

计算机组织结构

5 整数运算

任桐炜

2021年9月28日



南京大學
NANJING UNIVERSITY



教材对应章节



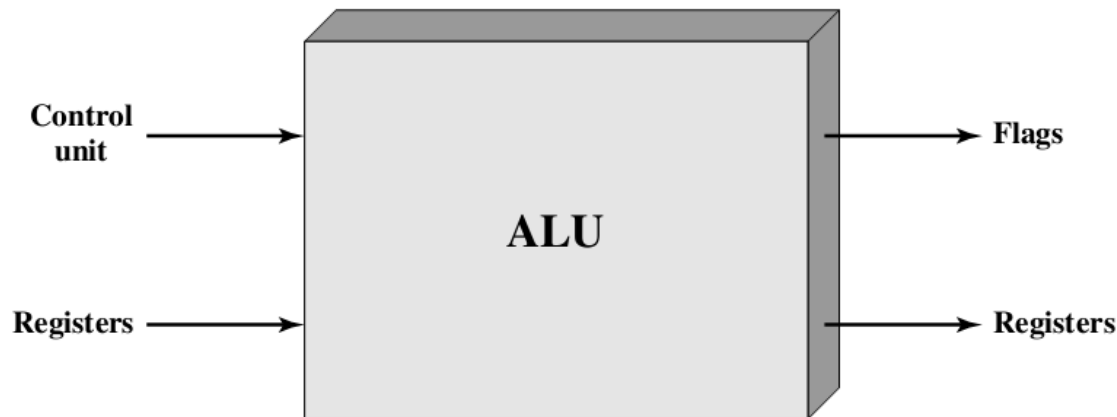
第3章 运算方法和运算部件



第9章 计算机算术

算术逻辑单元 (ALU)

- 算术逻辑单元 (ALU) 是计算机实际完成数据算术逻辑运算的部件
 - 数据由寄存器 (Registers) 提交给ALU，运算结果也存于寄存器
 - ALU可能根据运算结果设置一些标志 (Flags)，标志值也保存在处理器内的寄存器中
 - 控制器 (Control unit) 提供控制ALU操作和数据传入送出ALU的信号

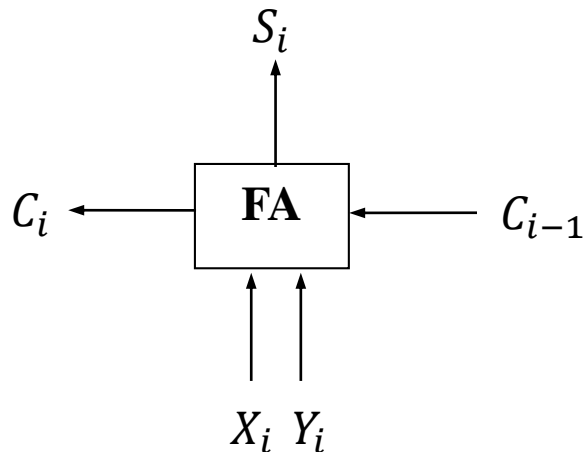


全加器

- 1位 (1bit) 加法: $X_i + Y_i$
- 第 i 位加法:

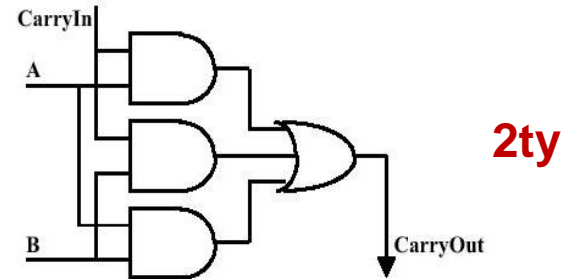
$$S_i = X_i \oplus Y_i \oplus C_{i-1}$$

$$C_i = X_i C_{i-1} + Y_i C_{i-1} + X_i Y_i$$

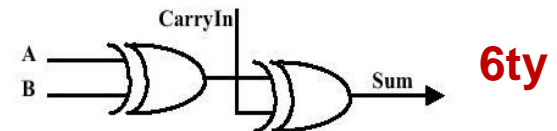


与门延迟: 1级门延迟 (1ty)
或门延迟: 1级门延迟 (1ty)
异或门延迟: 3级门延迟 (3ty)

