第3章 运算方法和运算部件

3.1 数值的表示方法和转换
3.2 带符号的二进制数据在计算机中的表示方法及加减法运算
3.3 二进制乘法运算
3.4 二进制除法运算
3.5 浮点的运算方法
3.6 运算部件
3.7 数据校验码

主要知识点:

1、数值的表示方法和转换
2、二进制数据的加减法运算及溢出判断方法
3、二进制乘法、除法运算
4、浮点数的加减乘除运算
5、运算部件
6、数据校验码

3.1 数据的表示方法和转换

3.1.1 数值型数据的表示和转换

1. 数制

- **十进制数$(N)_{10}$可表示为：**

$$(N)_{10}=D_m \cdot 10^m+D_{m-1} \cdot 10^{m-1}+\cdots+D_0 \cdot 10^0+D_{-1} \cdot 10^{-1}+\cdots+D_{-k} \cdot 10^{-k}$$

$$=\sum_{i=m}^{-k} D_i \cdot 10^i \qquad (3.1)$$

- **二进制数可表示为：**

$$(N)_2=D_m \cdot 2^m+D_{m-1} \cdot 2^{m-1}+\ldots+D_0 \cdot 2^0+D_{-1} \cdot 2^{-1}+\ldots+D_{-k} \cdot 2^{-k}$$

$$=\sum_{i=m}^{-k} D_i \cdot 2^i \qquad (3.2)$$

- **一个八进制数可表示为：**

$$(N)_8 = \sum_{i=m}^{-k} D_i \cdot 8^i \qquad (3.3)$$

- **一个十六进制数可表示为：**

$$(N)_{16} = \sum_{i=m}^{-k} D_i \cdot 16^i \qquad (3.4)$$

例3.1

$(1101.0101)_2=(1\cdot 2^3+1\cdot 2^2+0\cdot 2^1+1\cdot 2^0+0\cdot 2^{-1}+1\cdot 2^{-2}+0\cdot 2^{-3}+1\cdot 2^{-4})_{10}$

$=(8+4+0+1+0+0.25+0+0.0625)_{10}$

$=(13.3125)_{10}$

例3.2

$(15.24)_8=(1\cdot 8^1+5\cdot 8^0+2\cdot 8^{-1}+4\cdot 8^{-2})_{10}$

$=(8+5+0.25+0.0625)_{10}$

$=(13.3125)_{10}$

例3.3

$(0D.5)=(0\cdot 16^1+13\cdot 16^0+5\cdot 16^{-1})_{10}$

$=(0+13+0.3125)_{10}$

$=(13.3125)_{10}$

表3.1 二、八、十六和十进制数的对应关系

二进制数	八进制数	十六进制数	十进制数
0 0 0 0	0 0	0	0
0 0 0 1	0 1	1	1
0 0 1 0	0 2	2	2
0 0 1 1	0 3	3	3
0 1 0 0	0 4	4	4
0 1 0 1	0 5	5	5
0 1 1 0	0 6	6	6
0 1 1 1	0 7	7	7
1 0 0 0	1 0	8	8
1 0 0 1	1 1	9	9
1 0 1 0	1 2	A	1 0
1 0 1 1	1 3	B	1 1
1 1 0 0	1 4	C	1 2
1 1 0 1	1 5	D	1 3
1 1 1 0	1 6	E	1 4
1 1 1 1	1 7	F	1 5

2. 不同数制间的数据转换

- **二进制数、八进制数和十六进制数之间的转换**

例3.4

$$(1\ 101.010\ 1)_2=(001\ 101.010\ 100)_2=(15.24)_8$$

例3.5

$$(1\ 1101.0101)_2=(0001\ 1101.0101)2=(1D.5)_{16}$$

例3.6

$$(15.24)_8=(001\ 101.010\ 100)_2=(1101.0101)_2$$

•二进制数转换成十进制数

利用上面讲到的公式$(N)_2=\sum_{i=m}^{-k} D_i \cdot 2^i$进行计算。

•十进制数转换成二进制数

要对一个数的整数部分和小数部分分别进行处理，各自得出结果后再合并。

对整数部分，一般采用除2取余数法，规则如下：

将十进制数除以2，所得余数(0或1)即为对应二进制数**最低位的值。**然后对上次所得的商除以2，所得余数即为二进制数次低位的值，如此进行下去，直到商等于0为止，最后得出的余数是所求二进制数**最高位的值。**

对小数部分，一般用乘2取整数法，其规则如下：

将十进制数乘以2，所得乘积的整数部分即为对应二进制小数**最高位的值，**然后对所余的小数部分乘以2，所得乘积的整数部分为次高位的值，如此进行下去，直到乘积的小数部分为0，或结果已满足所需精度要求为止。

例3.7 将$(105)_{10}$转换成二进制。

2	105	余数	结果
2	52	1	最低位
2	26	0	
2	13	0	…
2	6	1	
2	3	0	
2	1	1	
	0	1	最高位

得出：$(105)_{10}=(1101001)_2$

例3.8 将$(0.3125)_{10}$和$(0.3128)_{10}$转换成二进制数(要求4位有效位)。

①结果		0.3125×2
最高位	0	.6250×2
…	1	.2500×2
	0	.5000×2
最低位	1	.0000

得出:(0.3125)10=(0.0101)2

② 结果		0.3128×2
最高位	0	.6256×2
…	1	.2512×2
	0	.5024×2
最低位	1	0048

得出:(0.312 8)10=(0.0101)2

•十进制数转换成八进制数

参照十进制数转换成二进制数的方法，将基数2改为8，即可实现转换。

例3.9 将$(13.312\ 5)_{10}$转换成八进制数，处理过程如下：

整数部分转换

8	13	余数
8	1	5
	0	1

$(13)_{10}=(15)_8$

小数部分转换

	0.3125×8
2	.5000×8
4	.0000

$(0.3125)_{10}=(0.24)_8$

得出： (13.3125)10=(15.24)8

3. 数据符号的表示

数据的数值通常以正(+)负(-)号后跟绝对值来表示，称之为**“真值”**。在计算机中正负号也需要数字化，一般用0表示正号，1表示负号。正号有时可省略。

3.1.2 十进制数的编码与运算

1. 十进制数位的编码与运算

在计算机中采用4位二进制码对每个十进制数位进行编码。

- **(1)有权码：**表示一位十进制数的二进制码的每一位有确定的权。一般用8421码，其4个二进制码的权从高到低分别为8、4、2和1。用0000，0001，…，1001分别表示0，1，…，9，每个数位内部满足二进制规则，而数位之间满足十进制规则，故称这种编码为“以二进制编码的十进制(binary coded decimal，简称BCD)码”。

在计算机内部实现BCD码算术运算，要对运算结果进行修正，对加法运算的修正规则是：**相加之和小于或等于(1001)2，即$(9)_{10}$，不需要修正；相加之和大于或等于$(10)_{10}$，要进行加6修正，并向高位进位，进位可以在首次相加或修正时产生。**

例3.10

① 1+8=9

```
   0 0 0 1
+  1 0 0 0
-----------
   1 0 0 1        不需要修正
```

②　4+9=13

0 1 0 0

+ 1 0 0 1

1 1 0 1

+ 0 1 1 0　修正

1 0 0 1 1

1

进位

③　9+7=16

1 0 0 1

+ 0 1 1 1

1 0 0 0 0

+ 0 1 1 0　修正

1 0 1 1 0

进位

另外几种有权码，如2421，5211，4311码，也是用4位二进制码表示一个十进制数位，但4位二进制码之间不符合二进制规则。这几种有权码有一特点，即任何两个相加之和等于$(9)_{10}$的二进制码互为反码。例如，在2421码中，0(0000)与9(1111)、1(0001)8(1110)、…，互为反码。

表3.2　4位有权码

十进制数	8421码	2421码	5211码	4311码
0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1	0 0 0 1	0 0 0 1
2	0 0 1 0	0 0 1 0	0 0 1 1	0 0 1 1
3	0 0 1 1	0 0 1 1	0 1 0 1	0 1 0 0
4	0 1 0 0	0 1 0 0	0 1 1 1	1 0 0 0
5	0 1 0 1	1 0 1 1	1 0 0 0	0 1 1 1
6	0 1 1 0	1 1 0 0	1 0 1 0	1 0 1 1
7	0 1 1 1	1 1 0 1	1 1 0 0	1 1 0 0
8	1 0 0 0	1 1 1 0	1 1 1 0	1 1 1 0
9	1 0 0 1	1 1 1 1	1 1 1 1	1 1 1 1

•**(2)无权码:**表示一个十进制数位的二进制码的每一位没有确定的权。用得较多的是余3码(Excess-3 Code)和格雷码(Gray Code)，格雷码又称“循环码”。

余3码:是在8421码基础上，把每个编码都加上0011而形成的(见表3.3)，其运算规则是：当两个余3码相加**不产生进位时，应从结果中减去0011**；**产生进位时，应将进位信号送入高位，本位加0011。**

例3.11 $(28)_{10}+(55)_{10}=(83)_{10}$

```
 0 1 0 1          1 0 1 1        (28)10
+) 1 0 0 0   1      1 0 0 0     (55)10
 ---------------------------------------
 1 1 1 0          0 0 1 1      低位向高位产生进位，
                               高位不产生进位。
-) 0 0 1 1       +) 0 0 1 1    低位+3，高位-3。
 ---------------------------------------
 1 0 1 1          0 1 1 0
```

格雷码的编码规则：任何两个相邻编码只有一个二进制位不同，而其余三个二进制位相同。其优点是从一个编码变到下一个相邻编码时，只有1位发生变化，用它构成计数器时可得到更好的译码波形。格雷码的编码方案有多种，表3.3给出两组常用的编码值。

表3.3 4位无权码

十进制数	余3码	格雷码(1)	格雷码(2)
0	0 0 1 1	0 0 0 0	0 0 0 0
1	0 1 0 0	0 0 0 1	0 1 0 0
2	0 1 0 1	0 0 1 1	0 1 1 0
3	0 1 1 0	0 0 1 0	0 0 1 0
4	0 1 1 1	0 1 1 0	1 0 1 0
5	1 0 0 0	1 1 1 0	1 0 1 1
6	1 0 0 1	1 0 1 0	0 0 1 1
7	1 0 1 0	1 0 0 0	0 0 0 1
8	1 0 1 1	1 1 0 0	1 0 0 1
9	1 1 0 0	0 1 0 0	1 0 0 0

3.2带符号的二进制数据在计算机中的表示方法及加减法运算

在计算机中表示的带符号的二进制数称为“机器数”。机器数有三种表示方式：**原码、补码和反码。**

为讨论方便，先假设机器数为小数，符号位放在最左面，小数点置于符号位与数值之间。数的真值用X表示。

3.2.1 原码、补码、反码及其加减法运算

1.原码表示法

机器数的最高位为符号位，0表示正数，1表示负数，数值跟随其后，并以绝对值形式给出。这是与真值最接近的一种表示形式。

原码的定义：

$$[X]_{原}=\begin{cases}X & 0\le X<1\\ 1-X=1+|X| & -1<X\le 0\end{cases}\qquad(3.5)$$

即 [X] 原=符号位+｜X｜。

例3.12

X=+0.1011，$[X]_{原}$=01011;

当X=-0.1011时，$[X]_{原}$=1-(-0.1011)=1.1011=11011。

零的真值有+0和-0两种表示形式，$[X]_{原}$也有两种表示形式：

$[+0]_{原}$=00000，$[-0]_{原}$=10000。

- **数的原码与真值之间的关系比较简单**，其算术运算规则与十进制运算规则类似，当运算结果不超出机器能表示的范围时，运算结果仍以原码表示。
- **它的最大缺点是在机器中进行加减法运算时比较复杂**。当两数相加时，先要判别两数的符号，如果两数是同号，则相加；两数是异号，则相减。而进行减法运算又要先比较两数绝对值的大小，再用大绝对值减去小绝对值，最后还要确定运算结果的正负号。
- **下面介绍的用补码表示的数在进行加减法运算时可避免这些缺点**。

2. 补码表示法

机器数的最高位为符号位，0表示正数，1表示负数，其定义如下：

$$[X]_{补}=\begin{cases} X & 0\le X<1 \\ 2+X=2-|X| & -1\le X<0 \end{cases} \qquad (3.6)$$

即 $[X]_{补}=2\cdot$符号位$+X \pmod 2$

此处2为十进制数，即为二进制的10。

例3.13　$X=+0.1011$,则 $[X]_{补}=0.1011$

$X=-0.1011$,则 $[X]_{补}=2+X=2+(-0.1011)=1.0101$

数值零的补码表示形式是唯一的，即：

$$[+0]_{补}=[-0]_{补}=0.0000$$

当补码加法运算的结果**不超出机器范围时，**可得出以下**重要结论：**

(1) 用补码表示的两数进行加法运算，其结果仍为补码。

(2) $[X+Y]_{补}=[X]_{补}+[Y]_{补}$。

(3) 符号位与数值位一样参与运算。

例3.14 设X=0.1010,Y=0.0101,两数均为正数:

$[X+Y]_{补}=[0.1010+0.0101]_{补}=[0.1111]_{补}=0.1111$

$[X]_{补}+[Y]_{补}=0.1010+0.0101=0.1111$

即 $[X+Y]_{补}=[X]_{补}+[Y]_{补}=0.1111$

例3.15 设X=0.1010,Y=-0.0101,X为正，Y为负:

$[X+Y]_{补}=[0.1010+(-0.0101)]_{补}=0.0101$

$[X]_{补}+[Y]_{补}=0.1010+[-0.0101]_{补}=0.1010+(2-0.0101)=2+0.0101=0.0101 \mod 2$

即 $[X+Y]_{补}=[X]_{补}+[Y]_{补}=0.0101$

例3.16 设X=-0.1010,Y=0.0101,X为负，Y为正:

$[X+Y]_{补}=[-0.1010+0.0101]_{补}=[-0.0101]_{补}=1.1011$

$[X]_{补}+[Y]_{补}=[-0.1010]补+[0.0101]补=1.0110+0.0101=1.1011$

即 $[X+Y]_{补}=[X]_{补}+[Y]_{补}=1.1011$

例3.17 设X=-0.1010,Y=-0.0101,X,Y均为负数:

$[X+Y]_{补}=[-0.1010+(-0.0101)]_{补}=[-0.1111]_{补}=1.0001$

$[X]_{补}+[Y]_{补}=1.0110+1.1011=10+1.0001=1.0001 \mod 2$

即: $[X+Y]_{补}=[X]_{补}+[Y]_{补}=1.0001$

例3.18 设X=-0.0000, Y=-0.0000

$[X]_{补}+[Y]_{补}=[X+Y]_{补}$

$=(2+0.0000)+(2+0.0000)=4+0.0000=0.0000 \mod 2$

例3.19 设X=-0.1011, Y=-0.0101,则有

$[X+Y]_{补}=[X]_{补}+[Y]_{补}=1.0101+1.1011=11.0000=1.0000 \mod 2$

X+Y的真值=-0.1011+(-0.0101)=-1.0000,为-1。由此说明一个数的补码值的范围在**-1和$(1-2^{-n})$之间**(假设数值部分为n位)。

对于减法运算，因为［X-Y］补=［X+(-Y)］补=［X］补+［-Y］补，所以计算时，可以先求出-Y的补码，然后再进行加法运算，这样在用逻辑电路实现加减法运算时，可以只考虑用加法电路，而不必设置减法电路。

