

## 第二章 运算方法与实现电路

# 第二章 运算方法与实现电路

---

- 2.1 数据的表示与数制之间的转换
- 2.2 机器数的编码表示
- 2.3 其他编码
- 2.4 定点数与浮点数

本章主要讨论数据在机器内部的表示方法、运算方法、算术逻辑部件构成、数据校验码等。

重点：数据表示，运算，ALU构成

难点：浮点数表示，二进制乘/除法运算，进位链传递。

思考问题：1、数据在计算机中如何存储？数学中的整数/实数怎么样存储？2、数据运算在计算机中如何实现？

# 数值型数据的表示

---

一般的，若有 $m+n$ 位的 $r$ 进制无符号数

$$a_{m-1}a_{m-2}\dots a_2a_1a_0.a_{-1}a_{-2}\dots a_{-n},$$

则它的值为:

$$a_{m-1}\times r^{m-1} + a_{m-2}\times r^{m-2} + \dots + a_1\times r^1 + a_0\times r^0 + a_{-1}\times r^{-1} + \dots + a_{-n}\times r^{-n}$$

例如:

$$\begin{aligned} 6543.21 &= 6\times 10^3 + 5\times 10^2 + 4\times 10^1 + 3\times 10^0 \\ &\quad + 2\times 10^{-1} + 1\times 10^{-2} \end{aligned}$$



# 不同进制数的转换

---

- 二、八、十六进制数之间的转换
  - 八、十六进制数都是由二进制数演变而来的，由3位二进制数对应1位八进制数，由4位二进制数对应1位十六进制数。对于有整数和小数部分的数来说，以小数点为界限，对小数点前后的两部分分别进行处理，不足的部分在两侧用0补上，对于整数部分0补在最左侧，对于小数部分0补在最右侧，这样数值不会发生变化。
  - 将八、十六进制转换成二进制时，只要把每一位对应写成3位、4位二进制即可。

# 例题

---

$$(1) \quad (\underline{10} \ \underline{0101} . \underline{1001})_2 \\ = (\underline{0010} \ \underline{0101} . \underline{1001})_2 = (25.9)_{16}$$

$$(2) \quad (12.5)_8 = (\underline{001} \ \underline{010} . \underline{101})_2 \\ = (1010.101)_2$$

$$(3) \quad (2A.3)_{16} = (\underline{0010} \ \underline{1010} . \underline{0011})_2 \\ = (101010.011)_2$$



# 十进制到二进制转换

---

- 进行转换时，通常是将数的整数部分与小数部分分别转换，然后再合并。
- 整数部分一般采用除2取余法，规则如下：将十进制整数除以2，所得的余数(0或1)即为对应二进制数最低位的值。然后对上次所得到的商再除以2，所得到的余数即为二进制数次低位的值，依次进行下去，直到商等于0为止，最后得到的余数是二进制数的最高位的值。
- 小数部分一般采用乘2取整法，规则同上。

# 例题

请将十进制数205.345转换成二进制数。

余数			
2	205		
2	102	.... 1	
2	51	.... 0	逆
2	25	.... 1	
2	12	.... 1	排
2	6	.... 0	
2	3	.... 0	列
2	1	.... 1	
	0	.... 1	



整数		
0.345		
×	2	
	0.690	.... 0
×	2	
	1.380	.... 1
×	2	
	0.760	.... 0
×	2	
	1.520	.... 1
×	2	
	1.040	.... 1



因此:  $205.345 = 11001101.01011_2$



# BCD码—有权码

- BCD码（Binary-Coded Decimal）亦称**二进制十进数或二-十进制代码**。用4位二进制数来表示1位十进制数中的0~9这10个数码。  
是一种**二进制的数字编码形式**，用二进制编码的十进制代码。  
BCD码这种编码形式利用了四个位元来储存一个十进制的数码，使二进制和十进制之间的转换得以快捷的进行。

用BCD码求 $57+14=?$

首先将两数用BCD码表示为：

0101 0111+0001 0100其相加结

果如右。

**加法需要调整**：当本位运算结

果超过1001或者产生进位时，

需要做加6调整

$$\begin{array}{r} \begin{array}{cccc} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ \hline 0 & 1 & 1 & 0 \end{array} & \begin{array}{cccc} 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ \hline 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{array} \\ + & + \\ \hline \begin{array}{cccc} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{array} & \end{array}$$



# BCD码—无权码

无权码是指表示一个十进制数位的二进制码的每一位没有确定的权。

$$(36)_{10} + (27)_{10} = (63)_{10}$$

采用余3码进行运算的过程为右所示。

**相加结果需要调整：**当两个余3码相加不产生进位时，应从结果减去0011；产生进位时，应将进位信号送高位，本位加0011

	0	1	1	0		1	0	0	1
+	0	1	0	1		1	0	1	0
<hr/>									
	1	1	0	0		0	0	1	1
-	0	0	1	1	+	0	0	1	1
<hr/>									
	1	0	0	1		0	1	1	0

# 第二章 运算方法与实现电路

---

- 2.1 数据的表示与数制之间的转换
- 2.2 机器数的编码表示
- 2.3 其他编码
- 2.4 定点数与浮点数
- 2.5 定点数运算方法
- 2.6 浮点数运算方法
- 2.7 数据校验码

# 无符号数

含义：没有符号的数，所有数位均用以表示数值。

基本常识：机器字长 === CPU中寄存器的位数

举例：



8 位

Min: 00000000 = 0

Max: 11111111 =  $2^8 - 1 = 255$



16 位

Min: 00...0000 = 0

Max: 11...1111 =  $2^{16} - 1 = 65535$



# 有符号数

含义：用一位作为符号位，其余位为数值。通常  $\begin{cases} 0 \text{ 表示正数} \\ 1 \text{ 表示负数} \end{cases}$

## (一) 机器数与真值

1. 真值：真正的数值，即带“+”、“-”号的数。

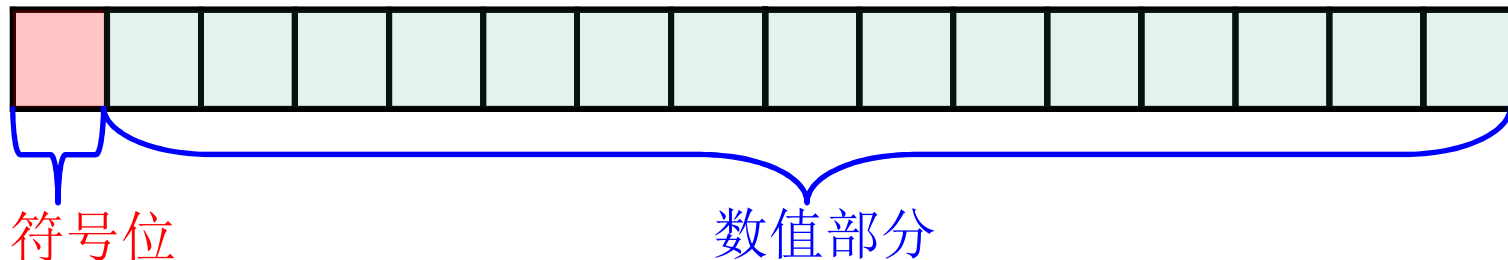
+0.1001	+1001
-0.0101	-1100

精髓：（即日常数学计算中所用到的数字表示形式）

思考： 如何将这些真值连同符号位在计算机中表示呢？

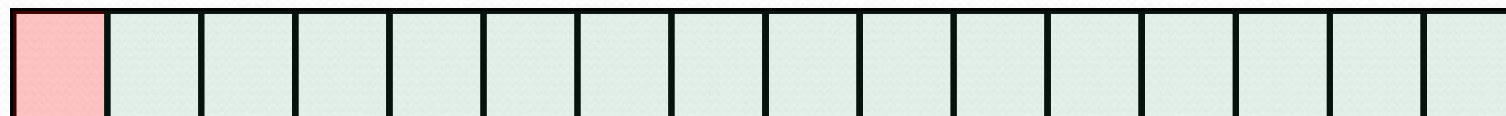
2. 机器数：将符号数字化之后的数。（用于存储在计算机中）

模型：



# 有符号数

模型：



符号

数值部分

一种新的编码

(保证编码后的数可以进行高效的各种运算)

依据编码方案不同，机器数分为：

原码

补码

反码

移码

重难点

# 原码表示法—定点小数

①公式

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

其中

$x$ : 真值

$[x]_{\text{原}}$ : 机器数的原码表示形式

②示例

$$X = +0.1001, \text{ 则 } [x]_{\text{原}} = \underline{0.1001}$$

$$X = -0.1001, \text{ 则 } [x]_{\text{原}} = \underline{1+0.1001} = \underline{1.1001}$$

③规律

{	若 $X = +0.x_1x_2\dots x_n$	$\longrightarrow$	则 $[x]_{\text{原}} = 0.x_1x_2\dots x_n$
	若 $X = -0.x_1x_2\dots x_n$	$\longrightarrow$	则 $[x]_{\text{原}} = 1.x_1x_2\dots x_n$
	若 $x = 0$	$\longrightarrow$	则 $[x]_{\text{原}} = \text{? ? ? ? ? ?}$



# 原码表示法—定点整数

①公式

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

其中  $\begin{cases} x \text{ 为真值} \\ [x]_{\text{原}} \text{ 为机器数的原码表示形式} \\ n \text{ 为整数的位数} \end{cases}$

②示例

$X = +1001$ , 则  $[x]_{\text{原}} = \underline{0,1001}$

$X = -1001$ , 则  $[x]_{\text{原}} = \underline{10000 - (-1001) = 1,1001}$

## ③规律

$\left\{ \begin{array}{ll} \text{若 } X = +x_1x_2\dots x_n & \longrightarrow \text{则 } [x]_{\text{原}} = 0, x_1x_2\dots x_n \\ \text{若 } X = -x_1x_2\dots x_n & \longrightarrow \text{则 } [x]_{\text{原}} = 1, x_1x_2\dots x_n \\ \text{若 } x = 0 & \longrightarrow \text{则 } [x]_{\text{原}} = \text{???} \end{array} \right.$

# 综合举例

原码  $\longleftrightarrow$  真值

例 1 已知  $[x]_{\text{原}} = 1.0011$  求  $x$

解：由定义得

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

例 2 已知  $[x]_{\text{原}} = 1.0101$  求  $x$

解：由定义得

$$x = 1 - [x]_{\text{原}} = 1 - 1.0101 = -0.0101$$

例 3 已知  $[x]_{\text{原}} = 1, 1010$  求  $x$

解：由定义得

$$x = 2^4 - [x]_{\text{原}} = 10000 - 1,1010 = -1010$$

# 原码特点

---

优点：简单、直观、易懂

缺点：加法运算复杂 { 同号则直接相加  
异号，则要进行减法运算。比较绝对值的大小  
然后大的减去小的，最后还要给结果加上恰当的符号。

能否找到一个与负数相等价的正数呢？？

减法运算转换成加法运算



# 补码表示法

- 补数的概念



时钟

欲表示3点钟

方法1

方法2

逆时针旋转 3 格

顺时针旋转 9 格

相互等价

可见： $-3$   $\longleftrightarrow$   $+9$

我们称： $+9$  是  $-3$  以  $12$  为模的补数

记作： $-3 \equiv +9 \pmod{12}$

同理： $-4 \equiv +8 \pmod{12}$

$-5 \equiv +7 \pmod{12}$

减法运算  $\longrightarrow$  加法运算

结论：确定了“模”，一个负数有其相等价的正数。 转化为

# 补码表示法

- 补数的概念

例1:  $X = 8$  ,  $Y = 3$  ,  
求  $X - Y \pmod{10}$

解:

$$\begin{aligned} 8 - 3 &= 8 + 7 \\ &= 15 \pmod{10} \\ &= (\text{丢掉} \blacksquare + 5) \pmod{10} \\ &= 5 \end{aligned}$$

例2:  $X = 1101$  ,  $Y = 1001$  ,  
求  $X - Y \pmod{2^4}$

解:

$$\begin{aligned} 1101 - 1001 &= 1101 + 0111 \\ &= 10100 \pmod{2^4} \\ &= \text{丢掉} \blacksquare + 0100 \pmod{2^4} \\ &= 0100 \end{aligned}$$

结论:

- ① 一个负数加上“模”即得该负数的补数。
- ② 两个互为补数的数 它们绝对值之和即为模数。
- ③ 正数的补数即为其本身。

# 补码表示法—定点小数

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

其中  $\begin{cases} x \text{ 为真值} \\ [x]_{\text{补}} \text{ 为机器数的补码表示形式} \end{cases}$

例1: 已知  $x = 0.1100$  , 则  $[x]_{\text{补}} = \underline{0.1100}$

例2: 已知  $x = -0.1100$  , 则  $[x]_{\text{补}} = \underline{2 - 0.1100 = 1.0100}$

例3: 已知  $[x]_{\text{补}} = 0.1010$  则  $x = \underline{0.1010}$

例4: 已知  $[x]_{\text{补}} = 1.1010$  则  $x = \underline{-0.0110}$

例5:  $[+0.0000]_{\text{补}} = \underline{0.0000}$

例6:  $[-0.0000]_{\text{补}} = \underline{10.0000 - 0.0000 = 10.0000 \pmod{2} = 0.0000}$



# 补码表示法—定点整数

$$[x]_{\text{补}} = \begin{cases} 0, x & 2^n > x \geq 0 \\ 2^{n+1} + x & 0 > x \geq -2^n \pmod{2^{n+1}} \end{cases}$$

其中  $\begin{cases} x \text{ 为真值} \\ n \text{ 为整数的位数} \\ [x]_{\text{补}} \text{ 为机器数的补码形式} \end{cases}$

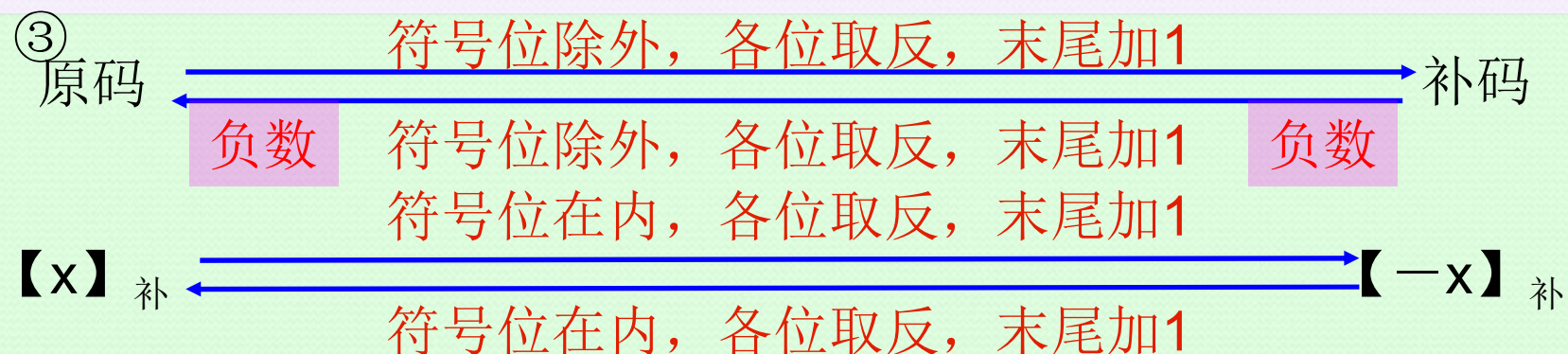
例1: 已知  $x = +1001$ , 则  $[x]_{\text{补}} = \underline{0, 1001}$

例2: 已知  $x = -1001$ , 则  $[x]_{\text{补}} = \begin{array}{r} 2^{4+1} - 1001 \\ = 100000 \\ - 1001 \\ \hline 1, 0111 \end{array}$

# 求补码的快捷方式

① 若  $x = +0.x_1x_2\dots x_n$  , 则  $【x】_{\text{原}} = \underline{0.x_1x_2\dots x_n}$   
 $【x】_{\text{补}} = \underline{0.x_1x_2\dots x_n}$   
 若  $x = -0.x_1x_2\dots x_n$  , 则  $【x】_{\text{原}} = \underline{1.x_1x_2\dots x_n}$   
 $【x】_{\text{补}} = \underline{1.\overline{x_1}\overline{x_2}\dots\overline{x_n} + 0.00\dots 1}$

② 若  $x = +x_1x_2\dots x_n$  , 则  $【x】_{\text{原}} = \underline{0, x_1x_2\dots x_n}$   
 $【x】_{\text{补}} = \underline{0, x_1x_2\dots x_n}$   
 若  $x = -x_1x_2\dots x_n$  , 则  $【x】_{\text{原}} = \underline{1, x_1x_2\dots x_n}$   
 $【x】_{\text{补}} = \underline{1,\overline{x_1}\overline{x_2}\dots\overline{x_n} + 0,00\dots 1}$





## 综合举例

- 例:

设  $x = -1010$  时

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

$$= 11111 + 1 - 1010$$

$$= 11111 + 1$$

$$\quad - 1010$$

$$\quad \boxed{10101} + 1$$

$$= 1,0110$$

$$\text{又 } [x]_{\text{原}} = \boxed{1,1010}$$

相反

结论:

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



# 综合举例

真值	$[x]_{\text{补}}$	$[x]_{\text{原}}$
$x = +70 = 1000110$	0, 1000110	
$x = -70 = -1000110$	1, 0111010	1, 1000110
$x = 0.1110$	0.1110	0.1110
$x = -0.1110$	1.0010	1.1110
$x = +0.0000$ $[+0]_{\text{补}} = [-0]_{\text{补}}$	0.0000	0.0000
$x = -0.0000$	0.0000	1.0000
$x = -1.0000$	1.0000	不能表示

由小数补码定义

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

# 反码表示法一定点小数

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

其中  $\begin{cases} x \text{ 为真值} \\ n \text{ 为整数的位数} \\ [x]_{\text{反}} \text{ 为机器码的反码表示} \end{cases}$

例1: 已知  $x = +0.1101$ , 则  $[x]_{\text{反}} = \underline{0.1101}$

例2: 已知  $x = -0.1001$ , 则  $[x]_{\text{反}} = 2 - 2^{-4} - 0.1001$

$$\begin{array}{r} = 10.0000 \\ - 0.0001 \\ \hline 1.1111 \end{array} \quad \begin{array}{r} 1.1111 \\ - 0.1001 \\ \hline 1.0110 \end{array}$$