$$\text{CarryOut} = B \& \text{CarryIn} \mid A \& \text{CarryIn} \mid A \& B$$

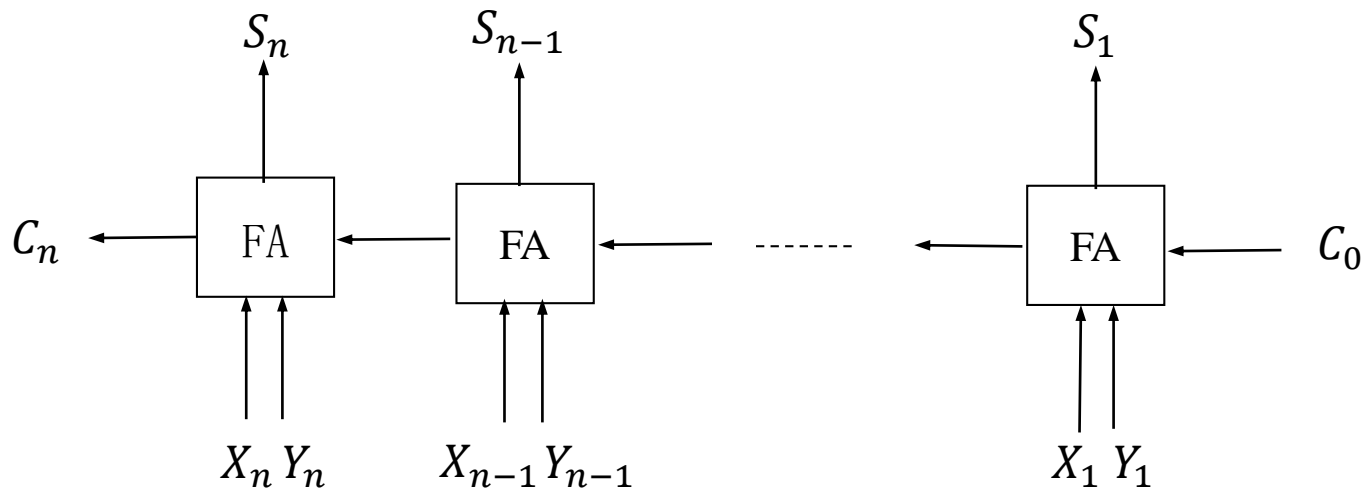


$$\text{Sum} = A \text{ XOR } B \text{ XOR } \text{CarryIn}$$



串行进位加法器

- 延迟
 - C_n : $2n$ ty
 - S_n : $(2n+1)$ ty
- 缺点：慢



全先行进位加法器

- 超前进位

$$C_i = X_i C_{i-1} + Y_i C_{i-1} + X_i Y_i$$

$$C_1 = X_1 Y_1 + (X_1 + Y_1) C_0$$

$$C_2 = X_2 Y_2 + (X_2 + Y_2) X_1 Y_1 + (X_2 + Y_2) (X_1 + Y_1) C_0$$

$$C_3 = X_3 Y_3 + (X_3 + Y_3) X_2 Y_2 + (X_3 + Y_3) (X_2 + Y_2) X_1 Y_1 \\ + (X_3 + Y_3) (X_2 + Y_2) (X_1 + Y_1) C_0$$

$$C_4 = \dots\dots$$

定义两个辅助函数: $P_i = X_i + Y_i$ $G_i = X_i Y_i$

$$C_1 = G_1 + P_1 C_0$$

$$C_2 = G_2 + P_2 G_1 + P_2 P_1 C_0$$

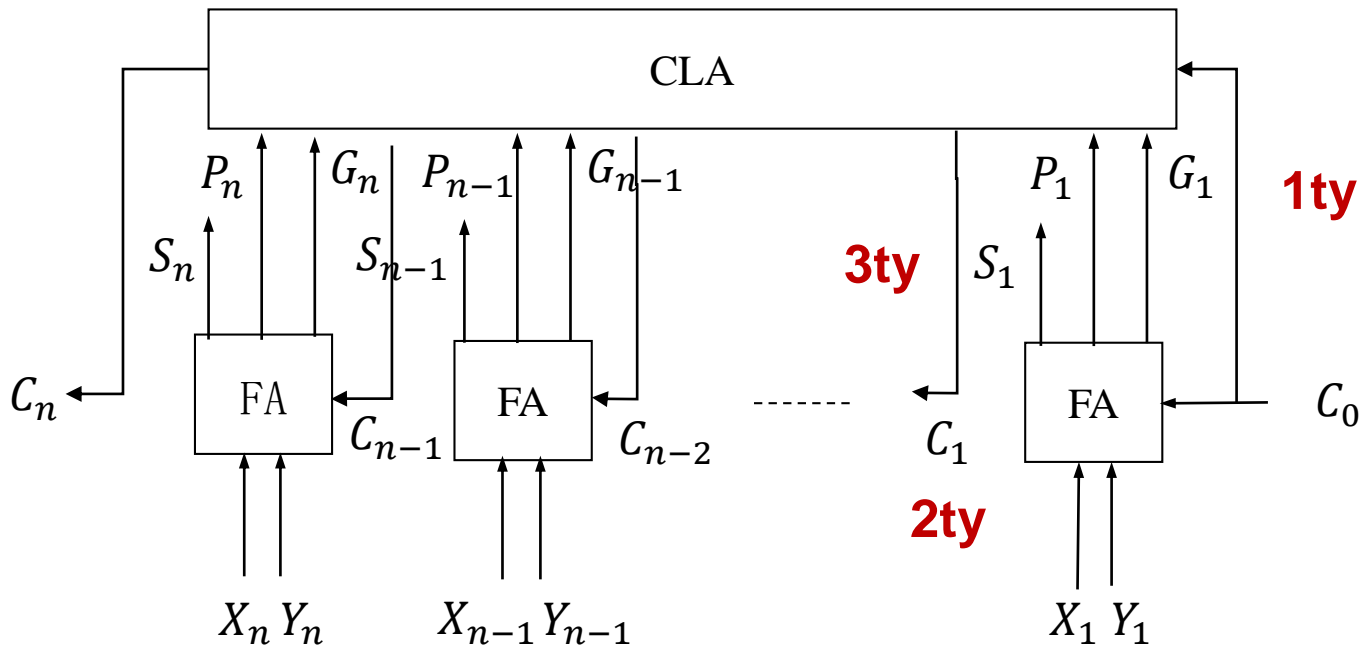
$$C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 C_0$$

$$C_4 = \dots\dots$$



全先行进位加法器 (续)