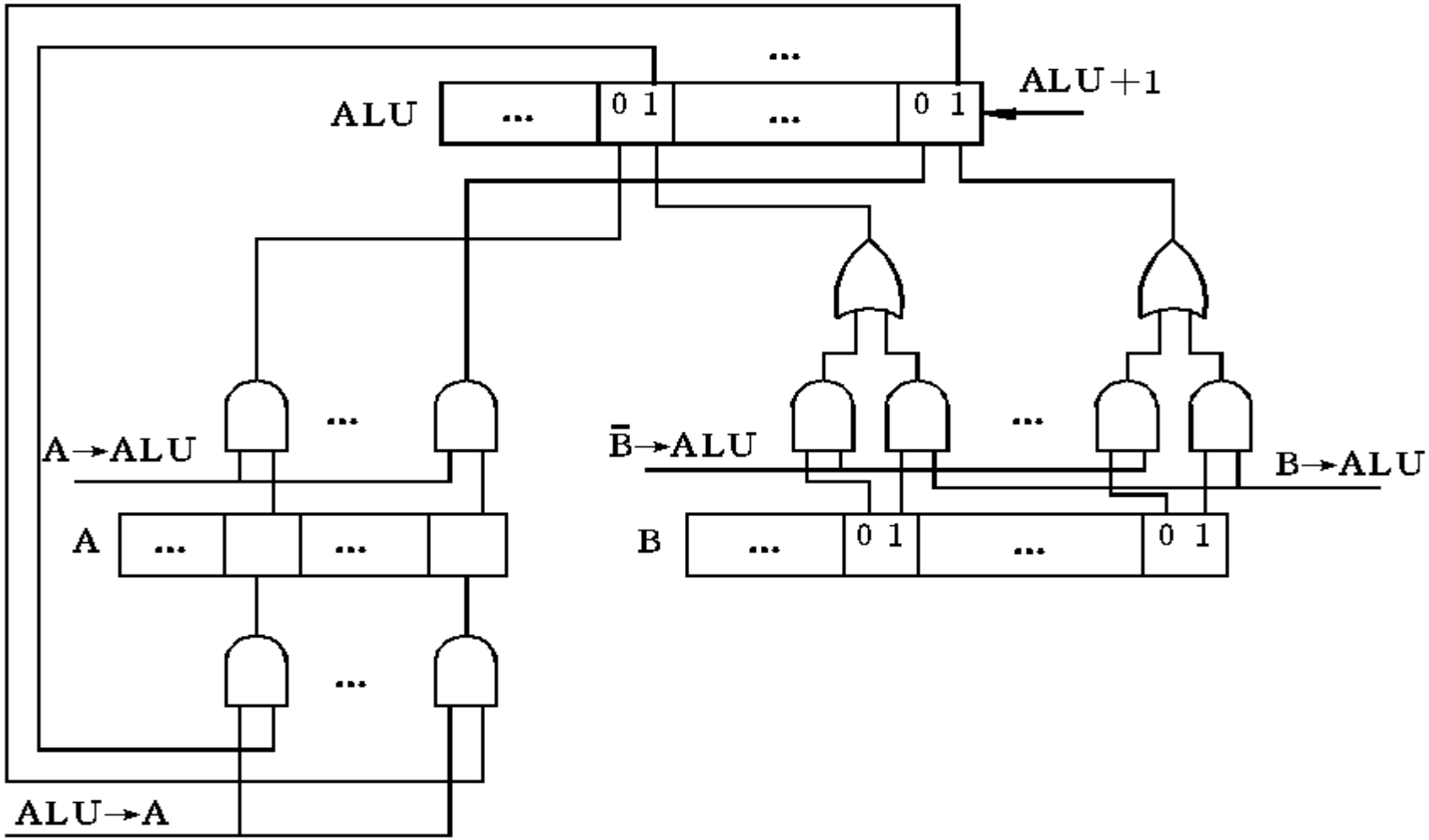


图3.1实现加法运算的逻辑示例

3. 反码表示法

机器码的最高位为符号，0表示正数，1表示负数。

反码的定义：

$$[X]_{反}=\begin{cases} X & 0\le X<1 \\ 2-2^{-n}+X & -1<X\le 0 \end{cases} \quad (3.7)$$

即：$[X]_{反}=(2-2^{-n})\cdot$符号位$+X \mod(2-2^{-n})$，

其中，n为小数点后的有效位数。

例3.20

$X=+0.1011(n=4)$,则 $[X]_{反}=0.1011$

$X=-0.1011(n=4)$,则 $[X]_{反}=2-2^{-4}+(-0.1011)=1.0100$

当X为正数时，$[X]_{反}=[X]_{原}$；当X为负数时保持[X]原符号位不变，而将数值部分取反。**反码运算是以$2-2^{-n}$为模，所以，当最高位有进位而丢掉进位(即2)时，要在最低位+1。**

例3.21 X=0.1011, Y=-0.0100，则有：

$[X]_{反}=0.1011$，$[Y]_{反}=1.1011$

$[X+Y]_{反}=[X]$反$+[Y]_{反}=[0.1011+1.1011]$反$=[\underline{1}0.0110]$

最高位1丢掉，并要在最低位加1。所以

得 $[X+Y]_{反}=0.0111 \mod(2-2^{-4})$。

例3.22 X=0.1011, Y=-0.1100,则有:

$[X]_{反}$=0.1011，$[Y]_{反}$=1.0011

$[X+Y]_{反}$= $[0.1011+1.0011]_{反}$=1.1110(其真值为-0.0001)

反码零有两种表示形式:

$[+0]_{反}$=0.0000， $[-0]_{反}$=1.1111

反码运算在最高位有进位时，要在最低位+1，此时要多进行一次加法运算，增加了复杂性，又影响了速度，因此很少采用。

正数的原码、补码和反码的表示形式是相同的，而负数则各不相同。

4. 数据从补码和反码表示形式转换成原码

(1) 将反码表示的数据转换成原码。

转换方法：符号位保持不变，正数的数值部分不变，负数的数值部分取反。

例3.23 设［X］反=0.1010,则［X］原=0.1010，真值X=0.1010。

例3.24 设［X］反=1.1010,则［X］原=1.0101，真值X=-0.0101。

(2) 将补码表示的数据转换成原码。

例3.25 设［X］$_{补}$=0.1010,则［X］$_{原}$=0.1010,真值X=0.1010。

例3.26 设［X］$_{补}$=1.1010,则［X］$_{原}$=1.0110,真值X=-0.0110。

(3) 原码和补码、反码之间相互转换的实现。

假设被转换的数X→B寄存器，结果→A寄存器中，将$[X]_{原}$→$[X]_{补}$

B寄存器的符号位f_B

f_B=0,发控制命令，B→ALU,ALU→A

f_B=1,发控制命令，$\bar{B}$→ALU,ALU+1,ALU→A

例3.27［X］原=1.1010,则［X］反=1.0101。

［X］反最低位加1，即可得［X］补=1.0110。

在计算机中，当用串行电路按位将原码转换成补码形式时(或反之)，经常采取以下方法：自低位开始转换，从低位向高位，在遇到第1个“1”之前，保持各位的“0”不变，第1个“1”也不变，以后的各位按位取反，最后保持符号位不变，经历一遍后，即可得到补码。

5. 整数的表示形式

设$X=X_n\cdots X_2X_1X_0$，其中X_n为符号位。

(1) 原码

$$[X]_{原}=\begin{cases} X & 0\le X<2^n \\ 2^n-X=2^n+|X| & -2^n<X\le 0 \end{cases} \quad (3.8)$$

(2) 补码

$$[X]_{补}=\begin{cases} X & 0\le X<2^n \\ 2^{n+1}+X=2^{n+1}-|X| & -2^n\le X<0 \end{cases} \quad (3.9)$$

(3) 反码

$$[X]_{反}=\begin{cases} X & 0\le X<2^n \\ (2^{n+1}-1)+X & -2^n<X\le 0 \end{cases} \quad (3.10)$$

例 设机器字长为16位，定点表示时，尾数15位，阶符1位。问

（1）定点原码整数表示时，最大正数为多少？最小负数为多少？

（2）定点补码整数表示时，最大正数为多少？最小负数为多少？

（3）定点原码小数表示时，最大正数为多少？最小负数为多少？

（4）定点补码小数表示时，最大正数为多少？最小负数为多少？

例 设机器字长为16位，定点表示时，尾数15位，阶符1位。问

（1）定点原码整数表示时，最大正数为多少？最小负数为多少？

（2）定点补码整数表示时，最大正数为多少？最小负数为多少？

（3）定点原码小数表示时，最大正数为多少？最小负数为多少？

（4）定点补码小数表示时，最大正数为多少？最小负数为多少？

解：（1）定点原码整数表示时

最大正数：$(2^{15}-1)=32767$；最小负数：$-(2^{15}-1)=-32767$

(2) 定点补码整数表示时

最大正数：$(2^{15}-1)=32767$；最小负数：$-2^{15}=-32768$

（3）定点原码小数表示时

最大正数：$1-2^{-15}$；最小负数：$-(1-2^{-15})$

(4)定点补码小数表示时

最大正数：$1-2^{-15}$；最小负数：-1

(4)移码

$$[X]_{移}=\begin{cases}2^n+X & 0\le X<2^n\\ 2^n+X & -2^n\le X<0\end{cases}$$

补码与移码之间的关系:

当$0\le X<2^n$时，$[X]_{移}=2^n+X=2^n+[X]_{补}$ (3.12)

当$-2^n\le X<0$时，$[X]_{移}=2^n+X=(2^{n+1}+X)-2^n=[X]_{补}-2^n$ (3.13)

因此把$[X]_{补}$的符号位取反，即得$[X]_{移}$。

例3.30 X=+1011 $[X]_{补}$=01011 $[X]_{移}$=11011

X=-1011 $[X]_{补}$=10101 $[X]_{移}$=00101

移码具有以下特点：

① 最高位为符号位，1表示正号，0表示负号。

②在计算机中，移码只执行加减法运算，且需要对得到的结果加以修正，修正量为2^n，即要对结果的符号位取反，得到 $[X]_{移}$。

$$[X]_{移}+[Y]_{移}=2^n+X+2^n+Y$$
$$=2^n+(2^n+(X+Y))$$
$$=2^n+[X+Y]_{移}$$

设X=+1010　Y=+0011,则 $[X]_{移}=11010$ $[Y]_{移}=10011$

执行加法运算

$[X]_{移}+[Y]_{移}=11010+10011=101101$,加$2^n$后得 $[X+Y]_{移}$，

$[X+Y]_{移}=01101+10000=11101$

③数据0有唯一的编码，即 $[+0]_{移}=[-0]_{移}=1000\ldots0$。

3.2.2 加减法运算的溢出处理

当运算结果超出机器数所能表示的范围时，称为**溢出**。显然，两个异号数相加或两个同号数相减，其结果是不会溢出的。仅当两个同号数相加或者两个异号数相减时，才有可能发生溢出的情况，一旦溢出，运算结果就不正确了，因此必须将溢出的情况检查出来。

1、补码溢出：以4位二进制补码正整数加法运算为例说明如下：

① 9+5=14

```
  0 1 0 0 1
+ 0 0 1 0 1
-----------
  0 1 1 1 0
```

② (-9)+(-5)= -14

```
  1 0 1 1 1
+ 1 1 0 1 1
-----------
1 1 0 0 1 0
```

③ 12+7=19(溢出)

```
  0 1 1 0 0
+) 0 0 1 1 1
-----------
  1 0 0 1 1
```

④ (-12)+(-7)=-19(溢出)

```
  1 0 1 0 0
+ 1 1 0 0 1
-----------
1 0 1 1 0 1
```

⑤ 14-1=13

```
  0 1 1 1 0
+) 0 0 0 0 1
-----------
  1 0 1 1 0 1
```

⑥-14+1= -13

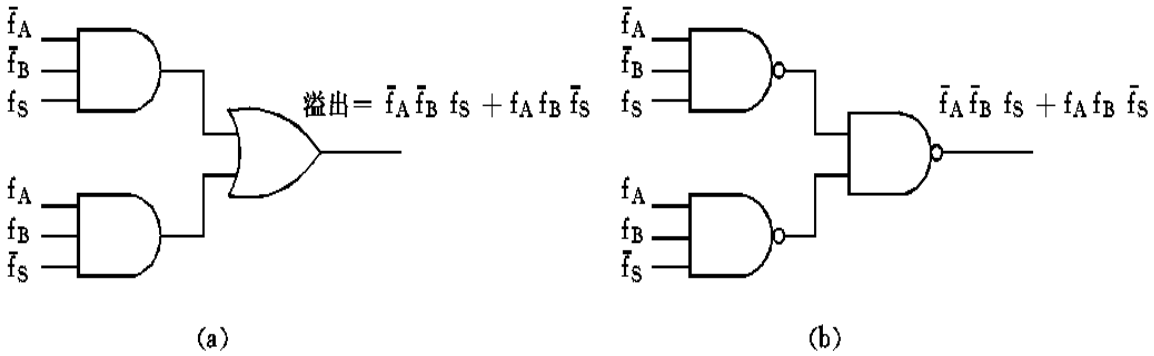
```
  1 0 0 1 0
+) 0 0 0 0 1
-----------
  1 0 0 1 1
```

在上例中，①、②、⑤和⑥得出正确结果，③和④为溢出。

- 以f_A,f_B表示两操作数(A、B)的符号位，f_S为结果的符号位。
- 符号位f_A、f_B直接参与运算，它所产生的进位以C_f表示。在以2^{n+1}为模的运算中**符号位有进位，并不一定表示溢出**
- 假如用**C来表示数值最高位产生的进位**，那么**C=1也不一定表示溢出**
- 究竟如何判断溢出，实现时有多种方法可供选择，采用其中一种方法即可，今将判别溢出的几种方法介绍如下：

  (1)当符号相同的两数相加时，**如果结果的符号与加数(或被加数)不相同，则为溢出**。即溢出条件=$f_A\bar{f}_B\bar{f}_S+\bar{f}_Af_Bf_S$。

  (2)当任意符号两数相加时，如果$C=C_f$,运算结果正确。如果$C\neq C_f$，则为溢出，所以溢出条件=$C\oplus C_f$。



•(3) 采用双符号位$f_{S1}f_{S2}$。正数的双符号位为00，负数的双符号位为11。符号位参与运算，当结果的两个符号位f_{S1}和f_{S2}不相同时，为溢出。所以溢出条件=$f_{S1}\oplus f_{S2}$，或者溢出条件=$f_{S1}\overline{f}_{S2}+\overline{f}_{S1}f_{S2}$。

① 9+5=14

0 0 1 0 0 1

\+ 0 0 0 1 0 1

0 0 1 1 1 0

C=C_f=0 不溢出，

或$f_{S1}=f_{S2}$ 不溢出

② 12+7=19

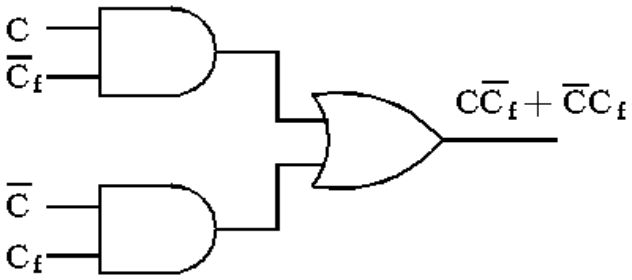
0 0 1 1 0 0

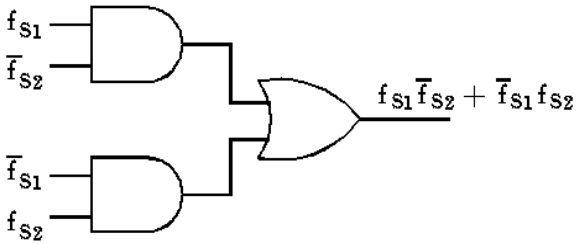
\+ 0 0 0 1 1 1

0 1 0 0 1 1

C=1,C_f=0溢出，

或$f_{S1}\neq f_{S2}$溢出





2、移码溢出

根据补码定义： $[Y]_{补}=2^{n+1}+Y \mod 2^{n+1}$

对同一个数值，**移码和补码的数值位完全相同，而符号位正好相反**。移码的加减法：

$$[X]_{移}+[Y]_{补}=2^{n}+X+2^{n+1}+Y$$
$$=2^{n+1}+(2^{n}+(X+Y))=[X+Y]_{移} \mod 2^{n+1}$$

同理有 $[X]_{移}+[-Y]_{补}=[X-Y]_{移}$。

如果运算的结果溢出，上述条件则不成立。

使用双符号位的加法器，并**规定移码的第二个符号位，即最高符号位恒用0参加加减运算.**

（1）溢出条件是结果的最高符号位为1。此时，当低位符号位为0时，表明结果上溢，为1时，表明结果下溢。

（2）最高符号位为0时，表明没有溢出，低位符号位为1，表明结果为正，为0时，表明结果为负。

假定阶码用四位表示，则其表示范围为-8到+7

当X=+011,Y=+110时，则有：

$[X]_{移}$=01011,$[Y]_{补}$=00110，$[-Y]_{补}$=11010

①$[X+Y]_{移}=[X]_{移}+[Y]_{补}$=10001，结果上溢

②$[X-Y]_{移}=[X]_{移}+[-Y]_{补}$=00101，结果正确，为-3

当X=-011,Y=-110时，则有：

$[X]_{移}$=00101,$[Y]_{补}$=11010，$[-Y]_{补}$=00110

①$[X+Y]_{移}=[X]_{移}+[Y]_{补}$=11111，结果下溢

②$[X-Y]_{移}=[X]_{移}+[-Y]_{补}$=01011，结果正确，为+3

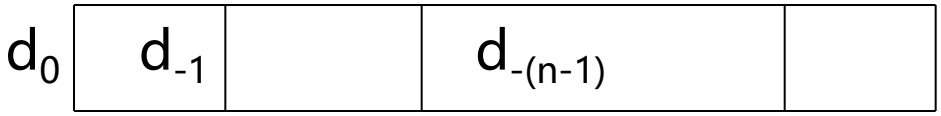
- **3.2.3 定点数和浮点数**
- **1. 定点数**
- 定点数是指小数点固定在某个位置上的数据，一般有小数和整数两种表示形式。
- 定点小数是把小数点固定在数据数值部分的左边，符号位的右边；

∧

原码的表示范围为 $-(1-2^{-n}) \le x \le 1-2^{-n}$

补码的表示范围为 $-1 \le x \le 1-2^{-n}$

- 整数是把小数点固定在数据数值部分的右边。



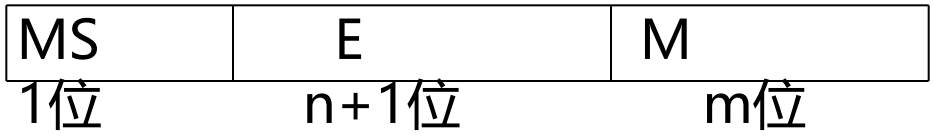
∧

原码的表示范围为 $-(2^n-1) \le x \le 2^n-1$

补码的表示范围为 $-2^n \le x \le 2^n-1$

2. 浮点数

浮点数是指小数点位置可浮动的数据，通常以下式表示：$N=M\cdot R^E$
其中，N为浮点数，M(mantissa)为尾数，E（exponent)为阶码，R(radix)称为“阶的基数(底)”，而且R为一常数，一般为2、8或16。浮点数的机内表示一般采用以下形式：



