计算机组成原理

翁睿

哈尔滨工业大学

Part 3

第 6 章　计算机的运算方法

6.1 无符号数和有符号数

6.2 数的定点表示和浮点表示

6.3 定点运算

6.4 浮点四则运算

6.5 算术逻辑单元

4. 原码乘法

(1) 原码一位乘运算规则

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 $x^* \cdot y^*$

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

参考笔算乘法运算

- ✓ 符号位单独处理
- ✓ 乘数的某一位决定是否加被乘数
- ？ 4个位积一起相加
- ✓ 乘积的位数扩大一倍

例4.21 已知 $x = -0.1110$ $y = 0.1101$ 求$[x \cdot y]_{原}$ 4.3

解：数值部分的运算

部分积	乘数	说明
0.0000 + 0.1110	1101	部分积 初态 $z_0 = 0$ + x^*
0.1110 逻辑右移 0.0111 + 0.0000	0110	→1，得 z_1 + 0
0.0111 逻辑右移 0.0011 + 0.1110	0 1011	→1，得 z_2 + x^*
1.0001 逻辑右移 0.1000 + 0.1110	10 1101	→1，得 z_3 + x^*
1.0110 逻辑右移 0.1011	110 0110	→1，得 z_4

例4.21 结果

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

② 数值部分按绝对值相乘

$$x^* \cdot y^* = 0.1\,0\,1\,1\,0\,1\,1\,0$$

则 $[x \cdot y]_{原} = 1.1\,0\,1\,1\,0\,1\,1\,0$

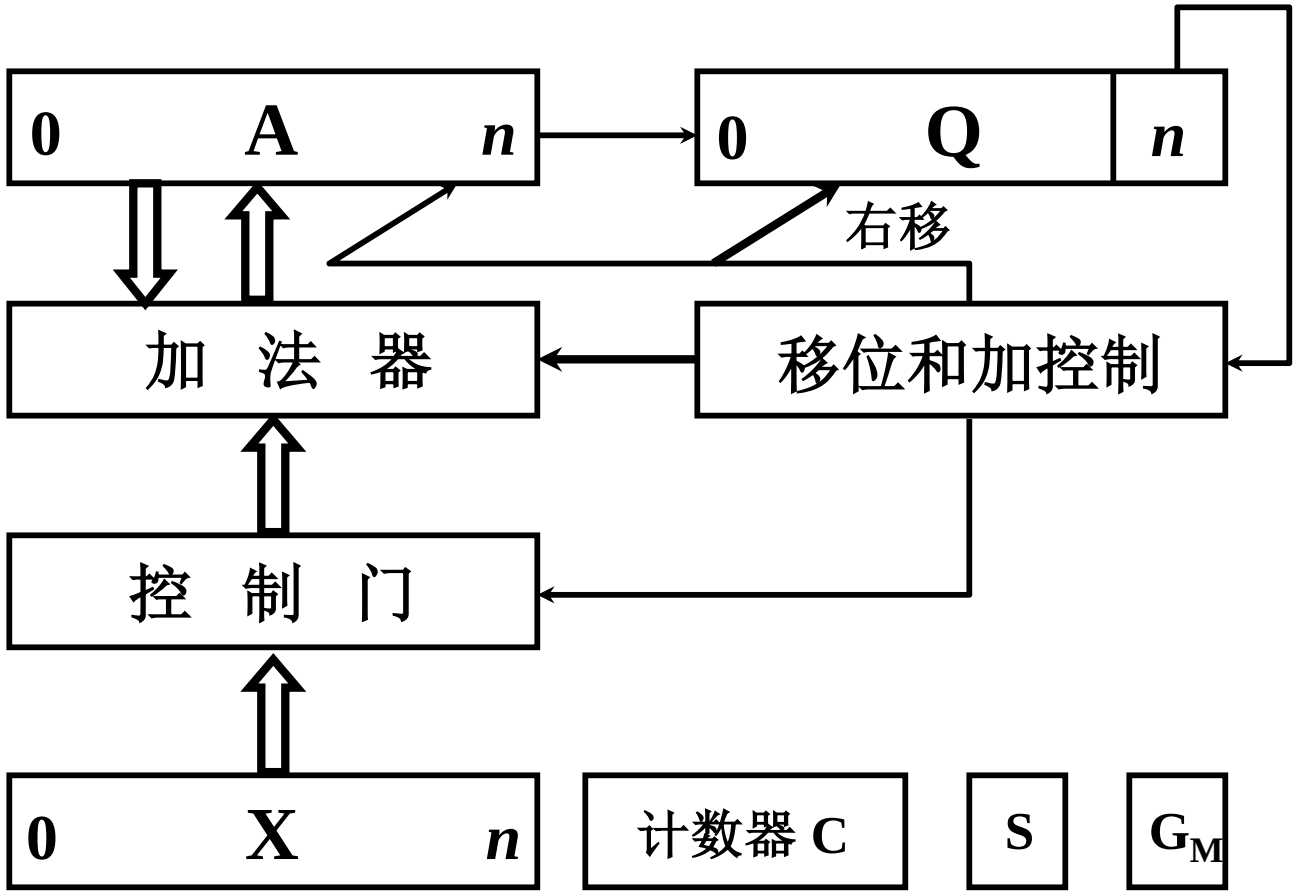
特点　绝对值运算（遵循无符号数规则）

用移位的次数判断乘法是否结束

逻辑移位

移位 n 次
加 n 次

(3) 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

5. 补码乘法

(1) 补码一位乘运算规则

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

乘数　$[y]_{补} = y_0.y_1y_2 \cdots y_n$

① 被乘数任意，乘数为正

同原码乘　但 加 和 移位 按 补码规则 运算

乘积的符号自然形成

② 被乘数任意，乘数为负

乘数$[y]_{补}$，去掉符号位，操作同 ①

最后 加$[-x]_{补}$，校正

④ Booth 算法

递推公式

$[z_0]_{补} = 0$

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

⋮

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

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

$y_{i+1} - y_i$ 的实现方法

y_i y_{i+1}	$y_{i+1} - y_i$	操作
0 0	0	→1
0 1	1	$+[x]_{补}$ →1
1 0	-1	$+[-x]_{补}$ →1
1 1	0	→1