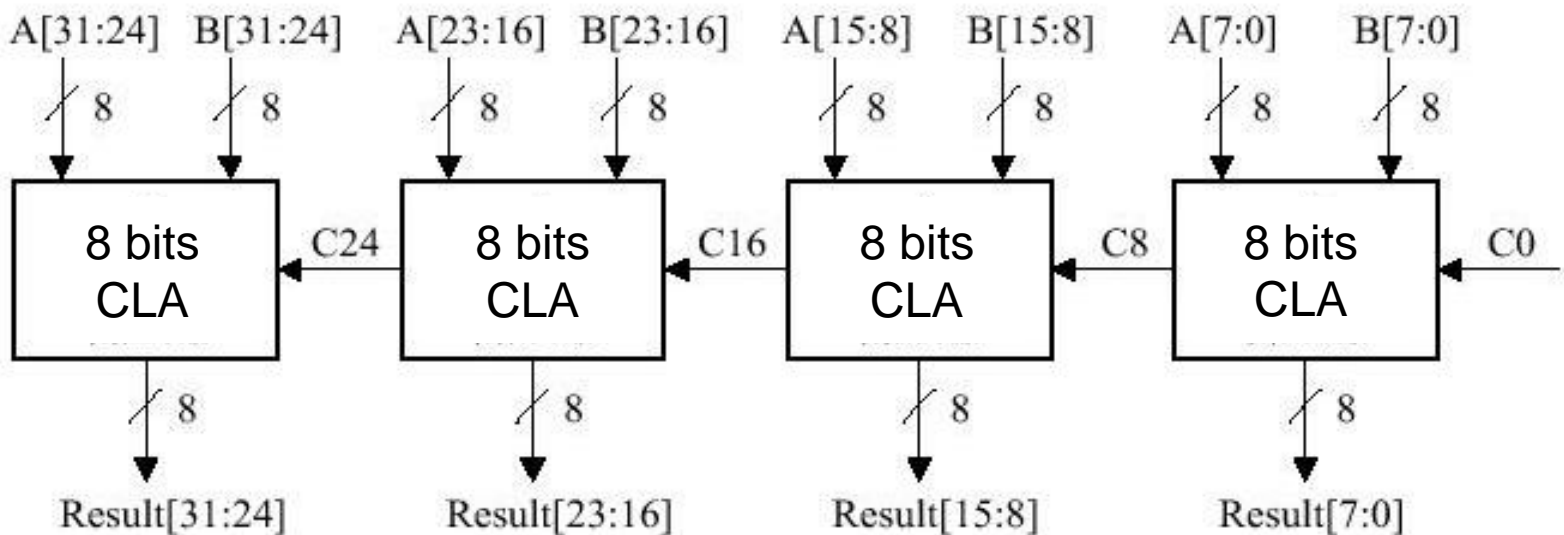
- 缺点：复杂



延迟: $1ty + 2ty + 3ty = 6ty$

部分先行进位加法器

- 思路
 - 采用多个CLA并将其串联，取得计算时间和硬件复杂度之间的权衡
- 例子



延迟: $3t_y + 2t_y + 2t_y + 5t_y = 12t_y$

加法

- $[X+Y]_C = [X]_C + [Y]_C \pmod{2^n}$
- 溢出:

$\begin{array}{r} -7 \\ + -6 \\ \hline -13 \end{array}$	$\begin{array}{r} 1001 \\ + 1010 \\ \hline 10011 \end{array}$	$\begin{array}{r} 7 \\ + 6 \\ \hline 13 \end{array}$	$\begin{array}{r} 0111 \\ + 0110 \\ \hline 1101 \end{array}$
	3		-3

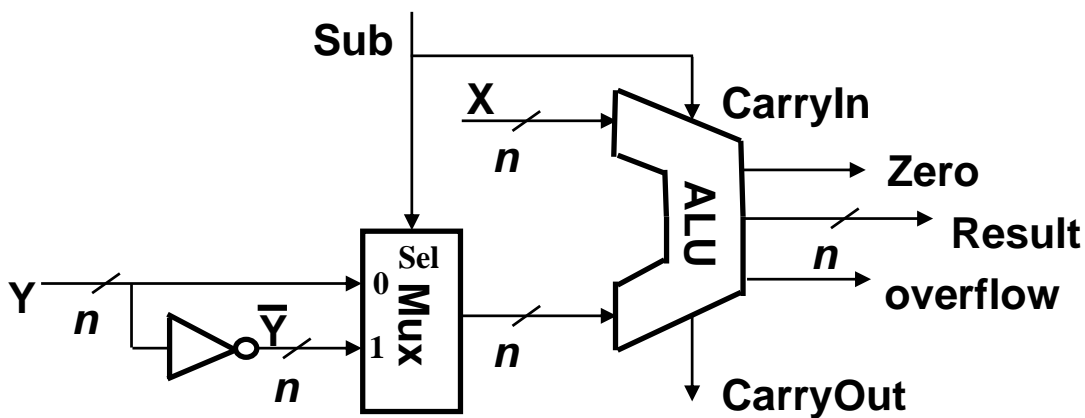
- $X_n = Y_n$ 且 $S_n \neq \overline{X_n} \oplus \overline{Y_n}$: $overflow = X_n Y_n \overline{S_n} + \overline{X_n} \overline{Y_n} S_n$
- $C_n \neq C_{n-1}$: $overflow = C_n \oplus C_{n-1}$



减法

- $[X-Y]_c = [X]_c + [-Y]_c \pmod{2^n}$
- 溢出：与加法相同

$$\begin{array}{r} -7 \\ - 6 \\ \hline -13 \end{array} \quad \begin{array}{r} 1001 \\ + 1010 \\ \hline 10011 \end{array} \quad 3$$



乘法

- 手工演算乘法

- 若 $Y_i = 0$, 部分积为 0; 否则, 部分积为 X
- 每一步的部分积相比之前左移一位
- 对所有部分积求和

	7		0111
×	6	×	0110
<hr/>		<hr/>	
	42		0000
			0111
			0111
			0000
			<hr/>
			0101010

- 计算机乘法的改进

- 每一步都计算部分积求和结果
- 右移部分积代替左移部分积
- 若 $Y_i = 0$, 只执行移位操作



乘法 (续)

