



计算方法

大连理工大学 赖晓晨



数的原码表示法

大连理工大学 赖晓晨

一、关于数制

1. 十进制的起源
2. 不同数制的区别

一、关于数制

1. 十进制的起源
2. 不同数制的区别
3. 喵星人的八进制



八进制

0

4

10

十进制

0

4

8

二、无符号数和有符号数

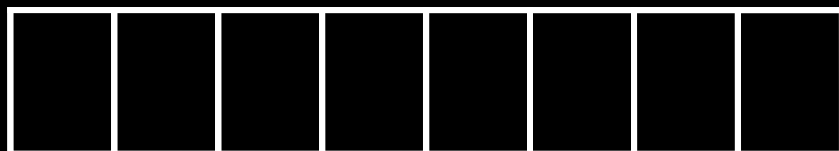
1. 数在运算时，均存放在寄存器中，寄存器的位数即机器字长。

2. 计算机中参与运算的有两类数：

- 无符号数
- 有符号数

无符号数

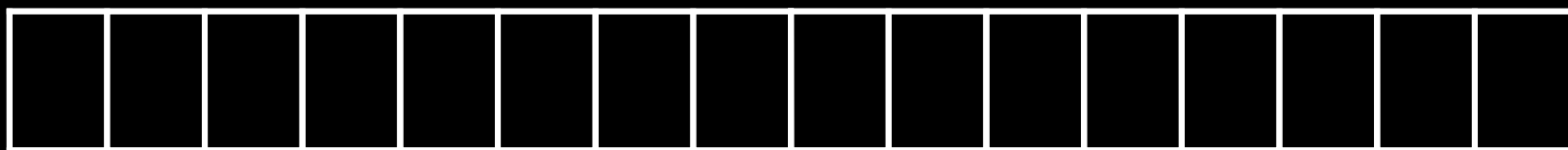
没有符号的数，寄存器的每一位存放的都是数值。



8位

0~255

unsigned char



16 位

0 ~ 65535

unsigned int

有符号数

- 数字的极性也用二进制数表示，0代表正数，1代表负数，即符号位也被数字化了，符号位放在数字的前端。

signed int

- 这种把符号“数字化”的数，叫做**机器数**，而数字原本的（带有正负号的）值称为**真值**。

机器数和真值

真值

带符号的数

+ 0.1011

- 0.1011

+ 1100

- 1100

机器数

符号数字化的数

0 | 1011

1 | 1011

0 | 1100

1 | 1100

小数点的位置

小数点的位置

小数点的位置

小数点的位置

机器数的编码形式

➤ 原码

➤ 反码

➤ 补码

➤ 移码

三、原码表示法

- 符号位为0表示正数，为1表示负数，数值部分为真值的绝对值，又称为带符号的绝对值。
- 约定书写时，整数的符号位与数值位用逗号分隔，小数的符号位与数值位用小数点分隔。

1. 原码的定义

整数

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

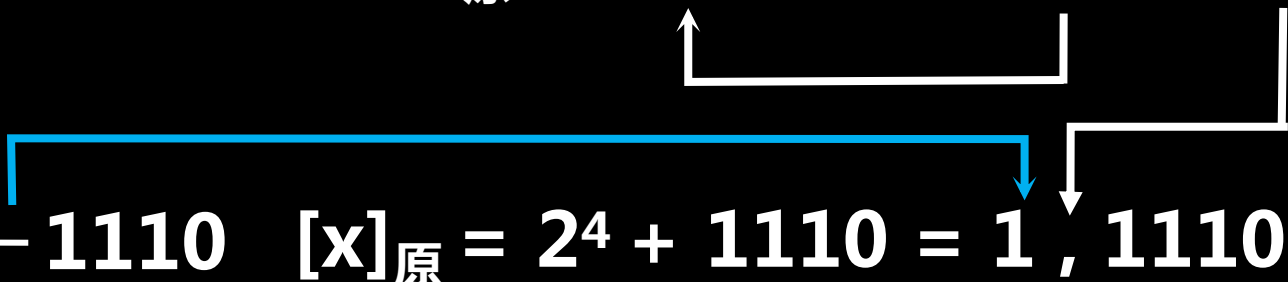
x 为真值 n 为整数的位数

用 **逗号** 将符号位和数值部分隔开

如 $x = +1110$ $[x]_{\text{原}} = 0, 1110$



$x = -1110$ $[x]_{\text{原}} = 2^4 + 1110 = 1, 1110$



1. 原码的定义

小数

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

如

用 **小数点** 将符号位和数值部分隔开

$x = +0.1101 \quad [x]_{\text{原}} = 0.1101$


$x = -0.1101 \quad [x]_{\text{原}} = 1 - (-0.1101) = 1.1101$

例题1

例：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：已知 $[x]_{\text{原}} = 1,1100$ 求 x 

解：由定义得

$$x = 2^4 - [x]_{\text{原}} = 10000 - 1,1100 = -1100$$

例题2

例：已知 $[x]_{\text{原}} = 0.1101$ 求 x

解：根据定义 $\because [x]_{\text{原}} = 0.1101$

$$\therefore x = +0.1101$$

例：求 $x = 0$ 的原码

解：设 $x = +0.0000$ $[+0.0000]_{\text{原}} = 0.0000$

$x = -0.0000$ $[-0.0000]_{\text{原}} = 1.0000$

同理，对于整数 $[+0]_{\text{原}} = 0,0000$

$[-0]_{\text{原}} = 1,0000$

$$\therefore [+0]_{\text{原}} \neq [-0]_{\text{原}}$$

2. 原码表示法总结

1. 优点：表示方法简单，直观。

2. 缺点：加减运算复杂

- 判断正负
- 确定位置
- 判断结果符号
- 电路复杂、不统一

3. 能否把减法转换为加法呢？



数的反码表示法

大连理工大学 赖晓晨

反码表示法

➤ 反码经常用作求补码的过渡。定义如下：

➤ 整数

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

x 为真值。 n 为整数的位数。

如 $x = +1101$ $x = -1101$

$$[x]_{\text{反}} = 0,1101 \quad [x]_{\text{反}} = (2^{4+1} - 1) - 1101$$

$$= 11111 - 1101$$

$$= 1,0010$$

用逗号将符号位
和数值部分隔开

反码表示法

➤ 小数

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

x 为真值。 n 为小数的位数。

如 $x = +0.1101$ $x = -0.1010$

$$\begin{aligned} [x]_{\text{反}} &= 0.1101 & [x]_{\text{反}} &= (2-2^{-4}) - 0.1010 \\ & & &= 1.1111 - 0.1010 \\ & & &= 1.0101 \end{aligned}$$

用**小数点**将符号位
和数值部分隔开



例题1

例：已知 $[x]_{\text{反}} = 0,1110$ 求 x 。

解：由定义得 $x = + 1110$

例：已知 $[x]_{\text{反}} = 1,1110$ 求 x 。

解：由定义得 $x = [x]_{\text{反}} - (2^{4+1}-1)$
 $= 1,1110 - 11111$
 $= - 0001$

例题2

例：求0的反码。

解：设 $x = + 0.0000$ $[+0.0000]_{\text{反}} = 0.0000$

$x = - 0.0000$ $[-0.0000]_{\text{反}} = 1.1111$

同理，对于整数

$[+0]_{\text{反}} = 0,0000$ $[-0]_{\text{反}} = 1,1111$

$\therefore [+0]_{\text{反}} \neq [-0]_{\text{反}}$

推荐阅读：ASC II

- ASCII（美国信息交换标准代码）是基于拉丁字母的一套电脑编码系统，主要用于显示现代英语。是现今最通用的单字节编码系统，等同于国际标准ISO/IEC 646。
- 最大缺点是只能显示26个基本拉丁字母、阿拉伯数字和英式标点符号，因此只能用于显示现代美国英语。
- 如果表示汉字，需要两个字节表示一个汉字，那么，如何区分当前的两个字节是两个ASCII码，还是一个汉字呢？

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	8	9	10	11	12	13	14	15	16	17	18	19	20	21
		+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	ctrl
0000	0	0	BLANK	^@	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	←	^[ESC	转意	43	+	59	;	75	K	91	[107	k	123	{
1100	C	12	♀	^L	FF	换页/新页	28	└	^\	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	F	15	☼	^O	SI	移入	31	▼	^~	US	单元分隔符	47	/	63	?	79	O	95	_	111	o	127	Δ

注：表中的ASCII字符可以用:ALT + “小键盘上的数字键”输入



BCD码表示法（自学）

大连理工大学 赖晓晨

BCD码的用途

- 计算机的输入输出部分绝大多数情况使用十进制数据，而计算机内部使用二进制，因此涉及到数制转换。
- 某些数据输入输出量很大的商用领域，为减少大量的数制转换，直接采用十进制进行计算。
- BCD (Binary Coded Decimal) 码为采用二进制编码的十进制数，某些计算机内部有专门的十进制数运算指令，以及对应的逻辑线路。

BCD码的分类

- **有权BCD码**：每个十进制数位的4个二进制数位，都有一个确定的权。

8421BCD码（自然BCD码）、2421BCD码、5211BCD码、84-2-1BCD码、4311BCD码

- **无权BCD码**：每个十进制数位的4个二进制数位，没有确定的权。
- **余3码、格雷码**

有权BCD码

十进制数	8421码	2421码	5211码	84-2-1码	4311码
0	0000	0000	0000	0000	0000
1	0001	0001	0001	0111	0001
2	0010	0010	0011	0110	0011
3	0011	0011	0101	0101	0100
4	0100	0100	0111	0100	1000
5	0101	1011	1000	1011	0111
6	0110	1100	1010	1010	1011
7	0111	1101	1100	1001	1100
8	1000	1110	1110	1000	1110
9	1001	1111	1111	1111	1111

无权BCD码

十进制数	余3码	格雷码(1)	格雷码(2)
0	0011	0000	0000
1	0100	0001	0100
2	0101	0011	0110
3	0110	0010	0010
4	0111	0110	1010
5	1000	1110	1011
6	1001	1010	0011
7	1010	1000	0001
8	1011	1100	1001
9	1100	0100	1000

8421BCD码的运算

- 如果两个BCD码相加之和小于等于 $(9)_{10}$, 则不需修正 ; 如相加之和大于 $(9)_{10}$, 则要进行加6修正 , 并向高位进位。

- 例题1 :

$$1+8=9;$$

$$4+9=13;$$

$$9+7=16$$

- 例题2 :

$$(28)_{10} + (55)_{10} = (83)_{10}$$

例题1

例：1+8=9;

$$\begin{array}{r} 0001 \\ + 1000 \\ \hline 1001 \end{array}$$

不需要修正

例：4+9=13;

$$\begin{array}{r} 0100 \\ + 1001 \\ \hline 1101 \\ + 0110 \text{ 修正} \\ \hline \underline{1}0011 \end{array}$$

进位

例：9+7=16

$$\begin{array}{r} 1001 \\ + 0111 \\ \hline \underline{1}0000 \\ + 0110 \text{ 修正} \\ \hline \underline{1}0110 \end{array}$$

进位

余三码及运算

- 余三码是在BCD码的基础上，每个编码都加0011而形成的。
- 当两个余三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.

例题2

例： $(28)_{10} + (55)_{10} = (83)_{10}$

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

$(28)_{10}$

$(55)_{10}$

低位向高位产生进位，
高位不产生进位。

低位+3，高位-3。
 $(0011)_2 = (3)_{10}$

自学：BCD码的分类和运算

- BCD (Binary Coded Decimal) 码为采用二进制编码的十进制数，某些计算机内部有专门的十进制数运算指令，以及对应的逻辑线路。
- 有权BCD码：每个十进制数位的4个二进制数位，都有一个确定的权。
8421BCD码（自然BCD码）、2421BCD码、5211BCD码、84-2-1BCD码、4311BCD码
- 无权BCD码：每个十进制数位的4个二进制数位，没有确定的权。
余3码、格雷码
- BCD码的运算规则

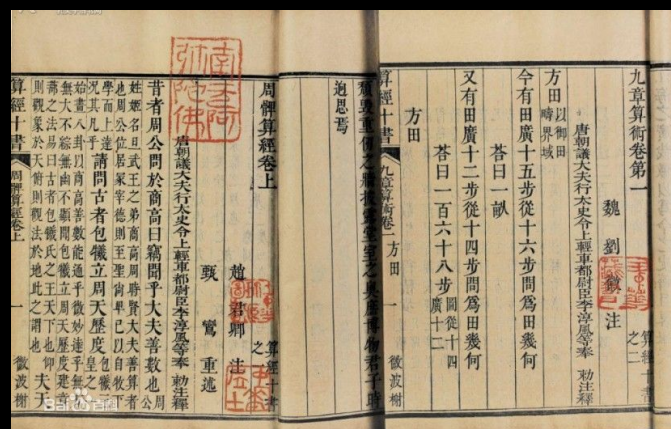


数的补码表示法

大连理工大学 赖晓晨

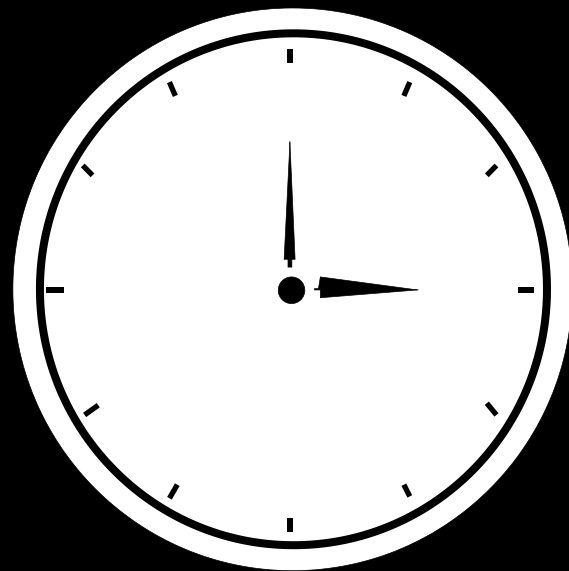
负数的由来

- 三国时期学者刘徽首先给出了正负数的定义，他说：“今两算得失相反，要令正负以名之。”
- 《九章算术》（成书于公元一世纪）中，最早提出了正负数加减法的法则。
- 与中国古代数学家不同，西方数学家更多的是研究负数存在的合理性。16、17世纪欧洲大多数数学家不承认负数是数。
- 19世纪整数理论基础的建立，负数在逻辑上的合理性才真正建立。



一、补码表示法

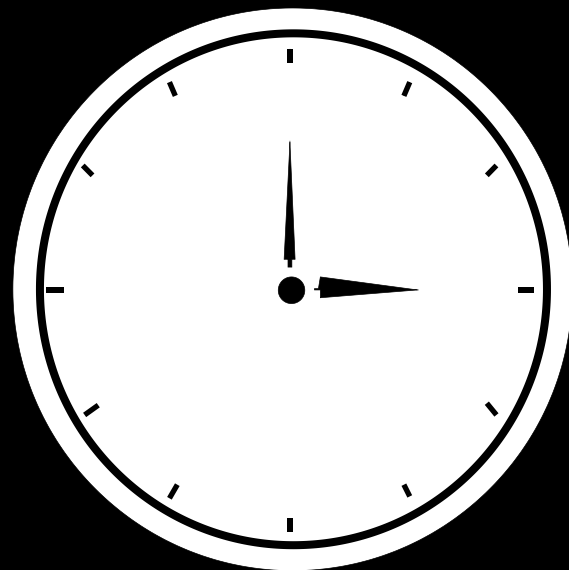
- 顺时针转3格，与逆时针转 $(12-3)$ 格的结果一致，与顺时针转 $(12+3)$ 格的结果也一致。
- 12是一个特殊的数字，是会被“自动丢弃”的，这个数字实际是钟表的“模”，而时间值需要对12求模。
- 3和-9这一对数，对于钟表的意义是相同的，这一对数，互称为以12为模的补数，+3可以用-9代替，同理-3可用+9代替。



进一步讨论

减去一个数，可以用加上这个数的补数来代替，从而可把减法转变为加法。归纳出以下结论：

- 一个负数可用其正的补数来代替，这个正补数可以用此负数加模求得。
- 一个正数和负数互为补数时，他们的绝对值之和即为模数。
- 正数的补数即该正数本身。
- 互为补数。



看一个二进制的例子

用二进制数计算6-3的值。

$$6 - 3 = [[6-3]]_{\text{补}}$$

$$= [[6] + [-3]]_{\text{补}}$$

$$= [0,110 + 1,101]_{\text{补}}$$

$$= [10,011]_{\text{补}}$$

$$= [0,011]_{\text{补}}$$

$$= +3$$

二、补码的定义

整数

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

x 为真值 n 为整数的位数（不包括符号位）

如 $x = +1010$

$$[x]_{\text{补}} = 0,1010$$

$x = -1011000$

$$[x]_{\text{补}} = 2^{7+1} + (-1011000)$$

$$= 100000000$$

$$\underline{-1011000}$$

$$1,0101000$$

用逗号将符号位
和数值部分隔开

二、补码的定义

小数

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

如 $x = +0.1110$ $x = -0.1100000$

$$[x]_{\text{补}} = 0.1110 \quad [x]_{\text{补}} = 2 + (-0.1100000)$$

$$= 10.0000000$$

$$- 0.1100000$$

$$1.0100000$$

用小数点将符号位
和数值部分隔开

三、求补码的方法

- 定义法
- (负数的补码) “反码+1”法
- (负数的补码) 符号位不变, 数值位从左到右依次取反, 最右边的1及其后的0不变。

$$x = -0.1100100$$

$$x = 1.0011100$$

- 补码的补码是原码

例题1

例：已知 $[x]_{\text{补}} = 0.0001$ 求 x 。

解：由定义得 $x = + 0.0001$

例：已知 $[x]_{\text{补}} = 1.0001$ 求 x 。

解：由定义得

$$\begin{aligned} x &= [x]_{\text{补}} - 2 \\ &= 1.0001 - 10.0000 \\ &= - 0.1111 \end{aligned}$$

