第 2 章 数据的表示和运算

中华人民共和国成立70周年　1949——2019 大连理工大学 建校70周年

主要内容：

（一） 数制与编码

1. 进位计数制及其相互转换
2. 真值和机器数
3. BCD 码
4. 字符与字符串
5. 校验码

（二） 定点数的表示和运算

1. 定点数的表示 ：无符号数的表示；有符号数的表示。
2. 定点数的运算 ：定点数的位移运算；原码定点数的加/减运算；补码定点数的加/减运算；定点数的乘/除运算；溢出概念和判别方法。

（三） 浮点数的表示和运算

1. 浮点数的表示 ：浮点数的表示范围；IEEE754 标准
2. 浮点数的加/减运算

（四） 算术逻辑单元 ALU

1. 串行加法器和并行加法器
2. 算术逻辑单元 ALU 的功能和机构

2.2 定点数的表示和运算

2.2.1 定点数的表示

（1）无符号数的表示

- 机器字长的全部位数均用来表示数值的大小，相当于数的绝对值。对于字长为 n 位的无符号数的表示范围为：$0 — 2^{n}-$

（2）带符号数的表示

- 带符号数是指在计算机中将数的符号数码化。在计算机中，一般规定二进制的最高位为符号位，最高位为“0”表示该数为正，为“1”表示该数为负。这种在机器中使用符号位也被数码化的数称为机器数。根据符号位和数值位的编码方法不同，机器数分为原码、补码和反码等。

(纯小数) 原码，反码，补码的定义

中华人民共和国成立70周年　1949——2019 大连理工大学 建校70周年

定点小数表示：$N_s\ N_1\ N_2 \cdots\ N_n$

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

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

$$[X]_{补} = \begin{cases} X & 0 \le X < 1 \\ 2 + X = 2 - |X| & -1 \le X \le 0 \end{cases} \quad \text{Mod}\ 2$$

海纳百川 自强不息 厚德笃学 知行合一

(纯小数) 原码的定义与说明

中华人民共和国成立70周年　1949——2019 大连理工大学 建校70周年

定点小数表示：$N_s\,N_1\,N_2\cdots\ N_n$

定义：$$[X]_{原} = \begin{cases} X & 0 \le X < 1 \\ 1 - X = 1 + |X| & -1 < X \le 0 \end{cases}$$

实例：　X1 = 0.1011　-0.1011　0.0000

$[X]_{原}$ = 0 1011　1 1011　0 0000

1 0000

说明：原码是符号位加数的绝对值，符号 0 正 1 负　原码零有两个编码，+0 和 -0 的编码不同　原码难以用于加减运算，但乘除方便

(纯小数) 反码的定义与说明

定点小数表示：$N_s\, N_1\, N_2 \cdots\ N_n$

定义：$[X]_反 = \begin{cases} X & 0 \le X < 1 \\ (2 - 2^{-n}) - |X| & -1 < X \le 0 \end{cases} \quad \text{Mod}\,(2 - 2^{-n})$

实例：　X1 = 0.1011　　-0.1011　　0.0000

$[X]_反$ =　0 1011　　1 0100　　0 0000

　　　　　　　　　　　　　　　1 1111

结论：反码负数为符号位跟每位的反, 符号 0 正 1 负

反码零有二个编码，分 +0 和 -0 反码难以用于算术运算，加减有循环进位问题

(纯小数) 补码的定义与说明

定点小数表示：$N_s\ N_1\ N_2 \cdots\ N_n$

定义：$$[X]_补 = \begin{cases} X & 0 \le X < 1 \\ 2 + X = 2 - |X| & -1 \le X \le 0 \end{cases} \quad \text{MOD } 2$$

实例：　X1 = 0.1011　　-0.1011　　0.0000

[X]补 = 0 1011　　1 0101　　0 0000

说明：补码最高一位是符号位，符号 0 正 1 负 补码表示为：2×符号位 + 数的真值 补码零只有一个编码，故能表示 -1（10000） 补码能很好地用于加减（乘除）运算

海纳百川 自强不息 厚德笃学 知行合一

补码的一些补充说明

➢ 得到一个数补码表示的简便办法

当 $X \geq 0$ 时，$[X]_{补}$的符号位取 0，数值位取 X 的各数值位上的值， 此时有$[X]_{补} = X$

当 $X<0$ 时，$[X]_{补}$的符号位取 1，将 X 的**各数值位取反,再在最低位加 1**，以得到 $[X]_{补}$的各数值位上的值

(见负数 $[X]_{补}$ 与 $[X]_{反}$ 的关系)

➢ $[X]_{原}$与$[X]_{补}$的相互转换简便方法

从$[X]_{原}$求$[X]_{补}$时，对正数或零，有$[X]_{补}=[X]_{原}$，对负数则符号位不变，**各数值位变反后再在最低位执行加 1** 操作。由$[X]_{补}$求 $[X]_{原}$时，对负数仍是符号位不变，**各数值位变反后再在最低位执行加 1** 操作。

已知 $[y]_{补}$ 如何简单求 $[-y]_{补}$

<I>

$$\boxed{[y]_{补} = 0.\ y_1 y_2 \cdots y_n}$$

$$y = 0.\ y_1 y_2 \cdots y_n$$

$$-y = -0.\ y_1 y_2 \cdots y_n$$

$$\boxed{[-y]_{补} = 1.\ \overline{y_1}\ \overline{y_2} \cdots \overline{y_n} + 2^{-n}}$$

<II>

$$\boxed{[y]_{补} = 1.\ y_1 y_2 \cdots y_n}$$

$$[y]_{原} = 1.\ y_1 y_2 \cdots y_n + 2^{-n}$$

$$y = -(0.\ y_1 y_2 \cdots y_n + 2^{-n})$$

$$-y = 0.\ y_1 y_2 \cdots y_n + 2^{-n}$$

$$\boxed{[-y]_{补} = 0.\ \overline{y_1}\ \overline{y_2} \cdots \overline{y_n} + 2^{-n}}$$

整数的编码表示

➢整数的 原码 反码 补码 表示

与小数的三种表示基本相同
差别仅表现在小数点的位置 可以认为整数
的小数点在最低数值位的右侧

➢因此整数的模与整数位数有关

讲课中不大用整数讲 原 反 补码定义
例如：整数 6 位编码(1 位符号位，5 位数值位)

