计算机组成原理

第四讲

张展

哈尔滨工业大学计算学部

容错与移动计算研究中心

第 6 章 计算机的运算方法

二、加减法运算 6.3

1. 补码加减运算公式

(1) 加法

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

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

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

2. 举例

例 6.18 设 $A = 0.1011$，$B = -0.0101$

求 $[A + B]_{补}$

解：

$$
\begin{array}{rcl}
[A]_{补} & = & 0.1011 \\
+[B]_{补} & = & 1.1011 \\
\hline
[A]_{补} + [B]_{补} & = & \boxed{1}\,0.0110 = [A + B]_{补}
\end{array}
$$

$\therefore A + B = 0.0110$

验证

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

例 6.19 设 $A = -9$，$B = -5$

求 $[A+B]_{补}$

解：

$$
\begin{array}{rcl}
[A]_{补} & = & 1,0111 \\
+[B]_{补} & = & 1,1011 \\
\hline
[A]_{补} + [B]_{补} & = & \boxed{1}\,1,0010 = [A + B]_{补}
\end{array}
$$

$\therefore A + B = -1110$

验证

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

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： $A = 15 = 0001111$

$B = 24 = 0011000$

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

$$+\ [-B]_{补} = 1,1101000$$

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

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

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

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

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

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

(2) 两位符号位判溢出 6.3

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

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

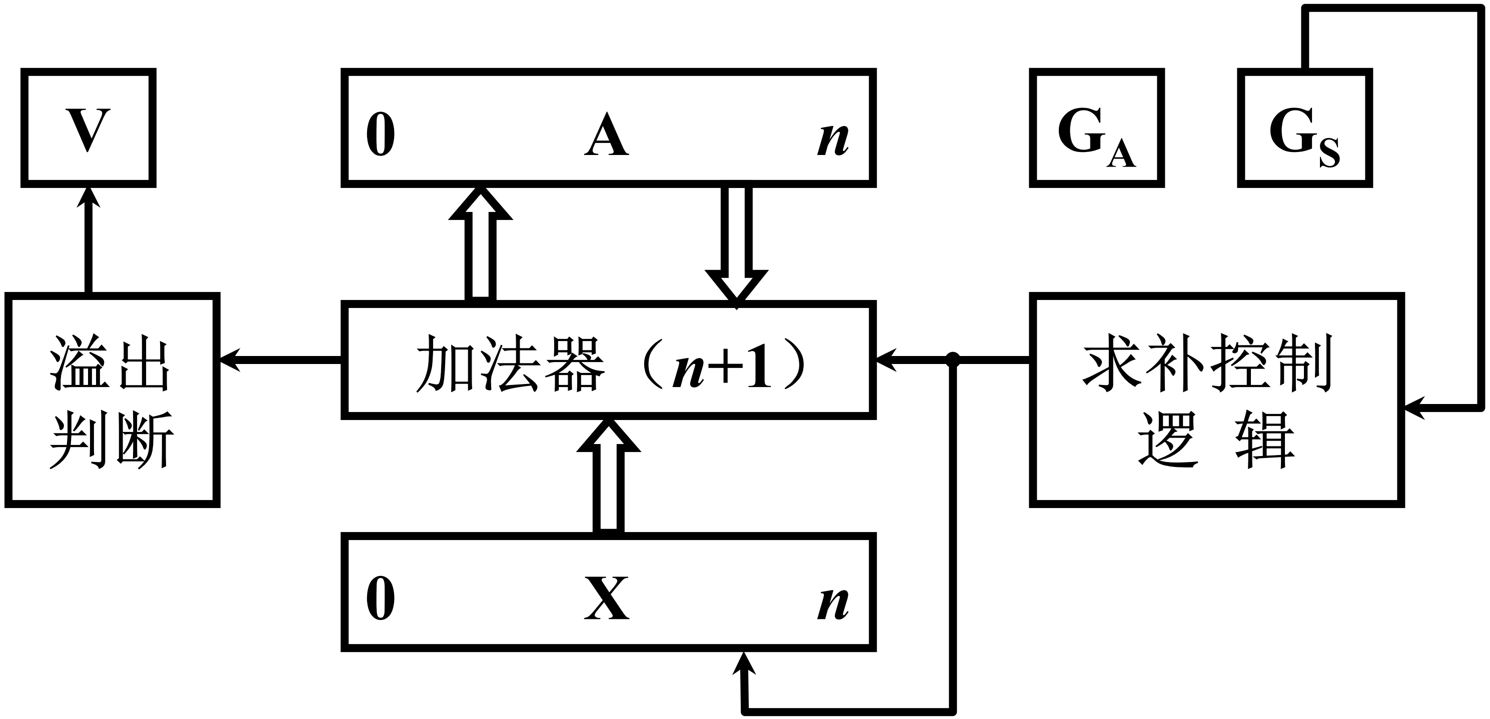
$$[x - y]_{补'} = [x]_{补'} + [-y]_{补'} \pmod 4$$