例题2

练习：求下列真值的原码和补码

真值	$[x]_{\text{补}}$	$[x]_{\text{原}}$
$x = 0.0000$	0.0000	0.0000
$x = -0.0000$	0.0000	1.0000
$x = -1.0000$	1.0000	不能表示

$$[+0]_{\text{补}} = [-0]_{\text{补}}$$

例题3 各种机器数的数值表示

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	± 0	+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

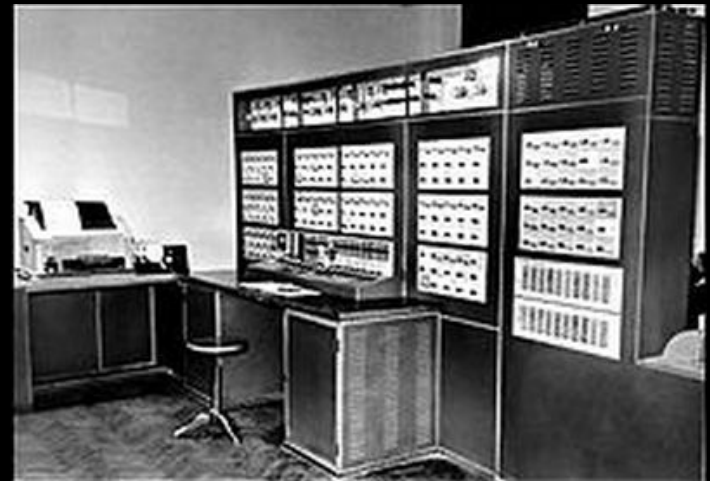


数的移码表示法

大连理工大学 赖晓晨

Сетунь——三进制计算机

- 三进制是以3为基数的进制，用正电压（真）、零电压（不知道）、负电压（假）表示。
- 三进制逻辑电路比二进制逻辑电路速度更快、可靠性更高，需要的设备和电能也更少。
- Сетунь是莫斯科国立大学研究员设计的第一批三进制计算机，1958年2月建成。
- Сетунь是一台带有快速乘法器的时序计算机。



根据数学极限原理，最优为 e 进制，最接近的整数为3，次接近的为2。

一、移码表示法的作用

补码表示很难直接判断其真值大小。

如	十进制	二进制	补码
	$x = +21$	$+10101$	$0,10101$
	$x = -21$	-10101	$1,01011$
	$x = +31$	$+11111$	$0,11111$
	$x = -31$	-11111	$1,00001$

一、移码表示法的作用

补码表示很难直接判断其真值大小。

如	十进制	二进制	补码
	$x = +21$	$+10101$	$0,10101$
	$x = -21$	-10101	$1,01011$
	$x = +31$	$+11111$	$0,11111$
	$x = -31$	-11111	$1,00001$

$x + 2^5$

$x = +21$	$+10101 + 100000 = 110101$
$x = -21$	$-10101 + 100000 = 001011$
$x = +31$	$+11111 + 100000 = 111111$
$x = -31$	$-11111 + 100000 = 000001$

二、移码的定义

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

x 为真值， n 为 整数的位数。

三、移码与补码的比较

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

x 为真值, n 为 整数的位数。

补码的定义

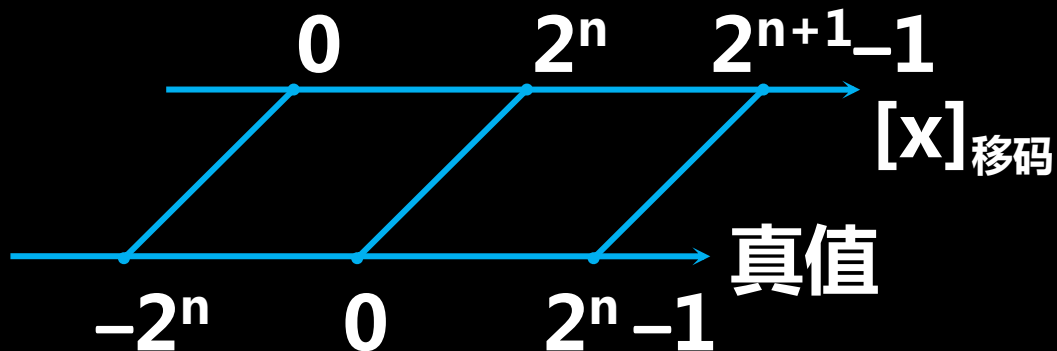
$$[x]_{\text{补}} = \begin{cases} 0, & x \geq 0 \\ 2^{n+1} + x, & -2^n \leq x < 0 \end{cases} \pmod{2^{n+1}}$$

x 为真值 n 为整数的位数 (不包括符号位)

移码与补码的数值部分相同, 符号位相反。

四、移码在数轴上的表示

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



如 $x = 10100$

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

$x = -10100$

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

用逗号将符号位
和数值部分隔开

五、移码的特点

$$\text{当 } x = 0 \text{ 时} \quad [+0]_{\text{移}} = 2^5 + 0 = 1,00000$$

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

$$\therefore [+0]_{\text{移}} = [-0]_{\text{移}}$$

$$\text{当 } n = 5 \text{ 时} \quad \text{最小的真值为 } -2^5 = -100000$$

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

可见，最小真值的移码为全0。用移码表示浮点数的阶码能方便地判断浮点数的阶码大小。

六、各种机器数的数值表示

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	± 0	+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

推荐阅读：Unicode

- **Unicode（统一码、万国码、单一码、标准万国码）是国际组织制定的可以容纳世界上所有文字和符号的字符编码方案；**
- **Unicode 为每种语言中的每个字符设定了统一并且唯一的二进制编码，以满足跨语言、跨平台进行文本处理的要求；**
- **历史上存在两个独立的尝试创立单一字符集的组织：ISO和统一码联盟；**
- **UTF-8是互联网上使用最广的一种Unicode实现方式，其他实现方式还有UTF-16和UTF-32。**





定点数与浮点数

大连理工大学 赖晓晨

Ariana 5火箭的坠落

- 1996年6月4日,由欧洲12国联合研制的"阿丽亚娜5型"(Ariane)运载火箭,在法属圭亚那库鲁航天中心首次发射。火箭升空约40秒时,在3500米以上高度发生爆炸,星箭俱毁。
- 火箭的水平速率比阿丽亚娜4型提高了五倍,但是64位浮点数向16位整数转换的程序沿用了阿丽亚娜4型的程序,导致溢出



一、数的定点表示

小数点按约定方式标出



定点机

小数

整数

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 $-(1 - 2^{-n}) \sim +(1 - 2^{-n})$

$-(2^n - 1) \sim +(2^n - 1)$

二、数的浮点表示

- 浮点数的一般形式 $N = S \times r^j$
- S 尾数 j 阶码 r 基数 (基值)

计算机中 r 取 2、4、8、16 等。

当 $r = 2$

$$N = 11.0101$$

$$\begin{aligned} \checkmark &= 0.110101 \times 2^{10} \\ &= 1.10101 \times 2^1 \\ &= 1101.01 \times 2^{-10} \\ &= 0.00110101 \times 2^{100} \end{aligned}$$

二进制表示

规格化数

S 小数、可正可负, j 整数、可正可负

浮点数在机器中的表示形式



S_f 代表浮点数的符号

n 其位数反映浮点数的精度

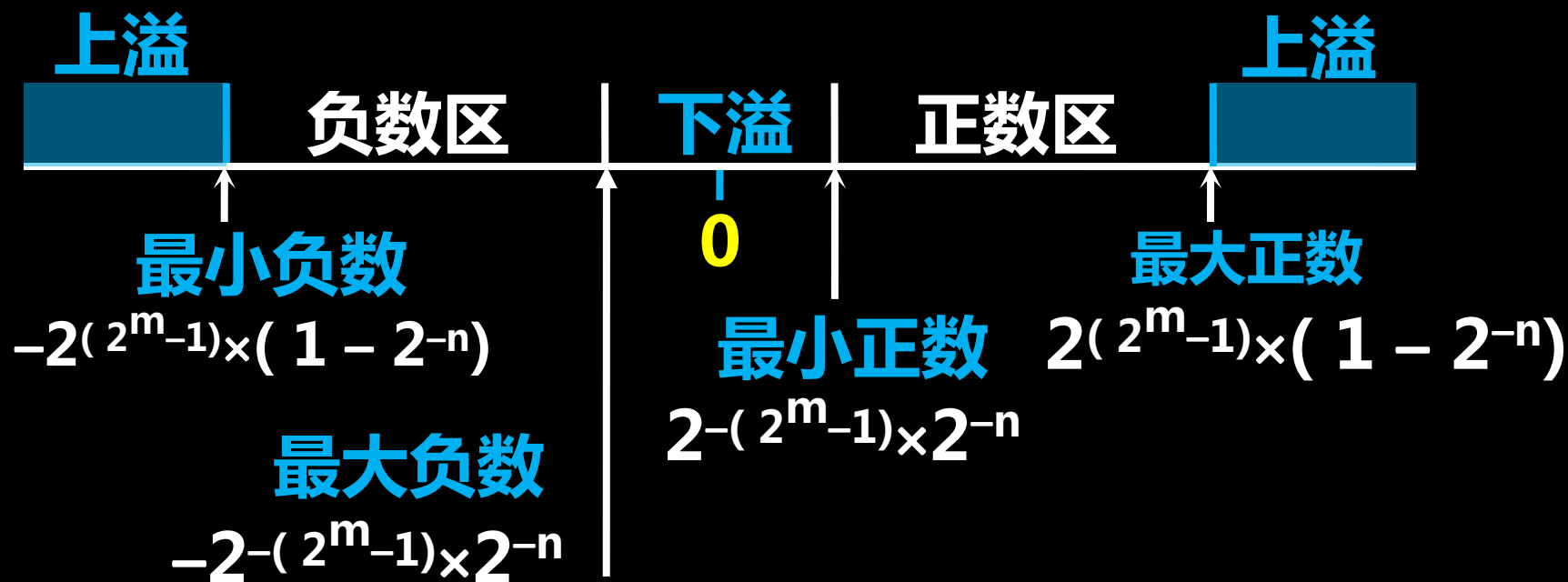
m 其位数反映浮点数的表示范围

j_f 和 m 共同表示小数点的实际位置

浮点数的表示范围

上溢 阶码 > 最大阶码 中断溢出处理

下溢 阶码 < 最小阶码 按机器零处理



例题1

练习：设机器数字长为 24 位，欲表示 ± 3 万的十进制数，试问在保证数的最大精度的前提下，除阶符、数符各取1位外，阶码、尾数各取几位？

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

\therefore **指数为15** 可反映 ± 3 万之间的十进制数

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

$m = 4, 5, 6,$

满足 **最大精度** 可取 **$m = 4, n = 18$**

浮点数的规格化表示

1. $r=2$, 尾数的最高位为1
2. $r=4$, 尾数的最高2位不全为0
3. $r=8$, 尾数的最高3位不全为0

浮点数的左规和右规

$r = 2$	左规	尾数左移 1 位 , 阶码减 1
	右规	尾数右移 1 位 , 阶码加 1
$r = 4$	左规	尾数左移 2 位 , 阶码减 1
	右规	尾数右移 2 位 , 阶码加 1
$r = 8$	左规	尾数左移 3 位 , 阶码减 1
	右规	尾数右移 3 位 , 阶码加 1

基数 r 越大 , 可表示的浮点数的范围越大

基数 r 越大 , 浮点数的精度降低

例题2

例：设机器数 $m = 4$, $n = 10$, $r = 2$, 尾数规格化后的浮点数表示范围

最大正数 $2^{+1111} \times 0.\underbrace{1111111111}_{10 \uparrow 1} = 2^{15} \times (1 - 2^{-10})$

最小正数 $2^{-1111} \times 0.\underbrace{1000000000}_{9 \uparrow 0} = 2^{-15} \times 2^{-1} = 2^{-16}$

最大负数 $2^{-1111} \times (-0.\underbrace{1000000000}_{9 \uparrow 0}) = -2^{-15} \times 2^{-1}$

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

例题3

例：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）。

例题3

解： 设 $x = +\frac{19}{128}$

二进制形式 $x = 0.0010011$

定点表示 $x = 0.0010011000$

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4

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）

例题4

解：设 $x = -58$

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 $[x]_{\text{原}} = 1, 0000111010$

$[x]_{\text{补}} = 1, 1111000110$

$[x]_{\text{反}} = 1, 1111000101$

浮点机中 $[x]_{\text{原}} = 0, 0110; 1. 1110100000$

$[x]_{\text{补}} = 0, 0110; 1. 0001100000$

$[x]_{\text{反}} = 0, 0110; 1. 0001011111$

$[x]_{\text{阶移、尾补}} = 1, 0110; 1. 0001100000$

三、定点数和浮点数比较

- 数的表示位数相同时，浮点数的表示范围远大于定点数的表示范围。
- 当浮点数为规格化数时，其相对精度比定点数高。
- 浮点数表示方法复杂，运算复杂，速度慢。
- 在溢出判断和编程方面，定点数比浮点数繁琐
(定点数：比例因子；浮点数：上溢)

四、机器零

- 浮点数尾数为0时，不论其阶码为何值按机器零处理
- 当浮点数 阶码等于或小于它所表示的最小数时，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：

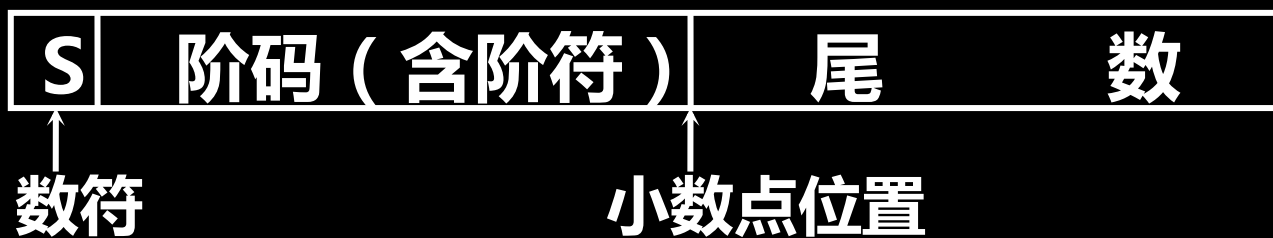
$\times, \times \times \times \times ; 0.00 \dots 0$
(阶码 = -16) $1, 0000 ; \times.\times \times \dots \times$

当阶码用移码，尾数用补码表示时，机器零为

$0, 0000 ; 0.00 \dots 0$

有利于机器中“判 0 ”电路的实现。

推荐阅读：IEEE754标准的浮点数格式



阶码用移码表示，尾数用原码表示，基数为2

	符号位 S	阶码	尾数	总位数
短浮点数	1	8	23	32
长浮点数	1	11	52	64
临时浮点数	1	15	64	80

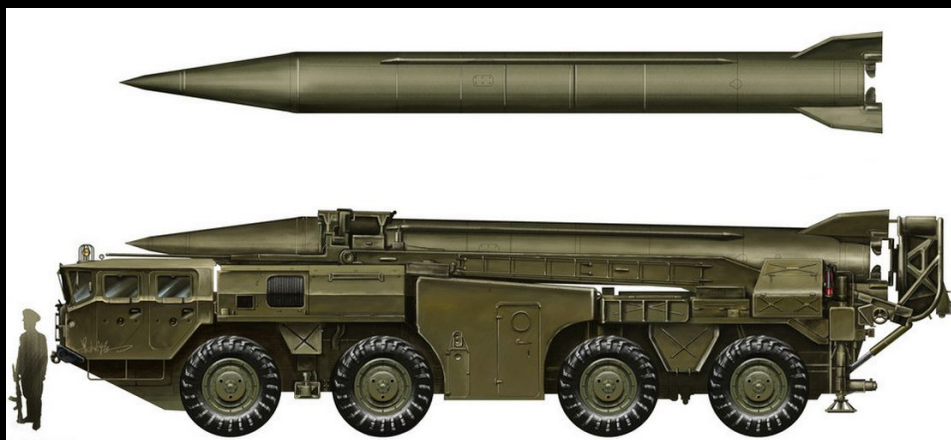


定点数移位运算

大连理工大学 赖晓晨

计算精度酿成的悲剧

- 1991年2月25日，一个爱国者导弹连未能成功地拦截飞向沙特宰赫兰兵营的飞毛腿导弹，致使美军28名士兵丧生，98名士兵受伤。
- 计算机运算的舍入误差，导致计算精度不够，最终拦截失败。



一、移位运算意义

$15.m = 1500.cm$ 小数点右移 2 位

机器用语：15相对于小数点左移2 位（小数点不动）

左移：绝对值扩大 右移：绝对值缩小

在计算机中，**移位与加减配合，能够实现乘除运算。**

移位导致寄存器中出现空位，如何处理？

二、算术移位（有符号数移位）规则

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1

设机器数字长为 8 位（含 1 位符号位），
写出 $A = +26$ 时，三种机器数左、右移一位和
两位后的表示形式及对应的真值，并分析结果
的正确性。

例题1

解：A = +26 = +11010

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
左移一位	0,0110100	+ 52
左移两位	0,1101000	+104
右移一位	0,0001101	+13
右移两位	0,0000110	+ 6

例题1

对于正数，三种机器数移位后符号位均不变，左移时最高位丢失1，则结果出错；右移时最低位丢失1，影响精度。

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
左移一位	0,0110100	+ 52
左移两位	0,1101000	+104
右移一位	0,0001101	+13
右移两位	0,0000110	+ 6

例题2

设机器数字长为 8 位（含 1 位符号位），写出 $A = -26$ 时，三种机器数左、右移一位和两位后的表示形式及对应的真值，并分析结果的正确性。

例题2

解： $A = -26 = -11010$

原码

移位操作	机 器 数	对应的真值
移位前	1,0011010	- 26
左移一位	1,0110100	- 52
左移两位	1,1101000	- 104
右移一位	1,0001101	- 13
右移两位	1,0000110	- 6