例4.23 已知 $x = +0.0011$ $y = -0.1011$ 求$[x \cdot y]_{补}$

解：

00.0000	1.0101	0
+ 11.1101		$+[-x]_{补}$
11.1101		
11.1110	1 1010	1 → 1
+ 00.0011		$+[x]_{补}$
00.0001	1	
00.0000	11 101	0 → 1
+ 11.1101		$+[-x]_{补}$
11.1101	11	
11.1110	111 10	1 → 1
+ 00.0011		$+[x]_{补}$
00.0001	111	
00.0000	1111 1	0 → 1
+ 11.1101		$+[-x]_{补}$
11.1101	1111	最后一步不移位

补码右移

补码右移

补码右移

补码右移

$[x]_{补} = 0.0011$

$[y]_{补} = 1.0101$

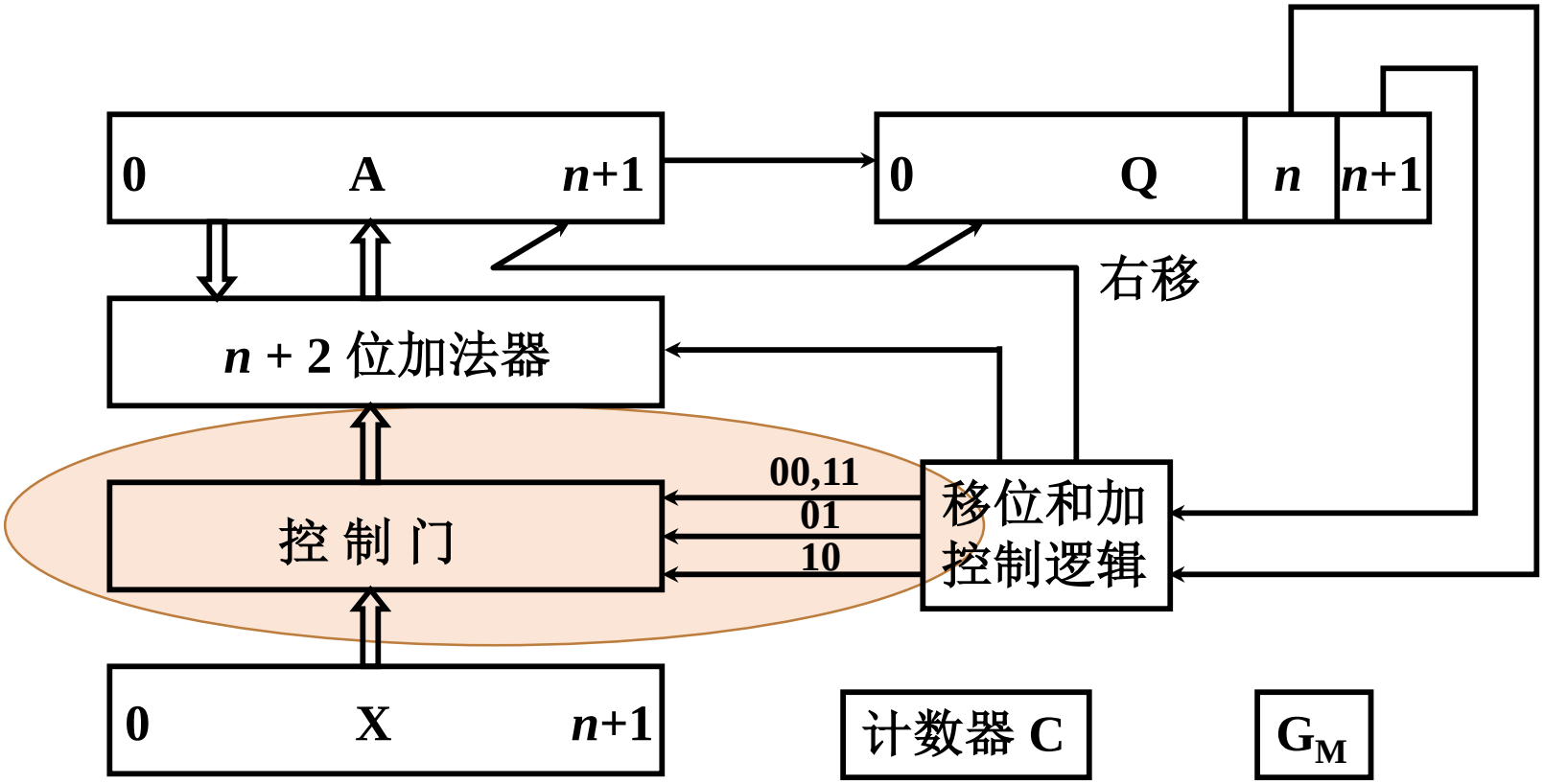
$[-x]_{补} = 1.1101$

$\therefore [x \cdot y]_{补}$

$= 1.11011111$

移位 n 次
加 $n+1$ 次

(2) Booth 算法的硬件配置



A、X、Q　均 $n+2$ 位

移位和加法操作受乘数末两位控制

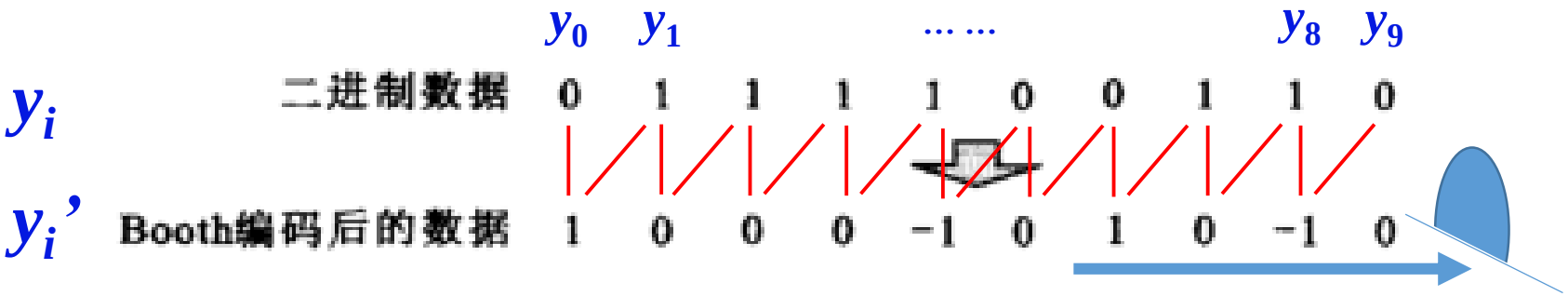
•Booth编码：

乘数中的每两位对应基-2 Booth编码中的一位。

Booth 编码中的每一位决定在当前位置加被乘数的系数

总结规律：

乘数火车向右走，
首 1 进洞减被乘。
连串 0 1 不加减，
尾 1 进洞加被乘。
被乘加减交替算，
加比移位多一程。



$+[x]_{补} \cdot y_i' \longrightarrow 1$

乘法小结

4.3

- 整数乘法与小数乘法完全相同
  可用 逗号 代替小数点

- 原码乘　符号位 单独处理
  补码乘　符号位 自然形成