- 例子

$$\begin{array}{r}
 7 \\
 \times 6 \\
 \hline
 42
 \end{array}
 \qquad
 \begin{array}{r}
 0111 \\
 \times 0110 \\
 \hline
 0000 \\
 0111 \\
 0111 \\
 0000 \\
 \hline
 0101010
 \end{array}$$

initial

0 ->

1 +

->

1 +

->

0 ->

部分积

0000

00000

01110

001110

101010

0101010

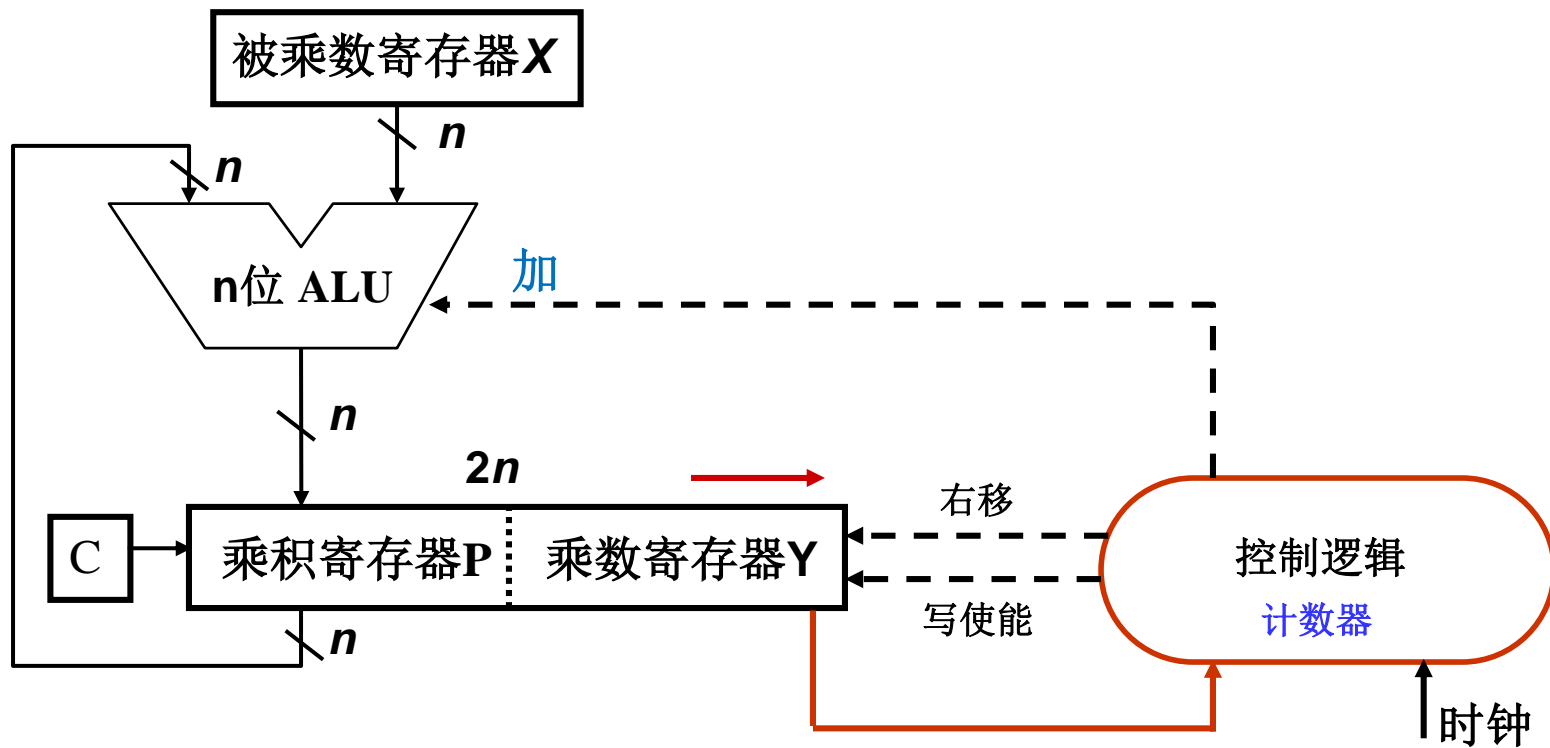
00101010

42



乘法 (续)

- 实现



乘法 (续)

- 实现 (续)

$$\begin{array}{r}
 7 \\
 \times 6 \\
 \hline
 42
 \end{array}
 \qquad
 \begin{array}{r}
 0111 \\
 \times 0110 \\
 \hline
 0000 \\
 0111 \\
 0111 \\
 0000 \\
 \hline
 0101010
 \end{array}$$

		结果	Y
初始化		0000	0110
0	->	0000	0011
1	+	0111	0011
	->	0011	1001
1	+	1010	1001
	->	0101	0100
0	->	0010	1010



乘法 (续)

- 问题: $[X \times Y]_c \neq [X]_c \times [Y]_c$

$$\begin{array}{r} 7 \\ \times 6 \\ \hline 42 \end{array} \quad \begin{array}{r} 0111 \\ \times 0110 \\ \hline 0101010 \end{array} \quad 42$$

$$\begin{array}{r} -7 \\ \times -6 \\ \hline 42 \end{array} \quad \begin{array}{r} 1001 \\ \times 1010 \\ \hline 1011010 \end{array} \quad -38$$

- 大致思路:
 - 将被乘数和乘数由补码表示改为原码表示
 - 将乘积结果由原码表示改为补码表示



乘法 (续)