例题2

解： $A = -26 = -11010$

原码

移位操作	机 器 数	对应的真值
移位前	1,0011010	- 26
左移一位	1,0110100	- 52
左移两位	1,1101000	- 104
右移一位	1,0001101	- 13
右移两位	1,0000110	- 6

负数原码，移位后符号位不变，左移时，高位丢1，结果出错；右移时，低位丢1，影响精度。

例题2

反码

移位操作	机 器 数	对应的真值
移位前	1,1100101	– 26
左移一位	1,1001011	– 52
左移两位	1,0010111	– 104
右移一位	1,1110010	– 13
右移两位	1,1111001	– 6

例题2

反码

移位操作	机 器 数	对应的真值
移位前	1,1100101	- 26
左移一位	1,100101 1	- 52
左移两位	1,00101 11	- 104
右移一位	1, 1 110010	- 13
右移两位	1, 11 11001	- 6

负数反码，移位后符号位不变。左移时，最高位丢失0，则结果出错；右移时最低位丢失0，影响精度。

例题2

补码

移位操作	机 器 数	对应的真值
移位前	1,1100110	– 26
左移一位	1,1001100	– 52
左移两位	1,0011000	– 104
右移一位	1,1110011	– 13
右移两位	1,1111001	– 7

例题2

补码

移位操作	机 器 数	对应的真值
移位前	1,1100110	- 26
左移一位	1,1001100	- 52
左移两位	1,0011000	- 104
右移一位	1,1110011	- 13
右移两位	1,1111001	- 7

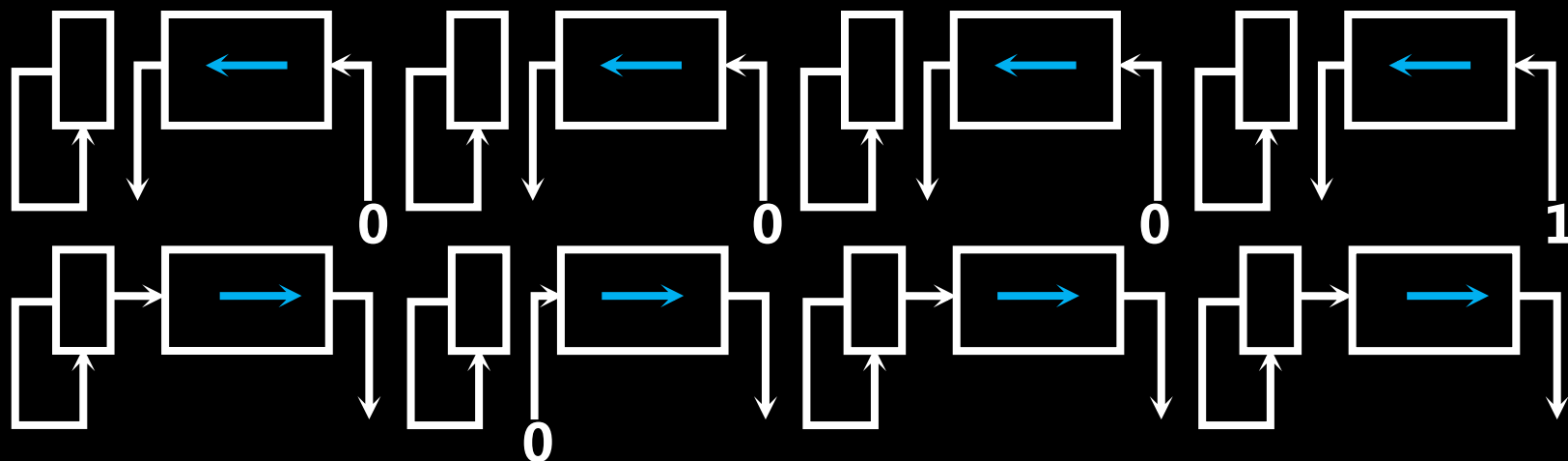
负数补码，移位后符号位不变。左移时，高位丢0，结果出错，右移时，低位丢1，影响精度。

三、算术移位和逻辑移位的区别

1. 算术移位：有符号数的移位
2. 逻辑移位：无符号数的移位
 - 逻辑左移：低位补0，高位移丢
 - 逻辑右移：高位补0，低位移丢

例如	01010011		10110010
逻辑左移	10100110	逻辑右移	01011001
算术左移	00100110	算术右移	11011001 (补码)

四、移位运算硬件实现



a)真值为正 b)负数的原码 c)负数的补码 d)负数的反码

← 丢 1 出错

出错

正确

正确

→ 丢 1 影响精度

影响精度

影响精度

正确

推荐阅读：算术精确性

- 整数可在最大值和最小值之间表示所有数，但是浮点数通常是一个无法表示的数的近似。
- 例如，在0和1之间的实数有无穷多个，但是64位浮点数只能精确表示 2^{53} 个。我们能做的就是给出尽可能接近实际数的浮点表示。
- IEEE754标准提供了几种方式：
 - 保护位、舍入位、粘贴位
 - 奇数加1，偶数截尾
 - 混合乘加

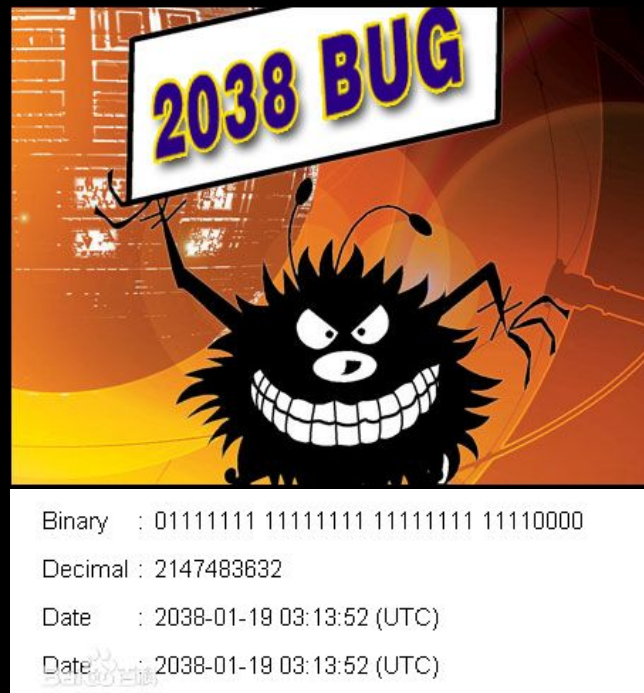


定点数加减法及溢出判断

大连理工大学 赖晓晨

2038年问题

- “千年虫”解决之后，“2038年”提出新的挑战。
- 大多数C语言程序都使用到一个叫做“标准时间库”的程序库，这个时间库用一个标准的4字节也就是32位的形式来储存时间信息。
- 32位存储空间表示有符号数时最大值是2147483647，从1970年1月1日0点开始计时，于2038年1月19日凌晨03:14:07达到最大值，在那惊心动魄的最后一秒后，会发生什么？



一、补码加减运算公式

1. 加法

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

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

2. 减法

$$A-B = A+(-B)$$

整数 $[A-B]_{\text{补}} = [A+(-B)]_{\text{补}} = [A]_{\text{补}} + [-B]_{\text{补}}$

小数 $[A-B]_{\text{补}} = [A+(-B)]_{\text{补}} = [A]_{\text{补}} + [-B]_{\text{补}}$

连同符号位一起相加，符号位产生的进位自然丢掉。

例题1

例：设 $A = 0.1011$, $B = -0.0101$ 求 $[A + B]_{\text{补}}$

解： $[A]_{\text{补}} = 0.1011$

$+ [B]_{\text{补}} = 1.1011$

$$\begin{array}{r} [A]_{\text{补}} + [B]_{\text{补}} = \boxed{1}0.0110 = [A + B]_{\text{补}} \end{array}$$

$\therefore A + B = 0.0110$

验证

$$\begin{array}{r} 0.1011 \\ - 0.0101 \\ \hline 0.0110 \end{array}$$

例：设 $A = -9$, $B = -5$ 求 $[A+B]_{\text{补}}$

解： $[A]_{\text{补}} = 1, 0111$

$+ [B]_{\text{补}} = 1, 1011$

$$\begin{array}{r} [A]_{\text{补}} + [B]_{\text{补}} = \boxed{1}1, 0010 = [A + B]_{\text{补}} \end{array}$$

验证

$$\begin{array}{r} - 1001 \\ + - 0101 \\ \hline - 1110 \end{array}$$

例题2

设机器数字长为 8 位（含 1 位符号位）且
 $A = 15$, $B = 24$, 用补码求 $A - B$

解：

$$A = 15 = 0001111$$
$$B = 24 = 0011000$$

$$\begin{array}{r} [A]_{\text{补}} = 0, 0001111 \\ + [-B]_{\text{补}} = 1, 1101000 \\ \hline \end{array}$$

$$[A]_{\text{补}} + [-B]_{\text{补}} = 1, 1110111 = [A - B]_{\text{补}}$$

$$\therefore A - B = -1001 = -9$$

练习1

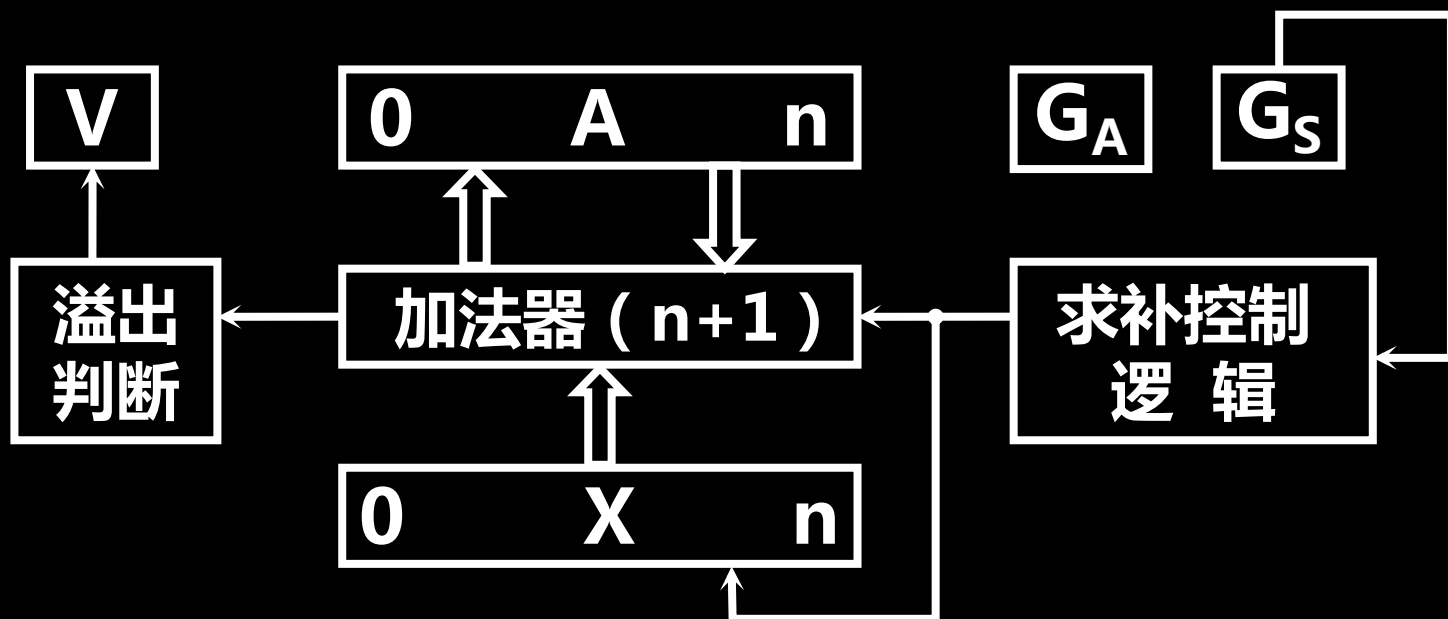
练习：设 $x = \frac{9}{16}$ $y = \frac{11}{16}$, 用补码求 $x+y$

$$x + y = -0.1100 = -\frac{12}{16} \quad \text{错}$$

练习：设机器数字长为 8 位（含 1 位符号位）
且 $A = -97$, $B = +41$, 用补码求 $A - B$ 。

$$A - B = +1110110 = +118 \quad \text{错}$$

补码加减法硬件电路



A、X 均 $n+1$ 位

用减法标记 G_S 控制求补逻辑

二、溢出

- 运算结果超出计算机字长所能表示范围的情况，称为溢出。

溢出判断

1. (符号相同两数) 一位符号位判断溢出：
参加运算的两个数 (减法时即为被减数和求补后的减数) 符号相同，其结果的符号与原操作数的符号不同，即为溢出。

$$A = -0.1011 \quad B = -0.0111$$

$$[A]_{\text{补}} = 1.0101$$

$$[B]_{\text{补}} = 1.1001$$

$$[A]_{\text{补}} + [B]_{\text{补}} = 10.1110 = 0.1110$$

两数符号位相同，结果的符号位发生变化，溢出。

溢出判断

2. (任意符号位两数) 一位符号位判断溢出：
参加运算的两个数，符号位可以不同，如果
运算结果中最高数值位的进位与符号位的进
位不同，则发生了溢出。

$$A = -0.1011 \quad B = -0.0111$$

$$[A]_{\text{补}} = 1.0101$$

$$[B]_{\text{补}} = 1.1001$$

$$[A]_{\text{补}} + [B]_{\text{补}} = 10.1110 = 0.1110$$

最高数值位无进位，符号位有进位，溢出。

溢出判断

3. 两位符号位判断溢出：参加运算的两个数，均采用双符号位（变形补码），如果结果的两位符号位不同，则发生了溢出。

$$A = -0.1011 \quad B = -0.0111$$

$$[A]_{\text{补}} = 11.0101$$

$$[B]_{\text{补}} = 11.1001$$

$$[A]_{\text{补}} + [B]_{\text{补}} = 110.1110 = 10.1110$$

结果两位符号位不同，溢出。

溢出判断

- 无论是否发生溢出，双符号位的高位总代表真正的符号，如果该位为1，表示结果为负，此时称为下溢，如果该位为0，表示结果为正，此时称为上溢。
- 上述结论对整数同样适用。

$$A = -0.1011 \quad B = -0.0111$$

$$[A]_{\text{补}} = 11.0101$$

$$[B]_{\text{补}} = 11.1001$$

$$[A]_{\text{补}} + [B]_{\text{补}} = 110.1110 = 10.1110$$

结果两位符号位不同，溢出。

推荐阅读：多媒体算术运算

- 在图像系统中，一般用8位表示一个颜色分量；在音频系统中，音频采用一般用16位精度
- 一般微处理器除了数据传输操作外，很少有支持这么短位长的操作。
- 解决办法是将64位加法器的进位链分段，使其可以同时处理8个8位数据，或者4个16位数据。
- 饱和操作是通用微处理器中不常出现的特征，饱和意味着计算结果溢出时，结果被设为最大的正数或很小的负数，而非取模。



定点数乘法

大连理工大学 赖晓晨

中国古代如何做乘法

- 《镜花缘》第79回，几位小姐妹聚在一起谈论数学。其中一位名叫青钿的，指着面前的圆桌，问道：“请教姐姐，这桌周围几尺？”
- 米兰芬向身边的宝云要过一把尺来，量出圆桌面的直径是三尺二寸。然后取笔画了一个“铺地锦”，之后回答说：“此桌周围一丈零零四分八。”

	三	二	
一	一	九	三
0	一	三	一
0	一	二	四
	四	八	

10.048

乘法运算

1. 定点数一位乘法

(1) 定点原码一位乘法

(2) 定点补码一位乘法

➤ 校正法

➤ 布斯法 (booth)

2. 定点数二位乘法

(1) 定点原码两位乘

(2) 定点补码两位乘

一、定点原码一位乘手算方法

手算分析：X=0.1101, Y= 0.1011；求X·Y

0.1101	列竖式
× 0.1011	按照乘数低位构造乘积
<hr/>	权值相同位对齐
1101	多数相加，得到乘积
1101	
0000	
1101	
<hr/>	积的符号为异或
0.10001111	

一、定点原码一位乘手算方法

手算分析：X=0.1101, Y= 0.1011；求X·Y

0.1101	列竖式
× 0.1011	按照乘数低位构造乘积
1101	权值相同位对齐
1101	多数相加，得到乘积
0000	
1101	
0.10001111	积的符号为异或

计算机运算的问题

- 多个部分积相加不能一次完成。
- 乘积是乘数的2倍长，加法器需倍长。

二、机器运算方法

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

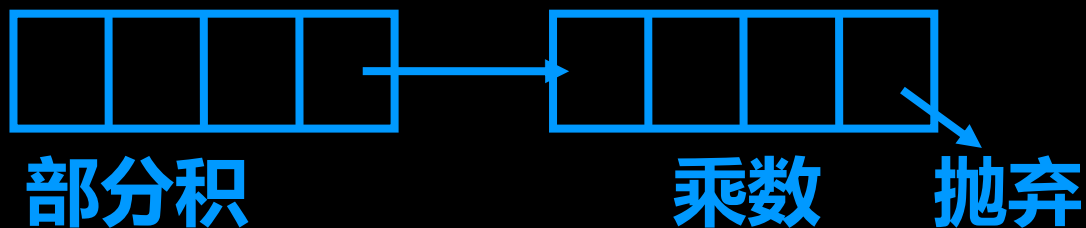
← 被乘数
← 乘数
← 部分积

1. 右移代替左移。部分积相加时最低位只用一次，此位将来不再参与运算，因此可将**部分积寄存器右移一位**
2. 乘数的各个位从右向左均只用一次，最低位不再使用，因此部分积右移时，**乘数寄存器也可右移一位**，即可用乘数寄存器的最高位接收部分积移出的位。