- MS是尾数的符号位，设置在最高位上。
- E为阶码，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，用来表示正阶或负阶。
- M为尾数，有m位，由MS和M组成一个定点小数。MS=0，表示正号，MS=1，表示负号。
- 在多数通用机中，浮点数的**尾数用原码或补码表示**，**阶码用补码或移码表示**。

浮点数的**尾数用补码表示**，**阶码用补码表示:**

阶符　阶码值　　数符　　尾数值

补码表示　　补码表示

浮点数的**尾数用补码表示**，**阶码用移码表示:**

数符　阶符　　阶码值　　尾数值

移码表示　　补码表示

（1）浮点数的规格化

为了保证数据精度，尾数通常用规格化形式表示：当R=2，且尾数值不为0时，其绝对值应大于或等于$(0.5)_{10}$。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。

①原码规格化后

正数为 0.1xx…xx

负数为 1.1xx...xx

②补码规格化后

正数为 0.1xx…xx

负数为 1.0xx...xx

例如：浮点数的尾数为0.0011，阶码为0100(设定R=2)，规格化时，将尾数左移2位，而成为0.1100，阶码减去$(10)_2$，修改成0010，浮点数的值保持不变。

(2) 溢出问题

当一个浮点数阶码大于机器的最大阶码时，称为上溢，小于最小阶码时，称为下溢。

当一个浮点数的尾数为0(不论阶码是何值)，或阶码的值比能在机器中表示的最小值还小时，计算机都把该浮点数看成零值，称为机器零。

当数据小于机器能表示的最小数时(移码≤-2^n)，称为机器零，将阶码(移码)置为0000…0，且不管尾数值大小如何，都按浮点数下溢处理。

（3）浮点数的表示范围

设指数部分（阶码）m位（含一位符号位），尾数部分n位（含一位符号位)

$a=2^{(m-1)}-1$，$b=-2^{(m-1)}$ 则规格化数所能表示的范围如下：

最大正数 $(1-2^{-(n-1)})\times 2^{a}$

最小正数 $2^{-1}\times 2^{b}$

最大负数 $-(2^{(n-1)}+2^{-1})\times 2^{b}$

最小负数 -2^{a}

浮点的数的阶码决定了浮点数表示的范围，浮点数的尾数决定了浮点数的表示精度

例1 某机字长32位，采用IEEE单精度格式，则浮点法表示的最大正数和最小负数

m=8, n=24

例1 某机字长32位，采用IEEE单精度格式，则浮点法表示的最大正数和最小负数

m=8, n=24

$a=2^{(m-1)}-1=128-1=127$

$b=-2^{(m-1)}=-128$

最大正数 $(1-2^{-23})\ 2^{127}$

最小负数 -2^{127}

(4) 根据IEEE 754国际标准，常用的**浮点数有两种格式**：

- **单精度浮点数(32位)**，阶码8位，尾数24位(内含1位符号位)。
  数值为 $(-1)^s \times 1.f \times 2^{e-127}$
- **双精度浮点数(64位)**，阶码11位，尾数53位(内含1位符号位)。
  数值为 $(-1)^s \times 1.f \times 2^{e-1023}$

其中 s=0 表示正数， s=1 表示负数

单精度e的取值为1--254（8位表示），f为23位，共32位

双精度e的取值为1--2046（11位表示），f为52位，共64位

例：写出下列十进制的IEEE754单精度编码

（1）0.15625 (2) -5

(1) 0.15625 二进制值 0.00101

IEEE754中的规格化表示为 1.01×2^{-3} e=127-3=124

编码为：0 01111100 01000000000000000000000

(2) -5 二进制值 -101

IEEE754中的规格化表示为 1.01×2^{2} e=127+2=129

编码为：1 10000001 01000000000000000000000

作业：

1、8位二进制补码表示整数的最小值和最大值?

2、8位二进制反码码表示整数的最小值和最大值?

3、若单精度浮点数IEEE754代码为 00111111010000000000000000000000 则其代表的十进制数是多少?

4、写出下列十进制的IEEE754单精度编码

（1）-0.15625（2) 16

3.3 二进制乘法运算

3.3.1 定点数一位乘法

1. 定点原码一位乘法

两个原码数相乘，其乘积的符号为相乘两数的异或值，数值则为两数绝对值之积。

假设 $[X]_{原}=X_0X_1X_2\ldots X_n$

$[Y]_{原}=Y_0Y_1Y_2\ldots Y_n$

则 $[X\cdot Y]_{原}=[X]_{原}\cdot[Y]_{原}$

$=(X_0\oplus Y_0)\,|\,(X_1X_2\ldots X_n)\cdot(Y_1Y_2\ldots Y_n)$

符号 | 表示把符号位和数值邻接起来。

例3.31 X=0.1101，Y=0.1011，计算乘积X·Y。

```
      0.1101
    ×  0.1011
    ----------
        1101
       1101
      0000
     1101
----------
0.10001111
```

即X·Y=0.10001111，符号为正。

- 在计算时，逐次按乘数每1位上的值是1还是0，决定相加数取被乘数的值还是取零值，而且相加数逐次向左偏移1位，最后一起求积。
- 运算方法在人工计算的基础上作了以下修改。

(1) 在机器内多个数据一般不能同时相加，一次加法操作只能求出两数之和，因此每求得一个相加数，就与上次部分积相加。

(2) 观察计算过程很容易发现，在求本次部分积时，前一次部分积的最低位，不再参与运算，因此可将其右移一位，相加数可直送而不必偏移，于是用N位加法器就可实现两个N数相乘。

(3) 部分积右移时，乘数寄存器同时右移一位，这样可以用乘数寄存器的最低位来控制相加数(取被乘数或零)，同时乘数寄存器的最高位可接收部分积右移出来的一位，因此，完成乘法运算后，A寄存器中保存乘积的高位部分，乘数寄存器中保存乘积的低位部分。

例3.32 设X=0.1101，Y=0.1011，求X·Y。

解： 计算过程如下：取双符号位

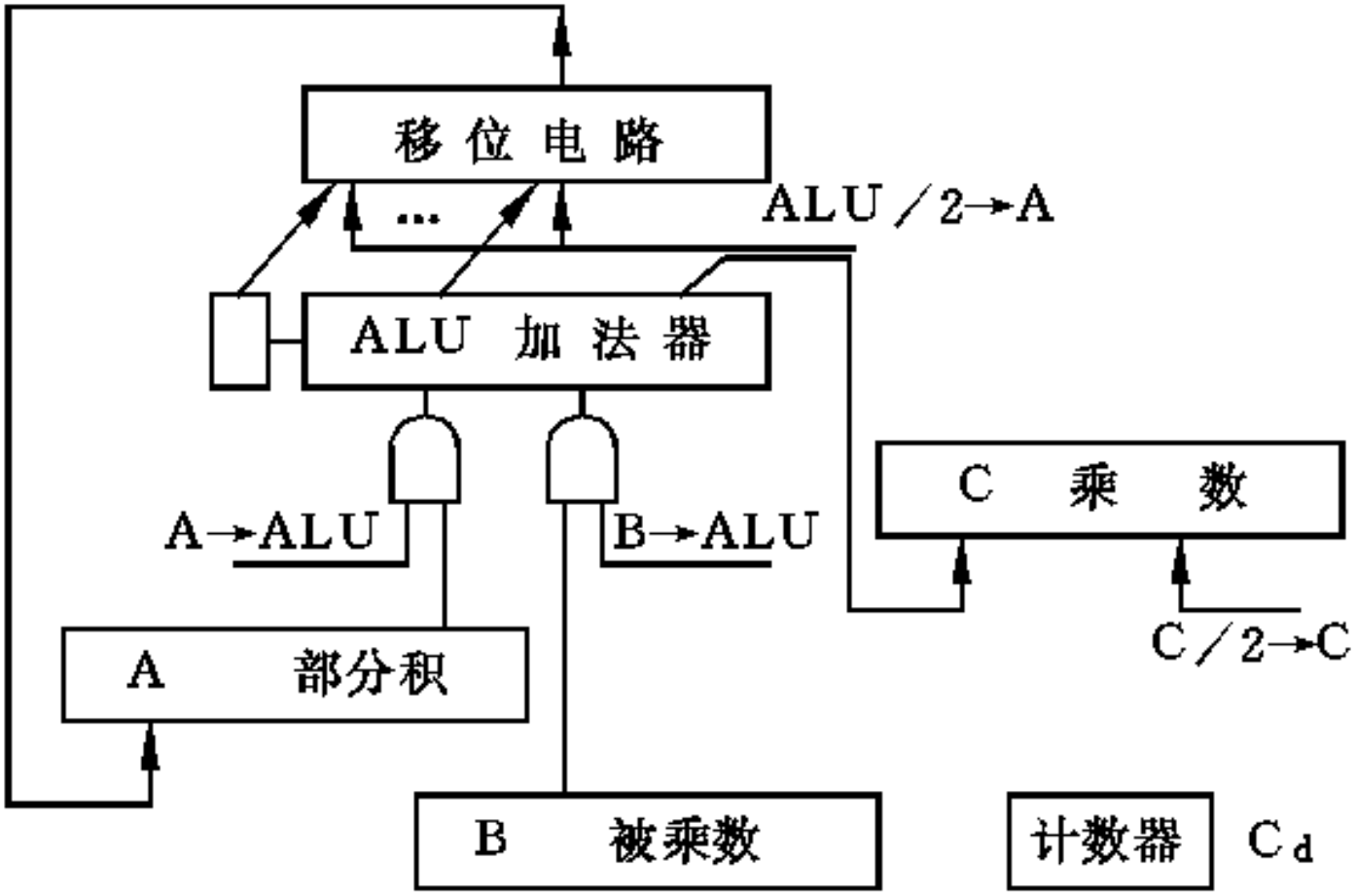
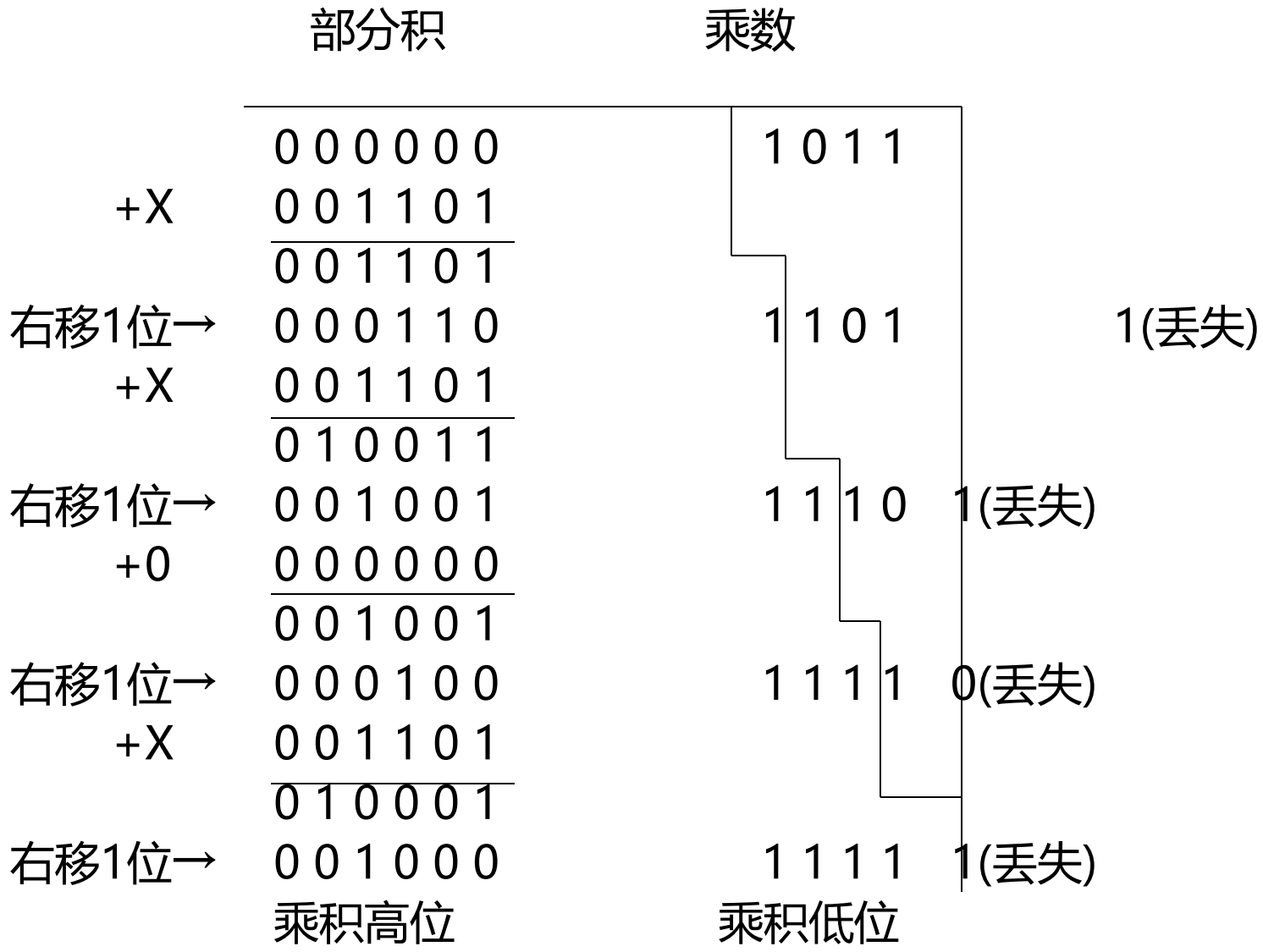


图3.5 实现原码一位乘法的逻辑电路框图


部分积
乘数
0 0 0 0 0 0
1 0 1 1
+X
0 0 1 1 0 1
0 0 1 1 0 1
右移1位→
0 0 0 1 1 0
1 1 0 1
1(丢失)
+X
0 0 1 1 0 1
0 1 0 0 1 1
右移1位→
0 0 1 0 0 1
1 1 1 0
1(丢失)
+0
0 0 0 0 0 0
0 0 1 0 0 1
右移1位→
0 0 0 1 0 0
1 1 1 1
0(丢失)
+X
0 0 1 1 0 1
0 1 0 0 0 1
右移1位→
0 0 1 0 0 0
1 1 1 1
1(丢失)
乘积高位
乘积低位

$$X \cdot Y = 0.10001111$$

乘积的符号位=$X_0 \oplus Y_0 = 0 \oplus 0 = 0$,乘积为正数。

乘法运算的控制流程图如图3.6所示。该图中，数据的位序号从左至右按0，1，…，n的次序编，0位表示符号，共n位数值。

从流程图上可以清楚地看到，这里的原码一位乘是通过循环迭代的办法实现的。每次迭代得到的部分积(P_0，P_1,…，P_n)可用下述式(3.14)表示：

$$\begin{cases} P_0 = 0 \\ P_1 = (P_0 + XY_n)2^{-1} \\ P_2 = (P_1 + XY_{n-1})2^{-1} \\ \cdots \\ P_{i+1} = (P_i + XY_{n-i})2^{-1} \\ \cdots \\ P_n = (P_{n-1} + XY_1)2^{-1} \end{cases} \qquad (3.14)$$