- 布斯算法

$$\begin{aligned} X \times Y &= X \times Y_n Y_{n-1} \dots Y_2 Y_1 \\ &= X \times (-Y_n \times 2^{n-1} + Y_{n-1} \times 2^{n-2} + \dots + Y_2 \times 2^1 + Y_1 \times 2^0) \\ &= X \times \left(-Y_n \times 2^{n-1} + Y_{n-1} \times (2^{n-1} - 2^{n-2}) + \dots \right. \\ &\quad \left. + Y_2 \times (2^2 - 2^1) + Y_1 \times (2^1 - 2^0) \right) \\ &= X \times \left((Y_{n-1} - Y_n) \times 2^{n-1} + (Y_{n-2} - Y_{n-1}) \times 2^{n-2} + \dots \right. \\ &\quad \left. + (Y_1 - Y_2) \times 2^1 + (Y_0 - Y_1) \times 2^0 \right) \quad Y_0 = 0 \end{aligned}$$

$$= 2^n \times \sum_{i=0}^{n-1} (X \times (Y_i - Y_{i+1}) \times 2^{-(n-i)})$$



$$P_{i+1} = 2^{-1} \times (P_i + X \times (Y_i - Y_{i+1}))$$



乘法 (续)

- 布斯算法 (续)

1. 增加 $Y_0 = 0$
2. 根据 $Y_{i+1}Y_i$, 决定是否增加 $+X$, $-X$, $+0$
3. 右移部分积
4. 重复步骤 2和步骤 3共 n 次, 得到最终结果



乘法 (续)

- 布斯算法 (续)

$$\begin{array}{r} -7 \\ \times -6 \\ \hline 42 \end{array}$$

$$\begin{aligned} [X]_c &= 1001 \\ [-X]_c &= 0111 \\ [Y]_c &= 1010 \end{aligned}$$

初始化

$$\begin{aligned} Y_0 - Y_1 &= 0 & \rightarrow \\ Y_1 - Y_2 &= -1 & -X \\ & & \rightarrow \\ Y_2 - Y_3 &= 1 & +X \\ & & \rightarrow \\ Y_3 - Y_4 &= -1 & -X \\ & & \rightarrow \end{aligned}$$

结果

Y

0000	10100	
0000	01010	
0111	01010	
0011	10101	
1100	10101	
0110	01010	
1101	01010	
0110	10101	106
		?



乘法 (续)

- 布斯算法 (续)

$$\begin{array}{r} -7 \\ \times -6 \\ \hline 42 \end{array}$$

$$\begin{aligned} [X]_c &= 1001 \\ [-X]_c &= 0111 \\ [Y]_c &= 1010 \end{aligned}$$

初始化

$$Y_0 - Y_1 = 0 \quad ->$$

$$Y_1 - Y_2 = -1 \quad -X$$

->

$$Y_2 - Y_3 = 1 \quad +X$$

->

$$Y_3 - Y_4 = -1 \quad -X$$

->

结果

Y

$$0000 \quad 10100$$

$$0000 \quad 01010$$

$$0111 \quad 01010$$

$$0011 \quad 10101$$

$$1100 \quad 10101$$

$$1110 \quad 01010$$

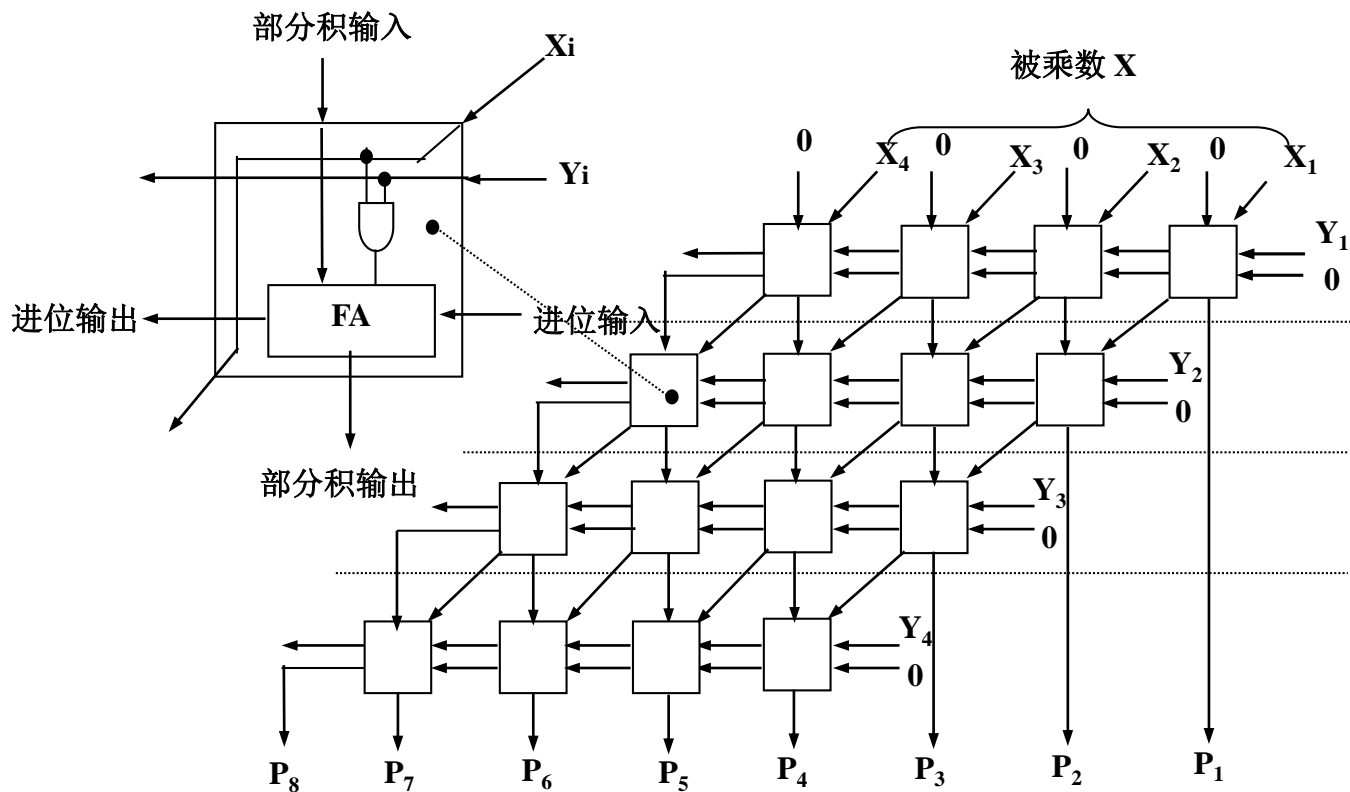
$$0101 \quad 01010$$

$$0010 \quad 10101 \quad 42$$



乘法 (续)