# 反码表示法一定点整数

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

其中  $\begin{cases} x \text{ 为真值} \\ n \text{ 为整数的位数} \\ [x]_{\text{反}} \text{ 为机器数的反码表示} \end{cases}$

例1: 已知  $x = +1001$ , 则  $【x】_{\text{反}} = \underline{\quad 0, 1001 \quad}$

例2: 已知  $x = -1001$ , 则  $【x】_{\text{反}} = 2^{4+1} - 1 - 1001$

100000	010111
-1001	-1
<hr/>	<hr/>
010111	1, 0110



# 反码表示规律

定点  
小数

若  $x = +0.x_1x_2\cdots x_n$  则  $【x】_{\text{反}} = \underline{0.x_1x_2\cdots x_n}$

若  $x = -0.x_1x_2\cdots x_n$  则  $【x】_{\text{反}} = \underline{1.\bar{x}_1\bar{x}_2\cdots\bar{x}_n}$

定点  
整数

若  $x = +x_1x_2\cdots x_n$  则  $【x】_{\text{反}} = \underline{0, x_1x_2\cdots x_n}$

若  $x = -x_1x_2\cdots x_n$  则  $【x】_{\text{反}} = \underline{1, \bar{x}_1\bar{x}_2\cdots\bar{x}_n}$

# 原码、补码、反码对照



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	<u>+0</u>	+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



# 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	
	$x = +21$	$+10101$	$0,10101$	 错大
	$x = -21$	$-10101$	$1,01011$	
	$x = +31$	$+11111$	$0,11111$	 错大
	$x = -31$	$-11111$	$1,00001$	

$x + 2^5$	$+10101 + 100000 = 110101$		大 正确
	$-10101 + 100000 = 001011$		
	$+11111 + 100000 = 111111$		大 正确
	$-11111 + 100000 = 000001$		



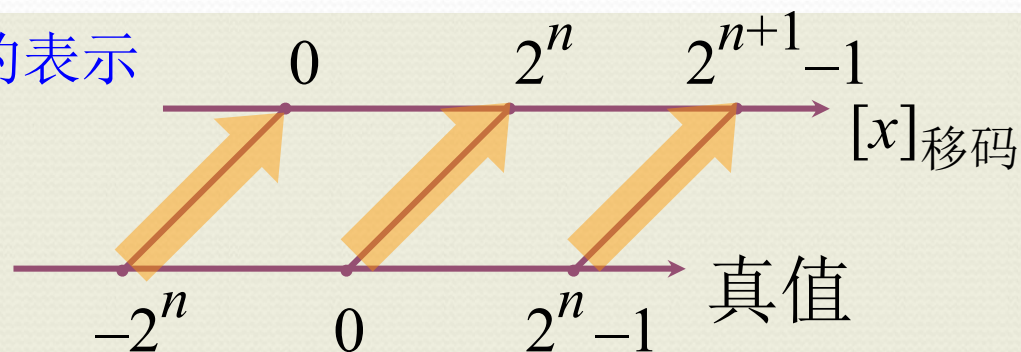
# 移码表示法

## (1) 定义

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

$\left\{ \begin{array}{l} x \text{ 为真值} \\ n \text{ 为 整数的位数} \\ [x]_{\text{移}} \text{ 为机器数的移码表示} \end{array} \right.$

## (2) 移码在数轴上的表示



## (3) 示例

$$x = 10100$$

$$[x]_{\text{移}} = 2^5 + 10100 = 1,10100$$

$$x = -10100$$

$$[x]_{\text{移}} = 2^5 - 10100 = 0,01100$$

用 逗号 将符号位  
和数值位隔开

# 移码和补码比较

---

设  $x = +1100100$  ; 字长为8

$$[x]_{\text{移}} = 2^7 + 1100100 = \textcolor{blue}{1},1100100$$

$$[x]_{\text{补}} = \textcolor{blue}{0},1100100$$

设  $x = -1100100$

$$[x]_{\text{移}} = 2^7 - 1100100 = \textcolor{blue}{0},0011100$$

$$[x]_{\text{补}} = \textcolor{blue}{1},0011100$$

可见: **补码与移码只差一个符号位!**



# 补码、移码对照

真值 $x$ ( $n=5$ )	$[x]_{\text{补}}$	$[x]_{\text{移}}$	$[x]_{\text{移}}$ 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

$[x]_{\text{补}}$

$[x]_{\text{移}}$

$[x]_{\text{移}}$  对应的十进制整数

人小不清楚

人小很清楚

对应



# 移码的特点

已知  $X = 0$ ，则  $【x】_{\text{移}} =$

$$【+0】_{\text{移}} = 2^5 + 0 = 1, 00000$$

$$【-0】_{\text{移}} = 2^5 - 0 = 1, 00000$$

$【+0】_{\text{移}}$

||

$【-0】_{\text{移}}$

当  $n = 5$  时      最小的真值为  $-2^5$        $= -100000$

$$[-100000]_{\text{移}} = 2^5 - 100000 = 000000$$

可见，最小真值的移码为全 0

移码的符号位： $\left\{ \begin{array}{l} \text{“0” 表示负数} \\ \text{“1” 表示正数} \end{array} \right.$

# 第二章 运算方法与实现电路

---

- 2.1 数据的表示与数制之间的转换
- 2.2 机器数的编码表示
- 2.3 其他编码
- 2.4 定点数与浮点数
- 2.5 定点数运算方法
- 2.6 浮点数运算方法
- 2.7 数据校验码



# 其他编码

---

- ASCII码
- 汉字编码
- 国标码=区位码+2020H
- 汉字内码=国标码+8080H
- Unicode(自行查阅网上相关资料)
- BCD码 (8421/2421/余3码/格雷码)
- 十进制数串



ASCII 字符代码表 一

高四位    低四位		ASCII非打印控制字符										ASCII 打印字符												
		0000					0001					0010	0011		0100		0101		0110		0111			
		0					1					2	3		4		5		6		7			
		+进制	字符	ctrl	代码	字符解释	+进制	字符	ctrl	代码	字符解释	+进制	字符	+进制	字符	+进制	字符	+进制	字符	+进制	字符	+进制	字符	ctrl
0000	0	0	BLANK NULL	^@	NUL	空	16	▶	^P	DLE	数据链路转意	32		48	0	64	@	80	P	96	`	112	p	
0001	1	1	☺	^A	SOH	头标开始	17	◀	^Q	DC1	设备控制 1	33	!	49	1	65	A	81	Q	97	a	113	q	
0010	2	2	☹	^B	STX	正文开始	18	↕	^R	DC2	设备控制 2	34	"	50	2	66	B	82	R	98	b	114	r	
0011	3	3	♥	^C	ETX	正文结束	19	!!	^S	DC3	设备控制 3	35	#	51	3	67	C	83	S	99	c	115	s	
0100	4	4	◆	^D	EOT	传输结束	20	¶	^T	DC4	设备控制 4	36	\$	52	4	68	D	84	T	100	d	116	t	
0101	5	5	♣	^E	ENQ	查询	21	♫	^U	NAK	反确认	37	%	53	5	69	E	85	U	101	e	117	u	
0110	6	6	♠	^F	ACK	确认	22	■	^V	SYN	同步空闲	38	&	54	6	70	F	86	V	102	f	118	v	
0111	7	7	●	^G	BEL	震铃	23	↑↓	^W	ETB	传输块结束	39	'	55	7	71	G	87	w	103	g	119	w	
1000	8	8	◼	^H	BS	退格	24	↑	^X	CAN	取消	40	(	56	8	72	H	88	X	104	h	120	x	
1001	9	9	○	^I	TAB	水平制表符	25	↓	^Y	EM	媒体结束	41	)	57	9	73	I	89	Y	105	i	121	y	
1010	A	10	◻	^J	LF	换行/新行	26	→	^Z	SUB	替换	42	*	58	:	74	J	90	Z	106	j	122	z	
1011	B	11	♂	^K	VT	垂直制表符	27	←	^I	ESC	转意	43	+	59	;	75	K	91	[	107	k	123	{	
1100	C	12	♀	^L	FF	换页/新页	28	└	^N	FS	文件分隔符	44	,	60	<	76	L	92	\	108	l	124		
1101	D	13	♪	^M	CR	回车	29	↔	^J	GS	组分隔符	45	-	61	=	77	M	93	]	109	m	125	}	
1110	E	14	🎵	^N	SO	移出	30	▲	^6	RS	记录分隔符	46	.	62	>	78	N	94	^	110	n	126	~	
1111		15	🎵	^O	SI	移入	31	▼	^-	US	单元分隔符	47	/	63	?	79	O	95	_	111	o	127	Δ	Back space

注：表中的ASCII字符可以用：ALT + “小键盘上的数字键”输入。

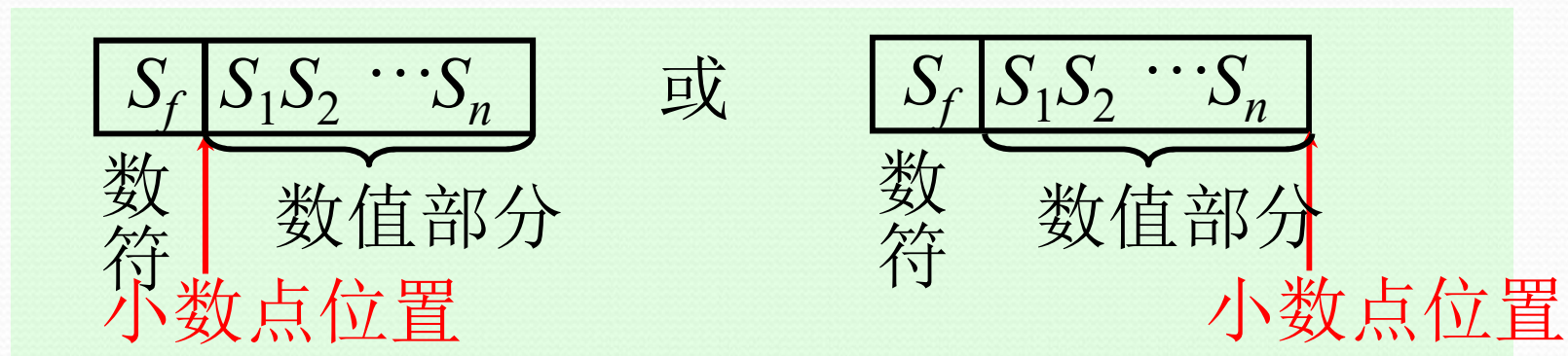
# 第二章 运算方法与实现电路

---

- 2.1 数据的表示与数制之间的转换
- 2.2 机器数的编码表示
- 2.3 其他编码
- 2.4 定点数与浮点数
- 2.5 定点数运算方法
- 2.6 浮点数运算方法
- 2.7 数据校验码



# 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

含义：小数点的位置是浮动的，不是固定的。

$$N = 111.010101011$$

$$N = 456789.1$$

$$N = 11.1010101011 \times 2^1$$

$$N = 45.67891 \times 10^4$$

$$N = 1.11010101011 \times 2^{10}$$

$$N = 4.567891 \times 10^5$$

浮点数的一般形式： $N = S \times r^j$

$S$  尾数

$r$  基数（基值）

$j$  阶码

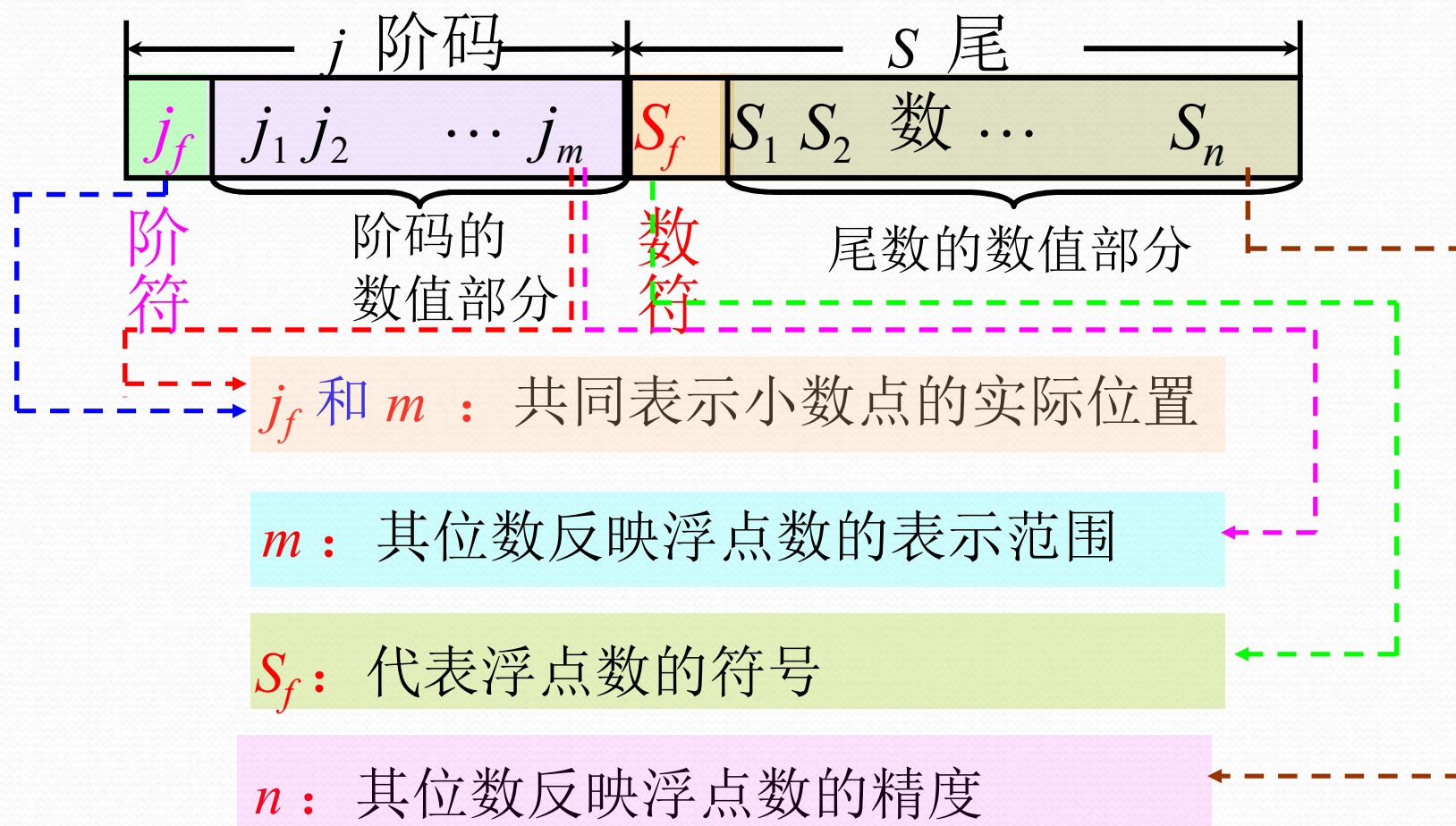
注：

①计算机中  $r$  取 2、4、8、16 等

②计算机中  $S$  小数、可正可负

③计算机中  $j$  整数、可正可负

# 浮点数的表示形式





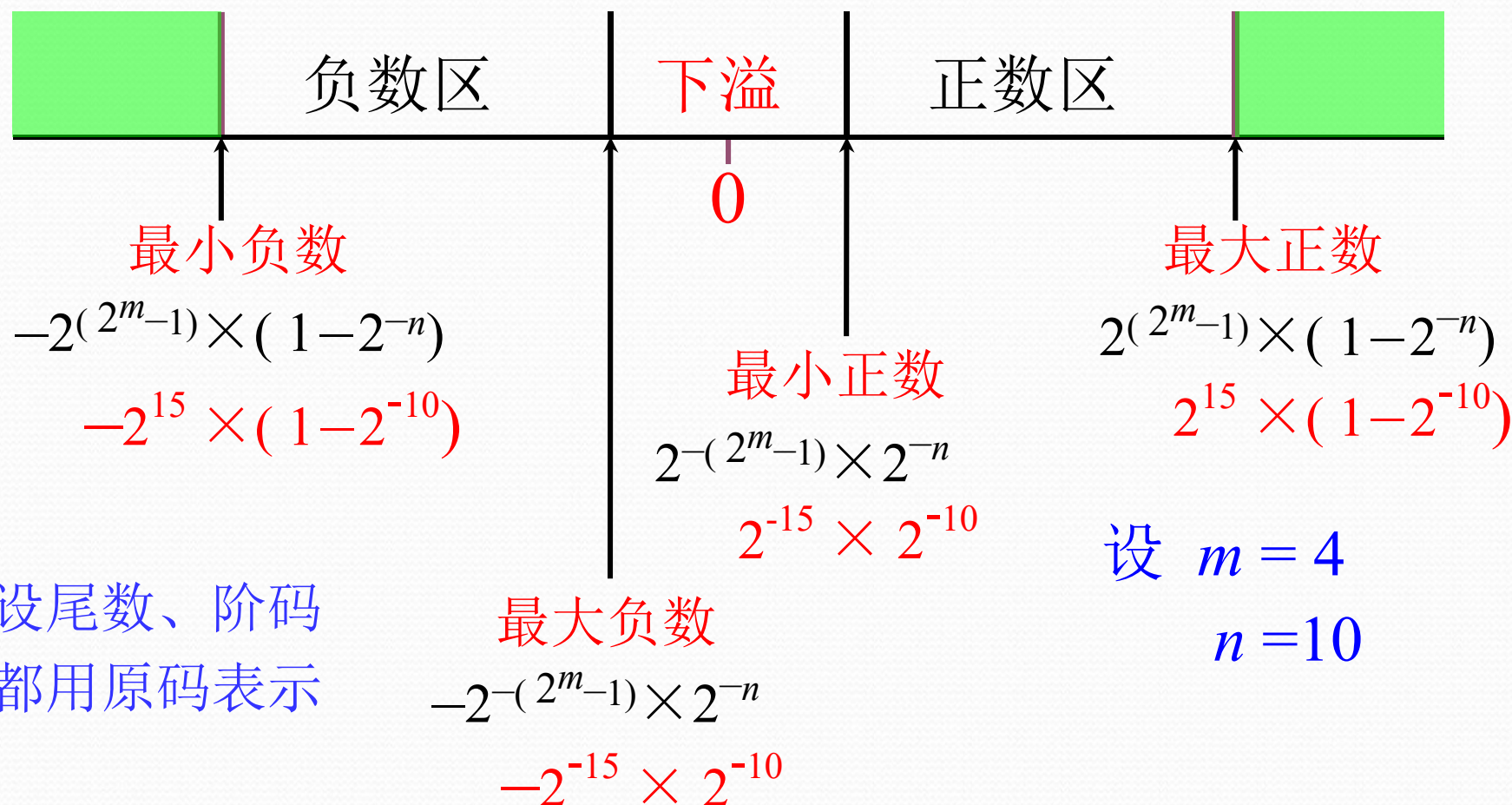
# 浮点数的表示范围

上溢 阶码 > 最大阶码

下溢 阶码 < 最小阶码 按 机器零 处理

上溢

上溢





# 练习

设机器数字长为 24 位，欲表示  $\pm 3$  万范围内的十进制数，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？请给出计算过程