- 原码乘去掉符号位运算　即为无符号数乘法

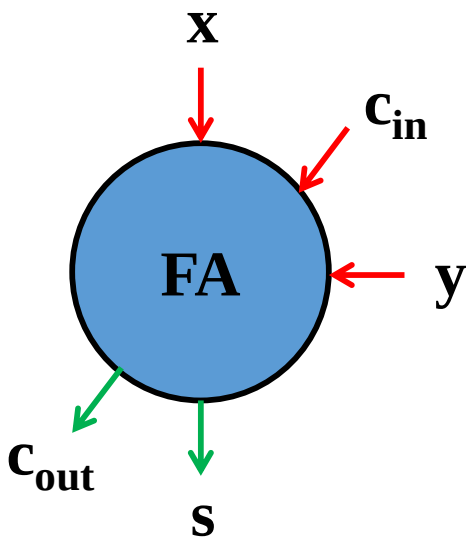
- 不同的乘法运算需有不同的硬件支持

● 快速乘法器

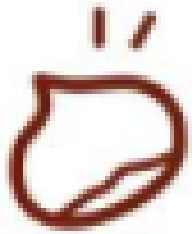
- ①阵列乘法器：
- 最直白的思路，用n^2个与门和n个加法器实现。

- 首先，为方便分析阵列乘法器的工作流程，我们把全加器画为如下图的形式：

* 注意每个位置对应的输入输出信号是什么。（下面要用到）



● 快速乘法器



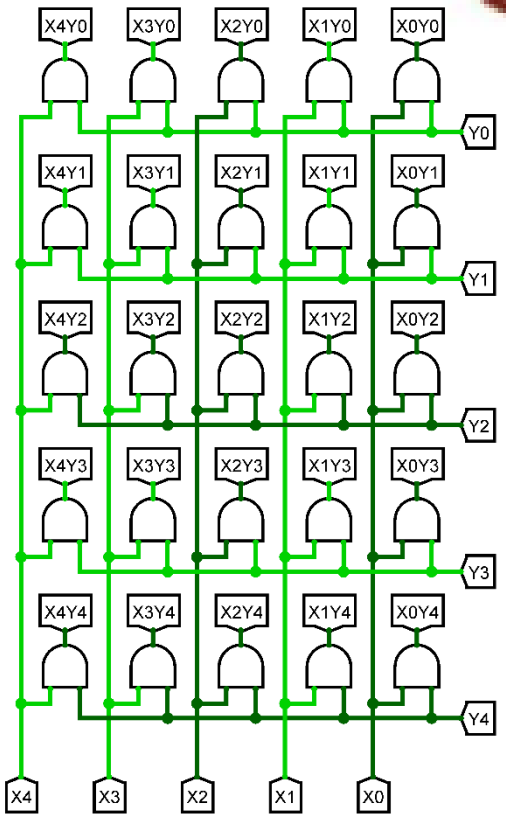
- **阵列乘法器**
- **举个例子：构造一个5位*5位的阵列乘法器：**
  - **被乘数：$a_4a_3a_2a_1a_0$　乘数：$b_4b_3b_2b_1b_0$　结果：R=a*b**

$$
\begin{array}{cccccccccc}
 & & & a_4 & a_3 & a_2 & a_1 & a_0 & & \\
 & & \times & b_4 & b_3 & b_2 & b_1 & b_0 & & \\
\hline
 & & & a_4b_0 & a_3b_0 & a_2b_0 & a_1b_0 & a_0b_0 & & \\
 & & a_4b_1 & a_3b_1 & a_2b_1 & a_1b_1 & a_0b_1 & & & \\
 & a_4b_2 & a_3b_2 & a_2b_2 & a_1b_2 & a_0b_2 & & & & \\
a_4b_3 & a_3b_3 & a_2b_3 & a_1b_3 & a_0b_3 & & & & & \\
+\ a_4b_4 & a_3b_4 & a_2b_4 & a_1b_4 & a_0b_4 & & & & & \\
\hline
R_8 & R_7 & R_6 & R_5 & R_4 & R_3 & R_2 & R_1 & R_0
\end{array}
$$

