计算机基础

第二章 数据的表示、运算和校验

2023/9/13

信息与软件工程学院
School of Information and Software Engineering

主 要 内 容

1. 数值型数据的表示方法
   - 进位计数制
   - 带符号数的表示
   - 定点数和浮点数
2. 字符的表示方法
3. 运算方法

   补码加减运算

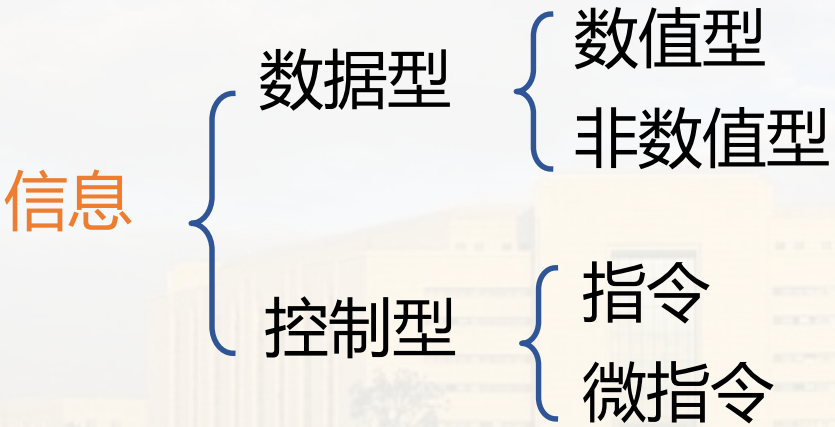
引言

重点：1、进位计数制及其相互转换

(二--八--十--十六进制间的转换)

2、机器数

3、IEEE754标准浮点表示格式



2.1 数值型数据的表示方法

- 01. 进位计数制
- 02. 带符号数的表示
- 03. 定点数和浮点数

一个数值型数据的完整表示需三个方面：

进位计数制、符号的数字化（机器数）、小数点的处理

数制的基和权

在任一数制中，每一个数位上允许使用的记数符号的个数（或每个数位中所允许的最大数码值+1）被称为该数制的基数。

每1位都对应1个表示该位在数码中的位置的值，这个值就称为数位的权值w。

[例] $1\underline{2}8_{10}$ $w=10^2$ $1\underline{1}01_2$ $w=2^3$

权：是一个与所在数位相关的常数。

权与基数的关系：相邻两位的权值之比等于基数值。

十进制计数

(1) 基数为十(计数的符号个数):0～9。

(2) 权值w:10^i

如果有m位整数，n位小数。则:

$$(S_{10}) = a_{m-1}10^{m-1} + a_{m-2}10^{m-2} + \cdots + a_0 10^0$$

$$+ a_{-1}10^{-1} + \cdots + a_{-n}10^{-n} = \sum_{i=-n}^{m-1} a_i 10^i \quad (a_i = 0 \sim 9)$$

例：$256.7 = 2\times10^2 + 5\times10^1 + 6\times10^0 + 7\times10^{-1}$

二进制计数

(1) 基数为二(计数的符号个数):0～1。

(2) 权值w:2^i

如果有m位整数，n位小数。则:

$$(S_2) = a_{m-1}2^{m-1} + a_{m-2}2^{m-2} + \cdots + a_0 2^0 + a_{-1}2^{-1} + \cdots + a_{-n}2^{-n}$$

$$= \sum_{i=-n}^{m-1} a_i 2^i \qquad (a_i = 0 \sim 1)$$

例：$(101.1)_2 = 1\times2^2 + 0\times2^1 + 1\times2^0 + 1\times2^{-1} = 5.5$

八进制计数

(1) 基数为八(计数的符号个数):0～7。

(2) 权值w:8^i

如果有m位整数，n位小数。则:

$$(s_8) = a_{m-1}8^{m-1} + a_{m-2}8^{m-2} + \cdots + a_0 8^0 + a_{-1}8^{-1} + \cdots a_{-n}8^{-n}$$

$$= \sum_{i=-n}^{m-1} a_i 8^i \quad (a_i = 0 \sim 7)$$

例：$(12.4)_8 = 1\times8^1 + 2\times8^0 + 4\times8^{-1} = 10.5$

十六进制计数

(1) 基数为十六(计数的符号个数):0～F。

(2) 权值w:16^i

如果有m位整数，n位小数。则:

$$(s_{16}) = a_{m-1}16^{m-1} + a_{m-2}16^{m-2} + \cdots + a_0 16^0 + a_{-1}16^{-1}$$

$$+ \cdots a_{-n}16^{-n} = \sum_{i=-n}^{m-1} a_i 16^i \qquad (a_i = 0 \sim F)$$

例：$(3A6)_{16} = 3\times16^2 + 10\times16^1 + 6\times16^0 = 934$

十六进制计数

二进制、十六进制、十进制对照关系表

二进制表示	对应十六进制	对应十进制	二进制表示	对应十六进制	对应十进制
0000	0	0	1000	8	8
0001	1	1	1001	9	9
0010	2	2	1010	A	10
0011	3	3	1011	B	11
0100	4	4	1100	C	12
0101	5	5	1101	D	13
0110	6	6	1110	E	14
0111	7	7	1111	F	15

二进制数特点

(1)二进制数只有0和1两个数码，故可以用晶体管的通、断或脉冲的有无来表示一位二进制数。

(2)二进制数运算规则简单，其特点是逢二进一，借一当二。

加法：0+0=0；0+1=1；1+0=1；1+1=10

减法：0-0=0；0-1=1；1-0=1；1-1=0

乘法：0×0=0；0×1=0；1×0=0；1×1=1

除法：0÷1=0；1÷1=1

一、进位计数制

例：1101 + 1011 = 11000

$$\begin{array}{r} 1101 \\ +1011 \\ \hline 11000 \end{array}$$

11101 − 10011 = 01010

$$\begin{array}{r} 11101 \\ -\ 10011 \\ \hline 01010 \end{array}$$

一、进位计数制

110×101= 11110

```
   110
 × 101
-------
   110
  000
 110
-------
 11110
```

10010001÷1011

=1101(余10)

```
        1101
    ------------
1011 | 10010001
       1011
    ------------
       1110
       1011
    ------------
        1101
        1011
    ------------
          10
```

二进制和十进制间的转换

(1) 十进制数转换为二进制数

整数部分：除以2取余数，直到商为0为止。

小数部分：乘以2取整数，直到小数为0(或到达要求精度)为止。

例　：$(29.6875)_{10}=(11101.1011)_2$

(2) 二进制数转换为十进制数

按权位展开求和。

例：$(11.1)_2 = 1\times2^1+1\times2^0+1\times2^{-1}=3.5$

一、进位计数制

(1) $29_{10} \rightarrow X_2$ 除2取余

2 | 29		
2 | 14	→ 1	↑ 低位
2 | 7	→ 0	
2 | 3	→ 1	
2 | 1	→ 1	
0	→ 1	↑ 高位

$X_2=11101_2$

(2) $0.6875_{10} \rightarrow X_2$ 乘2取整

$0.6875 \times 2 = 1.375$ → 1 （高位）

$0.375 \times 2 = 0.75$ → 0

$0.75 \times 2 = 1.5$ → 1

$0.5 \times 2 = 1.0$ → 1 （低位）

$X_2=0.1011_2$

(3) $1101.11_2 \rightarrow X_{10}$ 按权相加

$$X_{10}=1\times2^3+1\times2^2+0\times2^1+1\times2^0+1\times2^{-1}+1\times2^{-2}$$

$$=13.75_{10}$$

八进制、十六进制与二进制数的转换

(1) 二进制数转换为八进制数

从小数点起三位一组，整数部分不够三位的向前添0，小数部分不够三位的向后添0。

例1: $(1\ \underline{011}\ \underline{101.011}\ \underline{010}\ 1)_2 = (135.324)_8$

(2) 二进制数转换为十六进制数

从小数点起四位一组，整数部分不够四位的向前添0，小数部分不够四位的向后添0。

例2：$(101\ \underline{1101.0110}\ 101)_2 = (5D.6A)_{16}$

八进制、十六进制与二进制数的转换

(3) 八进制数和十六进制数转换为二进制数

按(1)(2)的逆过程进行转换。

例2：$(5D.6A)_{16}$ =(0101 1101 . 0110 1010)$_2$

十进制数与八进制数、十六进制数间的转换

(1) 十进制数转换为八进制数、十六进制数

整数部分除以8、16取余数，直到商为0止。小数部分乘以8、16取整数，直到小数为0或到要求精度止。

(2) 八进制数、十六进制数转换为十进制数

按权位展开求和。

一、进位计数制

例1：$(369)_{10}=(561)_8=(171)_{16}$

		余数	
8	369		
8	46	1	a_0
8	5	6	a_1
	0	5	a_2

		余数	
16	369		
16	23	1	a_0
16	1	7	a_1
	0	1	a_2

例2：$(561)_8=(369)_{10}$

$(561)_8=5\times8^2+6\times8^1+1\times8^0=5\times64+6\times8+1=369$

例3：$(171)_{16}=(369)_{10}$

$(171)_{16}=1\times16^2+7\times16^1+1\times16^0$

$=1\times256+7\times16+1=369$

真值与机器数

若以正号“+”和负号“-”来表示有符号的二进制数，称为符号数的真值。

符号数的真值+0.1011；-0.1011，这种表示方法不能直接用于计算机中。只有使符号数值化以后，才可以在计算机中使用，符号位习惯以0表示正数，以1表示负数。

计算机中使用的符号数称为机器数。

如+1011表示为01011，而-1011表示为11011。

真值与机器数

```
   110
× 101
------
   110
 000
110
------
11110
```

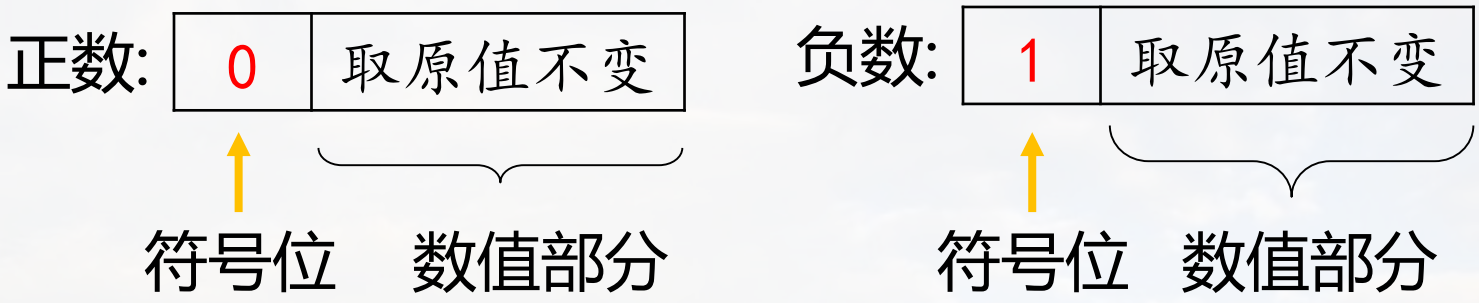
前面介绍的二进制数的加、减、乘、除运算，乘法运算实际上是作移位加法运算；除法运算则可用移位减法来完成。

注意：作减法时，必须先比较两个数绝对值的大小，将绝对值大的数减去绝对值小的数，最后再在运算结果前加上正确的符号。故作减法运算所需电路复杂，耗时长。

为了能变减法为作加法，下面提出了原码、反码和补码三种机器数的表示方法。

原码

编码规则：最高位为符号位，其余各数值位取原值不变。



原码

原码又称“符号 - 数值表示”，在以原码表示的正负数中，第一位为0(正数)；为1(负数)。

(1) 若二进制整数的原码序列为:$X_nX_{n-1}\cdots\cdots X_0$(其中n是符号位)则:

$$X_{原}=\begin{cases} X & 2^n>X\geq 0 \\ 2^n-X=2^n+|X| & 0\geq X>-2^n \end{cases}$$