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 6.3

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

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

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

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

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

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） 6.3

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

小结 6.3

- 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6.21 已知 $x = -0.1110$ $y = 0.1101$ 求 $[x \cdot y]_{原}$ 6.3

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6.21 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

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

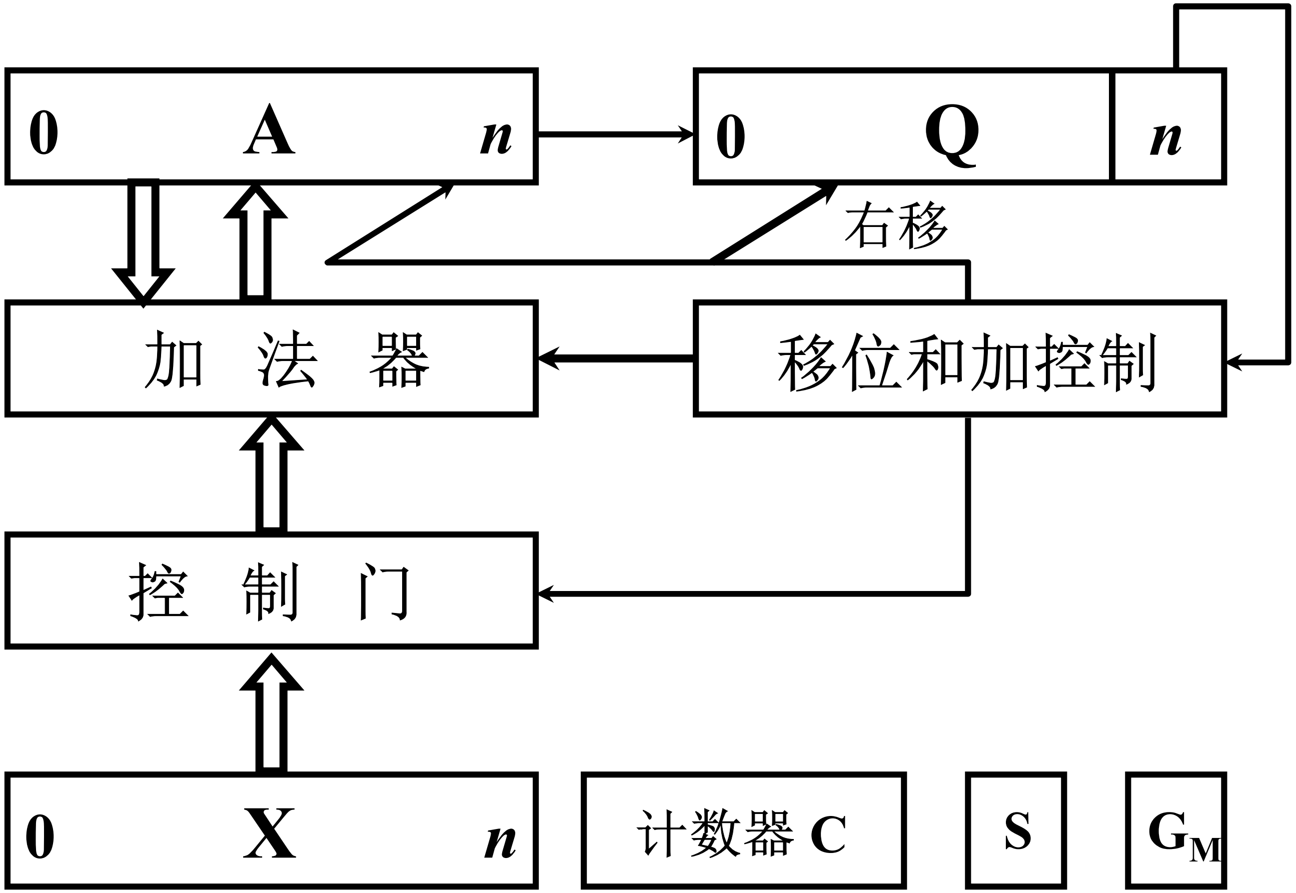
则 $[x \cdot y]_{原} = 1.1\,0\,1\,1\,0\,1\,1\,0$