● 快速乘法器

一位乘法的逻辑实现

- R=X*Y
  - 1 × 1 = 1
  - 1 × 0 = 0
  - 0 × 1 = 0
  - 0 × 0 = 0
- 与门实现一位乘法
- 用25个与门并发计算
- 一级门延迟，生成所有位积

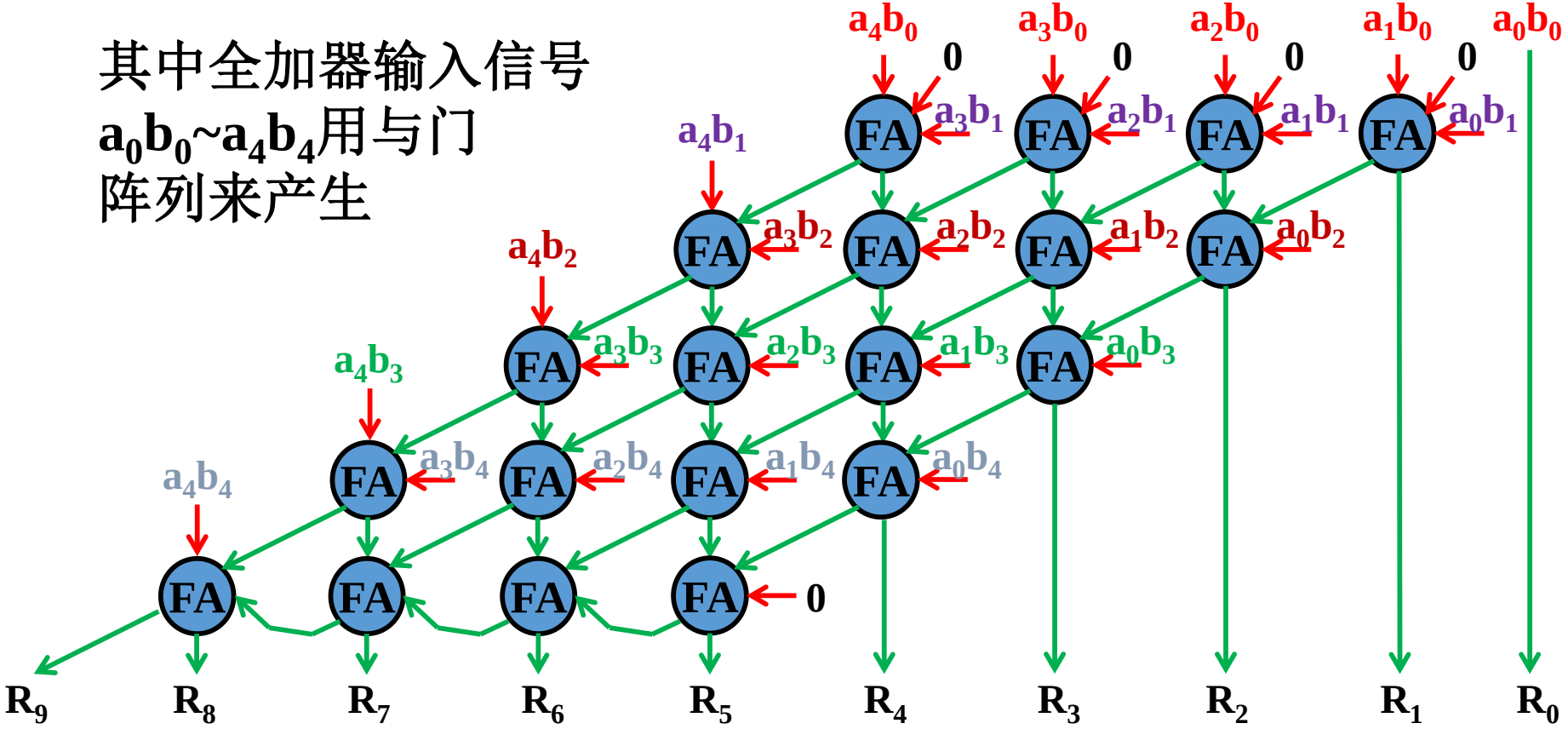


● 快速乘法器

- 阵列乘法器

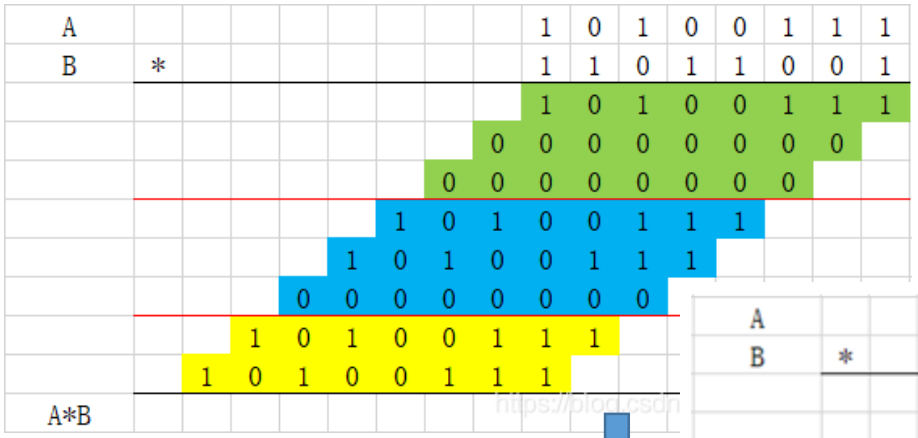
- 举个例子：构造一个5位*5位的斜向进位阵列乘法器：
  - 被乘数：$a_4a_3a_2a_1a_0$　乘数：$b_4b_3b_2b_1b_0$　结果：R=a*b

