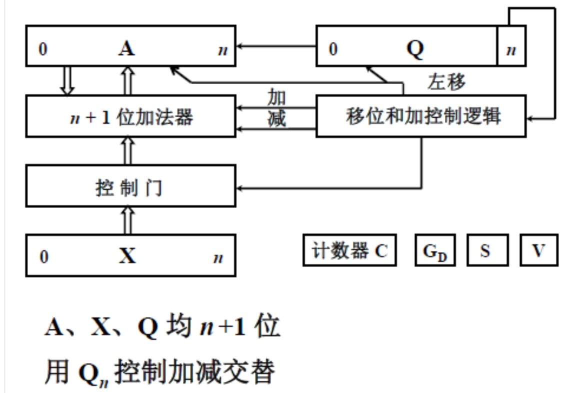


图 3.7 硬件配置

6.4 浮点四则运算

一、浮点数的加减运算

我们令 x=Sx • 2^{jx}，y= Sy • 2^{jy}

1.对阶

（1）求阶差。如图 4.1 所示。

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

图 4.1 阶码对齐方案

（2）对阶原则

小阶向大阶看齐。

例如：x=0.1101*2^{01}，y=(-0.1010)*2^{11}，求 x+y。

解：[x]补=00,01;00.1101

由于阶码不相等，所以先需要求阶差。[Δj]补=[jx]补-[jy]补=11,10。所以 x 要向 y 看齐。

进行对阶：[x]补'=00,11;00.0011

然后进行尾数求和，[Sx]补'+[Sy]补=11.1001。

所以[x+y]补=00,11;11.1001。

3.规格化

（1）规格化数的定义

r=2,1/2<=|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	0.1XX…X	真值	-0.1XX…X
原码	0.1XX…X	原码	1.1XX…X
补码	0.1XX…X	补码	1.0XX…X
反码	0.1XX…X	反码	1.0XX…X

原码：不论正数、负数，第一数位均为 1；补码，符号位和第一数位不同。

特例：

1. S=-1/2=-0.100…0，其原码为 1.100…0，补码为 1.100…0，所以[-1/2]补不上规格化的数。
2. S=-1，[S]补=1.000…0，所以[-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。

（4）右规

当**尾数溢出（>1）**时，需**右规**。即尾数出现 01.XX…X 或 10.XX…X 时。**尾数右移 1 位，阶码加 1**。

两个整数相加就有可能造成 01.XX…X 这种方式的溢出。其实 0 是符号位，1 是数值位。

例 6.27 x=0.1101*2^{10}，y=0.1011*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

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

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

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

② 尾数求和

$[S_x]_补 = 00.110100$
$+\ [S_y]_{补'} = 00.010110$　对阶后的 $[S_y]_{补'}$
$= 01.001010$　尾数溢出需右规

③ 右规

$[x+y]_补 = 00,010;01.001010$

右规后

$[x+y]_补 = 00,011;00.100101$

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

图 4.2 例 6.27 解答

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 6.28 如图 4.3 所示。

例 6.28 $x=(-\frac{5}{8})\times 2^{-5}$ $y=(\frac{7}{8})\times 2^{-4}$

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 $\therefore S_x \longrightarrow 1,\ j_x+1$

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

② 尾数求和

$$\begin{array}{r} [S_x]_{补'}=11.101100 \\ +\ [-S_y]_补=11.001000 \\ \hline 110.110100 \end{array}$$