二、带符号数的表示

[例] 138和-138

真值 +10001010　　-10001010

原码 0 10001010　　1 10001010

字长为8位的原码

表示范围为：-127～+127

$[+127]_{原}$ =0 1111111

$[-127]_{原}$ =1 1111111

数值“0”有两种原码形式：

$[+0]_{原}$=0 0000000

$[-0]_{原}$=1 0000000

原码符号位不是数值的一部分，它们是人为约定的，0为正，1为负。所以符号位在运算中要单独处理，不能当作数值的一部分直接参加运算。

反码

反码又称“1的补码”，用反码表示时，左边的第一位也为符号位，0代表正数，1代表负数。对于负数，反码的数值是将原码数值部分按位求反，符号位1保持不变。而对于正数，反码和原码相同。

反码的一般表示

(1) 若二进制整数形式为$X_nX_{n-1}\cdots\cdots X_0$(其中n是符号位)则：

$$X_{反}=\begin{cases} X & 2^n>X\geq 0 \\ (2^{n+1}-1)+X & 0\geq X>-2^n \end{cases}$$

反码

① 对于正数(设字长为8位)

$[X]_{反}=[X]_{原}$ （$X\geq 0$）

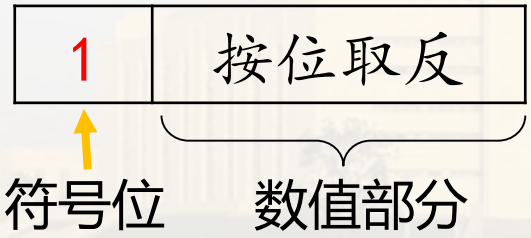
设： X=+1101001 （+105），则 $[X]_{反}$= 01101001

↑

符号位 数值位

② 对于负数

符号位仍为“1”，各数值位“按位取反”。



二、带符号数的表示

[例] X= -1101001 (真值-105)

$X_{原}$=1 1101001

$X_{反}$=1 0010110

字长为8位的反码

“0”的两种表示形式:

$[+0]_{反}$ =0 0000000

$[-0]_{反}$ =1 1111111

8位反码表示的数值范围为：+127～-127。

$[+127]_{反}$ =0 1111111

$[-127]_{反}$ =1 0000000

补码

补码又称“对2的补数”，补码表示法是：如果数为正，则正数的补码与原码表示形式相同；如果数为负，则将负数的原码除符号位外，其余各位取反后末尾再加1。

例：$X_1 = +10011$表示为 $X_{1补}=010011$

$X_2 = -01010$表示为 $X_{2补}=110110$

补码

时钟以12为计数循环，即以12为模。13点在舍去模12后，即为1点。从0点出发，反时针拨1格即为 - 1点，也可看成从0点顺时针拨11格，即11点。换句话说，在模12前提下， - 1可映射为 + 11。

引入补码表示法的目的是：让符号位也作为数值的一部分直接参与运算，以简化加、减运算的规则，同时又能化减为加！

补码

确定模以后，我们将某数X对该模的补数称作其的补码。定义如下：

$$X_{补} = M + X \quad (模M)$$

若X>0，则模M作为正常的溢出量可以舍去。如同时钟一例舍去12一样。因而正数的补码就是其本身，形式与原码相同。

例：若 X = +101（4位二进制整数）

则 $X_{补} = 10000 + 0101 = 0101$ $(模2^4=16)$

若X<0，则$X_{补} = M + X = M-|X|$。如同时钟一样，-1点的补码为+11点。

例：若 X = -101（4位二进制整数，原码为1101）

则 $X_{补} = 10000-0101 = 1011$ (模16)

补码

① 对于正数(字长=8位)

$[X]_{补}=[X]_{原}$ (即X≥0时)

② 对于负数(字长=8位)

符号位仍为“1”，各数值位“按位取反,末位再加1”，即:

$[X]_{补}=[X]_{反}+...1$ (即X<0时)

补码

补码的一般表示

若定点整数的补码序列为 $X_nX_{n-1}\cdots\cdots X_0$(其中n是符号位)则:

$$X_{补}=\begin{cases} X & 2^n>X\geq 0 \\ 2^{n+1}+X=2^{n+1}-|X| & 0>X\geq -2^n \end{cases}$$

例: - 10101的补码为100000 - 10101=101011 (n=5)

补码

“0”的表示形式：

$[+0]_{补}=[-0]_{补}=$ 0 0000000

8位补码，表示范围为：+127～-128

即：

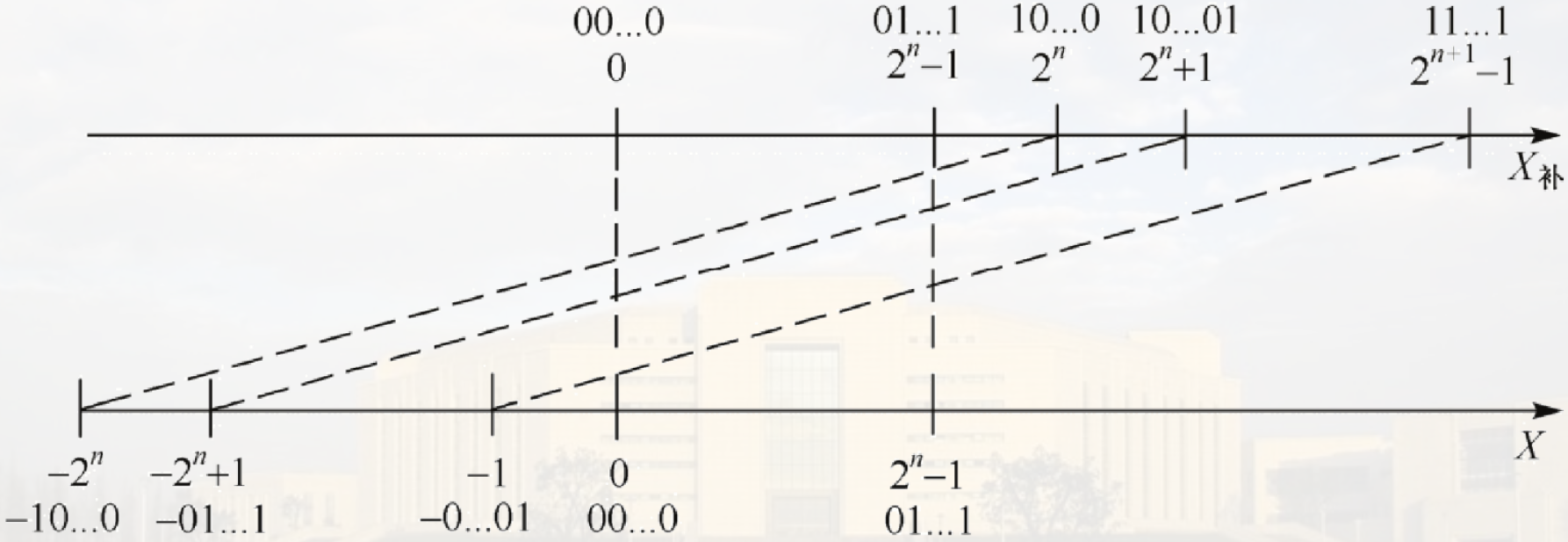
$[+127]_{补}$ =0 1111111；

$[-128]_{补}$ =1 0000000

【注意】8位补码可以比原码、反码多表示一个负数，即-128

补码

将负数X的真值与其补码X补作一映射图，可以进一步看出：负数补码表示的实质是将负数映射到正数域，因而可实现化减为加，达到简化运算的目的。



$2^{n+1}+X = 2^{n+1} - |X|$　　$0>X \geq -2^n$

补码

总结

1. 在补码表示中仍以最高位作为符号位，“0正1负”，这点与原码相同。但补码的符号值由补码定义式计算而得，它是数值的一部分，可以与尾数一起直接参与运算，不需要单独处理。
2. 在补码表示中，数值0只有一种表示，即00...00。也就是说：在原码中有“-0”，而补码中没有“-0”。
3. 负数补码的表示范围比原码稍宽一点。

二、带符号数的表示

二进制数	无符号数	原　　码	反　　码	补　　码
0000 0000	0	+0	+0	+0
0000 0001	1	+1	+1	+1
0000 0010	2	+2	+2	+2
⋮	⋮	⋮	⋮	⋮
0111 1110	126	+126	+126	+126
0111 1111	127	+127	+127	+127
1000 0000	128	−0	−127	−128
1000 0001	129	−1	−126	−127
1000 0010	130	−2	−125	−126
⋮	⋮	⋮	⋮	⋮
1111 1101	253	−125	−2	−3
1111 1110	254	−126	−1	−2
1111 1111	255	−127	−0	−1

码制转换

(1) 已知$[X]_{原}$，求$[X]_{补}$

【例】已知$[X]_{原}$= 1 0011010，求$[X]_{补}$

解：

$[X]_{原}$ =	1	0	0	1	1	0	1	0
	↓	↓	↓	↓	↓	↓	↓	
	1	1	1	0	0	1	0	1
+)								1
$[X]_{补}$ =	1	1	1	0	0	1	1	0

二、带符号数的表示

(2) 已知$[X]_{补}$，求$[X]_{原}$

$[[X]_{补}]_{补}=[X]_{原}$

【例】已知$[X]_{补}$= 1 1101100，求$[X]_{原}$

解：

$[X]_{补}$= 1 1 1 0 1 1 0 0

↓ ↓ ↓ ↓ ↓ ↓ ↓

1 0 0 1 0 0 1 1

+) 1

$[X]_{原}$ = 1 0 0 1 0 1 0 0

二、带符号数的表示

(3) 求补(变补)：已知$[X]_{补}$，求$[-X]_{补}$

$[X]_{补}$连同符号位一起变反，末位加1，得到$[-X]_{补}$。

【例】已知[X]补= 0 1 0 10110，求[-X]补

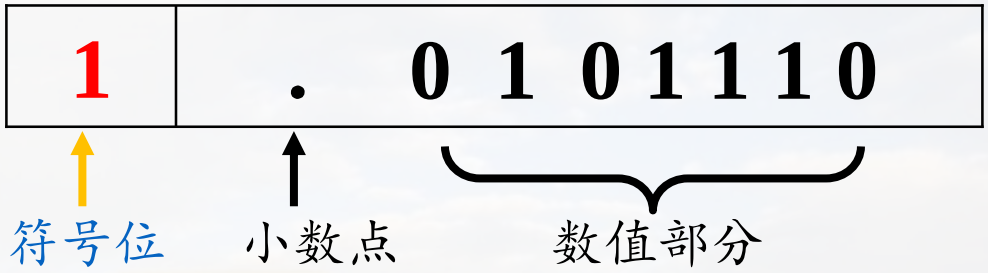
解：

[X]补 =	0	1	0	1	0	1	1	0
	↓	↓	↓	↓	↓	↓	↓	↓
	1	0	1	0	1	0	0	1
+)								1
[−X]补 =	1	0	1	0	1	0	1	0

定点数的表示

由程序设计者约定，该程序中所有数的小数点固定在同一位置不变。

① 带符号的定点小数：约定所有数的小数点的位置固定在符号位之后。



- 设字长= n+1位（$X_0.X_1.....X_{n-1}X_n$）
- 原码表示范围为：$-(1-2^{-n})$ ~ $1-2^{-n}$
- 补码表示范围为：-1 ~ $1-2^{-n}$
- 分辨率（精度）：2^{-n} 精细程度，如尺子的最小刻度值