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

$\therefore 15$  为二进制数可反映  $\pm 3$  万之间的十进制数

$$2^{15} \times 0.\underbrace{\times \times \times \dots \times \times \times}_{18\text{位}}$$

$m = 4, 5, 6, \dots$

满足最大精度可取  $m = 4, n = 18$

# 规格化浮点数

**目的:** 为了提高精度，统一将浮点数表示成为  $N = 0.11011 \times 2^{10101}$  这种形式。此时精度最高。

**办法:** 比如， $N = 0.001011 \times 2^{1101}$

左规: 尾数左移 1 位，阶码减 1

规格化形式:  $0.1011 \times 2^{1011}$

比如， $N = 1011.11 \times 2^{1001}$

右规: 尾数右移 1 位，阶码加 1

规格化形式:  $0.101111 \times 2^{1101}$



# 规格化浮点数

## 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      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

## 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



# 表数范围

例如：设尾数、阶码均用原码表示，其中  $m=4$ ,  $n=10$ ,  $r=2$

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  $2^{-1111} \times 0.1\underbrace{0000000000}_{9 \text{ 个 } 0} = 2^{-15} \times 2^{-1} = 2^{-16}$

最大负数  $2^{-1111} \times (-0.1\underbrace{0000000000}_{9 \text{ 个 } 0}) = -2^{-15} \times 2^{-1} = -2^{-16}$

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

# 例题

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：**  $x = +\frac{19}{128}$

二进制形式	$x = 0.0010011$
定点表示	$x = 0.0010011 \text{ } 000$
浮点规格化形式	$x = 0.1001100000 \times 2^{-10}$
定点机中	$[x]_{\text{原}} = [x]_{\text{补}} = [x]_{\text{反}} = 0.0010011000$
浮点机中	$[x]_{\text{原}} = 1, 0010; 0.1001100000$
	$[x]_{\text{补}} = 1, 1110; 0.1001100000$
	$[x]_{\text{反}} = 1, 1101; 0.1001100000$



# 例题

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：  $x = -58$

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
$[x]_{\text{原}} = 1, 0000111010$	$[x]_{\text{原}} = 0, 0110; 1. 1110100000$
$[x]_{\text{补}} = 1, 1111000110$	$[x]_{\text{补}} = 0, 0110; 1. 0001100000$
$[x]_{\text{反}} = 1, 1111000101$	$[x]_{\text{反}} = 0, 0110; 1. 0001011111$
	$[x]_{\text{阶移、尾补}} = 1, 0110; 1. 0001100000$



# 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
- 当浮点数阶码等于或小于它所表示的最小数时，  
不论尾数为何值，按机器零处理

如  $m = 4$        $n = 10$

当阶码和尾数都用补码表示时，机器零为：

$\times, \times \times \times \times; \quad 0.00 \dots 0$

(阶码 = -16)  $1, 0000; \quad \times.\times\times \dots \times$

当阶码用移码，尾数用补码表示时，机器零为

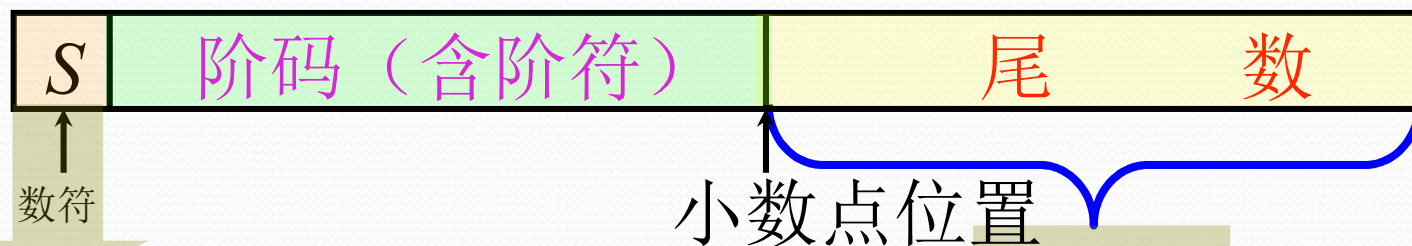
$0, 0000; \quad 0.00 \dots 0$

有利于机器中“判0”电路的实现

# IEEE 754标准

**常识：**采用浮点数表示形式的计算机中，浮点数一般有规定的格式标准。

IEEE 754  
标准



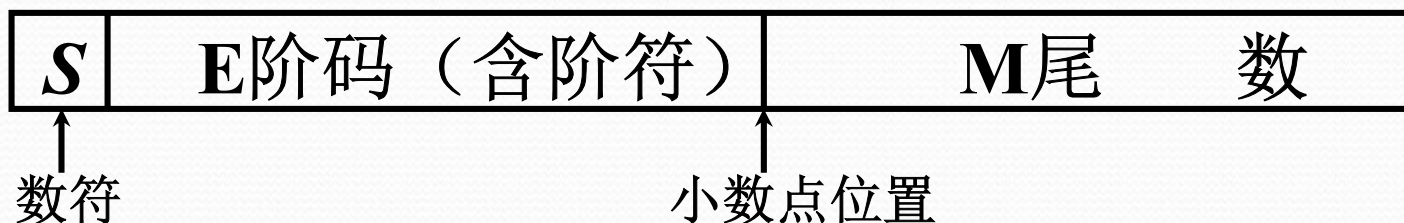
浮点数的正负

阶码用移码表示

尾数采用规格化形式



# IEEE 754标准



**尾数**：规格化表示，原码形式，非“0”的有效位最高位为“1”（隐含）；**阶码**：移码(偏移量不同)

对短实数，表示的数值  $v = (-1)^S \times 1.M \times 2^{E-127}$

	符号位 $S$	阶码	偏移量	尾数	总位数
短实数	1	8	7FH	23	32
长实数	1	11	3FFH	52	64
临时实数	1	15	3FFFH	64	80



# IEEE 754标准

---

例1：将 $(100.25)_{10}$ 转换成短浮点数格式  
解：

- (1) 将十进制转换成二进制数
- (2) 规格化二进制数
- (3) 计算阶码的移码值
- (4) 以IEEE 754标准浮点数形式存储

$$(1) \quad (100.25)_{10} = (1100100.01)_2$$

$$(2) \quad 1100100.01 = 1.10010001 \times 2^{110}$$

$$(3) \quad 1111111 + 110 = 10000101$$

$$(4) \quad 0, 10000101, 100100010000000000000000 = 42C88000H$$

# IEEE 754标准

---

例2：将短浮点数C1C90000H转换成十进制数  
解：

(1) 将机器数分离出数符、阶码、尾数

$$\begin{aligned} \text{C1C90000H} &= 11000001110010010000000000000000 \\ &= 1,10000011,100100100000000000000000 \end{aligned}$$

(2) 计算阶码真值

$$E = 10000011 - 1111111 = 100$$

(3) 以规格化形式写出数据

$$-1.1001001 \times 2^{100}$$

(4) 写出二进制形式：-11001.001

(5) 转换结果：-25.125



# IEEE 754标准

---

对短实数，表示的数值 $v = (-1)^S \times 1.M \times 2^{E-127}$

其中，S代表符号位，E为用移码表示的阶码，M表示尾数的小数部分

阶码真值为了表示一些特殊的数值，E的最小值0和最大值保留。因此E的取值在1~254之间，对应的阶码真值为-126~+127之间

机器0:当E和M为全0时

$\pm\infty$ :当E为全1，M为全0时，根据符号位表示

NaN(非数, Not A Number): E为全1，M非全零

非规格化数:E为全零，M为非零

作业: P.289-291: 6.5,6.6,6.12,6.14,6.16,6.17



# 第二章 运算方法与实现电路

---

- 2.1 数据的表示与数制之间的转换
- 2.2 机器数的编码表示
- 2.3 其他编码
- 2.4 定点数与浮点数
- 2.5 定点数运算方法
- 2.6 浮点数运算方法
- 2.7 数据校验码

# 移位运算

定点运算  
(定点数的运算)

浮点运算  
(浮点数的运算)

$888_{10}$

88800  $\xleftarrow{\text{左移两位}}$

$\xrightarrow{\text{右移两位}}$  8.88

当某个十进制数相对于小数点左移 $n$ 位时，相当于该数乘以 $10^n$ ；右移 $n$ 位时，相当于该数除以 $10^n$ 。

$101101_2$

10110100  $\xleftarrow{\text{左移两位}}$

$\xrightarrow{\text{右移两位}}$  1011.01

当某个二进制数相对于小数点左移 $n$ 位时，相当于该数乘以 $2^n$ ；右移 $n$ 位时，相当于该数除以 $2^n$ 。

**意义：**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

含义：有符号数的移位。有符号数  $\longleftrightarrow$  机器码/机器数

原码  
补码  
反码

机器数模型：

符号位

数值位（数值部分）

给定的机器数

1	0	1	0	1	0	1	0
---	---	---	---	---	---	---	---

左移怎么移呢？

??

右移怎么移呢？

??

# 算术移位规则

总原则：符号位不变

	码 制	添补代码
正数	原码、补码、反码	0
负数	原 码	0
	补 码	左移 添 0
		右移 添 1
	反 码	1



# 例题

设机器数字长为 8 位（含一位符号位），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：

$$A = +26 = +11010$$

$$\text{则 } [A]_{\text{原}} = [A]_{\text{补}} = [A]_{\text{反}} = 0,0011010$$

移位操作	机 器 数	对应的真值
	$[A]_{\text{原}} = [A]_{\text{补}} = [A]_{\text{反}}$	
移位前	0,0011010	+26
←1	0,0110100	+52
←2	0,1101000	+104
→1	0,0001101	+13
→2	0,0000110	+6

正确

影响  
精度

# 例题

设机器数字长为8位（含一位符号位），写出 $A = -43$ 时的原码、补码、反码及三种机器数左、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。

A真值	-101011
【A】 <sub>原</sub>	1, 0101011
【A】 <sub>补</sub>	1, 1010101
【A】 <sub>反</sub>	1, 1010100

原码

移位操作	机器数	对应真值	
移位前	1, 0101011	-43	
左移1位	1, 1010110	-86	正确
左移2位	1, 0101100	-44	错误
右移1位	1, 0010101	-21	
右移2位	1, 0001010	-10	影响精度



## 补码

移位操作	机器数	对应真值	
移位前	1, 1010101	-43	正确
左移1位	1, 0101010	-86	
左移2位	1, 1010100	-44	错误
右移1位	1, 1101010	-22	影响精度
右移2位	1, 1110101	-11	

## 反码

移位操作	机器数	对应真值	
移位前	1, 1010100	-43	正确
左移1位	1, 0101001	-86	
左移2位	1, 1010011	-44	错误
右移1位	1, 1101010	-21	影响精度
右移2位	1, 1110101	-10	

# 逻辑移位规则

(1) 含义：无符号数的移位

(2) 规则：逻辑左移时，高位移出，低位添0；逻辑右移时，低位移出，高位添0

1	0	1	0	1	0	1	0
---	---	---	---	---	---	---	---

0	1	0	1	0	1	0	0
---	---	---	---	---	---	---	---

逻辑左移

逻辑右移

0	1	0	1	0	1	0	1
---	---	---	---	---	---	---	---



# 补码运算规则

---

当补码加减运算的结果不超出机器范围时，有以下的运算规则：

- (1) 参加运算的两个操作数均用补码表示
- (2) 符号位作为数的一部分参加运算
- (3) 减法运算时，将转换成加法运算
- (4) 运算结果为补码
- (5) 符号位相加后产生的进位为模值，自然丢掉

# 补码加法公式

(1) 含义:

整数

$$[A]_{\text{补}} + [B]_{\text{补}} = [A+B]_{\text{补}} \pmod{2^{n+1}}$$

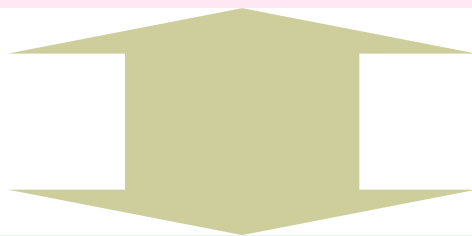
小数

$$[A]_{\text{补}} + [B]_{\text{补}} = [A+B]_{\text{补}} \pmod{2}$$

精髓:

和的补码等于补码的和

两个用补码表示的数在进行加法运算时，可以把符号位与数值位同等处理，运算后的结果按2或 $2^{n+1}$ 取模，即得本次运算结果。



即在模2(或模 $2^{n+1}$ )意义下，任意两数的补码之和等于两数之和的补码。



# 示例

例： 已知 $X=+0.10011$ ，  $Y=+0.01001$ ， 求 $X+Y$ 。

解：  $[X]_{\text{补}}+[Y]_{\text{补}}=0.10011+0.01001=0.11100$

$$[X+Y]_{\text{补}}=[0.10011+0.01001]_{\text{补}}=0.11100$$

这里 $X$ 和 $Y$ 均为正数， 得到

$[X]_{\text{补}}+[Y]_{\text{补}}=[X+Y]_{\text{补}}$ ， 符合补码加法规律。

例： 已知 $X=-0.10011$ ，  $Y=+0.10001$ ， 求 $X+Y$ 。

解：  $[X]_{\text{补}}=1.01101$      $[Y]_{\text{补}}=0.10001$

$$\begin{aligned}[X]_{\text{补}}+[Y]_{\text{补}} &= 1.01101+0.10001 \\ &= 1.11110\end{aligned}$$

$$\begin{aligned}[X+Y]_{\text{补}} &= [-0.10011+0.10001]_{\text{补}} \\ &= [-0.00010]_{\text{补}}=1.11110\end{aligned}$$

这里 $X$ 和 $Y$ 为一负一正， 同样有：  $[X]_{\text{补}}+[Y]_{\text{补}}=[X+Y]_{\text{补}}$ 。

# 补码减法运算

$$\left\{ \begin{array}{l} \text{【A】}_{\text{补}} + \text{【-B】}_{\text{补}} = \text{【A-B】}_{\text{补}} \pmod{2^{n+1}} \\ \text{【A】}_{\text{补}} - \text{【B】}_{\text{补}} = \text{【A-B】}_{\text{补}} \pmod{2^{n+1}} \end{array} \right.$$

$$\left\{ \begin{array}{l} \text{【A】}_{\text{补}} + \text{【-B】}_{\text{补}} = \text{【A-B】}_{\text{补}} \pmod{2} \\ \text{【A】}_{\text{补}} - \text{【B】}_{\text{补}} = \text{【A-B】}_{\text{补}} \pmod{2} \end{array} \right.$$

例：已知 $X=0.11001$ ， $Y=0.10011$ ，求 $X-Y$ 。

解：

$$[X]_{\text{补}} = 0.11001, [Y]_{\text{补}} = 0.10011, \text{ 则 } [-Y]_{\text{补}} = 1.01101$$

$$[X-Y]_{\text{补}} = [X]_{\text{补}} + [-Y]_{\text{补}} = 0.11001 + 1.01101 = \underline{1}0.00110$$



# 补码运算溢出判断

例：已知  $x = +0.1011$ ,  $y = +0.1001$ ,  
求  $x+y$ 。

解

$$[x]_{\text{补}} = 0.1011 \quad [y]_{\text{补}} = 0.1001$$

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

$$\begin{array}{r} = 0.1011 \\ + 0.1001 \\ \hline = 1.0100 \end{array}$$

$x+y < 0$

出错啦！

例：已知  $x = -0.1101$ ,  $y = -0.1011$ ,  
求  $x+y$ 。

解

$$[x]_{\text{补}} = 1.0010 \quad [y]_{\text{补}} = 1.0101$$

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

$$\begin{array}{r} = 1.0011 \\ + 1.0101 \\ \hline = 0.1000 \end{array}$$

丢掉

$x+y > 0$

出错啦！

# 溢出判断方法

---

当运算结果超出机器数所能表示的范围时，称为溢出。很显然，若两个异号数相加或两个同号数相减时，其结果是不会产生溢出的。仅当两个同号数相加或异号数相减时才有可能产生溢出，一旦产生溢出，则结果就是不正确的。

一般的判断溢出的方法有：

- (1) 进位判别法
- (2) 双符号位判别法
- (3) 其他判别法



# 进位判别法

例：设  $A = -0.1000$ ,  $B = -0.1000$  求  $【A+B】_{补}$

解

$$【A】_{补} = 1.1000 \quad 【B】_{补} = 1.1000$$

$$【A+B】_{补} = 【A】_{补} + 【B】_{补}$$

$$\begin{array}{r} \text{丢掉} \quad = \quad 1.1000 \\ + \quad 1.1000 \\ \hline 1.0000 \end{array}$$