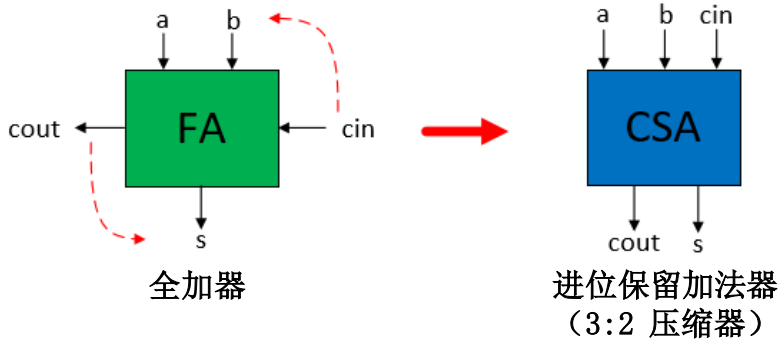
其中全加器输入信号a_0b_0~a_4b_4用与门阵列来产生

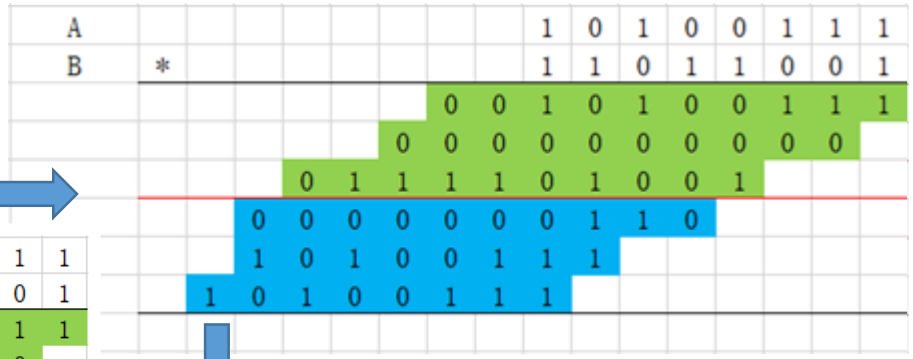


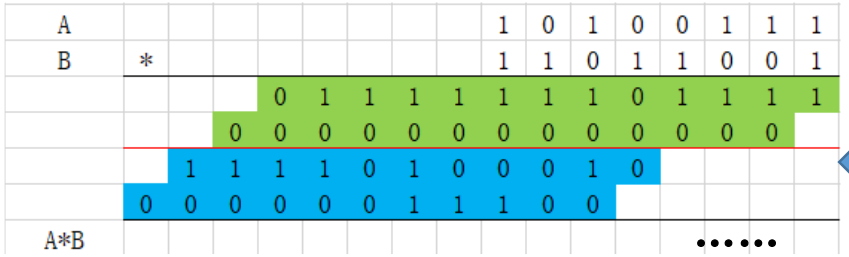
● 快速乘法器

➢华莱士树快速求和过程的主要思想：将求和项每3组压缩为2组，如此反复直至仅剩2组；最后用超前进位加法器求和并考虑进位的处理。









- 方法：将多个加法器组织成一个并行树 (Wallace Tree)
- 优点：易于应用流水线设计执行，可以同步支持多个乘法

乘法器性能提升小结

6.3

- **核心算法：n个部分积累加**
- **Booth一位乘 → Booth两位乘**
  - 一位乘法：n个全加器，n^2个全加器时延，面积小 (Intel 8086)
  - 两位乘法： 减少部分积的个数，速度更快，增加额外电路
- **斜向进位阵列乘法器 → 华莱士树**
  - 斜向进位：（n^2-n）个全加器，n级全加器时延，面积大
  - 华莱士树： 更多全加器，$\log_2 n$ 级全加器时延，面积更大
- **主流乘法器**
  - **二位booth算法 + 华莱士树 + 流水**

四、除法运算

6.3

1. 分析笔算除法

$x = -0.1011 \quad y = 0.1101 \quad$ 求 $x \div y$

```
                0.1101
          ------------
0.1101 ) 0.10110
         0.01101
         -------
         0.010010
         0.001101
         --------
         0.0001001 0
         0.0000110 1
         -----------
         0.0000011 1
```

✓商符单独处理

? 心算上商

? 余数不动低位补“0”
减右移一位的除数

? 上商位置不固定

$x \div y = -0.1101$ 商符心算求得

余数 0.0000111

2. 笔算除法和机器除法的比较

笔算除法	机器除法
商符单独处理	符号位异或形成
心算上商	$\|x\|-\|y\|>0$ 上商 1 $\|x\|-\|y\|<0$ 上商 0
余数 不动 低位补“0” 减右移一位 的除数	余数 左移一位 低位补“0” 减 除数
2 倍字长加法器	1 倍字长加法器
上商位置 不固定	在寄存器 最末位上商