$X = +01110 \longrightarrow [X]_{原} = 0\ 01110 \qquad [X]_{补} = 0\ 01110$

$X = -01110 \longrightarrow [X]_{原} = 1\ 01110 \qquad [X]_{补} = 1\ 10010$

整数的编码表示

中华人民共和国成立70周年　1949——2019 大连理工大学 建校70周年

x 为真值　　n 为整数的位数

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

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

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

海纳百川 自强不息 厚德笃学 知行合一

•设机器数字长为 8 位 (含一位符号位)，表示整数时，每个编码分别代表无符号数、原码、补码和反码的真值各为多少？

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

原、反、补码表示小结

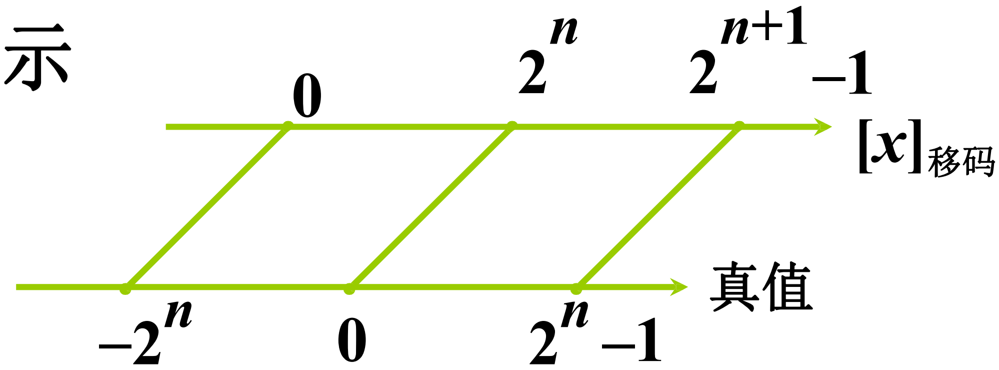
- 正数的 原码、反码、补码表示均相同，符号位为 0，数值位同数的真值。
- 零的原码和反码均有 2 个编码，补码只 1 个码
- 负数的 原码、反码、补码表示均不同，
  符号位为 1，数值位：原码为数的绝对值
  反码为每一位均取反码 补码
  为反码再在最低位+1
- 由 $[X]_{补}$ 求 $[-X]_{补}$：每一位取反后再在最低位+1

整数的移码表示 (用于浮点数阶码)

移码定义　　x 为真值，n 为 整数的位数

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

移码在数轴上的表示



例如：

$x = 10100$　　$[x]_{移} = 2^5 + 10100=100000+10100= 110100$

$x = -10100$　　$[x]_{移} = 2^5 - 10100=100000-10100= 001100$

真值、补码和移码的对照表

真值 x (n=5)	$[x]_{补}$	$[x]_{移}$	$[x]_{移}$对应的十进制整数
- 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

2.2.2 定点数的运算

（1）定点数的移位运算

- 移位是一种常用的操作，例如，在乘法中需要左移，在除法中需要右移，在代码处理中也经常需要移位操作。
- 移位可分为算术移位和逻辑移位，有左移和右移之分。可以对寄存器或存储单元中的数据进行移位。一次可以只移一位，也可以按指令中规定的次数移若干位。

① 算术移位：移位的对象是数值型数据，在移位后会发生数值大小的变化。

- 对于二进制数，左移，绝对值扩大；右移，绝对值缩小。
- 算术移位规则：符号位不变

② 逻辑移位：包括逻辑左移、逻辑右移、循环左移和循环右移等。逻辑移位使代码序列进行循环移位或非循环移位，参与移位的对象被视为纯逻辑意义上的代码组合，逻辑移位只是使数码位置发生变化，没有正、负性质，也没有数值大小问题。

③ 算术移位和逻辑移位的区别：

- 算术移位：有符号数移位
- 逻辑移位：无符号数移位

【例】 选择题：数据每左移 1 位相当于原数乘以 2 ，为防止左移操作造成 溢出，补码左移的前提条件是其原最高有效位（ ）。（当运算结果超出机器数所能表示的范围时，称为溢出。）

A．为 0　　　　　　B．为 1

C．与原符号位相同　　　　　　D．与原符号位相异

【分析】当最高有效位和符号位不一致时，如果左移 1 位，将会发生溢出。

【归纳总结】算术移位是带符号数的移位，移位前后符号位不应该发生变化。如果最高有效位和符号位不一致，则左移 1 位，符号位将发生变化。答案： C

【例】已知[X/2]补=C6H，计算机的字长为 8 位二进制编码，[X]补=?