③ 右规

$[x-y]_补=11,100;10.110100$

右规后

$[x-y]_补=11,101;11.011010$

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

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

图 4.3 例 28 解答

5.溢出判断（整个浮点数的溢出）

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

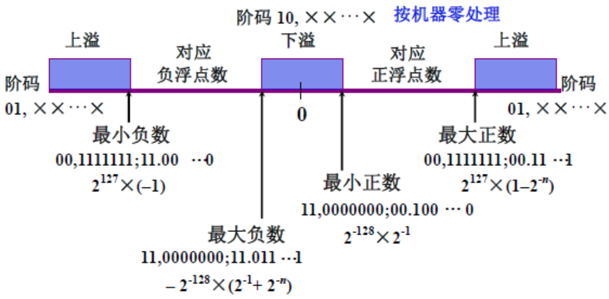


图 4.4 溢出判断

所谓下溢，就是阶码比-128 还小（阶码为 10,XX…X），按机器零处理；所谓上溢，就是

阶码比 127 还大（阶码为 01,xx…x）。

6.5 算数逻辑单元

一、ALU 电路

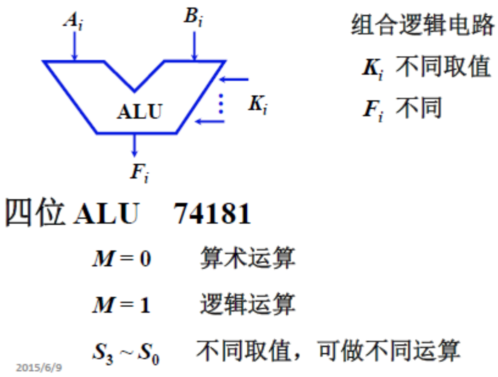


图 5.1 ALU 电路

ALU 是一个组合逻辑电路，是**没有记忆功能**的。为了对输出结果进行保存，要在 Ai 和 Bi 还有输出端有寄存器。控制端控制运算方式。例如 74181，M=0 做算术运算，M=1 做逻辑运算。

二、快速进位链

1.并行加法器

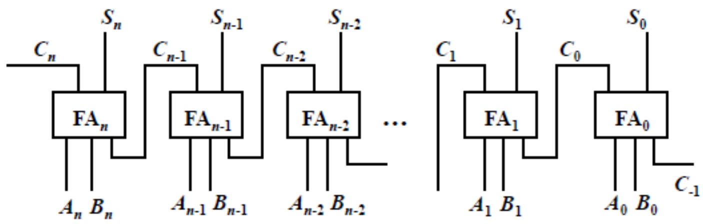


图 5.2 并行加法器

对于每一个 FA，有 3 个输入（2 个输入为加数，1 个为仅为信息 C）、2 个输出（一个是结果 Sn，一个是进位信息 C）。下面对这个并行加法器做一个分析：

$$Si = \bar{A}_i\bar{B}_iC_{i-1} + \bar{A}_iB_i\bar{C}_{i-1} + A_i\bar{B}_i\bar{C}_{i-1} + A_iB_iC_{i-1}$$

$$Ci = \bar{A}_iB_iC_{i-1} + A_i\bar{B}_iC_{i-1} + A_iB_i\bar{C}_{i-1} + A_iB_iC_{i-1}$$

$$=A_iB_i + (A_i + B_i)C_{i-1}$$

Ci 生成和 Ai、Bi 有关系。若 Ai 和 Bi 都为 1，则一定会进位，Ai 或 Bi 有一个为 1，那么 Ci-1 的值就会被传送给 Ci。所以，di=AiBi 称为本地进位；ti=Ai+Bi 称为传送条件。则 Ci=di+tiCi-1。

