计算机组成原理

第二章 运算方法和运算器

刘 超
中国地质大学 计算机学院

主要内容

- 2.1 数据与文字的表示方法
- 2.2 定点加法、减法运算
- 2.3 定点乘法运算
- 2.4 定点除法运算
- 2.5 定点运算器的组成
- 2.6 浮点运算和浮点运算器

2.2 定点加法、减法运算

- 补码加法运算公式
- 补码减法运算公式
- 溢出检测
- 二进制加/减法器
- 十进制加法器

补码的加法运算公式

- $[X+Y]_{补} = [X]_{补} + [Y]_{补}$ {mod 2 || mod 2^{n+1}}
  两数和的补码等于两数补码之和。
  - 对于定点小数，在模2意义下，任意两数的补码之和等于该两数之和的补码。
    - 前提：运算结果无溢出（$|x+y|<1$）。符号位的进位可以任意丢掉而不影响计算正确性。
  - 同理，对于定点整数，在模2^{n+1}意义下，任意两数的补码之和等于该两数之和的补码。

补码加法公式证明（以定点小数为例）

- $[X+Y]_{补} = [X]_{补} + [Y]_{补}$
  1. $x>0 \quad y>0$ （无需证明）
  2. $x>0 \quad y<0$
  3. $x<0 \quad y>0$ （2/3证明相同）
  4. $x<0 \quad y<0$
- 只需证明2/4两种情况即可

(2) $x>0$ $y<0$

$[x]_{补}=x$ $[y]_{补}=2+y$

$[x]_{补}+[y]_{补}=x+2+y=2+(x+y)$

当$x+y<0$时

$$上式 = [x+y]_{补} \pmod 2$$

当$x+y>0$时

$2+(x+y)>2$ 进位位舍去

$[x]_{补}+[y]_{补}=2+(x+y)=x+y \pmod 2$

$=[x+y]_{补} \pmod 2$

(4) $x<0 \quad y<0$

$x+y<0$

$[x]_{补}=2+x \quad [y]_{补}=2+y$

$[x]_{补}+[y]_{补}=2+x+2+y=2+(2+x+y) \mod 2$

$-1<x+y<0$

故 $0<2+x+y<2$

故 $2+(2+x+y)>2$

$2+(2+x+y) \mod 2 =(2+x+y)$

$=[x+y]_{补} \mod 2$

例 子

例 8 $x = 0.1001$，$y = 0.0101$

求 $x + y$

解：

$$
\begin{array}{rcl}
[x]_{补} & = & 0.1001 \\
+\ [y]_{补} & = & 0.0101 \\
\hline
[x]_{补} + [y]_{补} & = & 0.1110 = [x+y]_{补}
\end{array}
$$

$\therefore x + y = 0.1110$

例子

例 9　　$x = 0.1011$，$y = -0.0101$

求 $x + y$

验证

解：

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

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

符号位进位舍去，正常结果

$\therefore\ x + y = 0.0110$

- 由以上两例看到,补码加法的特点:
  - 一是符号位要作为数的一部分一起运算,
  - 二是要在模2的意义下相加,即超过2的进位要丢掉。

2.2 定点加法、减法运算

- 补码加法运算公式
- 补码减法运算公式
- 溢出检测
- 二进制加/减法器
- 十进制加法器

补码减法运算公式

- $[X-Y]_补 = [X]_补 - [Y]_补 = [X]_补 + [-Y]_补$
- $[-Y]_补 = -[Y]_补$ （mod 2）

- 补码减法公式证明:

$[X-Y]_补 = [X]_补 - [Y]_补$????

$[X-Y]_补 = [X]_补 + [-Y]_补$ (加法公式)

$[-Y]_补 = -[Y]_补$????

$[-Y]_补 + [Y]_补 = [Y+(-Y)]_补 = [0]_补 = 0$

故$[-Y]_补 = -[Y]_补$成立

- 提问，验证：y=0.1101 ????

补码减法运算公式

- $[-Y]_{补} = -[Y]_{补}$
- 已知$[y]_{补}$，求$[-y]_{补}$的法则是：**对$[y]_{补}$包括符号位"求反且最末位加1",即可得到$[-y]_{补}$**。写成运算表达式，则为：$[-y]_{补} = \neg[y]_{补} + 2^{-n}$
- 其中符号$\neg$表示对$[y]_{补}$作包括符号位在内的求反操作,2^{-n}表示最末位的1。

例 子

例 10 $\mathbf{x_1 = -0.1110}$，$\mathbf{x_2 = +0.1101}$

求 $[x_1]_{补},[-x_1]_{补},[x_2]_{补},[-x_2]_{补}$

解:

$[\mathbf{x_1}]_{补}=1.0010$，

$[\mathbf{-x_1}]_{补} = \neg[x_1]_{补}+2^{-4} = 0.1101 + 0.0001$

$=0.1110$，

$[\mathbf{x_2}]_{补} = 0.1101$

$[\mathbf{-x_2}]_{补} = \neg[x2]_{补}+2\text{-}4=1.0010 + 0.0001$

$=1.0011$ ，

例 子

例 11　　$x = +0.1101$，$y = +0.0110$

求 $x - y$

解:

$[x]_{补} = 0.1101$

$[y]_{补} = 0.0110$，　$[-y]_{补} = 1.1010$

$$
\begin{array}{lcl}
 & [x]_{补} = & 0.1101 \\
+ & [-y]_{补} = & 1.1010 \\
\hline
 & [x-y]_{补} = & \boxed{1}\,0.0111
\end{array}
$$

符号位进位舍去，正常结果

$\therefore x - y = +0.0111$

2.2 定点加法、减法运算

- 补码加法运算公式
- 补码减法运算公式
- 溢出检测
- 二进制加/减法器
- 十进制加法器

溢出概念与检测方法

- 在定点小数机器中,数的表示范围为$|x|<1$. 在运算过程中如出现大于1的现象,称为“溢出”。在定点机中,正常情况下溢出是不允许的。
- 正正得负，负负得正，结果溢出。
- 例12、例13

$$\begin{array}{r} 0.1011 \\ +\ 0.1001 \\ \hline \boxed{1}.0100 \end{array}$$

正正得负，正溢出

$$\begin{array}{r} 1.0011 \\ +\ 1.0101 \\ \hline \boxed{1}\,\boxed{0}.1000 \end{array}$$

负负得正，负溢出

判断“溢出”是否发生的几种检测方法

- 第一种方法：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。
  - 设两数符号位为$f_0 f_1$
  - 和数符号位f_s

$$V = \bar{f}_0 \bar{f}_1 f_s + f_0 f_1 \bar{f}_s = \overline{\overline{\bar{f}_0 \bar{f}_1 f_s} \bullet \qquad\qquad}$$

（三个与非门）

判断“溢出”是否发生的几种检测方法

- 第二种方法：单符号位法 $V = C_f \oplus C_n$

符号位进位C_f，最高位进位C_n

$$
\begin{array}{r}
0.1010 \\
+\ 0.0100 \\
\hline
0.1110
\end{array}
\qquad
\begin{array}{r}
1.\ 010 \\
+\ 1.\ 100 \\
\hline
\boxed{1}\ 1.0110
\end{array}
$$

$C_f = 0$，$C_n = 0$

$C_f = 1$，$C_n = 1$

$$
\begin{array}{r}
0.\ 011 \\
+\ 0.\ 001 \\
\hline
\boxed{1}.0100
\end{array}
\qquad
\begin{array}{r}
1.0011 \\
+\ 1.0101 \\
\hline
\boxed{1}\ \boxed{0}.1000
\end{array}
$$

$C_f = 0$，$C_n = 1$

$C_f = 1$，$C_n = 0$

判断“溢出”是否发生的几种检测方法

- 第三种方法：变形补码（双符号位法）$V = f_1 \oplus f_2$

$$
\begin{array}{r}
00.1010 \\
+\ 00.0100 \\
\hline
00.1110
\end{array}
$$

正常结果

$$
\begin{array}{r}
11.1010 \\
+\ 11.1100 \\
\hline
\boxed{1}\ 11.0110
\end{array}
$$

符号位进位舍去，正常结果

$$
\begin{array}{r}
00.1011 \\
+\ 00.1001 \\
\hline
\boxed{01}.0100
\end{array}
$$

非正常符号位，溢出

$$
\begin{array}{r}
11.0011 \\
+\ 11.0101 \\
\hline
\boxed{1}\ \boxed{10}.1000
\end{array}
$$

非正常符号位，溢出

变形补码

- 变形补码也称为“模4补码”,可使模2补码所能表示的数的范围扩大一倍。变形补码定义为

$$[x]_{补}=\begin{cases} x & 2>x\geq 0 \\ 4+x & 0>x\geq -2 \end{cases}$$

或用同余式表示为：$[x]_{补}=4+x \pmod 4$

下式也同样成立：$[x]_{补}+[y]_{补}=[x+y]_{补} \pmod 4$

1. 两个符号位都看作数码一样参加运算。
2. 两数进行以4为模的加法,即最高符号位上产生的进位要丢掉。采用变形补码后，如果两个数相加后，其结果的符号位出现“01”或“10”两种组合时,表示发生溢出。最高符号位永远表示结果的正确符号。

例子(利用变形补码计算)

- [例14] x = +0.1100, y = +0.1000,求 x + y。

  [解:] $[x]_补$ = 00.1100, $[y]_补$ = 00.1000

$$
\begin{array}{rr}
[x]_{补} & 00.1100 \\
\underline{+[y]_{补}} & \underline{00.1000} \\
 & \boxed{01}.0100
\end{array}
$$

两个符号位出现 "01",表示已溢出,即结果大于+1。

- [例15] x = -0.1100, y = -0.1000,求 x + y 。

  [解:] $[x]_补$ = 11.0100, $[y]_补$ = 11.1000

$$
\begin{array}{rr}
[x]_{补} & 11.0100 \\
\underline{+[y]_{补}} & \underline{11.1000} \\
 & 1\boxed{10}.1100
\end{array}
$$

两个符号位出现 "10",表示已溢出,即结果小于-1。

作 业

- 变形补码：5(2)(3)、6(1)(2)

2.2 定点加法、减法运算

- 补码加法运算公式
- 补码减法运算公式
- 溢出检测
- 二进制加/减法器
- 十进制加法器

基本的二进制加法/减法器

- 补充知识点：
- 1）逻辑门：反相门（T）/与门（2T）/或门（2T）/
  与非门（T）/或非门（T）/与或非门（T）/异或门（3T）
- 2)常用的逻辑运算公式：
  吸收律：　$A+AB=A,\ A(A+B)=A$
  反演律：　$\overline{A+B}=\bar{A}\bar{B},\ \overline{AB}=\bar{A}+\bar{B}$
- 3）卡诺图化简法。
  - 提问：$F = AB\bar{C}\bar{D} + ABC\bar{D} + AB\bar{C}D + ABCD + A\bar{B}CD + A\bar{B}C\bar{D} + \bar{A}BC\bar{D} + \bar{A}BCD$
- 半加器　　　—> 全加器
  $H_i=A_i \oplus B_i$　　　$F_i=A_i \oplus B_i \oplus C_i$
  $C_{i+1}=A_iB_i$　　　$C_{i+1}=A_iB_i+A_iC_i+B_iC_i$
- 加法器：串行加法器/并行加法器（串行链式、先行进位）

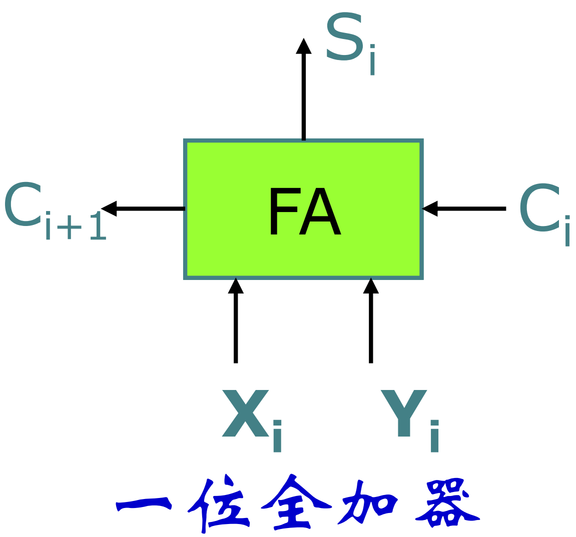
基本的二进制加法/减法器

- 二进制加法运算

$$
\begin{array}{rl}
 & X_{n-1} \cdots\cdots\cdots X_2\ X_1\ X_0 \\
+ & Y_{n-1} \cdots\cdots\cdots Y_2\ Y_1\ Y_0 \\
\hline
 & ?_{n-1} \cdots\cdots\cdots ?_2\ ?_1\ ?_0
\end{array}
$$

- 各位逐位相加，进位从右至左传递。
- 首先要考虑一位加法，然后考虑进位链。

带进位链的一位全加器



加数X_i	加数Y_i	低位进位C_i	和数S_i	进位C_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$S_i = X_i \oplus Y_i \oplus C_i$$

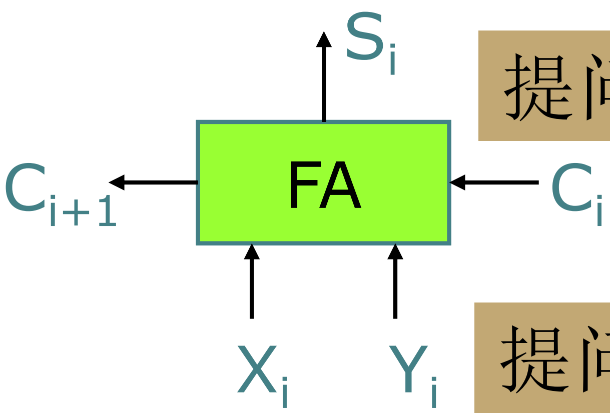
$$C_{i+1} = X_iY_i + (X_i \oplus Y_i)C_i$$

$$= \boldsymbol{X_iY_i + X_iC_i + Y_iC_i}$$

一位全加器逻辑实现

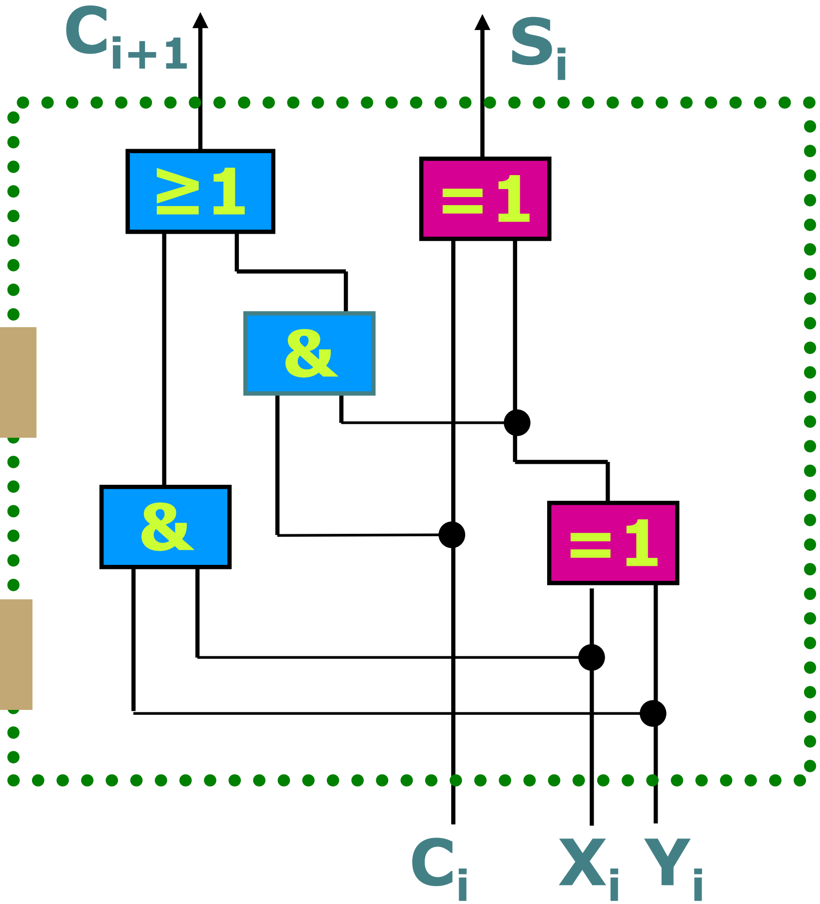
$$S_i = X_i \oplus Y_i \oplus C_i$$

$$C_{i+1} = X_i Y_i + (X_i \oplus Y_i) C_i$$



提问1：区别？

提问2：延迟？



S_i延迟:6T(2个异或)