P_n为乘积。2^{-1}表示二进制数据右移一位，相当于乘以2^{-1}。

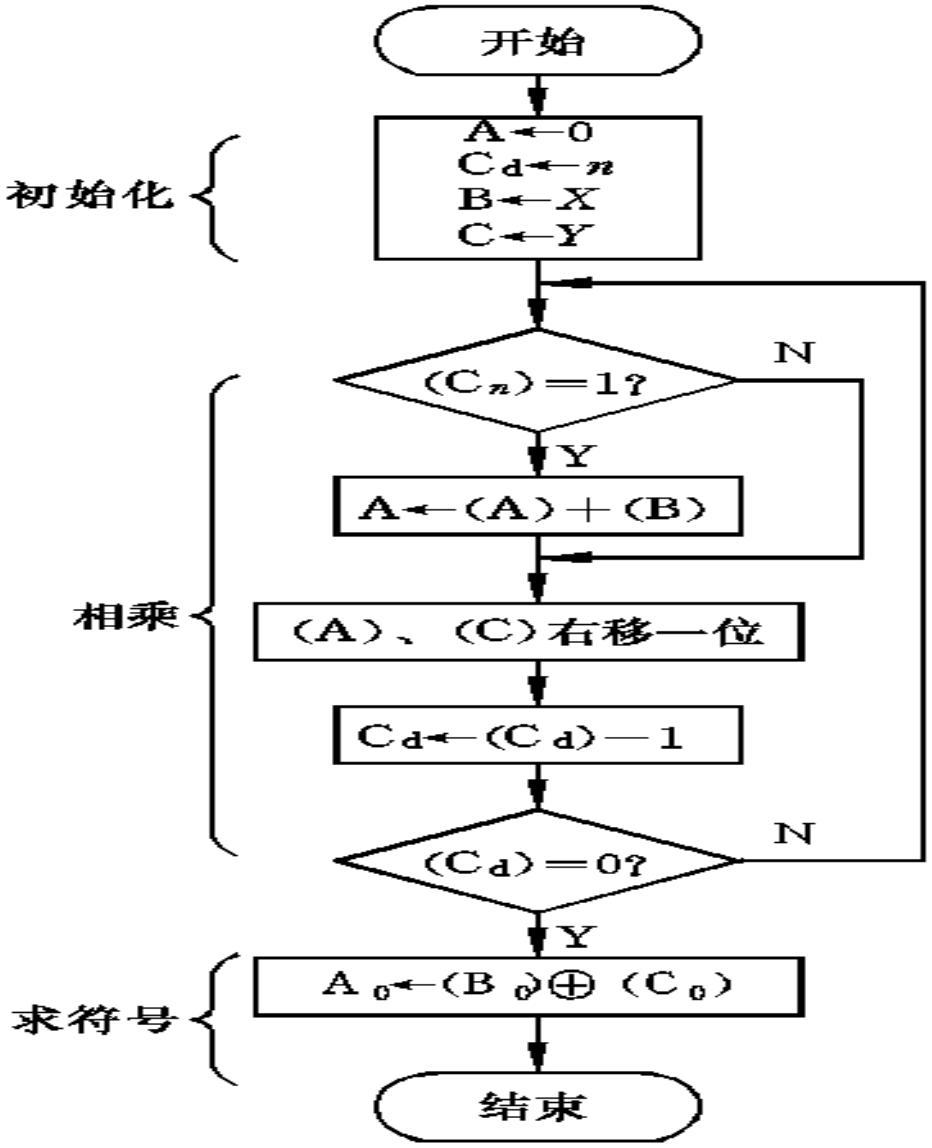


图3.6 乘法运算的控制流程

原码乘法的规则：

1、被乘数和乘数均取绝对值参加运算，符号位单独考虑

2、被乘数取双符号，部分积的长度同被乘数，初值为0

3、从乘数的最低位y_n开始判断，若$y_n=1$,则部分积加上被乘数|x|,然后右移一位，移入乘数寄存器的最高数值位。若 $y_n=0$，则部分积加上被乘数0,然后右移一位，移入乘数寄存器的最高数值位。

4、重复3，判断n次。最后的乘积。高位在部分积中，低位在乘数寄存器中。乘数在逐位移位中丢失

2. 定点补码一位乘法

(1) 补码与真值的转换关系

设 $[X]_{补}=X_0.X_1X_2\ldots X_n$，

当$X\geq 0$时，$X_0=0$，

$$[X]_{补}=0.X_1X_2\ldots X_n=\sum_{i=1}^{n}X_i2^{-i}$$

当$X<0$时，$X_0=1$，

$$[X]_{补}=1.X_1X_2\ldots X_n=2+X$$

$$X=[X]_{补}-2=-1+0.X_1X_2\ldots X_n=-1+\sum_{i=1}^{n}X_i2^{-i}$$

$$X=-X_0+\sum_{i=1}^{n}X_i2^{-i}=-X_0+0.X_1X_2\ldots X_n \qquad (3.15)$$

(2) 补码的右移

在补码运算的机器中，不论数的正负，连同符号位将数右移一位，并保持符号位不变，相当于乘1/2（或除2）。

设$[X]_{补}= X_0.X_1X_2\ldots X_n$

$X= =-X_0+ X\sum_{i=1}^{n}2^{-i}$

(3) 补码一位乘法

设被乘数 $[X]_{补}=X_0.X_1X_2\ldots X_n$,乘数 $[Y]_{补}=Y_0.Y_1Y_2\ldots Y_n$,则有

$$[X\cdot Y]_{补}= [X]_{补}\cdot(-Y_0+\sum_{i=1}^{n} Y_i2^{-i}) \qquad (3.16)$$

例3.33 设X=-0.1101，Y=0.1011

即：$[X]_{补}$=11.0011，$[Y]_{补}$=Y=0.1011，求$[X\cdot Y]_{补}$

部分积	乘数	说明
00.0000	1 0 1 1	初始值
$+[X]_{补}$ 11.0011		$+[X]_{补}$
11.0011		
右移一位 11.1001	1 1 0 1	右移一位
$+[X]_{补}$ 11.0011		$+[X]_{补}$
10.1100		
右移一位 11.0110	0 1 1 0	右移一位
+ 00.0000		+0
11.0110		
右移一位 11.1011	0 0 1 1	右移一位
$+[X]_{补}$ 11.0011		$+[X]_{补}$
10.1110		
右移一位 11.0111	0 0 0 1	右移一位

例3.34 X=-0.1101，Y=-0.1011

即：$[X]_{补}$=11.0011 $[Y]_{补}$=11.0101 求 $[X\cdot Y]_{补}$

部分积	乘数	说明
00.0000	0 1 0 1	初始值
$+[X]_{补}$ 11.0011		$+[X]_{补}$
11.0011		
右移一位 11.1001	1 0 1 0	右移一位
+0 00.0000		+0
11.1001		
右移一位 11.1100	1 1 0 1	右移一位
$+[X]_{补}$ 11.0011		$+[X]_{补}$
10.1111		
右移一位 11.0111	1 1 1 0	右移一位
+0 00.0000		+0
11.0111		
右移一位 11.1011	1 1 1 1	右移一位
$+[-X]_{补}$ 00.1101		$+[-X]_{补}$
00.1000	1 1 1 1	

将前述补码乘法公式进行变换，可得出另一公式，是由布斯(Booth)提出的，又称为“布斯公式”。

$$[X\cdot Y]_{补}=[X]_{补}\cdot(-Y_0+\sum_{i=1}^{n}Y_i\cdot 2^{-i})$$

$$=[X]_{补}\cdot[-Y_0+Y_1 2^{-1}+Y_2 2^{-2}+\ldots+Y_n\cdot 2^{-n}]$$
$$=[X]_{补}\cdot[-Y_0+(Y_1-Y_1 2^{-1})+(Y_2 2^{-1}-Y_2 2^{-2})+\ldots+(Y_n 2^{-(n-1)}-Y_n 2^{-n})]$$
$$=[X]_{补}\ [(Y_1-Y_0)+(Y_2-Y_1)2^{-1}+\ldots\ +(Y_n-Y_{n-1})2^{-(n-1)}+(0-Y_n)2^{-n}]$$
$$=[X]_{补}\ (\sum_{i=0}^{n}Y_{i+1}-Y_i)2^{-i} \qquad (3.17)$$

乘数的最低1位为Y_n，在其后面再添加1位Y_{n+1},其值为0。

将式(3.17)加以变换：按机器执行顺序求出每一步的部分积。

$[P_0]_{补}=0$

$[P_1]_{补}=\{[P_0]_{补}+(Y_{n+1}-Y_n)[X]_{补}\}2^{-1}\quad Y_{n+1}=0$

$[P_2]_{补}=\{[P_1]_{补}+(Y_n-Y_{n-1})[X]_{补}\}2^{-1}$

…

$[P_i]_{补}=\{[P_{i-1}]_{补}+(Y_{n-i+2}-Y_{n-i+1})[X]_{补}\}2^{-1}$

…

$$[P_n]_{补} = \{ [P_{n-1}]_{补} + (Y_2 - Y_1)\ [X]_{补} \}\ 2^{-1}$$
$$[P_{n+1}]_{补} = \{ [P_n]_{补} + (Y_1 - Y_0)\ [X]_{补} \} = [X \cdot Y]_{补}$$

Y_{i+1}与Y_i为相邻两位，$(Y_{i+1}-Y_i)$有0，1和-1三种情况，其运算规则如下：

(1) $Y_{i+1}-Y_i=0$($Y_{i+1}Y_i$=00或11)，部分积加0，右移1位

(2) $Y_{i+1}-Y_i=1$($Y_{i+1}Y_i$=10)，部分积加 $[X]_{补}$，右移 1位

(3) $Y_{i+1}-Y_i=-1$($Y_{i+1}Y_i$=01)，部分积加 $[-X]_{补}$，右移1位

最后一步(i=n+1)不移位

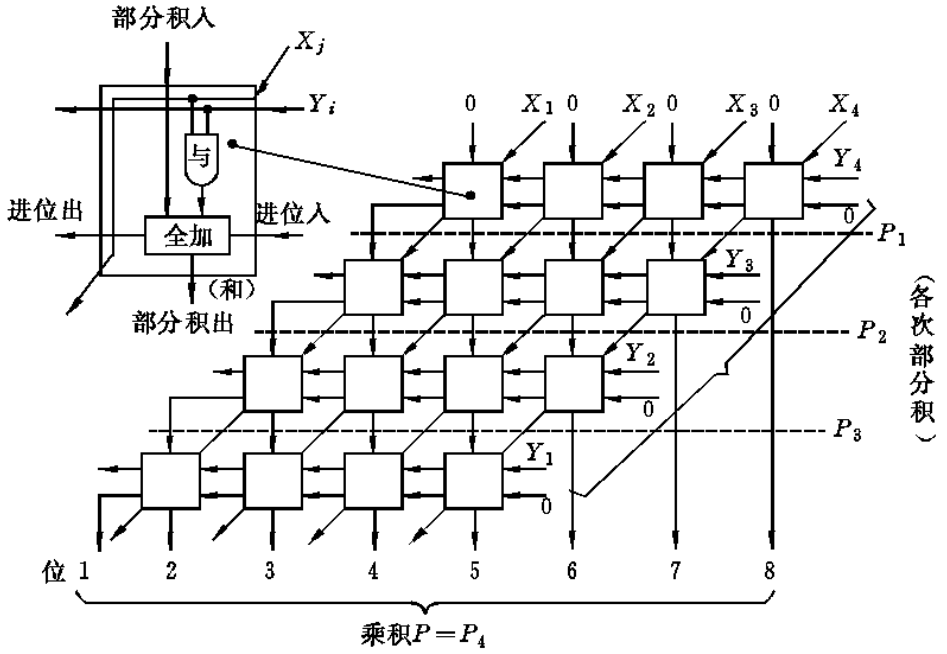
例3.35 设X=-0.1101，Y=0.1011

即：$[X]_{补}$=11.0011，$[Y]_{补}$=Y=0.1011，求$[X\cdot Y]_{补}$

部分积	乘数	说明
00 0000	0.1 0 1 1 0	初始值 最后一位补0
+ 00 1101		Y_5Y_4=01，$+[-X]_{补}$
00 1101		
右移一位 00 0110	1 0 1 0 1 1	右移一位
+ 00 0000		Y_4Y_3=11，+0
00 0110		
右移一位 00 0011	0 1 0 1 0 1	右移一位
+ 11 0011		Y_3Y_2=10，$+[X]_{补}$
11 0110		
右移一位 11 1011	0 0 1 0 1 0	右移一位
+ 00 1101		Y_2Y_1=01，$+[-X]_{补}$
00 1000		
右移一位 00 0100	0 0 0 1 0 1	右移一位
+ 11 0011		Y_1Y_0=10，$+[X]_{补}$
11 0111	0 0 0 1	

3.3.3 阵列乘法器

为了进一步提高乘法运算速度，可采用类似于人工计算的方法，用一个阵列乘法器完成X·Y乘法运算($X=X_1X_2X_3X_4, Y=Y_1Y_2Y_3Y_4$)。阵列的每一行送入乘数Y的每一位数位，而各行错开形成的每一斜列则送入被乘数的每一数位。图中每一个方框包括一个与门和一位全加器。该方案所用加法器数量很多，但内部结构规则性强，适于用超大规模集成电路实现。



3.4 二进制除法运算

3.4.1 定点除法运算

1. 定点原码一位除法

有恢复余数法和加减交替法两种方法。

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)恢复余数法

人工进行二进制除法的规则：判断被除数与除数的大小，若被除数小，则上商0，并在余数最低位补0，再用余数和右移一位的除数比，若够除，则上商1，否则上商0。然后继续重复上述步骤，直到除尽(即余数为零)或已得到的商的位数满足精度要求为止。

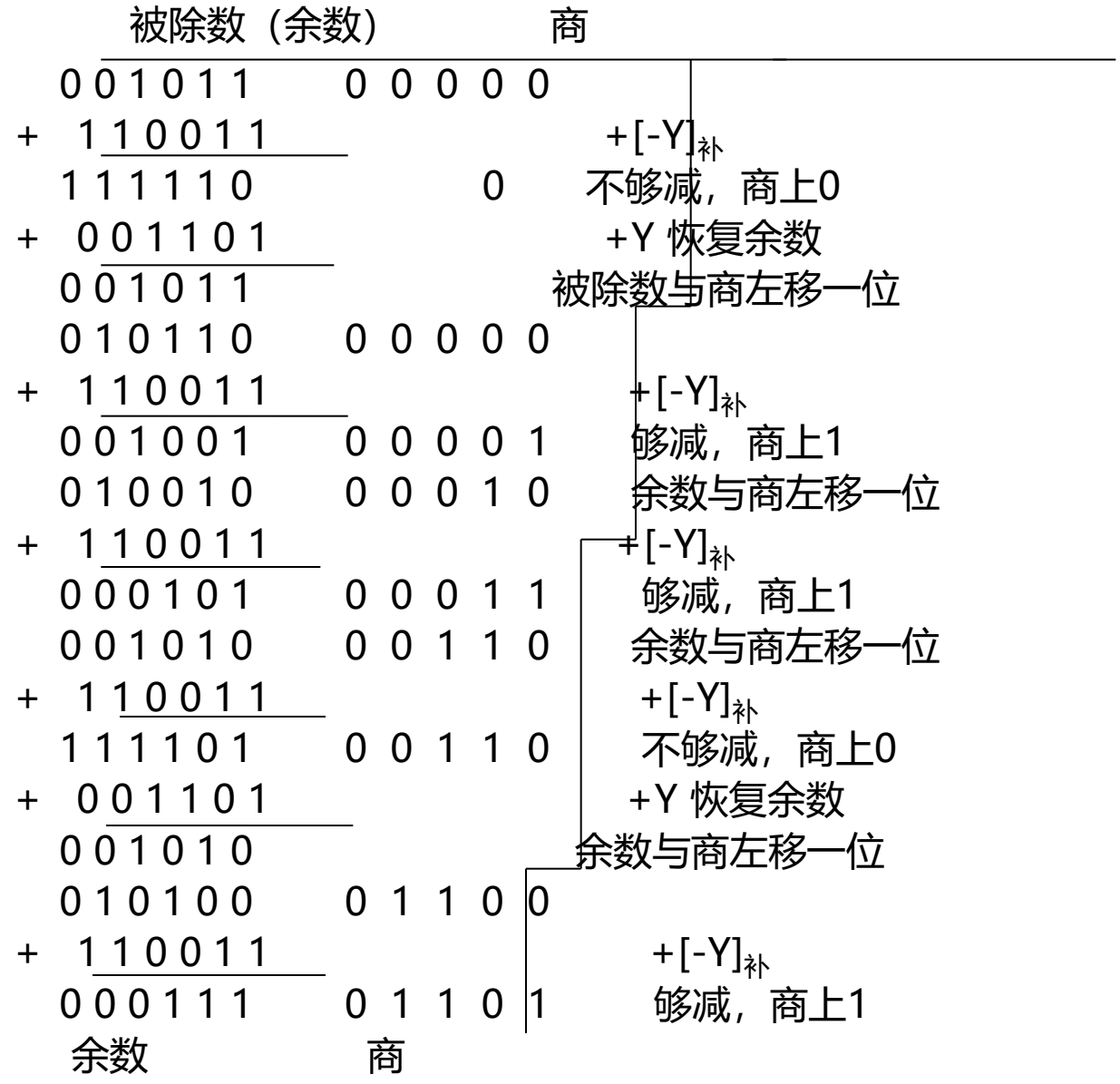
右移除数，可以通过左移被除数(余数)来替代，左移出界的被除数(余数)的高位都是无用的零，对运算不会产生任何影响。另外，上商0还是1是计算者用观察比较的办法确定的，而在计算机中，只能用做减法判结果的符号为负还是为正来确定。

假设被除数X=0.1011,除数Y=0.1101

```
          0. 1 1 0 1
        ┌──────────
0.1101  │ 0. 1 0 1 1 0
          1 1 0 1
          ─────────
          1 0 0 1 0
            1 1 0 1
            ─────────
            1 0 1 0 0
              1 1 0 1
              ─────────
              0 1 1 1
```

