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

6.2 数的定点表示和浮点表示

小数点按约定方式标出

一、定点表示

$$\underset{\text{数符}}{\boxed{S_f}}\;\uparrow_{\text{小数点位置}}\underbrace{\boxed{S_1S_2\cdots S_n}}_{\text{数值部分}}$$

或

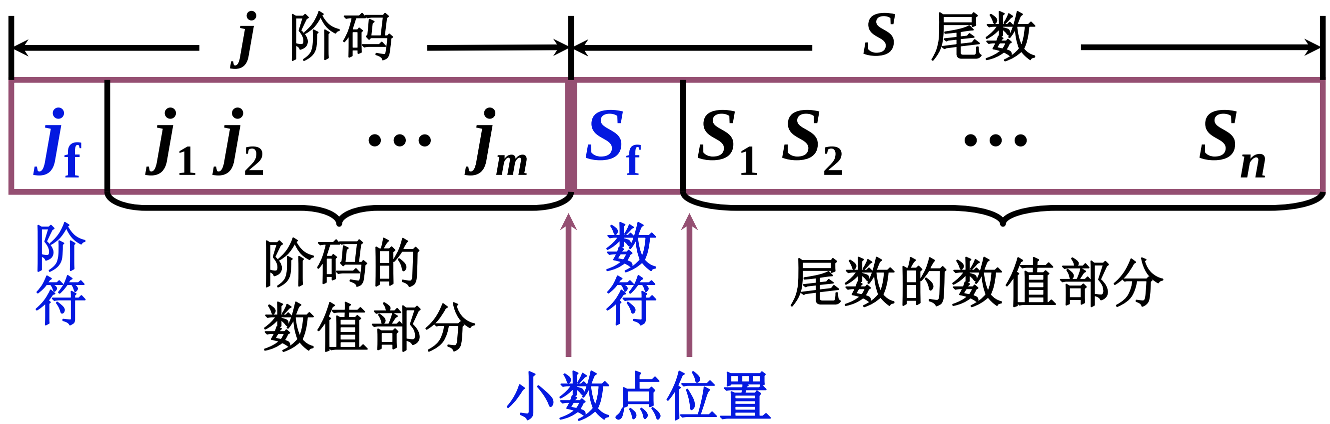
$$\underset{\text{数符}}{\boxed{S_f}}\;\underbrace{\boxed{S_1S_2\cdots S_n}}_{\text{数值部分}}\uparrow_{\text{小数点位置}}$$

定点机	小数定点机	整数定点机
原码	$-(1-2^{-n}) \sim +(1-2^{-n})$	$-(2^n-1) \sim +(2^n-1)$
补码	$-1 \sim +(1-2^{-n})$	$-2^n \sim +(2^n-1)$
反码	$-(1-2^{-n}) \sim +(1-2^{-n})$	$-(2^n-1) \sim +(2^n-1)$