定点数的表示

(2) 若定点小数的补码序列为 $X_0.X_1\cdots\cdots X_n$则:

$$X_{补}=\begin{cases} X & 1>X\geq 0 \\ 2+X=2-|X| & 0>X\geq -1 \end{cases}$$

例: - 0.1010的补码为10 - 0.1010 = 1.0110

三、定点数和浮点数

② 带符号的定点整数：约定所有数的小数点的位置固定在最低数值位之后。

设字长=n+1位

原码的表示范围为：$-(2^n-1) \sim 2^n-1$。

补码的表示范围为：$-2^n \sim 2^n-1$。

分辨率（精度）：1

③ 无符号定点整数：约定所有数的小数点的位置固定在最低数值位之后。

若代码序列为$X_nX_{n-1}...X_1X_0$，共n+1位，则有：

典型值	真值	代码序列
最大正数	$2^{n+1}-1$	11...11
最小非零正数	1	000001

原码、补码和反码相同，表示范围为:0 ~ $(2^{n+1}-1)$,

分辨率为: 1

特别注意: 定点数的小数点位置固定是一种人为的约定.不需要在计算机中设置专门的硬件来表示。

三、定点数和浮点数

如用8位二进制数表示，则其表示范围如下：

无符号数：00000000～11111111　　0～255

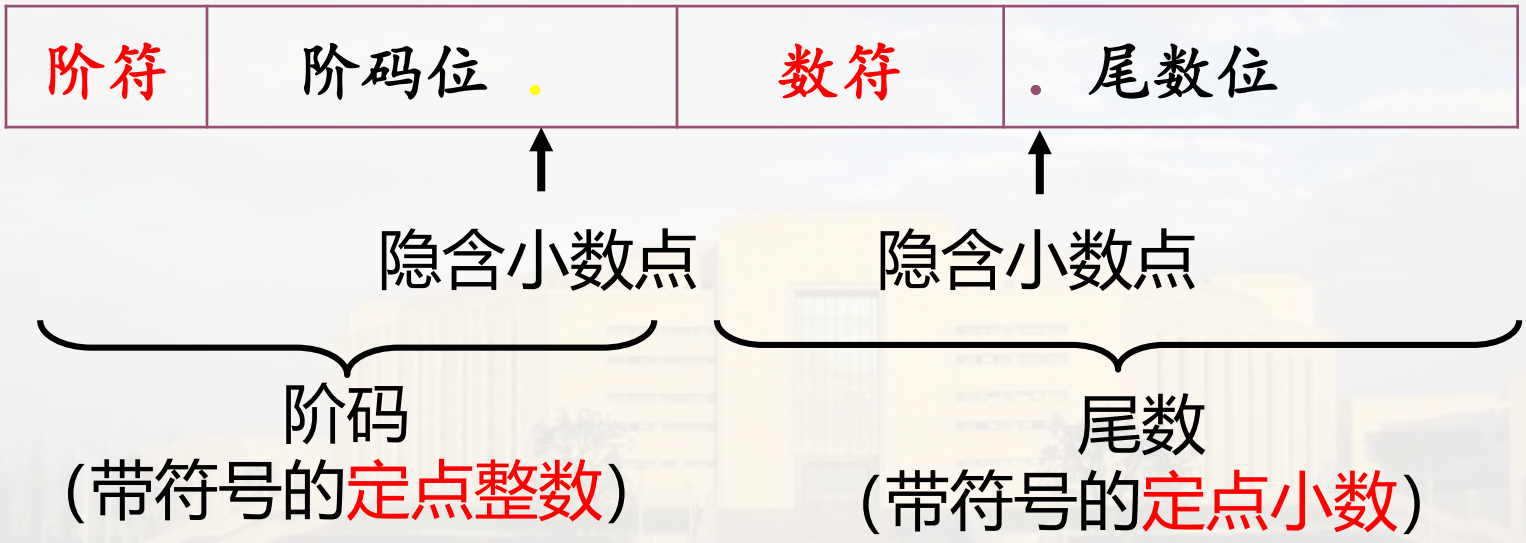
定点整数：
- 11111111原～01111111原　　-127～127
- 10000000补～01111111补　　-128～127

定点小数：
- 1.1111111原～0.1111111原　　$-(1-2^{-7})$～$(1-2^{-7})$
- 1.0000000补～0.1111111补　　-1～$(1-2^{-7})$

浮点数的表示

浮点表示法中，小数点的位置可按需要浮动。

表示如下：



引入浮点数的意义

[例] 某字长为8位的原码二进制数

定点数：11111111 ~ 01111111

整数：-127 127 精度：1

小数：$-(1-2^{-7})$ $1-2^{-7}$ 精度：2^{-7}

浮点数：5位阶码+3位尾数

01111111 ~ 01111011

$2^{15}\times(-0.75) \sim 2^{15}\times 0.75$

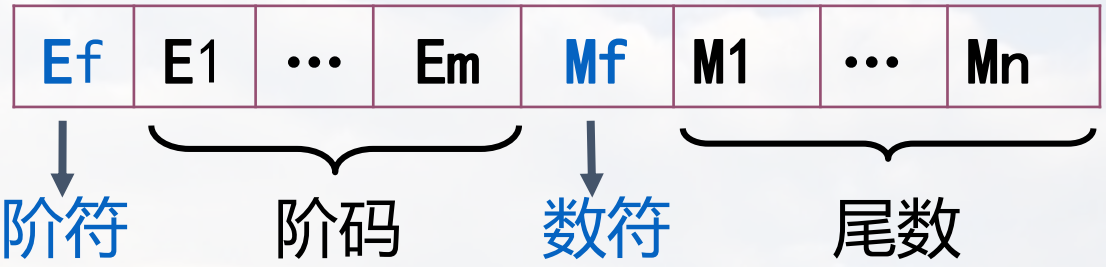
精度：11111001 $=2^{-15}\times 0.25$

相同字长时，浮点数的表示范围更大、精度更高！

三、定点数和浮点数

浮点数真值：$N = \pm R^E \times M$

浮点数机器格式：



R：阶码底，隐含约定（与尾数M的基数相同），一般为2。

E：阶码，为定点整数，补码或移码表示。
其位数决定数值范围；阶符表示阶码的正负。

M：尾数，为定点小数，原码或补码表示。
其位数决定数的精度； 数符表示数的正负。

1、移码

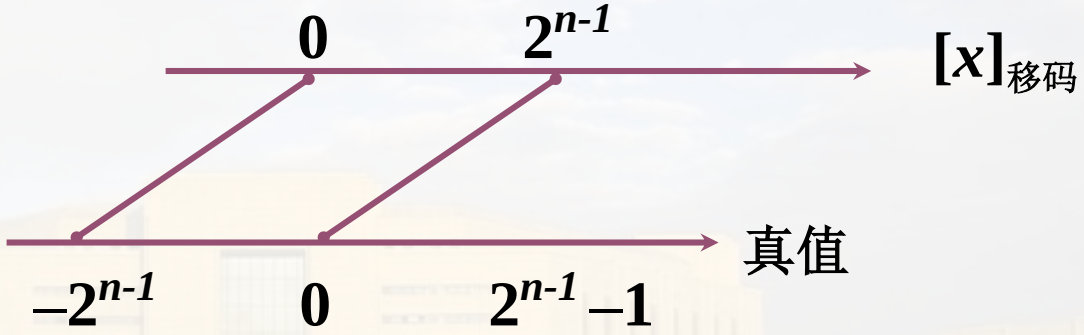
通常用于表示浮点数的阶码。

阶码一般为整数，故移码通常只用于表示整数

对定点整数x，它的移码是：

$$[x]_{移} = 2^{n-1} + x，其中 -2^{n-1} < x < 2^{n-1}$$

这里的n为$X_{原}$位数（包含符号位和数值位）



上述规则等价于将x正向平移或者增加2^{n-1}，因此称之为移码或增码。

三、定点数和浮点数

补码表示：　　难以直接判断真值大小

如　十进制	二进制真值	补码	
$x = +21$	+10101	0 10101	
$x = -21$	– 10101	1 01011	> ?
$x = +31$	+11111	0 11111	
$x = -31$	– 11111	1 00001	> ?

若：$x + 2^5$(偏置值)

+10101 + 100000 = 110101　>

–10101 + 100000 = 001011

+11111 + 100000 = 111111　>

–11111 + 100000 = 000001

[例]阶码的为6位，X表示其真值

$$X_{移} = 2^5 + X \quad (-2^5 < X < 2^5)$$

当正数X = +10101时, $X_{移} = 2^5 + X = 110101$

当负数X = -10101时, $X_{移} = 2^5 + X$

$= 2^5 - 10101$

$= 001011$

移码表示范围与补码一致，0也只有1个移码。

正数：将原码符号位变反，即得到移码。

负数：将原码连同符号位一起变反，末位再加1，即得到移码(与变补等效)。

补码和移码: 符号相反、数值位相同

2、尾数M的规格化表示

规格化的目的 → 使浮点数的表示代码“唯一”

$128=\underline{0.128}\times10^{3}=\underline{1.28}\times10^{2}=\underline{12.8}\times10^{1}$ ，…

科学计数法约定：$1\le|M|<10$

则规范形式：$128=1.28\times10^{2}$ （唯一）

$3.2=1.6\times2^{1}=0.8\times2^{2}=0.4\times2^{3}$, …

可以表示成任意多个代码形式 → 计算不便

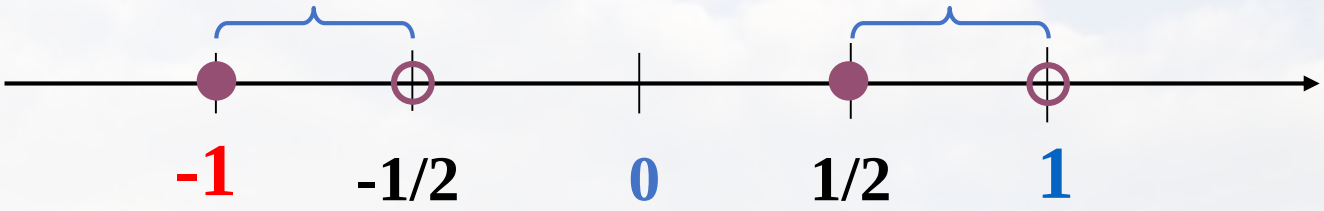
约定尾数M的值域，使数的表示是唯一的、确定的

→尾数M的“规格”化

三、定点数和浮点数

① 浮点数用原码表示时　$1/2 \leqslant |M| < 1$

② 浮点数用补码表示时　$-1 \leqslant M < -1/2$　或　$1/2 \leqslant M < 1$



对于原码：　规格化以后尾数的最高有效位为“1”

$M_{原}$=0.1000，1.1010　　$M_{原}$=0.0010，1.0110　非规格化

对于补码：

正数，规格化后最高数值位为“1”　如0.1010，0.1110

0.625　0.875

负数，规格化后最高数值位为“0”　如1.0010，1.0000

-0.875　-1

规格化后，尾数的最高数位必须是一个有效值。

三、定点数和浮点数

[例1]阶符1位、阶码m位，数符1位、尾数n位，如果表示成规格化**补码**，分析各真值对应的M、E取值情况。

$N = M \times 2^E = F(M, E)$

$$\begin{cases} -1 \leqslant M < -1/2 \text{或} 1/2 \leqslant M < 1 \\ -2^m \leqslant E \leqslant 2^m - 1 \end{cases}$$

最小浮点数	E→	为最大正数： $2^m - 1$
	M→	为最小负数： -1
最大浮点数	E→	为最大正数： $2^m - 1$
	M→	为最大正数： $1-2^{-n}$
最小浮点正数	E→	为最小负数： -2^m
	M→	为最小正数： 2^{-1}
最大浮点负数	E→	为最小负数： -2^m
	M→	为最大负数： $-(1/2+2^{-n})$