两个操作数符号均为1，结果的符号为1，二者相同故未溢出。

硬件实现 最高有效位的进位  $\oplus$  符号位的进位 = 1 溢出

如

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

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

# 双符号位判别法

例:  $x = +\frac{11}{16}$ ,  $y = +\frac{3}{16}$ ,  
求  $x+y$

解

$$【x】_{补} = 00.1011 \quad 【y】_{补} = 00.0011$$

$$【x+y】_{补} = 【x】_{补} + 【y】_{补}$$

$$\begin{array}{r} = 00.1011 \\ + 00.0011 \\ \hline 00.1110 \end{array}$$

- ① 两个符号位“00”，表示未溢出；
- ② 结果正确，结果为 0.1110

例:  $x = -\frac{11}{16}$ ,  $y = -\frac{7}{16}$ ,  
求  $x+y$

解

$$【x】_{补} = 11.0101 \quad 【y】_{补} = 11.1001$$

$$【x+y】_{补} = 【x】_{补} + 【y】_{补}$$

$$\begin{array}{r} = 11.0101 \\ + 11.1001 \\ \hline 10.1110 \end{array}$$

丢掉

- ① 两个符号位出现“10”，表示已溢出，即结果小于1；
- ② 但最高符号位“1”，是正确的。

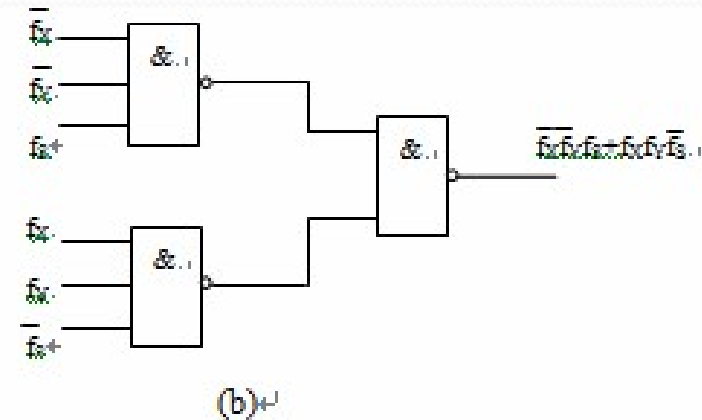
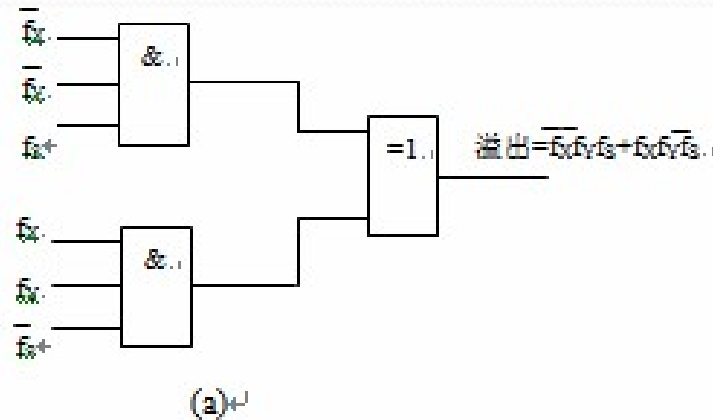
- ① 上述方法及结论对于整数同样试用之。
- ② 机器通过逻辑电路可自动检查出这种溢出，并进行中断处理。



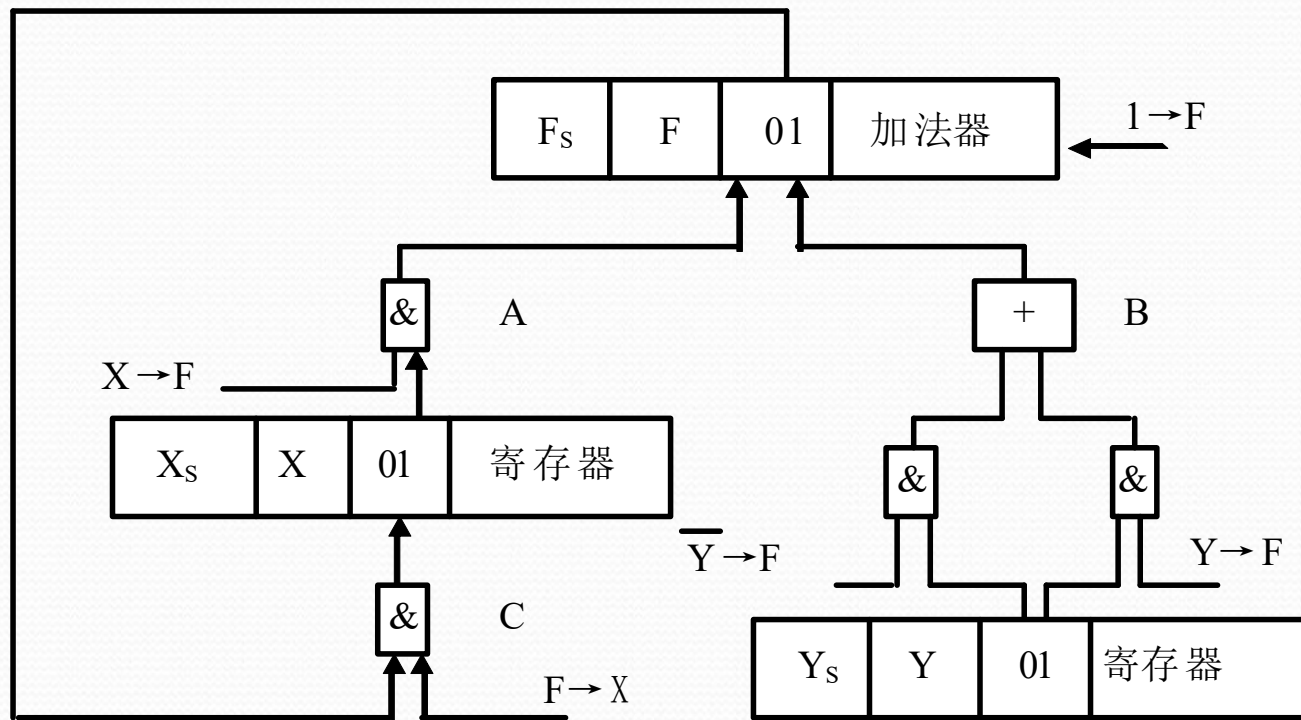
# 其他判别法

当符号相同的两个数相加时，如果结果的符号与加数(或被加数)符号不同，则说明溢出。即溢出条件为：

$$\overline{f_A} \overline{f_B} f_S + f_A f_B \overline{f_S}$$



# 补码加减法运算实现





# 定点乘法运算

---

- 笔算乘法分析
- 原码乘法
- 补码乘法\*
- 并行乘法

# 分析笔算乘法

---

$$A = -0.1101 \quad B = 0.1011$$

$$A \times B = -0.10001111 \quad \text{乘积的符号心算求得}$$

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

✓ 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

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

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

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

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

$$\text{右移一位} \quad = 0.1 \{A + 0.1[0 \cdot A + 0.1(A + 0.1A)]\}$$

$$= 2^{-1} \{A + 2^{-1}[0 \cdot A + 2^{-1}(A + 2^{-1}(A + 0))]\}$$

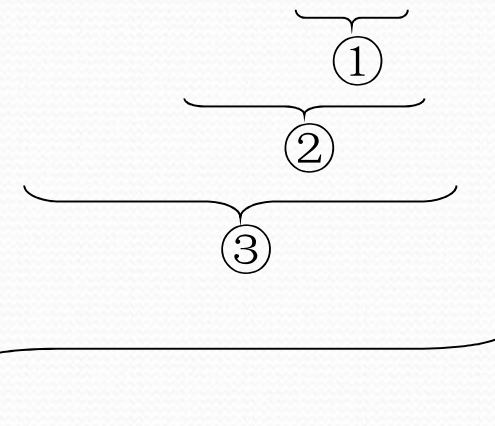
第一步 被乘数  $A + 0$

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



# 改进后的乘法过程(竖式)

部分积	乘数	说明
$\begin{array}{r} 0.0000 \\ + 0.1101 \\ \hline \end{array}$	$\begin{array}{r} 1011 \\ \hline \end{array}$	初态，部分积 = 0 乘数为 1，加被乘数
$\begin{array}{r} 0.1101 \\ 0.0110 \\ + 0.1101 \\ \hline \end{array}$	$\begin{array}{r} 1101 \\ \hline \end{array}$	→ 1，形成新的部分积 乘数为 1，加被乘数
$\begin{array}{r} 1.0011 \\ 0.1001 \\ + 0.0000 \\ \hline \end{array}$	$\begin{array}{r} 1 \\ 1110 \\ \hline \end{array}$	→ 1，形成新的部分积 乘数为 0，加 0
$\begin{array}{r} 0.1001 \\ 0.0100 \\ + 0.1101 \\ \hline \end{array}$	$\begin{array}{r} 11 \\ 1111 \\ \hline \end{array}$	→ 1，形成新的部分积 乘数为 1，加被乘数
$\begin{array}{r} 1.0001 \\ 0.1000 \\ \hline \end{array}$	$\begin{array}{r} 111 \\ 1111 \\ \hline \end{array}$	→ 1，得结果



# 小结

---

➤ 乘法运算可用加和移位实现

$n=4$ ，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，  
然后右移1位形成新的部分积，同时乘数右移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

# 定点乘法运算

---

- 笔算乘法分析
- 原码乘法
  - 原码一位乘法
  - 原码两位乘法
- 补码乘法\*
- 并行乘法



# 原码一位乘法运算规则

---

以小数为例

$$\text{设}[x]_{\text{原}} = x_0.x_1x_2 \cdots x_n$$

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

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

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

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

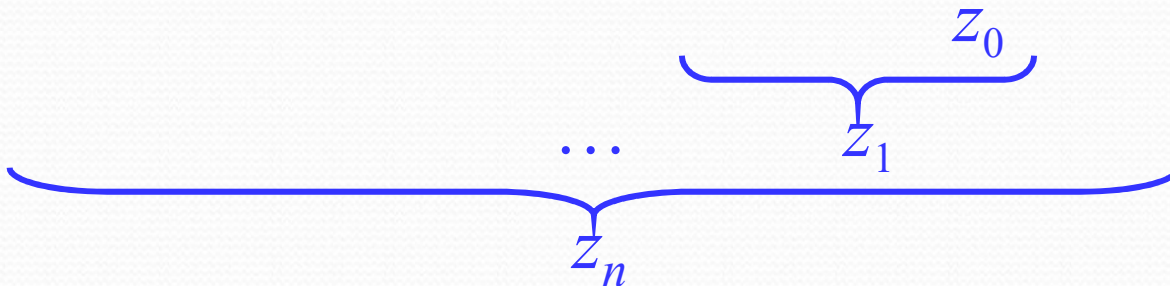
$y^* = 0.y_1y_2 \cdots y_n$  为  $y$  的绝对值

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  $x^* \cdot y^*$

# 原码一位乘法递推公式

---

$$\begin{aligned}x^*.y^* &= x^*(0.y_1y_2 \dots y_n) \\&= x^*(y_12^{-1}+y_22^{-2}+ \dots + y_n2^{-n}) \\&= 2^{-1}(y_1x^*+2^{-1}(y_2x^*+ \dots 2^{-1}(y_nx^* + 0) \dots))\end{aligned}$$


$$z_0 = 0$$

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

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

$$\vdots$$

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



例题： 已知  $x = -0.1110$   $y = 0.1101$  求  $[x \cdot y]_{\text{原}}$

解： 数值部分的运算

	部分积	乘数	说明
	$\begin{array}{r} 0.0000 \\ + 0.1110 \\ \hline \end{array}$	$\underline{1101}$	部分积 初态 $z_0 = 0$ $+ x^*$
逻辑右移	$\begin{array}{r} 0.1110 \\ 0.0111 \\ + 0.0000 \\ \hline \end{array}$	$\underline{0110}$	$\rightarrow 1$ , 得 $z_1$ $+ 0$
逻辑右移	$\begin{array}{r} 0.0111 \\ 0.0011 \\ + 0.1110 \\ \hline \end{array}$	$\underline{1011}$	$\rightarrow 1$ , 得 $z_2$ $+ x^*$
逻辑右移	$\begin{array}{r} 1.0001 \\ 0.1000 \\ + 0.1110 \\ \hline \end{array}$	$\underline{1101}$	$\rightarrow 1$ , 得 $z_3$ $+ x^*$
逻辑右移	$\begin{array}{r} 1.0110 \\ 0.1011 \\ \hline \end{array}$	$110$ $0110$	$\rightarrow 1$ , 得 $z_4$

## 例题结果

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

$$x^* \cdot y^* = 0.10110110$$

则  $[x \cdot y]_{\text{原}} = 1.10110110$

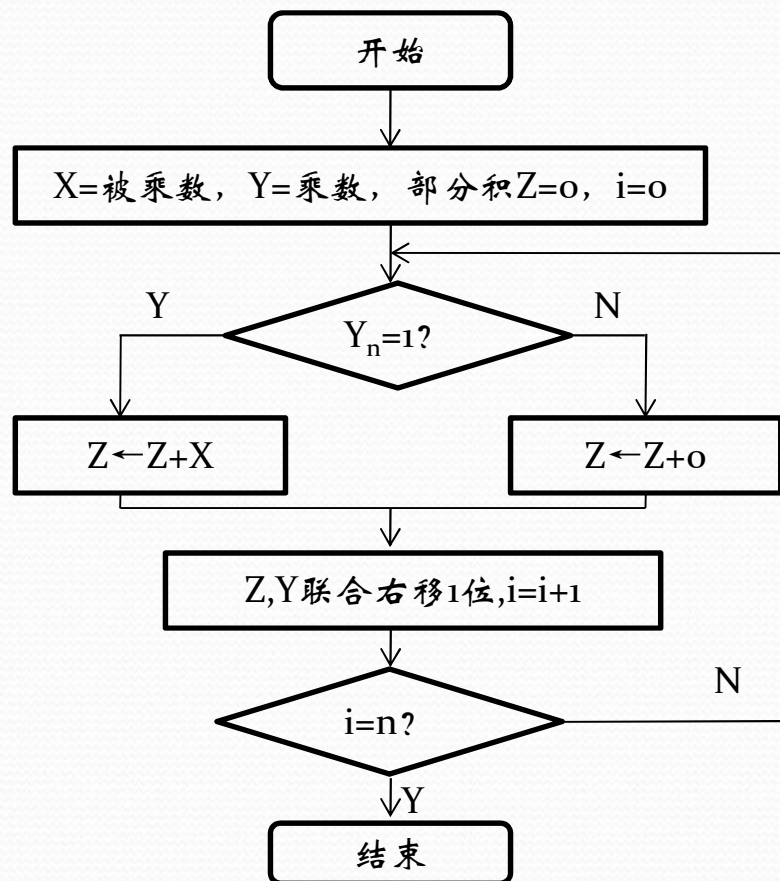
特点      绝对值运算

用移位的次数判断乘法是否结束

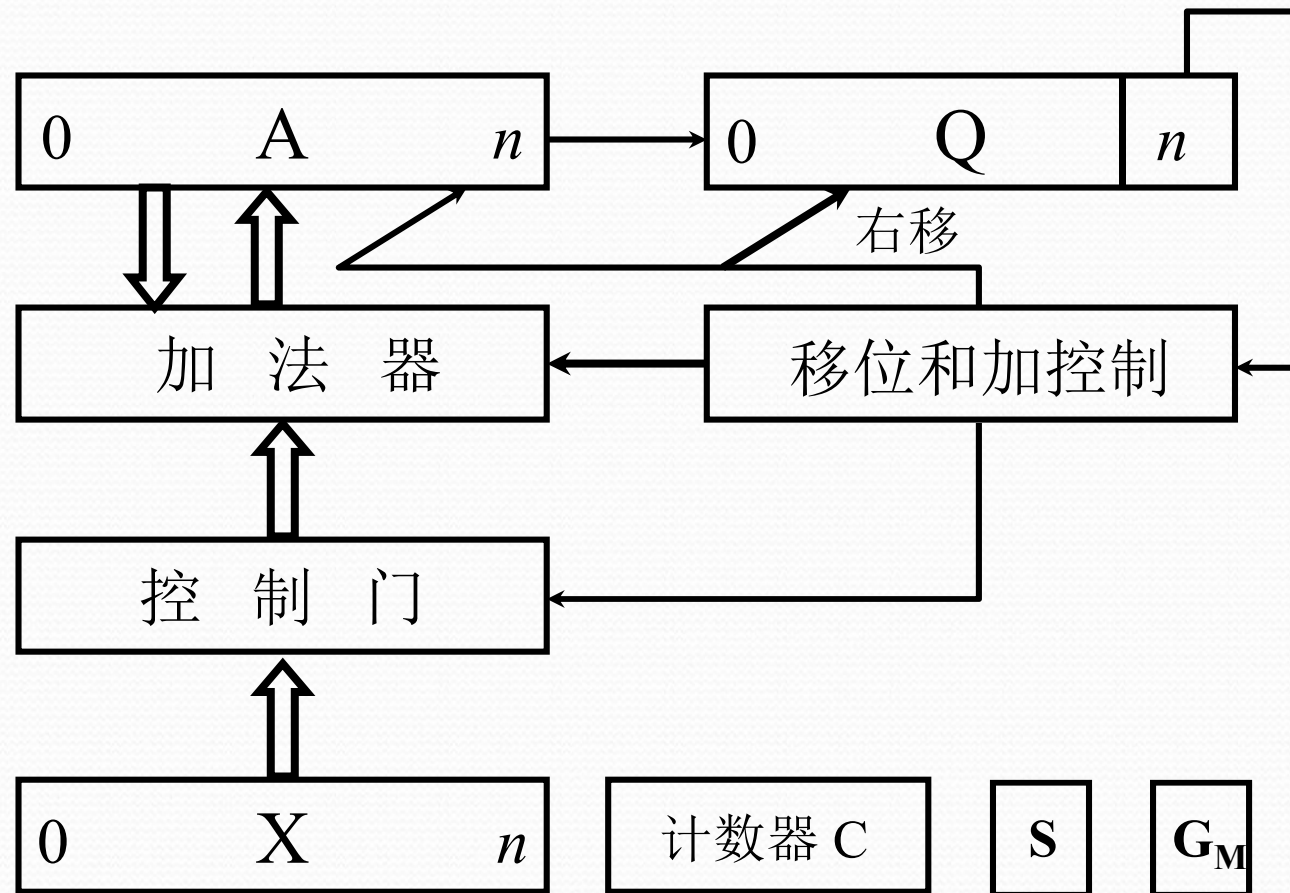
逻辑移位



# 原码一位乘法流程



# 原码一位乘法硬件配置



A、X、Q 均  $n+1$  位

移位和加受末位乘数控制



# 定点乘法运算

---

- 笔算乘法分析
- 原码乘法
  - 原码一位乘法
  - 原码两位乘法
- 补码乘法\*
- 并行乘法

# 原码两位乘法

原码乘      符号位 和 数值位 部分 分开运算

两位乘      每次用 乘数的 2 位判断 原部分积  
是否加 和 如何加 被乘数

乘数 $y_{n-1} y_n$	新的部分积
0 0	加 “0” $\rightarrow$ 2
0 1	加 1 倍的被乘数 $\rightarrow$ 2
1 0	加 2 倍的被乘数 $\rightarrow$ 2
1 1	加 3 倍的被乘数 $\rightarrow$ 2

$$\begin{array}{r} 3? \quad 4 \quad 100 \\ \quad -1 \quad -01 \\ \hline \quad 3 \quad 11 \end{array}$$