1. 浮点数的表示形式

$N = S \times r^{j}$ 浮点数的一般形式

S 尾数 j 阶码 r 基数（基值）



S_f 代表浮点数的符号

n 其位数反映浮点数的精度

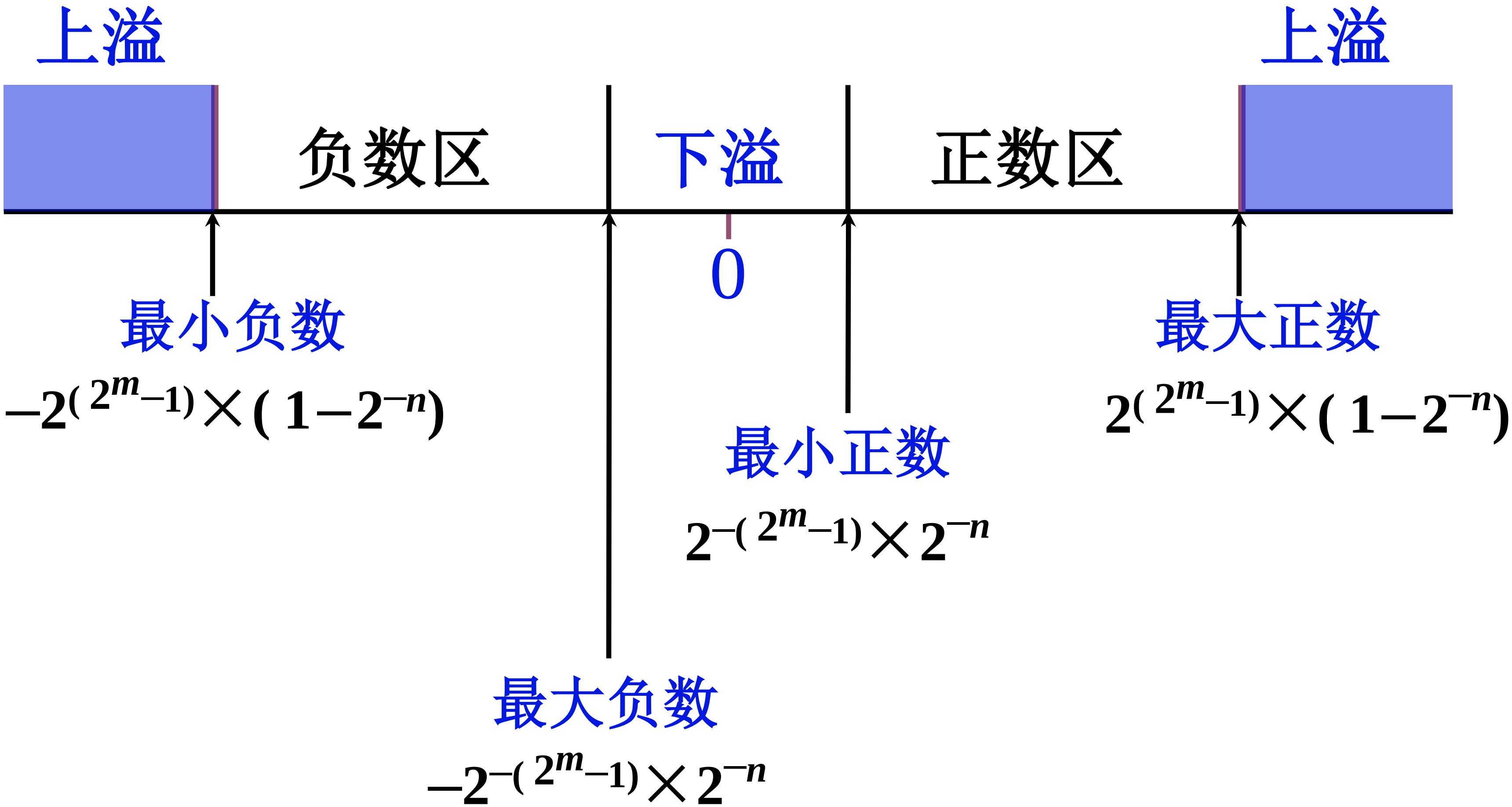
m 其位数反映浮点数的表示范围

j_f 和 m 共同表示小数点的实际位置

2. 浮点数的表示范围 (以原码为例)

上溢 阶码 > 最大阶码 超表示范围 结果出错

下溢 阶码 < 最小阶码 按 机器零 处理



3. 浮点数的规格化形式

$r = 2$ 尾数最高位为 1（原码）

$r = 4$ 尾数最高 2 位不全为 0

$r = 8$ 尾数最高 3 位不全为 0

基数不同，浮点数的规格化形式不同

最常见的情况 $r = 2$

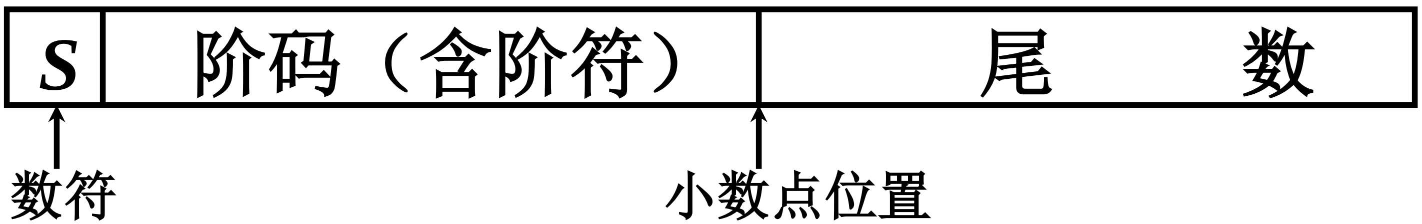
4. 浮点数的规格化

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

基数 r 越大，可表示的浮点数的范围越大

基数 r 越大，浮点数的精度降低

四、IEEE 754 标准 6.2



	符号位 *S*	阶码	尾数	总位数
短实数	1	8	23	32
长实数	1	11	52	64
临时实数	1	15	64	80

IEEE 754 Floating Point Standard

规格化数：+/-1.xxxxxxxxxx$_{two}$ x 2^{Exponent}

Single Precision： (Double Precision is similar)

S	Exponent	Significand
1 bit	8 bits	23 bits

非规格化数

Inf 尾数=0

NaN 尾数≠0

全0和全1用来表示特殊值！

- Sign bit: 1 表示negative ; 0表示 positive
- Exponent（阶码）：
  - •SP规格化数阶码范围为0000 0001 (-126) ~ 1111 1110 (127)
  - •bias为127 (single), 1023 (double)
- Significand（尾数）：
  - 规格化尾数最高位总是1，所以隐含表示，省1位
  - 1 + 23 bits （single），1 + 52 bits （double）