[例2]某个用规格化补码表示的浮点数，其中阶码6位(含1位阶符),尾数10位(含1位数符)。

表示范围：　011111 100…00 ~ 011111 011…11

$2^{31} \times (-1)$ ~ $2^{31} \times (1-2^{-9})$

绝对精度（最小的非0正数）：

100000 010…00　$=2^{-32} \times 0.5 = 2^{-33}$

相对精度（只与尾数的数值位数有关，即位数的精度）：

xxxxxx 000…01　$=2^{-9}$

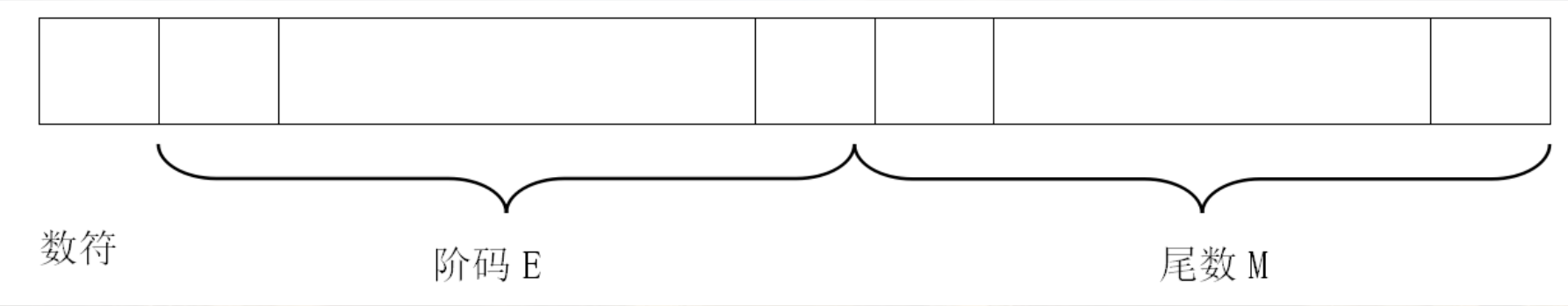
[思考]某十六进制浮点数$A3680000_{16}$，将其表示成补码，字长32位，阶码8位（含1位阶符），尾数24位（含1位数符），求该浮点数十进制的真值。

IEEE754格式的浮点数

IEEE754标准浮点格式

前面讨论的是原理性浮点格式，但实际计算机的浮点格式与此有一些差异。下面简要介绍在当前主流微机中广泛采用的IEEE754标准浮点格式。

按IEEE754标准，常用的浮点数的格式如下图所示：



三、定点数和浮点数

IEEE754有3种浮点表示格式，分别称为：短浮点数(或称短实数、或称单精度浮点数)、长浮点数（了解）（或称长实数、或称双精度浮点数）、临时浮点数（了解）（或称临时实数、或称扩展精度浮点数）。

类型	数符（位）	阶码（位）	尾数数值（位）	总位数（位）	偏置值	
					十六进制	十进制
短浮点数	1	8	23	32	7FH	127
长浮点数	1	11	52	64	3FFH	1023
临时浮点数	1	15	64	80	3FFFH	16383

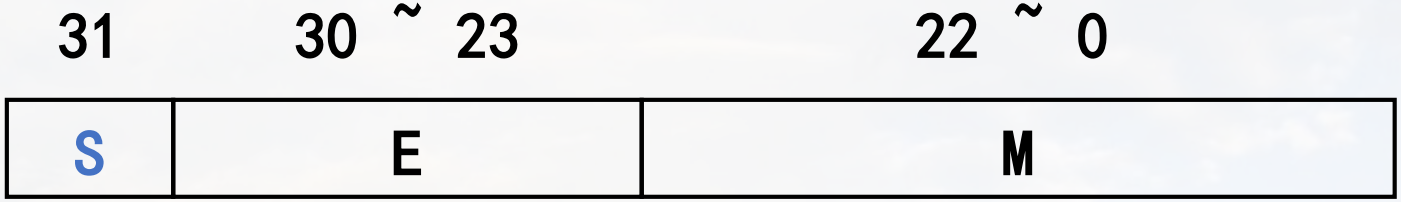
短浮点数又称为单精度浮点数，长浮点数又称为双精度浮点数，它们都采用隐含尾数最高数位（2^0）的方法，这样，无形中又增加了一位尾数，因此，相应地尾数真值实际上等于1+（23位尾数数值或52位尾数数值）。

临时浮点数又称为扩展精度浮点数，它没有隐含位，尾数真值就等于64位尾数数值。

IEEE754格式的浮点数

有32位浮点数（单精度）和64位浮点数（双精度）

① 32位短浮点数：



在上述的表示格式中：

- S =浮点数的符号位，0表示正数，1表示负数；
- E =阶码，8位，采用移码表示，阶符隐含；
- M=尾数，23位，纯小数表示，且真值=1+M（原码形式）；
- 阶码E采用移码形式，但只偏移2^7-1（不是2^7）。

几点注释：

阶码是以移码形式存储的。

短浮点数的偏置值为十进制127或十六进制7FH；

长浮点数的偏置值为十进制1023或十六进制3FFH；

临时浮点数的偏置值为十进制16383或十六进制3FFFH。

几点注释：

存储浮点数阶码部分之前，偏置值先要加到阶码真值上。

若阶码真值为3，在短浮点数中，移码表示的阶码为：十进制127+3=130或十六进制82H；

长浮点数中，移码表示的阶码为：十进制1023+3=1026或十六进制402H；

临时浮点数中，移码表示的阶码为：十进制16383+3=16386或十六进制4002H。

几点注释：

- 为了确保浮点数表示的唯一性，约定：$0 \le M < 1$。
- 浮点数所表示的范围显然远比定点数大。
- 注意E为全0/1的三种情况：

  （1）E全0，M全0时，F=0；

  （2）E全1，M全0时，F为±∞

  （3）E全1，M非全0时，F为无效数(NAN)；
- 隐含约定尾数的最高位2^0，即尾数位1.M

  所以规格化尾数的真值：$1 \le M_{真} < 2$

  $F_{真}=(-1)^S \times 1.M \times 2^{E-127}$

[例] 将十进制数20.59375转换成IEEE754的32位标准浮点数的2进制格式，并写出相应的16进制数。

[解]首先分别将整数和小数部分转换成二进制：

20.59375 = 10100.10011

然后移动小数点，使其在第1、2位之间

10100.10011 = $1.0100\,10011 \times 2^4$

小数点被左移了4位，于是得到：$e=4$

数符S =0，阶码E =4 + 127 =131，尾数M=010010011

最后得到32位浮点数的二进制代码：

0100 0001 1010 0100 1100 0000 0000 0000

= $(41A4C000)_{16}$

例:IEEE754单精度浮点数:C0A00000H的十进制值是多少

$(C0A00000)_{16}$ =(1100,0000,1010,0000, 0000, 0000, 0000,0000)$_2$

可得, 符号位S是1

阶码位E是10000001→$(10000001)_2=(129)_{10}$

尾数M是: 010,0000, 0000,0000,0000,0000

→$(0.\ 01000000000000000000000)_2=(0.25)_{10}$

因此,由公式 $(-1)^S \times 1.M \times 2^{E-127}$得

$(-1)^1 \times 1.25 \times 2^{129-127}=-1.25\times4=-5$

IEEE754表示法与真值的转换答题要点：

一、十进制真值转换为IEEE754，按以下步骤答题：

① 首先十进制数转换为二进制数一定要正确，如果这步都出错，后面的步骤就全错，基本没有得分了，需要务必小心。

② 写出符号位S的值。

③ 然后移动小数点，使其在第1、2位之间，根据移位写出阶码，注阶码是移动的位数+127。

④ 写出尾码和阶码的二进制形式。

⑤ 组合称为完整的IEEE754格式的浮点数形式。

IEEE754表示法与真值的转换答题要点：

二、 IEEE754转换为十进制真值，按以下步骤答题：

① 首先把十六进制数按二进制形式转换，这步务必细心，确保正确，否则后面的得分就都没有了。

② 根据二级制形式分离出S、E、M等各部分。

③ 根据E计算出e值。

④ 根据e值和M，写出真值的二进制表达形式。

⑤ 根二进制表达，按权位展开的方式，计算对应的10进制值（注：这步如果计算实在比较麻烦，可以只写出计算公式）。

三、定点数和浮点数

1. 某浮点数用IEEE754表示为2AB02700H，求其十进制的真值，并写出转换过程。
2. 将（52.25)10转换成IEEE754短浮点数格式，并写出转换过程。

1.2 字符的表示方法

- 01. ASCII码
- 02. 汉字编码简介（了解）

非数值型数据：一类是字符及以字符为基础的数据信息，另一类是可以数字化的、范围极其广泛的如逻辑信息、图形、图像、声音等等静态的或动态的信息。

本节内容是介绍字符型信息的表示方法。字符是非数值型信息的表示基础，也可以用它来间接描述范围广泛的绝大多数信息。

一、ASCII码

国际上广泛采用美国信息交换标准码（American Standard Code For Information Interchange）作为标准，简称为ASCII码。

在ASCII字符集中共有128种常用字符，每个ASCII字符自身有七位编码，存储器中的一个字节单元正好可以存放一个ASCII字符编码与一位奇偶校验位。这样，存储器也就常以字节为一个存储单元的编址单元。

二、汉字编码简介

汉字也是字符，是中文的基本组成单位。由于汉字数量极大，接近7万个，字形复杂、异体字多、同音字多，因此，汉字信息的处理比西文复杂得多。汉字信息的处理一般包括汉字的编码、输入、输出、存储、处理与传输。

在一个汉字信息处理系统的不同部位，需使用下述三类编码：输入码、内部码、交换码。

1、汉字输入码

每个汉字用一个或几个键输入的编码来表示。这种编码方式称为汉字的输入编码，也称外码。

目前汉字的编码方案有几百种之多，较常使用的也有几十种之多。归纳起来，所采用的方法可分为数字码、拼音码、字形码、音形结合、以及具有某些提示和联想功能的编码等几类。

2、汉字内部码

汉字内部码也称机内码，它是计算机内部供存储、处理、传输用的代码，简称内码，是汉字在设备或信息处理系统内部最基本的表达。目前国内采用的内部码有四十种左右。

3、汉字交换码

如前所述，早期的各种汉字系统的内码不统一，因此在各汉字系统之间或汉字系统与通信系统之间进行汉字信息交换（即传输）时，需要制定一种编码标准，即汉字交换码。

我国制定了《信息处理交换用的七位编码字符集》，后来成为国家标准GB1988。除个别字符（如货币符号）外，GB1988与ASCII是一致的，可视为ASCII的中国版本。

二、汉字编码简介

GB2312只收录了6763个汉字字符和682个非汉字图形字符（如间隔、标点、运算符、制表符、数字、汉语拼音、拉丁文字母、希腊文字母、俄文字母、日文假名等），已不能满足用户应用的需要。

2000年底又颁布了GB18030大字符集标准，该标准涵盖27484个汉字，简繁体字均处于同一平台，并在统一的编码框架下，为以后的扩充提供了充足的空间。GB18030采用单字节、双字节、四字节混合编码，总编码空间超过150万个。

二、汉字编码简介

目前内码推荐方案还不完全等同于汉字交换码国标，但二者之间存在简单的对应关系。

1.3 运算方法

- 01. 定点加减运算
- 02. 溢出的判断和移位
- 03. 定点乘法运算
- 04. 定点除法运算

一、定点加减运算

操作数用补码表示，符号位参与运算，结果用补码表示。

实际操作能否只取决于操作码?