例3.38 假设X=0.1011,Y=0.1101,求X/Y。

解：　[-Y]$_{补}$=11.0011，得出：X/Y=0.1101　余数=0.0111×2^{-4}

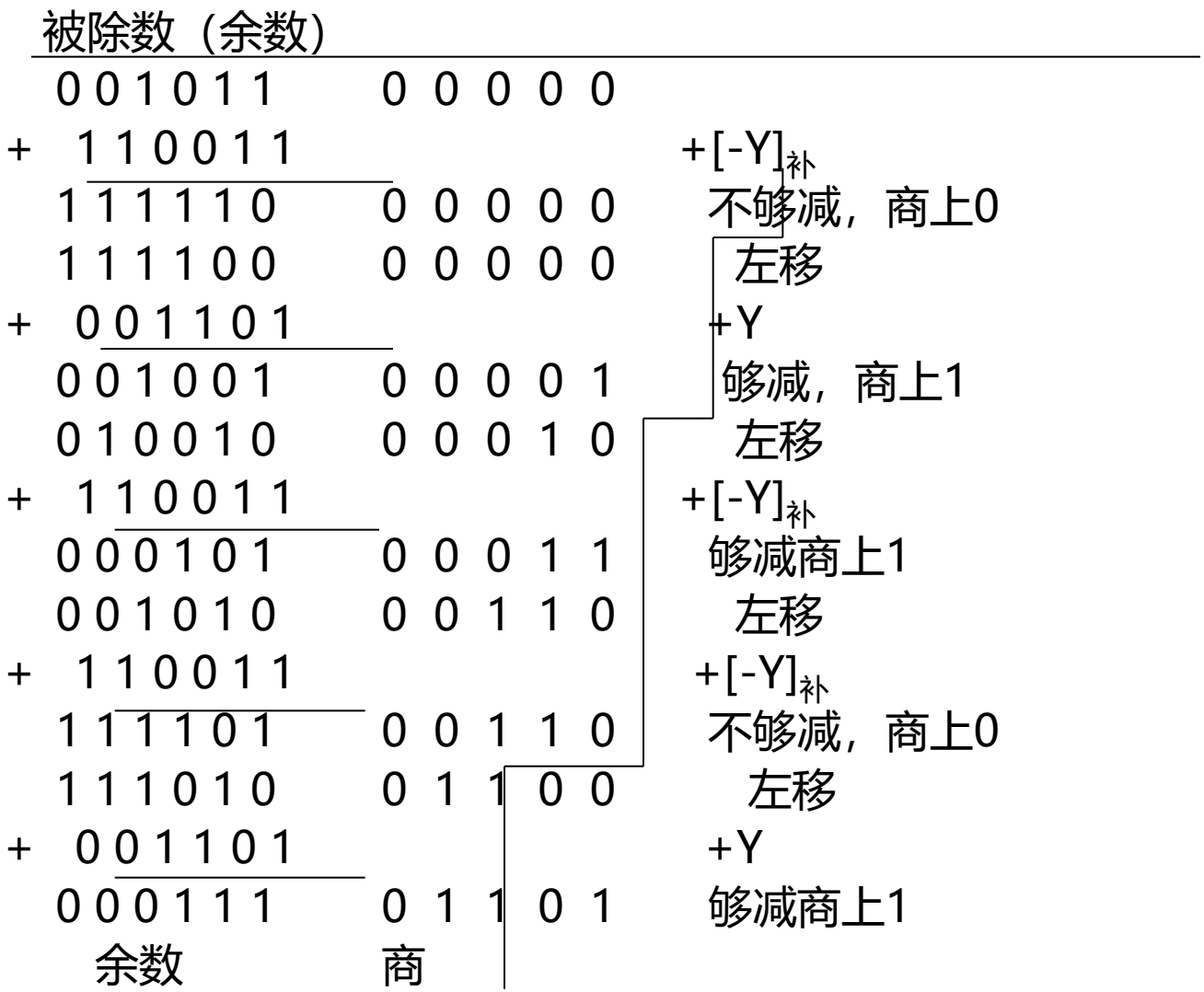

被除数（余数）
商
0 0 1 0 1 1
0 0 0 0 0
+ 1 1 0 0 1 1
+[-Y]补
1 1 1 1 1 0
0
不够减，商上0
+ 0 0 1 1 0 1
+Y 恢复余数
0 0 1 0 1 1
被除数与商左移一位
0 1 0 1 1 0
0 0 0 0 0
+ 1 1 0 0 1 1
+[-Y]补
0 0 1 0 0 1
0 0 0 0 1
够减，商上1
0 1 0 0 1 0
0 0 0 1 0
余数与商左移一位
+ 1 1 0 0 1 1
+[-Y]补
0 0 0 1 0 1
0 0 0 1 1
够减，商上1
0 0 1 0 1 0
0 0 1 1 0
余数与商左移一位
+ 1 1 0 0 1 1
+[-Y]补
1 1 1 1 0 1
0 0 1 1 0
不够减，商上0
+ 0 0 1 1 0 1
+Y 恢复余数
0 0 1 0 1 0
余数与商左移一位
0 1 0 1 0 0
0 1 1 0 0
+ 1 1 0 0 1 1
+[-Y]补
0 0 0 1 1 1
0 1 1 0 1
够减，商上1
余数
商

(2) 加减交替法

在恢复余数除法中，若第i-1次求商的余数为R_{i-1},下一次求商的余数为R_i，$R_i=2R_{i-1}-Y$

Ri>0，$R_{i+1}=2R_i-Y$

Ri<0, R_{i+1} .$R_{i+1}=2(R_i+Y)-Y=2R_i+Y$


被除数（余数）
0 0 1 0 1 1 0 0 0 0 0
+ 1 1 0 0 1 1 +[-Y]补
1 1 1 1 1 0 0 0 0 0 0 不够减，商上0
1 1 1 1 0 0 0 0 0 0 0 左移
+ 0 0 1 1 0 1 +Y
0 0 1 0 0 1 0 0 0 0 1 够减，商上1
0 1 0 0 1 0 0 0 0 1 0 左移
+ 1 1 0 0 1 1 +[-Y]补
0 0 0 1 0 1 0 0 0 1 1 够减商上1
0 0 1 0 1 0 0 0 1 1 0 左移
+ 1 1 0 0 1 1 +[-Y]补
1 1 1 1 0 1 0 0 1 1 0 不够减，商上0
1 1 1 0 1 0 0 1 1 0 0 左移
+ 0 0 1 1 0 1 +Y
0 0 0 1 1 1 0 1 1 0 1 够减商上1
余数 商

最后说明如下：

(1) 对定点小数除法，首先要比较除数和被除数的绝对值的大小，以检查是否出现商溢出的情况。

(2) 商的符号为相除二数的符号的半加和。

(3) 被除数的位数可以是除数的两倍，其低位的数值部分开始时放在商寄存器中。运算过程中，放被除数和商的寄存器同时移位，并将商寄存器中的最高位移到被除数寄存器的最低位中。

(4) 实现除法的逻辑电路与乘法的逻辑电路(图3.5)极相似，但在A寄存器中放被除数/余数，B寄存器中放除数，C寄存器放商(如被除数为双倍长，在开始时C中放被除数的低位)。此外，移位电路应有左移1位的功能，以及将Y/［-Y］$_{补}$送ALU的电路。

2. 定点补码一位除法(加减交替法)

在被除数的绝对值小于除数的绝对值(即商不溢出)的情况下，补码一位除法的运算规则如下：

(1) 如果被除数与除数同号，用被除数减去除数；若两数异号，用被除数加上除数。如果所得**余数与除数同号上商1，若余数与除数异号，上商0，该商即为结果的符号位。**

(2) 求商的数值部分。**如果上次上商1，将余数左移一位后减去除数；如果上次上商0，将余数左移一位后加上除数。然后判断本次操作后的余数，**如果余数与除数同号上商1；若余数与除数异号上商0。如此重复执行n-1次(设数值部分有n位)。

(3) 商的最后一位一般采用恒置1的办法，并省略了最低位+1的操作，此时最大误差为$\pm 2^{-n}$。如果对商的精度要求较高，则可按规则(2) 再进行一次操作，以求得商的第n位。当除不尽时，若商为负，要在商的最低一位加1，使商从反码值转变成补码值；若商为正，最低位不需要加1。

例3.40 设 $[X]_{补}=1.0111$，$[Y]_{补}=0.1101$，求 $[X/Y]_{补}$。

解： $[-Y]_{补}=11.0011$ $[X/Y]_{补}=1.0101$

例3.40最低位恒置1，余数不正确。如不采用恒置1的方法(采用反码+1的方法)，所得结果 $[X/Y]_{补}$仍为1.0101。

被除数（余数）　　商

1 1 0 1 1 1　　0 0 0 0 0

\+ 0 0 1 1 0 1　　两数异号　+[Y]补

0 0 0 1 0 0　　0 0 0 0 1　余数与除数同号，商上1

0 0 1 0 0 0　　0 0 0 1 0　左移

\+ 1 1 0 0 1 1　　上次商1，+[-Y]补

1 1 1 0 1 1　　0 0 0 1 0　余数与除数异号，商上0

1 1 0 1 1 0　　0 0 1 0 0　左移

\+ 0 0 1 1 0 1　　+[Y]补

0 0 0 0 1 1　　0 0 1 0 1　余数与除数同号，商上1

0 0 0 1 1 0　　0 1 0 1 0　左移

\+ 1 1 0 0 1 1　　+[-Y]补

1 1 1 0 0 1　　0 1 0 1 0　余数与除数异号，商上0

1 1 0 0 1　　1 0 1 0 1　左移，商的最低位恒置1

例3.41 设X=0.0100,Y=-0.1000,求 $[X/Y]_{补}$。

解: $[X_{补}]=00.0100$　$[Y]_{补}=11.1000$　$[-Y]_{补}=00.1000$

被除数（余数）	商	
0 0 0 1 0 0	0 0 0 0 0	
+ 1 1 1 0 0 0		两数异号 $+[Y]_{补}$
1 1 1 1 0 0	0 0 0 0 1	余数与除数同号，商上1
1 1 1 0 0 0	0 0 0 1 0	左移
+ 0 0 1 0 0 0		上次商1，$+[-Y]_{补}$
0 0 0 0 0 0	0 0 0 1 0	余数与除数异号，商上0
0 0 0 0 0 0	0 0 1 0 0	左移
+ 1 1 1 0 0 0		$+[Y]_{补}$
1 1 1 0 0 0	0 0 1 0 1	余数与除数同号，商上1
1 1 0 0 0 0	0 1 0 1 0	左移
+ 0 0 1 0 0 0		$+[-Y]_{补}$
1 1 1 0 0 0	0 1 0 1 1	余数与除数同号，商上1
1 1 0 0 0 0	1 0 1 1 0	左移
+ 0 0 1 0 0 0		$+[-Y]_{补}$
1 1 1 0 0 0	1 0 1 1 1	余数与除数同号，商上1
+ 0 0 1 0 0 0		恢复余数 $+[-Y]_{补}$
0 0 0 0 0 0		

$[X/Y]_{反}=1.0111$ $[X/Y]_{补}=[X/Y]_{反}+0.0001=1.1000$

商的最低位恒置1的方法，则得 $[X/Y]_{补}=1.0111$，其误差为$2^{-n}=2^{-4}$。

表3.6 补码除法规则($[r_i]_{补}$为余数，数值部分共n位

$X_{补}$，$Y_{补}$符号	商符	第一步操作	$r_{补}$,$Y_{补}$符号	上商	下一步操作(共n步)
同号	0	减	同号(够减) 异号(不够减)	1 0	$2\ [r_i]_{补}-Y_{补}$ $2\ [r_i]_{补}+Y_{补}$
异号	1	加	同号(不够减) 异号(够减)	1 0	$2\ [r_i]_{补}-Y_{补}$ $2\ [r_i]_{补}+Y_{补}$

说明:(1) 表中i=0～n-1。

(2) 商一般采用末位恒置“1”的方法，操作简便。如要提高精度，则按上述规则多求一位，再采用以下方法对商进行处理。

两数能除尽：如果除数为正，商不必加2^{-n}，如除数为负，商加2^{-n}；

两数除不尽：如果商为正，商不必加2^{-n}，如商为负，商加2^{-n}。

(3)余数的校正原则

若商为正，当余数与被除数异号时，则应将余数加上除数进行修正才能获得正确的余数

若商为负，当余数与被除数异号时，则应将余数减去除数进行修正才能获得正确的余数

例3.42 $[X]_{补}=1.0111$，$[Y]_{补}=1.0011$，

则 $[-Y]_{补}=0.1101$。 $[X/Y]_{补}=0.1011$ $[余数]_{补}=1.1111\times2^{-4}$

被除数（余数）	商	
1 1 0 1 1 1		
+ 0 0 1 1 0 1		两数同号 $+[-Y]_{补}$
0 0 0 1 0 0	0	余数与除数异号，商上0
0 0 1 0 0 0		
+ 1 1 0 0 1 1		$+[Y]_{补}$
1 1 1 0 1 1	0 1	余数与除数同号，商上1
1 1 0 1 1 0		
+ 0 0 1 1 0 1		$+[-Y]_{补}$
0 0 0 0 1 1	0 1 0	余数与除数异号，商上0
0 0 0 1 1 0		
+ 1 1 0 0 1 1		$+[Y]_{补}$
1 1 1 0 0 1	0 1 0 1	余数与除数同号，商上1
1 1 0 0 1 0		
+ 0 0 1 1 0 1		$+[-Y]_{补}$
1 1 1 1 1 1	0 1 0 1 1	余数与除数同号，商上1

3.4.2 提高除法运算速度的方法举例

1. 跳0跳1除法

提高规格化小数绝对值相除速度的算法。可根据余数前几位代码值再次求得几位同为1或0的商。其规则是：

(1) 如果余数R≥0，且R的高K个数位均数0，则本次直接得商1，后跟K-1个0。R左移K位后，减去除数Y，得新余数。

(2) 如果余数R＜0，且R的高K个数位均为1，则本次商为0，后跟K-1个1，R左移K位后，加上除数Y，得新余数。

(3) 不满足(1)和(2)中条件时，按一位除法上商。

例3.43 设X=0.1010000,Y=0.1100011,求X/Y。

解： $[-Y]_{补}$=1.0011101

X/Y=0.1100111

被除数　　商

0.1 0 1 0 0 0 0

+ 1.0 0 1 1 1 0 1　　-Y

1.1 1 0 1 1 0 1　　0 1　　R<0,符合后有2个1，商01

1.0 1 1 0 1 0 0　　0 1 0 0　左移2位

+ 0.1 1 0 0 0 1 1　　+Y

0.0 0 1 0 1 1 1　　0 1 1 0　R>0,符合后有2个0，商10

0.1 0 1 1 1 0 0　　0 1 1 0 0 0　左移2位

+ 1.0 0 1 1 1 0 1　　-Y

1.1 1 1 1 0 0 1　0 1 1 0 0 1 1 1 R<0,符合后有4个1，商0111

(左移4位，+Y 继续求商)

2. 除法运算通过乘法操作来实现

在计算机运行时，执行乘法指令的几率比除法高。某些CPU中设置有专门的乘法器，一般没有专用除法器，在这种情况下，利用乘法来完成除法运算可提高速度。

设X为被乘数，Y为乘数，按下式完成X/Y。

$$\frac{X}{Y}=\frac{X\bullet F_0\bullet F_1\cdots F_r}{Y\bullet F_0\bullet F_1\cdots F_r}$$

式中$F_i(0\le i\le r)$为迭代系数，如果迭代几次后，可以使分母$Y\times F_0\times F_1\times\ldots\times F_r\rightarrow 1$，则分子即为商：$X\cdot F_0\cdot F_1\ldots F_r$

因此，问题是如何找到一组迭代系数，使分母很快趋近于1。

若X和Y为规格化正小数二进制代码，可写成：

$$Y=1-\delta\ (0<\delta\le 1/2)$$

取 $F_0=1+\delta$，第一次迭代结果：$Y_0=Y\cdot F_0=(1-\delta)(1+\delta)=1-\delta^2$

取$F_1=1+\delta^2$，第二次迭代结果：$Y_1=(1-\delta^2)(1+\delta^2)=1-\delta^4$

…

取$F_i=1+\delta^{2^i}$，第i+1次迭代结果:$Y_i=Y_{i-1}\cdot F_i=(1-\delta^{2^i})(1+\delta^{2^i})=1-\delta^{2^{i+1}}$

当i增加时，Y将很快趋近于1，其误差为$\delta^{2^{i+1}}$。

实际上求得F_i的过程很简单，即

$F_i=1+\delta^{2^i}=2-1+\delta^{2^i}=2-(1-\delta^{2^i})=2-Y_{i-1}$

即F_i就是$(-Y_{i-1})$的补码$(0\le i\le r)$。

例3.44 设X=0.1000,Y=0.1011

则δ=1-Y=0.0101，F_0=1+δ=1.0101

$$\frac{X_0}{Y_0}=\frac{X\cdot F_0}{Y\cdot F_0}=\frac{0.1000\cdot 1.0101}{0.1011\cdot 1.0101}=\frac{0.1011}{0.1110}$$

分子分母分别进行乘法运算。

F_1=2-Y_0=2-0.1110=1.0010

$$\frac{X_1}{Y_1}=\frac{X_0\cdot F_1}{Y_0\cdot F_1}=\frac{0.1011\cdot 1.0010}{0.1110\cdot 1.0010}=\frac{0.1100}{0.1111}$$ 分母趋近于1

所以 $\frac{X}{Y}=\frac{X_1}{Y_1}$

X_1=0.1100

3.5 浮点数的运算方法

浮点数的表示形式(以2为底)：**$N=M\cdot 2^E$**

- M为浮点数的尾数，一般为绝对值小于1的规格化二进制小数，用原码或补码形式表示；E为浮点数的阶码，一般是用移码或补码表示的整数。
- 阶码的底除了2以外，还有用8或16表示的，这里先以2为底进行讨论，然后再简介以8或16为底的数的运算。