- 阵列乘法器



除法

- 不同情形的处理
 - 若被除数为0，除数不为0：商为0
 - 若被除数不为0，除数为0：发生“除数为0”异常
 - 若被除数、除数均为0：发生“除法错”异常
 - 若被除数、除数均不为0：进行进一步除法运算



除法 (续)

- 手工演算除法

- 在被除数的左侧补充符号位，将除数的最高位与被除数的次高位对齐
- 从被除数中减去除数，若够减，则上商为1；若不够减，则上商为0
- 右移除数，重复上述步骤

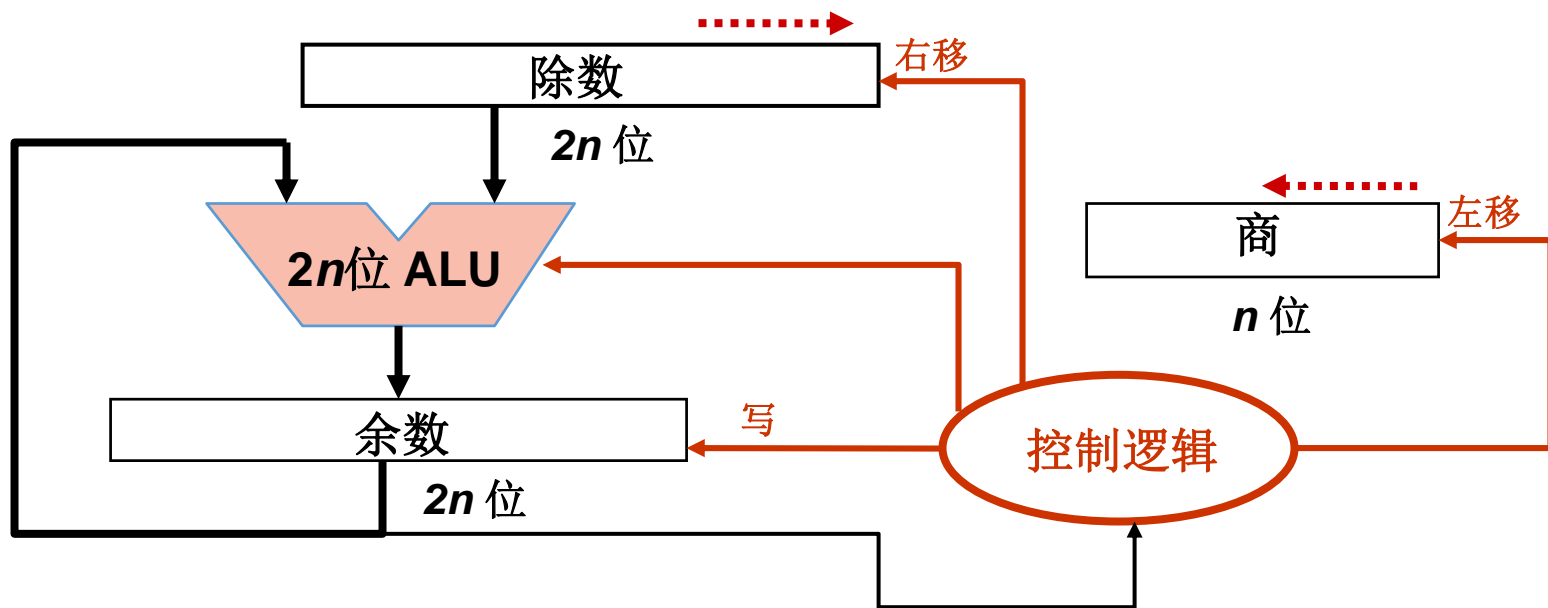
$$\begin{array}{r} 2 \\ 3 \overline{) 761} \\ \underline{6} \\ 1 \end{array}$$

$$\begin{array}{r} 0010 \\ 0011 \overline{) 00000111} \\ \underline{0000} \\ 0001 \\ \underline{0000} \\ 0011 \\ \underline{0011} \\ 0001 \\ \underline{0000} \\ 0001 \end{array}$$



除法 (续)

- 实现



除法 (续)

• 例子

$$\begin{array}{r}
 0011 \overline{) 00000111} \\
 \underline{0000} \\
 000111 \\
 \underline{0000} \\
 00111 \\
 \underline{0011} \\
 0001 \\
 \underline{0000} \\
 0001
 \end{array}$$