解： [X]补= [X/2]补×2 ，C6H=11000110B，左移 1 位变成 10001100B=8CH [X]补=8CH

（2）原码定点数的加/减运算

- 对原码表示的两个操作数进行加减运算时，计算机的实际操作是加还是减，不仅取决指令中的操作码，还取决于两个操作数的符号。而且运算结果的符号判断也较复杂。

- 例如，加法指令指示做（+X）+（-Y），由于一操作数为负，实际操作是做减法（+X）-（+Y），结果符号与绝对值大的符号相同。同理，在减法指令中指示做

  （+X）-（-Y），实际操作做加法（+X）+（+Y），结果与被减数符号相同。

- 由于原码加减法比较繁琐，相应地需要由复杂的硬件逻辑才能实现，因此在计算机中很少被采用。

（3）补码定点数的加/减运算

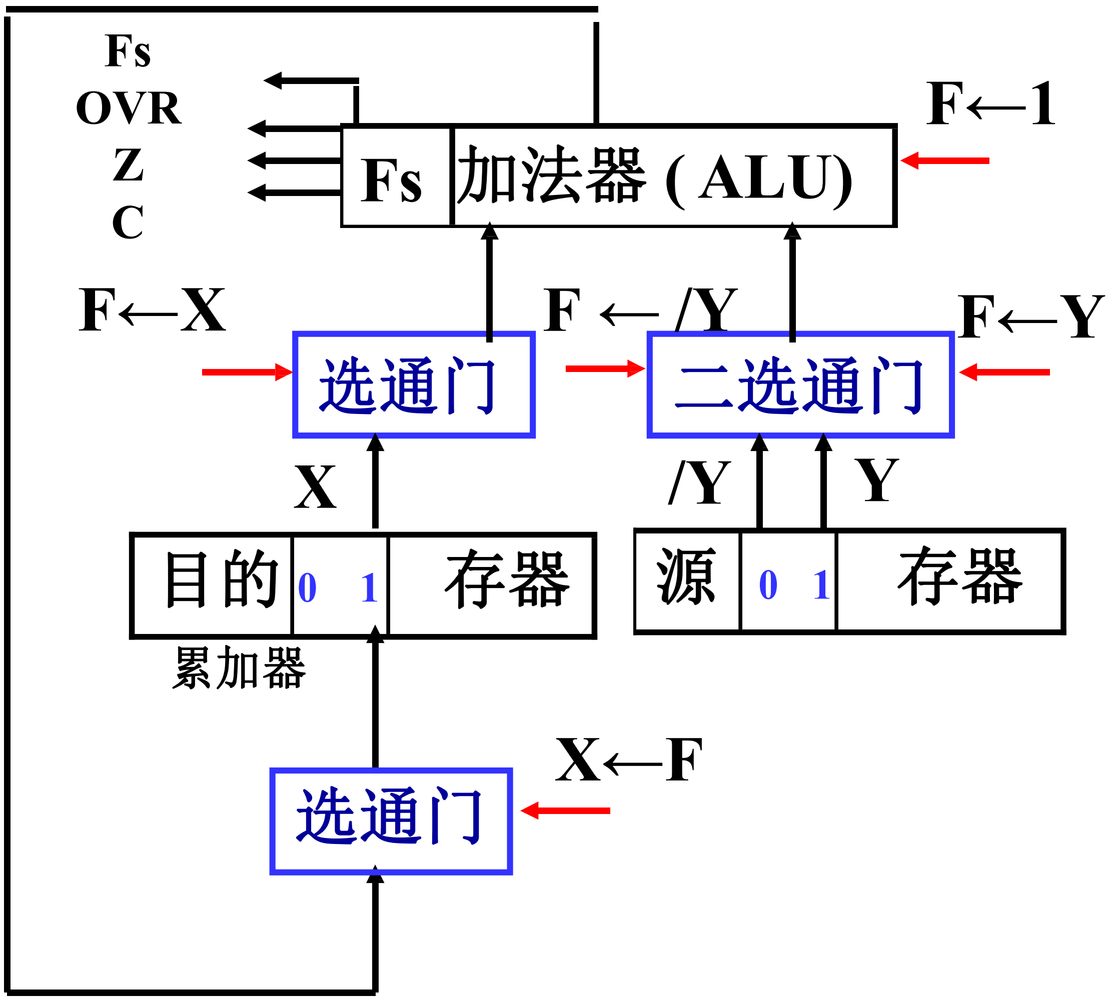
① 加法

- 整数 $[X+Y]_{补}=[X]_{补}+[Y]_{补}\ (\mathrm{mod}\ 2^{n+1})$
- 小数 $[X+Y]_{补}=[X]_{补}+[Y]_{补}\ (\mathrm{mod}\ 2)$

② 减法

- 整数 $[X-Y]_{补}=[X+(-Y)]_{补}=[X]_{补}+[-Y]_{补}\ (\mathrm{mod}\ 2^{n+1})$
- 小数 $[X-Y]_{补}=[X+(-Y)]_{补}=[X]_{补}+[-Y]_{补}\ (\mathrm{mod}\ 2)$
- 无需符号判定，数值位连同符号位一起相加，符号位产生的进位自然丢掉。
- 关键是由$[Y]_{补}$求$[-Y]_{补}$， $[-Y]_{补}=$ 对 $[Y]_{补}$ 逐位取反再在最低位加 1。（包括符号位）