3.5.1 浮点数的加减法运算

设有两浮点数X，Y实现X±Y运算，其中：

$X=M_X\cdot 2^{E_X}$; $Y=M_Y\cdot 2^{E_Y}$。均为规格化数。执行以下五步完成运算:

(1) “对阶”操作

比较两浮点数阶码的大小，求出其差ΔE，并保留其大值E，$E=\max(E_X,E_Y)$。当ΔE≠0时，将阶码值小的数的尾数右移ΔE位，并将其阶码值加上ΔE，使两数的阶码值相等，这一操作称之为“对阶”。尾数右移时，对原码表示的尾数，符号位不参加移位，尾数数值部分的高位补0；对补码表示的尾数，符号位参加右移，并保持原符号位不变。

(2) 尾数的加/减运算

执行对阶后，两尾数进行加/减运算，得到两数之和/差。

(3) 规格化操作

规格化的目的是使尾数部分的绝对值尽可能以最大值的形式出现。设尾数M的数值部分有n位，规格化数的范围为：$1/2 \le |[M]_{原}| \le 1-2^{-n}$，$1/2 \le [M]_{补} \le 1-2^{-n}$(当M为正)，$1/2 \le |[M]_{补}| \le 1$(当M为负)。

当运算的结果(和/差)不是规格化数时，需将它转变成规格化数。

规格化操作的规则是：

① 如果结果的两个符号位的值不同，表示加/减运算尾数结果溢出，此时将尾数结果右移1位，阶码E+1，称为“向右规格化”，简称“右规”。

② 如果结果的两个符号位的值相同，表示加/减运算尾数结果不溢出。但若最高数值位与符号位相同，此时尾数连续左移，直到最高数值位与符号位的值不同为止；同时从E中减去移位的位数，这称之为“向左规格化”，简称“左规”。

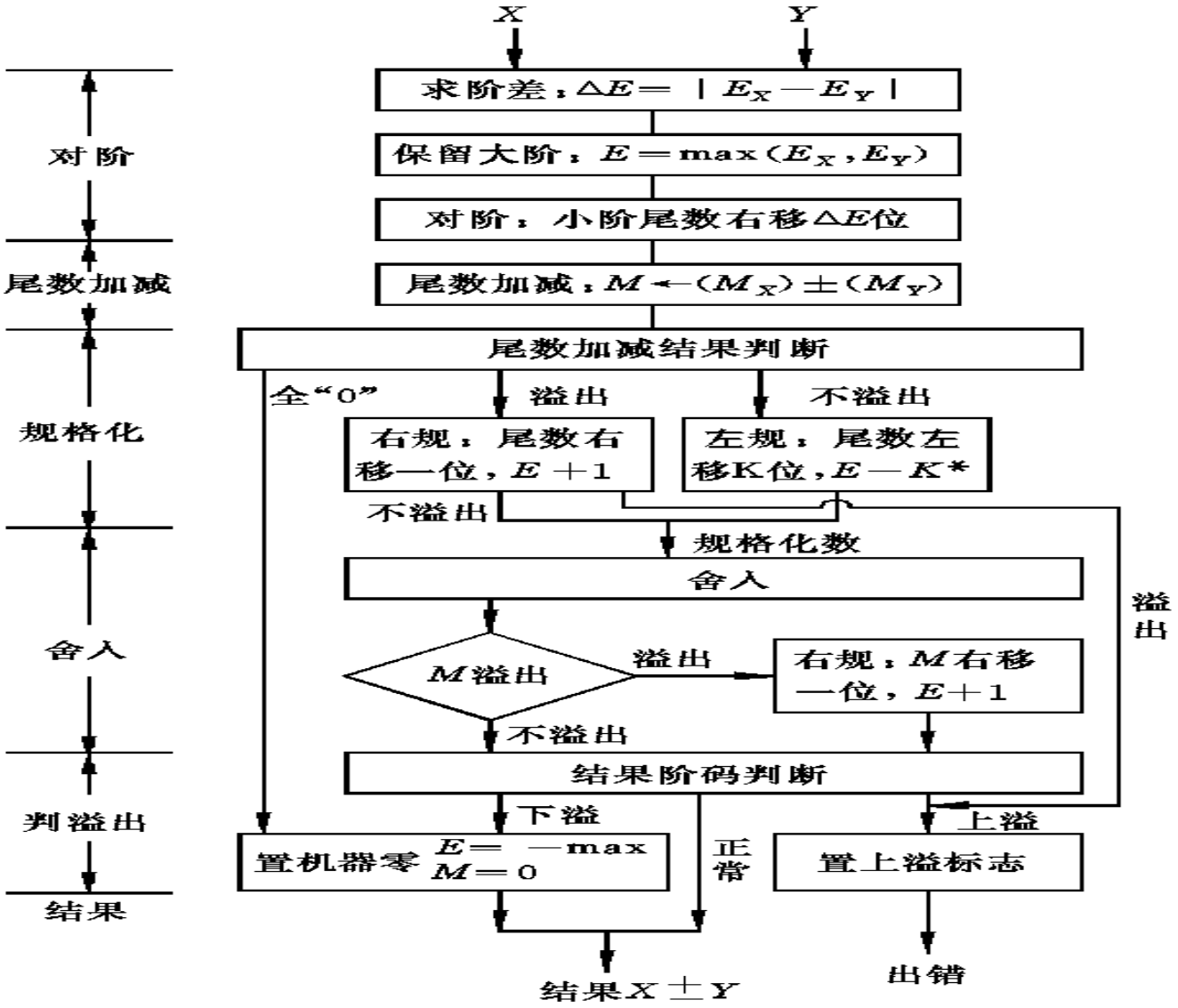
(4) 舍入

在执行右规或对阶时，尾数低位上的数值会移掉，使数值的精度受到影响，常用“0”舍“1”入法。

(5) 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，则置运算结果为机器零，若上溢，则置溢出标志。

图3.8为规格化浮点数加减运算流程。



*如果已为规格化数，则K＝0，不移位。

图3.8 规格化浮点数加减运算流程

例3.45 两浮点数相加，求X+Y。

已知：$X=2^{010}\cdot 0.11011011$, $Y=2^{100}\cdot(-0.10101100)$

解：计算过程：

① 对阶操作

阶差$\Delta E=[E_X]_{补}+[-E_Y]_{补}=00010+11100=11110$

X阶码小，M_X右移2位，保留阶码E=00100。

$[M_X]_{补}=00\ 00\ 110\ 110\ \underline{11}$

下划线上的数是右移出去而保留的附加位。

② 尾数相加

$[M_X]_{补}+[M_Y]_{补}$

$=000011011\underline{11}+1101010100=1110001010\ \underline{11}$

③ 规格化操作

左规，移1位，结果=1100010101 <u>10</u>;阶码-1，E=00011。

④ 舍入

附加位最高位为1，在所得结果的最低位+1，得新结果：

$[M]_{补}=1100010110$,M=-0.11101010。

⑤ 判溢出

阶码符号位为00，故不溢出，最终结果为：

$X+Y=2^{011}\cdot(-0.11101010)$

3.5.2 浮点数的乘除法运算

两浮点数**相乘**，其乘积的阶码为**相乘两数阶码之和**，其尾数应为相乘两数的**尾数之积**。两个浮点数**相除**，商的阶码为被除数的阶码减去除数的阶码得到的**差**，尾数为被除数的尾数除以除数的尾数所得的**商**。参加运算的两个数都为规格化浮点数。乘除运算都可能出现结果不满足规格化要求的问题，因此也必须进行**规格化、舍入和判溢出**等操作。规格化时要修改阶码。

浮点乘法运算步骤

阶码4位(移码)，尾数8位(补码，含1符号位)，阶码以2为底。运算结果仍取8位尾数。

设：$X=2^{-5}\cdot 0.1110011$, $Y=2^{3}\cdot(-0.1110010)$。X，Y为真值，此处阶码用十进制表示，尾数用二进制表示。运算过程中阶码取双符号位。

(1) 求乘积的阶码。乘积的阶码为两数阶码之和。

$[E_X+E_Y]_{移}=[E_X]_{移}+[E_Y]_{补}=00011+00011=00110$

(2) 尾数相乘。用定点数相乘的办法，

$[X\cdot Y]_{补}=$ 1.0011001 1001010 (尾数部分)

高位部分 低位部分

(3) 规格化处理。本例尾数已规格化，不需要再处理。如未规格化，需左规。

(4) 舍入。尾数(乘积)低位部分的最高为1，需要舍入，在乘积高位部分的最低位加1，因此

$[X\cdot Y]_{补}=1.0011010$ (尾数部分)

(5) 判溢出。阶码未溢出，故结果为正确。

X·Y： 0110 10011010

阶码（移码） 尾数（补码）

- $X\cdot Y=2^{-2}\cdot(-0.1100110)$

在求乘积的阶码(即两阶码相加)时，有可能产生上溢或下溢的情况；在进行规格化处理时，有可能产生下溢。

4. 浮点数乘法运算(阶码的底为8或16)

为了用相同位数的阶码表示更大范围的浮点数，在一些计算机中也有选用阶码的底为8或16的。此时浮点数N被表示成

$$N=8^E\cdot M \quad 或 \quad N=16^E\cdot M$$

阶码E和尾数M还都是用二进制表示的，其运算规则，与阶码以2为底基本相同，但关于对阶和规格化操作有新的相应规定。

- 当阶码以8为底时，只要尾数满足**$1/8\le M<1$或$-1\le M<-1/8$**就是规格化数。执行对阶和规格化操作时，**每当阶码的值增或减1，尾数要相应右移或左移三位。**
- 当阶码以16为底时，只要尾数满足**$1/16\le M<1$或$-1\le M<-1/16$**就是规格化数。执行对阶和规格化操作时，**阶码的值增或减1，尾数必须移四位。**
- 判别为规格化数或实现规格化操作，均**应使数值的最高三位(以8为底)或四位(以16为底)中至少有一位与符号位不同。**

5. 浮点数除法运算步骤

与乘法运算类似，也分求商的阶码、尾数相除、规格化、舍入和判溢出5个步骤。

3.6 运算部件

1. 定点运算部件

定点运算部件由**算术逻辑运算部件ALU**、**若干个寄存器**、**移位电路**、**计数器**、**门电路等组成**。ALU部件主要完成加减法算术运算及逻辑运算，其中还应包含有快速进位电路。

2. 浮点运算部件

通常由**阶码运算部件和尾数运算部件**组成，其各自的结构与定点运算部件相似。但阶码部分仅执行加减法运算。其尾数部分则执行加减乘除运算，左规时有时需要左移多位。为加速移位过程，有的机器设置了可移动多位的电路。

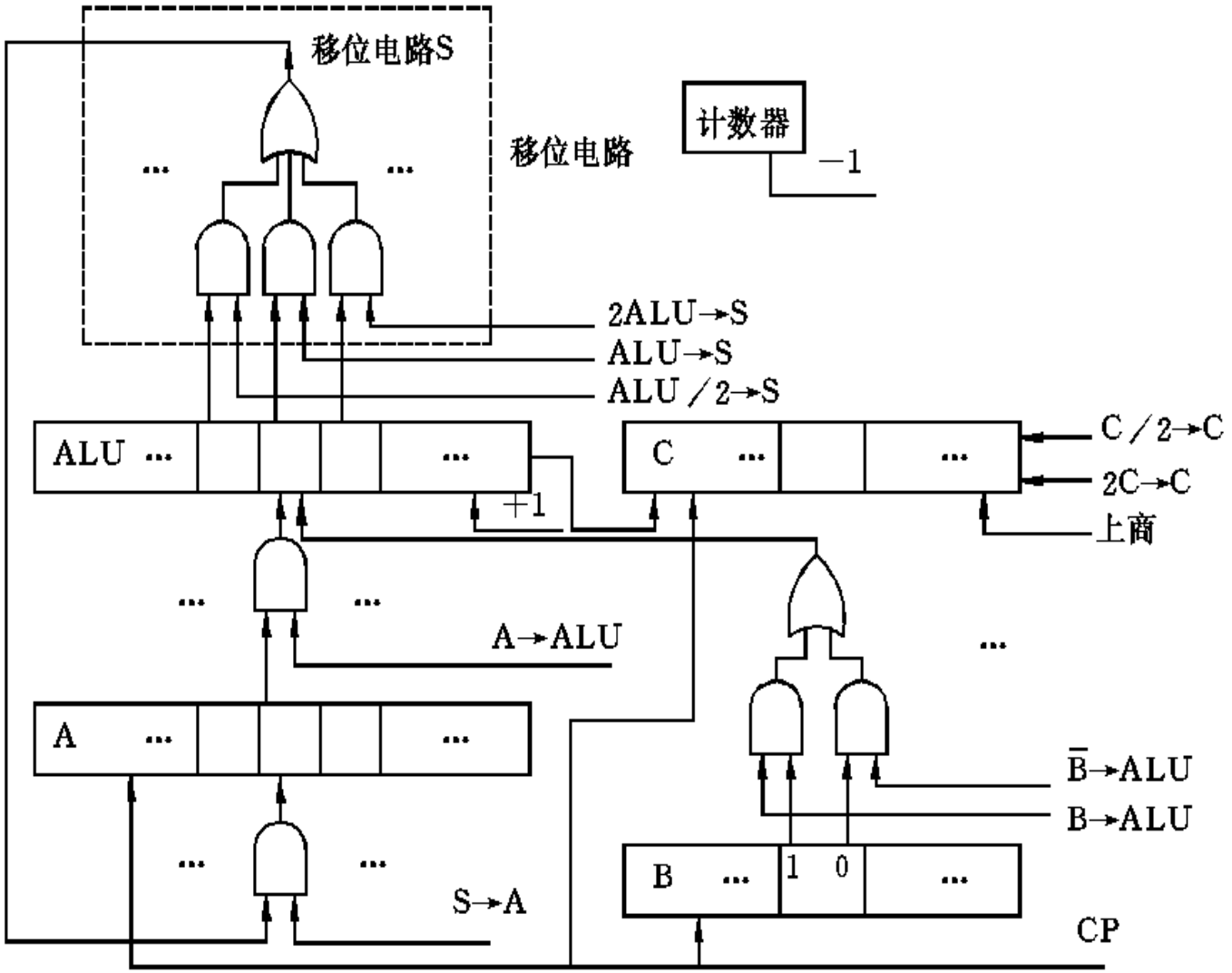


图3.9 定点运算部件框图

表3.7 A，B，C寄存器的作用

运算	A	B	C
加法	被加数 运算结果	加数	无用
减法	被减数 运算结果	减数	无用
乘法	部分积 乘积高位	被乘数	乘数，乘积低位
除法	被除数 余数	除数	商

3.7 数据校验码

计算机系统中的数据，在读写、存取和传送的过程中可能产生错误。为减少和避免这类错误，一方面是精心设计各种电路，提高计算机硬件的可靠性；**另一方面是在数据编码上找出路，即采用某种编码法，通过少量的附加电路，使之能发现某些错误，甚至能确定出错位置，进而实现自动改错的能力。**

- **数据校验码:**一种常用的带有发现某些错误或自动改错能力的数据编码方法。
- **实现原理:**加进一些**冗余码**，使合法数据编码出现某些错误时，就成为**非法编码。**通过检测编码的合法性来达到发现错误的目的。**合理地安排非法编码数量和编码规则，就可以提高发现错误的能力，或自动改正错误的目的。**
- **码距:根据任意两个合法码之间至少有几个二进制位不相同而确定的，仅有一位不同，称其码距为1。**一般来说，合理地增大码距，就能提高发现错误的能力，但码所使用的二进制位数变多，增加了数据存储的容量或数据传送的数量。**在确定与使用数据校验码的时候，通常要考虑在不过多增加硬件开销的情况下，尽可能发现或改正更多的错误。**

3.7.1 奇偶校验码

实现原理，码距由1增加到2。若编码中有一个二进制位的值出错了，由1变成0，或由0变成1，这个码都将成为**非法编码**。

实现的具体方法，通常是为一个字节补充一个二进制位，称为**校验位**，用设置校验位的值为0或1，使字节的8位和该校验位含有1值的个数为奇数或偶数。在使用奇数个1的方案进行校验时，称为奇校验，反之，则称为偶校验。

数据	奇校验	偶校验
00000000	100000000	000000000
01010100	001010100	101010100
011111110	101111110	001111110