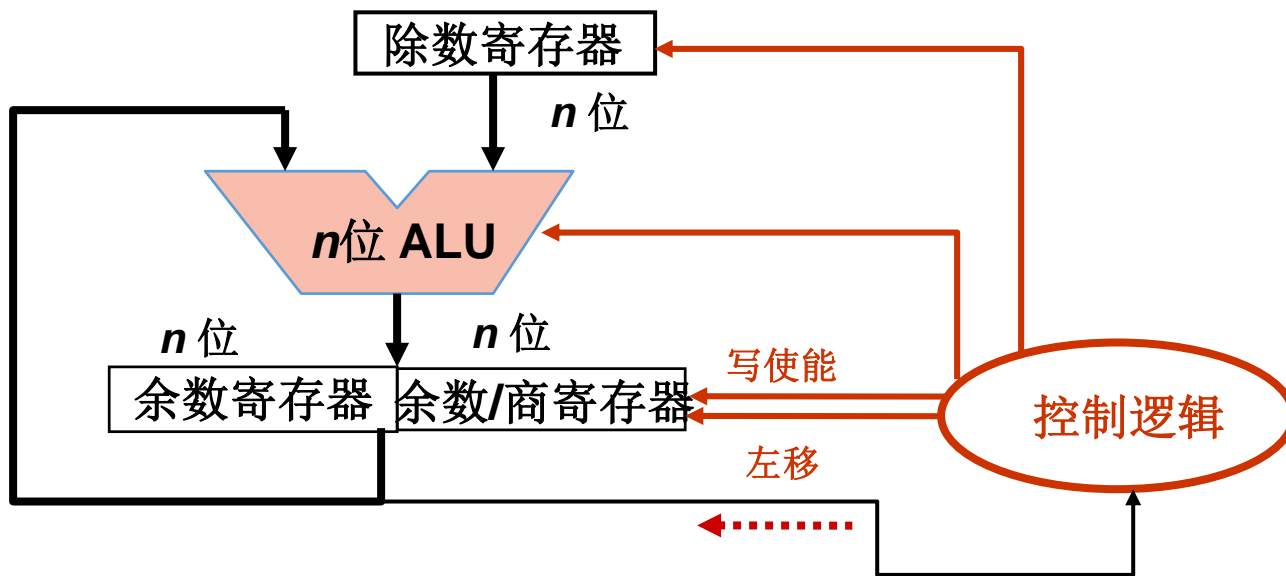
$$\begin{array}{r}
 3 \overline{) 7} \\
 \underline{6} \\
 1
 \end{array}$$

	余数		除数		商
初始化	00000111		00110000		0000
	00000111	->	00011000		0000
不够减	00000111		00011000	<-	0000
	00000111	->	00001100		0000
不够减	00000111		00001100	<-	0000
	00000111	->	00000110		0000
够减	00000001	-	00000110	<-	0001
	00000001	->	00000011		0001
不够减	00000001		00000011	<-	0010



除法 (续)

- 实现



除法 (续)

• 例子 (续)

$$\begin{array}{r}
 0010 \\
 0011 \overline{) 00000111} \\
 \underline{0000} \\
 000111 \\
 \underline{0000} \\
 00111 \\
 \underline{0011} \\
 0001 \\
 \underline{0000} \\
 0001
 \end{array}$$

$$\begin{array}{r}
 2 \\
 3 \overline{) 7} \\
 \underline{6} \\
 1
 \end{array}$$

	余数	商	除数
初始化	0000	0111	0011
< -	0000	111	0011
不够减	0000	1110	0011
< -	0001	110	0011
不够减	0001	1100	0011
< -	0011	100	0011
够减 -	0000	1001	0011
< -	0001	001	0011
不够减	0001	0010	0011



除法 (续)

- 如何判断 “**够减**”：余数是否足够 “大”
 - 如果余数和除数的符号相同：**减法**
 - 如果余数和除数的符号不同：**加法**

中间余数 R	除数 Y	减法		加法	
		0	1	0	1
0	0	够	不够	----	----
0	1	----	----	够	不够
1	0	----	----	不够	够
1	1	不够	够	----	----



除法 (续)

- 步骤过程
 - 通过在前面加n位符号扩展被除数，并存储在余数寄存器和商寄存器中
 - 将余数和商左移，判断是否“够减”
 - 如果“够”，则做减法（同号）或者加法（异号），并上商为1
 - 如果“不够”，则上商为0
 - 重复以上步骤
 - 如果除数和被除数不同号，则将商替换为其相反数
 - 余数存在余数寄存器中



除法 (续)

• 例子

$$\begin{array}{r}
 0011 \overline{) 11111001} \\
 \underline{+ 0000} \\
 111001 \\
 \underline{+ 0000} \\
 11001 \\
 \underline{+ 0011} \\
 1111 \\
 \underline{+ 0000} \\
 1111
 \end{array}$$

$\xrightarrow{\text{blue arrow}} 1110$
 $\xrightarrow{\text{blue arrow}} 1111$

	余数	商	除数
初始化	1111	1001	0011
< -	1111	< - 001	0011
+	0010	001	0011
不够减 (恢复)	- 1111	0010	0011
< -	1110	< - 010	0011
+	0001	010	0011
不够减 (恢复)	- 1110	0100	0011
< -	1100	< - 100	0011
够减	+ 1111	1001	0011
< -	1111	001	0011
+	0010	001	0011
不够减 (恢复)	- 1111	0010	0011
	\downarrow 1111	\downarrow 1110	



除法 (续)

- 问题：恢复余数成本高
- 大致思路：不恢复余数
 - 只考虑减法
 - 如果余数 R_i 足够大

$$R_{i+1} = 2R_i - Y$$

- 如果余数 R_i 不够大

$$R_{i+1} = 2(R_i + Y) - Y = 2R_i + Y$$



除法 (续)

- 步骤过程
 - 通过在前面加n位符号扩展被除数，并存储在余数寄存器和商寄存器中
 - 如果除数和被除数符号相同，则做减法；否则，做加法
 - 如果余数和除数符号相同，则商 $Q_n = 1$ ；否则， $Q_n = 0$
 - 如果余数和除数符号相同， $R_{i+1} = 2R_i - Y$ ；否则， $R_{i+1} = 2R_i + Y$
 - 如果新的余数和除数符号相同，使商为1；否则，使商为0
 - 重复以上步骤