3. 原码除法

以小数为例

$$[x]_{原} = x_0.x_1x_2 \cdots x_n$$

$$[y]_{原} = y_0.y_1y_2 \cdots y_n$$

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

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

商的符号位单独处理 $x_0 \oplus y_0$

数值部分为绝对值相除 $\frac{x^*}{y^*}$

约定

小数定点除法 $x^* < y^*$　　整数定点除法 $x^* > y^*$

被除数不等于 0

除数不能为 0

(1) 恢复余数法

6.3

例6.24 $x = -0.1011$ $y = -0.1101$ 求 $[\frac{x}{y}]_{原}$

解：$[x]_{原} = 1.1011$ $[y]_{原} = 1.1101$ $[y^*]_{补} = 0.1101$ $[-y^*]_{补} = 1.0011$

① $x_0 \oplus y_0 = 1 \oplus 1 = 0$

②

被除数（余数）	商	说　明
0.1011	0.0000	
+ 1.0011		$+[-y^*]_{补}$
1.1110	0	余数为负，上商 0
+ 0.1101		恢复余数　$+[y^*]_{补}$
0.1011	0	恢复后的余数
1.0110	0	← 1
+ 1.0011		$+[-y^*]_{补}$
0.1001	01	余数为正，上商 1
1.0010	01	← 1
+ 1.0011		$+[-y^*]_{补}$

逻辑左移

逻辑左移

6.3

被除数（余数）	商	说　明
0.0101	011	余数为正，上商 1
逻辑左移 0.1010	011	← 1
+ 1.0011		$+[-y^*]_{补}$
1.1101	0110	余数为负，上商 0
+ 0.1101		恢复余数 $+[y^*]_{补}$
0.1010	0110	恢复后的余数
逻辑左移 1.0100	0110	← 1
+ 1.0011		$+[-y^*]_{补}$
0.0111	01101	余数为正，上商 1