实现补码加减运算的逻辑电路



加

F ← X
F ← Y
X ← F

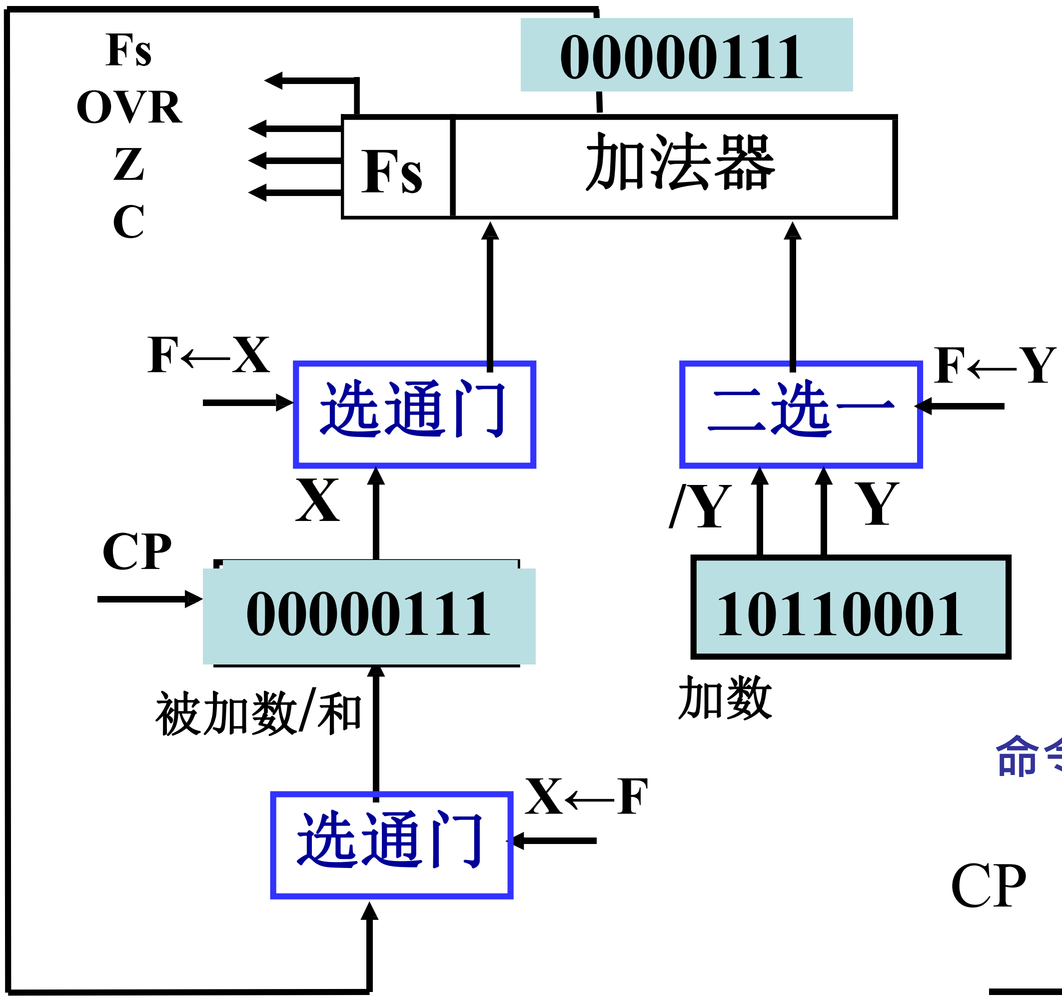
X ← X+Y

X ← X-Y

减

F ← X
F ← /Y
F ← 1
X ← F

实现补码加运算的执行过程



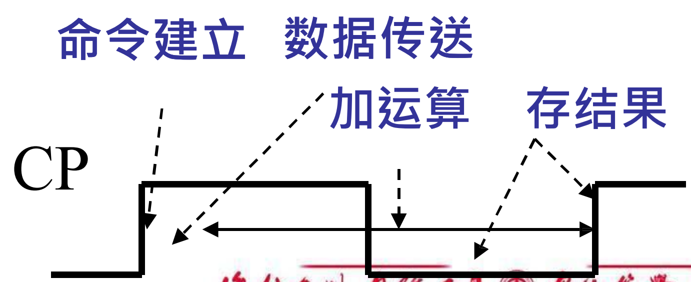
X ← X+Y

完成加运算，需要把被加数和加数送 ALU 的输入端，运算结果要接收到累加器，需要给出命令：

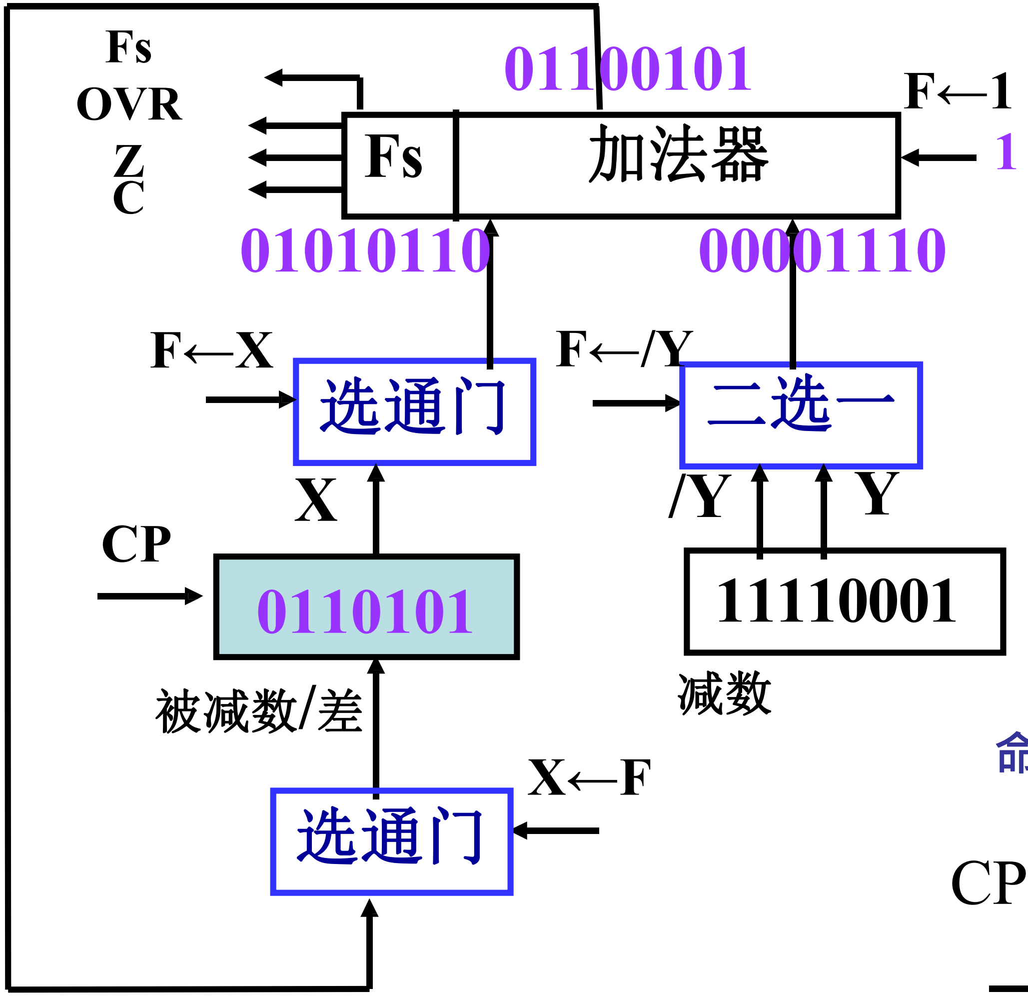
F ← X

F ← Y

X ← F



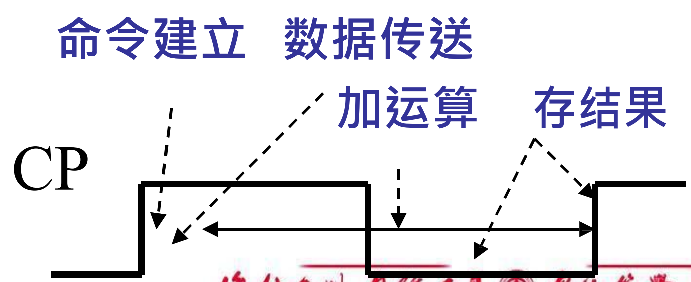
实现补码减运算的逻辑电路



X ← X−Y

完成减运算，需要把被减数和减数送 ALU 的输入 端，运算结果要 接收到累加器， 需要给出命令：

F ← X F ←/Y ，F←1



（4）溢出概念和判别方法

- 当运算结果超出机器数所能表示的范围时，称为溢出。两个异号数相加或两个同号数相减，其结果是不会溢出的。仅当两个同号数相加或者两个异号数相减时，才有可能发生溢出，一旦溢出，运算结果就不正确了，因此必须将溢出的情况检查出来。判别方法有三种：

①当符号相同的两数相加时，如果结果的符号与加数（或被加数）不相同，则为溢出。

② 当任意符号两数相加时，如果 **C=Cf**，运算结果正确， 其中 **C** 为数值最高位的进位，**Cf** 为符号位的进位。如果 **C≠Cf**，则为溢出，所以溢出条件=**C⊕Cf**

③采用双符号 **fs2fs1**，正数的双符号位为 **00**，负数的双符号位为 **11**。符号位参与运算，当结果的两个符号位不相同时为溢出，所以溢出条件= **fs2⊕fs1**

补码加减法溢出判断

- 方法之一：

  单符号位，正 + 正 得负　或　负 + 负 得正

- 方法之二：　数值位有向符号位的进位，但符号位不产生向更高位的

  进位，数值位没有向符号位的进位，但符号位产生向更高位的进位

- 方法之三：

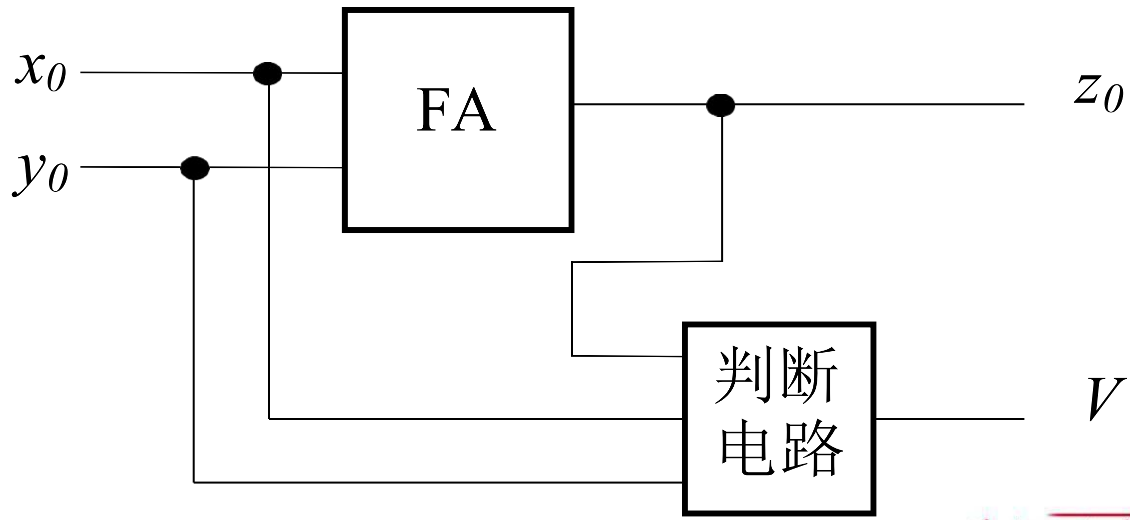
  双符号位的结果为 01 或 10，最高符号位 代表其真正的符号

单符号位判断

正加正得负或负加负得正表明溢出

$$V = x_0 \; y_0 \; \overline{z_0} \; + \; \overline{x_0} \; \overline{y_0} \; z_0$$