除法 (续)

- 步骤过程 (续)
 - 左移商
 - 如果商是负的 (被除数和除数的符号不同), 商加1
 - 余数和被除数符号不同, 修正余数
 - 若被除数和除数符号相同, 最后余数加除数; 否则, 最后余数减除数



除法 (续)

• 示例

$$\begin{array}{r}
 \begin{array}{r}
 11101 \\
 0011 \overline{) 11111001} \\
 + 0011 \\
 \hline
 00101001 \\
 - 0011 \\
 \hline
 0010001 \\
 - 0011 \\
 \hline
 000101 \\
 - 0011 \\
 \hline
 11111 \\
 + 0011 \\
 \hline
 0010
 \end{array} \\
 \begin{array}{r}
 -2 \\
 3 \overline{) -7} \\
 -6 \\
 \hline
 -1
 \end{array}
 \end{array}$$

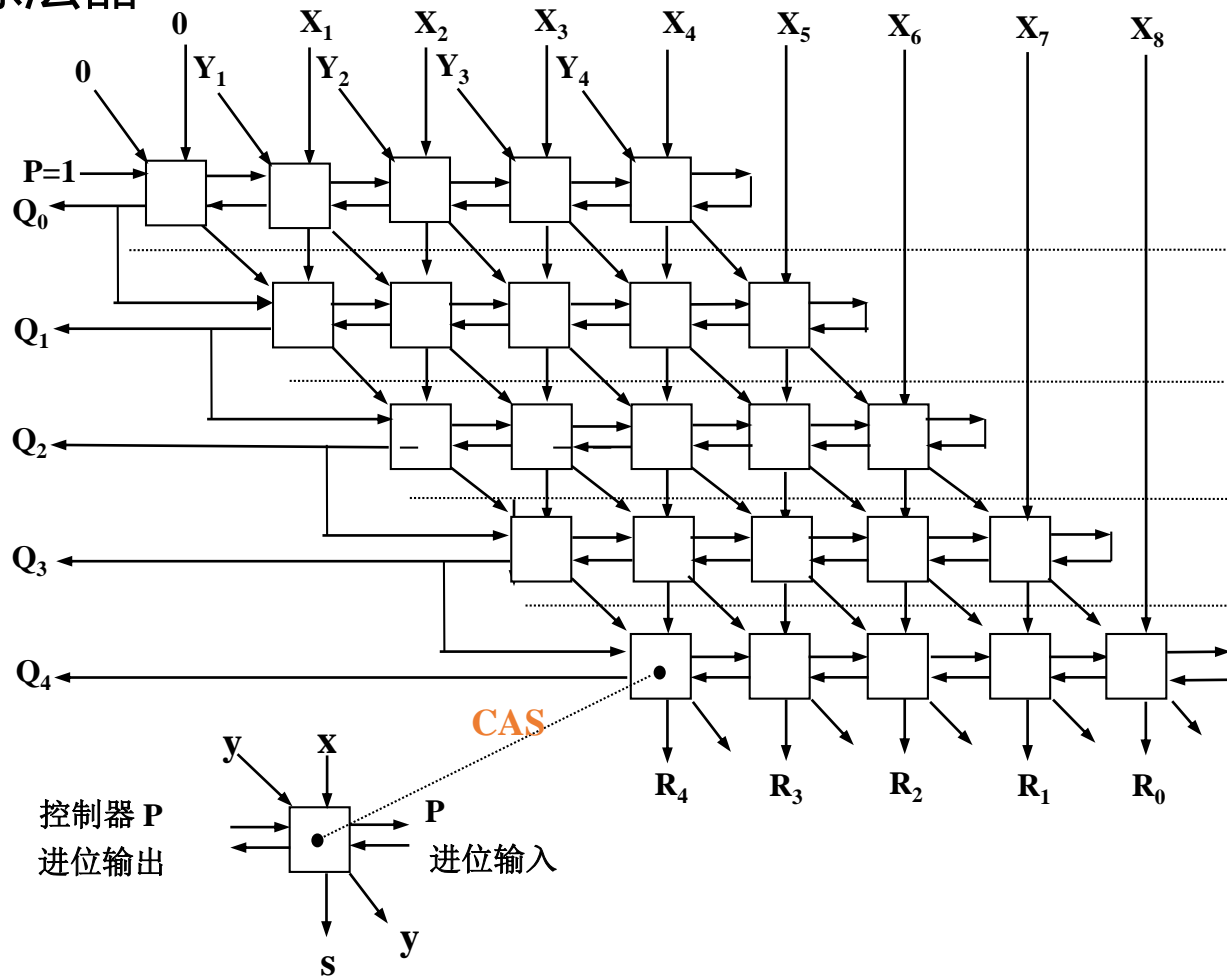
1110 (blue arrow from 11101)
 1111 (green arrow from 0010)

	余数	商	除数
初始化	1111	1001	0011
+	0010	1001 1	0011
<-	0101	<- 0011	0011
-	0010	0011 1	0011
<-	0100	<- 0111	0011
-	0001	0111 1	0011
<-	0010	<- 1111	0011
-	1111	1111 0	0011
<-	1111	<- 1110	0011
+	0010	1110 1	0011
	↓ -	↓ <-, +1	
	1111	1110	



除法 (续)

- 阵列除法器



总结

- 算术逻辑单元 (ALU)
 - 全加器
 - 串行进位加法器, 全先行进位加法器, 部分先行进位加法器
- 补码表示的整数运算
 - 加法: 溢出判定
 - 减法: 硬件实现
 - 乘法: 布斯算法
 - 除法: 恢复余数 / 不恢复余数



谢谢

rentw@nju.edu.cn



南京大學
NANJING UNIVERSITY