结果是否需要修正?

如何将减法转换为加法?

1.补码的加减法

$$(X+Y)_{补}=X_{补}+Y_{补} \quad (1)$$

$$(X-Y)_{补}=X_{补}+(-Y)_{补} \quad (2)$$

数学依据：

$$(X+Y)_{补}=X+Y+2^n=X+Y+2^n+2^n=X_{补}+Y_{补}$$

$$(X-Y)_{补}=X-Y+2^n=X-Y+2^n+2^n=X_{补}+(-Y)_{补}$$

全过程以2^n为模，即除以2^n后取余数。

其中，$(-Y)_{补}=[Y_{补}]_{变补}$

Y的符号置反后再表示成补码

$Y_{补}$连同符号一起变反、末尾+1

例. 求$(X+Y)_{补}$

1) X=3, Y=2

$X_{补}$=0 0011

$Y_{补}$=0 0010 +

0 0101(+5补码)

2) X=–3, Y=–2

$X_{补}$=1 1101

$Y_{补}$=1 1110 +

1 1011(–5补码)

例. 求$(X-Y)_{补}$

1） X=4，　Y=－5

$X_{补}$=0 0100

$Y_{补}$=1 1011

$(-Y)_{补}$=0 0101＋

0 1001（+9补码）

2） X=－4，　Y= 5

$X_{补}$=1 1100

$Y_{补}$=0 0101

$(-Y)_{补}$=1 1011＋

1 0111（－9补码）

$Y_{补}$ 将$Y_{补}$变补 $(-Y)_{补}$：不管$Y_{补}$为正或负，将其符号连同尾数一起各位变反，末位加1。

一、定点加减运算

注意：某数的补码表示与某数变补的区别。

例. $1\ 0101_{原}$ —补码表示→ $1\ 1011$

$0\ 0101_{原}$ —补码表示→ $0\ 0101$

{ 符号位不变；
负数尾数改变，
正数尾数不变。

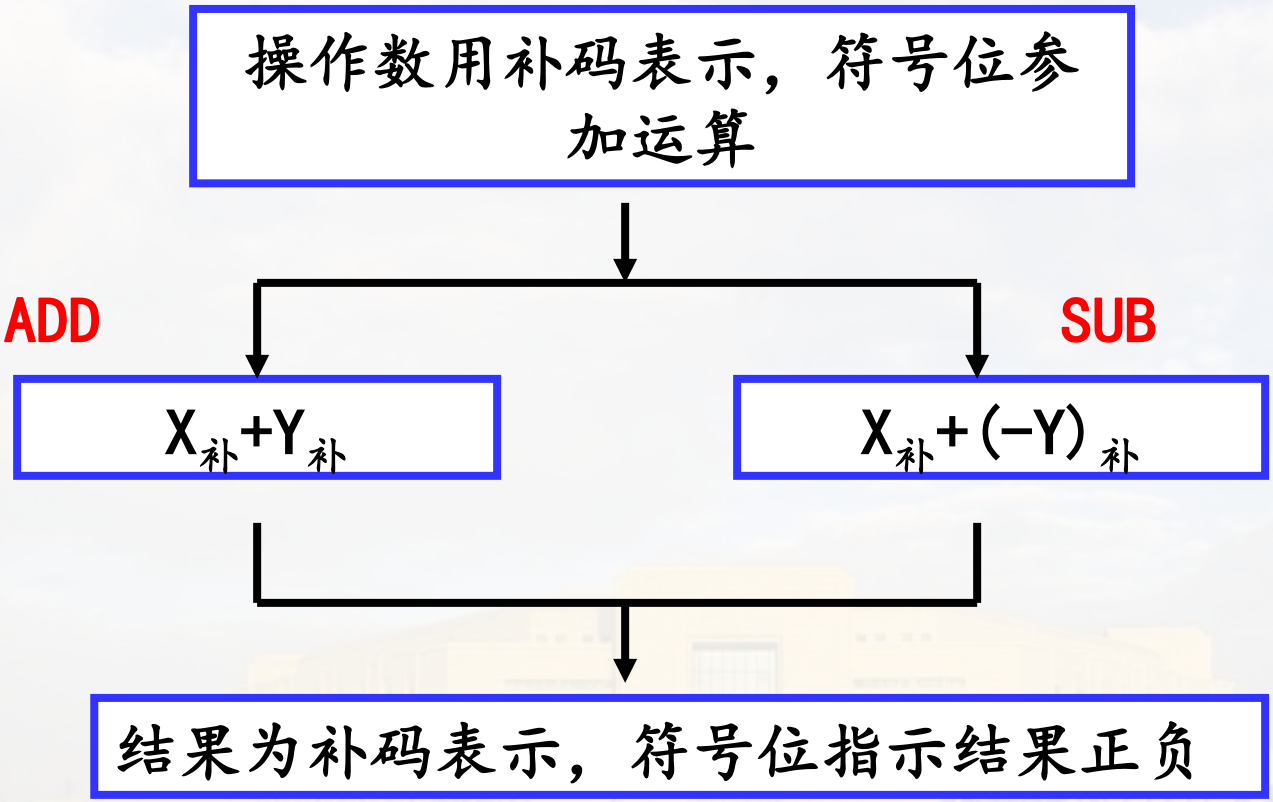
$1\ 0011_{补}$ —变补→ $0\ 1101$

$0\ 0011_{补}$ —变补→ $1\ 1101$

{ 符号位改变，
尾数改变。

补码的机器负数

2.补码的运算规则



3.逻辑实现(了解)

功能	所需控制命令				
加	+A	+B		Σ→A	CP_A
减	+A	+$\overline{B}$	+1	Σ→A	CP_A

加法器输入端：

+A：打开控制门，将A送∑。

+B：打开控制门，将B送∑。

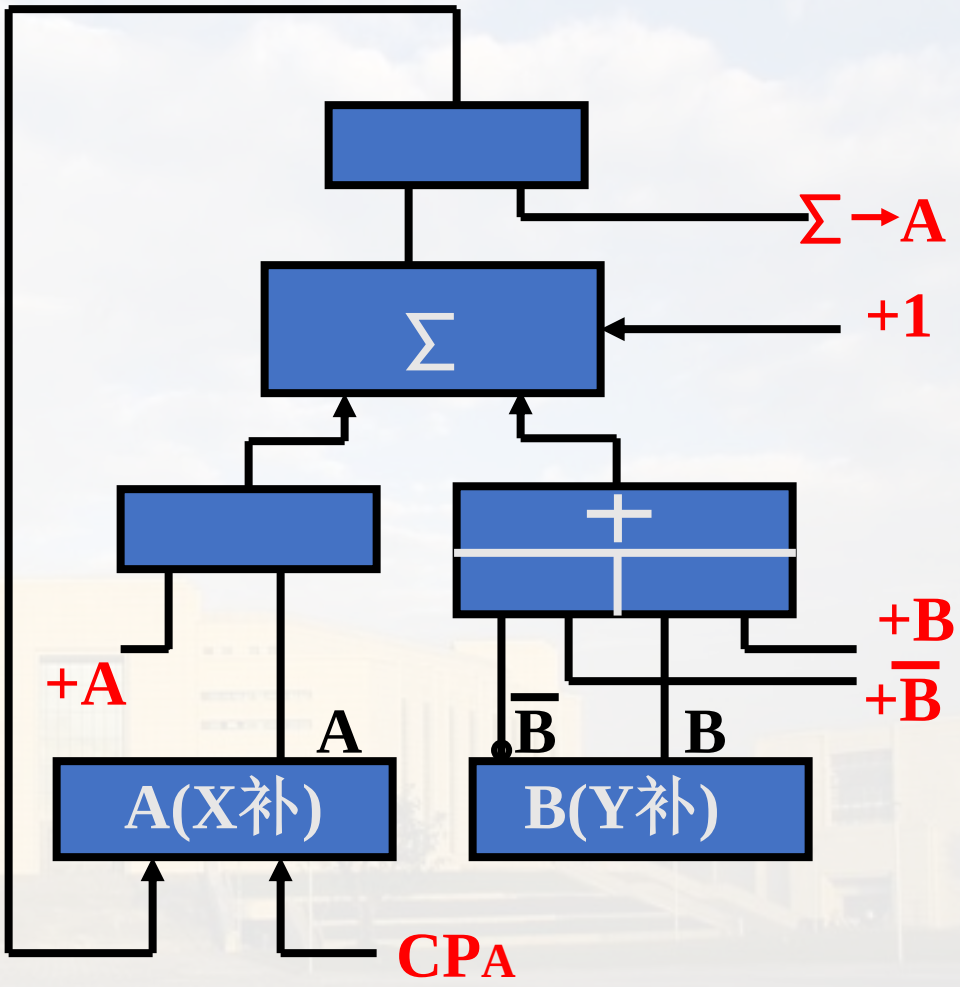
+$\overline{B}$：打开控制门，将$\overline{B}$送∑。

+1：控制末位加 1 。

加法器输出端：

∑→A：打开控制门，将结果送A输入端。

CPA：将结果打入A。



1.溢出的判断

溢出：运算结果超出机器数的表示范围

正溢：两正数相加绝对值超出允许的表示范围

负溢：两负数相加绝对值超出允许的表示范围

[假设] 补码表示的A、B两数做加减运算

数A有4位尾数，1位符号S_A

数B有4位尾数，1位符号S_B

结果符号S_f

符号位进位C_f

尾数最高位进位C

> 补码，符号位参加运算

二、溢出判断和移位

例（1）A=3　B=2

3+2：

```
  0 0011
  0 0010
+ ------
  0 0101
```

正确

（2）A=10　B=7

10+7：

```
  0 1010
  0 0111
+ ------
  1 0001
```

正溢

（3）A= -3　B= -2

-3+(-2)：

```
  1 1101
  1 1110
+ ------
  1 1011
```

正确

（4）A= -10　B= -7

-10+(-7)：

```
  1 0110
  1 1001
+ ------
  0 1111
```

负溢

二、溢出判断和移位

（5）A=6　B= -4

6+(-4)：

```
  0 0110
  1 1100   +
-----------
  0 0010
```

正确

（6）A= -6　B=4

-6+4：

```
  1 1010
  0 0100   +
-----------
  1 1110
```

正确

硬件判断逻辑一（SA、SB与Sf的关系）

A=10　B=7

10+7 ：

```
  0 1010
  0 0111   +
-----------
  1 0001
```

A= -10　B= -7

-10+(-7)：

```
  1 0110
  1 1001   +
-----------
  0 1111
```

溢出= $\overline{SA}\ \overline{SB}\ Sf + SA\ SB\ \overline{Sf}$

二、溢出判断和移位

硬件判断逻辑二 (Cf与C的关系)