SP: $(-1)^S \times (1 + \text{Significand}) \times 2^{(\text{Exponent}-127)}$

DP: $(-1)^S \times (1 + \text{Significand}) \times 2^{(\text{Exponent}-1023)}$

赠送小题两道

1. 一个浮点数，当其尾数右移时，欲使其值不变，阶码必须__增大__。尾数右移一位，阶码 __+1__ 。_

2.设机器代码为 FCH，机器数为补码形式（采用 1 位符号），则对应的十进制真值为__−4__，其原码形式为__84H__，反码形式为__FBH__。

4.3 定 点 运 算

一、移位运算

1. 移位的意义

数值相对于小数点 左/右移 n 位

（ 小数点不动 ）

← 左移 绝对值扩大

→ 右移 绝对值缩小

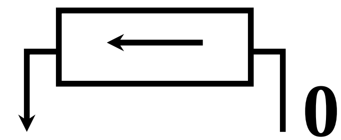
在计算机中，移位与加减配合，能够实现乘除运算

4. 算术移位和逻辑移位的区别

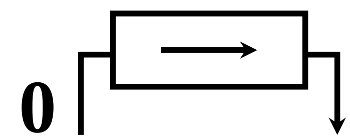
算术移位　　有符号数的移位

逻辑移位　　无符号数的移位

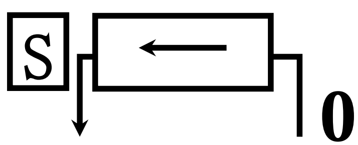
逻辑左移　　低位添 0，高位移丢



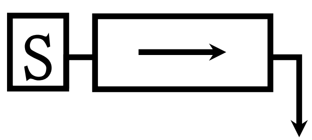
逻辑右移　　高位添 0，低位移丢



算数左移　　低位添 0，高位移丢



算数右移　　高位添 S，低位移丢



（补码）　　S 代表符号位

二、加减法运算

4.3

1. 补码加减运算公式

(1) 加法　（和的补码等于补码的和）

整数 $[A]_{补} + [B]_{补} = [A+B]_{补}$（mod 2^{n+1}）

小数 $[A]_{补} + [B]_{补} = [A+B]_{补}$（mod 2）

(2) 减法　（作差即与相反数求和）

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

整数 $[A-B]_{补} = [A+(-B)]_{补} = [A]_{补} + [-B]_{补}$ (mod 2^{n+1})

小数 $[A-B]_{补} = [A+(-B)]_{补} = [A]_{补} + [-B]_{补}$ (mod 2)

连同符号位一起相加，符号位产生的进位自然丢掉

例 6.20 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：$A = 15 = 0001111$

$B = 24 = 0011000$

$[A]_{补} = 0, 0001111$ $[B]_{补} = 0, 0011000$

$+ [-B]_{补} = 1, 1101000$

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

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

练习 1 设 $x = \dfrac{9}{16}$ $y = \dfrac{11}{16}$，用补码求 $x+y$

$x + y = -0.1100 = -\dfrac{12}{16}$ 错

练习 2 设机器数字长为 8 位（含 1 位符号位）且 $A = -97$，$B = +41$，用补码求 $A - B$

$A - B = +1110110 = +118$ 错

3. 溢出判断

(1) 一位符号位判溢出

参加操作的 两个数（减法时即为被减数和“求补”以后的减数）符号相同，其结果的符号与原操作数的符号不同，即为溢出

硬件实现

最高有效位的进位 $\oplus$ 符号位的进位 = 1　　溢出

如

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

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

(2) 两位符号位判溢出 4.3

$$[x]_{补'} = \begin{cases} x & 1 > x \geq 0 \\ 4 + x & 0 > x \geq -1 \pmod 4 \end{cases}$$

$$[x]_{补'} + [y]_{补'} = [\, x + y \,]_{补'} \pmod 4$$

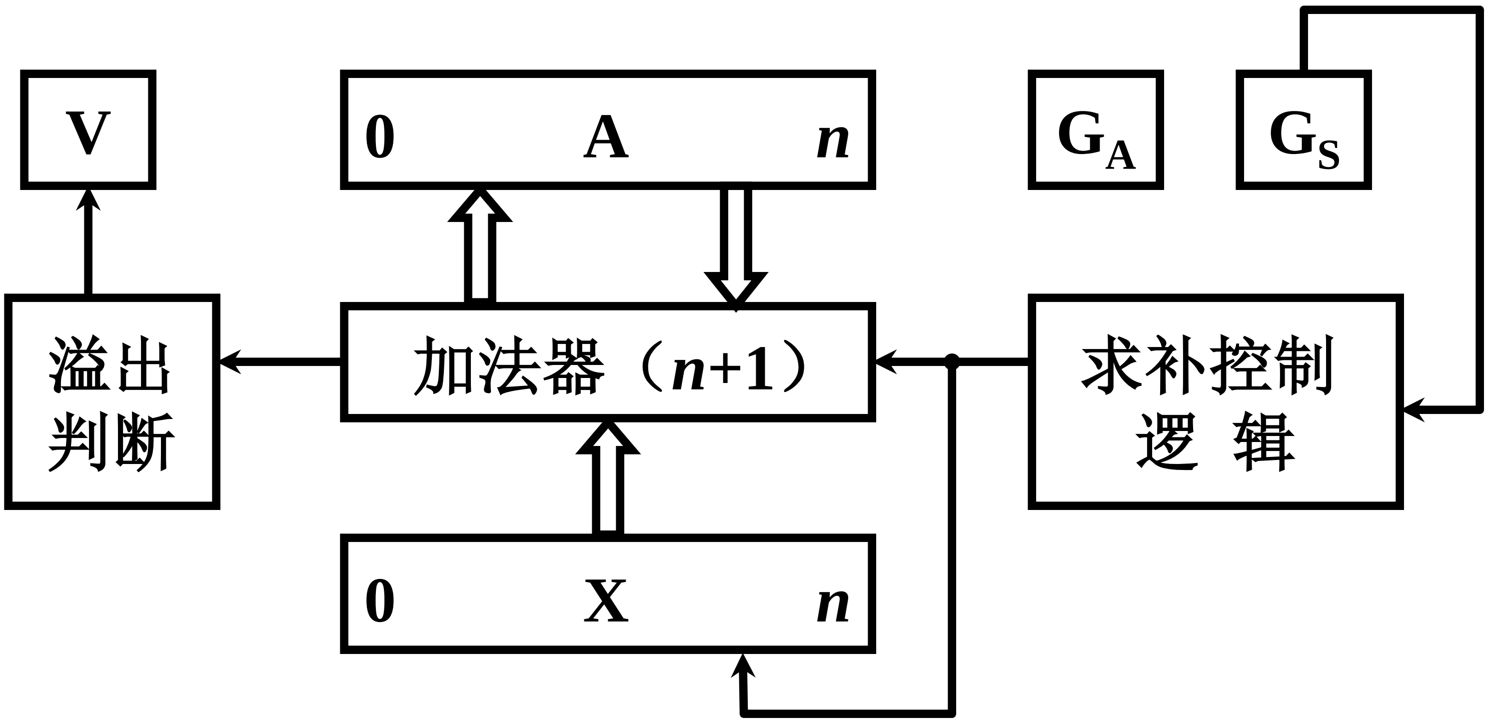
$$[x - y]_{补'} = [x]_{补'} + [-\,y]_{补'} \pmod 4$$