先 减 1 倍 的被乘数  
再 加 4 倍 的被乘数



例题：已知  $x = 0.111111$   $y = -0.111001$  求  $[x \cdot y]_{\text{原}}$

解：数值部分的运算

	部分积	乘数	$C_j$	说明
	000.0000000	00.11100 <u>01</u>	<u>0</u>	初态 $z_0 = 0$
	+000.111111			$+x^*$ , $C_j = 0$
补码右移	000.111111			
	000.001111	11 0011 <u>10</u>	<u>0</u>	$\rightarrow 2$
	+001.111110			$+2x^*$ , $C_j = 0$
补码右移	010.001101	11		
	000.100011	0111 00 <u>11</u>	<u>0</u>	$\rightarrow 2$
	+111.000001			$-x^*$ , $C_j = 1$
补码右移	111.100100	0111		
	111.111001	000111 <u>00</u>	<u>1</u>	$\rightarrow 2$
	+000.111111			$+x^*$ , $C_j = 0$
	000.111000	000111		

## 例题结果

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

② 数值部分的运算

$$x^* \cdot y^* = 0.111000000111$$

$$\text{则 } [x \cdot y]_{\text{原}} = 1.111000000111$$

特点      绝对值的补码运算

用移位的次数判断乘法是否结束

算术移位



# 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)

思考  $n$  为奇数时，原码两位乘 移？次 最多加？次

# 定点乘法运算

---

- 笔算乘法分析
- 原码乘法
- 补码乘法\*
  - 补码一位乘法 (Booth算法)
  - 补码两位乘法
- 并行乘法



# 运算规则

---

以小数为例 设被乘数  $[x]_{\text{补}} = x_0.x_1x_2 \cdots x_n$

乘数  $[y]_{\text{补}} = y_0.y_1y_2 \cdots y_n$

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 $[y]_{\text{补}}$ ，去掉符号位

最后加 $[-x]_{\text{补}}$ ，校正

# Booth算法

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

$$[x \cdot y]_{\text{补}}$$

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

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

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

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

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

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

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

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

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

附加位  $y_{n+1}$

$$y_1 2^{-1} + \cdots + y_n 2^{-n}$$



# Booth算法递推公式

$$[z_0]_{\text{补}} = 0$$

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

⋮

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

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

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	$+ [x]_{\text{补}} \rightarrow 1$
1	0	-1	$+ [-x]_{\text{补}} \rightarrow 1$
1	1	0	$\rightarrow 1$

例题： 已知  $x = +0.0011$   $y = -0.1011$  求  $[x \cdot y]_{\text{补}}$

解：	0 0 . 0 0 0 0	1 . 0 1 0 <u>1</u> <u>0</u>	
	+ 1 1 . 1 1 0 1		$+[-x]_{\text{补}}$
	1 1 . 1 1 0 1		
补码 右移	1 1 . <u>1</u> 1 1 0	1 1 0 1 <u>0</u> <u>1</u>	$\rightarrow 1$
	+ 0 0 . 0 0 1 1		$+ [x]_{\text{补}}$
	0 0 . 0 0 0 1	1	
补码 右移	0 0 . <u>0</u> 0 0 0	1 1 1 0 <u>1</u> <u>0</u>	$\rightarrow 1$
	+ 1 1 . 1 1 0 1		$+ [-x]_{\text{补}}$
	1 1 . 1 1 0 1	1 1	
补码 右移	1 1 . <u>1</u> 1 1 0	1 1 1 1 <u>0</u> <u>1</u>	$\rightarrow 1$
	+ 0 0 . 0 0 1 1		$+ [x]_{\text{补}}$
	0 0 . 0 0 0 1	1 1 1	
补码 右移	0 0 . <u>0</u> 0 0 0	1 1 1 1 <u>1</u> <u>0</u>	$\rightarrow 1$
	+ 1 1 . 1 1 0 1		$+ [-x]_{\text{补}}$
	1 1 . 1 1 0 1	1 1 1 1	最后一步不移位

$$[x]_{\text{补}} = 0.0011$$

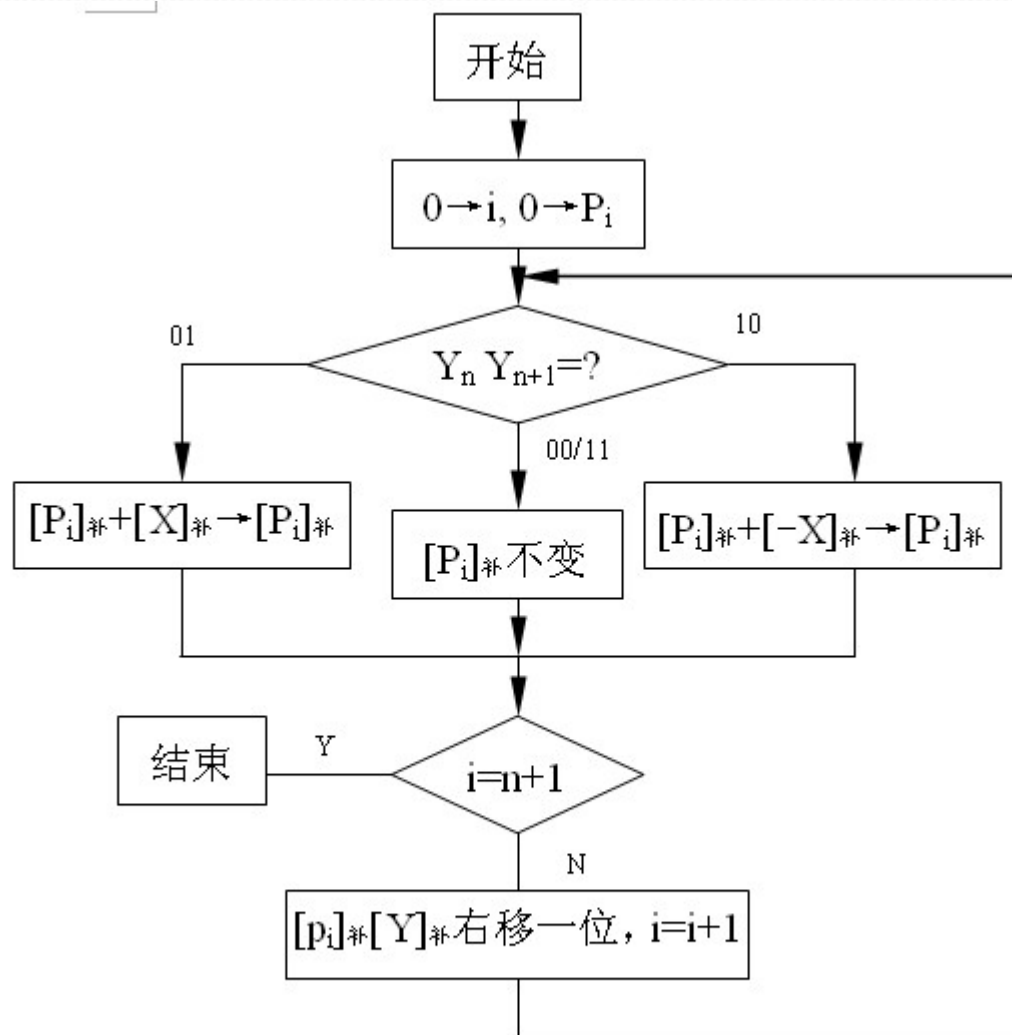
$$[y]_{\text{补}} = 1.0101$$

$$[-x]_{\text{补}} = 1.1101$$

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

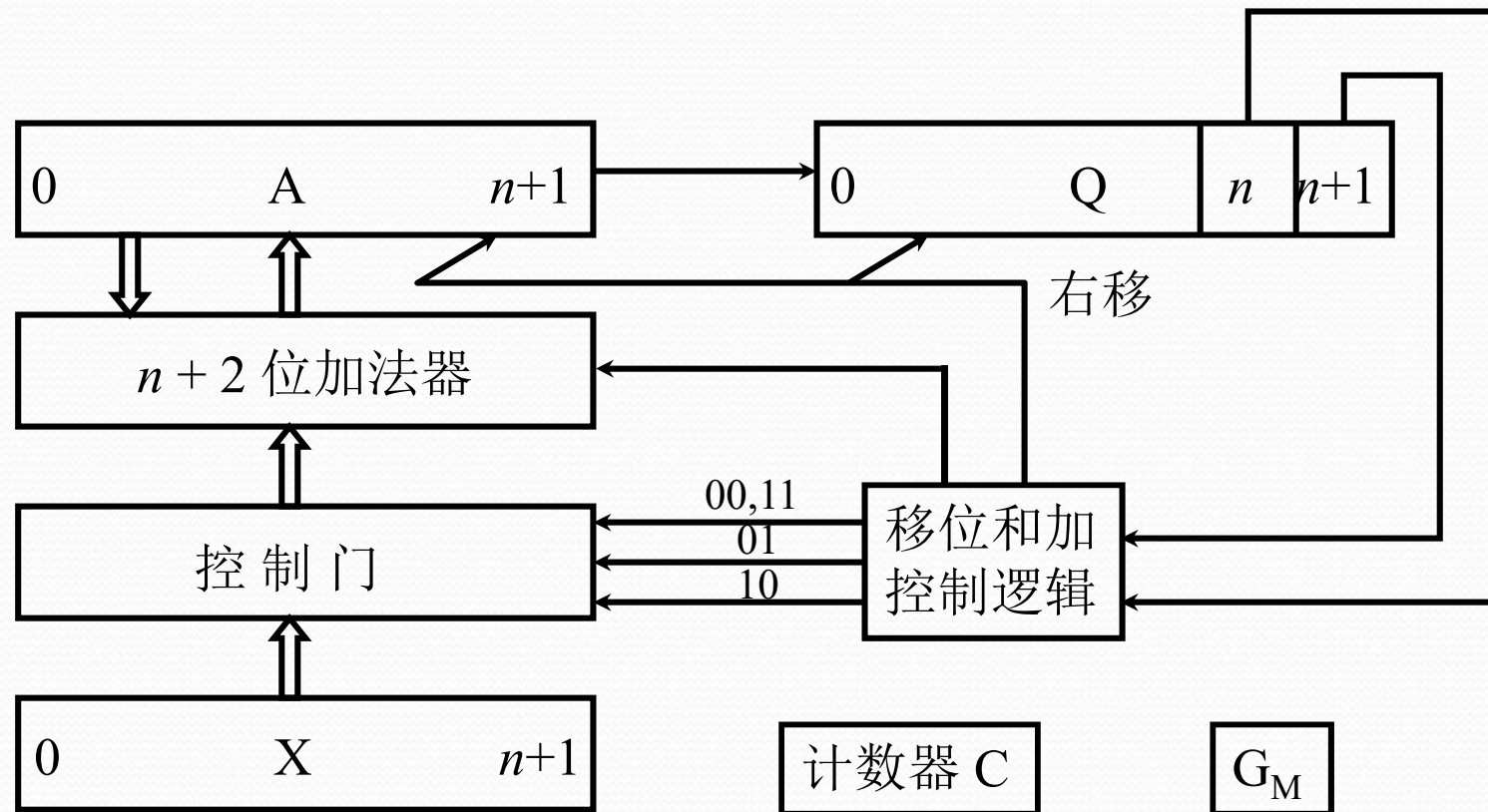


# Booth算法的流程



图补码一位乘法算法流程图

# Booth算法硬件配置



A、X、Q 均  $n+2$  位  
移位和加受末两位乘数控制



# 定点乘法运算

---

- 笔算乘法分析
- 原码乘法
- 补码乘法\*
  - 补码一位乘法 (Booth算法)
  - 补码两位乘法
- 并行乘法

# 补码两位乘法

---

设上次乘法的部分积为 $[Z']_{\text{补}}$ ，本次运算结束之后的部分积为

$[Z'']_{\text{补}}$ ，根据Booth算法有：

$$[Z']_{\text{补}} = 2^{-1} \{ [Z]_{\text{补}} + (Y_{i+1} - Y_i)[X]_{\text{补}} \}$$

$$[Z'']_{\text{补}} = 2^{-1} \{ [Z']_{\text{补}} + (Y_i - Y_{i-1})[X]_{\text{补}} \}$$

$$= 2^{-1} \{ 2^{-1} \{ [Z]_{\text{补}} + (Y_{i+1} - Y_i)[X]_{\text{补}} \} + (Y_i - Y_{i-1})[X]_{\text{补}} \}$$

$$= 2^{-2} \{ [Z]_{\text{补}} + (Y_{i+1} - Y_i)[X]_{\text{补}} + 2(Y_i - Y_{i-1})[X]_{\text{补}} \}$$

$$= 2^{-2} \{ [Z]_{\text{补}} + (Y_{i+1} + Y_i - 2Y_{i-1})[X]_{\text{补}} \}$$

由此可见，三位共有8种组合值及其对应操作，如下表所示。



### 组合值 $Y_{n+1}$ , $Y_n$ , $Y_{n-1}$ 与部分积补码的关系

$\underline{Y_{n-1}}_{-1}^{\circ}$	$\underline{Y_{n-1}}^{\circ}$	$\underline{Y_{n-1}}_{+1}^{\circ}$	组合值 $^{\circ}$	部分积 $^{\circ}$
0 $^{\circ}$	0 $^{\circ}$	0 $^{\circ}$	0 $^{\circ}$	$\{[\underline{Z_i}]_{\text{补}}+0\} * 2^{-2^{\circ}}$
0 $^{\circ}$	0 $^{\circ}$	1 $^{\circ}$	1 $^{\circ}$	$\{[\underline{Z_i}]_{\text{补}}+[\underline{X}]_{\text{补}}\} * 2^{-2^{\circ}}$
0 $^{\circ}$	1 $^{\circ}$	0 $^{\circ}$	1 $^{\circ}$	$\{[\underline{Z_i}]_{\text{补}}+[\underline{X}]_{\text{补}}\} * 2^{-2^{\circ}}$
0 $^{\circ}$	1 $^{\circ}$	1 $^{\circ}$	2 $^{\circ}$	$\{[\underline{Z_i}]_{\text{补}}+2[\underline{X}]_{\text{补}}\} * 2^{-2^{\circ}}$
1 $^{\circ}$	0 $^{\circ}$	0 $^{\circ}$	-2 $^{\circ}$	$\{[\underline{Z_i}]_{\text{补}}+2[-\underline{X}]_{\text{补}}\} * 2^{-2^{\circ}}$
1 $^{\circ}$	0 $^{\circ}$	1 $^{\circ}$	-1 $^{\circ}$	$\{[\underline{Z_i}]_{\text{补}}+[-\underline{X}]_{\text{补}}\} * 2^{-2^{\circ}}$
1 $^{\circ}$	1 $^{\circ}$	0 $^{\circ}$	-1 $^{\circ}$	$\{[\underline{Z_i}]_{\text{补}}+[-\underline{X}]_{\text{补}}\} * 2^{-2^{\circ}}$
1 $^{\circ}$	1 $^{\circ}$	1 $^{\circ}$	0 $^{\circ}$	$\{[\underline{Z_i}]_{\text{补}}+0\} * 2^{-2^{\circ}}$

在执行补码两位乘过程中有5种操作，即：

$[\underline{Z_i}]_{\text{补}}+0$ ,  $[\underline{Z_i}]_{\text{补}}+[\underline{X}]_{\text{补}}$ ,  $[\underline{Z_i}]_{\text{补}}+[-\underline{X}]_{\text{补}}$ ,  $[\underline{Z_i}]_{\text{补}}+2[\underline{X}]_{\text{补}}$ ,  $[\underline{Z_i}]_{\text{补}}+2[-\underline{X}]_{\text{补}}$ 。

# 补码两位乘法

---

补码两位乘中求部分积的次数和右移操作的控制问题：  
被乘数和部分积取3位符号位，当乘数的数值位 $n$ 为偶数时，乘数取双符号位，共需作 $n/2+1$ 次累加， $n/2$ 次移位（最后一次不移位）；当 $n$ 为奇数时，乘数只取一个符号位，共需作 $(n+1)/2$ 次累加和移位，但最后一次仅移位1位。



例：  $X = -0.1101$ ,  $Y = 0.0110$ ,  
求  $[X \cdot Y]_{\text{补}}$ 。

解：取3位符号位，

$$[X]_{\text{补}} = 111.0011,$$

$$[-X]_{\text{补}} = 000.1101,$$

$$2[-X]_{\text{补}} = 001.1010,$$

$$2[X]_{\text{补}} = 110.0110,$$

$$[Y]_{\text{补}} = 0.0110$$

$n=4$ , 乘数取2位符号位

部分积 $P_i$	乘数	附加位	说明
000. 0 0 0 0	0 0. 0 1 1 0	0	初始值
+ 001. 1 0 1 0			加 $2[-X]_{\text{补}}$
001. 1 0 1 0			
→ 000. 0 1 1 0	1 0	1	右移两位
+ 110. 0 1 1 0			加 $2[X]_{\text{补}}$
110. 1 1 0 0			
→ 111. 1 0 1 1	0 0	0	右移两位
+ 000. 0 0 0 0			加 0
111. 1 0 1 1	0 0		不移位