二、机器运算方法

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

← 被乘数
← 乘数
← 部分积



例题1

例：设 $X=0.1101$ ， $Y=0.1011$ ，求 $X \cdot Y$ 。

例题1

例：设 $X=0.1101$ ， $Y=0.1011$ ，求 $X \cdot Y$ 。

部 分 积	乘 数	说 明

例题1

例：设 $X=0.1101$ ， $Y=0.1011$ ，求 $X \cdot Y$ 。

部 分 积	乘 数	说 明
0.0000		

例题1

例：设 $X=0.1101$ ， $Y=0.1011$ ，求 $X \cdot Y$ 。

部分积	乘数	说明
0.0000	101 <u>1</u>	

例题1

例：设 $X=0.1101$ ， $Y=0.1011$ ，求 $X \cdot Y$ 。

部 分 积	乘 数	说 明
0 . 0 0 0 0	1 0 1 <u>1</u>	初态，部分积 = 0

例题1

例：设 $X=0.1101$ ， $Y=0.1011$ ，求 $X \cdot Y$ 。

部 分 积	乘 数	说 明
0 . 0 0 0 0	1 0 1 1 =	初态，部分积 = 0 乘数为 1，加被乘数

例题1

例：设 $X=0.1101$ ， $Y=0.1011$ ，求 $X \cdot Y$ 。

部 分 积	乘 数	说 明
$\begin{array}{r} 0.0000 \\ +0.1101 \end{array}$	$\begin{array}{r} 1011 \\ = \end{array}$	初态，部分积 = 0 乘数为 1，加被乘数

例题1

例：设 $X=0.1101$ ， $Y=0.1011$ ，求 $X \cdot Y$ 。

部 分 积	乘 数	说 明
$\begin{array}{r} 0.0000 \\ +0.1101 \\ \hline \end{array}$	$\begin{array}{r} 1011 \\ = \end{array}$	初态，部分积 = 0 乘数为 1，加被乘数

例题1

例：设 $X=0.1101$ ， $Y=0.1011$ ，求 $X \cdot Y$ 。

部分积	乘数	说明
$ \begin{array}{r} 0.0000 \\ +0.1101 \\ \hline 0.1101 \end{array} $	$1011 =$	初态，部分积 = 0 乘数为 1，加被乘数

例题1

例：设 $X=0.1101$ ， $Y=0.1011$ ，求 $X \cdot Y$ 。

部分积	乘数	说明
$\begin{array}{r} 0.0000 \\ +0.1101 \\ \hline \end{array}$	$\begin{array}{r} 1011 \\ = \end{array}$	初态，部分积 = 0 乘数为 1，加被乘数
$\begin{array}{r} 0.1101 \\ 0.0110 \\ \hline \end{array}$	$\begin{array}{r} 1101 \\ = \end{array}$	→ 1，形成新的部分积

例题1

例：设 $X=0.1101$ ， $Y=0.1011$ ，求 $X \cdot Y$ 。

部 分 积	乘 数	说 明
$\begin{array}{r} 0.0000 \\ +0.1101 \\ \hline \end{array}$	$\begin{array}{r} 1011 \\ = \end{array}$	初态，部分积 = 0 乘数为 1，加被乘数
$\begin{array}{r} 0.1101 \\ 0.0110 \\ \hline \end{array}$	$\begin{array}{r} 1101 \\ = \end{array}$	→ 1，形成新的部分积 乘数为 1，加被乘数

例题1

例：设 $X=0.1101$ ， $Y=0.1011$ ，求 $X \cdot Y$ 。

部分积	乘数	说明
$\begin{array}{r} 0.0000 \\ +0.1101 \\ \hline \end{array}$	$\begin{array}{r} 1011 \\ = \end{array}$	初态，部分积 = 0 乘数为 1，加被乘数
$\begin{array}{r} 0.1101 \\ 0.0110 \\ +0.1101 \\ \hline \end{array}$	$\begin{array}{r} 1101 \\ = \end{array}$	→ 1，形成新的部分积 乘数为 1，加被乘数

例题1

例：设 $X=0.1101$ ， $Y=0.1011$ ，求 $X \cdot Y$ 。

部 分 积	乘 数	说 明
$\begin{array}{r} 0.0000 \\ +0.1101 \\ \hline \end{array}$	$\begin{array}{r} 1011 \\ = \end{array}$	初态，部分积 = 0 乘数为 1，加被乘数
$\begin{array}{r} 0.1101 \\ 0.0110 \\ +0.1101 \\ \hline \end{array}$	$\begin{array}{r} 1101 \\ = \end{array}$	→ 1，形成新的部分积 乘数为 1，加被乘数

例题1

例：设 $X=0.1101$ ， $Y=0.1011$ ，求 $X \cdot Y$ 。

部 分 积	乘 数	说 明
$\begin{array}{r} 0.0000 \\ +0.1101 \\ \hline \end{array}$	$\begin{array}{r} 1011 \\ = \end{array}$	初态，部分积 = 0 乘数为 1，加被乘数
$\begin{array}{r} 0.1101 \\ 0.0110 \\ +0.1101 \\ \hline \end{array}$	$\begin{array}{r} 1101 \\ = \end{array}$	→ 1，形成新的部分积 乘数为 1，加被乘数
1.0011	1	

例题1

例：设 $X=0.1101$ ， $Y=0.1011$ ，求 $X \cdot Y$ 。

部 分 积	乘 数	说 明
$\begin{array}{r} 0.0000 \\ +0.1101 \\ \hline \end{array}$	$\begin{array}{r} 1011 \\ = \end{array}$	初态，部分积 = 0 乘数为 1，加被乘数
$\begin{array}{r} 0.1101 \\ 0.0110 \\ +0.1101 \\ \hline \end{array}$	$\begin{array}{r} 1101 \\ = \end{array}$	$\rightarrow 1$ ，形成新的部分积 乘数为 1，加被乘数
$\begin{array}{r} 1.0011 \\ 0.1100 \\ \hline \end{array}$	$\begin{array}{r} 1 \\ 1110 \\ = \end{array}$	$\rightarrow 1$ ，形成新的部分积

例题1

例：设 $X=0.1101$ ， $Y=0.1011$ ，求 $X \cdot Y$ 。

部 分 积	乘 数	说 明
$\begin{array}{r} 0.0000 \\ +0.1101 \\ \hline \end{array}$	$\begin{array}{r} 1011 \\ = \end{array}$	初态，部分积 = 0 乘数为 1，加被乘数
$\begin{array}{r} 0.1101 \\ 0.0110 \\ +0.1101 \\ \hline \end{array}$	$\begin{array}{r} 1101 \\ = \end{array}$	→ 1，形成新的部分积 乘数为 1，加被乘数
$\begin{array}{r} 1.0011 \\ 0.1001 \\ \hline \end{array}$	$\begin{array}{r} 1 \\ 1110 \\ = \end{array}$	→ 1，形成新的部分积 乘数为 0，加 0

例题1

例：设 $X=0.1101$ ， $Y=0.1011$ ，求 $X \cdot Y$ 。

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

例题1

例：设 $X=0.1101$ ， $Y=0.1011$ ，求 $X \cdot Y$ 。

部 分 积	乘 数	说 明
$\begin{array}{r} 0.0000 \\ +0.1101 \\ \hline \end{array}$	$\begin{array}{r} 1011 \\ = \end{array}$	初态，部分积 = 0 乘数为 1，加被乘数
$\begin{array}{r} 0.1101 \\ 0.0110 \\ +0.1101 \\ \hline \end{array}$	$\begin{array}{r} 1101 \\ = \end{array}$	→ 1，形成新的部分积 乘数为 1，加被乘数
$\begin{array}{r} 1.0011 \\ 0.1001 \\ +0.0000 \\ \hline \end{array}$	$\begin{array}{r} 1 \\ 1110 \\ = \end{array}$	→ 1，形成新的部分积 乘数为 0，加 0

例题1

例：设 $X=0.1101$ ， $Y=0.1011$ ，求 $X \cdot Y$ 。

部 分 积	乘 数	说 明
$\begin{array}{r} 0.0000 \\ +0.1101 \\ \hline \end{array}$	$\begin{array}{r} 1011 \\ = \end{array}$	初态，部分积 = 0 乘数为 1，加被乘数
$\begin{array}{r} 0.1101 \\ 0.0110 \\ +0.1101 \\ \hline \end{array}$	$\begin{array}{r} 1101 \\ = \end{array}$	→ 1，形成新的部分积 乘数为 1，加被乘数
$\begin{array}{r} 1.0011 \\ 0.1001 \\ +0.0000 \\ \hline \end{array}$	$\begin{array}{r} 1 \\ 1110 \\ = \end{array}$	→ 1，形成新的部分积 乘数为 0，加 0
$\begin{array}{r} 0.1001 \end{array}$	$\begin{array}{r} 11 \end{array}$	

例题1

例：设 $X=0.1101$ ， $Y=0.1011$ ，求 $X \cdot Y$ 。

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

例题1

例：设 $X=0.1101$ ， $Y=0.1011$ ，求 $X \cdot Y$ 。

部 分 积	乘 数	说 明
$\begin{array}{r} 0.0000 \\ +0.1101 \\ \hline \end{array}$	$\begin{array}{r} 1011 \\ = \end{array}$	初态，部分积 = 0 乘数为 1，加被乘数
$\begin{array}{r} 0.1101 \\ 0.0110 \\ +0.1101 \\ \hline \end{array}$	$\begin{array}{r} 1101 \\ = \end{array}$	→ 1，形成新的部分积 乘数为 1，加被乘数
$\begin{array}{r} 1.0011 \\ 0.1001 \\ +0.0000 \\ \hline \end{array}$	$\begin{array}{r} 1 \\ 1110 \\ = \end{array}$	→ 1，形成新的部分积 乘数为 0，加 0
$\begin{array}{r} 0.1001 \\ 0.0100 \\ \hline \end{array}$	$\begin{array}{r} 11 \\ 1111 \\ = \end{array}$	→ 1，形成新的部分积 乘数为 1，加 被乘数

例题1

例：设 $X=0.1101$ ， $Y=0.1011$ ，求 $X \cdot Y$ 。

部 分 积	乘 数	说 明
$\begin{array}{r} 0.0000 \\ +0.1101 \\ \hline \end{array}$	$\begin{array}{r} 1011 \\ = \end{array}$	初态，部分积 = 0 乘数为 1，加被乘数
$\begin{array}{r} 0.1101 \\ 0.0110 \\ +0.1101 \\ \hline \end{array}$	$\begin{array}{r} 1101 \\ = \end{array}$	→ 1，形成新的部分积 乘数为 1，加被乘数
$\begin{array}{r} 1.0011 \\ 0.1001 \\ +0.0000 \\ \hline \end{array}$	$\begin{array}{r} 1 \\ 1110 \\ = \end{array}$	→ 1，形成新的部分积 乘数为 0，加 0
$\begin{array}{r} 0.1001 \\ 0.0100 \\ +0.1101 \\ \hline \end{array}$	$\begin{array}{r} 11 \\ 1111 \\ = \end{array}$	→ 1，形成新的部分积 乘数为 1，加 被乘数

例题1

例：设 $X=0.1101$ ， $Y=0.1011$ ，求 $X \cdot Y$ 。

部 分 积	乘 数	说 明
$\begin{array}{r} 0.0000 \\ +0.1101 \\ \hline \end{array}$	$\begin{array}{r} 1011 \\ = \end{array}$	初态，部分积 = 0 乘数为 1，加被乘数
$\begin{array}{r} 0.1101 \\ 0.0110 \\ +0.1101 \\ \hline \end{array}$	$\begin{array}{r} 1101 \\ = \end{array}$	→ 1，形成新的部分积 乘数为 1，加被乘数
$\begin{array}{r} 1.0011 \\ 0.1001 \\ +0.0000 \\ \hline \end{array}$	$\begin{array}{r} 1 \\ 1110 \\ = \end{array}$	→ 1，形成新的部分积 乘数为 0，加 0
$\begin{array}{r} 0.1001 \\ 0.0100 \\ +0.1101 \\ \hline \end{array}$	$\begin{array}{r} 11 \\ 1111 \\ = \end{array}$	→ 1，形成新的部分积 乘数为 1，加 被乘数

例题1

例：设 $X=0.1101$ ， $Y=0.1011$ ，求 $X \cdot Y$ 。

部 分 积	乘 数	说 明
$\begin{array}{r} 0.0000 \\ +0.1101 \\ \hline \end{array}$	$\begin{array}{r} 1011 \\ = \end{array}$	初态，部分积 = 0 乘数为 1，加被乘数
$\begin{array}{r} 0.1101 \\ 0.0110 \\ +0.1101 \\ \hline \end{array}$	$\begin{array}{r} 1101 \\ = \end{array}$	→ 1，形成新的部分积 乘数为 1，加被乘数
$\begin{array}{r} 1.0011 \\ 0.1001 \\ +0.0000 \\ \hline \end{array}$	$\begin{array}{r} 1 \\ 1110 \\ = \end{array}$	→ 1，形成新的部分积 乘数为 0，加 0
$\begin{array}{r} 0.1001 \\ 0.0100 \\ +0.1101 \\ \hline \end{array}$	$\begin{array}{r} 11 \\ 1111 \\ = \end{array}$	→ 1，形成新的部分积 乘数为 1，加被乘数
$\begin{array}{r} 1.0001 \\ \hline \end{array}$	$\begin{array}{r} 111 \\ \hline \end{array}$	

例题1

例：设 $X=0.1101$ ， $Y=0.1011$ ，求 $X \cdot Y$ 。

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

运算结果

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

2. 数值部分按绝对值相乘

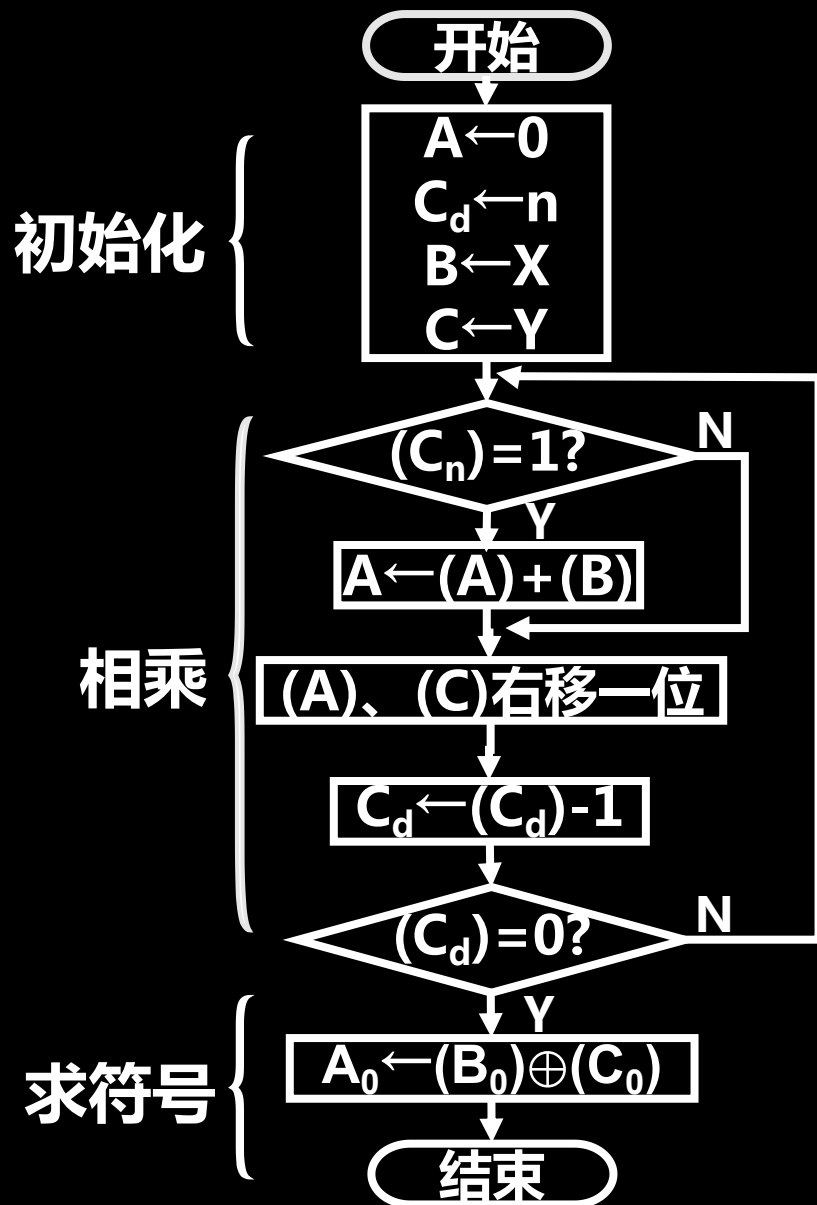
$$x^* \cdot y^* = 0.10001111$$

$$\text{则 } x \cdot y = 0.10001111$$

特点：

- 用移位的次数判断乘法是否结束
- 绝对值运算
- 逻辑移位

三、乘法运算的控制流程



部分积循环迭代

$$z_0 = 0$$

$$z_1 = (z_0 + XY_n)2^{-1}$$

$$z_2 = (z_1 + XY_{n-1})2^{-1}$$

...

$$z_{i+1} = (z_i + XY_{n-i})2^{-1}$$

...