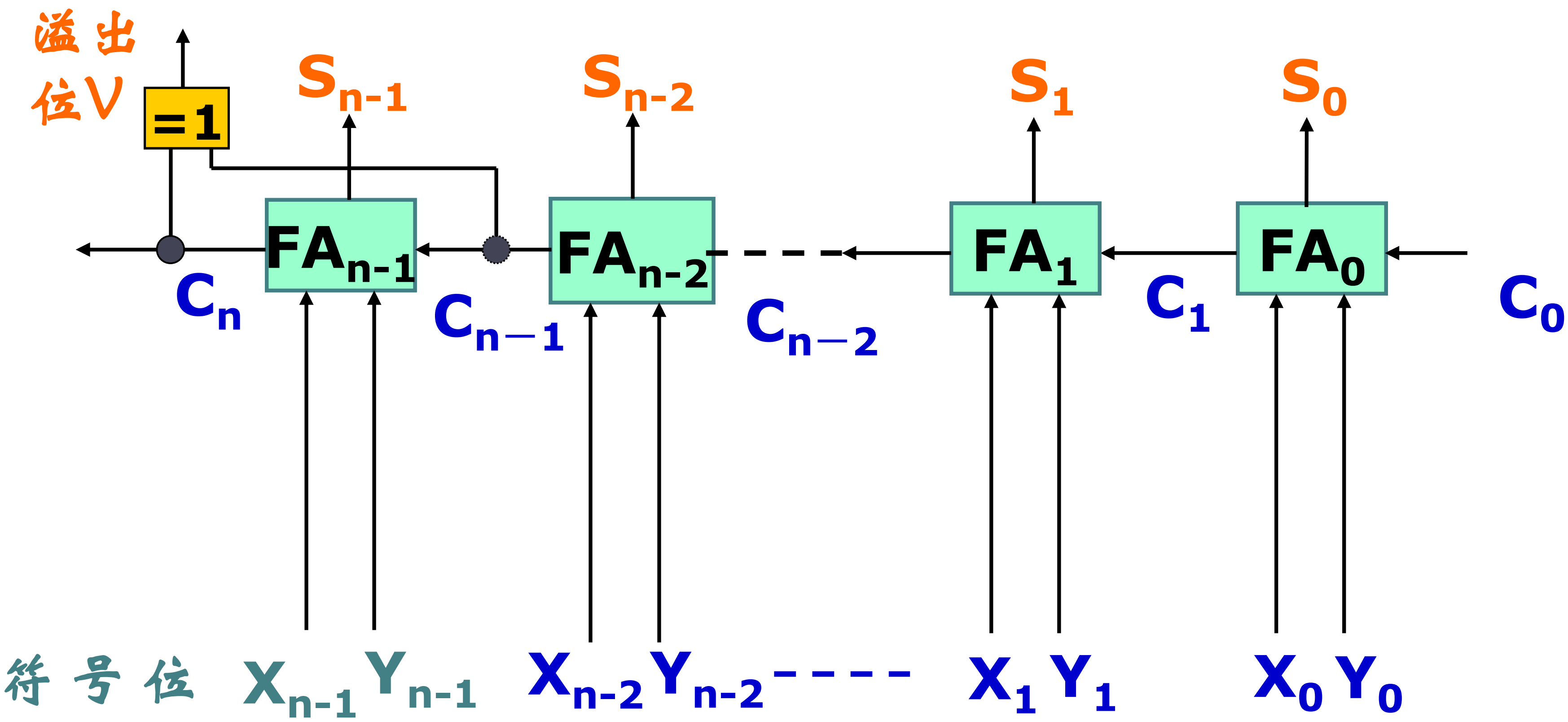
C_{i+1}延迟:7T(异或+与+或)

C_{i+1}延迟 :5T(异或+2个与非)

多位加法器

- N位加法器包含n个全加器
- 将多个一位全加器串联
- 低位进位输出连接到高位进位输入

单符号位补码加法器电路



补码减法电路实现

- 补码减法可以转换为加法

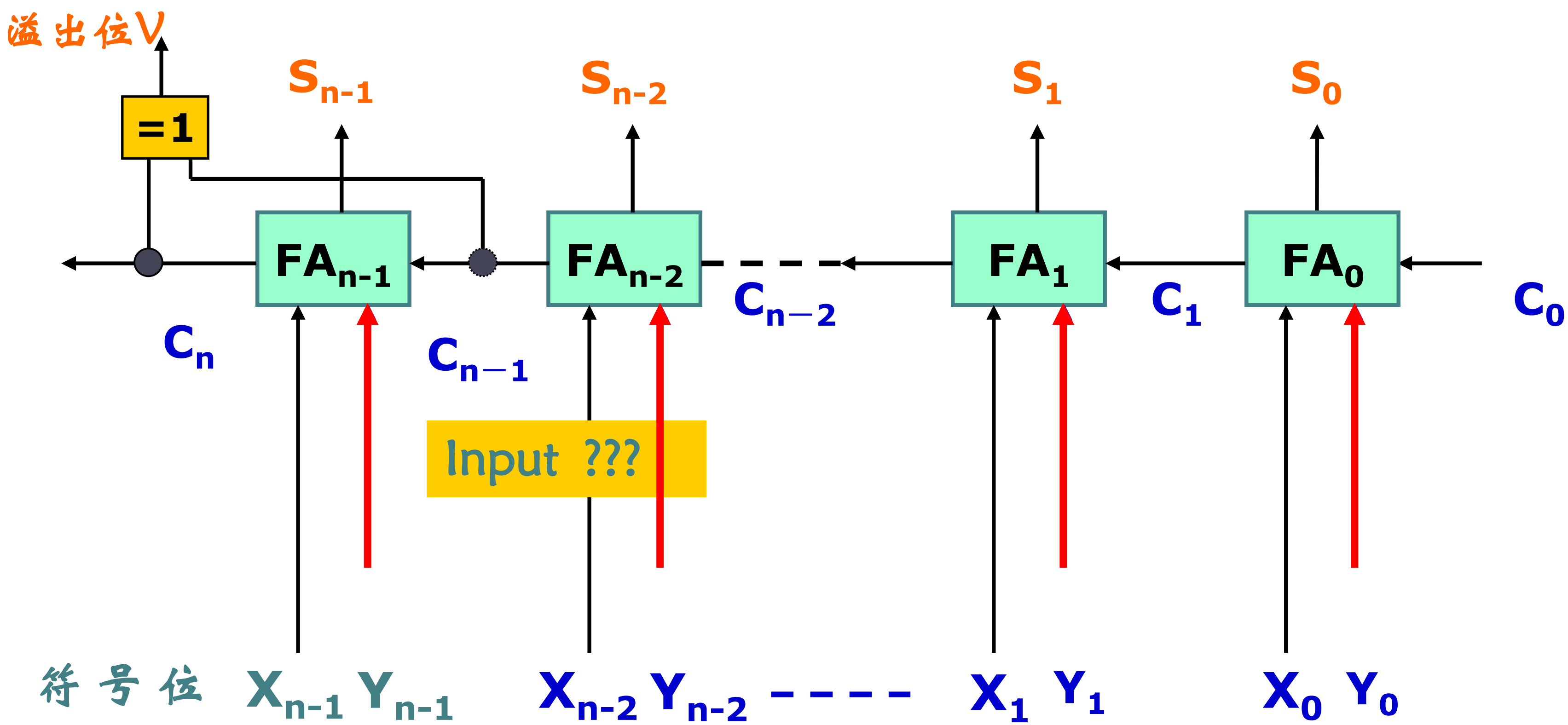
$[X]_{补} - [Y]_{补} = [X]_{补} + [-Y]_{补}$

实现减法的关键是求**减数Y乘以负1的补码**。

方法：**将$Y_{补}$连同符号位一起逐位取反末位加1**

公式： $[-y]_{补} = \neg[y]_{补} + 2^{-n}$ **(2.21)**

加法器的改造



- 加法器输入$Y_{补}$作加法，　如果输入$[-Y]_{补}$则作减法

加法器的改造

- 引入控制位 M
- M=0时送入加法器的是$Y_{补}$
  - 两操作数做加法运算
- M=1时送入加法器的是$[-Y]_{补}$
  - 两操作数做减法运算