$$\frac{x^*}{y^*} = 0.1101$$

$$\therefore \left[\frac{x}{y}\right]_{原} = 0.1101$$

上商 5 次

第一次上商判溢出

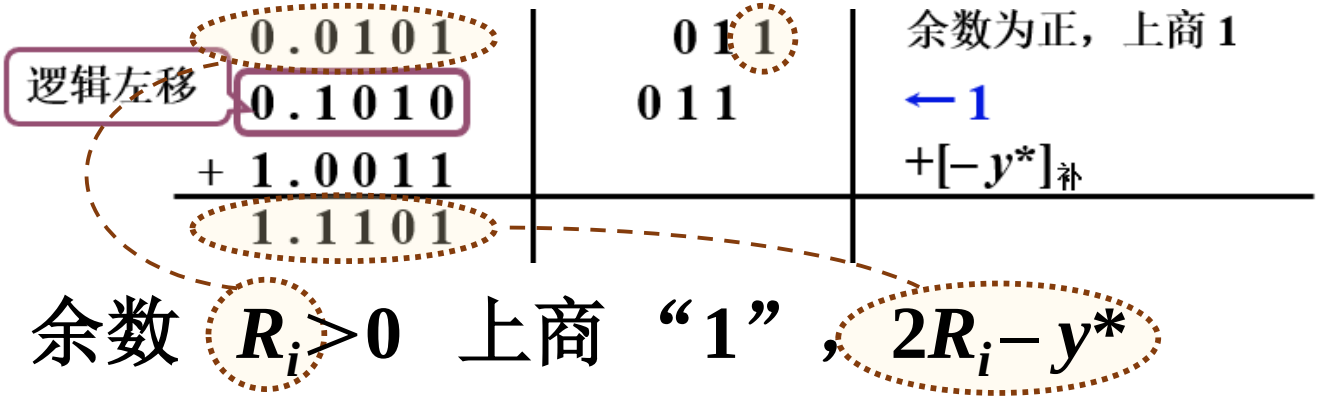
移 4 次

余数为正　上商 1

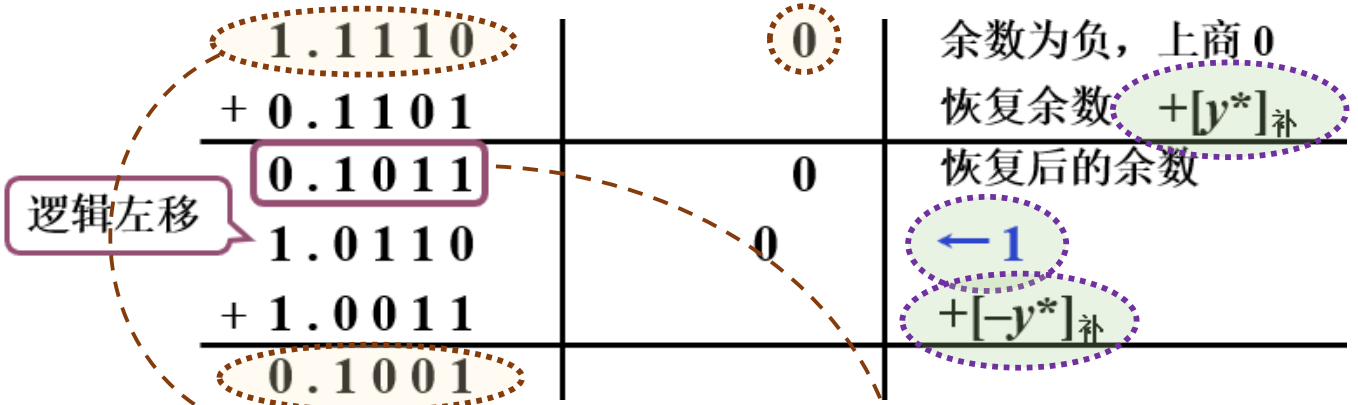
余数为负　上商 0，恢复余数

(2) 不恢复余数法

- 恢复余数法运算规则



余数 $R_i>0$ 上商 “1”，$2R_i-y^*$



余数 $R_i<0$ 上商 “0”，R_i+y^* 恢复余数

$$2(R_i+y^*)-y^*=2R_i+y^*$$

(2) 不恢复余数法（加减交替法） 6.3

- 恢复余数法运算规则

余数 $R_i>0$ 上商 “1”，$2R_i-y^*$

余数 $R_i<0$ 上商 “0”， R_i+y^* 恢复余数

$$2(R_i+y^*)-y^*=2R_i+y^*$$

- 不恢复余数法运算规则

余数 $R_i>0$ 上商 “1” $2R_i-y^*$

余数 $R_i<0$ 上商 “0” $2R_i+y^*$

加减交替

例6.25 $x = -0.1011$ $y = -0.1101$ 求 $[\frac{x}{y}]_{原}$ 6.3

解：

0.1011	0.0000	
+1.0011		$+[-y^*]_{补}$
1.1110	0	余数为负，上商 0
1.1100	0	← 1
+0.1101		$+[y^*]_{补}$
0.1001	01	余数为正，上商 1
1.0010	01	← 1
+1.0011		$+[-y^*]_{补}$
0.0101	011	余数为正，上商 1
0.1010	011	← 1
+1.0011		$+[-y^*]_{补}$
1.1101	0110	余数为负，上商 0
1.1010	0110	← 1
+0.1101		$+[y^*]_{补}$
0.0111	01101	余数为正，上商 1