$$z_n = (z_{n-1} + XY_1)2^{-1}$$

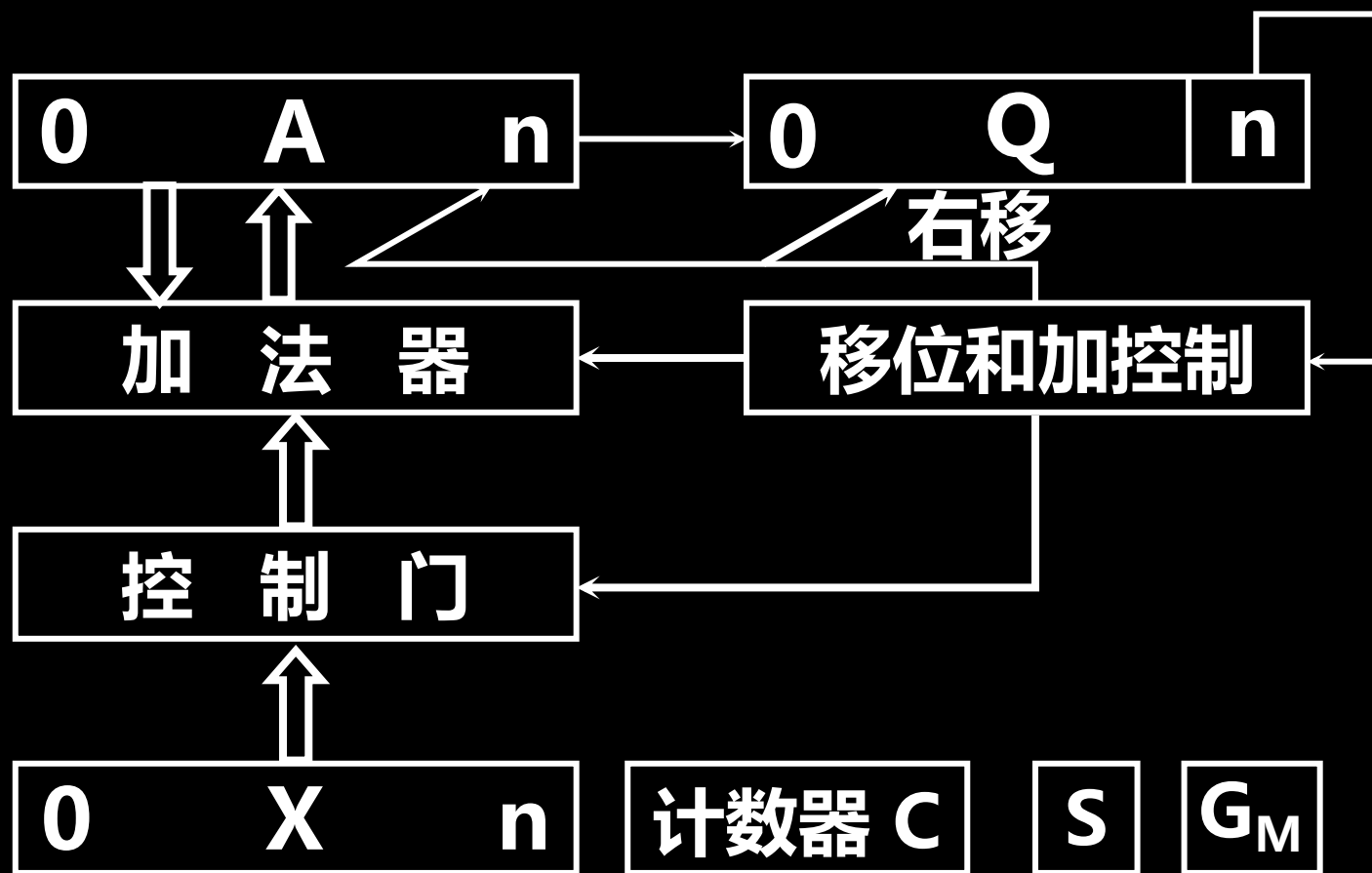
移位- 相加

例题2

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

	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	$\begin{array}{r} 0.1110 \\ 0.0111 \\ +0.0000 \\ \hline \end{array}$	$0\underline{110}$	$\rightarrow 1$, 得 z_1 + 0
逻辑右移	$\begin{array}{r} 0.0111 \\ 0.0011 \\ +0.1110 \\ \hline \end{array}$	0 $\underline{1011}$	$\rightarrow 1$, 得 z_2 + x^*
逻辑右移	$\begin{array}{r} 1.0001 \\ 0.1000 \\ +0.1110 \\ \hline \end{array}$	10 $\underline{1101}$	$\rightarrow 1$, 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

四、原码一位乘的硬件配置



A、X、Q 均 $n+1$ 位 移位和加受末位乘数控制



定点数补码一位乘

大连理工大学 赖晓晨

一、校正法

定点补码一位乘法预备知识1：

补码与真值的转换关系

设 $[X]_{\text{补}} = X_0 . X_1 X_2 \dots X_n$

$$X = -X_0 + \sum_{i=1}^n X_i 2^{-i} = -X_0 + 0.X_1 X_2 \dots X_n$$

当 $X < 0$ 时：

$$[X]_{\text{补}} = 1.X_1 X_2 \dots X_n = 2 + X$$

$$X = [X]_{\text{补}} - 2 = 1.X_1 X_2 \dots X_n - 2$$

$$= -1 + 0.X_1 X_2 \dots X_n$$

定点补码一位乘法预备知识2：

补码右移

- 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。

$$[X]_{\text{补}} = X_0.X_1X_2...X_n$$

$$[(1/2)*X]_{\text{补}} = X_0.X_0X_1X_2...X_n$$

- 以5个二进制位为例，一个符号位，四个数值位： $[-6] = 1,1010$

右移一位，符号位不变：

$$1,1101 = [-3] = [-6/2]$$

定点补码一位乘的校正法公式

➤ 补码一位乘用到的公式

$$\text{设 } [X]_{\text{补}} = X_0 . X_1 X_2 \dots X_n$$

$$[Y]_{\text{补}} = Y_0 . Y_1 Y_2 \dots Y_n$$

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

$$\sum_{i=1}^n Y_i 2^{-i} = (0 . Y_1 Y_2 \dots Y_n)$$

校正法实现补码一位乘

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

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

当y为正时，不校正；

$$[X \cdot Y]_{\text{补}} = [X]_{\text{补}} \cdot Y = [X]_{\text{补}} (0.Y_1Y_2...Y_n)$$

当y为负时，须校正；

$$[X \cdot Y]_{\text{补}} = [X]_{\text{补}} (0.Y_1Y_2...Y_n) + [-X]_{\text{补}}$$

校正法实现补码一位乘

由Y的补码的最低位乘X的补码，然后右移一位，再加Y的补码的次低位乘X的补码，然后接着右移一位，再加。。。直到Y补码的最高数据位乘X补码，最后右移一位，得到XY积的补码

当y为正时，不校正；

$$[X \cdot Y]_{\text{补}} = [X]_{\text{补}} \cdot Y = [X]_{\text{补}} (0.Y_1Y_2\ldots Y_n)$$

当y为负时，须校正；

$$[X \cdot Y]_{\text{补}} = [X]_{\text{补}} (0.Y_1Y_2\ldots Y_n) + [-X]_{\text{补}}$$

例 设 $X = -0.1101$, $Y = +0.1011$ 求 $[X \cdot Y]_{\text{补}}$

设 $X = -0.1101$, $Y = -0.1011$ 求 $[X \cdot Y]_{\text{补}}$

例题1

部分积P

乘数Y

例: 设 $X = -0.1101$,
 $Y = 0.1011$, 求 $X \cdot Y$ 。

$[X]_{\text{补}} = 11.0011$

$[Y]_{\text{补}} = 00.1011$

$[X \cdot Y]_{\text{补}} = 1.01110001$

$X \cdot Y = -0.10001111$

	0 0 . 0 0 0 0	1 0 1 1
+ [X]	1 1 . 0 0 1 1	
	1 1 . 0 0 1 1	
右移1位→	1 1 . 1 0 0 1	1 1 0 1
+ [X]	1 1 . 0 0 1 1	
	1 0 . 1 1 0 0	
右移1位→	1 1 . 0 1 1 0	0 1 1 0
+ 0	0 0 . 0 0 0 0	
	1 1 . 0 1 1 0	
右移1位→	1 1 . 1 0 1 1	0 0 1 1
+ [X]	1 1 . 0 0 1 1	
	1 0 . 1 1 1 0	
右移1位→	1 1 . 0 1 1 1	0 0 0 1

乘积高位

乘积低位

例题1

部分积P

乘数Y

为什么用两个符号位？

$$[X]_{\text{补}} = 11.0011$$

$$[Y]_{\text{补}} = 00.1011$$

$$[X \cdot Y]_{\text{补}} = 1.01110001$$

$$X \cdot Y = -0.10001111$$

0 0 . 0 0 0 0

+ [X] 1 1 . 0 0 1 1

1 1 . 0 0 1 1

右移1位→ 1 1 . 1 0 0 1

+ [X] 1 1 . 0 0 1 1

1 0 . 1 1 0 0

右移1位→ 1 1 . 0 1 1 0

+ 0 0 0 . 0 0 0 0

1 1 . 0 1 1 0

右移1位→ 1 1 . 1 0 1 1

+ [X] 1 1 . 0 0 1 1

1 0 . 1 1 1 0

右移1位→ 1 1 . 0 1 1 1

乘积高位

1 0 1 1

1 1 0 1

0 1 1 0

0 0 1 1

0 0 0 1

乘积低位

例题1

部分积P

乘数Y

例: 设 $X = -0.1101$,
 $Y = 0.1011$, 求 $X \cdot Y$ 。

0 0 . 0 0 0 0

1 0 1 1

+ [X] 1 1 . 0 0 1 1

1 1 . 0 0 1 1

右移1位→ 1 1 . 1 0 0 1

1 1 0 1

+ [X] 1 1 . 0 0 1 1

1 0 . 1 1 0 0

右移1位→ 1 1 . 0 1 1 0

+ 0 0 0 . 0 0 0 0

1 1 . 0 1 1 0

右移1位→ 1 1 . 1 0 1 1

+ [X] 1 1 . 0 0 1 1

1 0 . 1 1 1 0

右移1位→ 1 1 . 0 1 1 1

乘积高位

乘积低位

用补码运算，“加”后的和可正可负，高位符号位代表了实际极性，所以可以防止因溢出而丢掉正确的符号。

0001

001111

例题1

部分积P

乘数Y

	0 0. 0 0 0 0
+ [X]	1 1. 0 0 1 1
	1 1. 0 0 1 1
右移1位→	1 1. 1 0 0 1
+ [X]	1 1. 0 0 1 1
	1 0. 1 1 0 0
右移1位→	1 1. 0 1 1 0
+ 0	0 0. 0 0 0 0
	1 1. 0 1 1 0
右移1位→	1 1. 1 0 1 1
+ [X]	1 1. 0 0 1 1
	1 0. 1 1 1 0
右移1位→	1 1. 0 1 1 1
乘积高位	

1 0 1 1
1 1 0 1
0 1 1 0
0 0 1 1
0 0 0 1
乘积低位

注意：乘数用Y的补码！

$$[Y]_{\text{补}} = 00.1011$$

$$[X \cdot Y]_{\text{补}} = 1.01110001$$

$$X \cdot Y = -0.10001111$$

二、布斯 (Booth) 法

$$\begin{aligned}[X \cdot Y]_{\text{补}} &= [X]_{\text{补}} \cdot (-Y_0 + \sum_{i=1}^n Y_i \cdot 2^{-i}) \\&= [x](-Y_0 + Y_1 2^{-1} + Y_2 2^{-2} + \dots + Y_n 2^{-n}) \\&= [x][-Y_0 + (Y_1 - Y_1 2^{-1}) + (Y_2 2^{-1} - Y_2 2^{-2}) \\&\quad + \dots + (Y_n 2^{-(n-1)} - Y_n 2^{-n})] \\&= [x][(Y_1 - Y_0) + (Y_2 - Y_1) 2^{-1} + \dots + (Y_n - Y_{n-1}) 2^{-(n-1)} \\&\quad + (0 - Y_n) 2^{-n}]\end{aligned}$$

补充一项 $Y_{n+1}=0$, 则上式可写为 :

$$[X \cdot Y] = [X] \cdot \sum_{i=0}^n (Y_{i+1} - Y_i) 2^{-i}$$

Booth法控制流程

$$[X \cdot Y] = [X] \cdot \sum_{i=0}^n (Y_{i+1} - Y_i) 2^{-i}$$

按机器执行顺序求出每一步的部分积。

$$[z_0]_{\text{补}} = 0$$

$$[z_1]_{\text{补}} = \{ [z_0]_{\text{补}} + (Y_{n+1} - Y_n) [X]_{\text{补}} \} 2^{-1} \quad Y_{n+1} = 0$$

$$[z_2]_{\text{补}} = \{ [z_1]_{\text{补}} + (Y_n - Y_{n-1}) [X]_{\text{补}} \} 2^{-1}$$

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

$$[z_n]_{\text{补}} = \{ [z_{n-1}]_{\text{补}} + (Y_2 - Y_1) [X]_{\text{补}} \} 2^{-1}$$

$$[z_{n+1}]_{\text{补}} = \{ [z_n]_{\text{补}} + (Y_1 - Y_0) [X]_{\text{补}} \} = [X \cdot Y]_{\text{补}}$$

Booth法控制流程

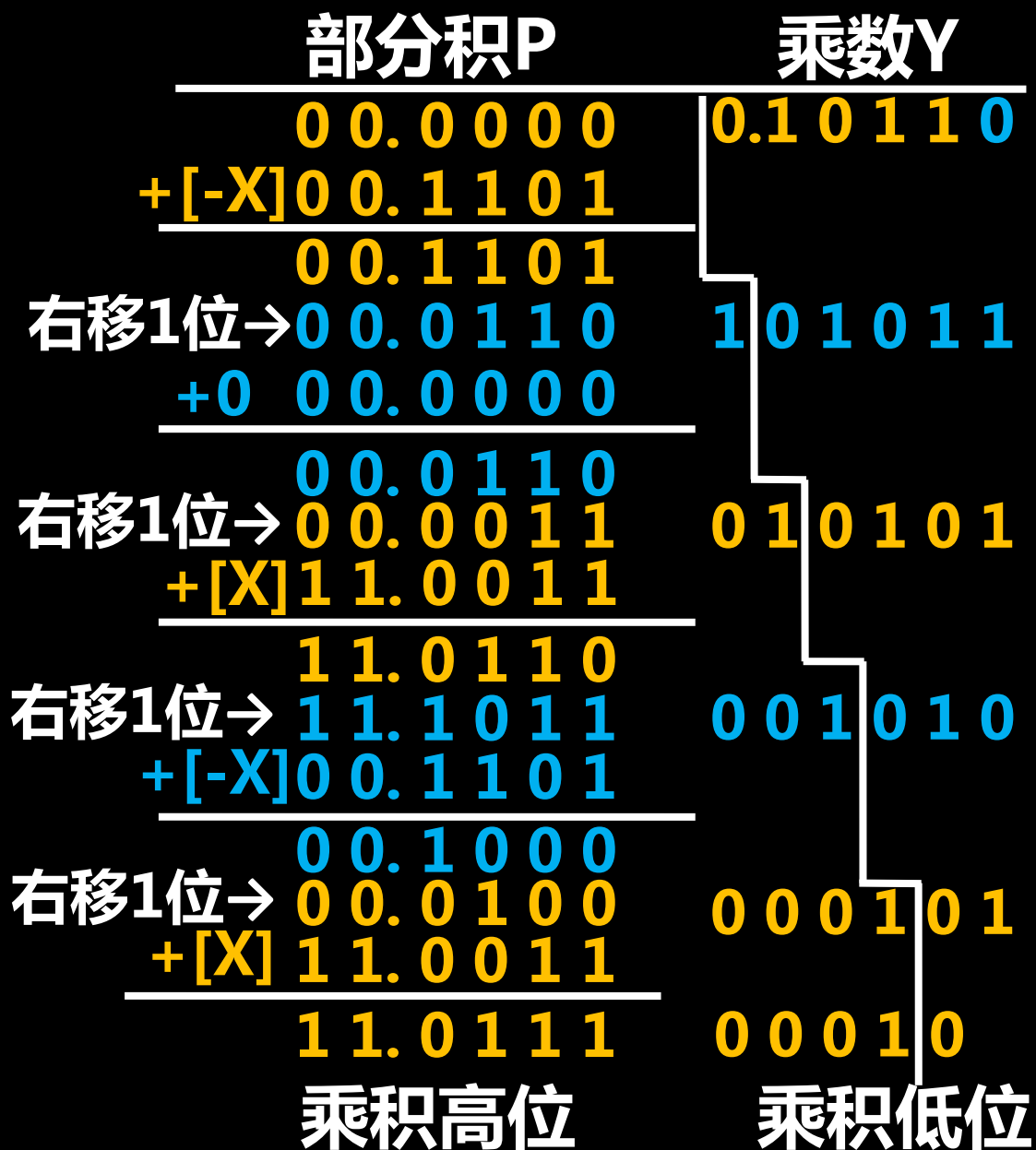
具体步骤：

- 乘数附加位置，0，即 $Y_{n+1} = 0$
- 分析[X]系数可能性， $(Y_{i+1} - Y_i)$ 可能有不同的组合

Y_{i+1}	Y_i	操作
0/1	0/1	$P_i + 0$, 右移1位
1	0	$P_i + [X]_{\text{补}}$, 右移1位
0	1	$P_i + [-X]_{\text{补}}$, 右移1位

- 加法重复n+1步，最后一次不移位，得 $[X \cdot Y]_{\text{补}}$

例题2



例：设 $X = -0.1101$,
 $Y = 0.1011$, 求 $X \cdot Y$.

$$[X]_{\text{补}} = 11.0011$$

$$[Y]_{\text{补}} = 00.1011$$

$$[-X]_{\text{补}} = 00.1101$$

$Y_{i+1} Y_i$	
相同	$P_i + 0$
10	$P_i + [X]_{\text{补}}$
01	$P_i + [-X]_{\text{补}}$

例题2

部分积P

00.0000

+ [-X] 00.1101

00.1101

右移1位→ 00.0110

+ 0 00.0000

00.0110

右移1位→ 00.0011

+ [X] 11.0011

11.0110

右移1位→ 11.1011

+ [-X] 00.1101

00.1000

右移1位→ 00.0100

+ [X] 11.0011

11.0111

乘积高位

乘数Y

0.10110

101011

001010

000101

00010

乘积低位

例：设 $X = -0.1101$ ， $Y = 0.1011$ ，求 $X \cdot Y$ 。
[X]补 = 11.0011

[X·Y]补 = 1.01110001
 $X \cdot Y = -0.10001111$

$Y_{i+1} \ Y_i$	
相同	$P_i + 0$
10	$P_i + [X]_{补}$
01	$P_i + [-X]_{补}$

例题2

部分积P	乘数Y
00.0000	0.10110
+ [-X] 00.1101	
00.1101	
右移1位 → 00.0110	101011
+ 0 00.0000	
00.0110	
右移1位 → 00.0011	0101
+ [X] 11.0011	
11.0110	
右移1位 → 11.1011	001
+ [-X] 00.1101	
00.1000	
右移1位 → 00.0100	000101
+ [X] 11.0011	
11.0111	00010
乘积高位	乘积低位

例：设 $X = -0.1101$,
 $Y = 0.1011$, 求 $X \cdot Y$.