补码减法电路实现

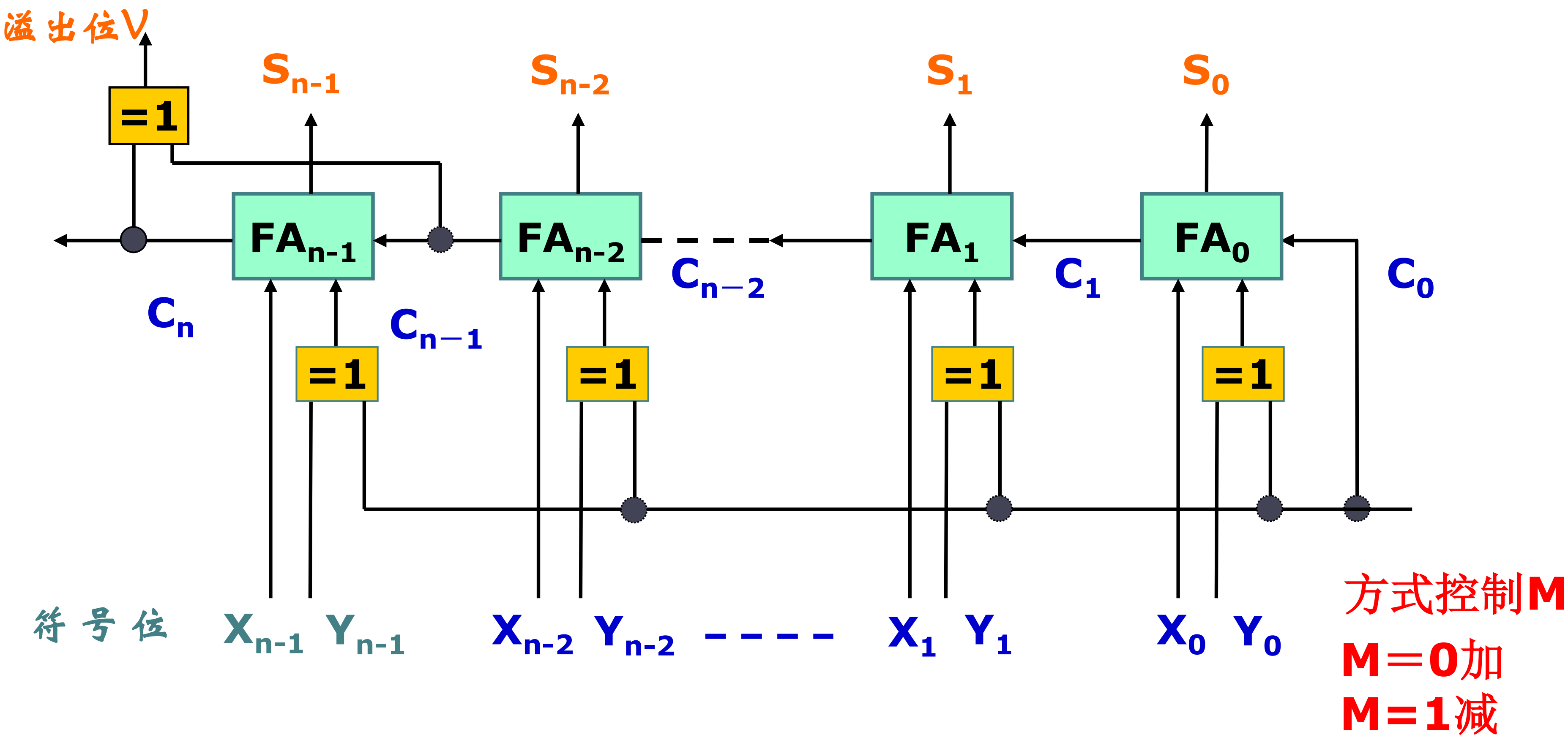
- 方法：将$Y_{补}$连同符号位一起逐位取反末位加一

$$[-Y]_{补} = [[Y]_{补}]_{补}$$

Y_i	M(0加/1减)	Input
0	0	0
1	1	0
1	0	1
0	1	1

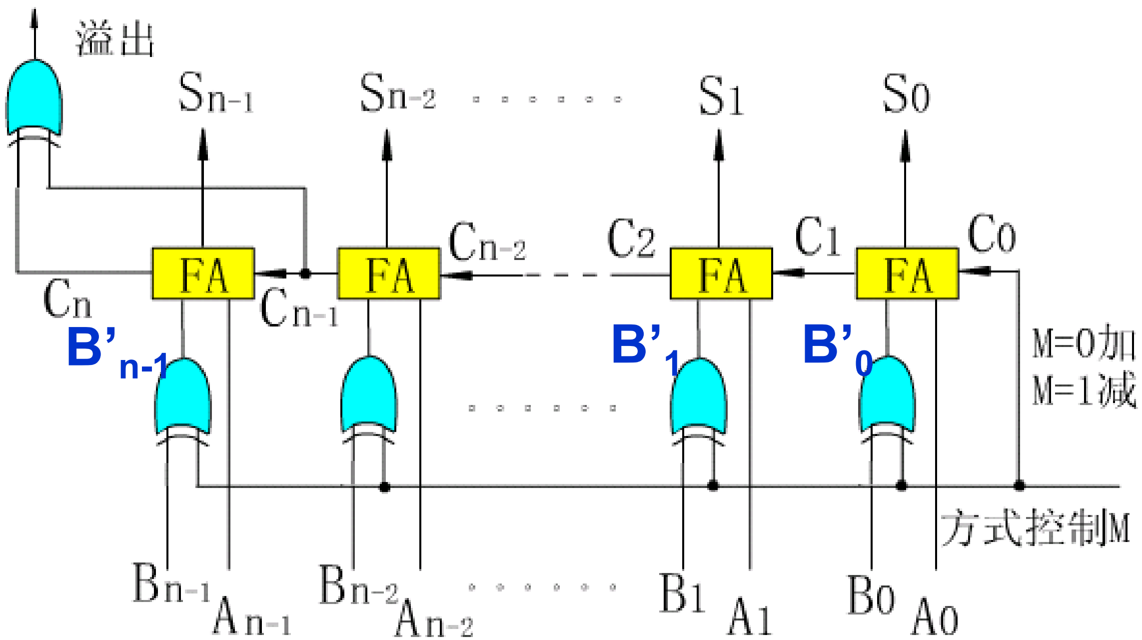
$$Input = Y_i \oplus M$$

单符号补码加/减器电路实现



行波进位补码加减法器

（使用补码、可以实现加减法）



1）加法实现

方式控制码M=0

$B'_i=B_i\oplus M=B_i\oplus 0=B_i$

$[S_i]_{补}=[A_i]_{补}+[B'_i]_{补}$

$=[A_i]_{补}+[B_i]_{补}$

2）减法实现

方式控制码M=1

$B'_i=B_i\oplus M=B_i\oplus 1=\overline{B_i}$

$C_0=1$

$[S_i]_{补}=[A_i]_{补}+[-B_i]_{补}$

$=[A_i-B_i]_{补}$

3）判溢

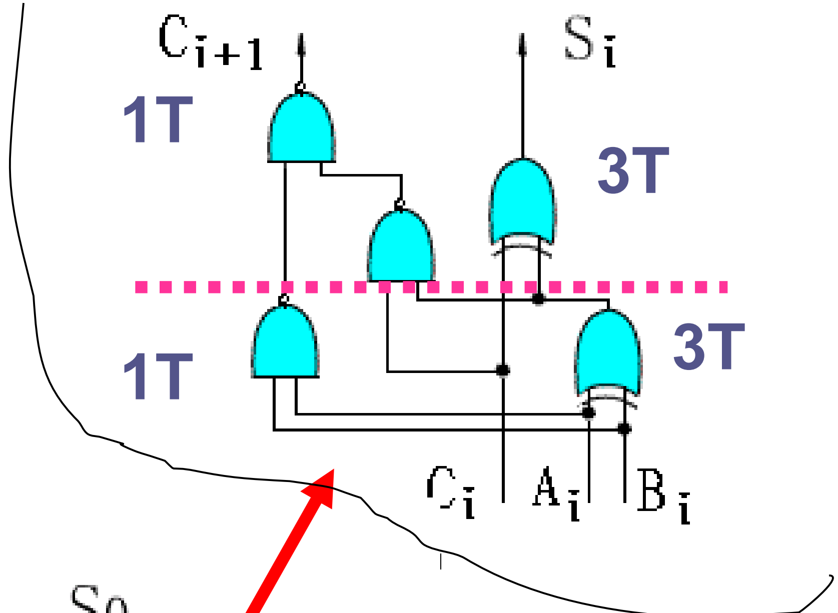
$OVR=C_n\oplus C_{n-1}$

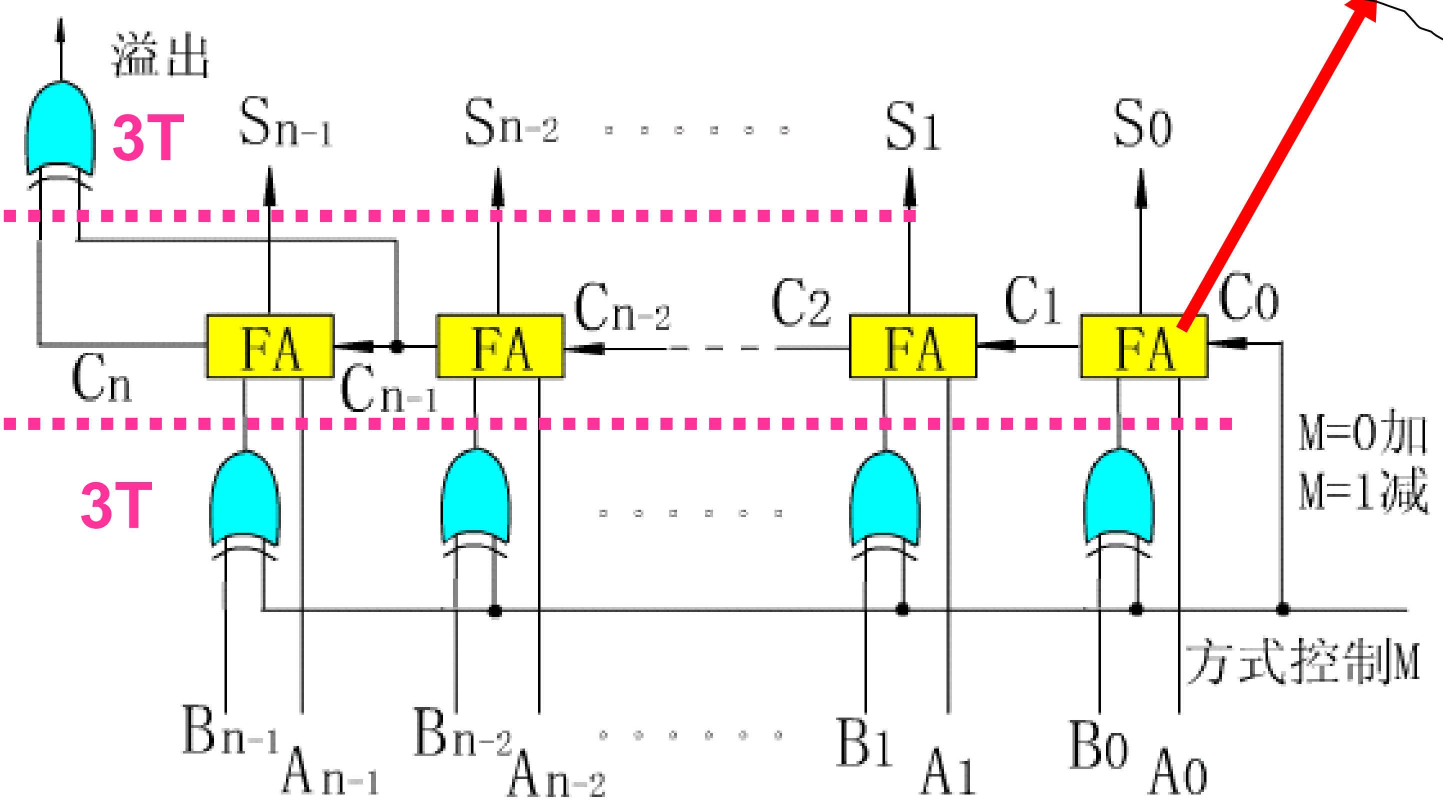
4）结果形成时间

S_{n-1}：(n-1)*2T+9T

溢出：n*2T+9T

提问：任意C_i、S_i延迟？





2.2 定点加法、减法运算

- 补码加法运算公式
- 补码减法运算公式
- 溢出检测
- 二进制加/减法器
- 十进制加法器

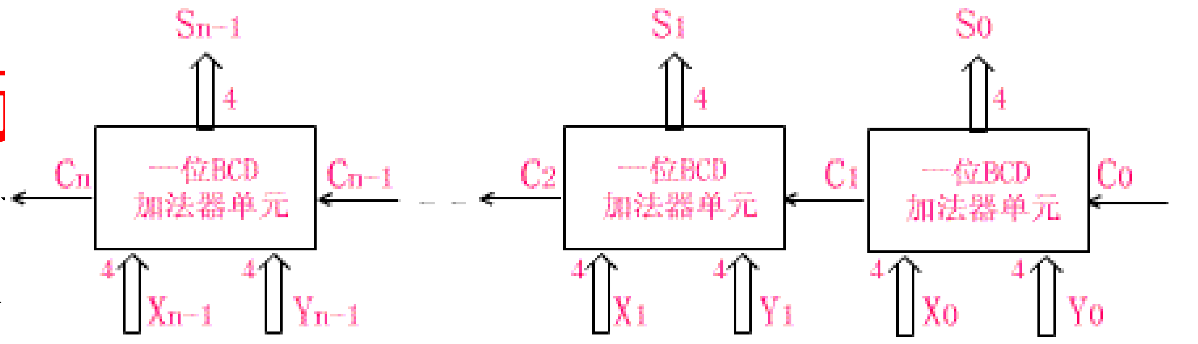
十进制加法器

- 十进制加法器可由BCD码来设计,它可以在二进制加法器的基础上加上适当的“校正”逻辑来实现。
- **校正条件:**
  - **用BCD码完成十进制数运算时,当和数大于9时,必须对和数进行加6修正。**
  - 设S'_i代表4位二进制数和, C'_{i+1}为输出进位,而S_i代表正确的BCD和, C_{i+1}代表正确的进位。
  - 当 $x_i + y_i + C_i < 10$时, $S_i = S'_i$;

    当$x_i + Y_i + C_i \geq 10$时, $S_i = S'_i + 6$
  - 当$C'_{i+1} = 1$或$S'_i \geq 10$时,输出进位$C_{i+1} = 1$。
  - $C_{i+1} = 1$, 校正因子为6; $C_{i+1} = 0$, 校正因子为0

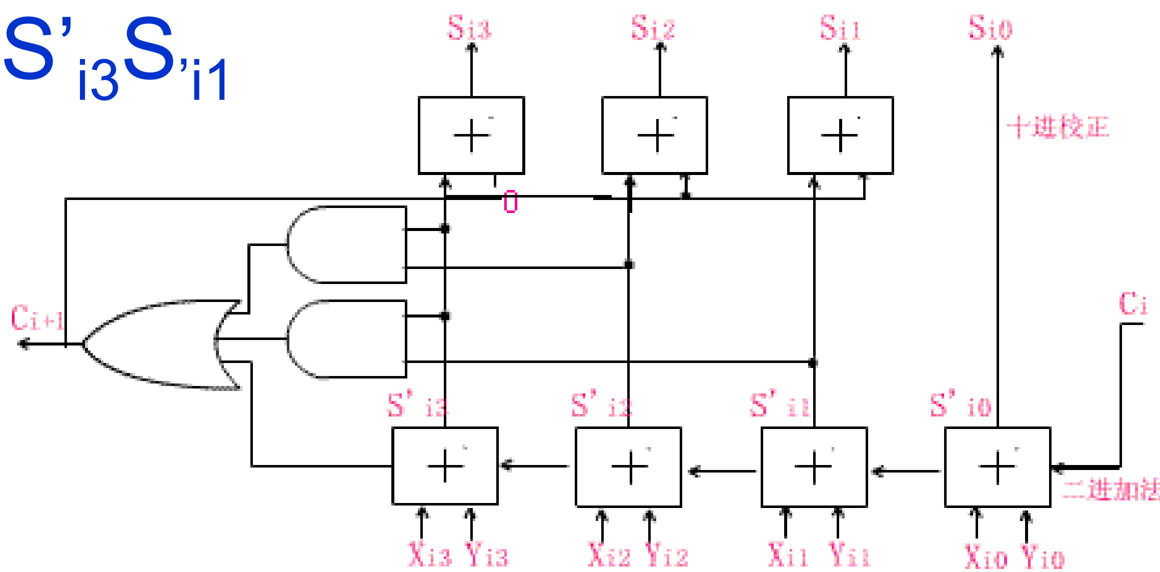
十进制加法器逻辑结构

- n位BCD码行
  所示，每一位
  逻辑结构示于



(a) n位数字的行波进位加法器

$F=C'_{i+1}+S'_{i3}S'_{i2}+S'_{i3}S'_{i1}$



(b) 一位BCD加法器单元的逻辑结构

小节作业：

- 14, 余三码编码的十进制加法器单元电路

主要内容

- 2.1 数据与文字的表示方法
- 2.2 定点加法、减法运算
- 2.3 定点乘法运算
- 2.4 定点除法运算
- 2.5 定点运算器的组成
- 2.6 浮点运算和浮点运算器

定点乘法运算

- 原码一位乘法
- 阵列乘法
- 补码一位乘法
- 补码阵列乘法

原码乘法

- 在定点计算机中,原码相乘的运算规则是:
  - 1) 乘积的符号位由两数的符号位按异或运算得到;
  - 2) 而乘积的数值部分则是两个正数相乘之积。
- 设n位被乘数和乘数用定点小数表示(定点整数类似)
- 被乘数 $[x]_{原}=x_f.x_{n-1}\cdots x_1x_0$

  乘数 $[y]_{原}=y_f.y_{n-1}\cdots y_1y_0$
- 乘积 $[z]_{原}=$

  $(x_f\oplus y_f)+(0.x_{n-1}\cdots x_1x_0)(0.y_{n-1}\cdots y_1y_0)$

原码一位乘法(了解)

```
          0. 1 1 0 1  (x)
×         0. 1 0 1 1  (y)
-----------------------------
             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  (z)
```

- 设 $x=0.1101$, $y=0.1011$.求乘积过程如下:

```
     0. 1 1 0 1                    x
×    0. 1 0 1 1                    y
-----------------------------------
     0. 0 0 0 0 1 1 0 1            x共右移4次(需2n位长)
     0. 0 0 0 1 1 0 1              x共右移3次
     0. 0 0 0 0 0 0                x共右移2次
+    0. 0 1 1 0 1                  x共右移1次
-----------------------------------
     0. 1 0 0 0 1 1 1 1        (z)
```

部分积累加的数学表示

- 设被乘数x,乘数y都是小于1的n位定点正小数:

$x=0.x_1x_2\ldots x_n$ ， $y=0.y_1y_2\ldots y_n$ 其乘积为

$x\bullet y=x(0.y_1y_2\ldots y_n)=x(y_1 2^{-1}+y_2 2^{-2}+\ldots+y_n 2^{-n})$

$=2^{-1}(y_1x+2^{-1}(y_2x+2^{-1}(\ldots+2^{-1}(y_{n-1}x+2^{-1}(y_nx+0))\ldots)))$

令Z_i表示第i次部分积，则上式可写成如下递推公式:

$$z_0=0$$

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

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

$$\vdots$$

$$z_i=2^{-1}(y_{n-i+1}x+z_{i-1})$$

$$\vdots$$

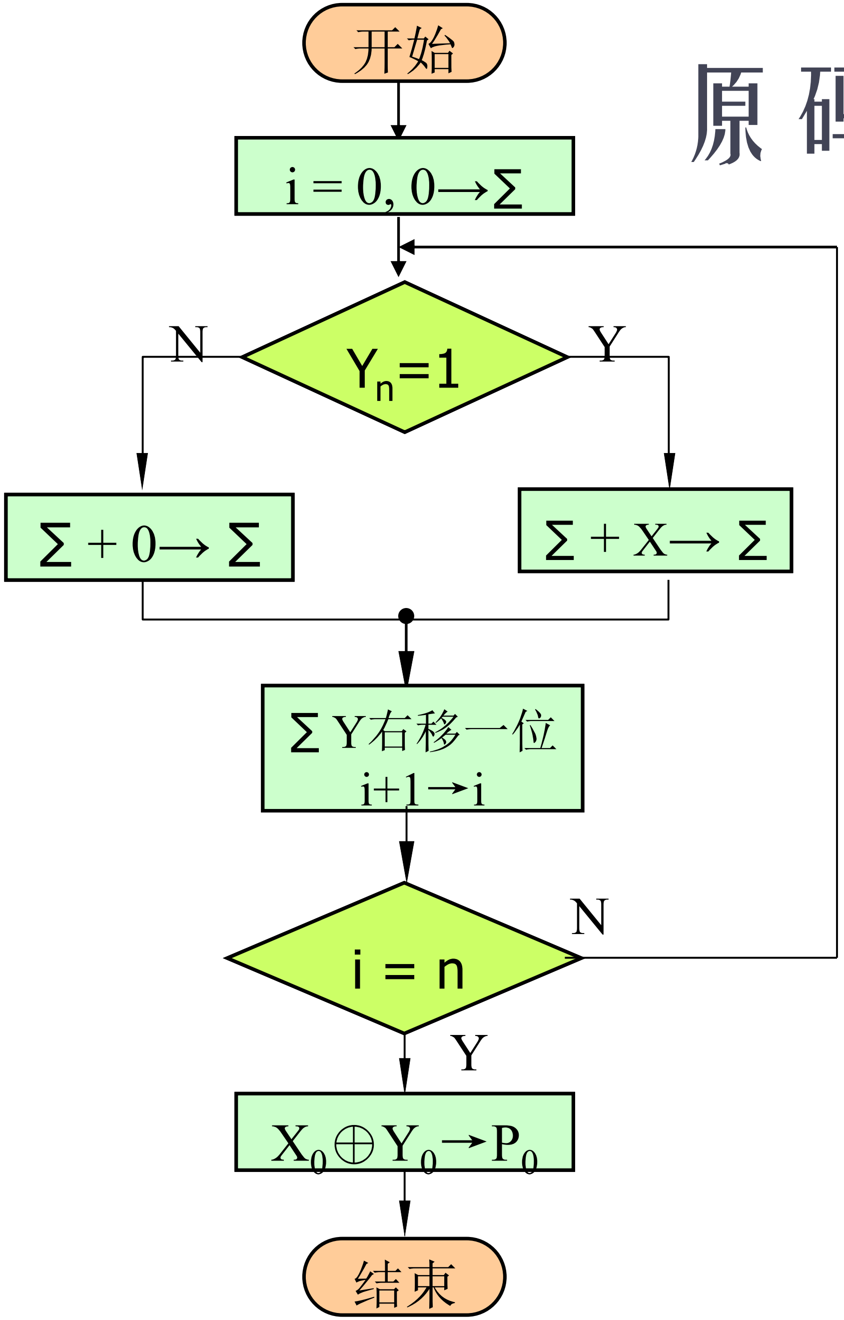
$$z_n=x\bullet y=2^{-1}(y_1x+z_{n-1})$$

实现原码一位乘法的规则：

$z_i=2^{-1}(y_{n-i+1}\,x+z_{i-1})$

- 求x·y，
  - 1）需设置一个保存部分积的累加器。
  - 2）乘法开始时，令部分积的初值$z_0=0$，然后求$y_n x$加上z_0，右移1位得第1个部分积z_1。
  - 3）将$y_{n-1}x$加上z_1，再右移1位得第2个部分积z_2。
  - 4）依此类推，直到求得$y_1 x$加上z_{n-1}并右移1位得最后部分积z_n，即得乘积$x\cdot y=z_n$。
- 显然，两个n位数相乘，需重复进行n次“加”及“右移”操作，才能得到最后乘积。

原码一位乘法算法流程

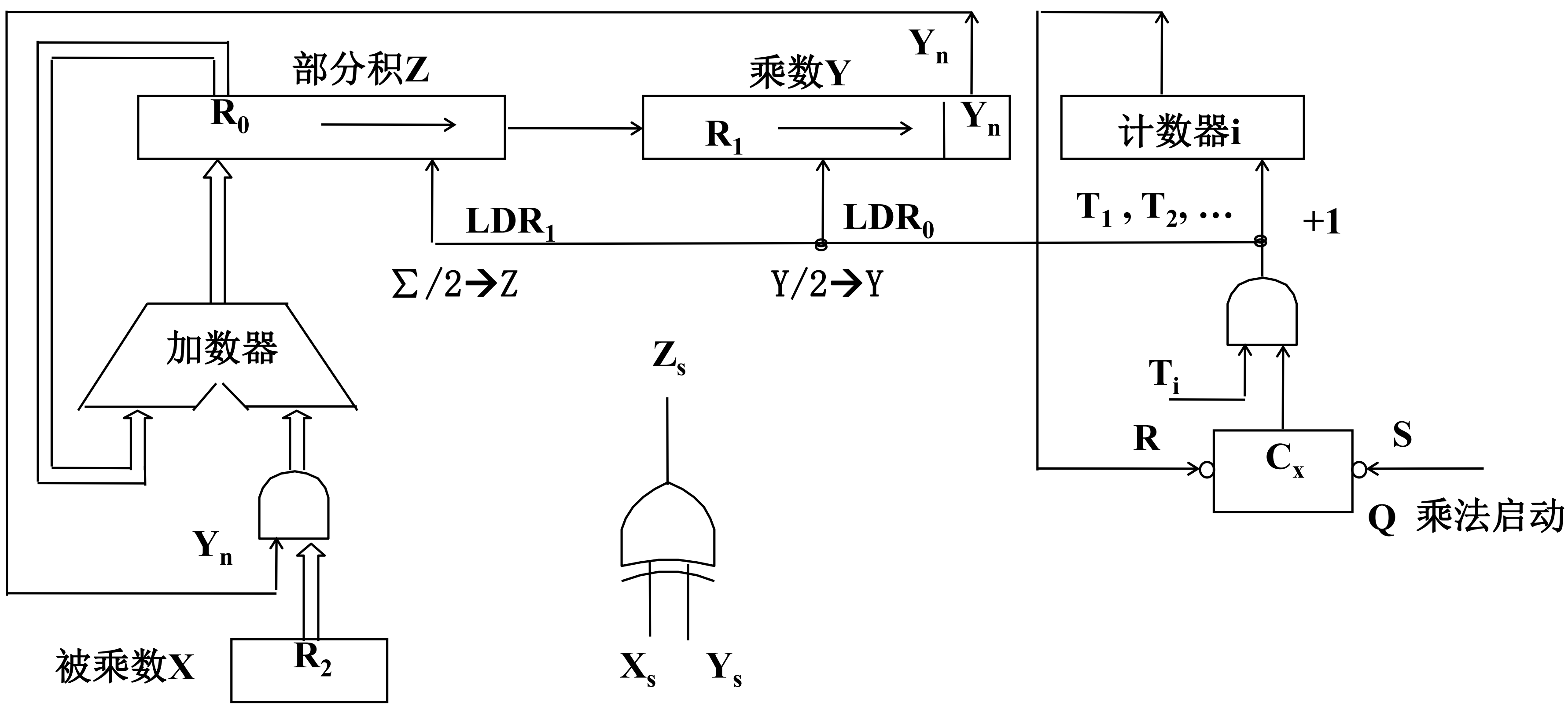


- 加法次数，n次
- 作为加法，一定移位
- 符号位单独计算

例子:已知X=0.1101 Y=-0.1011，计算$[X]_{原}\times[Y]_{原}$

部分积	乘数	判断位	说明
00.0000			$P_0=0$
+ 00.1101		$Y_0.1011$	$Y_4=1,+\|X\|$
00.1101			
→ 00.0110	1	$Y_0.101$	右移一位得P_1
+ 00.1101			$Y_4=1,+\ \|X\|$
$\boxed{01.}$0011	1		
→ 00.1001	11	$Y_0.10$	右移一位得P_2
+ 00.0000			$Y_4=0,+0$
00.1001	11		
→ 00.0100	111	$Y_0.1$	右移一位得P_3
+ 00.1101			$Y_4=1,+\ \|X\|$
01.0001	111		
→ 00.1000	1111	Y_0	右移一位得$P_4=\|X\|\cdot\|Y\|$