逻辑左移

逻辑左移

逻辑左移

逻辑左移

$[x]_{原} = 1.1011$

$[y]_{原} = 1.1101$

$[x^*]_{补} = 0.1011$

$[y^*]_{补} = 0.1101$

$[-y^*]_{补} = 1.0011$

例6.25 结果

① $x_0 \oplus y_0 = 1 \oplus 1 = 0$

② $\frac{x^*}{y^*} = 0.1101$

$\therefore \ [\frac{x}{y}]_{原} = 0.1101$

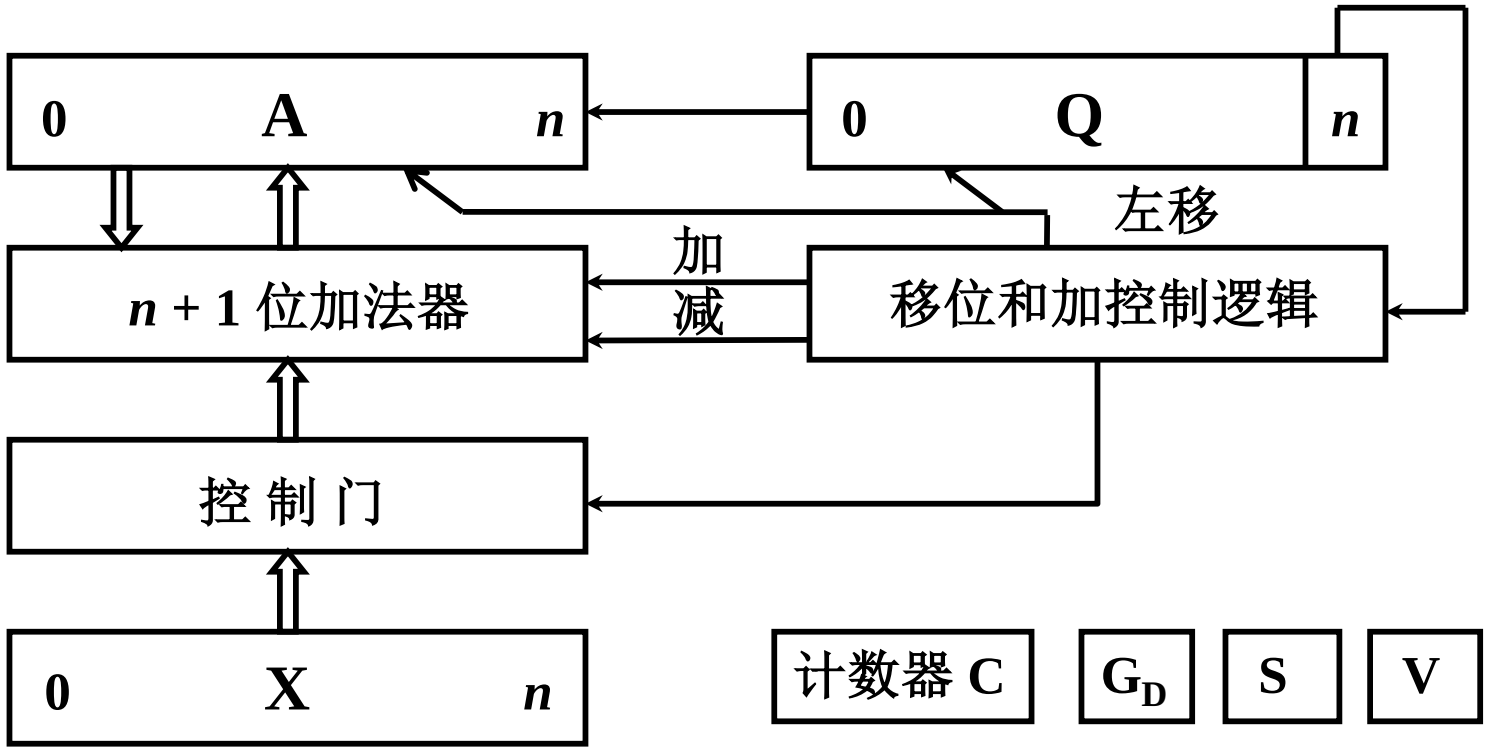
特点 上商 $n+1$ 次

第一次上商判溢出

移 n 次，加 $n+1$ 次

用移位的次数判断除法是否结束

(3) 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

6.4 浮点四则运算

一、浮点加减运算

$$x = S_x \cdot 2^{j_x} \qquad y = S_y \cdot 2^{j_y}$$

1. 对阶

(1) 求阶差

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

(2) 对阶原则

小阶向大阶看齐

例如 $x = 0.1101 \times 2^{01}$ $y = (-0.1010) \times 2^{11}$

6.4

求 $x+y$

解：$[x]_{补} = 00, 01; 00.1101$ $[y]_{补} = 00, 11; 11.0110$

1. 对阶

① 求阶差 $[\Delta j]_{补} = [j_x]_{补} - [j_y]_{补}$ $= 00, 01$

$+ \quad 11, 01$

$11, 10$

阶差为负（-2） $\therefore S_x \longrightarrow 2$ $j_x + 2$

② 对阶 $[x]_{补'} = 00, 11; 00.0011$

2. 尾数求和

$[S_x]_{补'} = 00.0011$ 对阶后的$[S_x]_{补'}$

$+ \quad [S_y]_{补} = 11.0110$

11.1001

$\therefore [x+y]_{补} = 00, 11; 11.1001$

3. 规格化

(1) 规格化数的定义

$$r=2 \qquad \frac{1}{2} \le |S| < 1$$

(2) 规格化数的判断

$S>0$	规格化形式	$S<0$	规格化形式
真值	0.1××…×	真值	–0.1××…×
原码	0.1××…×	原码	1.1××…×
补码	0.1××…×	补码	1.0××…×
反码	0.1××…×	反码	1.0××…×

原码　不论正数、负数，第一数位为1

补码　符号位和第 一数位不同

特例

$$S = -\frac{1}{2} = -0.100\cdots0$$

$$[S]_{原} = 1.100\cdots0$$

$$[S]_{补} = 1.100\cdots0$$

$\therefore$ $[-\frac{1}{2}]_{补}$ 不是规格化的数

$$S = -1$$

$$[S]_{补} = 1.000\cdots0$$

$\therefore$ $[-1]_{补}$ 是规格化的数

(3) 左规

尾数左移一位，阶码减 1，直到数符和第一数位不同为止

上例　$[x+y]_{补} = 00,\ 11;\ 11.\ 1001$

左规后　$[x+y]_{补} = 00,\ 10;\ 11.\ 0010$

$$\therefore\ x + y = (-0.1110) \times 2^{10}$$

(4) 右规

当 尾数溢出（ >1）时，需 右规

即尾数出现 01. ××　…×或 10. ××　…×时

尾数右移一位，阶码加 1

例6.27 $x = 0.1101\times 2^{10}$ $y = 0.1011\times 2^{01}$ 6.4

求 $x+y$（除阶符、数符外，阶码取 3 位，尾数取 6 位）

解：$[x]_{补} = 00, 010; 00.\ 110100$

$[y]_{补} = 00, 001; 00.\ 101100$

① 对阶

$[\Delta j]_{补} = [j_x]_{补} - [j_y]_{补}$ = 00, 010

\+ 11, 111

100, 001

阶差为 +1　　$\therefore S_y \longrightarrow 1,\ j_y+1$

$\therefore [y]_{补'} = 00, 010; 00.\ 010110$

② 尾数求和

$[S_x]_{补} = 00.\ 110100$

$+\ [S_y]_{补'} = 00.\ 010110$　　对阶后的$[S_y]_{补'}$

01. 001010　　尾数溢出需右规

③ 右规

$[x+y]_{补} = 00,010;\ 01.001010$

右规后

$[x+y]_{补} = 00,011;\ 00.100101$

$\therefore\ x+y = 0.100101 \times 2^{11}$

4. 舍入

在 对阶 和 右规 过程中，可能出现 尾数末位丢失 引起误差，需考虑舍入

(1) 0 舍 1 入法

(2) 恒置 “1” 法

6.4

例 6.28 $x = (-\frac{5}{8}) \times 2^{-5}$ $y = (\frac{7}{8}) \times 2^{-4}$

求 $x-y$（除阶符、数符外，阶码取 3 位，尾数取 6 位）

解： $x = (-0.101000) \times 2^{-101}$ $y = (0.111000) \times 2^{-100}$

$[x]_{补} = 11,011;\ 11.011000$ $[y]_{补} = 11,100;\ 00.111000$

① 对阶

$$[\Delta j]_{补} = [j_x]_{补} - [j_y]_{补} = 11,011 + 00,100 = 11,111$$

阶差为 –1 ∴ $S_x \longrightarrow 1$, $j_x + 1$

∴ $[x]_{补'} = 11,100;\ 11.101100$

② 尾数求和

$$[S_x]_{补'} = 11.101100$$
$$+\ [-S_y]_{补} = 11.001000$$
$$\overline{\qquad\qquad\qquad\qquad}$$
$$110.110100$$

③ 右规

$$[x-y]_{补} = 11,100;\ 10.110100$$

右规后

$$[x-y]_{补} = 11,101;\ 11.011010$$

$$\therefore\ x-y = (-0.100110)\times 2^{-11}$$

$$= \left(-\frac{19}{32}\right)\times 2^{-3}$$

5. 溢出判断

6.4

设机器数为补码，尾数为 规格化形式，并假设阶符取 2 位，阶码的数值部分取 7 位，数符取 2 位，尾数取 n 位，则该 补码 在数轴上的表示为