结果的双符号位 相同　　未溢出　　00, ×××××
11, ×××××

结果的双符号位 不同　　溢出　　10, ×××××
01, ×××××

最高符号位 代表其 真正的符号

4. 补码加减法的硬件配置



A、X 均 n+1 位

用减法标记 G_S 控制求补逻辑

三、乘法运算

1. 分析笔算乘法

$A = -0.1101 \qquad B = 0.1011$

$A \times B = -0.10001111$ 乘积的符号心算求得

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

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

2. 笔算乘法改进

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

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

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

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

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

右移一位

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

第一步 被乘数A + 0 ①

第二步 右移 一 位，得新的部分积 ②

第三步 部分积 + 被乘数 ③

⋮

第八步 右移 一 位，得结果 ⑧

3. 改进后的笔算乘法过程（竖式） 4.3

部分积	乘数	说明
0 . 0 0 0 0 + 0 . 1 1 0 1	1 0 1 1	初态，部分积 = 0 乘数为 1，加被乘数
0 . 1 1 0 1 0 . 0 1 1 0 + 0 . 1 1 0 1	 1 1 0 1	 →1，形成新的部分积 乘数为 1，加被乘数
1 . 0 0 1 1 0 . 1 0 0 1 + 0 . 0 0 0 0	1 1 1 1 0	 → 1，形成新的部分积 乘数为 0，加 0
0 . 1 0 0 1 0 . 0 1 0 0 + 0 . 1 1 0 1	1 1 1 1 1 1	 → 1，形成新的部分积 乘数为 1，加 被乘数
1 . 0 0 0 1 0 . 1 0 0 0	1 1 1 1 1 1 1	 →1，得结果

3. 改进后的笔算乘法过程（竖式） 4.3

部分积	乘数	说明
0.0000	1011	初态，部分积 = 0
+ 0.1101		乘数为 1，加被乘数
0.1101		
0.0110	1 101	→1，形成新的部分积
+ 0.1101		乘数为 1，加被乘数
1.0011	1	
0.1001	11 10	→ 1，形成新的部分积
+ 0.0000		乘数为 0，加 0
0.1001	11	
0.0100	111 1	→ 1，形成新的部分积
+ 0.1101		乘数为 1，加 被乘数
1.0001	111	
0.1000	1111	→1，得结果

小结

- 乘法 运算可用 加和移位实现

  $n = 4$，加 4 次，移 4 次

- 由乘数的末位决定被乘数是否与原部分积相加，然后 ⟶1 位形成新的部分积，同时 乘数 ⟶ 1 位（末位移丢），空出高位存放部分积的低位。

- 被乘数只与部分积的高位相加

硬件　　3 个寄存器，具有移位功能

　　　　1 个全加器

4. 原码乘法

(1) 原码一位乘运算规则

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 $x^* \cdot y^*$

(2) 原码一位乘递推公式

$$x^* \cdot y^* = x^*(0.y_1y_2 \ \cdots \ y_n)$$

$$= x^*(y_1 2^{-1} + y_2 2^{-2} + \ \cdots \ + y_n 2^{-n})$$

$$= \underbrace{2^{-1}(y_1x^* + 2^{-1}(y_2x^* + \ \cdots \underbrace{2^{-1}(y_nx^* + \underbrace{0}_{z_0})}_{z_1} \cdots))}_{z_n}$$