原码一位乘法逻辑结构



原码一位乘法逻辑结构原理图

	部分积R_0	乘数R_1	Y_n判断位
	0 0 0 0 0	0 1 0 1	1
+ |X|	0 1 1 0 1		
	0 1 1 0 1	0 1 0 1	1
→	0 0 1 1 0	1 0 1 0	1
+ |X|	0 1 1 0 1		
	1 0 0 1 1		
→	0 1 0 0 1	1 1 0 1	0
+ 0	0 0 0 0 0		
	0 1 0 0 1		
→	0 0 1 0 0	1 1 1 0	1
+ |X|	0 1 1 0 1		
	1 0 0 0 1		
→	0 1 0 0 0	1 1 1 1	0

|X|= 0.1101，
|Y|= 0.1011

总结原码一位乘法

- 需要三个寄存器：
  - 1）R_0存放部分积z（乘法开始前R_0应清“0”，因为z_0=0）
  - 2）R_2存放被乘数x；
  - 3）R_1存放乘数y。
- 乘法开始时先从乘数的最低位y_n开始，以后则使用y_{n-1},y_{n-2}，…，y_1，因此乘数寄存器R_1是具有右移功能的移位寄存器。
- 假定加法器不具备右移功能，那么由于部分积需要右移，R_0也应当是具有右移功能的移位寄存器。
- 除了三个寄存器Ro，R_1，R_2外，还需一个加法器和一个计数器，前者完成部分积与位积的累加，后者对移位的次数进行计数，以便判断乘法运算是否结束。

工作原理

（I）乘法开始时，“启动”信号使控制触发器C_x置“1”，于是开启时序脉冲T。

（II）当乘数寄存器R_1最末位为“1”时，部分积Z_i和被乘数X在加法器中相加，其结果输出至R_0的输入端。

（III）一旦打入控制脉冲T到来，控制信号LDR_0使部分积右移一位，与此同时，R_1也在控制信号LDR_1作用下右移一位，且计数器i计数一次。

（IV）当计数器i=n时，计数器的溢出信号使触发器C_x置“0”，关闭时序脉冲T，乘法宣告结束。

若将R_0和R_1连接起来，乘法结束时乘积的高n位部分在R_0，低n位部分在R_1，R_1中原来的乘数Y由于移位而全部丢失。所得乘积为2n＋1位（其中包括1位符号位）.

- 乘法操作的总时间为 $t_m=n(t_a+t_r)$

其中t_a为加法器执行一次加法操作的时间，t_r为执行一次移位操作的时间，n为尾数位数。如果加法操作和移位操作同时进行，则t_r项可省去。

定点乘法运算

- 原码一位乘法
- 阵列乘法
- 补码一位乘法
- 补码阵列乘法

阵列乘法器

- 早期计算机中为了简化硬件结构,采用串行的1位乘法方案,即多次执行“加法—移位”操作来实现。
- 特点：
  - 这种方法并不需要很多器件。
  - 然而，串行方法执行太慢。
- 自从大规模集成电路问世以来,出现了各种形式的流水式阵列乘法器,它们属于并行乘法器。

不带符号的阵列乘法器

- 设有两个不带符号的二进制整数：

$$A = a_{m-1} \ldots a_1 a_0$$

$$B = b_{n-1} \ldots b_1 b_0$$

它们的数值分别为a和b,即

$$a = \sum_{i=0}^{m-1} a_i 2^i \qquad b = \sum_{j=0}^{n-1} b_j 2^j$$

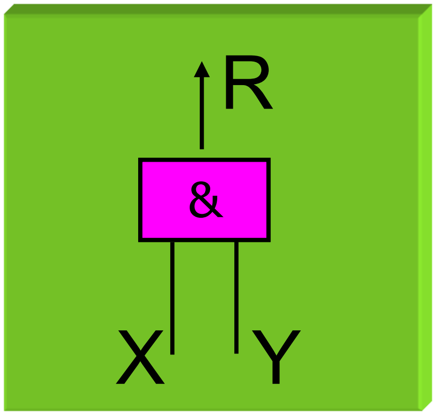
在二进制乘法中,被乘数A与乘数B相乘,产生$m+n$位乘积P:

$$P = p_{m+n-1} \cdots p_1 p_0$$

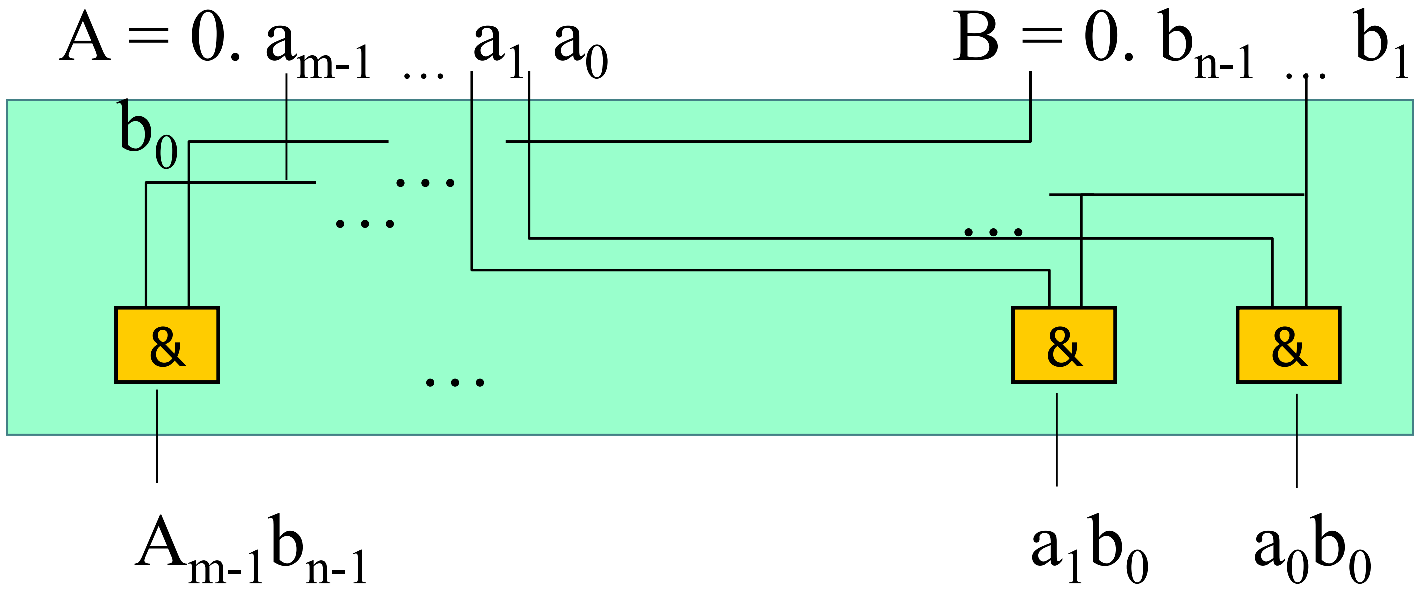
乘积P 的数值为

$$\boxed{p = ab = \left(\sum_{i=0}^{m-1} a_i 2^i\right)\left(\sum_{j=0}^{n-1} b_j 2^j\right) = \sum_{i=0}^{m-1}\sum_{j=0}^{n-1}(a_i b_j)2^{i+j} = \sum_{k=0}^{m+n-1} p_k 2^k}$$

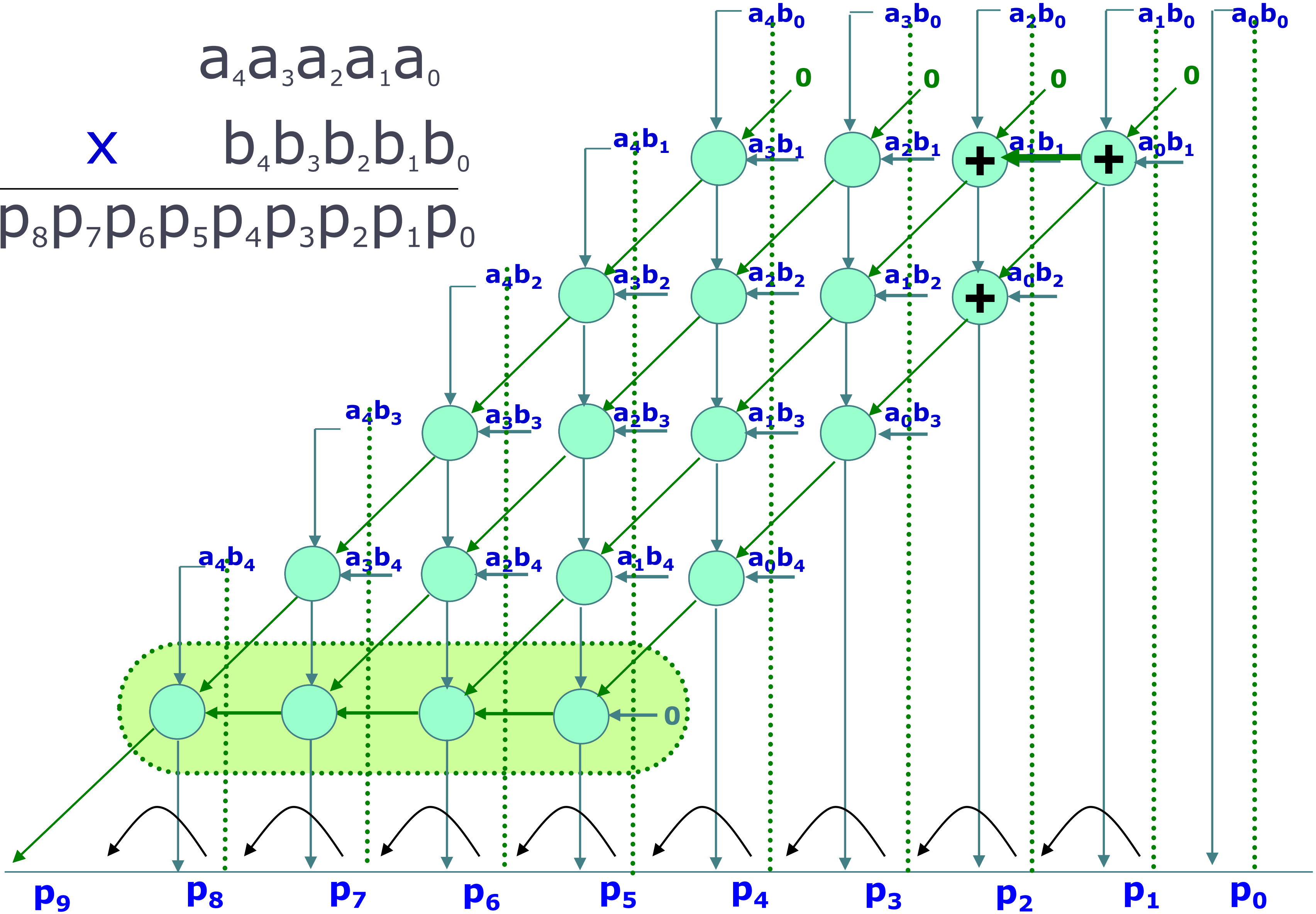
- 一个与门即可实现一位乘法



- 相加数产生部件
  - 经过一级门电路延迟，即可得到所有的相加数



$$
\begin{array}{r}
a_4a_3a_2a_1a_0 \\
\times \quad b_4b_3b_2b_1b_0 \\
\hline
p_9p_8p_7p_6p_5p_4p_3p_2p_1p_0
\end{array}
$$

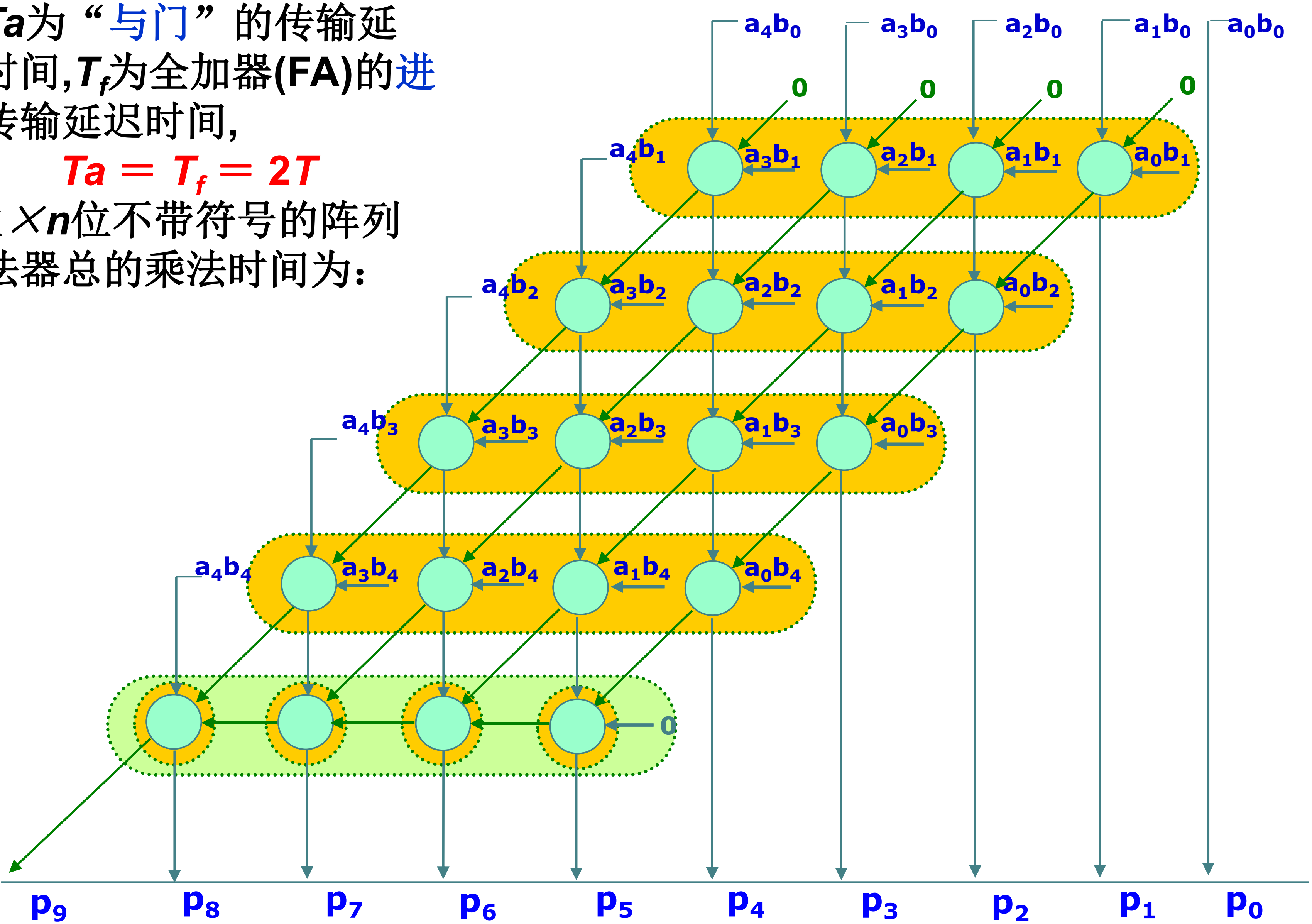


5位无符号数阵列乘法器电路

令*Ta*为“与门”的传输延迟时间，T_f为全加器(FA)的进位传输延迟时间，

$$Ta = T_f = 2T$$

*n*位×*n*位不带符号的阵列乘法器总的乘法时间为：



$$t_m = Ta + (n-1)\times 6T + (n-1)\times T_f + 3T = 5T + (n-1)\times 8T = (8n-3)T$$