判断电路



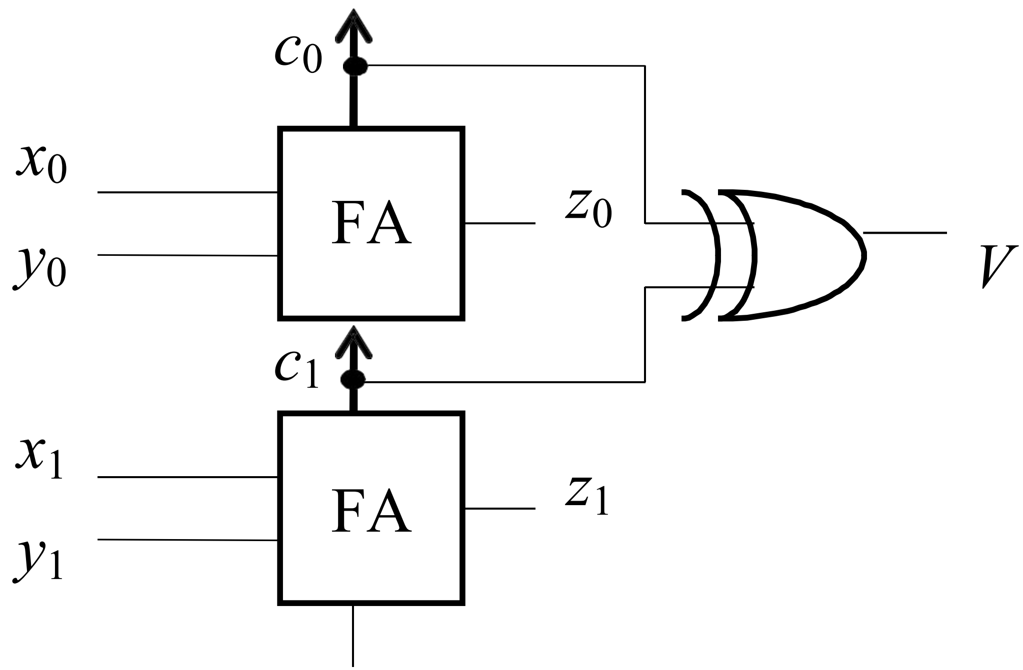
符号位与最高数值位判断

数值位向符号位有进位但符号位无进位输出或

数值位向符号位没有进位但符号位本身有进位输出是溢出

$V = C_0 \overline{C_1} + \overline{C_0} C_1$

判断电路

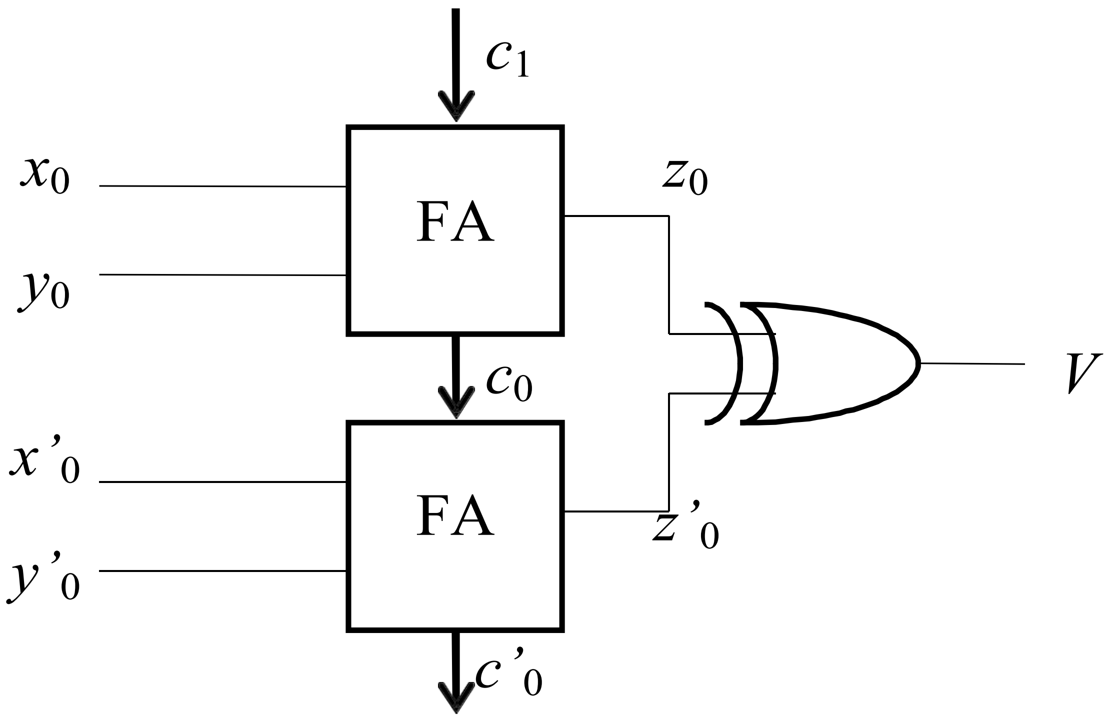


双符号位判断

$$V = z_0'\overline{z_0} + \overline{z'_0}\,z_0 = z_0' \oplus z_0$$

运算结果的 2 个符号位的值不相同表明有溢出

判断电路



补码加减法运算实例

判断溢出的 3 套方案是一个事实的 3 种不同的表述

X = 0.1011　　y = -0.0101

$[X]_{补}$ = 00 1011,　$[Y]_{补}$ = 11 1011　补码

$[-Y]_{补}$ = 00 0101

```
  00 1011
 +11 1011
 ---------
 100  0110
```

X+Y (不溢出)

正数加负数不会溢出

符号位和数值位都产生进位

双符号位结果相同不是溢出

```
   00  1011
 + 00  0101
 ----------
   01  0000
```

X-Y (溢出)

正数加正数结果为负是溢出

数值位有进位符号位无进位是溢出

双符号位结果不相同是溢出

真题解析

单选题：（2009 年）一个 C 语言机器在一台 32 位机器上运行。程序中定义了三个变量 x,y 和 z，其中 x 和 z 是 int 型，y 为 short 型。当 x = 127,y = -9 时，执行赋值语句 z = x+y 后，x,y 和 z 的值分别是（ ）。

A. x = 0000007FH， y = FFF9H， z = 00000076H

B. x = 0000007FH， y = FFF9H， z = FFFF0076H

C. x = 0000007FH， y = FFF7H， z = FFFF0076H

D. x = 0000007FH， y = FFF7H， z = 00000076H

分析： short 型数据为 16 位；int 型为 32 位（从选择项中也可知）。对于 y=-9，在机器中用补码表示为 FFF7H（关键），计算 z=x+y=0000007FH+FFFFFFF7H（符号扩展为 32 位）= 00000076H