例：  $X=0.0110011$ ,  $Y=-0.0110010$ ,  
求  $[X \cdot Y]_{\text{补}}$ 。  $[x^*y]_{\text{补}}=1.11011000001010$

请自己演算。(参见蒋本珊教材P.111-112)

解：取3位符号位，

$$[X]_{\text{补}}=000.0110011,$$

$$[-X]_{\text{补}}=111.1001101,$$

$$2[X]_{\text{补}}=000.1100110,$$

$$2[-X]_{\text{补}}=111.0011010,$$

$$[Y]_{\text{补}}=1.1001110$$

$n=7$ , 乘数取1位符号位，共做4次累加和移位，最后一次只移1位。



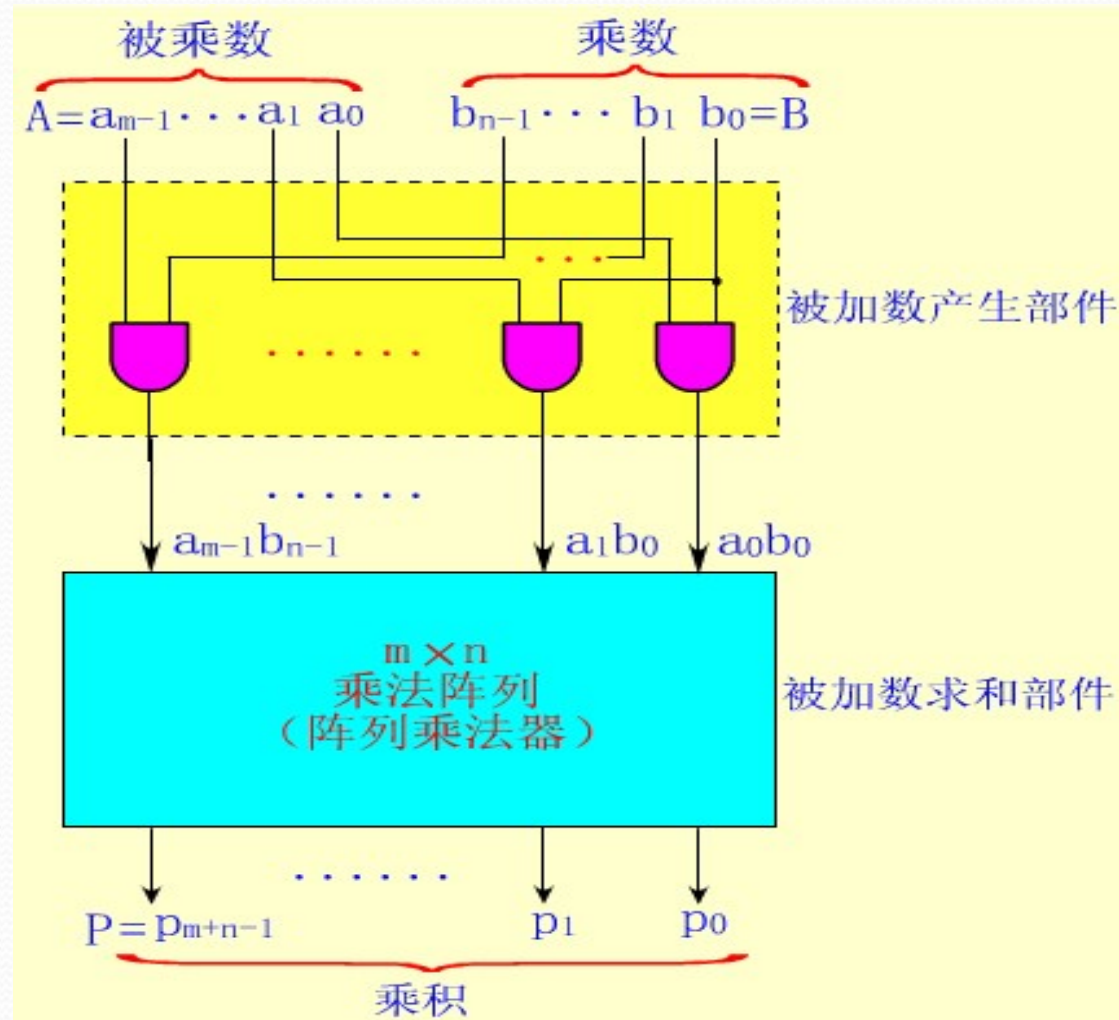
# 定点乘法运算

---

- 笔算乘法分析
- 原码乘法
- 补码乘法\*
- 并行乘法

# 并行乘法

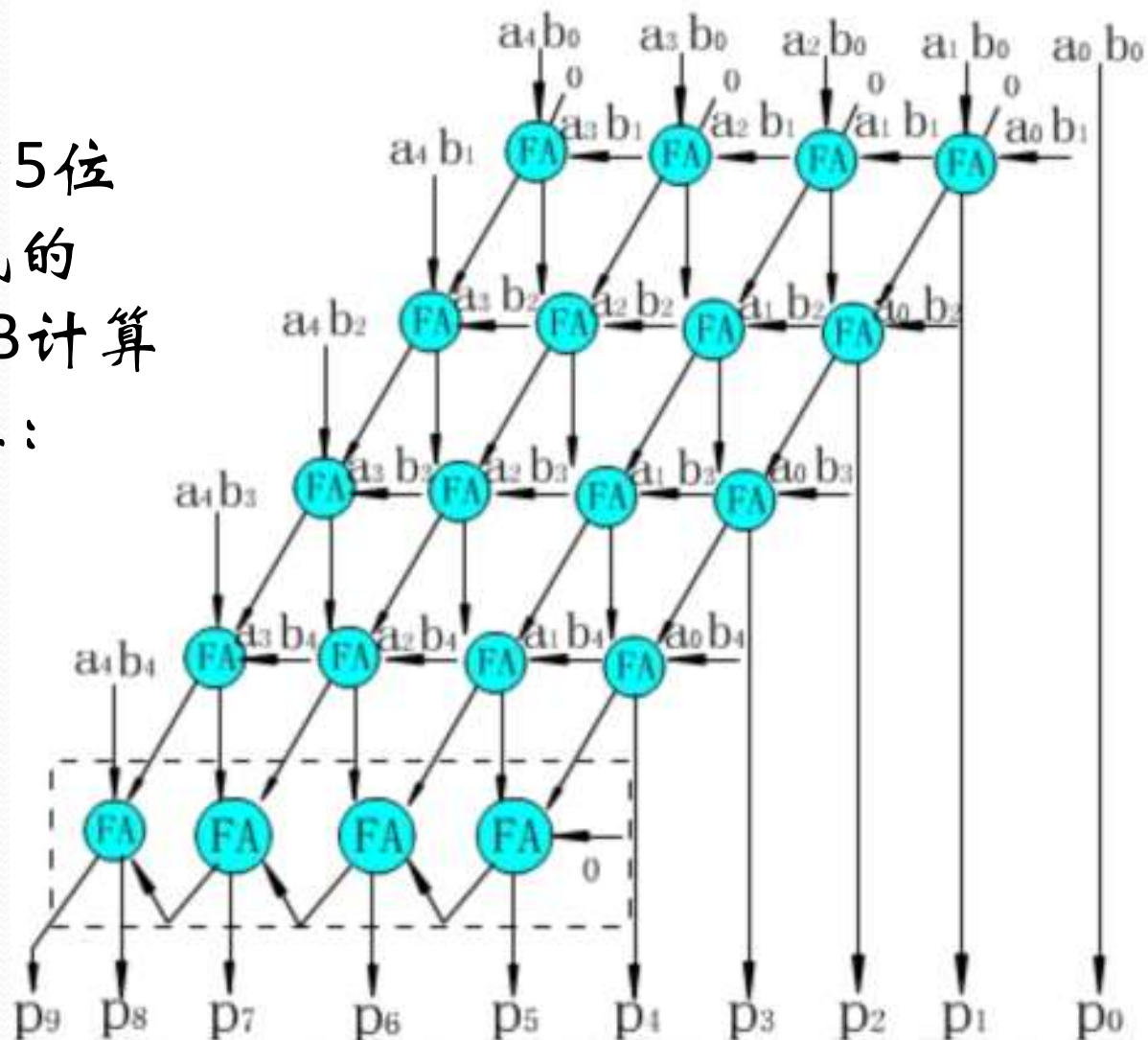
设A和B为  
无符号数  
，A由m  
位构成，  
B由n位构  
成，求  
 $A \times B$





# 并行乘法

各由5位  
构成的  
 $A \times B$ 计算  
过程：



# 定点乘法小结

---

- 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



# 定点除法运算

---

- 笔算除法分析
- 原码一位除法
  - 恢复余数法
  - 加减交替法
  - 跳0跳1除法
- 补码一位除法

# 笔算除法分析

$$x = -0.1011 \quad y = 0.1101 \quad \text{求 } x \div y$$

$$\begin{array}{r}
 \phantom{0.}0.1101 \\
 0.1101 \overline{) 0.1101} \\
 \underline{0.01101} \phantom{0} \\
 0.01001 \phantom{0} \\
 \underline{0.001101} \phantom{0} \\
 0.000101 \phantom{00} \\
 \underline{0.00001101} \phantom{0} \\
 0.00000111
 \end{array}$$

✓ 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

$$x \div y = -0.1101 \quad \text{商符心算求得}$$

$$\text{余数 } 0.00000111$$



# 定点除法运算

---

- 笔算除法分析
- 原码一位除法
  - 恢复余数法
  - 加减交替法
  - 跳0跳1除法
- 补码一位除法

# 笔算除法和机器除法比较

---

## 笔算除法

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

2 倍字长加法器

上商位置 不固定

## 机器除法

符号位异或形成

$|x| - |y| > 0$  上商 1

$|x| - |y| < 0$  上商 0

余数 左移一位 低位补“0”  
减 除数

1 倍字长加法器

在寄存器 最末位上商



# 原码一位除法

以小数为例：

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

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

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

式中  $x^* = 0.x_1x_2 \dots x_n$  为  $x$  的绝对值  
 $y^* = 0.y_1y_2 \dots y_n$  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

---

上商0还是1，用做减法判结果的符号为负还是正来确定。当差为负时，上商0，同时还应把除数再加到差上去，恢复余数为原来的正值之后再将其左移一位。若减得的差为0或为正值时，就没有恢复余数的操作。上商为1，余数左移一位。



例：

$X = -0.1011$ ,

$Y = 0.1101$ ,

求  $X \div Y$ 。

解：

$[-Y]_{\text{补}} = 11.0011$

商的符号为

$1 \oplus 0 = 1$ ,

结果  $X \div Y =$

$-0.1101$ ,

余数

$0.0111 \times 2^{-4}$

	被除数(余数)	商	说明
	00. 1 0 1 1	0 0 0 0 0	初始值
+	11. 0 0 1 1		加 $[-Y]_{\text{补}}$
	11. 1 1 1 0	0 0 0 0 0	不够减, 商上 0
+	00. 1 1 0 1		+Y, 恢复余数
←	00. 1 0 1 1		余数与商左移一位
	01. 0 1 1 0	0 0 0 0 0	
+	11. 0 0 1 1		加 $[-Y]_{\text{补}}$
	00. 1 0 0 1	0 0 0 0 1	够减, 商上 1
←	01. 0 0 1 0	0 0 0 1 0	余数与商左移一位
+	11. 0 0 1 1		加 $[-Y]_{\text{补}}$
	00. 0 1 0 1	0 0 0 1 1	够减, 商上 1
←	00. 1 0 1 0	0 0 1 1 0	余数与商左移一位
+	11. 0 0 1 1		加 $[-Y]_{\text{补}}$
	11. 1 1 0 1	0 0 1 1 0	不够减, 商上 0
+	00. 1 1 0 1		+Y, 恢复余数
	00. 1 0 1 0		
←	01. 0 1 0 0	0 1 1 0 0	余数与商左移一位
+	11. 0 0 1 1		加 $[-Y]_{\text{补}}$
	00. 0 1 1 1	0 1 1 0 1	够减, 商上 1

# 加减交替法

---

在恢复余数法中，若第 $i-1$ 次求商的部分余数为 $r_{i-1}$ ，则第 $i$ 次求商操作为：

$$r_i = 2r_{i-1} - Y$$

当余数为正时，即 $r_i > 0$ ，商上1。

若余数为负，即 $r_i < 0$ ，恢复余数， $r' = r_i + Y$ ，然后做第 $i+1$ 步，即被除数左移1位减除数，得新余数。

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

此式即是不恢复余数法的理论依据！



# 加减交替法

---

由此得到不恢复余数法的运算规则，也称**加减交替法**。

- 被除数、除数采用双符号位，符号位单独处理，但运算过程中采用补码运算
- 用被除数减去除数，得到余数
- 若余数为正，商上1，部分余数左移1位，再减去除数；
- 若余数为负，商上0，部分余数左移1位，再加上除数。