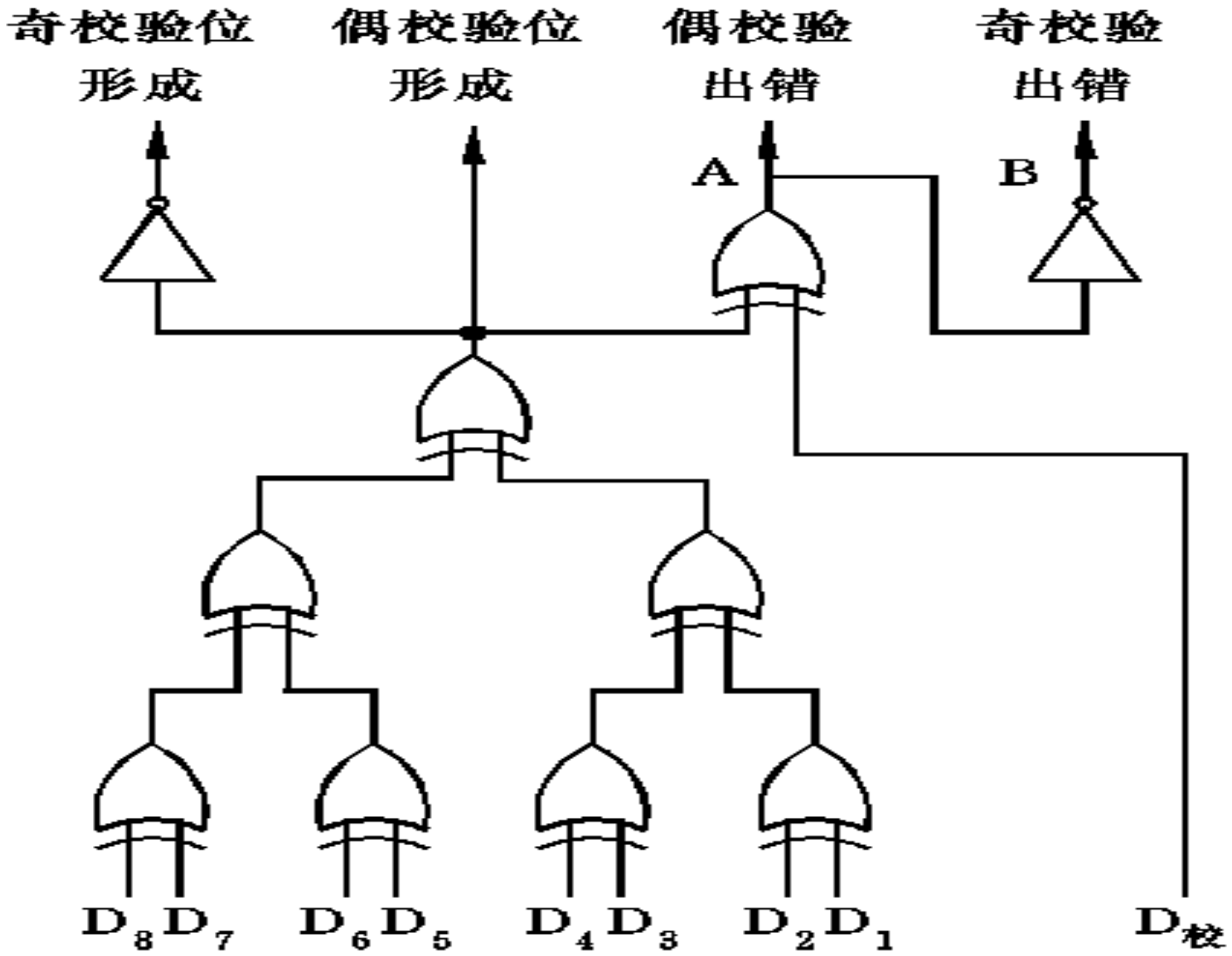


图3.10 奇偶校验位的形成及校验

3.7.2 海明校验码

实现原理:是在数据中加入几个校验位，并把数据的每一个二进制位分配在几个奇偶校验组中。当某一位出错后，就会引起有关的几个校验组的值发生变化，这不但可以发现出错，还能指出是哪一位出错，为自动纠错提供了依据。

假设校验位的个数为r，则它能表示2^r个信息，用其中的一个信息指出“没有错误”，其余的2^r-1个信息指出错误发生在哪一位。然而错误也可能发生在校验位，因此只有$k=2^r-1-r$个信息能用于纠正被传送数据的位数，也就是说要满足关系：

$$2^r \geq k+r+1 \qquad (3.18)$$

如要能检测与自动校正一位错，并发现两位错，此时校验位的位数r和数据位的位数k应满足下述关系：

$$2^{r-1} \geq k+r \qquad (3.19)$$

表3.8 数据位k与校验位r的对应关系

k 值	最 小 的 r 值
1-4	4
5-11	5
12-26	6
27-57	7
58-120	8

若海明码的最高位号为m，最低位号为1，即$H_mH_{m-1}\ldots H_2H_1$，则此**海明码的编码规律通常是：**

(1) **校验位与数据位之和为m**，每个校验位**Pi**在海明码中被分在位号**2^{i-1}**的位置，其余各位为数据位，并按从低向高逐位依次排列的关系分配各数据位。

(2) 海明码的每一位码H_i(包括数据位和校验位本身)由多个校验位校验，**其关系是被校验的每一位位号要等于校验它的各校验位的位号之和。**

$H_{13}\ H_{12}\ H_{11}\ H_{10}\ H_9\ H_8\ H_7\ H_6\ H_5\ H_4\ H_3\ H_2\ H_1$

$P_5\ D_8\ D_7\ D_6\ D_5\ P_4\ D_4\ D_3\ D_2\ P_3\ D_1\ P_2\ P_1$

表3.9 出错的海明码位号和校验位位号的关系

海明码位号	数据位/校验位	参与校验的校验位位号	被校验位的海明码位号=校验位位号之和
H_1	P_1	1	1=1
H_2	P_2	2	2=2
H_3	D_1	1，2	3=1+2
H_4	P_3	4	4=4
H_5	D_2	1，4	5=1+4
H_6	D_3	2，4	6=2+4
H_7	D_4	1，2，4	7=1+2+4
H_8	P_4	8	8=8
H_9	D_5	1，8	9=1+8
H_{10}	D_6	2，8	10=2+8
H_{11}	D_7	1，2，8	11=1+2+8
H_{12}	D_8	4，8	12=4+8
H_{13}	P_5	13	13=13

P值的偶校验结果：

$P_1=D_1\oplus D_2\oplus D_4\oplus D_5\oplus D_7$

$P_2=D_1\oplus D_3\oplus D_4\oplus D_6\oplus D_7$

$P_3=D_2\oplus D_3\oplus D_4\oplus D_8$

$P_4=D_5\oplus D_6\oplus D_7\oplus D_8$

要分清两位出错还是一位出错，增加一个总校验位：

$P_5=D_1\oplus D_2\oplus D_3\oplus D_4\oplus D_5\oplus D_6\oplus D_7\oplus D_8\oplus P_1\oplus P_2\oplus P_3\oplus P_4$

按如下关系得到的海明码实现偶校验

$S_1=P_1\oplus D_1\oplus D_2\oplus D_4\oplus D_5\oplus D_7$

$S_2=P_2\oplus D_1\oplus D_3\oplus D_4\oplus D_6\oplus D_7$

$S_3=P_3\oplus D_2\oplus D_3\oplus D_4\oplus D_8$

$S_4=P_4\oplus D_5\oplus D_6\oplus D_7\oplus D_8$

$S_5=D_1\oplus D_2\oplus D_3\oplus D_4\oplus D_5\oplus D_6\oplus D_7\oplus D_8\oplus P_1\oplus P_2\oplus P_3\oplus P_4\oplus P_5$

S_1-S_5反映13位海明码的出错情况。任何偶数位出错，S_5一定为0

H=12，数据位k=8，校验位r=4的海明校验线路，记作(12，8)分组码。

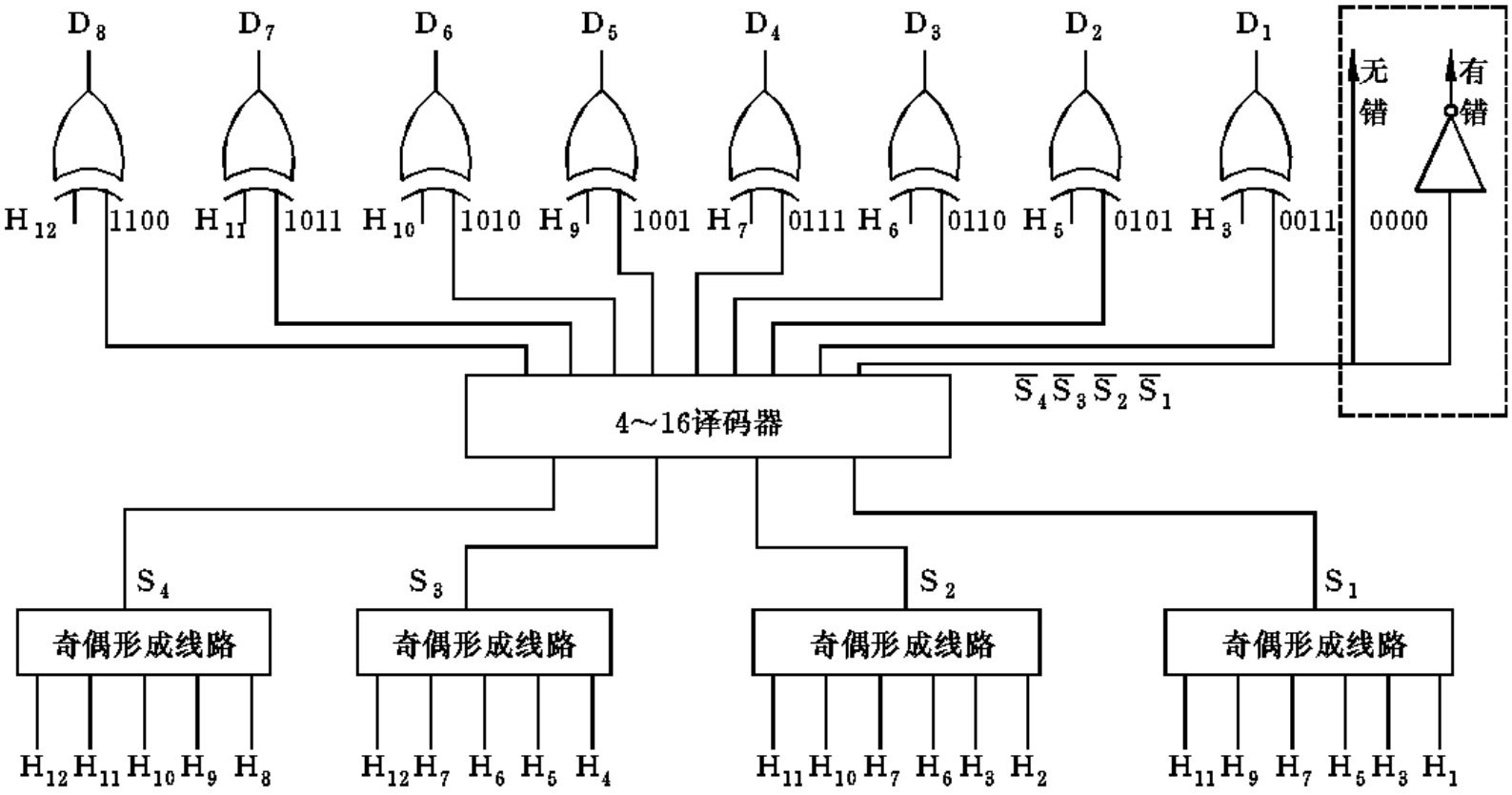


图3.11 (12，8)分组码海明校验线路

假如要进一步判别是1位错还是2位错，则再增加一个校验位。

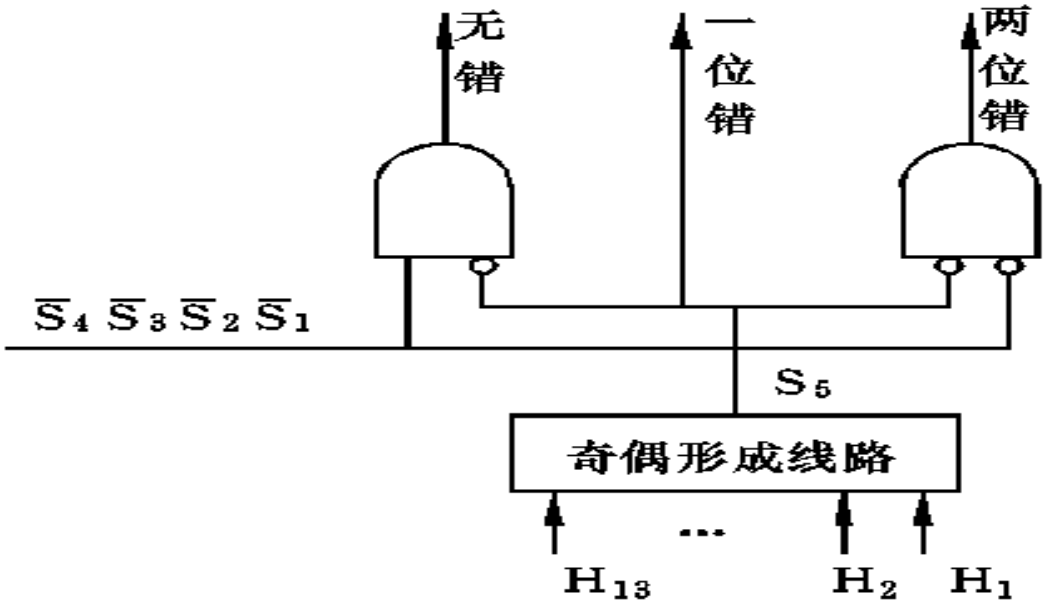


图3.12 判1位/2位错的附加线路

例 求信息码01101110的海明校验码，画出能指示和纠正1位出错位的海明校验逻辑图

例 求信息码01101110的海明校验码，画出能指示和纠正1位出错位的海明校验逻辑图

（1）求01101110的海明校验码

①确定海明校验位的位数

设R为校验位的位数，则整个码字的位数满足：

$N=R+K\leq 2^R-1$

$R=4$

②确定校验位的位置.即2^0、2^1、2^2、2^3作为校验位，记作P1、P2、P3、P4,余下的为有效信息位。

1	2	3	4	5	6	7	8	9	10	11	12
P1	P2	D7	P3	D6	D5	D4	P4	D3	D2	D1	D0

③分组：有4个校验位，将12位分4组

1	2	3	4	5	6	7	8	9	10	11	12				
P1	P2	D7	P3	D6	D5	D4	P4	D3	D2	D1	D0				
	0		1	1	0		1	1	1	0					
第一组（P1）	√		√			√		√		√					
第二组（P2）		√	√				√	√			√	√			
第三组（P3）				√	√	√	√					√			
第四组（P4）								√	√	√	√	√			