技巧：可用排除法，得到 =-9 的补码为 FFF7H 可排除 A 和 B，再由 127+（-9）=118 可知 z 为正数，即可排除 C，再验证 D 中 z=76H=118，确认选 D。

考查知识点： 16 进制负数的补码表示，符号扩展，补码运算。

答案： D

P47 二.4 题

(5) 舍入处理

- 对于 固定字长的数，右移将舍去低位部分。

- 两种常用的舍入规则：

  - 0 舍 1 入法：
  - 末位恒置 1 法：

（6）定点数的乘/除运算

■ 定点乘法运算

- 原码一位乘法：两个原码数相乘，其乘积的符号为相乘两数的异或值，数值为两数绝对值之积。

- 定点补码一位乘法：有的机器为方便加减法运算，数据以补码形式存放。校正法和比较法

✓校正法：乘法直接用补码进行，以减少转换次数。具体规则如下：

$$[X\times Y]_{补}=[X]_{补}\times(0.Y1Y2\cdots Yn)+[-X]_{补}Y0$$

当 Y 为负时，求$[X\times Y]_{补}$，需要用$[X]_{补}$乘上$[Y]_{补}$的数值位，再加上$[-X]_{补}$

该算法需要区分乘数的符号，不方便。

✓比较法—布斯（Booth）法：用相邻两位乘数比较的结果决定加[X]补、[—X]补或 0。

布斯公式：在乘数 Y_n 后添加 Y_{n+1}=0。按照 Y_{n+1}，Y_n 相 邻两位的三种情况，其运算规则如下：

$Y_{n+1}Y_n$ =00 或 11，部分积加 0，右移 1 位;

$Y_{n+1}Y_n$ =10 ，部分积加[-X]补，右移 1 位;

$Y_{n+1}Y_n$ =01 ，部分积加[X]补，右移 1 位 最后一步不移位。

原码乘运算-手算方案

$[X^*Y]_{原} = (X_S \oplus Y_S)(|X|^*|Y|)$ 例

符号异或，
绝对值相乘

如： X = 0.1101　　Y = 0.1011

```
        0. 1 1 0 1
   *    0. 1 0 1 1
   ---------------
           1 1 0 1
         1 1 0 1
       0 0 0 0
  +  1 1 0 1
  -----------------
  0 . 1 0 0 0 1 1 1 1
```

手工运算过程

最终乘积原码表示：

0 1 0 0 0 1 1 1 1

该方案用于计算机会有问题：

1. 加法器只有两个数据输入端
2. 加法器与乘运算数据位数相同
3. 如何判断乘数每一位是 0 或者 1

解决方案：

1. 每次求出部分积,不是一次总累加
2. 变每次左移被乘数为右移部分积，移出的部分保存起来
3. 乘数放到一个移位寄存器中，用最低的一位来控制相加数（取被乘数或 0）。

原码一位乘法

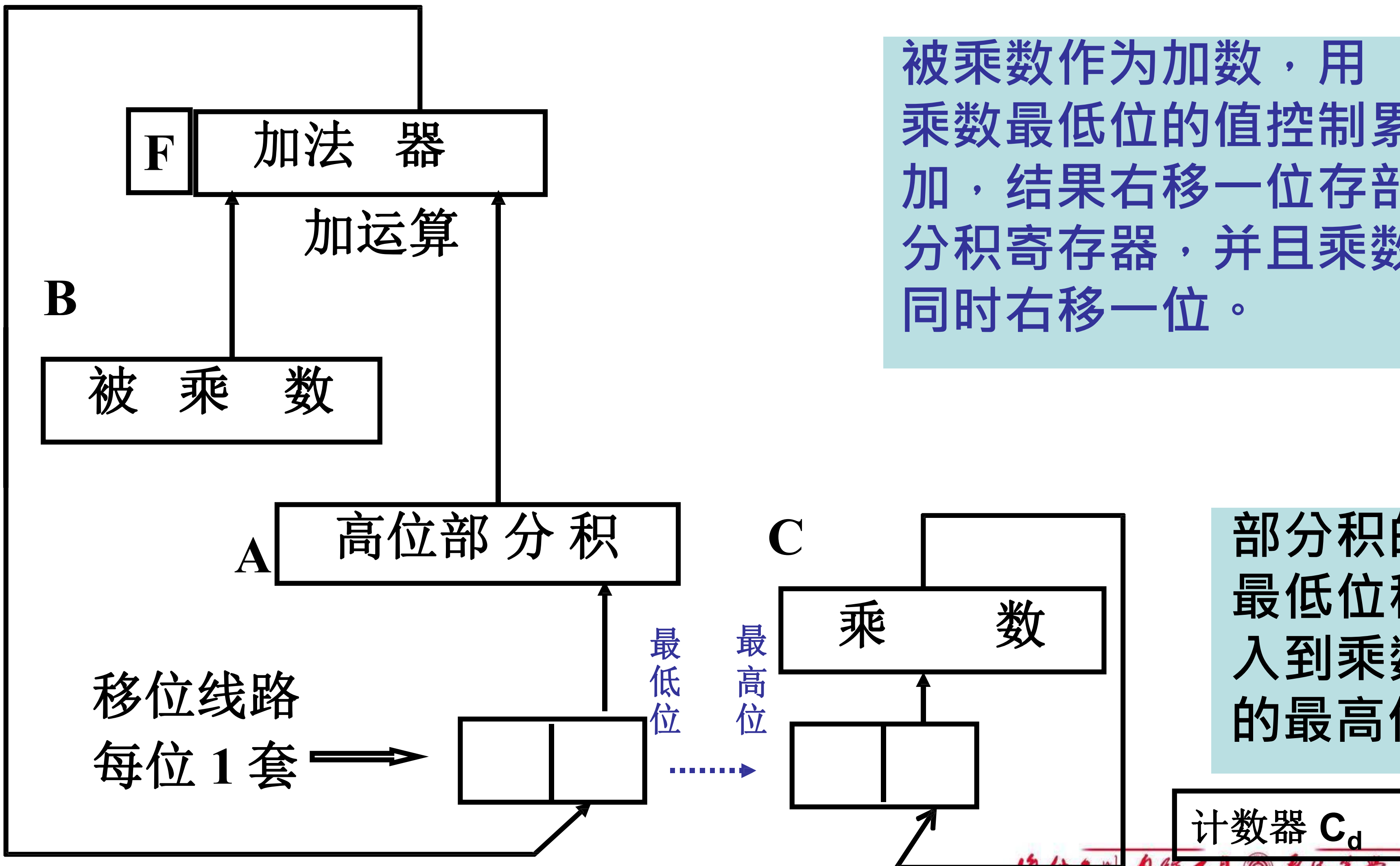
- 涉及三个寄存器：A 寄存器初始为零，作为初始部分积；B 寄存器，用来存放被乘数；C 寄存器，用来存放乘数。

- 实现部分积与被乘数相加是在 ALU 中完成的。

原码一位乘运算规则

① 操作数、结果用原码表示
② 绝对值运算，符号单独处理
③ 被乘数(B)、部分积(A)取双符号位
④ 乘数末位(Cn)为判断位，其状态决定下步操作
⑤ 作 n 次循环（累加、右移）