$$z_0 = 0$$

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

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

$$\vdots$$

$$z_n = 2^{-1}(y_1x^* + z_{n-1})$$

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

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

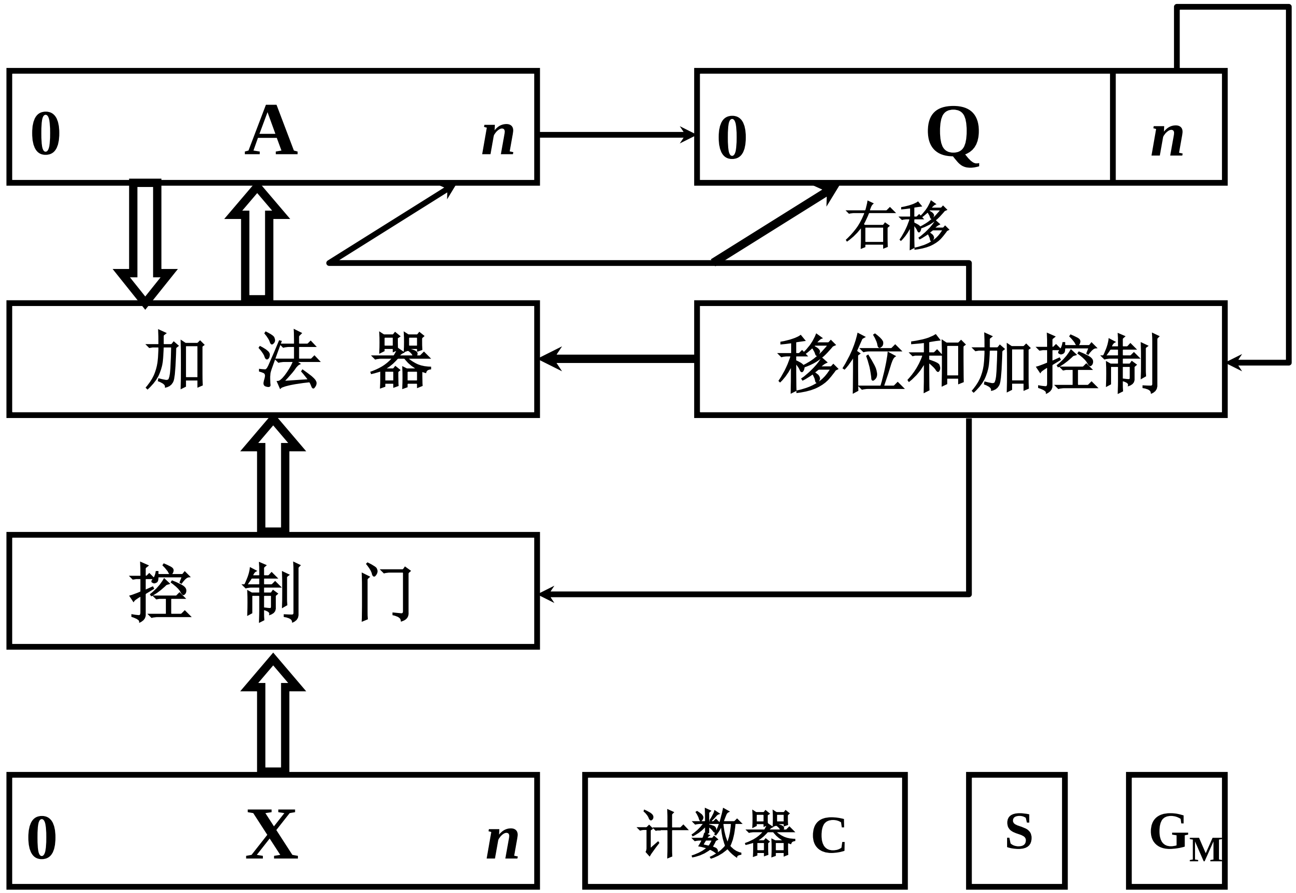
则 $[x \cdot y]_{原} = 1.\,1\,0\,1\,1\,0\,1\,1\,0$

特点 绝对值运算

用移位的次数判断乘法是否结束

逻辑移位

(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

③ Booth 算法（被乘数、乘数符号任意） 4.3

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

$[x \cdot y]_补$

$-[x]_补 = +[-x]_补$

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

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

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

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

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

附加位 y_{n+1}

$= [x]_补[(y_1-y_0)+\underbrace{(y_2-y_1)}_{y'_1}2^{-1}+ \cdots +\underbrace{(y_{n+1}-y_n)}_{y'_n}2^{-n}]$

$y'_1 2^{-1} + \cdots + y'_n 2^{-n}$

④ Booth 算法递推公式 4.3

$[z_0]_补 = 0$

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

$\vdots$

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

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

如何实现 $y_{i+1} - y_i$ ？

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

例4.23 已知 $x = +0.0011$ $y = -0.1011$ 求$[x \cdot y]_补$ 4.3

解：

	部分积	乘数		
	00.0000	1.0101	0	
	+11.1101			$+[-x]_补$
	11.1101			
补码右移	11.1110	1 1010	1	$\rightarrow 1$
	+00.0011			$+[x]_补$
	00.0001	1		
补码右移	00.0000	11 101	0	$\rightarrow 1$
	+11.1101			$+[-x]_补$
	11.1101	11		
补码右移	11.1110	111 10	1	$\rightarrow 1$
	+00.0011			$+[x]_补$
	00.0001	111		
补码右移	00.0000	1111 1	0	$\rightarrow 1$
	+11.1101			$+[-x]_补$
	11.1101	1111		最后一步不移位