④校验位的形成

P1=D7⊕D6⊕D4⊕D3⊕D1=0⊕1⊕0⊕1⊕1=1

P2=D7⊕D5⊕D4⊕D2⊕D1=0⊕1⊕0⊕1⊕1=1

P3=D6⊕D5⊕D4⊕D0=1⊕1⊕0⊕0=0

P4=D3⊕D2⊕D1⊕D0=1⊕1⊕1⊕0=1

信息码01101110的海明校验码为110011011110

(2）校验原理

分别求G1、G2、G3、G4

G1=P1⊕D7⊕D6⊕D4⊕D3⊕D1

G2=P2⊕D7⊕D5⊕D4⊕D2⊕D1

G3=P3⊕D6⊕D5⊕D4⊕D0

G4=P4⊕D3⊕D2⊕D1⊕D0

G4G3G2G1=0000时，接收的数据无误，否则G4G3G2G1的二进制编码为出错位号。

D0出错 G4G3G2G1=1100

G1=1⊕0⊕1⊕0⊕1⊕1=0

G2=1⊕0⊕1⊕0⊕1⊕1=0

G3=0⊕1⊕1⊕0⊕1=1

G4=1⊕1⊕1⊕1⊕1=1

D1出错 G4G3G2G1=1011

G1=1⊕0⊕1⊕0⊕1⊕0=1

G2=1⊕0⊕1⊕0⊕1⊕0=1

G3=0⊕1⊕1⊕0⊕0=0

G4=1⊕1⊕1⊕0⊕0=1

D2，D3，D4，D5，D6，D7，D8出错 1010，1001，0111，0110，0101，0011

P1出错 G4G3G2G1=1011

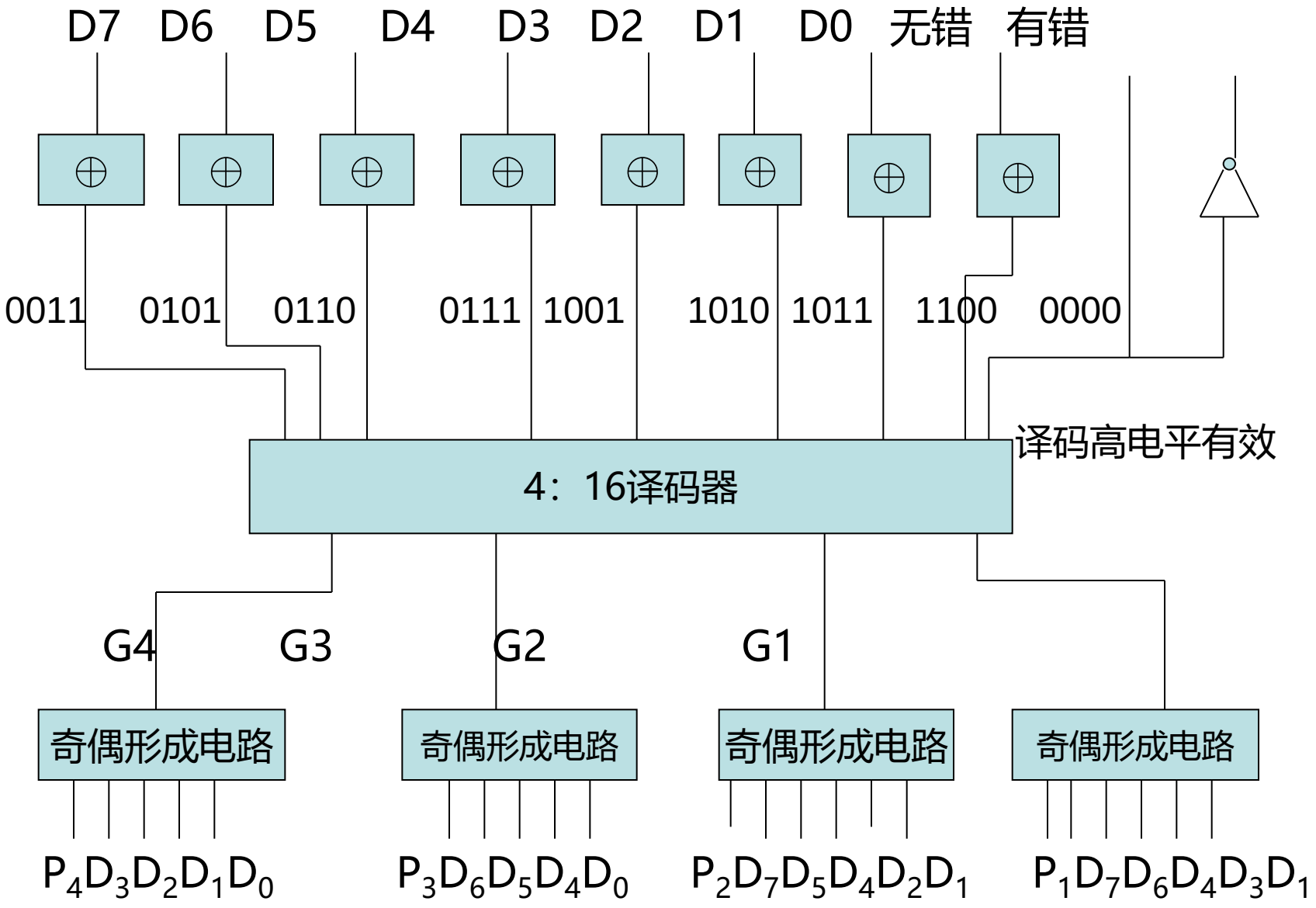
G1=0⊕0⊕1⊕0⊕1⊕1=1

G2=1⊕0⊕1⊕0⊕1⊕0=0

G3=0⊕1⊕1⊕0⊕0=0

G4=1⊕1⊕1⊕1⊕0=0

P2，P3，P4 0010，0100，1000


D7
D6
D5
D4
D3
D2
D1
D0
无错
有错
0011
0101
0110
0111
1001
1010
1011
1100
0000
译码高电平有效
4：16译码器
G4
G3
G2
G1
奇偶形成电路
奇偶形成电路
奇偶形成电路
奇偶形成电路
$P_4D_3D_2D_1D_0$
$P_3D_6D_5D_4D_0$
$P_2D_7D_5D_4D_2D_1$
$P_1D_7D_6D_4D_3D_1$

3.7.3 循环冗余校验(CRC)码

二进制信息位流沿一条线逐位在部件之间或计算机之间传送称为串行传送。CRC(cyclic redundancy check)码可以发现并纠正信息存储或传送过程中连续出现的多位错误，

CRC码一般是指k位信息码之后拼接r位校验码。应用CRC码的关键是如何从k位信息位简便地得到r位校验位(编码)，以及如何从k+r位信息码判断是否出错。

1. CRC码的编码方法

模2运算是指以按位模2相加为基础的四则运算，运算时不考虑进位和借位。

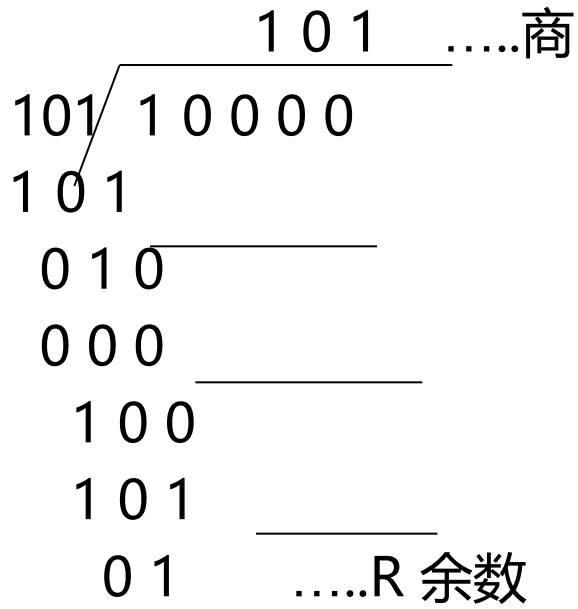
(1) 模2加减：即按位加，可用异或逻辑实现。模2加与模2减的结果相同，即0±0=0，0±1=1，1±0=1，1±1=0。两个相同的数据的模2和为0。

(2) 模2乘——按模2加求部分积之和。

例3.48

$$
\begin{array}{r}
1\,0\,1\,0 \\
\times)\quad 1\,0\,1 \\
\hline
1\,0\,1\,0 \\
0\,0\,0\,0 \\
1\,0\,1\,0 \\
\hline
1\,0\,0\,0\,1\,0
\end{array}
$$

(3) 模2除——按模2减求部分余数。每求一位商应使部分余数减少一位。上商的原则是：当部分余数的首位为1时，商取1；当部分余数的首位为0时，商取0。当部分的余数的位数小于除数的位数时，该余数即为最后余数。


1 0 1 .....商
101 1 0 0 0 0
1 0 1
0 1 0
0 0 0
1 0 0
1 0 1
0 1 .....R 余数

CRC码的编码方法：

将待编码的k位有效信息位组表达为多项式M(x)：

$$M(x)=C_{k-1}x^{k-1}+C_{k-2}x^{k-2}+\ldots+C_ix^i+\ldots+C_1x+C_0$$

式中C_i为0或1。

若将信息位组左移r位，则可表示为多项式$M(x)\cdot x^r$。可以空出r位，以便拼接r位校验位 。

CRC码是用多项式$M(x)\cdot x^r$除以称为生成多项式G(x)(产生校验码的多项式)所得余数作为校验位的。为了得到r位余数(校验位)，G(x)必须是r+1位。

设所得余数表达为R(x)，商为Q(x)。将余数拼接在信息位组左移r位空出的r位上，就构成这个有效信息的CRC码。这个CRC码可用多项式表达为：

$$M(x)\cdot x^r+R(x)=[Q(x)\cdot G(x)+R(x)]+R(x)$$

$$=[Q(x)\cdot G(x)]+[R(x)+R(x)]$$

$$=Q(x)\cdot G(x)$$

因此所得CRC码可被G(x)表示的数码除尽

例3.49 对4位有效信息(1100)求循环校验编码，选择生成多项式(1011)。

$M(x)=x^3+x^2=1100$ (k=4)

$M(x)\cdot x^3=x^6+x^5=1100000$ (左移r=3位)

$G(x)=x^3+x+1=1011$ (r+1=4位)

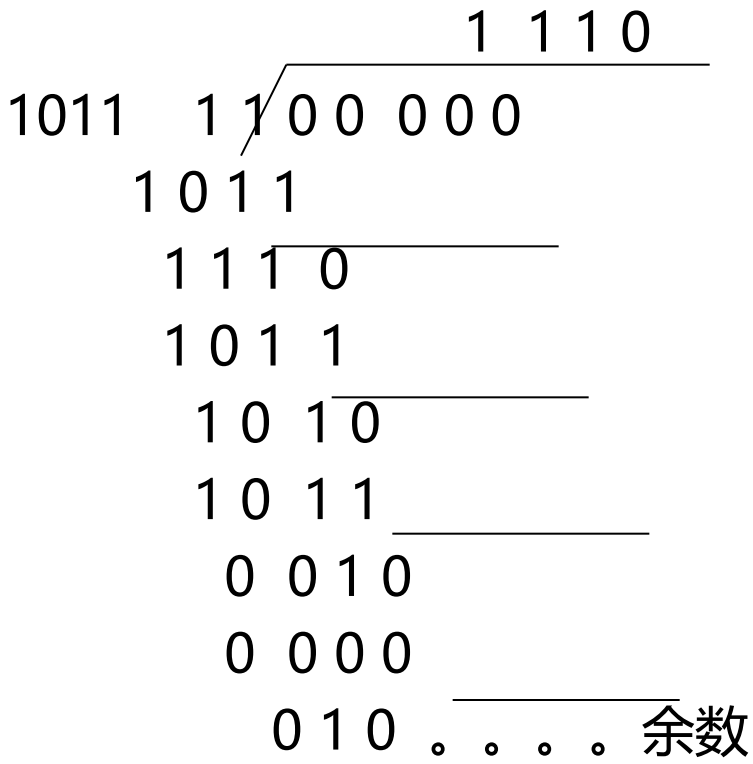
$M(x)\cdot x^3/G(x)=1100000/1011=1110+010/1011$(模2除)

$M(x)\cdot x^3+R(x)=1100000+010=1100010$ (模2加)

将编好的循环校验码称为(7，4)码，即n=7,k=4。

2. CRC的译码与纠错

将收到的循环校验码用约定的生成多项式G(x)去除，如果码字无误则余数应为0，如有某一位出错，则余数不为0，不同位数出错余数不同。**更换不同的待测码字可以证明：余数与出错位的对应关系是不变的，只与码制和生成多项式有关。**。



CRC校验码为1100010

表3.10 1100(7，4)循环码的出错模式(生成多项式G(x)=1011)

	A_1	A_2	A_3	A_4	A_5	A_6	A_7	余数	出错位
正 确	1	1	0	0	0	1	0	0 0 0	无
错 误	1	1	0	0	0	1	**1**	0 0 1	7
	1	1	0	0	0	**0**	0	0 1 0	6
	1	1	0	0	**1**	1	0	1 0 0	5
	1	1	0	**1**	0	1	0	0 1 1	4
	1	1	**1**	0	0	1	0	1 1 0	3
	1	**0**	0	0	0	1	0	1 1 1	2
	0	1	0	0	0	1	0	1 0 1	1

1011 ⟌ **0 1 0** ……**第一个余数**

1 0 0 ……**第二个余数**

1 0 0 0

1 0 1 1

0 1 1 ……**第三个余数**

1 1 0 ……**第四个余数**

1 1 0 0

1 0 1 1

1 1 1 ……**第五个余数**

1 1 1 0

1 0 1 1

1 0 1 ……**第六个余数**

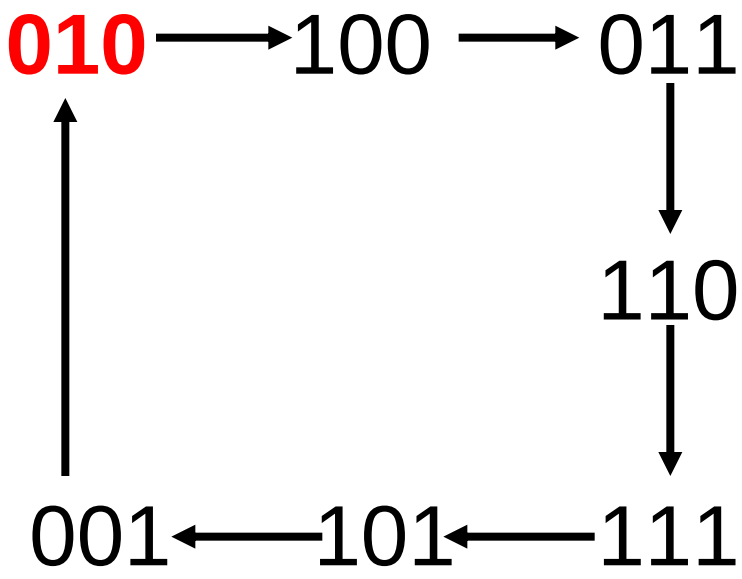
1 0 1 0

1 0 1 1

0 0 1 ……**第七个余数**

0 1 0 ……**第一个余数**

1 0 0 ……**第二个余数**


010
100
011
110
001
101
111

例3.50 对4位有效信息(1010)求循环校验编码，选择生成多项式(1011)

```
          1 0 0 1
      ┌──────────
1011  1 0 1 0 0 0 0
      1 0 1 1
      ─────────
        0 0 1 0
        0 0 0 0
        ───────
          0 1 0 0
          0 0 0 0
          ───────
            1 0 0 0
            1 0 1 1
            ───────
              0 1 1
```

CRC校验码为1010011

表3.11 1010(7，4)循环码的出错模式(生成多项式G(x)=1011)

	A_1	A_2	A_3	A_4	A_5	A_6	A_7	余数	出错位
正确	1	0	1	0	0	1	1	0 0 0	无
错误	1	0	1	0	0	1	**0**	0 0 1	7
	1	0	1	0	0	**0**	1	0 1 0	6
	1	0	1	0	**1**	1	1	1 0 0	5
	1	0	1	**1**	0	1	1	0 1 1	4
	1	0	0	0	0	1	1	1 1 0	3
	1	**1**	1	0	0	1	1	1 1 1	2
	0	0	1	0	0	1	1	1 0 1	1

1011) 0 1 1 ……第一个余数

1 1 0 ……第二个余数

1 1 0 0

1 0 1 1

1 1 1 ……第三个余数

1 1 1 0

1 0 1 1

1 0 1 ……第四个余数

1 0 1 0

1 0 1 1

0 0 1 ……第五个余数

0 1 0 ……第六个余数

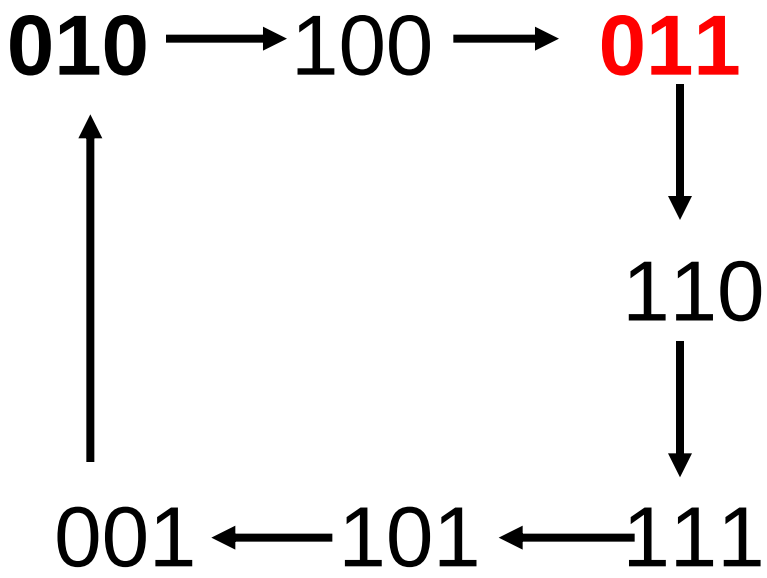
1 0 0 ……第七个余数

1 0 0 0

1 0 1 1

0 1 1 ……第一个余数

1 1 0 ……第二个余数


010
100
011
110
111
101
001

3. 关于生成多项式

并不是任何一个(r+1)位多项式都可以作为生成多项式的。从检错及纠错的要求出发，生成多项式应能满足下列要求：

(1) 任何一位发生错误都应使余数不为0。

(2) 不同位发生错误应当使余数不同。

(3) 对余数继续作模2除，应使余数循环。

将这些要求反映为数学关系是比较复杂的，对一个(n,k)码来说，可将(x^n-1)分解为若干质因子(注意是模2运算)，根据编码所要求的码距选取其中的因式或若干因式的乘积作为生成多项式。

例3.50 $x^7-1=(x+1)(x^3+x+1)(x^3+x^2+1)$ (模2运算)

选择G(x)=x+1=11，可构成(7，6)码，**只能判一位错。**

选择$G(x)=x^3+x+1=1011$，或$G(x)=x^3+x^2+1=1101$，可构成(7，4)码，**能判两位错或纠一位错。**

选择$G(x)=(x+1)(x^3+x+1)=11101$，可构成(7，3)码，**能判两位错并纠正一位错。**

表3.11 生成多项式

n	k	码距d	G(x)多项式	G(x)二进制码
7	4	3	$G_1(x)=(x^3+x+1)$ 或(x^3+x^2+1)	1011 1101
	3	4	$G_2(x)=(x^3+x+1)(x+1)$ 或$(x^3+x^2+1)(x+1)$	11101 10111
15	11 7	3 5	(x^4+x+1) $(x^4+x+1)(x^4+x^3+x^2+x+1)$	10011 111010001
31	26 21	3 5	(x^5+x^2+1) $(x^5+x^2+1)(x^5+x^4+x^3+x^2+1)$	100101 11101101001
63	57 51	3 5	(x^4+x+1) $(x^4+x+1)(x^6+x^4+x+1)$	1000011 1010000110101
1041	1024		$(x^{16}+x^{15}+x^2+1)$	1100000000000101