带符号的阵列乘法器

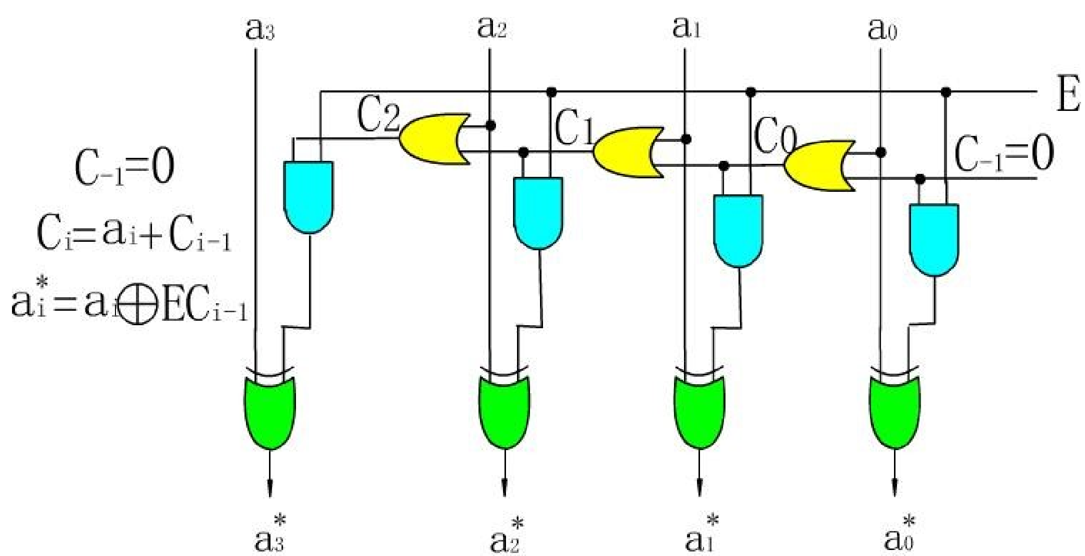
◻ (1) 对2求补器电路

算术运算部件设计中经常用到的求补电路。一个具有使能控制的二进制对2求补器电路图2.6，其逻辑表达式如下：

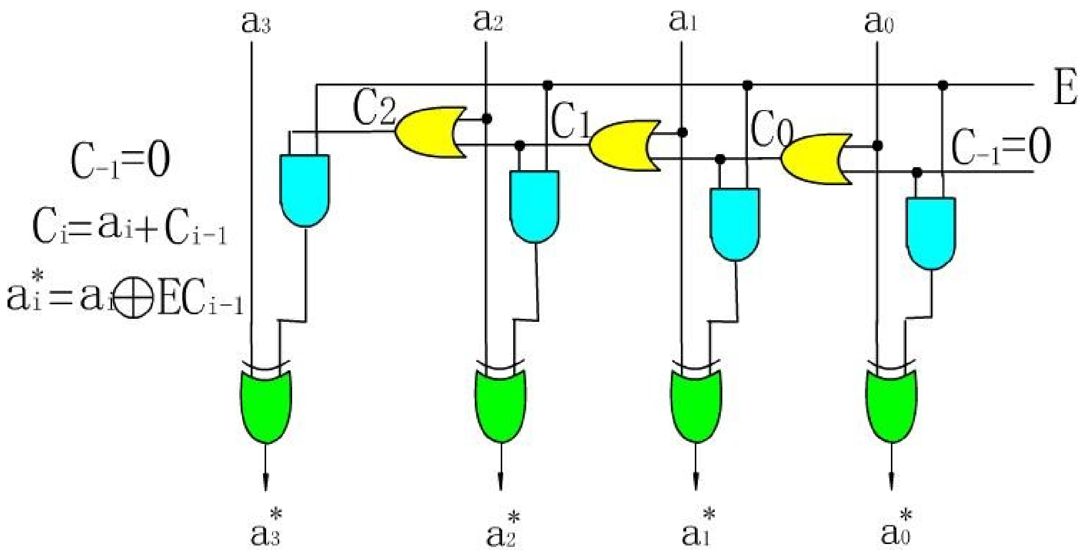
$C_{-1}=0$, $C_i=a_i+C_{i-1}$

$a_i^*=a_i\oplus EC_{i-1}, 0\leqslant i\leqslant n$

例如,在一个4位的对2求补器中,如果输入数为1010,那么输出数应是0110,其中从右算起的第2位,就是所遇到的第一个“1”的位置



分 析



- 用这种对2求补器来转换一个$(n+1)$为带符号的数，所需的总时间延迟为

  $t_{TC}=n\cdot 2T+5T=(2n+5)T$

  其中每个扫描级需$2T$延迟，而$5T$则是由于“与”门和“异或”门引起的。

带符号的阵列乘法器

- $(n+1)\times(n+1)$位带求补器的阵列乘法器逻辑方框图，见图2.7

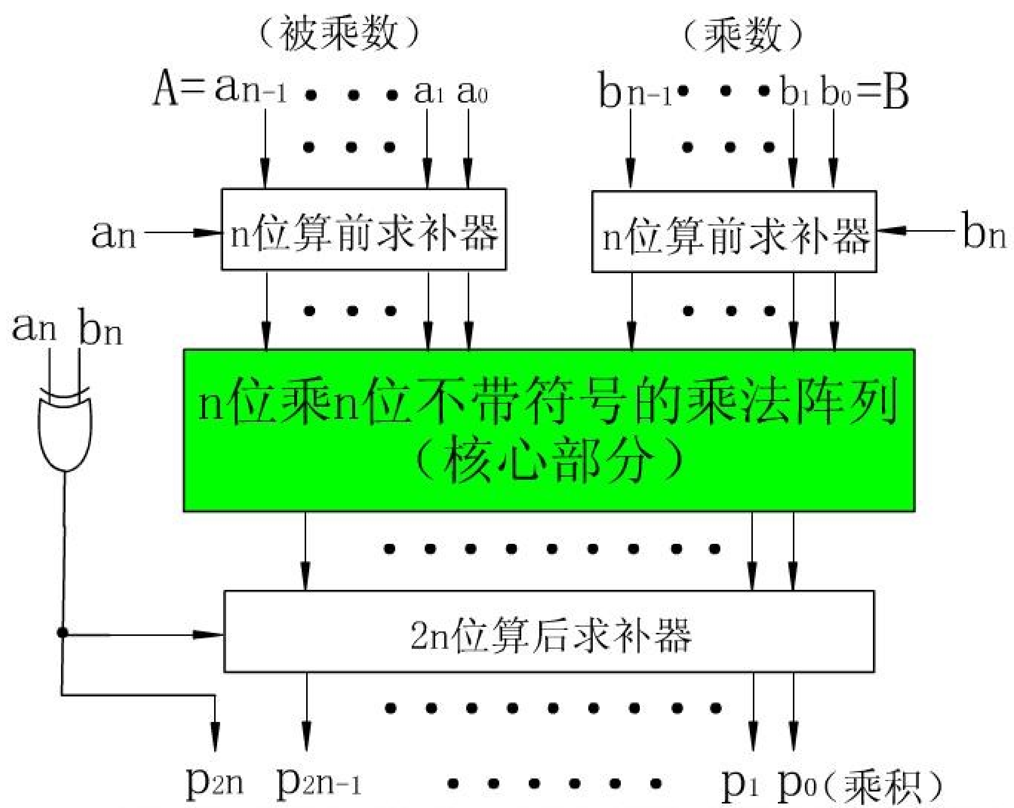


图2.7 （n+1）位乘（n+1）位带补级的阵列乘法器

- 在这种逻辑结构中,共使用三个求补器。其中两个**算前求补器**：将两个操作数***A***和***B***在被不带符号的乘法阵列(核心部件)相乘以前,先变成正整数。
- **算后求补器**：当两个输入操作数的符号不一致时,把运算结果变成带符号的数。

带符号的阵列乘法器

- 设$A=a_n a_{n-1}\ldots a_1 a_0$和$B=b_n b_{n-1}\ldots b_1 b_0$均为用定点表示的$(n+1)$位带符号整数。在必要的求补操作以后,$A$和$B$的码值输送给$n\times n$位不带符号的阵列乘法器,并由此产生$2n$位真值乘积:

$$A\cdot B=P=p_{2n-1}\cdots p_1 p_0$$

$p_{2n}=a_n\oplus b_n$，其中p_{2n}为符号位。

- 图2.7所示的带求补级的阵列乘法器既适用于原码乘法,也适用于间接的补码乘法。不过在原码乘法中,算前求补和算后求补都不需要。而间接的补码阵列乘法所需要增加的硬件较多，时间大约比原码阵列乘法增加1倍。

例 子

- 例17 设 $x = +15$, $y = -13$,用带求补器的原码阵列乘法器求出乘积 $x \cdot y = ?$

[解:]设最高位为符号位,则输入数据为,

$[x]_{原} = 01111$ $[y]_{原} = 11101$

- 符号位单独考虑,算前求补级后 $|x| = 1111$, $|y| = 1101$ 算后经求补级输出并加上乘积符号位1,则原码乘积值为111000011。
- 换算成真值是 $x \cdot y = (-11000011)_2 = (-195)_{10}$

十进制数验证: $x \times y = 15 \times (-13) = -195$相等。

```
          1 1 1 1
×         1 1 0 1      符号位运算: 0⊕1=1
-----------------
          1 1 1 1
        0 0 0 0
      1 1 1 1
+   1 1 1 1
-----------------
  1 1 0 0 0 0 1 1
```

补码一位乘法(不作要求)

1) 被乘数[X]符号任意，乘数[Y]为正

$[X]_{补}=X_{0.}X_1X_2\ldots X_n$ $[Y]_{补}=0.Y_1Y_2\ldots Y_n$

$[X]_{补}\times[Y]_{补}=(2+X)\times Y=(2^{n+1}+X)\times Y$

$=2^{n+1}Y+XY$

$=2\times 2^{n}\times 0.Y_1Y_2\ldots Y_n+XY$

$=2(Y_1Y_2\ldots Y_n)+XY$

$=2+XY$

$=[X\times Y]_{补}$

$[X\times Y]_{补}=[X]_{补}\times[Y]_{补}$

补码一位乘法

1) 被乘数[X]符号任意，乘数[Y]为负数

$[X]_{补}=X_{0.}X_1X_2\ldots X_n$ $[Y]_{补}=1.Y_1Y_2\ldots Y_n$

$[Y]_{补}=2+Y \quad Y=[Y]_{补}-2$

$=0.Y_1Y_2\ldots Y_n-1$

$[X\times Y]_{补}=[X\times(0.Y_1Y_2\ldots Y_n-1)]_{补}$

$=[X\times 0.Y_1Y_2\ldots Y_n-X]_{补}$

$=[X\times 0.Y_1Y_2\ldots Y_n]_{补}-[X]_{补}$

$=[X]_{补}\times 0.Y_1Y_2\ldots Y_n-[X]_{补}$

$=[X]_{补}\times 0.Y_1Y_2\ldots Y_n-Y_0[X]_{补}$

定点除法运算

- 被除数 x,其原码为$[x]_{原}= x_f . x_{n-1} \cdots x_1 x_0$
  除数 y,其原码为 $[y]_{原}= y_f . y_{n-1} \cdots y_1 y_0$
- 则有商$q = x / y$,其原码为

$$[q]_{原}=(x_f \oplus y_f)+(0. x_{n-1} \cdots x_1 x_0 / 0. y_{n-1} \cdots y_1 y_0)$$

- 仅讨论数值部分的运算，限定条件：
  - 由于定点小数的绝对值小于1，如果被除数大于或等于除数，则商就大于或等于 1，产生溢出，这是不允许的。

定点除法运算

0.1101

0.1011) 0.1001　　不够减，商上0，

\- 0.01011　　除数右移1位，够减，减除数，商上1

0.001110

\- 0.001011　　除数右移1位，够减，减除数，商上1

0.0000110

\- 0.0001011　　除数右移1位，不够减，商上0

0.00001100　　除数右移1位，够减，减除数，商上1

\- 0.00001011

0.00000001

$x \div y$的商q=0.1101,余数为r=0.00000001。

定点除法运算

□除数右移等价于余数左移。

商$q=0.1101$,

余数为$r=r_4*2^{-4}=0.00000001$

```
          0.1101
        ___________
00.1011/  00.1001        x<y,商0
       ← 01.0010
          11.0101        被除数左移1位,2x>y, 商1,
        ___________      减y, 即+[-y]补 , 余数r1
          00.0111
       ← 00.1110         左移1位,2r1>y, 商1
          11.0101
        ___________
          00.0011        余数r2
       ← 00.0110         左移1位,2r2<y, 商0, r3
       ← 00.1100         左移1位,2r3>y, 商1
          11.0101
        ___________
          00.0001        r4
```

$x<y$,商0

被除数左移1位,$2x>y$, 商1, 减y, 即$+[-y]_补$, 余数r_1

左移1位,$2r_1>y$, 商1

余数r_2

左移1位,$2r_2<y$, 商0, r_3

左移1位,$2r_3>y$, 商1

r_4

恢复余数除法(淘汰，仅了解)

- 人会心算，一看就知道够不够减，但是机器Out，闷头先作减法，若余数为正，才知道够减；若余数为负，才知道不够减。不够减时必须恢复原来的余数，以便再继续往下运算。此方法称为**恢复余数法。**
- 如何判断是否够减？
  - 原码运算判断借位
  - 利用补码作减法，判断余数符号即可
- 余数为负数时，恢复余数的方法？
  - 即将余数加除数，恢复成原来的值
- 求下一位商，必须将余数左移一位，再与除数比较，再计算当前位商（或者恢复），余数再移位，再比较，如此反复，直到获得商所需要的位数为止。

例子：用恢复余数法求x÷y

例1 x=0.1001， y=0.1011，用恢复余数法求x÷y。

解：$[x]_{补}=0.1001$

$[y]_{补}=0.1011$，$[-y]_{补}=1.0101$

[x]补=0.1001， [y]补=0.1011， [-y]补=1.0101

被除数/余数	商	上商位	说明
00.1001			
+[−Y]补 11.0101			减Y比较
11.1110		0	余数R_0<0，商=0
+ [Y]补 00.1011			加Y恢复余数
00.1001		0	左移一位
← 01.0010			
+[−Y]补 11.0101			减Y比较
00.0111		0.1	余数R_1>0，商上1
← 00.1110			左移一位
+[−Y]补 11.0101			减Y比较
00.0011		0.11	R_2>0，商上1
← 00.0110			左移一位
+[−Y]补 11.0101			减Y比较
11.1011		0.110	R_3<0，商上0
+ [Y]补 00.1011			加Y恢复
00.0110		0.110	左移一位
← 00.1100			减Y比较
+[−Y]补 11.0101			
00. 0001		0.1101	R_4>0，商上1

不恢复余数法（加减交替法）

- **恢复余数法存在的问题?**
  - 由于需要进行恢复余数操作，除法操作过程的步数不确定，故运算时间不固定，影响除法速度，控制也复杂。实际应用通常采用不恢复余数乘法。
- **不恢复余数法（加减交替法）：**
  - 步数固定，控制简单。

不恢复余数法（加减交替法）

- 设某次余数为R_i，求下位商需要将R_i左移一位，然后减去除数进行比较，此过程可表为

  $2R_i$-Y **(<==>左移，减Y)**

- 当结果小于0时商上0。此时，为获得下一位商需要恢复余数，左移一位，减Y比较三步操作，即

  （**$2R_i$-Y**）+Y= $2R_i$;//恢复

  $2*2R_i$ -Y = $4R_i$ –Y ;//左移后再减

  = 2*（**$2R_i$-Y**）+Y **(<==>继续左移，加Y)**

- 结论：第*i*步除数的余数R_i=$2R_{i-1}$-y若为负，要求得下一步的新余数R_{i+1}，不必恢复余数，只要将R_i继续左移一位（乘2）再加上y即得R_{i+1}，然后再由R_{i+1}的正负决定上商值。