但若最后一次上商为0而又需得到正确余数，则在这最后一次仍需恢复余数。

例

$$X = -0.1011$$

$$Y = 0.1101$$

用加减交替

法求  $X \div Y$

解：

$$[-Y]_{\text{补}} = 11.0011$$

$$\text{商符: } 1 \oplus 0 = 1$$

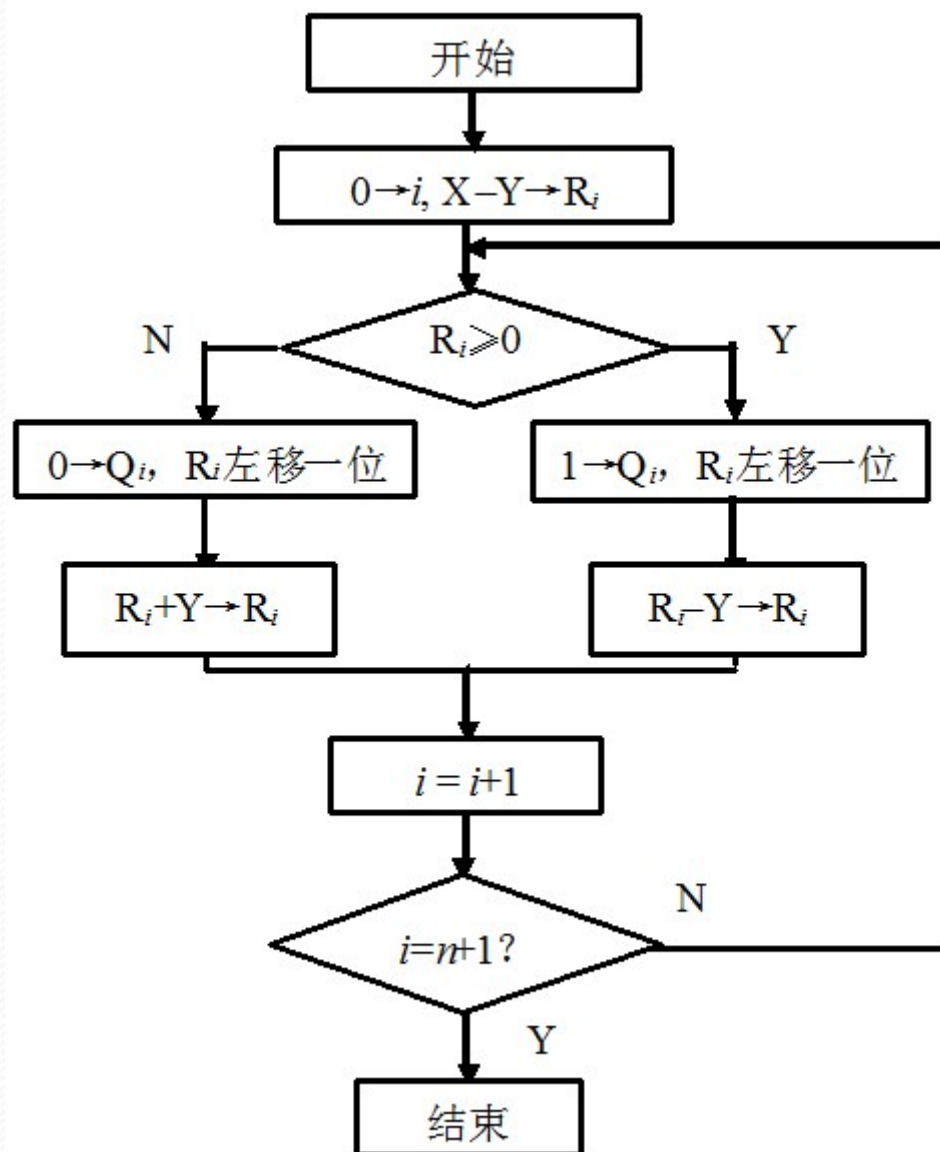
$$\text{商: } -0.1101$$

$$\text{余: } 0.0111 \times 2^{-4}$$

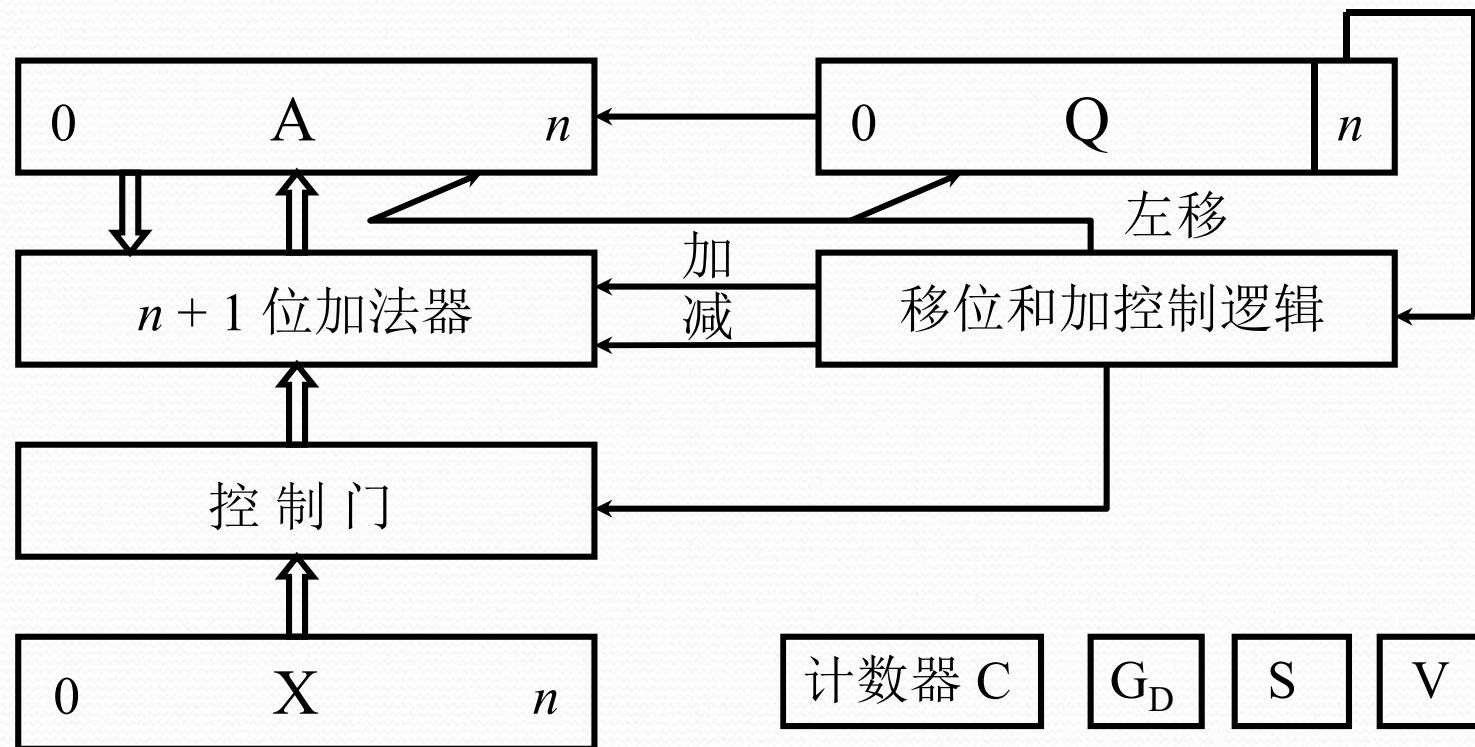
	被除数(余数 $R_i$ )	$Q_i$	说明
	00. 1 0 1 1	0 0 0 0 0	初始值
+	11. 0 0 1 1		加 $[-Y]_{\text{补}}$
	11. 1 1 1 0	0 0 0 0 0	不够减, 商上 0
←	11. 1 1 0 0	0 0 0 0 0	余数与商左移一位
+	00. 1 1 0 1		加 $Y$
	00. 1 0 0 1	0 0 0 0 1	够减, 商上 1
←	01. 0 0 1 0	0 0 0 1 0	余数与商左移一位
+	11. 0 0 1 1		加 $[-Y]_{\text{补}}$
	00. 0 1 0 1	0 0 0 1 1	够减, 商上 1
←	00. 1 0 1 0	0 0 1 1 0	余数与商左移一位
+	11. 0 0 1 1		加 $[-Y]_{\text{补}}$
	11. 1 1 0 1	0 0 1 1 0	不够减, 商上 0
←	11. 1 0 1 0	0 1 1 0 0	余数与商左移一位
+	00. 1 1 0 1		加 $Y$
	00. 0 1 1 1	0 1 1 0 1	够减, 商上 1



# 加减交替法流程图



# 加减交替法硬件设置



A、X、Q 均  $n+1$  位  
用  $Q_n$  控制加减交替



# 注意事项

---

做除法时注意：

- ①对定点小数除法，首先要比较除数和被除数的绝对值大小，检查是否出现商溢出。
- ②商的符号为相除两数符合半加和(逻辑异或)。
- ③运算过程中，放被除数和商的寄存器同时移动。

# 跳0跳1除法\*

---

提高除法运算速度的方法——跳0跳1除法(限原码)规则是:

(1)如果余数 $R \geq 0$ 且 $R$ 的高 $k$ 个数位均为0, 则本次上商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 $k$ 位个数位均为1, 则本次商为0, 后跟 $k-1$ 个1,  $R$ 左移 $k$ 位后, 加上除数 $Y$ , 得新的余数。

(3)不满足上述规则①和②中条件时, 按一位除法上商。



例：  $X = -0.1001$ ,  $Y = 0.1011$ , 求  $X \div Y$ 。

解：  $[-Y]_{\text{补}} = 11.0101$

被除数(余数)	商数	说明
0. 1 0 0 1		初始值
+ 1. 0 1 0 1		加 $[-Y]_{\text{补}}$
1. 1 1 1 0	0 1 1	余数 $< 0$ , 符号后有三个 1, 本次商为 011
← 1. 0 0 0 0		余数左移三位
+ 0. 1 0 1 1		加 $Y$
1. 1 0 1 1	0 1 1 0	余数 $< 0$ , 符号后有 1 个 1, 本次商为 0
← 1. 0 1 1 0		余数左移一位
+ 0. 1 0 1 1		加 $Y$
0. 0 0 0 1	0 1 1 0 1 0 0	余数 $> 0$ , 符号后有三个 0, 本次商为 100

商：  $-0.1101$ , 余数  $= 0.0001 \times 2^{-4}$

# 定点除法运算

---

- 笔算除法分析
- 原码一位除法
  - 恢复余数法
  - 加减交替法
  - 跳0跳1除法
- 补码一位除法



# 补码一位除法

补码除法的规则比原码除法的运算规则复杂。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判符号位

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值1的办法,并省略了最低位+1的操作,此时最大误差为 $\pm 2^{-n}$ 。

若要求商的精度较高,可按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,最低位不需加1。

例：

$$X = -0.1001,$$

$$Y = 0.1011,$$

求  $X \div Y$

解：

$$[X]_{\text{补}} = 11.0111,$$

$$[Y]_{\text{补}} = 00.1011,$$

$$[-Y]_{\text{补}} = 11.0101$$

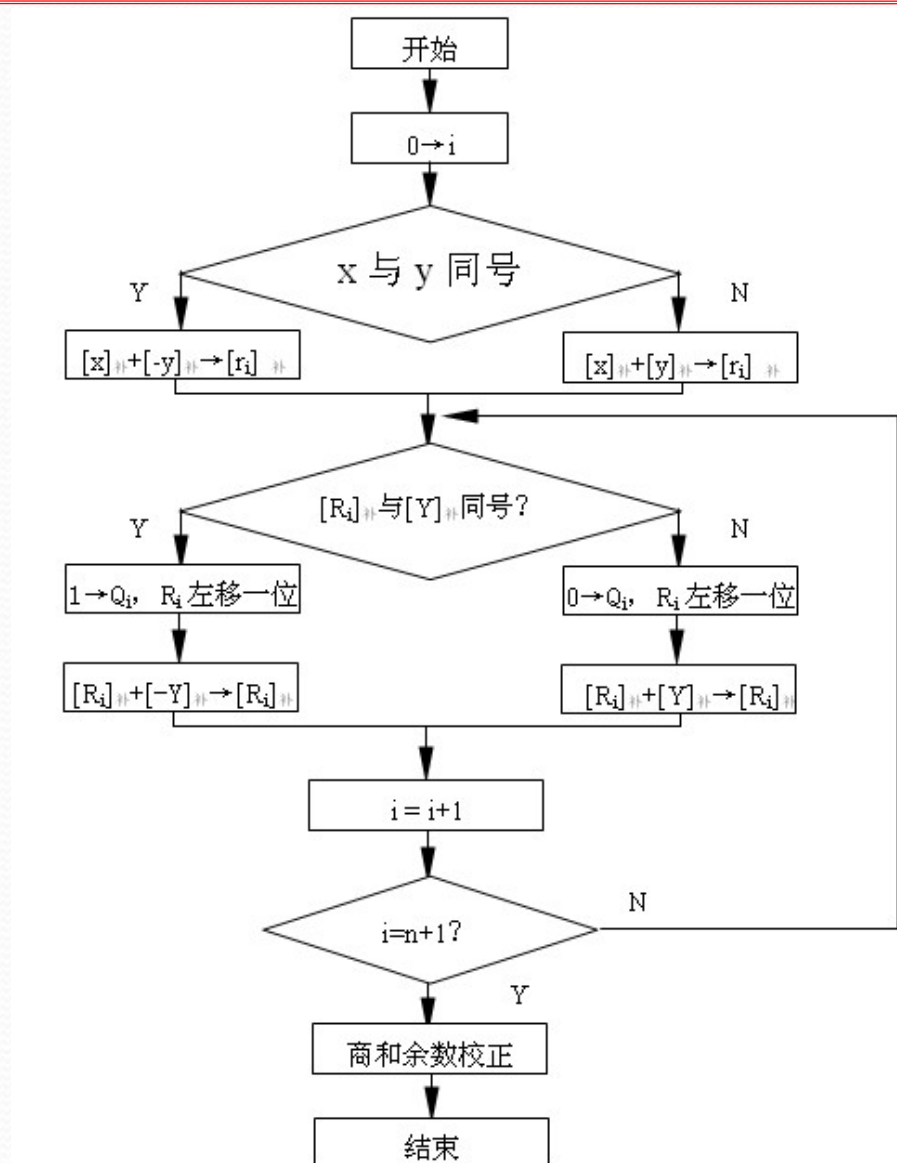
商： 1.0011

余：  $11.1111 \times 2^{-4}$

被除数(余数)										说明	
	11.	0	1	1	1	0	0	0	0	0	初始值
+	00.	1	0	1	1						异号，加 $[Y]_{\text{补}}$
	00.	0	0	1	0	0	0	0	0	1	余数与除数同号商上 1
←	00.	0	1	0	0	0	0	0	1	0	被除数与商左移一位
+	11.	0	1	0	1						加 $[-Y]_{\text{补}}$
	11.	1	0	0	1	0	0	0	1	0	余数与除数异号商上 0
←	11.	0	0	1	0	0	0	1	0	0	被除数与商左移一位
+	00.	1	0	1	1						加 $[Y]_{\text{补}}$
	11.	1	1	0	1	0	0	1	0	0	余数与除数异号商上 0
←	11.	1	0	1	0	0	1	0	0	0	被除数与商左移一位
+	00.	1	0	1	1						加 $[Y]_{\text{补}}$
	00.	0	1	0	1	0	1	0	0	1	余数与除数同号商上 1
←	00.	1	0	1	0	1	0	0	1	0	被除数与商左移一位
+	11.	0	1	0	1						加 $[-Y]_{\text{补}}$
	11.	1	1	1	1	1	0	0	1	0	余数与除数异号商上 0
						1	0	0	1	1	商为负，最低位加 1



# 补码一位除法流程图



# 补码一位除法运算规则

补码一位除法规则( $[R_i]_{补}$ 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
同号	0	减	同号(够减)	1	$2[R_i]_{补} - Y_{补}$
			异号(不够减)	0	$2[R_i]_{补} + Y_{补}$
异号	1	加	同号(不够减)	1	$2[R_i]_{补} - Y_{补}$
			异号(够减)	0	$2[R_i]_{补} + Y_{补}$



# 补码除和原码除比较

	原码除	补码除
商符	$x_0 \oplus y_0$	自然形成
操作数	绝对值补码	补码
上商原则	余数的正负	比较余数和除数的符号
上商次数	$n + 1$	$n + 1$
加法次数	$n + 1$	$n$
移位	逻辑左移	逻辑左移
移位次数	$n$	$n$
第一步操作	$[x^*]_{\text{补}} - [y^*]_{\text{补}}$	同号 $[x]_{\text{补}} - [y]_{\text{补}}$ 异号 $[x]_{\text{补}} + [y]_{\text{补}}$

# 第二章 运算方法与实现电路

---

- 2.1 数据的表示与数制之间的转换
- 2.2 机器数的编码表示
- 2.3 其他编码
- 2.4 定点数与浮点数
- 2.5 定点数运算方法
- 2.6 浮点数运算方法
- 2.7 数据校验码



# 浮点加减运算

---

设有两浮点数 $X$ ,  $Y$ , 实现 $X \pm Y$ 运算, 其中

$$X = M_X \cdot 2^{E_X} \quad Y = M_Y \cdot 2^{E_Y}$$

按以下5步来进行:

对阶

尾数加减运算

规格化

舍入

溢出判断与处理

# 浮点加减运算

## 1. 对阶

### (1) 求阶差

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

### (2) 对阶原则

小阶向大阶看齐



# 浮点加减运算

---

## 2. 尾数的加/减运算

在对阶以后，执行两尾数的加/减运算，得到两数之和/差。

## 3. 规格化操作

即当运算结果不是规格化数时要进行左规、右规。

**注意：**原码与补码表示时，规格化的不同。

理解其规则：双符号位的原码规格化尾数其值最高位为1；

补码规格化形式为： $00.1 \times \times \times \times$ ；

$11.0 \times \times \times \times$  ( $\times$ 表示0；1任意)

# 浮点加减运算

---

## 4. 舍入

在右规和对阶时，尾数低位上的数值会移掉，使其精度受到影响。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，若阶码正常，则加/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。



# 例题

---

例:  $X=2^{010} \cdot 0.11011011$ ,

$Y=2^{100} \cdot (-0.10101100)$ , 求  $X+Y$ 。

解:  $X$ ,  $Y$ 在机器内, 采用双符号位, 补码表示

	阶符	阶码	数符	尾数
X	00	010	00	11011011
Y	00	100	11	01010100

# 例题

过程如下：

## 1. “对阶”

$$\Delta E = [E_X]_{\text{补}} + [-E_Y]_{\text{补}} = 00,010 + 11,100 = 11,110$$

$X$ 阶码小， $M_X$ 右移2位 保留阶码 $E=00100$

$[M_X]_{\text{补}} = 00,00110110$ **11**，被移出去的。

## 2. 尾数相加

$$[M_X]_{\text{补}} + [M_Y]_{\text{补}} = 00.00110110$$
**11** + 11.01010100 = 11.10001010**11**

## 3. 规格化操作

左规：尾数左移1位，结果为11.00010101，阶码减1。

即  $E-1=00011$



# 例题

## 4. 舍入

附加位最高位为1，所以结果最低位加1，得新的结果 $[M]_{\text{补}} = 11.00010101$ ， $M = -0.11101011$  (0舍1入法，恒置1法)

## 5. 判溢出

阶码符号位为00，故不溢出，所以

$$X+Y=2^{011} \cdot (-0.11101011)$$

例：设 $x=2^{-101} \times (-0.101000)$ ， $y=2^{-100} \times (0.111011)$

设阶符取2位，阶码数值部分取3位，数符取2位，尾数数值部分取6位，计算 $x-y$ 。

(自行计算，结果 $(x-y)_{\text{补}} = 11,101; 11.011001$ )

# 浮点数的乘除法运算

---

- 浮点乘法运算步骤
- 浮点数的阶码运算
- 浮点数的舍入处理



# 浮点乘法的运算步骤

---

- (1) 求乘积的阶码。即两数阶码之和。
- (2) 尾数相乘。
- (3) 规格化处理。
- (4) 舍入（一般0舍1入法）。
- (5) 判断溢出（判断阶码是否溢出）。

# 浮点数的阶码运算

---

## (1) 移码运算

直接用移码实现求阶码之和时，结果最高位多加了个1，要得到移码形式的结果，需将结果符号位取反。

而根据补码—移码的关系又可得：

$[X+Y]_{\text{移}} = [X]_{\text{移}} + [Y]_{\text{补}}$ ，同理， $[X]_{\text{移}} + [-Y]_{\text{补}} = [X-Y]_{\text{移}}$ 。也就是说：执行移码加或减时，取加数或减数符号位反码进行运算。



# 浮点数的阶码运算

---

## (2) 判断溢出

使用双符号位的阶码加法器，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，则溢出条件是：

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。当最高符号位为0时，表明没有溢出，低位符号为1，表明结果为正，为0时，表明结果为负。即：

00：无溢出，结果为正

01：无溢出，结果为负

10：有溢出，上溢

11：有溢出，下溢

# 浮点数的舍入处理

---

尾数位数确定，而运算后，可能超出给定的位数。一般采用下面几种比较简单易于实现的处理方法。

(1) 截断处理（恒舍法）

(2) 舍入处理；

(3) 恒置“1”法

(4) 查表舍入法



# 第二章 运算方法与实现电路

---

- 2.1 数据的表示与数制之间的转换
- 2.2 机器数的编码表示
- 2.3 其他编码
- 2.4 定点数与浮点数
- 2.5 定点数运算方法
- 2.6 浮点数运算方法
- 2.7 数据校验码

# 数据校验码

---

数据校验码是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。

它的实现原理是加入一些冗余码，使合法数据编码出现某些错误时，就成为非法编码。由此就可以通过检测编码的合法性来达到发现错误的目的。

**码距**是根据任意两个合法码之间至少有几个二进制位不同而确定的。

**码距为1的编码不具有发现错误的能力**



# 数据校验码

---

- 奇偶校验码
- 海明校验码
- 循环冗余校验码

# 奇偶校验码

---

奇偶校验码是应用最广泛的一种检错码，是一种开销最小，能发现数据代码中一位出错情况的编码。它的实现原理是：使得码距由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，这个码都将变成非法编码。

采用这种方案只能发现一位错或奇数个位错，但是不能确定是哪一位错，也不能发现偶数个位错，不能纠错。



# 奇偶校验码

下面以实例来说明8位数据的奇偶校验编码

数 据	奇校验码的编码	偶校验码的编码
00001010	100001010	000001010
00010110	000010110	100010110
10001101	110001101	010001101

其中校验码中的最高位为校验位，其余8位为数据位。从中可以看到，校验位的值取1或0取决于数据位中1的个数。

# 数据校验码

---

- 奇偶校验码
- 海明校验码
- 循环冗余校验码



# 海明校验码

---

海明校验码是由Richard Hamming于1950年提出的，到目前还被广泛应用。它的实现原理是在数据中加入几个校验位，并将数据的每一个二进制位分配在几个奇偶校验组中。当一位出错后，就会引起有关的几个校验组的值发生变化，这样，不仅可以发现错误，还能指出是哪一位出错，为自动纠错提供了依据。