为什么Y的符号位也参与运算？
 而且一共加了5次，且最后一次加不移位？

相同	$P_i + 0$
10	$P_i + [X]_{\text{补}}$
01	$P_i + [-X]_{\text{补}}$

布斯 (Booth) 法

布斯(Booth)法

$$[X \cdot Y]_{\text{补}} = [x] [(Y_1 - Y_0) + (Y_2 - Y_1)2^{-1} + \dots + (Y_n - Y_{n-1})2^{-(n-1)} + (0 - Y_n)2^{-n}]$$

N+1项相加，最高项权为1

推荐阅读：原码两位乘

- 按乘数每两位的取值情况，一次求出对应于该两位的部分积，此时，只要增加少量逻辑电路，就可以使乘法速度提高一倍。
 - 两位乘数有四种可能组合，对应于以下操作：
 - 00——相当于 $0 \cdot X$ 。部分积 P_i ，右移2位；
 - 01——相当于 $1 \cdot X$ 。部分积 $P_i + X$ ，右移2位；
 - 10——相当于 $2 \cdot X$ 。部分积 $P_i + 2X$ ，右移2位；
 - 11——相当于 $3 \cdot X$ 。部分积 $P_i + 3X$ ，右移2位。
- 3X : $(4X - X)$ 本次 $-X$ ，下次 $+X$ ，进位C
- 控制位： Y_{i-1} Y_i C



定点数除法

大连理工大学 赖晓晨

除号的由来

- 1544年，德国数学家施蒂费尔以一个或一对括号作除号，如以 $8)24$ 或 $8)24$ (表示 24 除以 8 ；
- 奥特雷德则以 $a)b(c$ 来表示 $b \div a = c$ ；
- J. 马洪 (1701年) 则以 $D)A+B-C$ 表示 $(A+B-C) \div D$ 。
- 1545年，施蒂费尔又改以大写德文字母 **D** 表示除。
- 现今之除号“ \div ”称为**雷恩记号**，是瑞士人 J. H. 雷恩于1659年出版的一本代数书中引用为除号。



除法分类

1. 定点原码一位除

➤恢复余数法

➤加减交替法

2. 定点补码一位除

一、除法运算分析

➤ 手算十进制除法： $X=934$ $Y=2$ $X/Y=?$

$$\begin{array}{r} 467 \\ 2 \overline{) 934} \\ \underline{8} \\ 13 \\ \underline{12} \\ 14 \\ \underline{14} \\ 0 \end{array}$$

一、除法运算分析

➤ 手算十进制除法： $X=934$ $Y=2$ $X/Y=?$

$$\begin{array}{r} 467 \\ 2 \overline{) 934} \\ \underline{8} \\ 13 \\ \underline{12} \\ 14 \\ \underline{14} \\ 0 \end{array}$$

- 除法是一系列减法，必须是大数减去小数
- 商需要不断凑试

一、除法运算分析

➤ 手算二进制除法：X=0.1011 Y=0.1101 X / Y = ?

$$\begin{array}{r} 0.1101 \overline{) 0.10110000} \\ \underline{1101} \\ 10010 \\ \underline{1101} \\ 10100 \\ \underline{1101} \\ 111 \end{array}$$

一、除法运算分析

➤ 手算二进制除法：X=0.1011 Y=0.1101 X / Y = ?

$$\begin{array}{r} 0.1101 \\ 0.1101 \overline{) 0.10110000} \\ \underline{1101} \\ 10010 \\ \underline{1101} \\ 10100 \\ \underline{1101} \\ 111 \end{array}$$

- 特点：商采用凑试法。计算机不会凑试！
- 问题：除数右移导致加法器位数变长

二、恢复余数法

特点

- 左移被除数（余数）来代替右移除数。

上商规则

- 将部分余数 R_i 与除数 Y 相减，判断余数的符号，确定商的值。
- 差为正：商上1，不恢复余数，部分余数左移1位。
- 差为负：商上0，恢复余数，部分余数左移1位。
- 特点：速度慢，控制方式复杂。

例题1

例：设 $X=0.1011$ ， $Y=0.1101$ ，求 X/Y 。

解： $[-Y]_{\text{补}}=11.0011$

被除数 (余数)	商	
0 0 1 0 1 1	0 0 0 0 0	开始
+ [-Y] 1 1 0 0 1 1		-Y即+ [-Y]
1 1 1 1 1 0	0 0 0 0 0	结果为负, 商0
+Y 0 0 1 1 0 1		恢复余数
0 0 1 0 1 1		左移一位
0 1 0 1 1 0	0 0 0 0	
+ [-Y] 1 1 0 0 1 1		
0 0 1 0 0 1	0 0 0 0 1	够减, 商1, 左移
0 1 0 0 1 0	0 0 0 1	
+ [-Y] 1 1 0 0 1 1		
0 0 0 1 0 1	0 0 0 1 1	够减, 商1 左移
0 0 1 0 1 0	0 0 1 1	
+ [-Y] 1 1 0 0 1 1		
1 1 1 1 0 1	0 0 1 1 0	不够减, 商0
+Y 0 0 1 1 0 1		恢复余数
0 0 1 0 1 0		左移一位
0 1 0 1 0 0	0 1 1 0	
+ [-Y] 1 1 0 0 1 1		
0 0 0 1 1 1	0 1 1 0 1	够减, 商1

被除数 (余数)	商	
0 0 1 0 1 1	0 0 0 0 0	开始
+ [-Y] 1 1 0 0 1 1		-Y即+ [-Y]
1 1 1 1 1 0	0 0 0 0 0	结果为负, 商0
+Y 0 0 1 1 0 1		恢复余数
0 0 1 0 1 1		左移一位

商：X/Y=0.1101，符号为正

余数：0.0111*2⁻⁴

运算过程中，被除数寄存器不断左移，右侧补零；

商的初值为0，不断左移；

可以认为是将商的高位移入了被除数寄存器的低位部分。

所以初始时可以用存放商的寄存器存放被除数的低n位，实现2n位数除以n位数。

+ [-Y] 1 1 0 0 1 1		
0 0 0 1 1 1	0 1 1 0 1	够减, 商1

三、加减交替法

1. 是恢复余数除法的一种修正：

- 当求得的差值为正时，商上1，余数左移一位减除数，新的余数为： $R_i = 2R_{i-1} - Y$ 。
- 当求得的差值为负时，商上0，恢复余数，左移一位减除数，新余数为：

$$R_i = 2(R_{i-1} + Y) - Y = 2R_{i-1} + Y。$$

2. 商为1时，求下一轮余数的方法为左移余数减除数，商为0时，左移余数并且加除数。

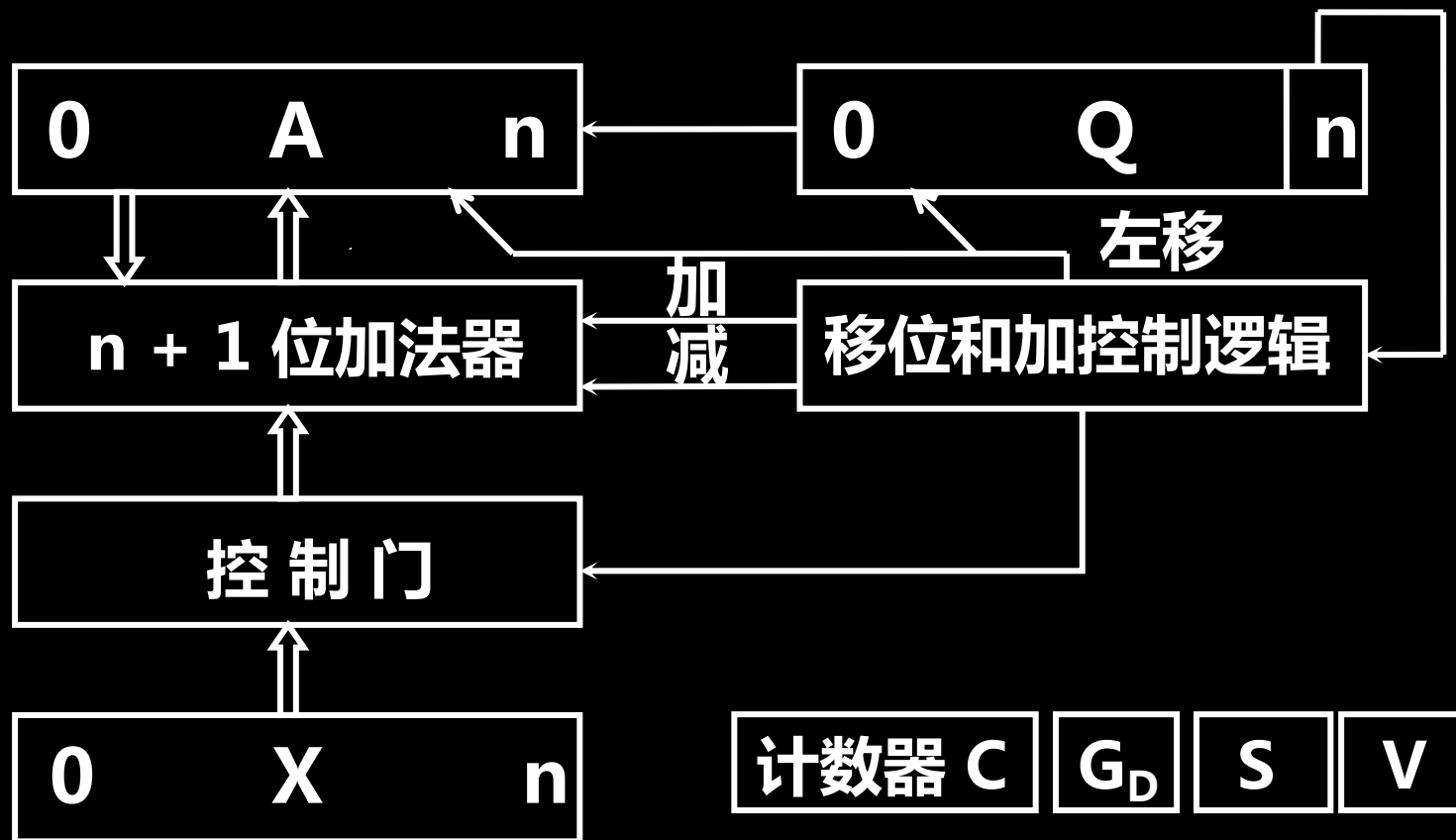
3. 若最后一次上商为0，需恢复余数，即 $R_n + Y$ 。

例题2

例：设 $X=0.1011$ ， $Y=0.1101$ ，求 X/Y 。

	被除数 (余数)	商	说明
$[-Y]_{\text{补}}$	00 10 11	00000	开始
$=11.0011$	$+ [-Y]$ 11 00 11		$-Y$ 即 $+ [-Y]$
	11 11 10	00000	不够减，商0
移 n 次，	11 11 00	0000	左移
加 n+1 次	$+Y$ 00 11 01		$+Y$
	00 10 01	00001	够减，商1
	01 00 10	0001	左移
	$+ [-Y]$ 11 00 11		减Y
	00 01 01	00011	够减，商1
	00 10 10	0011	左移
	$+ [-Y]$ 11 00 11		
	11 11 01	00110	不够减，商0
	11 10 10	0110	左移
商：			
$X/Y=0.1101$	$+Y$ 00 11 01		
余数：	00 01 11	01101	够减，商1
0.0111×2^{-4}			

原码加减交替除法硬件配置



A、X、Q 均 $n+1$ 位
用 Q_n 控制加减交替

自学：定点补码一位除法

1. 如果被除数与除数同号，用被除数减去除数；如异号，用被除数加除数。如果余数与除数同号，商为1，如果异号，商为0。该商即符号位。
2. 求商的数值部分：如上次商为1，则余数左移一位后减去除数；如上次商为0，则余数左移一位加除数。求商方法同1。如此重复执行 $n-1$ 次。
3. 商的最后一位可采用恒置1法，或按2的方法再求一步，得到商的第 n 位。除不尽时，如商为负，要在商的最低位加1，商为正不需加1。

推荐阅读：除法是右移吗？

- 无符号数的除法确实是右移操作
- 考虑有符号数： $(-5) \div 4$ 以32位机为例

1111 1111 1111 1111 1111 1111 1111 1011₂

- 右移两位后是:1073741822，显然不对

0011 1111 1111 1111 1111 1111 1111 1110₂

- 考虑算术右移时用符号位扩展：-2

1111 1111 1111 1111 1111 1111 1111 1110₂

- -2虽然与正确答案-1很接近，
但是仍然不对，那么应该怎么办呢？



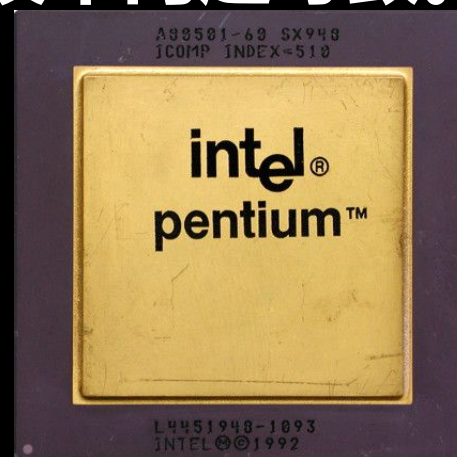


浮点数运算

大连理工大学 赖晓晨

pentium处理器浮点除法运算错误

- 1994年6月，在奔腾处理器推出前几天，英特尔发现浮点除法运算会发生偏差，修复代价为几十万美元。90亿次除法可能出现1次错误，每27000年中会遇到一次。
- 弗吉尼亚州的数学教授尼斯利博士发现了该错误。但是英特尔仅仅指出尚没有人发现类似错误，但未肯定是因为奔腾芯片技术问题导致。
- 最后，Intel向公众道歉，并承诺为所有用户无条件更换处理器，损失为5亿美金。



浮点运算方法

1. 浮点运算 $x = S_x \cdot 2^{j_x}$ $y = S_y \cdot 2^{j_y}$

➤ 加减法

➤ 乘除法

2. 浮点加减运算步骤

➤ 对阶、尾数相加减、规格化、舍入、判溢出。

一、浮点数加减法流程

1. “对阶”操作：使尾数加减可以进行。

- 比较两阶码， $\Delta J = J_X - J_Y$
- 将小阶码数的尾数右移 $|\Delta J|$ 位，其阶码值加 $|\Delta J|$
- 注意尾数右移特点：原码补0，补码补符号位
- 右移过程中移掉的位可用附加位电路暂时保存

2. 尾数加/减运算 $\Delta S = S_X \pm S_Y$

3. 尾数规格化：尽可能提高精度

$$\begin{aligned} 2^{-1} &\leq | [S]_{\text{原}} | \leq 1 - 2^{-n} \\ 2^{-1} &\leq | [S]_{\text{补}} | \leq 1 \end{aligned}$$

不同机器数的规格化特点

1. “对阶”操作：使尾数加减可以进行。

$S > 0$	规格化形式	$S < 0$	规格化形式
真值	$0.1 \times \times \cdots \times$	真值	$-0.1 \times \times \cdots \times$
原码	$0.\boxed{1} \times \times \cdots \times$	原码	$1.\boxed{1} \times \times \cdots \times$
补码	$\boxed{0.1} \times \times \cdots \times$	补码	$\boxed{1.0} \times \times \cdots \times$
反码	$0.1 \times \times \cdots \times$	反码	$1.0 \times \times \cdots \times$

$$2^{-1} \leq |[S]_{\text{原}}| \leq 1 - 2^{-n}$$
$$2^{-1} \leq |[S]_{\text{补}}| \leq 1$$

规格化操作的规则

规格化操作的规则：

- 右规：结果的两个符号位不同表示尾数结果溢出，右移1位尾数，阶码 $J+1$ 。
- 左规：结果的两个符号位相同，如是原码且数值位最高位为0，则左移，如是补码且数值位与符号位相同也需左移。同时阶码减去移动的位数。

一、浮点数加减法流程

4. 舍入

- 截断法
- 恒置1法
- 0舍1入法

5. 判断阶码溢出

- 阶码下溢，则置运算结果为机器零；
- 阶码上溢，则置溢出标志，报警中断。

例题1

例：已知 $X=2^{0010} \cdot (0.11011011)$
 $Y=2^{0100} \cdot (-0.10101100)$ ，求 $X+Y$ 。

1. 对阶操作

$$\Delta J = [J_X]_{\text{补}} + [-J_Y]_{\text{补}} = 00010 + 11100 = 11110$$

$$[S_X]_{\text{补}} = 00.00\ 110\ 110\ \underline{11} \quad J = 00100。$$

2. 尾数相加 $[S_X]_{\text{补}} + [S_Y]_{\text{补}}$

$$= 00.00110110\underline{11} + 11.01010100$$

$$= 11.\textcolor{red}{1}0001010\ \underline{11}$$

例题1

3. 规格化操作

左规，移1位，结果=11.00010101 10;