不恢复余数法（加减交替法）

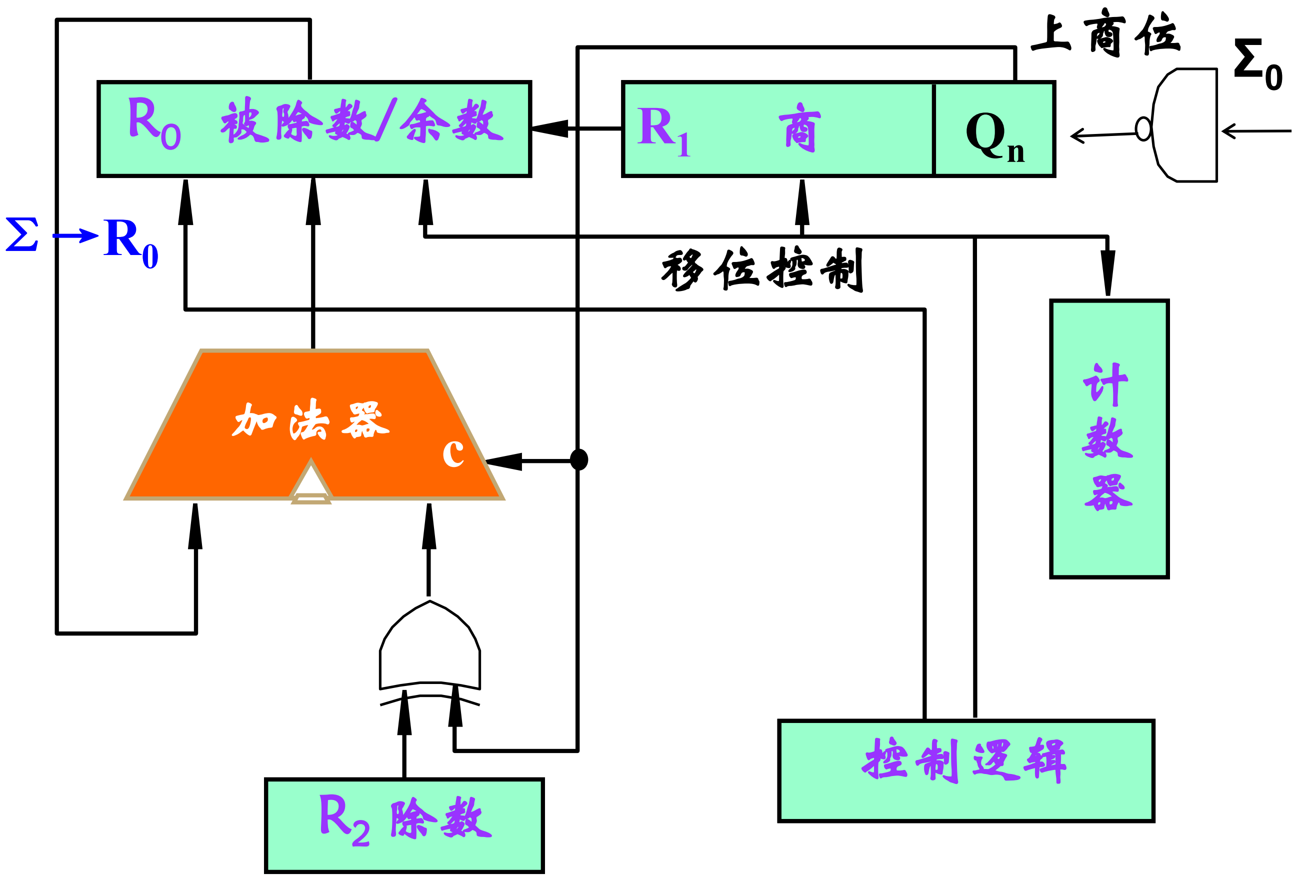
- **加减交替法**的规则是：当余数为正时，商“1”，余数左移一位，减除数；当余数为负时，商“0”，余数左移一位，加除数。

当余数为正时，商“1”，余数左移一位，减除数；
当余数为负时，商“0”，余数左移一位，加除数

被除数/余数	商	上商位	说明
00.1001			
$+[-Y]_{补}$ 11.0101			减Y比较
11.1110		0	$R_0<0$ 商上0
← 11.1100			左移一位 加Y比较
$+[Y]_{补}$ 00.1011			
00.0111		0.1	$R_1>0$，商上1
← 00.1110			左移一位，减Y比较
$+[-Y]_{补}$ 11.0101			
00.0011		0.11	$R_2>0$，商上1
← 00.0110			左移一位，减Y比较
$+[-Y]_{补}$ 11.0101			
11.1011		0.110	R3<0 商上0
← 11.0110			左移一位，加Y比较
$+[Y]_{补}$ 00.1011			
00. 0001		0.1101	$R_4>0$，商上1

$q=x\div y=0.1101$,余数 $r=2^{-4}xr_4$

原码不恢复余数除法逻辑结构（略/不要求）



例子：（略）

- 用原码不恢复余数法计算$[X]_{补}\div[Y]_{补}$。

  (1) $X = 0.10101$，$Y = 0.11011$

  解： $[-Y]_{补}=1.00101$

最后加1次Y，
$R=0.11000*2^{-5}$，
q=0.11000

X = 0.10101，Y = 0.11011， [-Y]补=1.00101

被除数/余数	商	上商位	说明
00.10101			
+[−Y]补 11.00101			减Y比较
11.11010		0	$R_0<0$ 商上0
11.10100			左移一位 加Y比较
+[Y]补 00.11011			
00.01111		0.1	$R_1>0$，商上1
00.11110			左移一位，减Y比较
+[−Y]补 11.00101			
00.00011		0.11	$R_2>0$，商上1
00.00110			左移一位，减Y比较
+[−Y]补 11.00101			
11.01011		0.110	$R3<0$ 商上0
10.10110			左移一位，加Y比较
+[Y]补 00.11011			
11. 10001		0.1100	$R_4<0$，商上0
11. 00010			左移一位，加Y比较
+[Y]补 00.11011			
11. 11101		0.11000	$R_5<0$ 商上0

补码一位除法（略/了解）

- 和补码加、减、乘法一样，补码除法也是符号位与数值位一起参加运算，商的符号位与数位由统一的算法求得。
- **补码加减交替法算法**

  在补码一位除法中也必须比较被除数（余数）和除数的大小，并根据比较的结果上商。另外，为了避免溢出，商的绝对值不能大于1，即被除数的绝对值一定要小于除数的绝对值。
- 补码加减交替法的**算法规则**如下：
  - **1） 求第一位商要判断两个数符号的同异，被除数与除数同号，被除数减除数；被除数与除数异号，被除数加除数。**
  - **2）余数左移一位，上商，余数与除数同号，商 1 ，下次减除数，求下一位商；余数与除数异号，商 0 ，下次加除数，求下位商;**
  - **3）重复步骤（2），包括符号位在内，共做n+1步。**
  - **4）修正余数**

并行除法器

可控制加/减法(CAS)单元

- 用于并行除法流水逻辑阵列中，它有四个输出端和四个输入端。当输入线P＝0时,CAS作加法运算；当P＝1时,CAS作减法运算。

$S_i=A_i \oplus (B_i \oplus P) \oplus C_i$

$C_{i+1}=(A_i+C_i)(B_i \oplus P)+A_iC_i$ (2.32)

当 P=0时，方程式（2.32）就是一位全加器（FA）的公式：

$$S_i=A_i \oplus B_i \oplus C_i$$

$$C_{i+1}=A_iB_i+B_iC_i+A_iC_i$$

当 P=1时。则得求差公式：

$S_i=A_i \oplus \overline{B}_i \oplus C_i$

$C_{i+1}=A_i\overline{B}_i +\overline{B}_iC_i + A_iC_i$ (2.41)

其中 $B_i=B_i \oplus 1$.

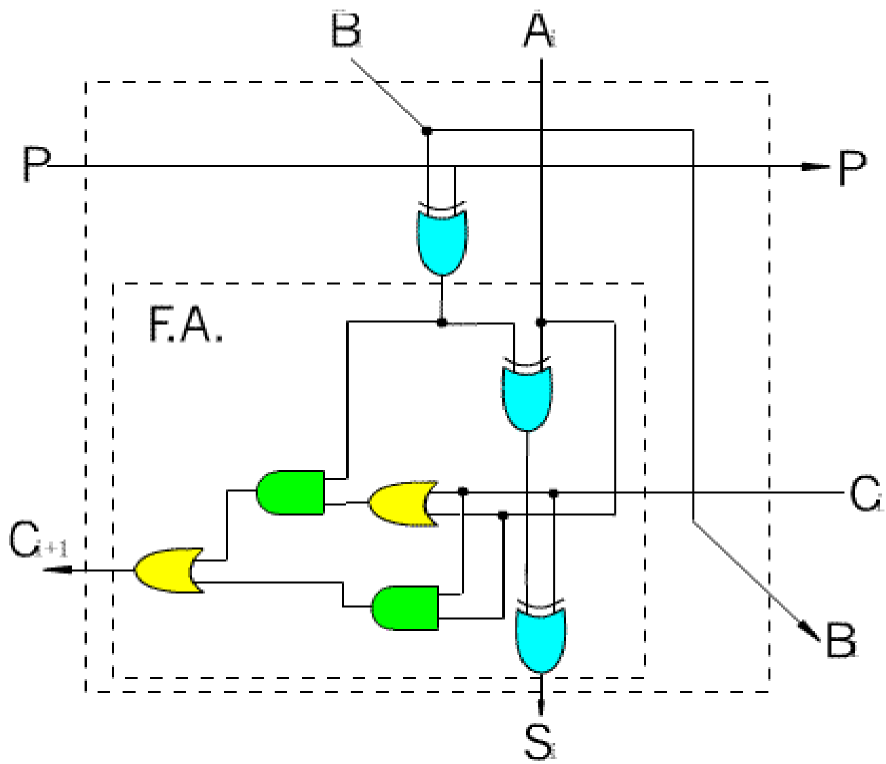


图2.9 不恢复余数阵列除法器逻辑结构图

并行除法器

方程式(2.32)加以变换,可得如下形式:

$$S_i = A_i \oplus (B_i \oplus P) \oplus C_i$$

$$= A_iB_i\bar{C}_iP + A_i\bar{B}_i\bar{C}_iP + \bar{A}_iB_iC_iP + A_iB_iC_i\bar{P} +$$

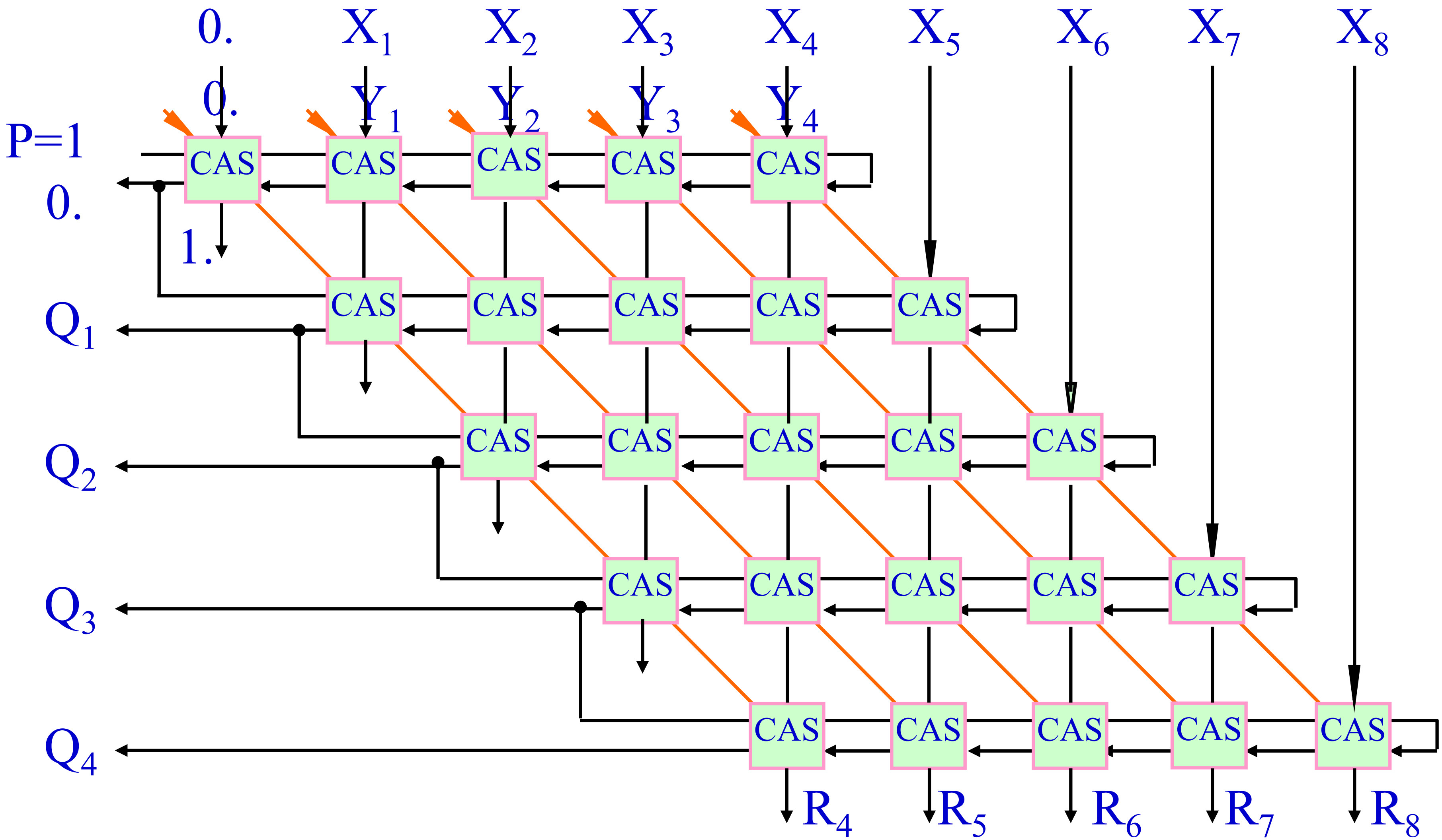
$$A_i\bar{B}_iC_iP + \bar{A}_i\bar{B}_i\bar{C}_iP + \bar{A}_iB_i\bar{C}_i\bar{P} + \bar{A}_i\bar{B}_iC_i\bar{P}$$

$$C_{i+1} = (A_i + C_i)(B_i \oplus P) + A_iC_i$$

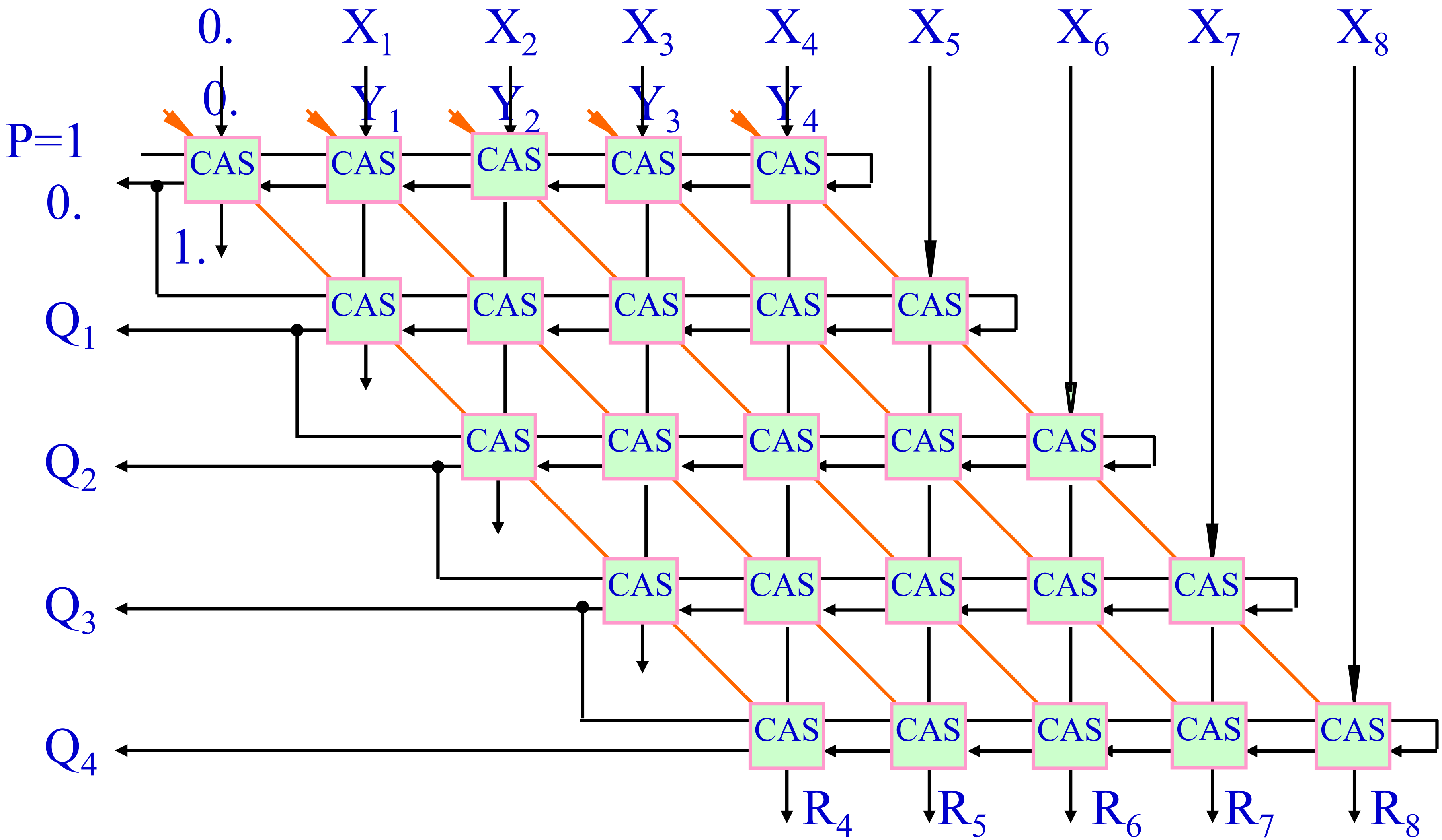
$$= A_iB_i\bar{P} + A_i\bar{B}_iP + B_iC_i\bar{P} + \bar{B}_iC_iP + A_iC_i$$