# 海明校验码

假设信息位为 $k$ 位，  
校验位为 $n$ 位，如果能检测并能自动校正一位错，并发现两位错，此时校验位的位数 $n$ 和数据位的位数 $k$ 应满足关系：

$$2^{n-1} \geq n + k$$

数据位  $k$  与校验位  $n$  的对应关系

$k$ 值	最小的 $n$ 值
1~4	4
5~11	5
12~26	6
27~57	7
58~120	8



# 编码规则

---

海明码的编码规则如下：假设海明码的最高位号为 $m$ ，最低位号为 $1$ ，即 $H_m H_{m-1} \cdots H_2 H_1$ 。

- 校验位与数据位之和为 $m$ ，每个校验位 $P_i$ 在海明码中被分在位号 $2^{i-1}$ 的位置，其余各位为数据位，并按从低到高逐位依次排列的关系分配各数据位。
- 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。这样做是希望校验的结果能正确反映出出错的位号。

# 编码规则

- 下面讨论一个字节的海明码的编码：每个字节由8个二进制位组成，此时的 $n=8$ ，按上表求出校验位的位数 $k$ 应该等于5，所以海明码的总位数为 $8+5=13$ ，可表示为：

$$H_{13}H_{12}H_{11}\cdots H_3H_2H_1$$

- 5个校验位 $P_5\sim P_1$ 对应的海明码位号应分别是 $H_{13}$ ， $H_8$ ， $H_4$ ， $H_2$ ， $H_1$ 。由于海明码的最高位是 $H_{13}$ ，所以 $P_5$ 只能放在此位，其余的4位满足 $P_i$ 在海明码中被分在位号 $2^{i-1}$ 的位置的关系。海明码中其余位是数据位 $D_i$ ，则有如下的排列关系：

$$P_5D_8D_7D_6D_5P_4D_4D_3D_2P_3D_1P_2P_1$$



# 海明校验码

海明码位号与校验位位号的关系

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 <sub>1</sub>	P <sub>1</sub>	1	1=1
H <sub>2</sub>	P <sub>2</sub>	2	2=2
H <sub>3</sub>	D <sub>1</sub>	1, 2	3=1+2
H <sub>4</sub>	P <sub>3</sub>	4	4=4
H <sub>5</sub>	D <sub>2</sub>	1, 4	5=1+4
H <sub>6</sub>	D <sub>3</sub>	2, 4	6=2+4
H <sub>7</sub>	D <sub>4</sub>	1, 2, 4	7=1+2+4
H <sub>8</sub>	P <sub>4</sub>	8	8=8
H <sub>9</sub>	D <sub>5</sub>	1, 8	9=1+8
H <sub>10</sub>	D <sub>6</sub>	2, 8	10=2+8
H <sub>11</sub>	D <sub>7</sub>	1, 2, 8	11=1+2+8
H <sub>12</sub>	D <sub>8</sub>	4, 8	12=4+8
H <sub>13</sub>	P <sub>5</sub>	13	13=13

# 海明校验码

---

- 校验位的形成:

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

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

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

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

- 上式中 $D_4$  $D_7$ 出现了3次, 其余为2次, 造成码距不相等。为此, 再补充一个总校验位 $P_5$ 使得码距为4:

$$P_5 = D_1 \oplus D_2 \oplus D_3 \oplus D_5 \oplus D_6 \oplus D_8$$



# 校验

---

接收端将收到的海明码按如下关系进行偶校验：

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

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

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

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

$$S_5 = P_5 \oplus D_1 \oplus D_2 \oplus D_3 \oplus D_5 \oplus D_6 \oplus D_8$$

- 校验结果为  $S_5 \sim S_1$

# 校验规则

- 当 $S_5 \sim S_1$ 为00000时，表明没有错误
- 当 $S_5 \sim S_1$ 中仅有1位不为0，则表明某一位出错或三位同时出错。认为出错位是 $S_i$ 对应的 $P_i$ 位
- 当 $S_5 \sim S_1$ 中有2位不为0，表明两位海明码出错，只能发现，无法确定具体位
- 当 $S_5 \sim S_1$ 中有3位不为0，则表明某一位出错或三位同时出错。认为出错位由 $S_4 \sim S_1$ 确定
- 当 $S_5 \sim S_1$ 中有4/5位不为0，则表明系统出错情况严重，应检查硬件的正确性。



## 出错情况表

### 海明码出错情况表

[illegible]

# 数据校验码

---

- 奇偶校验码
- 海明校验码
- 循环冗余校验码



# 模2运算

CRC码是基于模2运算建立的校验码制。

1、模2运算(不考虑进位和借位的运算)

(1) 模2加和模2减结果相同

$0 \pm 0 = 0$ ;  $0 \pm 1 = 1$ ;  $1 \pm 0 = 1$ ;  $1 \pm 1 = 0$ ;

两个数相同结果为0, 两个数不同结果为1,  
可用异或实现.

(3) 模2除是按模2减求部分余数

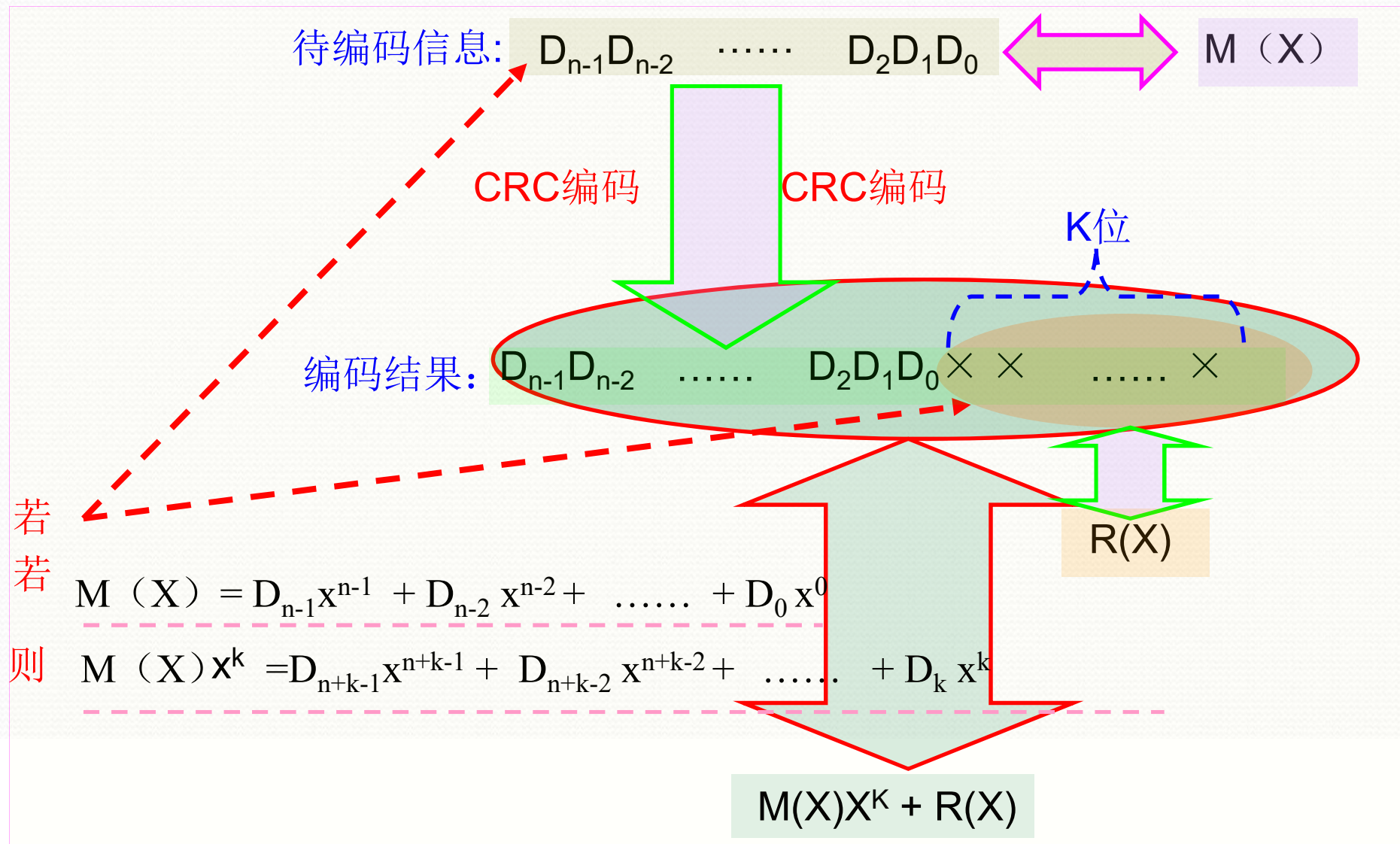
例: 
$$\begin{array}{r} 1101 \overline{) 101011} \\ \underline{1101} \phantom{00} \\ 1111 \\ \underline{1101} \phantom{00} \\ 0101 \\ \underline{0000} \phantom{00} \\ 101 \end{array}$$

(2) 模2乘是按模2和求部分积之和

例: 
$$\begin{array}{r} 1101 \\ \times 110 \\ \hline 0000 \\ 1101 \\ + 1101 \\ \hline 101110 \end{array}$$

原则: ①部分余数首位为1, 上商1;  
部分余数首位为0, 上商0;  
②部分余数位数小于除数位数时,  
即为最后余数.

# 编码方法





# 校验方法

现在我们已经得到了一个事实：编码结果 =  $M(X)X^K + R(X)$

如何将  $M(X)X^K + R(X)$  表示成一个多项式的乘积形式呢？

若  $M(X)X^K = Q(X)G(X) + R(X)$

则利用模2运算

问题得到解决

因为：  $M(X)X^K + R(X) = Q(X)G(X) + R(X) + R(X)$

所以：  $M(X)X^K + R(X) = Q(X)G(X)$

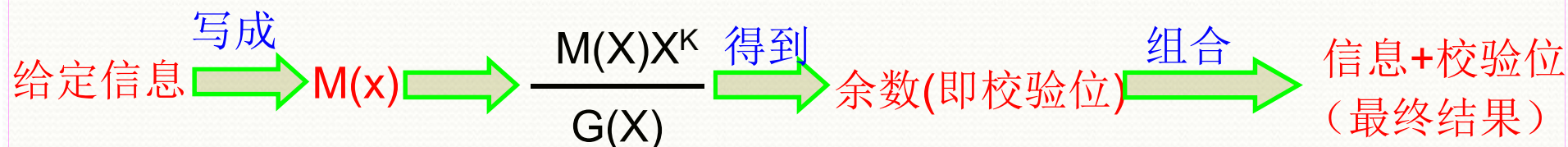
结论：  $M(X)X^K + R(X)$  可被给定的多项式  $G(X)$  除尽。

$$\text{或 } \frac{M(X)X^K}{G(X)} = \frac{Q(X)G(X)}{G(X)} + \frac{R(X)}{G(X)} = Q(X) + \frac{R(X)}{G(X)}$$

# 校验方法

精髓:

CRC码就是用多项式 $M(x)x^k$ 除以给定的生成多项式 $G(X)$ ,所得余数作为校验位。为了得到 $k$ 为余数(即校验位),  $G(x)$ 必须是 $k+1$ 位。



特点:

- ① CRC码是一个能被生成多项式 $G(X)$ 模2整除的信息码组, 利用该特点可以进行检错。
- ② 发送时双方事先约定生成多项式 $G(X)$ (被除数)。
- ③ CRC校验实现简单, 检错能力强, 被广泛应用在各种数据校验应用中, 占用系统资源少, 用软硬件均能实现。是进行数据传输差错检测的一种很好手段。



# 应用举例

olor

XML

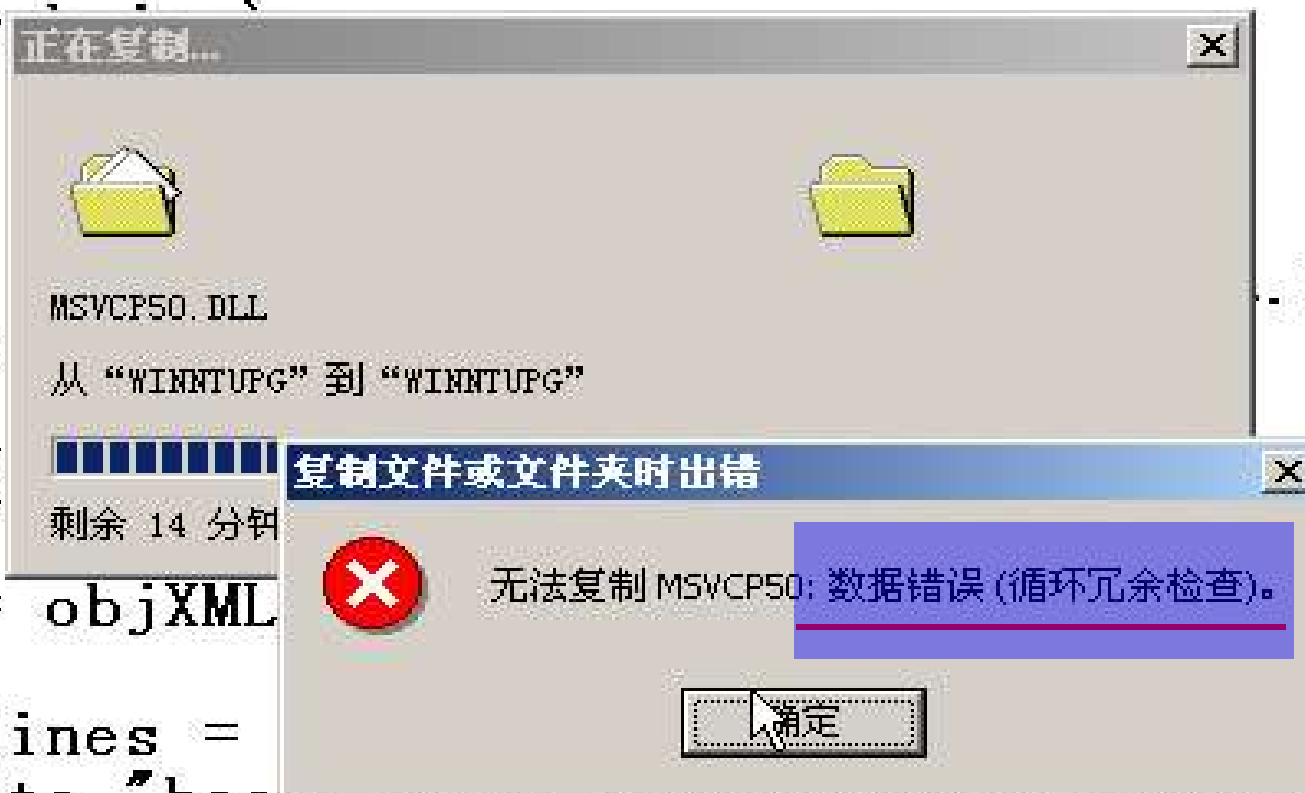
sync

oad(

st = objXML

eadlines =

.Write "headlines. &#160;001headlines"



图：CRC应用

# 例题

例：有效信息为1100，试用生成多项式 $G(X) = X^3 + X + 1 = 1011$ ，编制CRC码。

解：由  $G(X) = X^3 + X + 1 = 1011$ ，得 $K=3$

$$M(X) = X^3 + X^2 = 1100$$

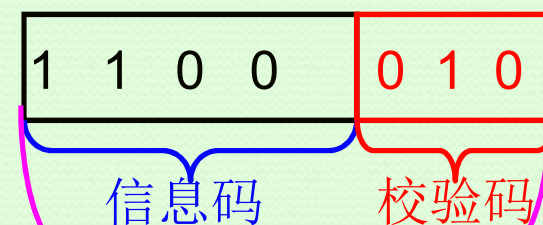
$$M(X) X^3 = X^6 + X^5 = 1100000$$

将 $M(x)X^3$ 模2除 $G(X)$ ，有

A long division diagram showing the division of 1100000 by 1011. The divisor 1011 is written below the dividend 1100000. The quotient is 1110, and the remainder is 010. The steps are as follows:

1100000	
1011	
---	
1110	
1011	
---	
1010	
1011	
---	
0010	
0000	
---	
010	

则 CRC 码为：



称为 (7, 4) 码制



# 校 验

(7, 4) 码校验情况表 ( $G(x)=1011$ )

	$A_1$	$A_2$	$A_3$	$A_4$	$A_5$	$A_6$	$A_7$	余数	出错位
正确码	1	1	0	0	0	1	0	000	无
错误码	1	1	0	0	0	1	1	001	7
	1	1	0	0	0	0	0	010	6
	1	1	0	0	1	1	0	100	5
	1	1	0	1	0	1	0	011	4
	1	1	1	0	0	1	0	110	3
	1	0	0	0	0	1	0	111	2
	0	1	0	0	0	1	0	101	1

# 检错与纠错

---

## (1) 检错

接收方将收到的CRC码模2除 $G(x)$ ,若传输无误,则余数为0;

- 如果某一位出错,则余数不为0
- 不同的出错位,其余数也不同。

## (2) 纠错

- ① 出错位与余数有一定的对应关系,该对应关系与码制及 $G(x)$ 有关系。
- ② 纠错过程: 某位出错,余数不为0,将余数补0继续作模2除,得到下一信息位的出错余数。即出错信息位对应的余数构成一个循环——“循环码”



# 生成多项式的选择

---

生成多项式应满足以下要求：

- 任何一位发生错误，都必须使余数不为0；
- 不同的位出现错误，应使余数也不同；
- 对余数继续作模2运算，应使得余数循环。

计算机和通信系统中广泛使用下述两个生成多项式：

$$G(x) = x^{16} + x^{15} + x^2 + 1$$

$$G(x) = x^{16} + x^{12} + x^6 + 1$$

# 小结

---

## (1) 数据表示

原码、反码、补码、移码，定点数与浮点数表示及表示范围，IEEE 754标准

## (2) 定点数运算

补码加减法，乘法(原码一位乘，Booth，两位乘法\*)，除法(原码一位除法，加减交替法)

## (3) 浮点数运算

## (4) 数据校验码



# 作业

---

- P.289-292: 6.20(原码1位乘)、6.21(原码)、6.26、6.28(补码定点除法不做)、6.31。
- 补充: 有一个(7, 3)码, 生成多项式为  $G(x)=x^4 +x^3 +x^2+1$ , 写出代码001的校验码和循环余数。

谢谢！