溢出= Cf⊕C

（1） 3+2：

0 0011

C_f =0 0 0010 +

C =0 0 0101 正确

（2） 10+7：

0 1010

C_f =0 0 0111 +

C =1 1

1 0001 正溢

（3）-3+(-2)：

1 1101

C_f =1 1 1110 +

C =1 1 1

1 1011 正确

（4） -10+(-7)：

1 0110

C_f =1 1 1001 +

C =0 1

0 1111 负溢

（5） 6+(-4)：

0 0110

C_f =1 1 1100 +

C =1 1 1

0 0010 正确

（6） -6+4：

1 1010

C_f =0 0 0100 +

C =0

1 1110 正确

二、溢出判断和移位

硬件判断逻辑三（双符号位）

溢出= Sf1⊕Sf2

（1）3+2：

00 0011
00 0010 +
00 0101 正确

（2）10+7：

00 1010
00 0111 +
01 0001 正溢

第一符号位Sf1

第二符号位Sf2

（3）−3+(−2)：

11 1101
11 1110 +
11 0111 正确

（4）−10+(−7)：

11 0110
11 1001 +
10 1111 负溢

（5）6+(−4)：

00 0110
11 1100 +
00 0010 正确

（6）−6+4：

11 1010
00 0100 +
11 1110 正确

小结：硬件判断逻辑

（1） 硬件判断逻辑一（SA、SB与Sf的关系）

溢出= $\overline{SA}\,\overline{SB}\,Sf + SA\,SB\,\overline{Sf}$

（2） 硬件判断逻辑二（Cf与C的关系）

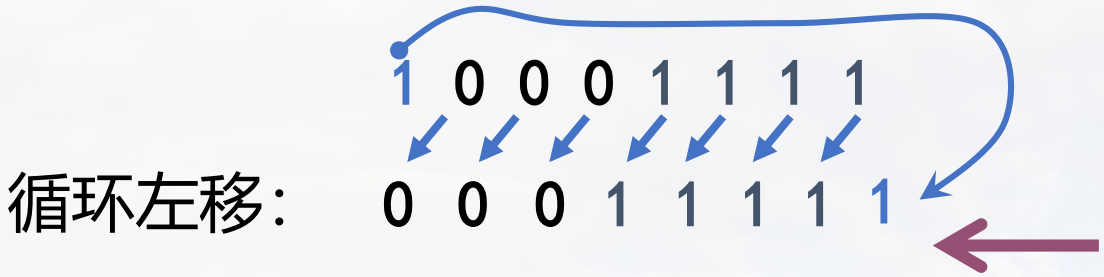
溢出= $Cf \oplus C$

（3）硬件判断逻辑三（双符号位）

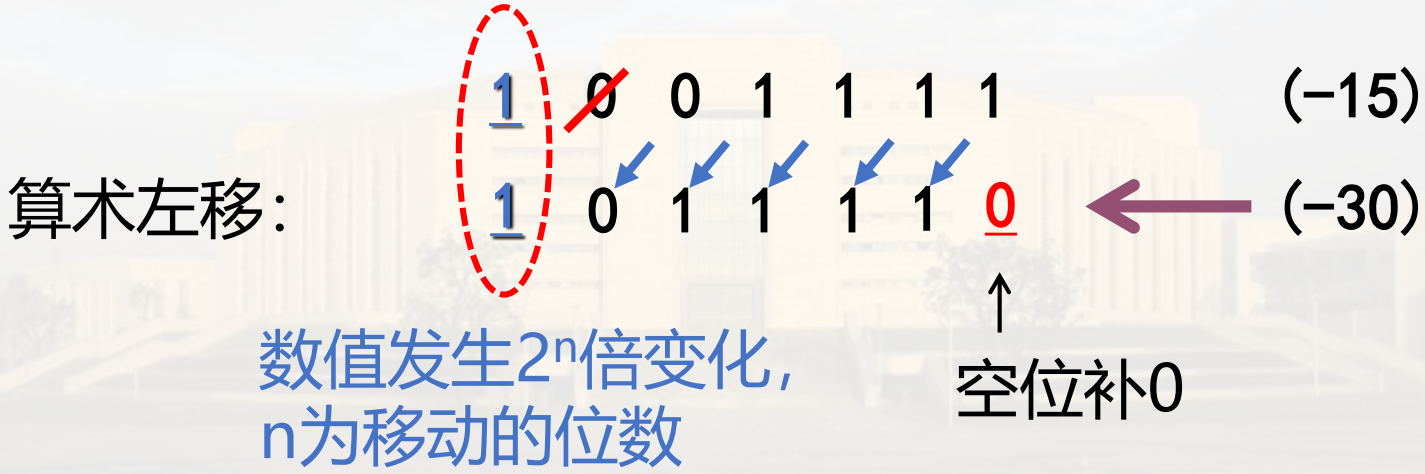
溢出= $Sf1 \oplus Sf2$

2.移位操作

(1)逻辑移位 ： 数码位置变化。



(2)算术移位 ： 符号位不变、数码位置变化



二、溢出判断和移位

①正数补码/原码移位规则

※移位规则

数符不变(单：符号位不变；双：第1符号位不变)

空位补0 (右移时第2符号位移至尾数最高位)

（1）单符号位：

	0 0111	7
左移	0 1110	14
右移	0 0111	7
右移	0 0011	3

（2）双符号位：

	00 0111	
左移	00 1110	
左移	01 1100	28
右移	00 1110	
右移	00 0111	

②负数补码移位

(1) 移位规则

数符不变 (单：符号位不变；双：第1符号位不变)

左移空位补0

右移空位补1 (第二符号位移至尾数最高位)。

(2) 单符号位：

	1 1011	−5
左移	1 0110	−10
右移	1 1011	
右移	1 1101	

(3) 双符号位:

	11 0110	−10
左移	10 1100	−20
右移	11 0110	
右移	11 1011	

二、溢出判断和移位

※易出错处：

00 1110

左 ← ~~00 1100~~

正确：01 1100

01 1100

右 → ~~00 0110~~

正确：00 1110

11 0110

左 ← ~~11 1100~~

正确：10 1100

10 1100

右 → ~~11 1110~~

正确：11 0110

3.舍入方法

① 0舍1入（原码、补码采用相同的方法）

[例] 保留4位尾数：

$0\ \ 00100_{原} \longrightarrow 0\ \ 0010_{原}$

$1\ \ 00101_{原} \longrightarrow 1\ \ 0011_{原}$

$1\ \ 11011_{补} \longrightarrow 1\ \ 1110_{补}$

② 末位恒置1（原码、补码）

[例] 保留4位尾数：

$0\ \ 00100_{原} \longrightarrow 0\ \ 0011_{原}$

$1\ \ 00101_{原} \longrightarrow 1\ \ 0011_{原}$

$1\ \ 11011_{补} \longrightarrow 1\ \ 1101_{补}$

三、定点乘法运算（了解）

乘法 ⟶ 部分积累加、移位。

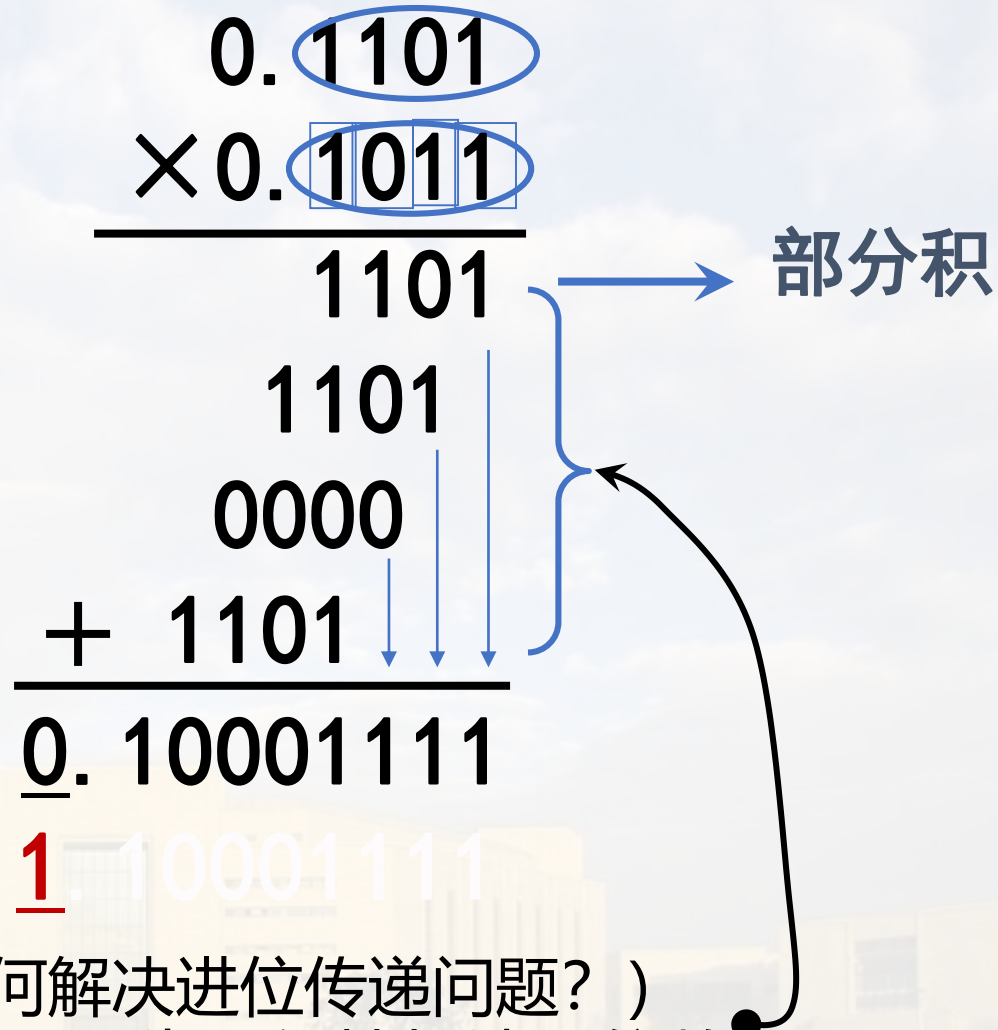
X原　　Y原

例. 0.1101×1.1011

乘积 $P = |X| \times |Y|$

积符 $S_P = S_X \oplus S_Y$

手工运算



改进：

将一次总加改为分步移位累加。

困难：

添加符号：1.0001111

1）加数增多情况（即如何解决进位传递问题？）

2）加数的位数增多的情况，为了保持加法器位数不变，能否改变移位方法？

1.原码一位乘法

每次将一位乘数所对应的部分积与原部分积的累加和相加，并右移。

例. 计算 0.1101×1.1011

设置寄存器：

A：存放部分积累加和、乘积高位

B：存放被乘数的绝对值

C：存放乘数的绝对值、乘积低位

设置初值：A = 00.0000　　B =|X|= 00.1101

C =|Y|= .1011

步数	条件	操作	A	C	Cn
			00.0000	.1011	
1）	Cn=1	+B	+ 00.1101		
			00.1101		
		→	00.0110	1.101	
2）	Cn=1	+B	+ 00.1101		
			01.0011		
		→	00.1001	11.10	
3）	Cn=0	+0	+ 00.0000		
			00.1001		
		→	00.0100	111.1	
4）	Cn=1	+B	+ 00.1101		
			01.0001		
		→	00.1000	1111	

$X_{原} \times Y_{原} = 1.10001111$

运算规则总结

① 寄存器分配与初始值：A,B,C三个寄存器

- A存放部分积累加和，初始值为0(双符号位00表示)；
- B存放被乘数X（绝对值），此时，符号位为双符号位00（在乘的过程中，B中的值一直保持不变）；
- C存放乘数Y（绝对值），将符号位去掉；C寄存器的初始值是乘数Y的尾数（有效位数），以后每乘一次，将已处理的低位乘数右移舍去，同时将A寄存器的末位移入C寄存器的高位。

② 符号位：结果符号位单独处理（区别于补码运算）

③符号位：A,B均设置双符号位

④基本操作：

- 在原码一位乘中，每步只处理一位乘数，即位于C寄存器末位的乘数，也称之为判断位Cn；
- 若Cn=1，则部分积为B，执行A+B操作，然后将累加和右移一位，用“→”表示。（Cn 位去掉）；
- 若Cn=0，则部分积为0，执行A+0操作，然后右移。或直接让A右移一位。（Cn 位去掉）
- 右移时，A的末位移入C的高位，A的第二符号位移入尾数最高位，第一符号位移入第二符号位，而第一符号位本身保持不变。