阶码-1，J=00011。

4. 舍入

[S]补=11.00010110, S=-0.11101010。

5. 判溢出

未溢出

最终结果为： $X+Y=2^{0011} \cdot (-0.11101010)$

二、浮点数乘法除法

1. 浮点乘法、除法运算规则

- 设有两个浮点数 x 和 y

$$X = 2^{E_x} \cdot M_x$$

$$Y = 2^{E_y} \cdot M_y$$

- 浮点乘除法运算的公式

$$X \times Y = 2^{(E_x + E_y)} \cdot (M_x \times M_y)$$

$$X \div Y = 2^{(E_x - E_y)} \cdot (M_x \div M_y)$$

2. 浮点乘除法运算的步骤：

- 操作数检查
- 计算阶码(加减)和尾数(乘除)
- 结果规格化、舍入和判溢出

二、浮点数乘法

3. 浮点数的阶码运算

(1) 阶码通常有两种表示：补码 / 移码

(2) 两个要用到的关系： $[X]_{\text{移}} = 2^n + X$,
 $[Y]_{\text{补}} = 2^{n+1} + Y$

(3) 用移码表示阶码，计算两数乘法时有两种情况

① 移码的修正：

$$[X]_{\text{移}} + [Y]_{\text{移}} = 2^n + [X+Y]_{\text{移}}$$

② 补移码混合：

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

$$[X]_{\text{移}} + [-Y]_{\text{补}} = [X-Y]_{\text{移}}$$

二、浮点数乘法除法

(4)移码加减后结果溢出的判断：双符号位判断

00 – 结果负	} 最高符号位0, 未溢出
01 – 结果正	
10 – 上溢出	} 最高符号位1, 有溢出
11 – 下溢出	

4. 浮点数的舍入处理(rounding)

(1) 截断法

(2) 舍入处理：恒置1法、0舍1入法、
有1就进位法

二、浮点数乘法

(3) IEEE754标准中的舍入处理

- 就近舍入：实质就是通常所说的“四舍五入”。
- 朝0舍入：即朝数轴原点方向舍入，就是简单的截尾。
- 朝 $+\infty$ 舍入：对正数来说,只要多余位不全为0则向最低有效位进1；对负数来说则是简单的截尾。
- 朝 $-\infty$ 舍入：处理方法正好与朝 $+\infty$ 舍入情况相反。对正数来说，只要多余位不全为0则简单截尾；对负数来说，向最低有效位进1。

例题1

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。

例题1

设： $X=2^{-5} \cdot 0.1110011$, $Y=2^3 \cdot (-0.1110010)$

(1) 求乘积的阶码

$$[E_X + E_Y]_{\text{移}} = [E_X]_{\text{移}} + [E_Y]_{\text{补}} = 0011 + 0011 = 0110$$

(2) 尾数相乘：

$$[X \cdot Y]_{\text{补}} = 11.00110011001010 \text{ 尾数部分}$$

(3) 规格化处理：本例已经是规格化形式了

(4) 舍入： $[X \cdot Y]_{\text{补}} = 1.0011010$ （尾数部分）

(5) 判溢出：未溢出

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

推荐阅读：并行性和计算机算术的结合律

- 通常情况下，程序是为顺序执行编写的，然后才考虑并行问题，那么，两个版本答案相同吗？
- 如果将100万个整数相加，无论用1个处理器，还是用1000个处理器，结果均相同。
- 由于浮点数是实数的近似，而且计算机是有精度限制的，因此对浮点数来说，上述结论并不成立，即浮点加法是不符合结合律的。
- 并行机的操作系统会根据程序调度不同数目的处理器，处理器数目不同会造成结果不同。
- 数学库：LAPACK/SCALAPAK



加法和算术逻辑单元

大连理工大学 赖晓晨

帕斯卡加法器

- 1639年，布莱斯·帕斯卡16岁，看着父亲费力地计算税率税款，想到了要为父亲制做一台可以帮助计算的机器。
- 他耗费了整整三年的光阴，1642年，“加法器”问世。“加法器”由一种系列齿轮组成，逢十进一，能做6位加法和减法。
- 1971年面世的PASCAL语言，就是为了纪念这位先驱而命名的。

人只不过是一根芦苇，是自然界最脆弱的东西，但我是一根有思想的芦苇。



一、一位半加器

实现两个一位二进制数相加的电路，称为半加器。
半加器有两个输入端（被加数和加数），两个输出端（和与进位）

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

半加器逻辑表达式

- 半加器的逻辑表达式如下

$$S = A \oplus B$$

$$C = AB$$

- 用一个异或门和一个与门即可实现半加器。

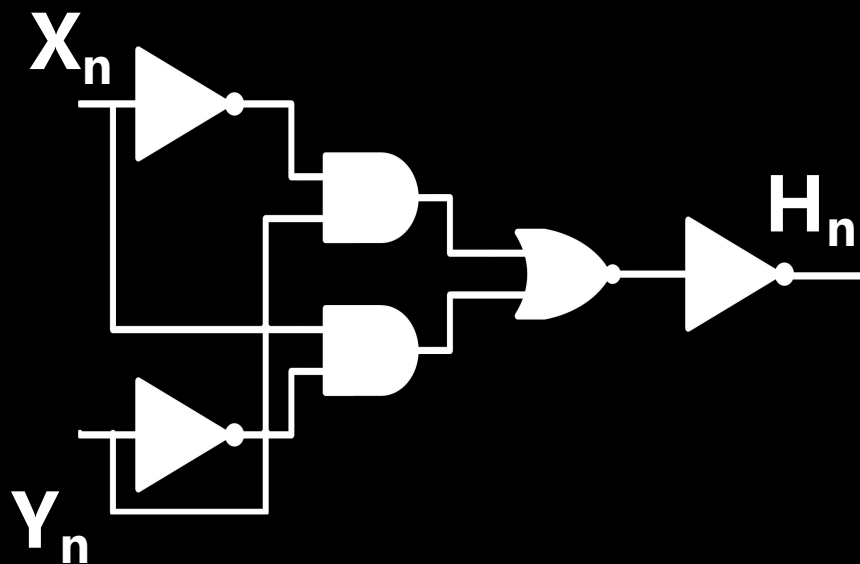
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

半加器的逻辑电路

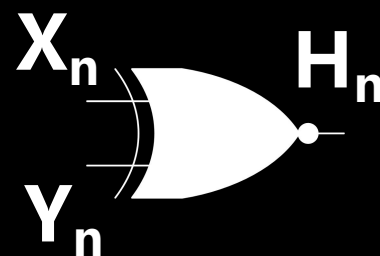
功能表

X_n	Y_n	H_n
0	0	0
1	0	1
0	1	1
1	1	0

(a)



(b)



(c)

二、一位全加器

当多位二进制数据相加时，对每一位而言，除了有被加数和加数之外，还有从低位送来的进位，考虑到进位的加法器称为全加器。

全加器真值表

A	B	Ci	Si	Co
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

全加器逻辑表达式

➤ 逻辑表达式

$$\begin{aligned} S &= \overline{A}\overline{B}C_i + \overline{A}B\overline{C}_i \\ &\quad + A\overline{B}\overline{C}_i + ABC_i \\ C_o &= \overline{A}BC_i + A\overline{B}C_i \\ &\quad + AB\overline{C}_i + ABC_i \end{aligned}$$

➤ 化简

$$\begin{aligned} S &= A \oplus B \oplus C_i \\ C &= AB + BC_i + AC_i \end{aligned}$$

A	B	C _i	S	C _o
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

全加器逻辑电路

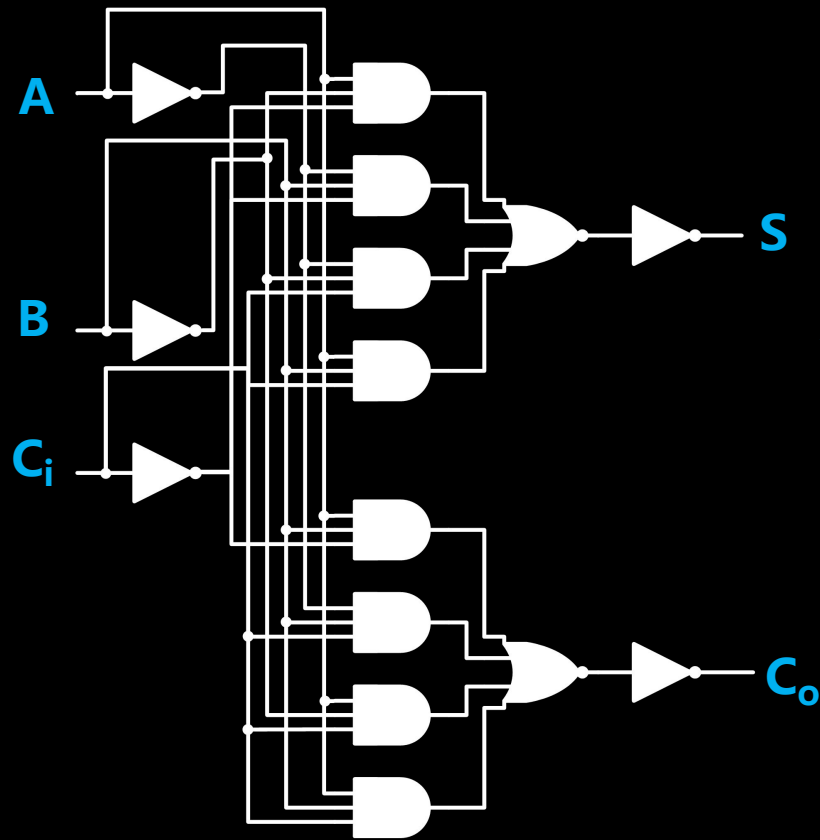
$$S = A \oplus B \oplus C_i$$

$$C_o = AB + BC_i + AC_i$$

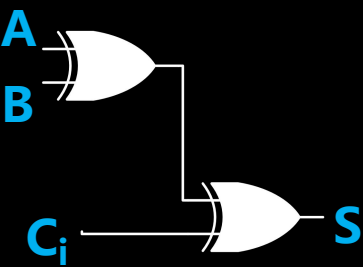
功能表

X_n	Y_n	C_{n-1}	F_n	C_n
0	0	0	0	0
0	0	1	1	0
1	0	0	1	0
1	0	1	0	1
0	1	0	1	0
0	1	1	0	1
1	1	0	0	1
1	1	1	1	1

(a) 功能表



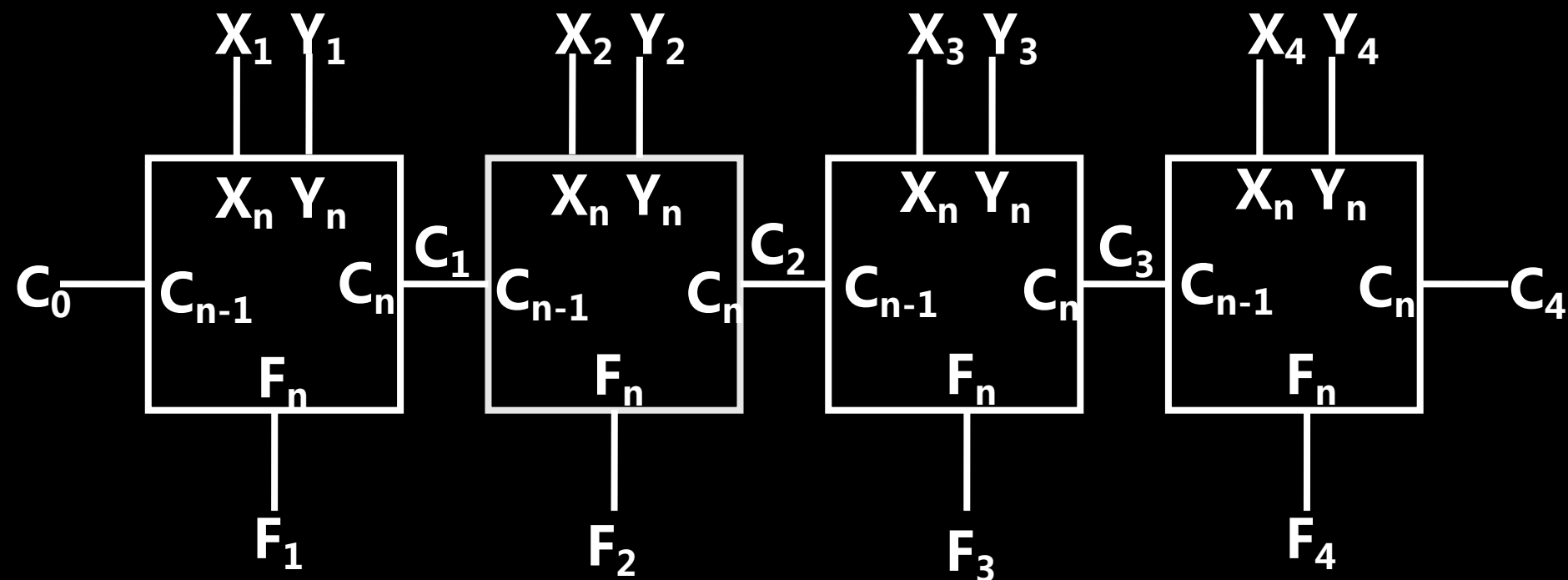
(b) 逻辑图



(c) 逻辑图

三、n位串行加法器

多位二进制数据的加法可用多个全加器来完成，参加运算的两组数据并行加入，进位信号串行传递，称为n位串行加法器。

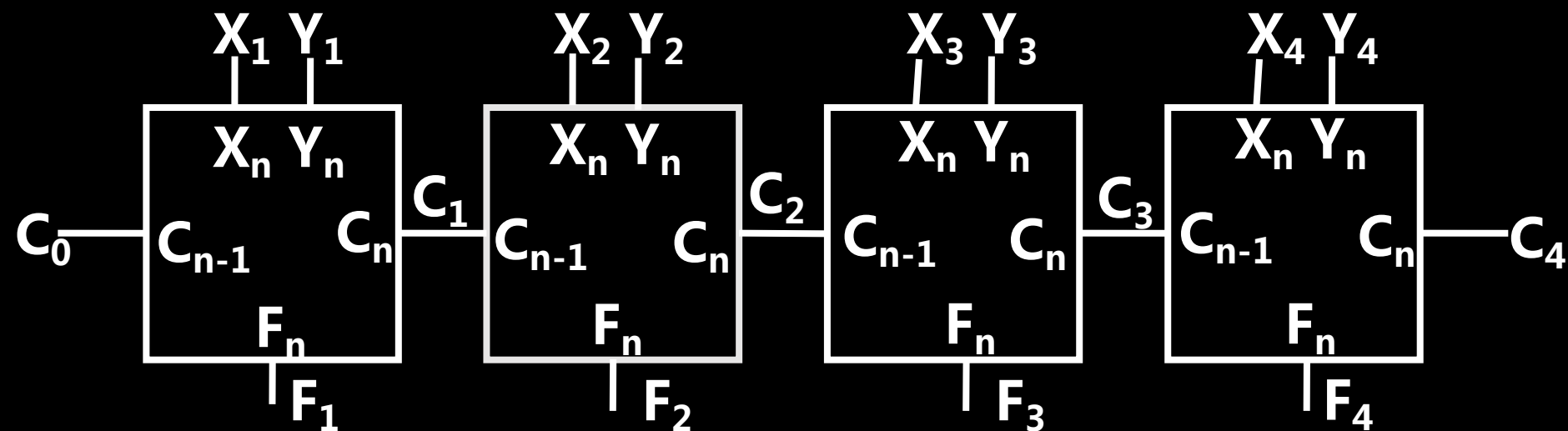


n位串行加法器的进位分析

C_1 形成的条件：

- X_1, Y_1 均为1。
- X_1, Y_1 任一为1，同时 C_0 为1。

$$C_1 = X_1 Y_1 + (X_1 + Y_1) C_0$$



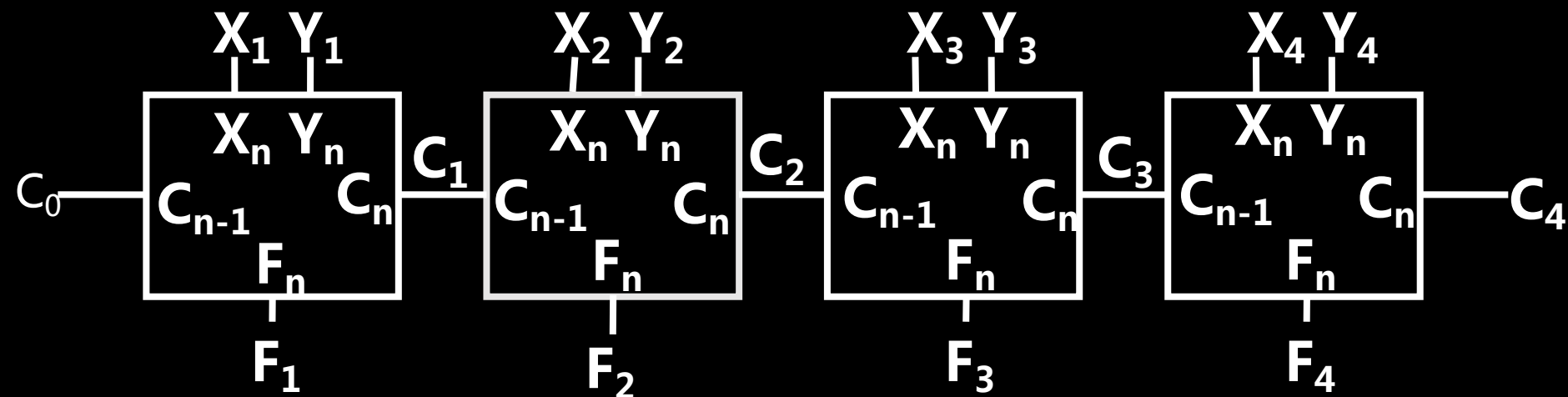
n位串行加法器的进位分析

2. C_2 形成的条件：

- X_2, Y_2 均为1
- X_2, Y_2 任一为1, X_1, Y_1 均为1
- X_2, Y_2 任一为1, X_1, Y_1 任一为1, 同时 C_0 为1

C_1 为1

$$C_2 = X_2 Y_2 + (X_2 + Y_2) X_1 Y_1 + (X_2 + Y_2) (X_1 + Y_1) C_0$$



n位串行加法器的进位分析

3. C_3 形成的条件：

➤ ...