$[x]_补 = 0.0011$

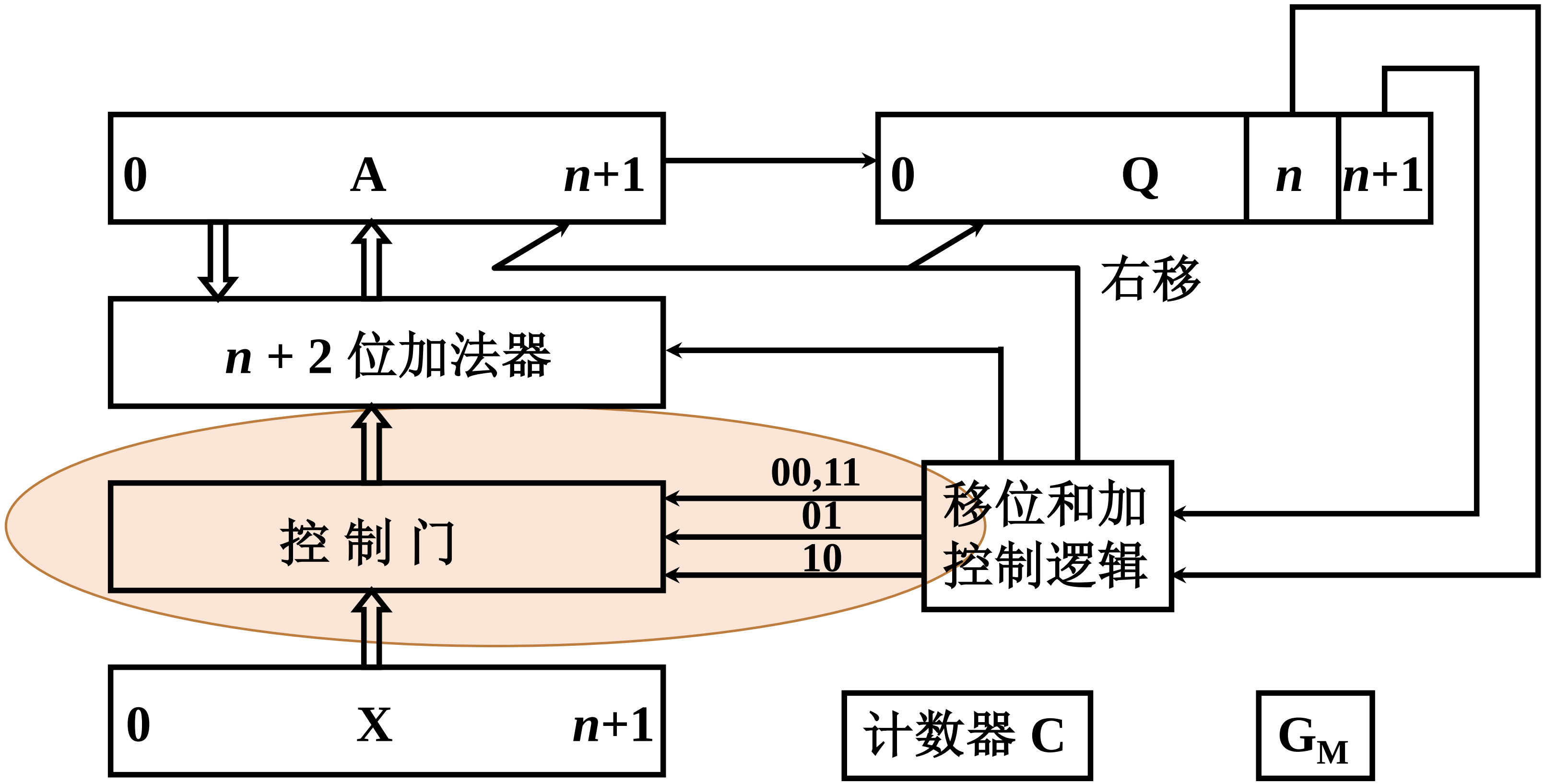
$[y]_补 = 1.0101$

$[-x]_补 = 1.1101$

$\therefore [x \cdot y]_补$

$= 1.11011111$

(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：

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

乘数中的每两位对应基-2 Booth编码中的一位。

		y_0	y_1			……				y_8	y_9
y_i	二进制数据	0	1	1	1	1	0	0	1	1	0
y_i'	Booth编码后的数据	1	0	0	0	−1	0	1	0	−1	0