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 $[x]_{补} = x_0 . x_1 x_2 \cdots x_n$

乘数 $[y]_{补} = y_0 . y_1 y_2 \cdots y_n$

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） 6.3

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

$[x \cdot y]_{补}$

$-[x]_{补} = +[-x]_{补}$

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

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

$2^{-1} = 2^0 - 2^{-1}$

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

$2^{-2} = 2^{-1} - 2^{-2}$

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

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

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

附加位 y_{n+1}

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

④ Booth 算法递推公式 6.3

$[z_0]_补 = 0$

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

$\vdots$

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

$[x \cdot y]_补 = [z_n]_补 + (y_1 - y_0)[x]_补$ 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6.23 已知 $x = +0.0011$ $y = -0.1011$ 求 $[x \cdot y]_{补}$

$[x]_{补} = 0.0011$

$[y]_{补} = 1.0101$

$[-x]_{补} = 1.1101$

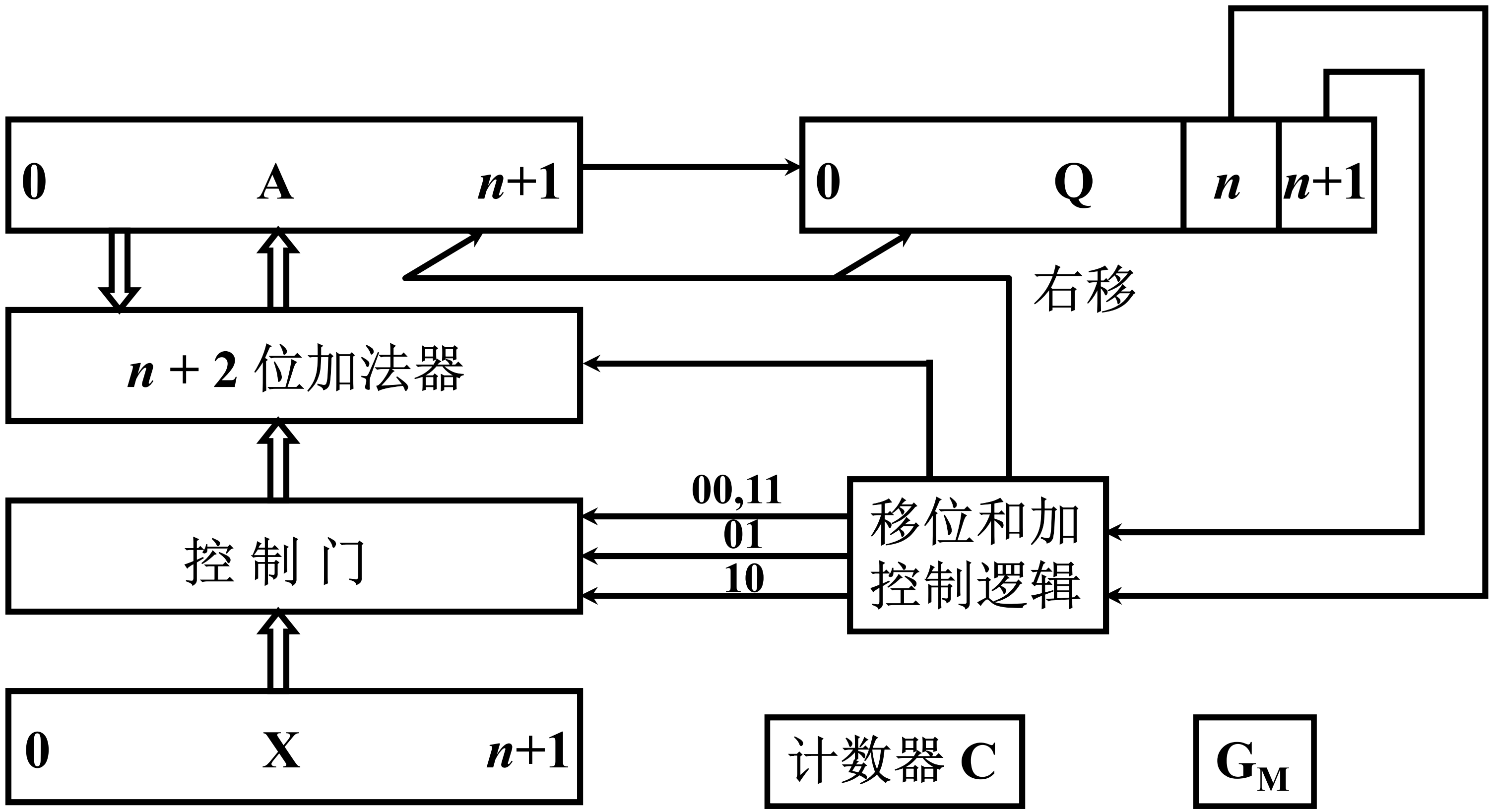
解：

	部分积	乘数	附加位	说明
	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

$\therefore [x \cdot y]_{补}$

$= 1.11011111$

(2) Booth 算法的硬件配置 6.3



A、X、Q 均 $n+2$ 位

移位和加法操作受乘数末两位控制

累加器乘法小结

- 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

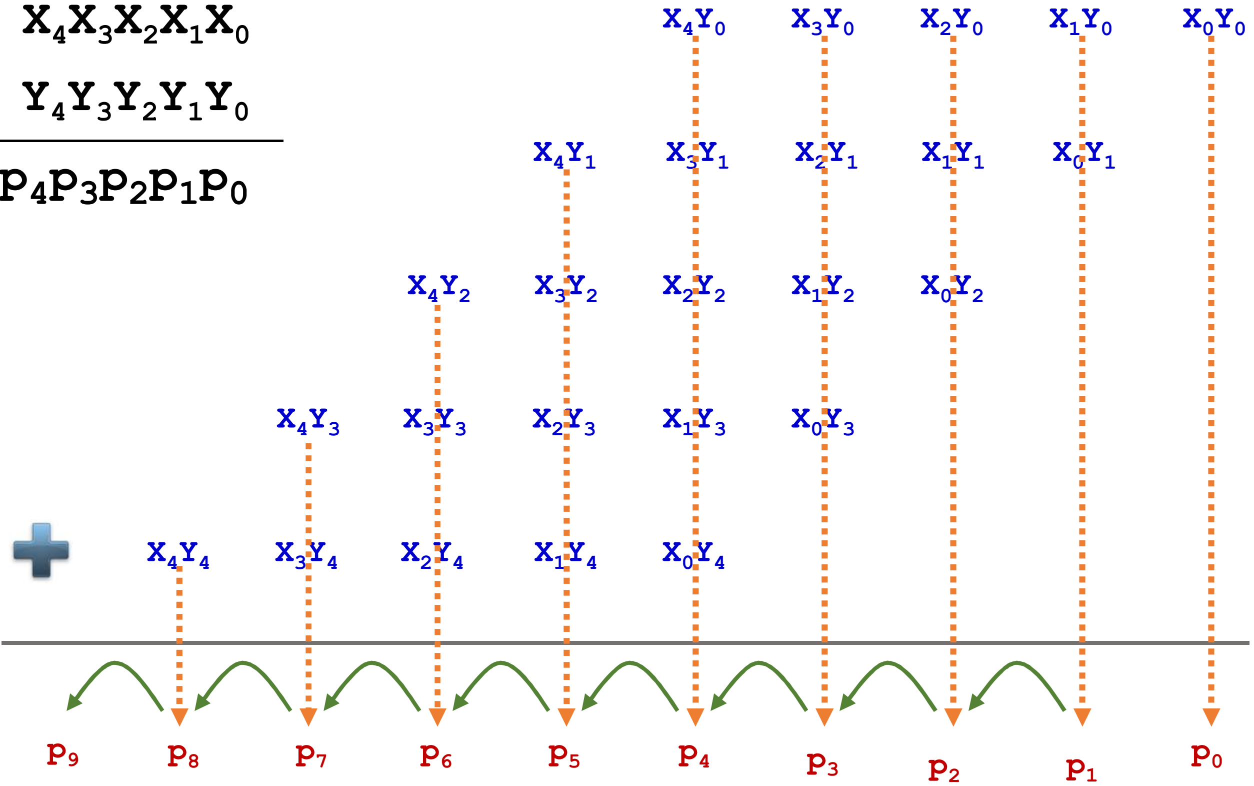
- 不同的乘法运算需有不同的硬件支持

乘法运算实现方法

- 执行乘法运算子程序实现乘法运算
  - 零成本、Intel 8008/8080、RISCV32-I指令集
- 利用加法器多次累加实现乘法运算
  - 原码一位乘法的运算方法与逻辑实现
  - 补码一位乘法的运算方法与逻辑实现
  - 成本低
- 设置专用乘法器实现乘法运算
  - 成本高 （原码、补码乘法器）