2.串行进位链（通常情况下）

进位链就是传送进位的电路。串行进位链就是进位进行串行传送。以 4 位全加器为例，每一位的进位表达式为如图 5.3 中式子所示。设与非门的级延迟时间为 t_y。

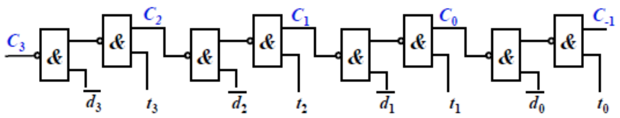


图 5.3 串行进位链

4 位全加器产生进位的全部时间为 $8t_y$，n 为全加器产生进位的全部时间为 $2nt_y$。

3.并行进位链（先行进位，跳跃进位）

n 位加法器的进位同时产生，以 4 位加法器为例如图 5.4 所示。

$$C_0 = d_0 + t_0C_{-1}$$
$$C_1 = d_1 + t_1C_0 = d_1 + t_1d_0 + t_1t_0C_{-1}$$
$$C_2 = d_2 + t_2C_1 = d_2 + t_2d_1 + t_2t_1d_0 + t_2t_1t_0C_{-1}$$
$$C_3 = d_3 + t_3C_2 = d_3 + t_3d_2 + t_3t_2d_1 + t_3t_2t_1d_0 + t_3t_2t_1t_0C_{-1}$$

当 d_it_i 形成后，只需 $2.5t_y$ 产生全部进位

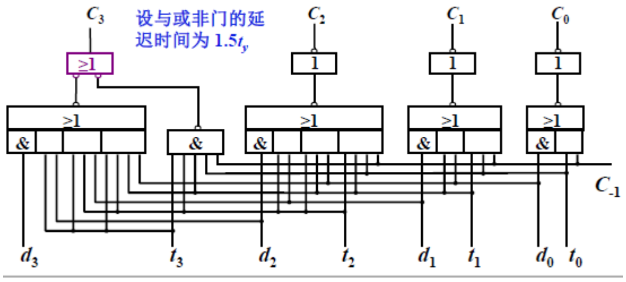


图 5.4 并行进位链

（1）单重分组跳跃进位链

n 位全加器分若干小组，小组中的进位同时产生，小组与小组之间采用串行进位。以 n=16 为例。如图 5.5 所示。

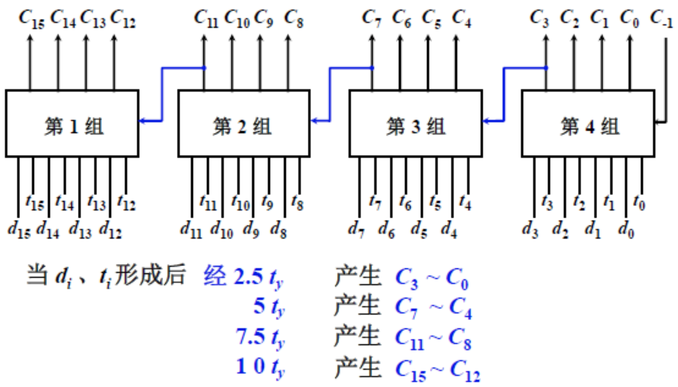


图 5.5 单重分组跳跃进位链

（2）双重分组跳跃进位链

n 位全加器分若干大足，大足中又包含若干小组。每个大租中小组的最高位进位同时产生。大组和大组之间采用串行进位。

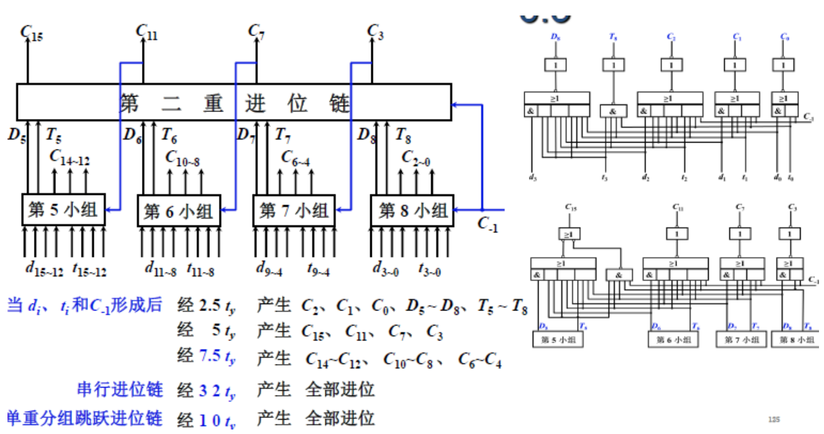


图 5.6 双重分组跳跃进位链