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

乘法小结

- 整数乘法与小数乘法完全相同
  可用 逗号 代替小数点

- 原码乘　符号位 单独处理
  补码乘　符号位 自然形成

- 原码乘去掉符号位运算　即为无符号数乘法

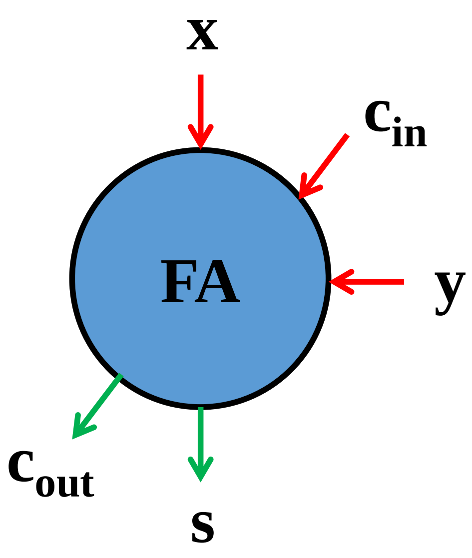
- 不同的乘法运算需有不同的硬件支持

● 快速乘法器

- 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：

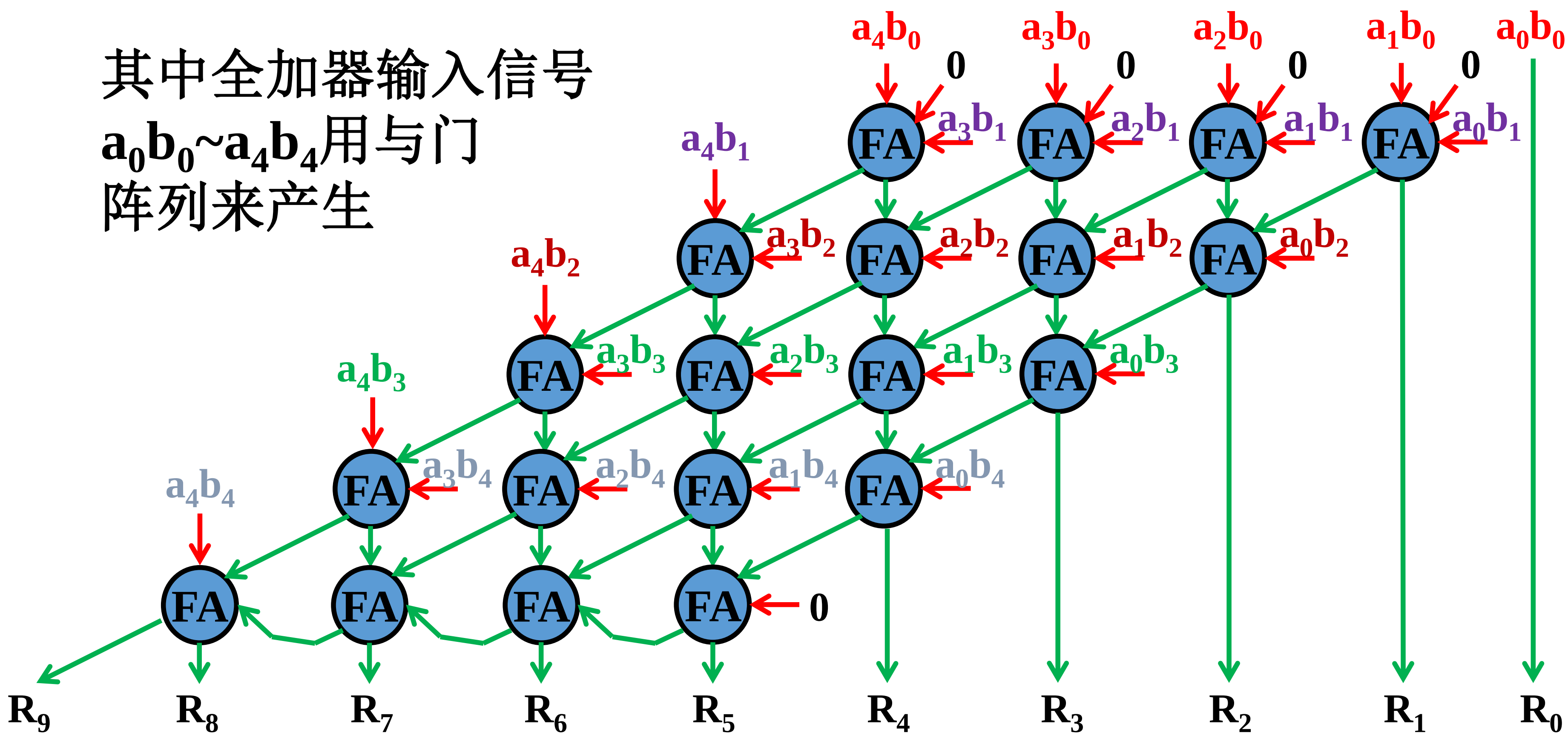
- **被乘数：$a_4a_3a_2a_1a_0$　乘数：$b_4b_3b_2b_1b_0$　结果：R=a*b**