二进制手工乘法运算 6.3

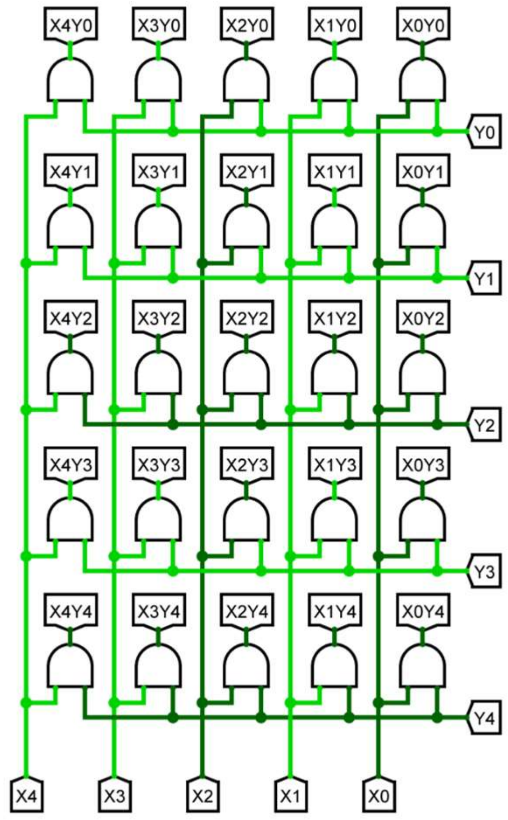
$$
\begin{array}{r}
X_4X_3X_2X_1X_0 \\
\times \quad Y_4Y_3Y_2Y_1Y_0 \\
\hline
P_9P_8P_7P_6P_5P_4P_3P_2P_1P_0
\end{array}
$$