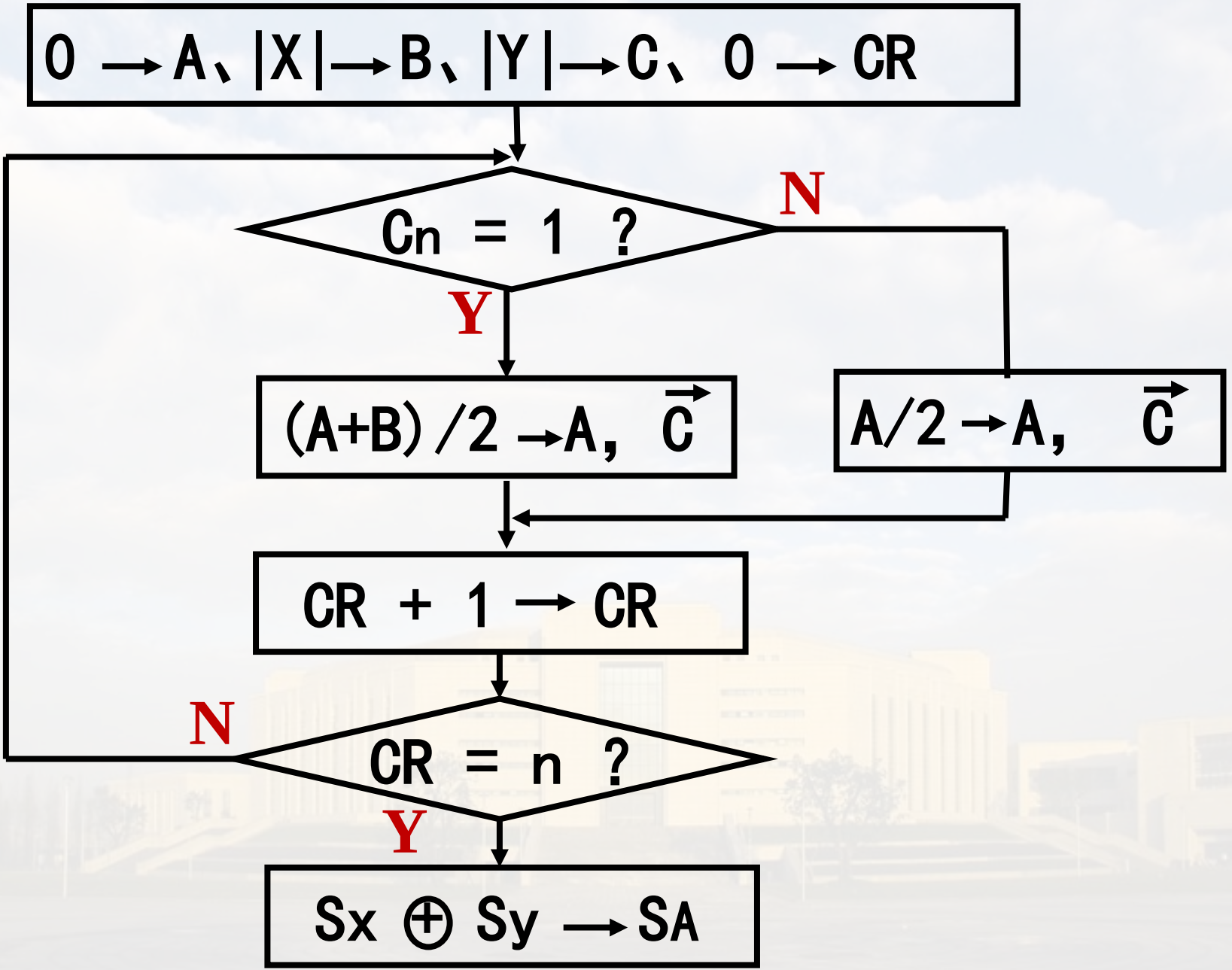
二、定点乘法运算

④ 操作步骤：

N次累加与n次移位（最后一次累加后要移位）

⑤ 加符号位

（2）算法流程



2.补码一位乘法

定义：操作数与结果均以补码表示，连同符号位一起，按相应算法运算。

(1)校正法- 算法分析

$$X_{补} = X_0.X_1X_2\cdots\cdots X_n$$

不管被乘数$X_{补}$的符号如何，只要乘数$Y_{补}$为正，则可像原码乘法一样进行运算，其结果不需校正。（证明略）

(a) Y为正：$Y_{补} = 0.Y_1Y_2\cdots\cdots Y_n$

$$(XY)_{补} = X_{补}(0.Y_1Y_2\cdots\cdots Y_n)$$

(b) Y为负：$Y_{补} = 1.Y_1Y_2\cdots\cdots Y_n$

$$(XY)_{补} = X_{补}(0.Y_1Y_2\cdots\cdots Y_n)+(-X)_{补}$$

2.补码一位乘法

(2)比较法 - 算法分析

将校正法的两种情况统一起来，演变为比较法。

（c）Y符号任意：

$$(XY)_{补} = X_{补}(0.Y_1Y_2.....Y_n)+(-X)_{补}Y_0$$

符号位

2.补码一位乘法

（d）展开为部分积的累加和形式：

$$(XY)_{补} = X_{补}(0.Y_1Y_2\ldots\ldots Y_n)+(-X)_{补}Y_0$$

$$= X_{补}(0.Y_1Y_2\ldots\ldots Y_n)-X_{补}Y_0$$

$$= X_{补}(-Y_0+2^{-1}Y_1+2^{-2}Y_2+\ldots\ldots+2^{-n}Y_n)$$

$$= X_{补}\left[-Y_0+(Y_1-2^{-1}Y_1)+(2^{-1}Y_2-2^{-2}Y_2)+\ldots\ldots+(2^{-(n-1)}Y_n-2^{-n}Y_n)\right]$$

$$= X_{补}\left[(Y_1-Y_0)+2^{-1}(Y_2-Y_1)+2^{-2}(Y_3-Y_2)+\ldots\ldots+2^{-n}(Y_{n+1}-Y_n)\right]$$

比较法：用相邻两位乘数比较的结果决定$+X_{补}$、$-X_{补}$或$+0$。

2.补码一位乘法

$$
\begin{aligned}
&[A_0]_{补} = 0\\
&[A_1]_{补} = 2^{-1}\{[A_0]_{补} + (Y_{n+1} - Y_n)[X]_{补}\}\\
&[A_2]_{补} = 2^{-1}\{[A_1]_{补} + (Y_n - Y_{n-1})[X]_{补}\}\\
&\quad\vdots\\
&[A_n]_{补} = 2^{-1}\{[A_{n-1}]_{补} + (Y_2 - Y_1)[X]_{补}\}\\
&[XY]_{补} = [A_n]_{补} + (Y_1 - Y_0)[X]_{补}
\end{aligned}
$$

(3)运算方法及关系式：比较法

$$[XY]_{补} = [A_n]_{补} + (Y_{n+1} - Y_n)[X]_{补}$$

注意：Y_{n+i}为低位，Y_{n+i-1}为高位

$$= X_{补}\left[(Y_1 - Y_0) + 2^{-1}(Y_2 - Y_1) + 2^{-2}(Y_3 - Y_2) + \cdots\cdots + 2^{-n}(Y_{n+1} - Y_n)\right]$$

二、定点乘法运算

$$[XY]_{补} = [A_n]_{补} + (Y_{n+1} - Y_n)[X]_{补}$$

比较法算法

Yn（高位）	Yn+1（低位）		操作（A补为部分积累加和）
0	0	（ 0 ）	A补/2
0	1	（ 1 ）	（A补+X补）/2
1	0	（-1 ）	（A补-X补）/2
1	1	（ 0 ）	A补/2

运算实例

X= -0.1101, Y= -0.1011, 求(XY)补。

初值：A= 00.0000, B=X补=11.0011,

-B=(-X)补=00.1101, C =Y补=1.0101

步数	条件 C_nC_{n+1}	操作	A	C	C_n C_{n+1}
			00.0000	1.0101	1 0
1）	1 0	−B	+ 00.1101		
			00.1101		
		→	00.0110	11.010	0 1
2）	0 1	+B	+ 11.0011		
			11.1001		
		→	11.1100	111.01	0
3）	1 0	−B	+ 00.1101		
			00.1001		
		→	00.0100	1111.0	1
4）	0 1	+B	+ 11.0011		
			11.0111		
		→	11.1011	11111.	0
5）	1 0	−B	+ 00.1101		
		修正	00.1000	1111	$(XY)_{补}$

(4)运算规则

① 寄存器分配、初始值及符号位：

A存放部分积累加和，初始值为0(双符号位00表示)；

B存放被乘数$X_{补}$，（双符号位00、或11表示）；

C存放乘数$Y_{补}$，单符号位（符号位参与运算），Y的末位添0，称为附加位Y_{n+1}。

② 基本操作：

用C寄存器最末两位（含增加的C_{n+1}）作判断位，即(Y_n, Y_{n+1})为判断位。

若Y_n Y_{n+1}为00或11，执行A+0，右移，实际上可直接让A右移一位；

若Y_n Y_{n+1}为01，执行$A+X_{补}$，右移；

若Y_n Y_{n+1}为10，执行$A+[-X]_{补}$，右移。

在右移时，A寄存器中的第二符号位移入尾数的最高位（有效位的最高位），第一符号位移入第二符号位，第一符号位本身不变，而A寄存器末位移入C寄存器。

③ 操作步数：为有效位位数的n+1

三、定点除法运算

例. 0. 10110÷0. 11111

```
              0.10110
0. 11111 ) 0. 101100
              -11111
               110100
               -11111
                101010
                -11111
         0. 00000 10110
```

商： 0. 10110

余数：0. 10110×2 $^{-5}$

实现除法的关键：比较余数、除数绝对值大小，以决定上商。

三、定点除法运算

手工变为机器实现时，需要解决三个问题：

① 如何判断够减

方法
- 先判后减（硬件方式）
- 先减后判（软件方式）
  - 恢复余数法
  - 不恢复余数法（加减交替除法）

② 如何处理符号位

③ 如何提高除法运算速度

除法器分类：

方法
- 常规除法器
- 迭代除法器
- 阵列除法器

1、原码不恢复余数除法

（1）定义：取两个操作数的绝对值相除，符号位单独处理。

（2）运算关系式：

根据余数r_i符号判断是否够减：(设Y表示除数，r表示余数。)

r_i为正表示够减，上商$Q_i = 1$；

r_i为负表示不够减，上商$Q_i = 0$；

若第i步够减，$Q_i=1$，则第i+1步应做$2r_i-Y$；

若第i步不够减，$Q_i=0$，则第i+1步应做$2r_i+Y$。

即：出现不够减时，也可以不恢复余数而直接做下一步，改作加除数，结果与恢复余数法等效。

因此，通式为：$r_{i+1} = 2r_i + (1 - 2Q_i)Y$

三、定点除法运算

求n位商，作n步操作；若第n步余数为负，则第n+1步（最后一步操作）恢复余数，不移位。

(3)实例：　X=0.10110，Y=-0.11111，

求X/Y，给出商Q和余数R（n=5）

初值：A=|X|= 00.10110，　B=|Y|=00.11111

-B= 11.00001，　C=|Q|= 0.00000

步数	条件	操作	A		C	C_n
	r		00.10110	r0	0.00000	
1）		←	01.01100	2r0		
		−B	+11.00001			
	为正		00.01101	r1	0.00001	Q1
2）		←	00.11010	2r1		
		−B	+11.00001			
	为负		11.11011	r2	0.00010	Q2
3）		←	11.10110	2r2		
		+B	+00.11111			
	为正		00.10101	r3	0.00101	Q3
4）		←	01.01010	2r3		
		−B	+11.00001			
	为正		00.01011	r4	0.01011	Q4