在这两个表达式中,每一个都能用一个三级组合逻辑电路(包括反向器)来实现。因此每一个基本的CAS单元的延迟时间为3T单元。

5位除5位原码阵列除法器



5位除5位原码阵列除法器



- 对不恢复余数阵列除法器来说,在进行运算时,沿着每一行都有进位(或借位)传播,同时所有行在它们的进位链上都是串行连接。而每个CAS单元的延迟时间为$3T$单元,因此,对一个2n位除以n位的不恢复余数阵列除法器来说,单元的数量为$(n+1)^2$,考虑最大情况下的信号延迟,其除法执行时间为,

$$t_d = 3(n+1)^2T \quad \textbf{(2.34)}$$

，其中n为尾数位数。

例 子

- [例20/23] x＝0.101001, y＝0.111, 求 $x \div y$。

[解:]　[－ y]补＝1.001

		0.1 0 1 0 0 1		
第一步做减法	$+[-y]_{补}$	1.0 0 1		
余数为负，商上0，下一步做加法		1.1 1 0 0 0 1	＜0	$q_0=0$
除数右移1位加，	$+2^{-1}[y]_{补}$	0.0 1 1 1		
余数为正，商上1，下一步做减法		0.0 0 1 1 0 1	＞0	$q_1=1$
除数右移2位加，	$+2^{-2}[-y]_{补}$	1.1 1 0 0 1		
余数为负，商上0，下一步做加法		1.1 1 1 1 1 1	＜0	$q_2=0$
除数右移1位加，	$+2^{-3}[y]_{补}$	0.0 0 0 1 1 1		
余数为正		0.0 0 0 1 1 0	＞0	$q_3=1$

故得，

商 $q=q_0.q_1q_2q_3=0.101$

余数 $r=(0.00r_3r_4r_5r_6)=0.000110$

例子另解（余数左移）：

- [例20] $x=0.101001$, $y=0.111$, 求 $x \div y$。

[解:]　$[-y]_补=1.001$

	0.1 0 1 0 0 1		
$+[-y]_补$	1.0 0 1		
余数为负	1.1 1 0 0 0 1	<0	$q_0=0$
余数左移	1.1 0 0 0 1		
$+[y]_补$	0.1 1 1		
余数为正	0.0 1 1 0 1	>0	$q_1=1$
余数左移	0.1 1 0 1		
$+[-y]_补$	1.0 0 1		
余数为负	1.1 1 1 1	<0	$q_2=0$
余数左移	1.1 1 1		
$+[y]_补$	0.1 1 1		
余数为正	0.1 1 0	>0	$q_3=1$

故得，

商 $q=q_0.q_1q_2q_3=0.101$

余数 $r=(0.00r_3r_4r_5r_6)=0.000110$

结论：余数左移等价于商右移。

例：x=0.1001，y=0.1011，用恢复余数法和不恢复余数法求x÷ y

解： $[y]_{补}$=00.1011

$[-y]_{补}$=11.0101

恢复余数法：

	被除数/余数	商	说明
	00.1001	00000	
$+[-y]_{补}$	11.0101		
	11.1110	00000	不够上0
$+[y]_{补}$	00.1011		恢复
	00.1001		
	01.0010	00000	左移
$+[-y]_{补}$	11.0101		
	00.0111	00001	够减上1
	00.1110	00010	左移
$+[-y]_{补}$	11.0101		
	00.0011	00011	够减上1
	00.0110	00110	左移
$+[-y]_{补}$	11.0101		
	11.1011	00110	不够上0
$+[y]_{补}$	00.1011		恢复
	00.0110		
	00.1100	01100	左移
$+[-y]_{补}$	11.0101		
	00.0001	01101	够减上1

不恢复余数法：

	被除数/余数	商	说明
	00.1001	00000	
$+[-y]_{补}$	11.0101		
	11.1110	00000	上0
	11.1100	00000	左移
$+[y]_{补}$	00.1011		
	00.0111	00001	上1
	00.1110	00010	左移
$+[-y]_{补}$	11.0101		
	00.0011	00011	上1
	00.0110	00110	左移
$+[-y]_{补}$	11.0101		
	11.1011	00110	上0
	11.0110	01100	左移
$+[y]_{补}$	00.1011		
	00.0001	01101	上1

所以：$[商]_{原}$=0.1101，$[余数]_{原}=0.0001\times 2^{-4}$

作 业

- 8（1），原码阵列除法（采用不恢复余数法计算）

主要内容

定点运算器的组成

- 由一位全加器(FA)构成的行波进位加法器,它可以实现补码数的加法运算和减法运算。
  - 存在不足:
  - 1）由于串行进位，运算时间很长。
  - 2）就行波进位加法器本身来说，它只能完成加法和减法两种操作而不能完成逻辑操作。
- 多功能算术/逻辑运算单元(ALU)，不仅具有多种算术运算和逻辑运算的功能，而且具有先行进位逻辑，从而能实现高速运算。

基本思想

- 一位全加器的逻辑表达式为

$$F_i = A_i \oplus B_i \oplus C_i$$

$$C_{i+1} = A_iB_i + B_iC_i + C_iA_i$$

- 将A_i和B_i先组合成由控制参数S_0, S_1, S_2, S_3控制的组合函数X_i和Y_i，然后再将X_i, Y_i和下一位进位数C_i通过全加器进行全加。这样，不同的控制参数可以得到不同的组合函数，因而能够实现多种算术运算和逻辑运算。

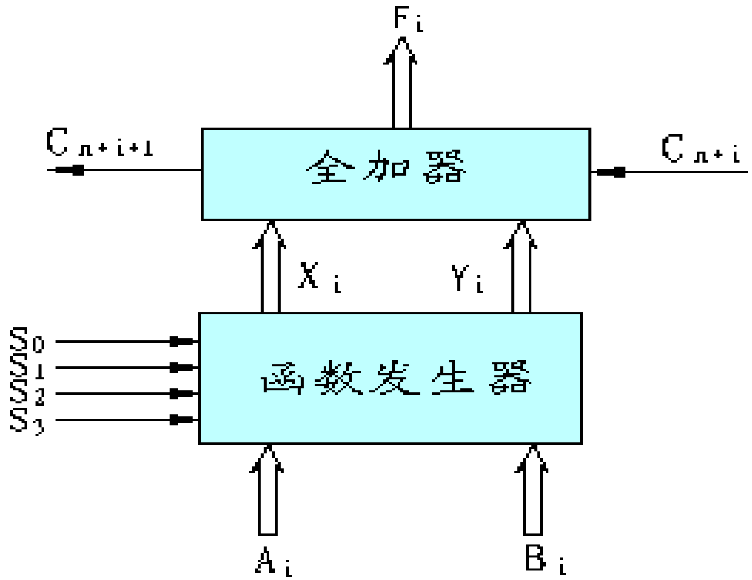


图2.10　ALU的逻辑结构原理框图

- 一位算术/逻辑运算单元的逻辑表达式为

  $$F_i = X_i \oplus Y_i \oplus C_{n+i}$$

  $$C_{n+i+1} = X_iY_i + Y_iC_{n+i} + C_{n+i}X_i$$

- 上式中进位下标用$\mathbf{n+}\boldsymbol{i}$代替原来一位全加器中的$i$。$i$代表集成在一片电路上的ALU的二进制位数。对于4位一片的ALU，i＝0,1,2,3。n代表若干片ALU组成更大字长的运算器时每片电路的进位输入，例如当4片组成16位字长的运算器时，n＝0,4,8,12。

□逻辑表达式

控制参数S_0 ,S_1 控制输入A_i ,产生Y；S_2 ,S_3 控制输入B_i ,产生X。其中Yi是受S_0 ,S_1控制的A_i和B_i的组合函数,而X_i是受S_2 ,S_3控制的A_i和B_i组合函数,其函数关系如表2.4所示。

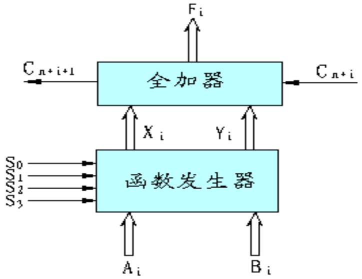


表2.4 X_i,Y_i与控制参数和输入量的关系

S_0 S_1	Y_i	S_2 S_3	X_i
0 0	$\bar{A}_i$	0 0	1
0 1	$\bar{A}_iB_i$	0 1	$\bar{A}_i+\bar{B}_i$
1 0	$\bar{A}_i\bar{B}_i$	1 0	$\bar{A}_i+B_i$
1 1	0	1 1	$\bar{A}_i$

根据上面所列的函数关系,即可列出X_i和Y_i的逻辑表达式

$$X_i=\bar{S}_2\bar{S}_3+\bar{S}_2S_3（\bar{A}_i+\bar{B}_i）+S_2\bar{S}_3（\bar{A}_i+B_i）+S_2S_3\bar{A}_i$$

$$Y_i=\bar{S}_0\bar{S}_1\bar{A}_i+\bar{S}_0S_1\bar{A}_iB_i+S_0\bar{S}_1\bar{A}_i\bar{B}_i$$

- 化简并代入前面的求和与进位表达式，可得ALU的某一位逻辑表达式如

$X_i=S_3A_iB_i+S_2A_i\overline{B_i}$　　　　$Y_i=A_i+S_0B_i+S_1\overline{B_i}$

$F_i=Y_i\oplus X_i\oplus C_{n+i}$　　　　$C_{n+i+1}=Y_i+X_iC_{n+i}$　（2.36）

4位之间采用先行进位公式,每一位的进位公式可递推如下：

第0位向第1位的进位公式为

$C_{n+1}=Y_0+X_0C_n$　　其中C_n是向第0位（末位）的进位。

第1位向第2位的进位公式为

$C_{n+2}=Y_1+X_1C_{n+1}=Y_1+Y_0X_1+X_0X_1C_n$

第2位向第3位的进位公式为

$C_{n+3}=Y_2+X_2C_{n+2}=Y_2+Y_1X_1+Y_0X_1X_2+X_0X_1X_2C_n$

第3位的进位输出（即整个4位运算进位输出）公式为

$C_{n+4}=Y_3+X_3C_{n+3}=Y_3+Y_2X_3+Y_1X_2X_3+Y_0X_1X_2X_3+X_0X_1X_2X_3C_n$

设　$G=Y_3+Y_2X_3+Y_1X_2X_3+Y_0X_1X_2X_3$

　　$P=X_0X_1X_2X_3$

则　$C_{n+4}=G+PC_n$

优点：第0位的进位输入C_n可以直接传送到最高位上去,因而可以实现高速运算。

$X_i = \overline{S_3 A_i B_i + S_2 A_i \overline{B_i}}$ $\quad$ $Y_i = \overline{A_i + S_0 B_i + S_1 \overline{B_i}}$

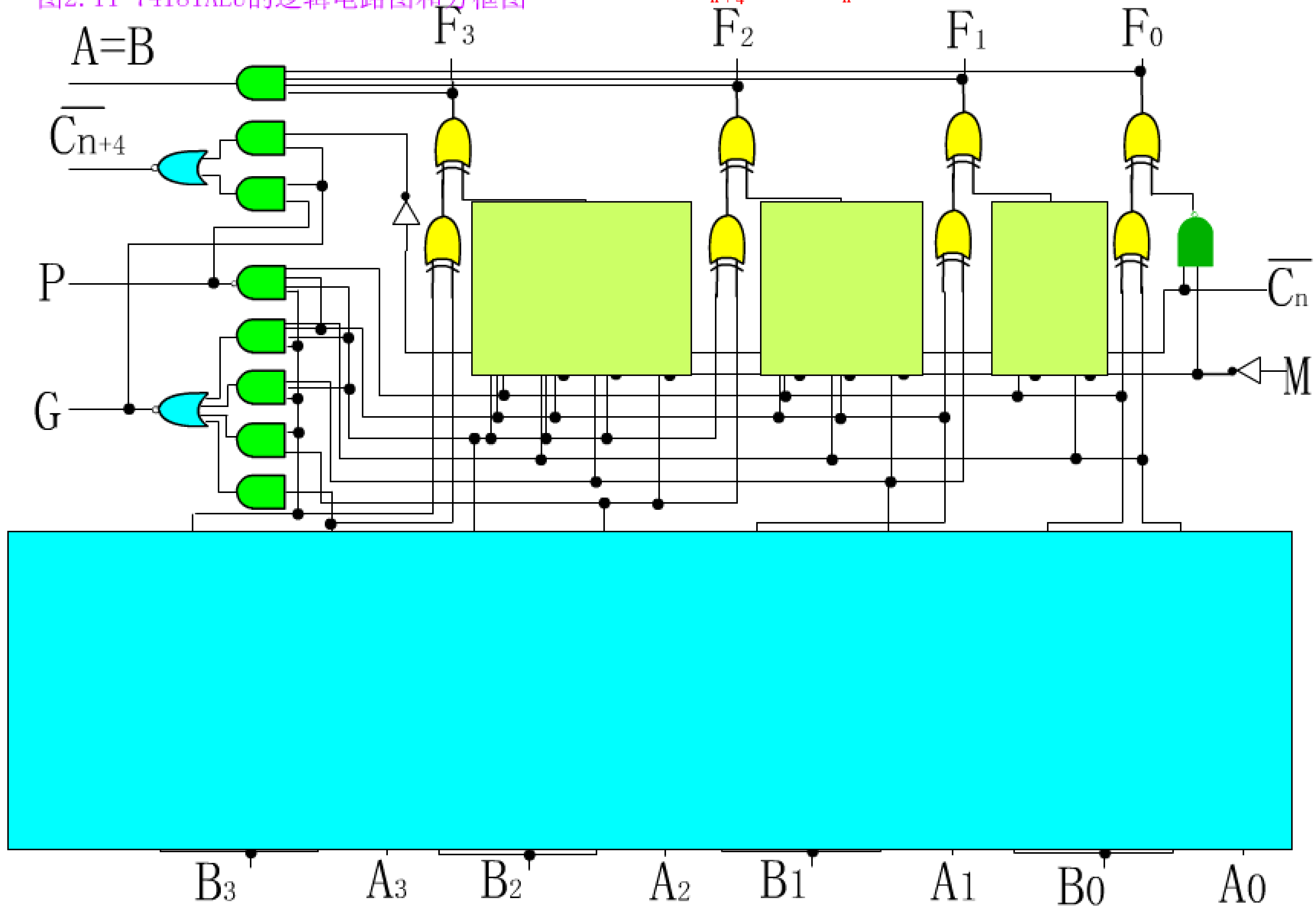
$F_i = Y_i \oplus X_i \oplus C_{n+i}$ $\quad$ $C_{n+i+1} = Y_i + X_i C_{n+i}$

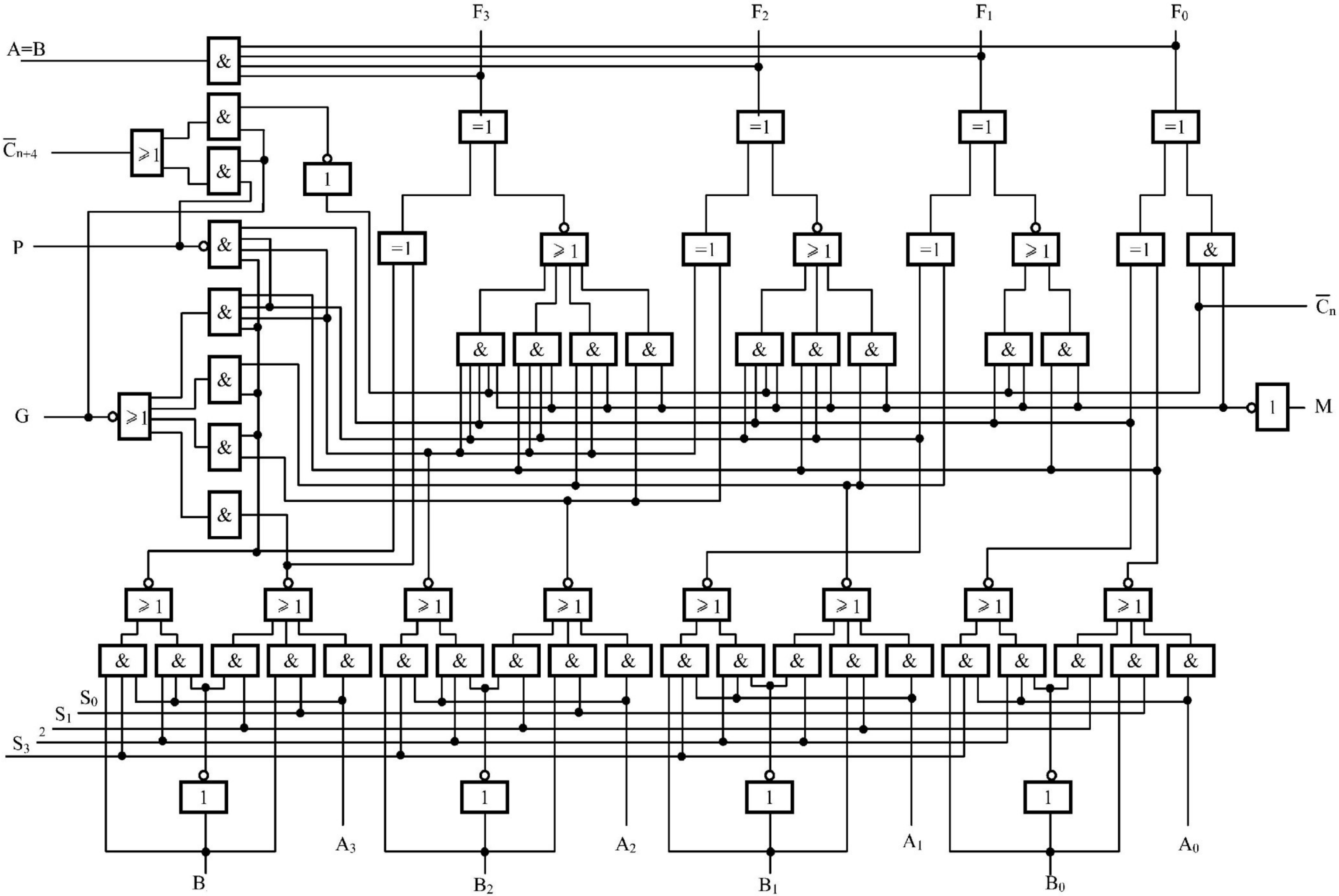
$G = Y_3 + Y_2 X_3 + Y_1 X_2 X_3 + Y_0 X_1 X_2 X_3$