先计算相加数，然后逐列相加

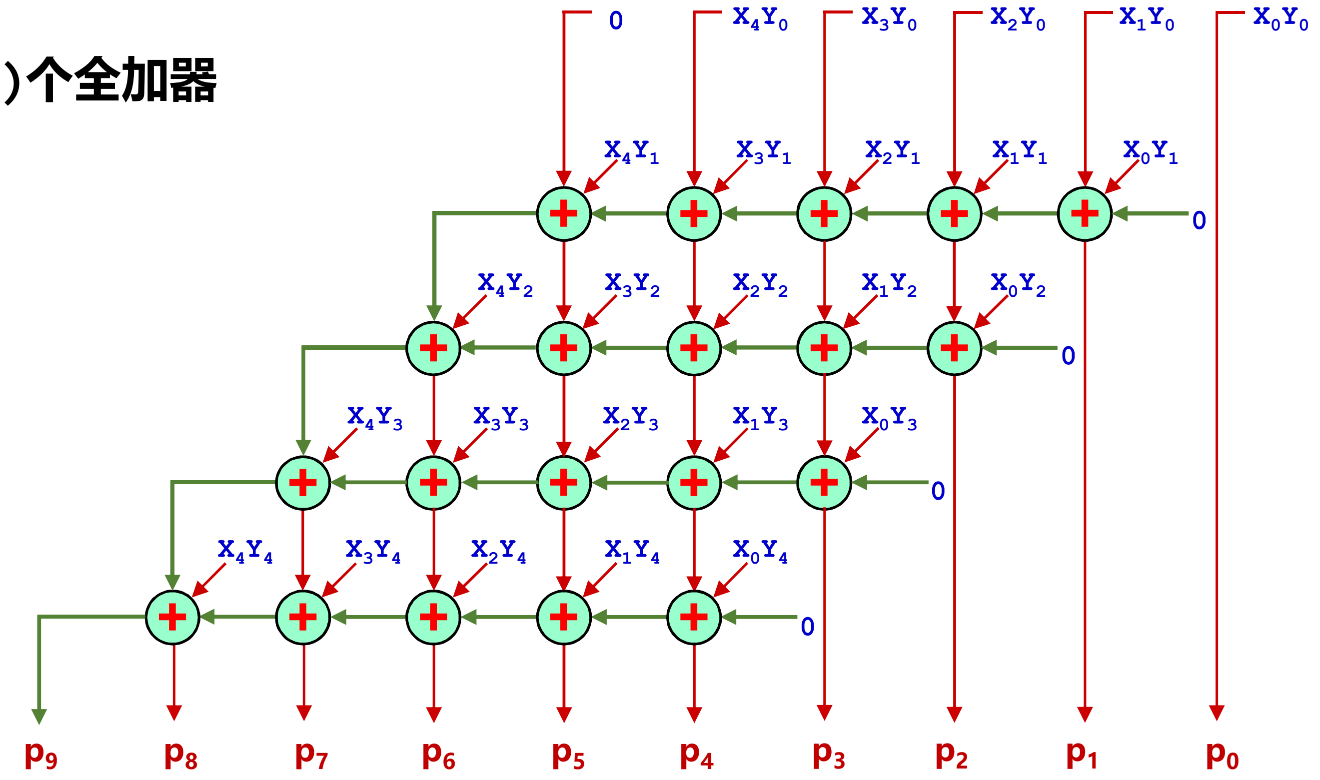
一位乘法逻辑实现

- R=X*Y
  - 1 × 1 = 1
  - 1 × 0 = 0
  - 0 × 1 = 0
  - 0 × 0 = 0
- 与门实现一位乘法
- 25个与门并发
- 一级门延迟，生成所有相加数