4. C_4 形成的条件：

➤ ...

n位串行加法器的进位分析

➤ 令 $P_i = X_i + Y_i$, $G_i = X_i Y_i$ 则：

$$C_1 = X_1 Y_1 + (X_1 + Y_1) C_0$$

$$C_2 = X_2 Y_2 + (X_2 + Y_2) X_1 Y_1 + (X_2 + Y_2) (X_1 + Y_1) C_0$$

➤ 可化简为：

$$C_1 = G_1 + P_1 C_0$$

$$C_2 = G_2 + P_2 G_1 + P_2 P_1 C_0$$

➤ 同理 C_3 、 C_4 做同样化简。

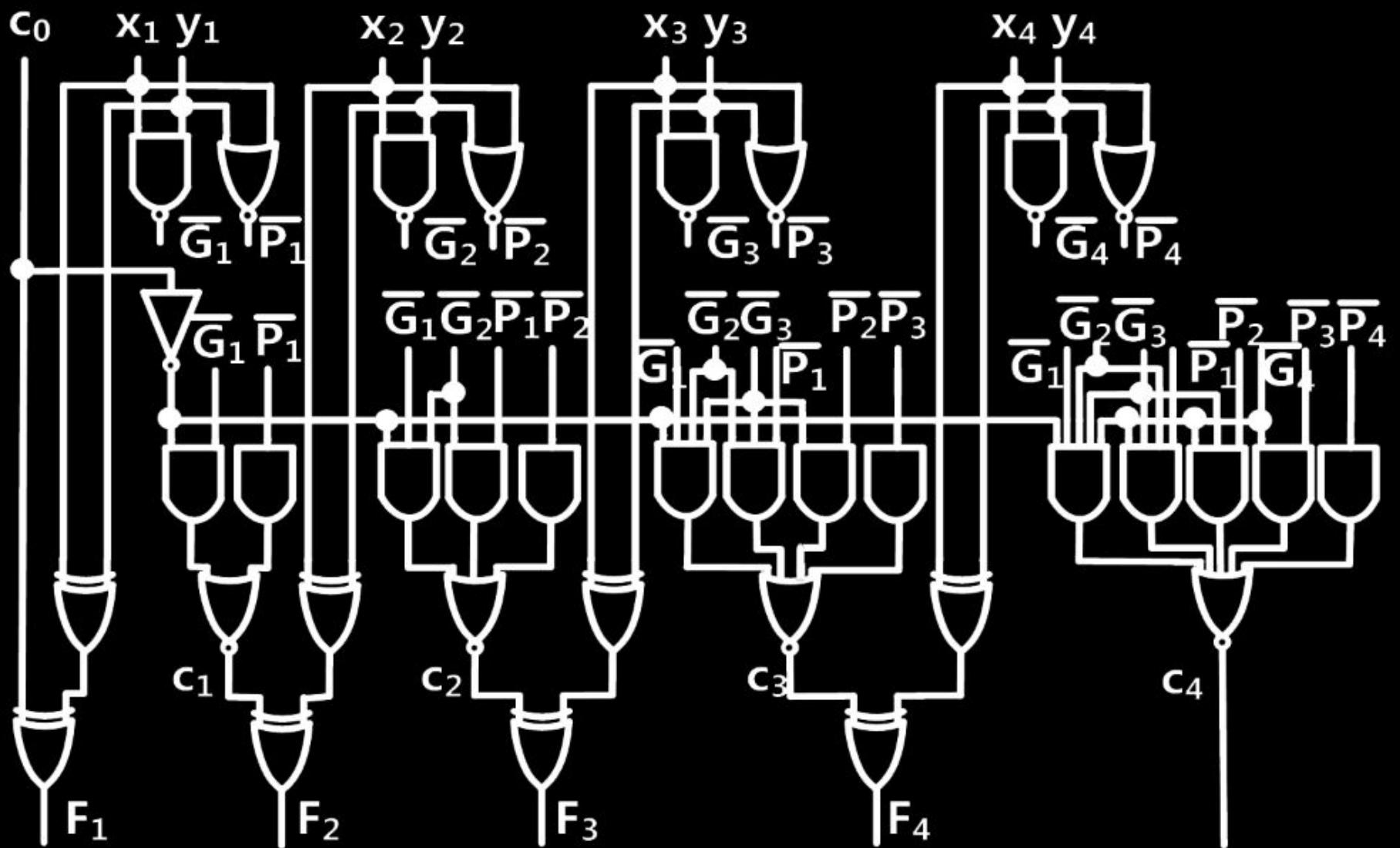
n位串行加法器的进位分析

- 进一步化简：

$$C_1 = G_1 + P_1 C_0 = \overline{\overline{P_1} + \overline{G_1} \overline{C_0}}$$

$$C_2 = G_2 + P_2 G_1 + P_2 P_1 C_0 = \overline{\overline{P_2} + \overline{G_2} \overline{P_1} + \overline{G_1} \overline{G_2} \overline{C_0}}$$

- 同理 C_3 、 C_4 做同样变化，用 P_i 、 G_i 的反变量来表示。



$$C_1 = G_1 + P_1 C_0 = \overline{\overline{P_1 + G_1 C_0}}$$

$$C_2 = G_2 + P_2 G_1 + P_2 P_1 C_0 = \overline{\overline{\overline{\overline{P_2 + G_2 P_1 + G_1 G_2 C_0}}}}$$

四、算术逻辑单元

1. ALU--进行多种算术运算和逻辑运算

2. 基本逻辑结构是超前进位加法器

➤ 四位算术逻辑单元SN74181基本原理

➤ M: 状态控制端

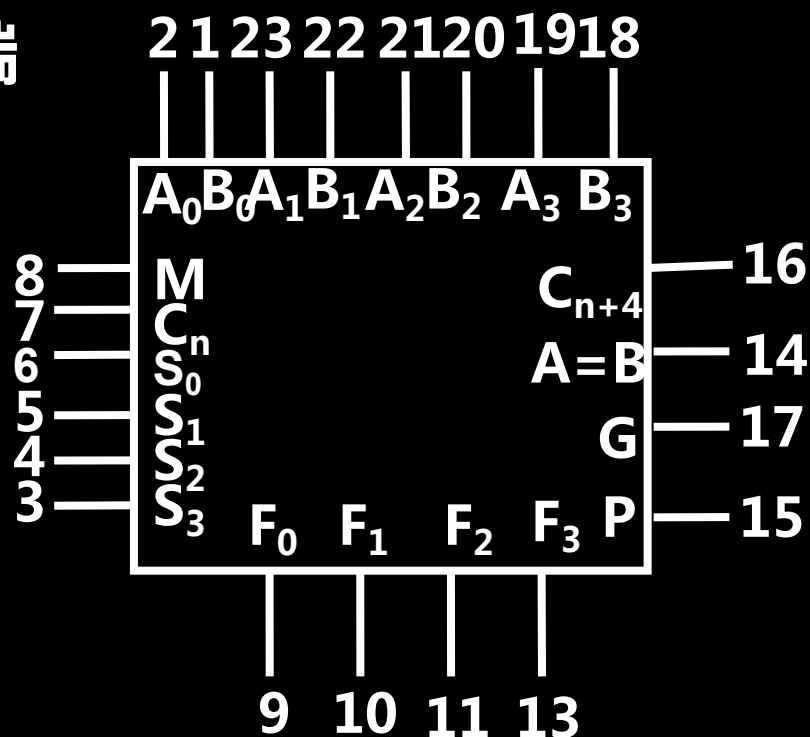
➤ S0~S3: 运算选择控制端

➤ A3~A0/B3~B0:

参加运算的两个数

➤ Cn:最低位进位输入

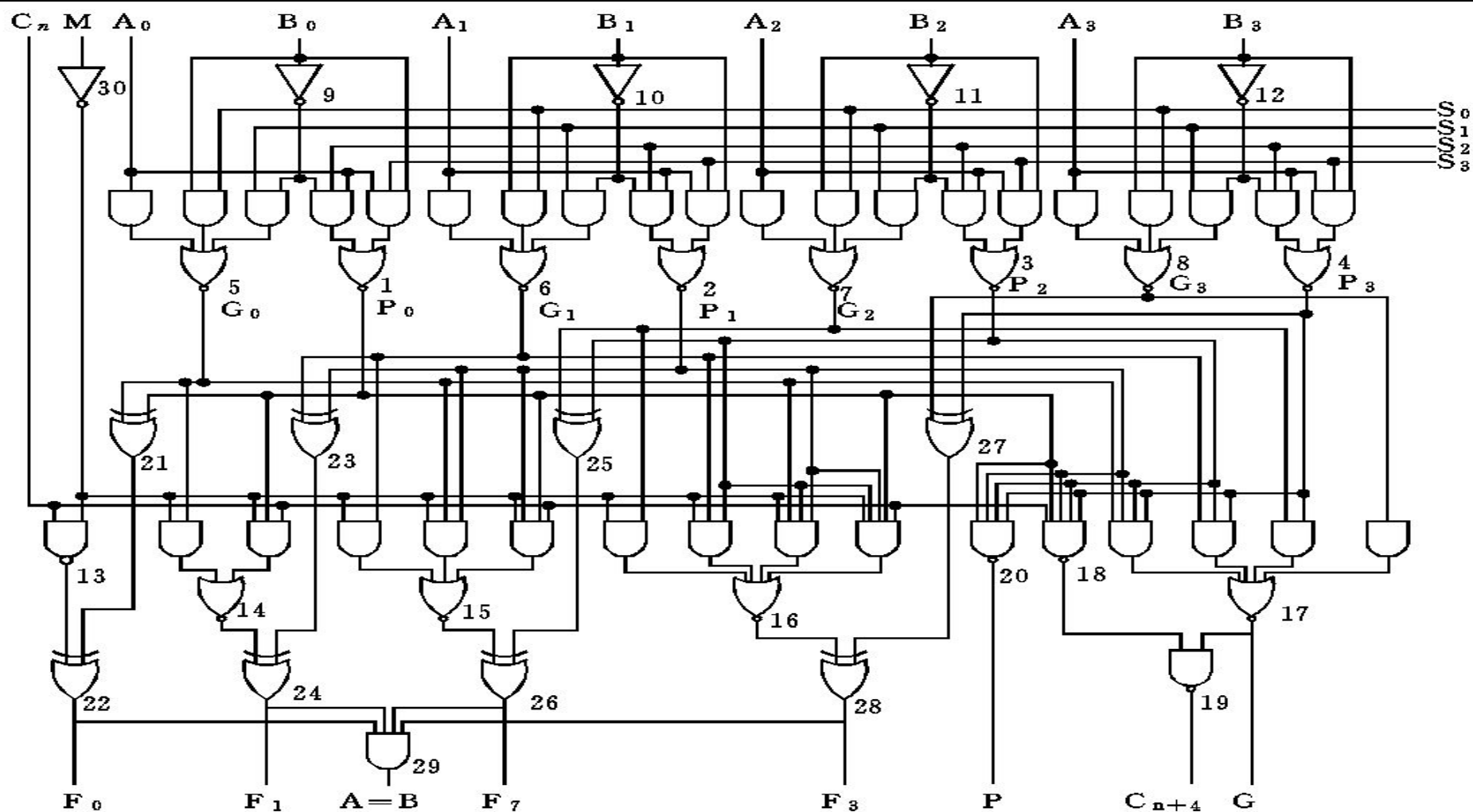
➤ F3 ~ F0:运算结果



四位算术逻辑单元功能表

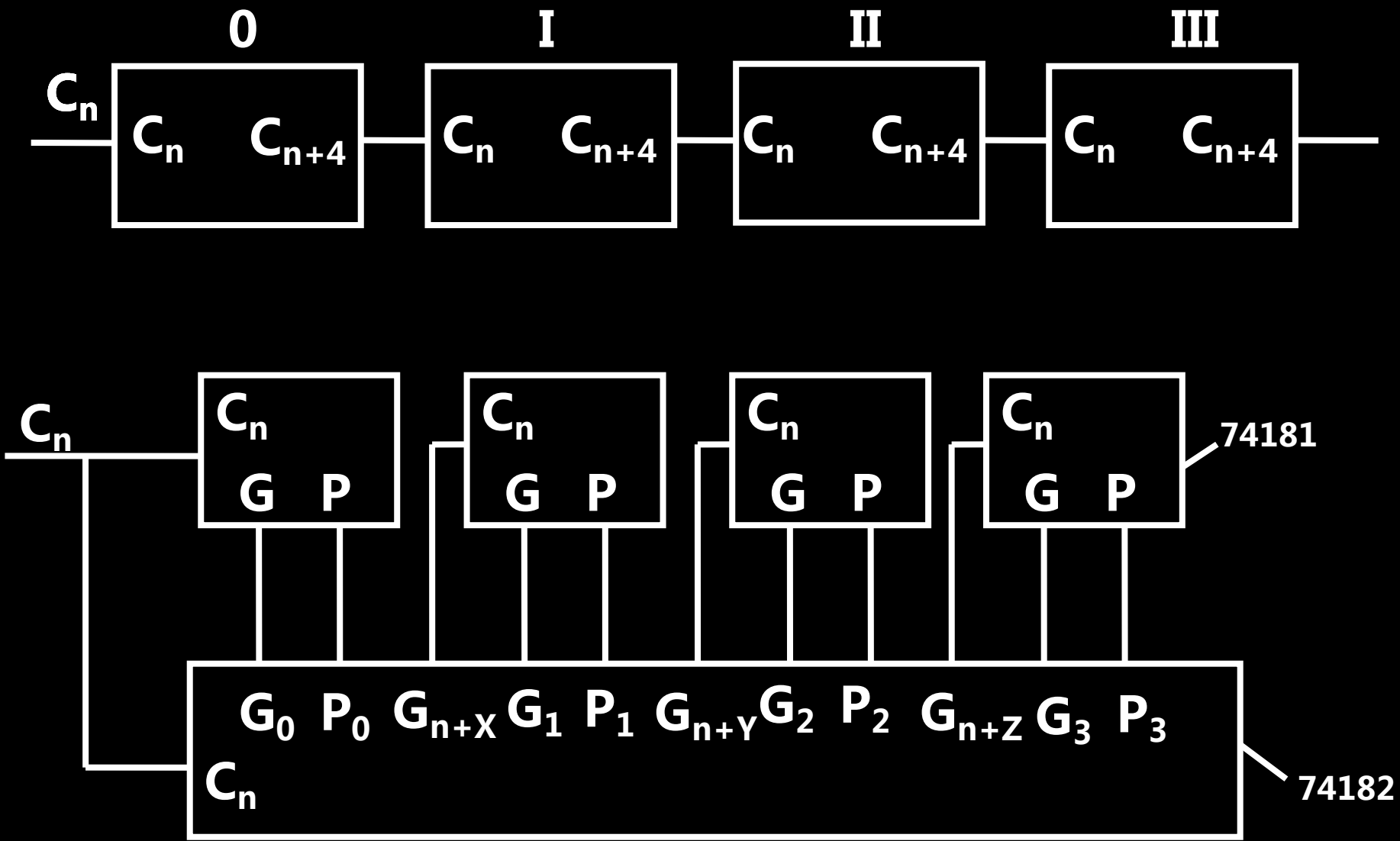
工作方式选择 $S_3S_2S_1S_0$				负 逻 辑		正 逻 辑	
				逻辑 运算 $M=H$	算数运算 $M=L, C_{-1}=0$	逻辑 运算 $M=H$	算术运算 $M=L, C_{-1}=1$
L	L	L	L	\overline{A}	A减1	\overline{A}	A
L	L	L	H	\overline{AB}	AB减1	$\overline{A+B}$	A+B
L	L	H	L	$\overline{A+B}$	\overline{AB} 减1	\overline{AB}	$A+\overline{B}$
L	L	H	H	逻辑1	减1	逻辑0	减1

四位算术逻辑单元逻辑电路图



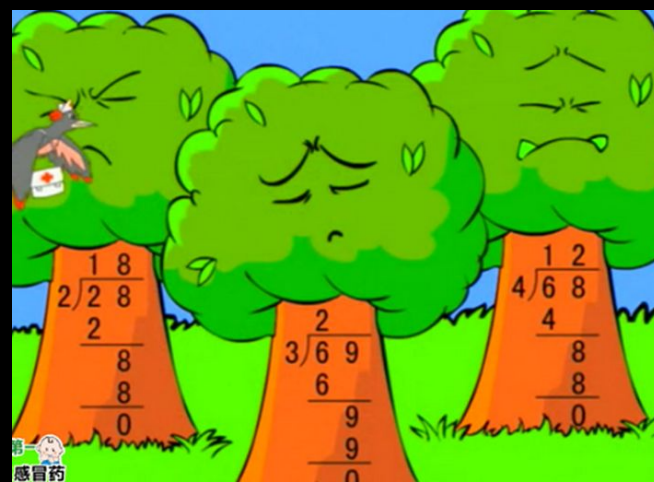
(a) 逻辑图

四位ALU功能表



推荐阅读：提高除法运算速度的方法

- 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。
- 除法运算通过乘法操作来实现：计算机运行时，执行乘法操作的几率比除法高，有些计算机没有专门的除法器，此时可利用乘法来完成除法运算以提高速度。





计算机组织与结构

大连理工大学 赖晓晨