三、定点除法运算

步数	条件	操作	A		C	
						Cn
	为正		00.01011	r4	0.01011	Q4
5）		←	00.10110	2r4		
		−B	+11.00001			
	为负		11.10111	r5’	0.10110	Q5
6）		+B	+00.11111			
		恢复余数	00.10110	r5		

$Q = -0.10110$

$R = 0.10110 \times 2^{-5}$

$$X/Y = -0.10110 + \frac{0.10110 \times 2^{-5}}{-0.11111}$$

（3）运算规则

① 寄存器分配与符号位：

A,B,C三个寄存器；

A初始值存放被除数（绝对值），以后存放各次余数，

A取双符号位，从第一符号位判断是否够减，从而决定商值；

B寄存器存放除数的绝对值，取双符号位；

C存放商，取单符号位；商由末位置入，在每次置入新商时，原商同时左移一位。

② 基本操作与上商：

a.第一步操作必为$2r_0$ -Y

b.以后各部根据如下条件进行：

r_i为正表示够减，即Q_i=1，则第i+1步应为$2r_i$-Y，

r_i为负表示不够减，即Q_i=0，则第i+1步应为$2r_i$+Y；

c.最后一步：若第n步（最后一步）余数为负，则需增加一步恢复余数，这增加的一步不移位，操作为r_n+Y。

三、定点除法运算

③ 操作步数：

要求得n位商（不含符号位），则需做n步（次）

“左移---加减”循环。

④ 符号：同号相除为正，异号反之。

（4）微命令设置

r_i 符号位=0，即Q_i = 1，则$2r_i$-Y：+2A、+$\bar{B}$ 、+1，

Σ→A，$\overleftarrow{C}$，Q_i→C_n，CP_A、CP_C

r_i符号位=1，即Q_i = 0，则$2r_i$+Y：+2A、+B、

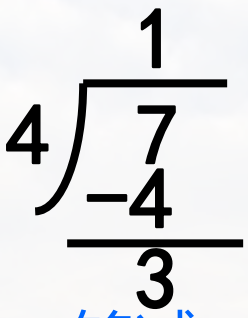
Σ→A，$\overleftarrow{C}$ ，Q_i→C_n，CP_A、CP_C

最后一步中，若余数为负，则需要恢复余数操作，即r_i+ Y。

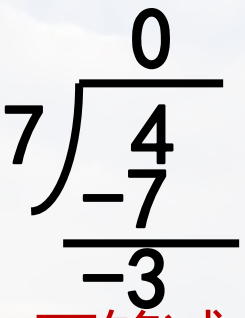
2、补码不恢复余数除法

(1)判够减

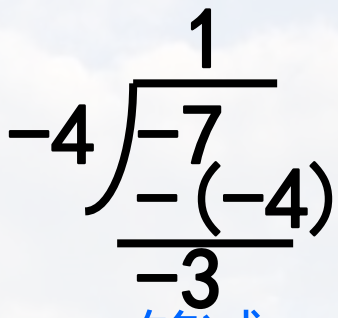
同号相除



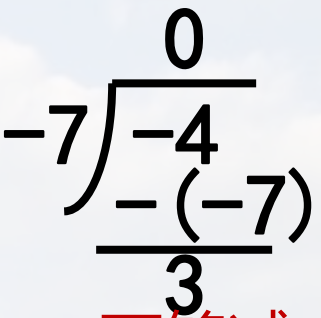
够减



不够减



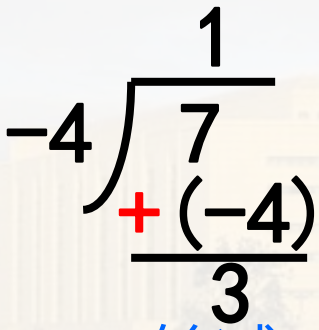
够减



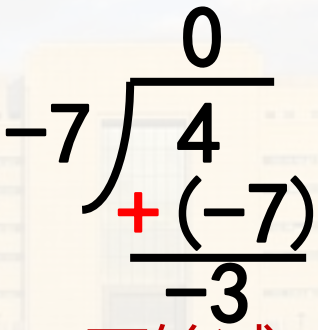
不够减

够减：r与X、Y同号；　　不够减：r与X、Y异号。

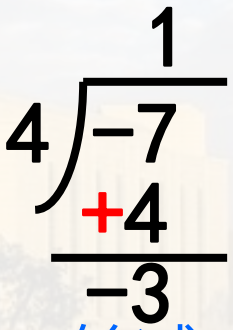
异号相除



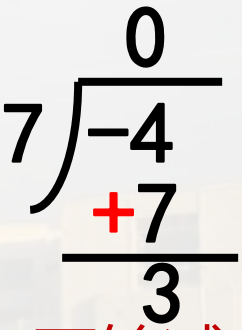
够减



不够减



够减



不够减

够减：r与X同号,Y异号；　　不够减：r与X异号,Y同号

三、定点除法运算

判断规则：

$\frac{r_{补}}{Y_{补}}$ 同号操作：$r_{补}-Y_{补}$
- 够减：$r_{补}$与$Y_{补}$同号
- 不够减：$r_{补}$与$Y_{补}$异号

$\frac{r_{补}}{Y_{补}}$ 异号操作：$r_{补}+Y_{补}$
- 够减：$r_{补}$与$Y_{补}$异号
- 不够减：$r_{补}$与$Y_{补}$同号

(2) 求商值

$\frac{r_{补}}{Y_{补}}$
- 同号：商为正
  - 够减商1 （r、Y同号）
  - 不够减商0 （r、Y异号）
- 异号：商为负
  - 够减商0 （r、Y异号）
  - 不够减商1 （r、Y同号）

上商规则： $Q_i = \overline{S_{ri} \oplus S_Y}$

余数与除数同号商1，异号商0。

三、定点除法运算

(3)算法

$$(r_{i+1})_{补} = 2r_{i补} + (1-2Q_{i补})Y_{补}$$

第i+1步操作作

$\frac{r_{补}}{Y_{补}}$

- 同号：商为正
  - 够减商1 (r、Y同号) $2r_{i补} - Y_{补}$
  - 不够减商0 (r、Y异号) $2r_{i补} + Y_{补}$
- 异号：商为负
  - 够减商0 (r、Y异号) $2r_{i补} + Y_{补}$ ？
  - 不够减商1 (r、Y同号) $2r_{i补} - Y_{补}$

异号相除的时候，余数和除数作加法，其实就是减去其绝对值。

$r_{i补}$与$Y_{补}$同号，则$Q_{i补}$为1，第i+1步作$2r_{i补}-Y_{补}$；

$r_{i补}$与$Y_{补}$异号，则$Q_{i补}$为0，第i+1步作$2r_{i补}+Y_{补}$。

(3)算法

$$(r_{i+1})_{补} = 2r_{i补} + (1-2Q_{i补})Y_{补}$$

$r_{i补}$与$Y_{补}$同号，则$Q_{i补}$为1，第i+1步作$2r_{i补}-Y_{补}$；

$r_{i补}$与$Y_{补}$异号，则$Q_{i补}$为0，第i+1步作$2r_{i补}+Y_{补}$。

(4)求商符

补码符号位参与运算，因而商符是通过运算得到，

令$X_{补} = r_{0补}$（初始余数），则：

$r_{0补}$与$Y_{补}$ 同号：$Q_{0补}=1$；异号：$Q_{0补}=0$

商符

与实际商符相反

(5)商的校正

$$\frac{X补}{Y补} = (1 + 2^{-n} + \sum_{i=0}^{n-1} 2^{-i} Q_{i补}) + \frac{2^{-n} r_{n补}}{Y补}$$

余数

商

(a) $\sum_{i=0}^{n-1} 2^{-i} Q_{i补} = Q_0.Q_1Q_2\ldots\ldots Q_{n-1}$ n-1位商 (假商)

(b) 2^{-n} 第n位商(末位商)恒置1

(c) 1 商符变反

真商 = 假商 + 1.000...01 (n位)

(d) 余数求至r_n

(6) 实例

X=0.10110，Y=−0.11111，求X/Y，给出商Q和余数R。

初值：A $=X_{补}$=00.10110　　B $=Y_{补}$=11.00001

−B =00.11111　　C $=Q_{补}$=0.00000

步数	条件	操作	A		C	C_{n-1}
	r、Y异号	求商符	00.10110	r0	0.0000	Q0
1）		←	01.01100	2r0		
		+B	+11.00001			
	异号		00.01101	r1	0.0000	Q1
2）		←	00.11010	2r1		
		+B	+11.00001			
	同号		11.11011	r2	0.0001	Q2

步数	条件	操作	A		C	C_{n-1}
	r、Y		11.11011	r2	0.0001	Q2
3）		←	11.10110	2r2		
		−B	+00.11111			
	异号		00.10101	r3	0.0010	Q3
4）		←	01.01010	2r3		
		+B	+11.00001			
	异号		00.01011	r4	0.0100	Q4
5）		←	00.10110	2r4		
		+B	+11.00001			
			11.10111	r5		

假商=0.0100

真商=0.0100+1.00001=1.01001

Q= −0.10111　　R= -0.01001×2^{-5}

$$X/Y=-0.10111+\frac{-0.01001\times2^{-5}}{-0.11111}$$

(6) 运算规则总结

① 寄存器分配与符号位:

A,B,C三个寄存器;

A初始值存放被除数（补码表示），以后存放各次余数，A取双符号位;

B寄存器存放除数（补码表示），取双符号位;

C存放商，取单符号位，初始值为0。

三、定点除法运算

② 假商符：在第一步操作之前，先根据r_0（即X）、Y符号比较确定假商符（与真商符相反），即：

r_0、Y同号为1

r_0、Y异号为0

③ 基本操作

a.第一步操作，假商符为1(r_0、Y同号)，$2X_{补}-Y_{补}$

假商符为0(r_0、Y异号)，$2X_{补}+Y_{补}$

b.其余操作根据如下规则进行：

三、定点除法运算

若$X_{i补}$、$Y_{补}$同号：

$r_{i补}$ $Y_{补}$同号（够减），上商1，下一步，$2r_{i补}-Y_{补}$

$r_{i补}$ $Y_{补}$异号（不够减），上商0，下一步，$2r_{i补}+Y_{补}$

若$X_{i补}$，$Y_{补}$异号：

$r_{i补}$ $Y_{补}$同号（不够减），上商1，下一步，$2r_{i补}-Y_{补}$

$r_{i补}$ $Y_{补}$异号（够减），上商0，下一步，$2r_{i补}+Y_{补}$

c.最后一步要对假商校正。

(4) 微命令设置

$r_{i补}$、$Y_{补}$同号，即$2r_{i补}-Y_{补}$：

$+2A$、$+\bar{B}$、$+1$，$\Sigma\rightarrow A$，C，$Q_i\rightarrow C_n$，CP_A、CP_C

$r_{i补}$、$Y_{补}$异号，即$2r_{i补}+Y_{补}$：

$+2A$、$+B$、$\Sigma\rightarrow A$，C，$Q_i\rightarrow C_n$，CP_A、CP_C

本章小结

重点：

1、数的转换问题（2进制、8进制、16进制）

2、定点数的表示(表示范围和精度)

3、掌握IEEE754（short 32位）

4、定点数加减运算

第一章作 业

P75页　　习题2

2大题、6大题、9大题、12大题、13大题、14大题、17大题、18大题

作业

题号	分数
2大题	10
6大题	15
9大题	10
12大题	24
13大题	24
14大题	12
17大题	20
18大题	20
总分	135

谢谢观看

计算机基础

2023/9/13

信息与软件工程学院
School of Information and Software Engineering