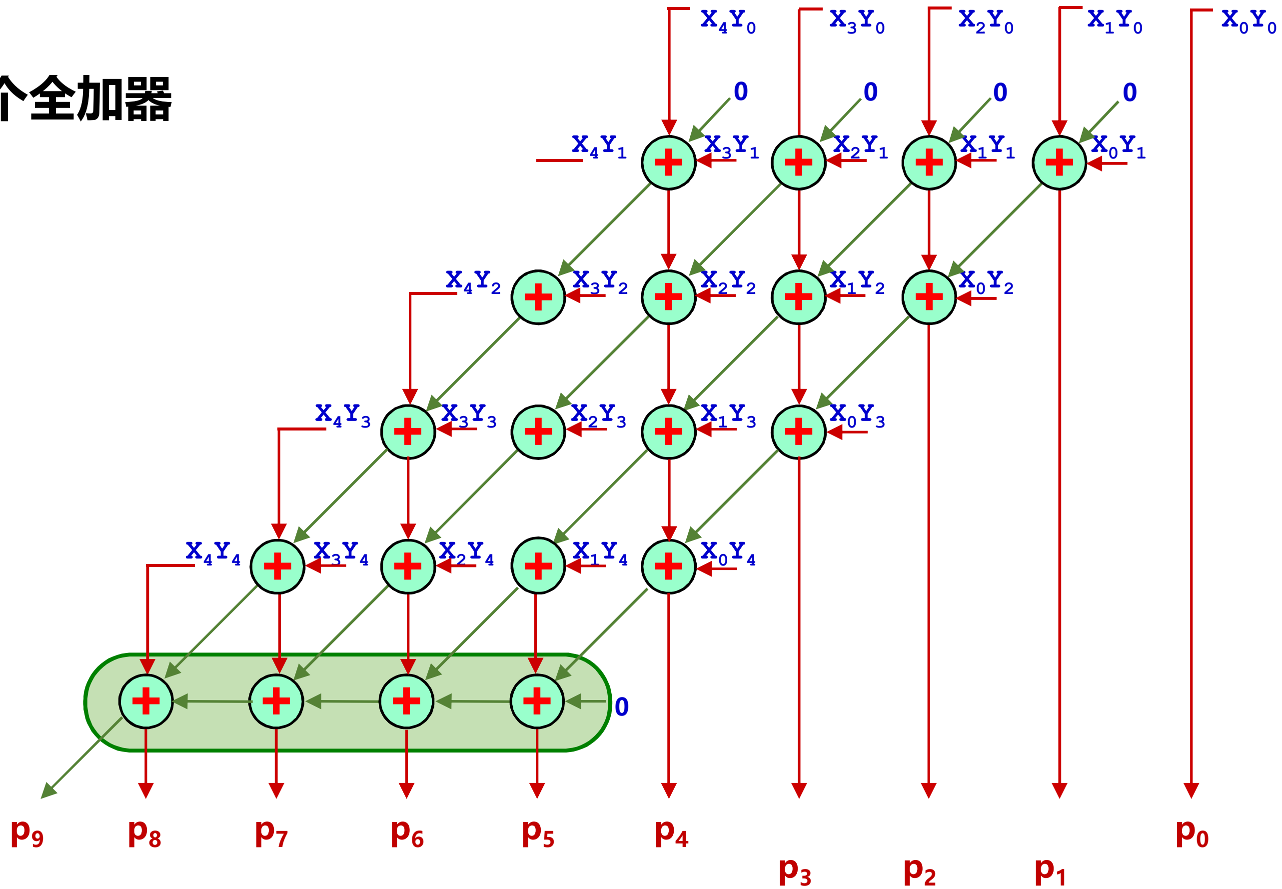
横向进位无符号阵列乘法器

- **n*(n-1)个全加器**



斜向进位无符号阵列乘法器 6.3

- n*(n-1)个全加器

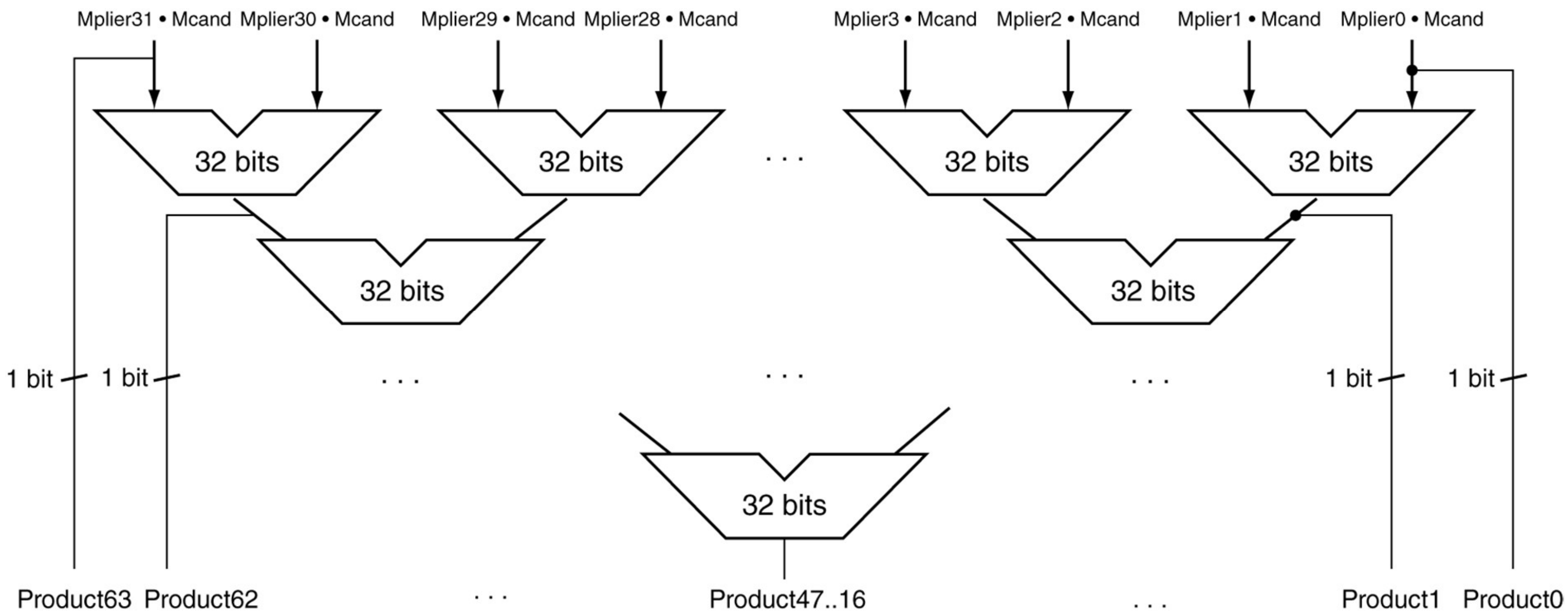


乘法器性能提升

- 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： 减少相加数，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

快速乘法器（并行树） 6.3

➢快速的乘法运算主要思想：为乘数的每一位提供一个32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。