实现原码一位乘法的逻辑线路图



被乘数作为加数，用乘数最低位的值控制累加，结果右移一位存部分积寄存器，并且乘数同时右移一位。

部分积的最低位移入到乘数的最高位

原码一位乘法

- 乘法开始时，A 寄存器被清为零，作为初始部分积。被乘数放在 B 寄存器中，乘数放在 C 寄存器中。实现部分积与被乘数相加是在 ALU 中完成的。
- 每步运算，部分积最低一位的值将右移入 C 寄存器 的最高数值位，使相乘之积的低位部分保存进 C 寄 存器中，原来的乘数在逐位右移过程中丢失掉。寄存器 A 最终存放为乘积的高 n 位，寄存器 C 最终存放乘积的低 n 位。
- 另外还需要一个计数器 C_d 用来控制逐位相乘的次数，它的初值存放为乘数的位数值，在计算的过程中每完成一位乘计算就执行减 1 操作，待计数到 0 时，给出结束乘运算的控制信号。

原码一位乘运算规则

①操作数、结果用原码表示

②绝对值运算，符号单独处理

③被乘数(B)、部分积(A)取双符号位

④乘数末位(Cn)为判断位，其状态决定下步操作

⑤作 n 次循环（累加、右移）

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。

原码一位乘运算规则推导

以小数为例

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

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

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

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

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 $x^* \cdot y^*$

原码一位乘运算过程举例

X = 0.1101　　Y = 0.0011

高位部分积	低位部分积/乘数		说明
00 0000	0 0 1 1		起始情况
+ 00 1101			乘数最低位为 1，加 X
00 1101			
00 0110	1 0 0 1	1（丢失）	右移部分积和乘数
+ 00 1101			乘数最低位为 1，加 X
01 0011			
00 1001	1 1 0 0	1（丢失）	右移部分积和乘数
+ 00 0000			乘数最低位为 0，加 0
00 1001			
00 0100	1 1 1 0	0（丢失）	右移部分积和乘数
+ 00 0000			乘数最低位为 0，加 0
00 0100			
00 0010	0 1 1 1	0（丢失）	右移部分积和乘数

【例】设 X=0.8125,Y=0.6875,用原码 1 位乘的方法，求 X×Y。 其中 寄存器 B=X ,计数器 Cd=4。计算过程:

$[X]_{原}=0.1101$,

$[Y]_{原}=0.1011$,

	部分积 A	乘数 C	
	0 0 0 0 0 0	1 0 1 1	
+x	0 0 1 1 0 1		
	0 0 1 1 0 1		
右移一位→	0 0 0 1 1 0	1 1 0 1	1(丢失)
+x	0 0 1 1 0 1		
	0 1 0 0 1 1		
右移一位→	0 0 1 0 0 1	1 1 1 0	1(丢失)
+0	0 0 0 0 0 0		
	0 0 1 0 0 1		
右移一位→	0 0 0 1 0 0	1 1 1 1	0(丢失)
+x	0 0 1 1 0 1		
	0 1 0 0 0 1		
右移一位→	0 0 1 0 0 0	1 1 1 1	1(丢失)
	乘积高位	乘积低位	

符号位:

$x_0 \oplus y_0=0$

X•Y=0.10001111

原码一位乘运算规则

① 操作数、结果用原码表示

② 绝对值运算，符号单独处理

③ 被乘数(B)、部分积(A)取双符号位

④ 乘数末位(Cn)为判断位，其状态决定下步操作

⑤ 作 n 次循环（累加、右移）

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

原码一位分步乘法 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，并移位。

硬件：设置 3 个寄存器（具有右移位功能）：

A：存放部分积累加和、乘积高位

B：存放被乘数 C：存放乘数、乘积低位

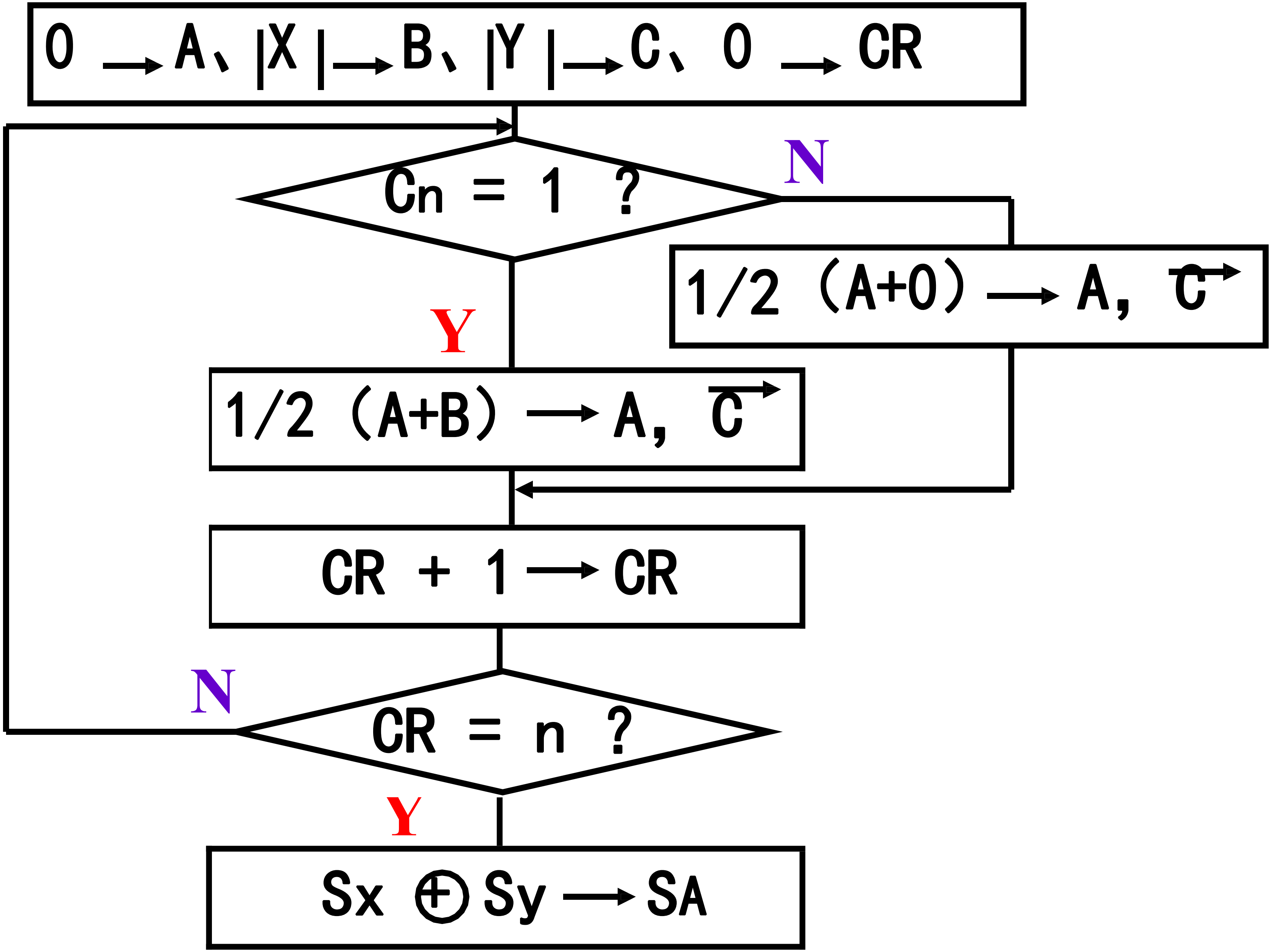
一个全加器 设置初值：

$$A = 00.0000$$

$$B = X = 00.1101$$

$$C = |Y| = .1011$$

算法流程



定点补码一位乘法

- 原码乘法存在的缺点是符号位需要单独运算，并要在最后给乘积冠以正确的符号。
- 补码乘法是指采用操作数的补码进行乘法运算，最后乘积仍为补码，能自然得到乘积的正确符号。