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

● 快速乘法器

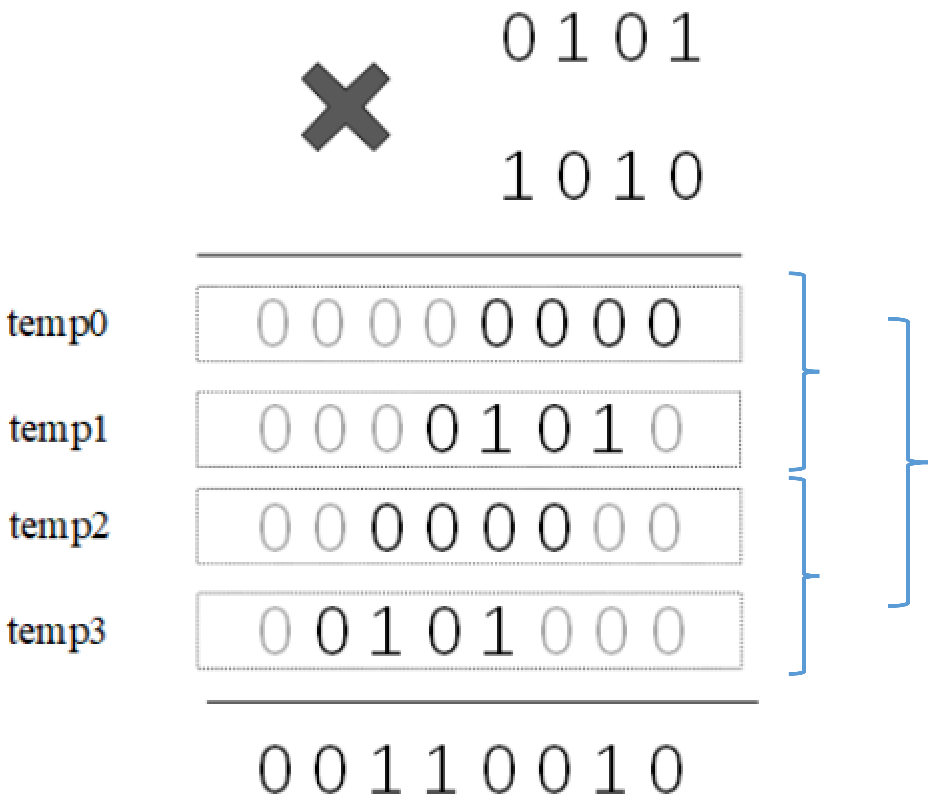
- 阵列乘法器
- 举个例子：构造一个5位*5位的阵列乘法器：
  - 被乘数：$a_4a_3a_2a_1a_0$　乘数：$b_4b_3b_2b_1b_0$　结果：R=a*b

其中全加器输入信号a_0b_0~a_4b_4用与门阵列来产生



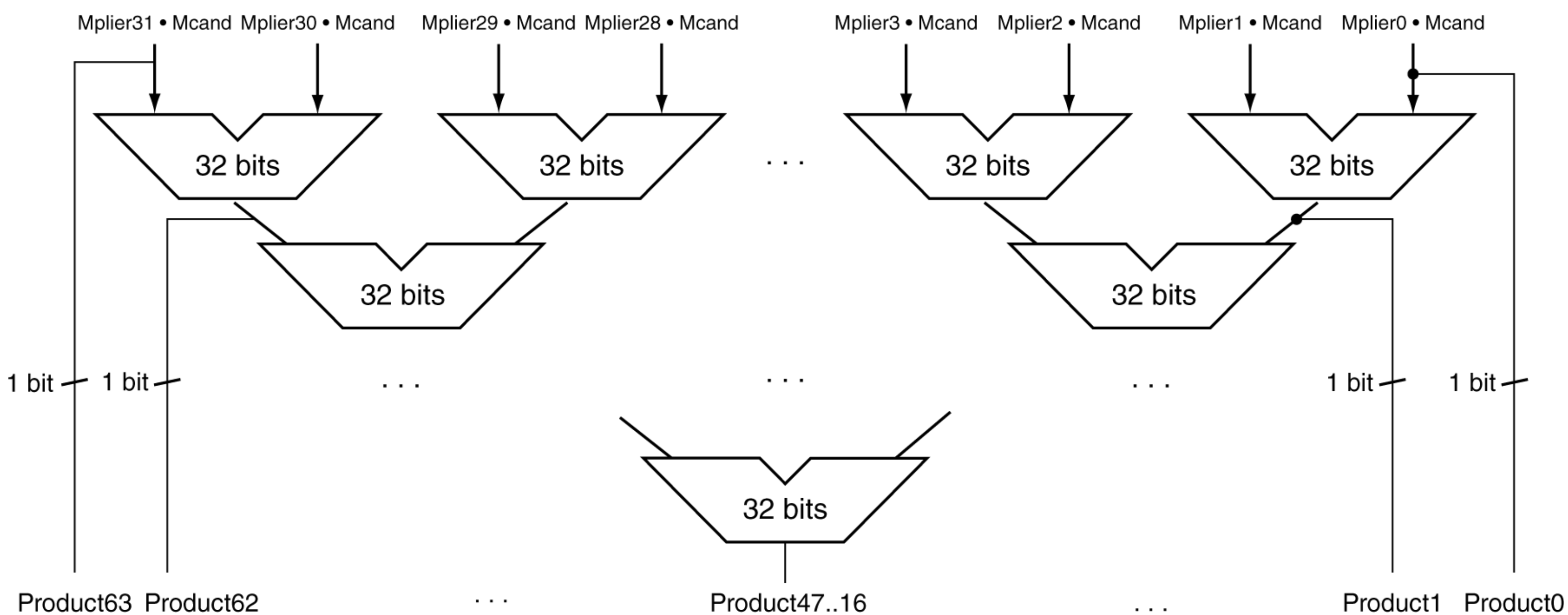
● 快速乘法器

➢快速的乘法运算主要思想：为乘数的每一位提供一个**n位**的加法器；一个用来输入被乘数和一乘数位相与的结果；一个是上一个加法器的输出。



● 快速乘法器

➢快速的乘法运算主要思想：为乘数的每一位提供一个**n位**的加法器；一个用来输入被乘数和一乘数位相与的结果；一个是上一个加法器的输出。



- 方法：将**31**个加法器组织成一个**并行树**
- 优点：易于应用**流水线**设计执行，可以同步支持多个乘法。

乘法小结

6.3

- 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