算法分析:校正法

$X_补 = X_0.X_1X_2......X_n$

① Y为正：$Y_补 = 0.Y1Y2......Yn$

$(XY)_补 = X_补(0.Y1Y2......Yn)$

② Y为负：$Y_补 = 1.Y1Y2......Yn$

由 $Y = -Y0 + 0.Y1Y2......Yn$，得

$(XY)_补 = X_补(0.Y1Y2......Yn)+(-X)_补$

③ Y符号任意：

$(XY)_补 = X_补(0.Y1Y2......Yn)+(-X)_补 Y0$

符号位

算法分析：校正法

$X_{补} = X_0.X_1X_2……X_n$

① Y 为正：$Y_{补} = 0.Y_1Y_2……Y_n$

校正法的过程是先按原码乘法那样直接乘，最后再根据乘数符号进行校正。

$(XY)_{补} = X_{补}(0.Y_1Y_2……Y_n)$

其算法规则如下：

② Y 为负：$Y_{补} = 1.Y_1Y_2……Y_n$

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。如果乘数 $Y_{补}$ 为负，则先按原码乘法运算，结果再加上一个校正量 $-X_{补}$。

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

③ Y 符号任意：

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

符号位

计算机组成原理 丁男

比较法算法 （Booth 算法）：

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

比较法— Booth 算法：用相邻两位乘数比较的结果决定 +X 补、-X 补或+0。

Booth 算法运算规则

①部分积 A、被乘数 B 取双符号位，**符号位参加运算**；

② 乘数 C 取单符号位，符号参加移位，以决定最后是否 修正；

③ C 末位设置附加位 Cn+1，初值为 0，CnCn+1 组成判断
位，决定运算操作，作 n 步循环；

④ 第 n+1，仅修正，不移位。

注意，最后一步不移位，因为这一步是用来处理符号位的。

【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=0.1011

计算过程:

A=00.0000,
B =[X]补=11.0011,
-B=[-X]补=00.1101,
C =[Y]补=0.1011

	部分积	乘数 Y Yi Yi+1	说明
	0 0 0 0 0 0	0. 1 0 1 1 0	初始值,最后一位补 0
+	0 0 1 1 0 1		Y4Y5=10 +[-X]补
	0 0 1 1 0 1		
→	0 0 0 1 1 0	1 0 1 0 1 1	右移一位
+	0 0 0 0 0 0		Y3Y4=11 +0
	0 0 0 1 1 0		
→	0 0 0 0 1 1	0 1 0 1 0 1	右移一位
+	1 1 0 0 1 1		Y2Y3=01 +[X]补
	1 1 0 1 1 0		
→	1 1 1 0 1 1	0 0 1 0 1 0	右移一位
+	0 0 1 1 0 1		Y1Y2=10 +[-X]补
	0 0 1 0 0 0		
→	0 0 0 1 0 0	0 0 0 1 0 1	右移一位
+	1 1 0 0 1 1		Y0Y1=01 +[X]补
	1 1 0 1 1 1	0 0 0 1	不移位
	乘积高位	乘积低位	

[XY]补=1.01110001, XY= - 0.10001111

定点除法运算

- 定点原码一位除法

✓ 恢复余数法：被除数（余数）减去除数，如果为 0 或者为正值时，上商为 1，不恢复余数；如果结果为负，上商为 0，再将除数加到余数 中，恢复余数。余数左移 1 位。

✓ 加减交替法：当余数为正时，商上 1，求下一位商的办法，余数左移一位，再减去除数；当余数为负时，商上 0，求下一位商的办法，余 数左移一位，再加上除数。

- 定点补码一位除法（加减交替法）

✓ 如果被除数与除数同号，用被除数减去除数；若两数异号，被除数加上除数。如果所得余数与除数同号商上 1，否则，商上 0，该商为 结果的符号位。

✓ 求商的数值部分。如果上次商上 1，将除数左移一位后减去除数；如 果上次商上 0，将余数左移一位后加除数。然后判断本次操作后的余 数，如果余数与除数同号商上 1，如果余数与除数异号商上 0。如此 重复执行 n-1 次（设数值部分 n 位）。

✓ 商的最后一位一般采用恒置 1 的办法，并省略了最低+1 的操作。此时最大的误差为 2^{-n}。

【例】设被除数 X=0.1011, Y=0.1101, 用原码加减交替法求 X/Y=?

解：设置寄存器：

A 寄存器中开始时存放被除数的绝对值，以后将存放各次余数，取双符号位。

B 寄存器存放除数的绝对值，取双符号位。

C 寄存器同来存放商，取单符号位。

加减交替法处理思想：先减后判，如减后发现不够减，则在下一步改作加除数操作。要点：

- ✓ 被除数|X|<除数|Y|，取原码尾数的绝对值相除，符号位单独处理，商 的符号为相除两数符号的异或。
- ✓ 被除数的位数要扩展成除数位数 n 的两倍(2n 位)，其低位的数值部分开始时放在商寄存器中。在具体运算中，放被除数和商的 A、C 寄存器同时 移位，并将商寄存器 C 中最高位移到被除数寄存器 A 的最低位中。

$[|Y|]_{补}=00.1101$， $[-|Y|]_{补}=11.0011$

计算过程:

	被除数(余数 R)	（被除数)(商）	操作说明
	00 1011	0 0 0 0 0	开始情形
+)	11 0011		+[-Y]补
	11 1110	0 0 0 0 0	不够减,商上 0
	11 1100	0 0 0 0 0	左移
+)	00 1101		+Y
	00 1001	0 0 0 0 1	够减,商上 1
	01 0010	0 0 0 1 0	左移
+)	11 0011		+[-Y]补
	00 0101	0 0 0 1 1	够减,商上 1
	00 1010	0 0 1 1 0	左移
+)	11 0011		+[-Y]补
	11 1101	0 0 1 1 0	不够减,商上 0
	11 1010	0 1 1 0 0	左移
+)	00 1101		+Y
	00 0111	0 1 1 0 1	够减,商上 1
	余数	商	

商 X/Y=0.1101,

余数 R=0.0111