$P = X_0 X_1 X_2 X_3$

$C_{n+4} = G + P C_n$

图2.11 74181ALU的逻辑电路图和方框图





SN74181 逻辑电路

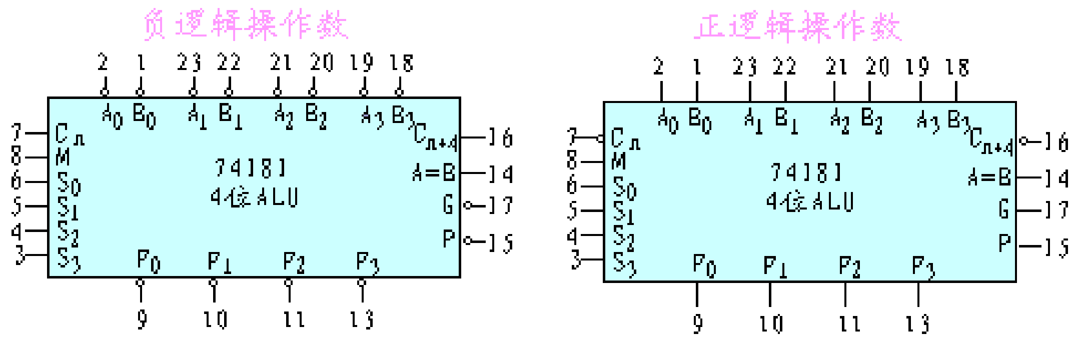
- 用正逻辑表示的4位算术/逻辑运算单元(ALU)
- **算术逻辑运算的实现**

以上演示图中除了S_0－S_3四个控制端外,还有一个控制端M,它使用来控制ALU是进行算术运算还是进行逻辑运算的。

当M＝0时,M对进位信号没有任何影响。此时F 不仅与本位的被操作数Y和操作数X 有关,而且与本位的进位输出,即C 有关,因此 *M*＝0时,进行**算术操作**。

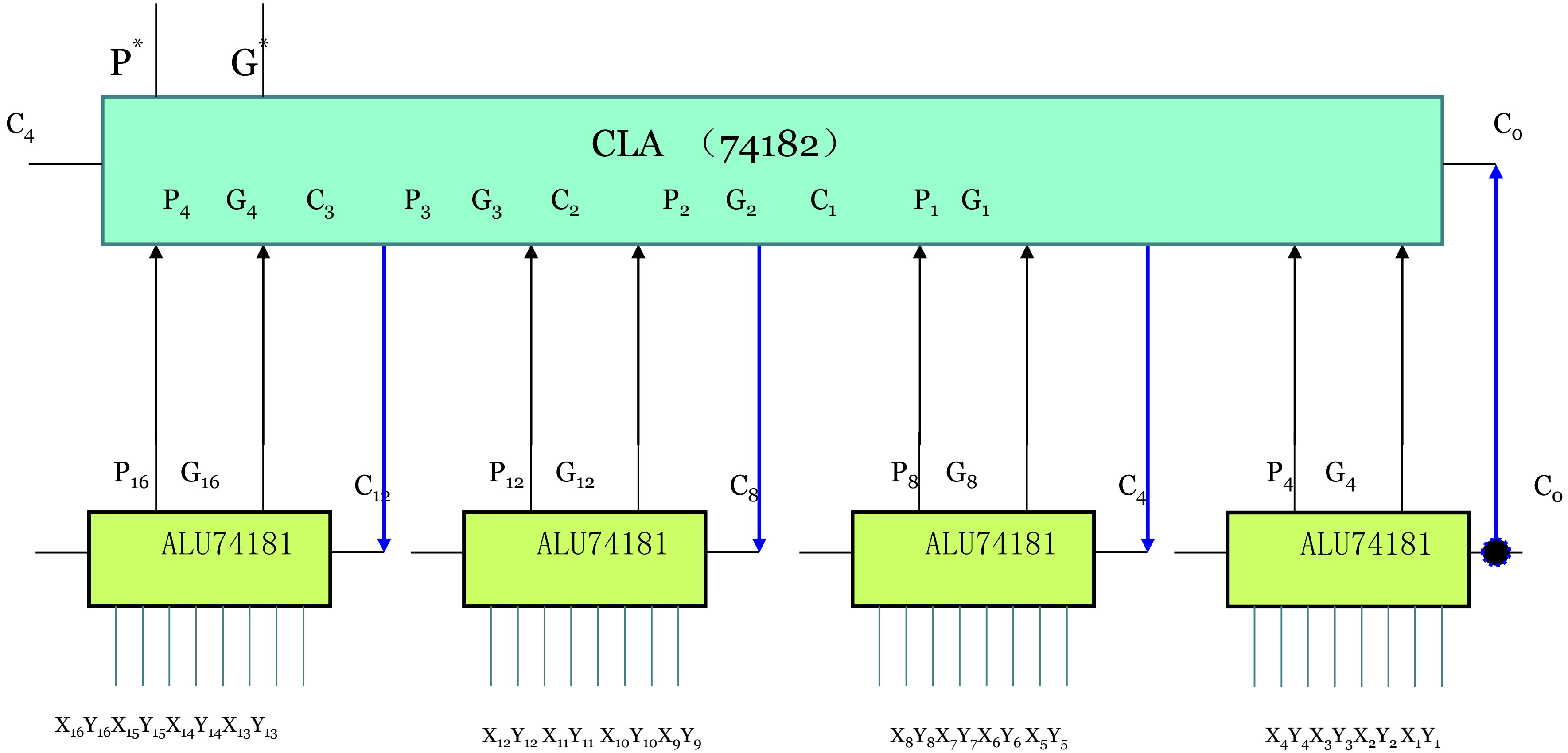
当M＝1时,封锁了各位的进位输出,即C ＝0,因此各位的运算结果F 仅与Y 和X 有关,故M＝1时,进行**逻辑操作**。

图2.11(b)是负逻辑和正逻辑操作数方式的74181ALU方框图。器件执行的正逻辑输入/输出方式的一组算术运算和逻辑操作与负逻辑输入/输出方式的一组算术运算和逻辑操作是等效的。



(b) 负逻辑或正逻辑操作数方式的74181ALU方框图

- 用4片74181电路可以组成16位的ALU，片内先行进位快速处理，片间进位还是逐片串行传递。
- 74182是16位组内先行进位，组间也先行进位。



两级先行进位的ALU逻辑公式(了解)

74181ALU设置了P和G两个本组先行进位输出端。如果将四片74181的P,G输出端送入到74182先行进位部件（CLA）,又可实现第二级的先行进位,即组与组之间的先行进位。

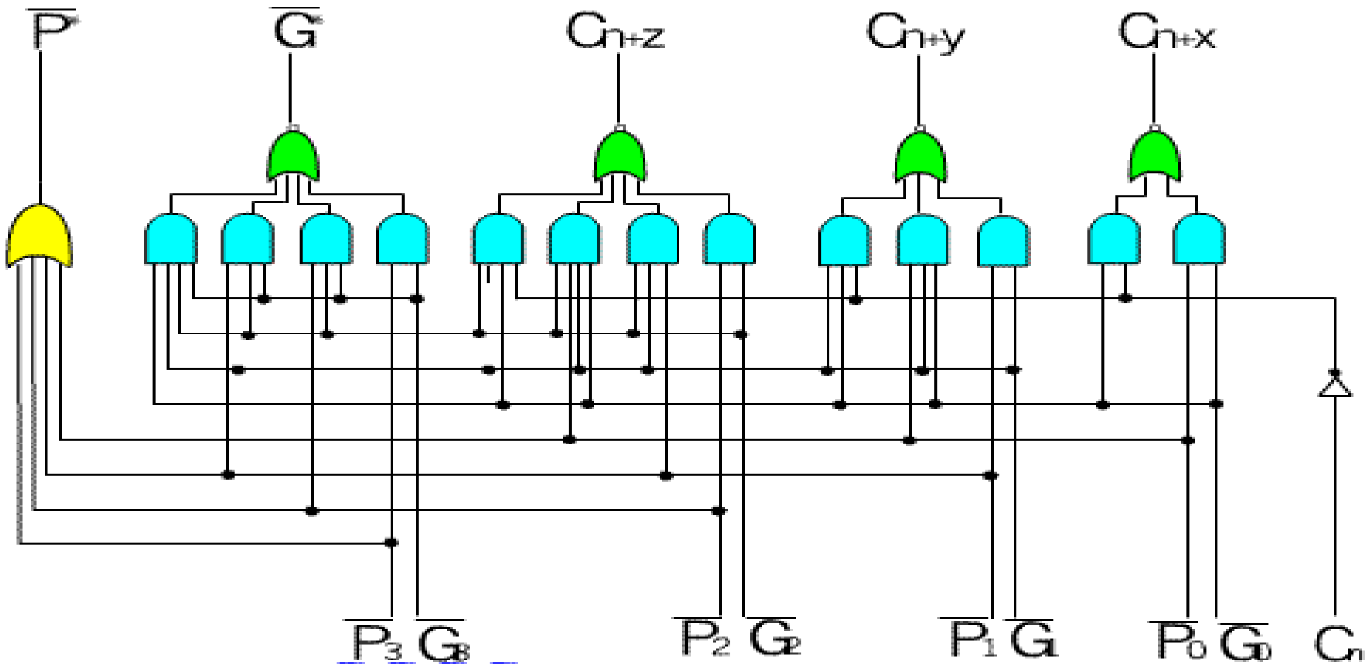
假设4片（组）74181的先行进位输出依次为$P_0,G_0,G_1P_1,P_2,G_2,P_3,G_3$,那么参考式(2.37)的进位逻辑表达式,先行进位部件74182CLA所提供的进位逻辑关系如下：

$$C_{n+x}=G_0+P_0C_n$$
$$C_{n+y}=G_1+P_1C_{n+x}=G_1+G_0P_1+P_0P_1C_n$$
$$C_{n+z}=G_2+P_2C_{n+y}=G_2+G_1P_2+G_0P_1P_2+P_0P_1P_2C_n \qquad \mathbf{(2.38)}$$
$$C_{n+4}=G_3+P_3C_{n+z}=G_3+G_2P_3+G_1P_1P_2+G_0P_1P_2P_3+P_0P_1P_2P_3C_n$$
$$=G^*+P^*C_n$$

其中 $P^*=P_0P_1P_2P_3$

$$G^*=G_3+G_2P_3+G_1P_1P_2+G_0P_1P_2P_3$$

根据以上表达式,用TTL器件实现的成组先行进位部件74182的逻辑电路图如图所示其中G*称为**成组进位发生输出**,P*称为**成组进位传送输出**。



$\overline{P^*} = \overline{P_0P_1P_2P_3}$

$\overline{G^*} = \overline{G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0}$

图2.12 成组先行进位部件CLA的逻辑电路图

$C_{n+x} = G_0 + P_0C_n$

$C_{n+y} = G_1 + P_1C_{n+x} = G_1 + G_0P_1 + P_0\mathrm{P1}C_n$

$C_{n+z} = G_2 + P_2C_{n+y} = G_2 + G_1P_2 + G_0P_1P_2 + P_0P_1P_2C_n$

$C_{n+4} = G_3 + P_3C_{n+z} = G_3 + G_2P_3 + G_1P_1P_2 + G_0P_1P_2P_3 + P_0P_1P_2P_3C_n$

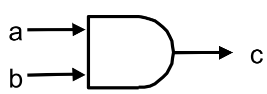
2.5.3 内部总线

- 由于计算机内部的主要工作过程是信息传送和处理的过程,机器内部各部件之间的数据传送非常频繁。
- 为了减少内部的传送线并便于控制,通常将一些寄存器之间数据传送的通路加以归并,组成总线结构,使不同来源的信息在此传输线上分时传送。
- 根据总线所在位置,总线分为内部总线和外部总线两类。**内部总线**是指CPU内各部件的连线,而**外部总线**是指系统总线,又分为存储总线和I/O总线,即CPU与存储器、I/O系统之间的连线。
- 按总线的逻辑结构来说,总线可分为单向传送总线和双向传送总线。所谓**单向总线**,就是信息只能向一个方向传送。所谓**双向总线**,就是信息可以分两个方向传送,既可以发送数据,也可以接收数据。

2.5.4 定点运算器的基本结构

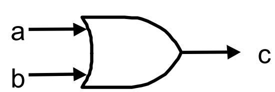
- 运算器包括ALU\阵列乘除器\寄存器\多路开关\三态缓冲器\数据总线等逻辑部件。

1. AND gate (c = a . b)



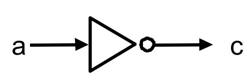
a	b	c = a . b
0	0	0
0	1	0
1	0	0
1	1	1

2. OR gate (c = a + b)



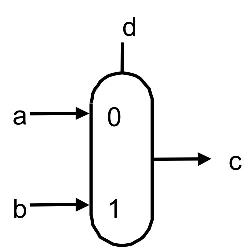
a	b	c = a + b
0	0	0
0	1	1
1	0	1
1	1	1

3. Inverter (c = a)



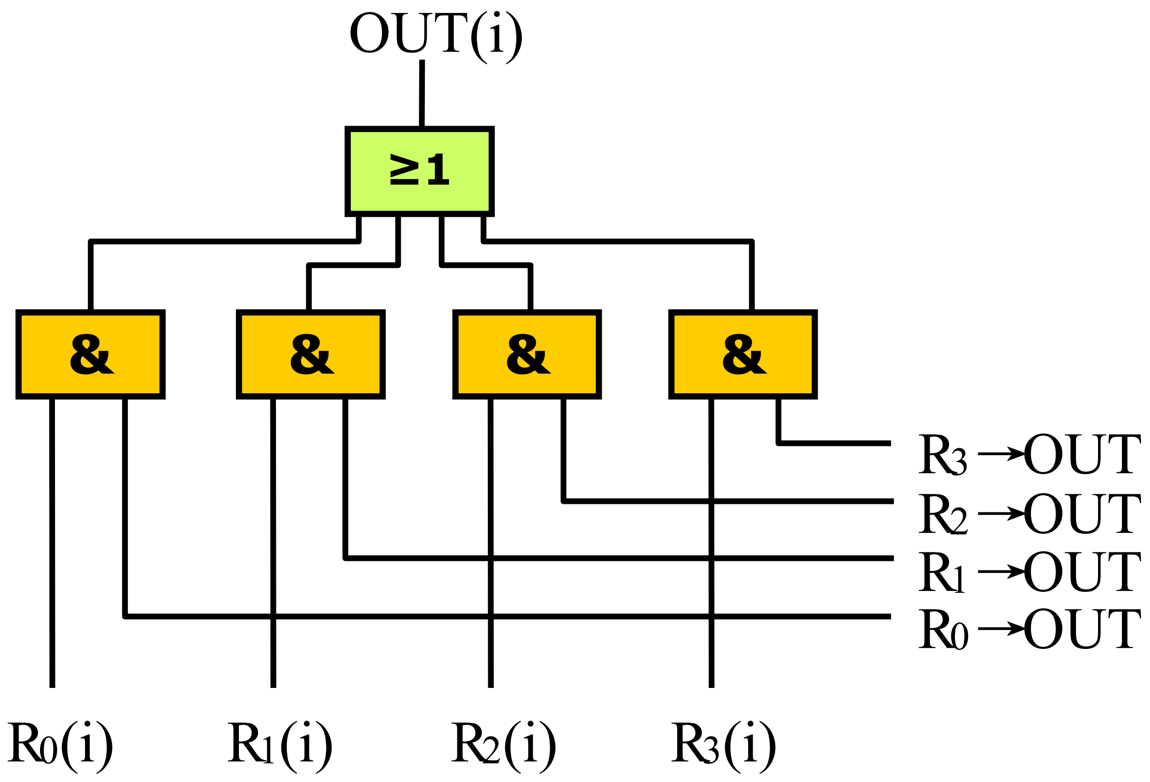
a	$c = \bar{a}$
0	1
1	0

4. Multiplexor
   (if d = = 0, c = a;
   else c = b)

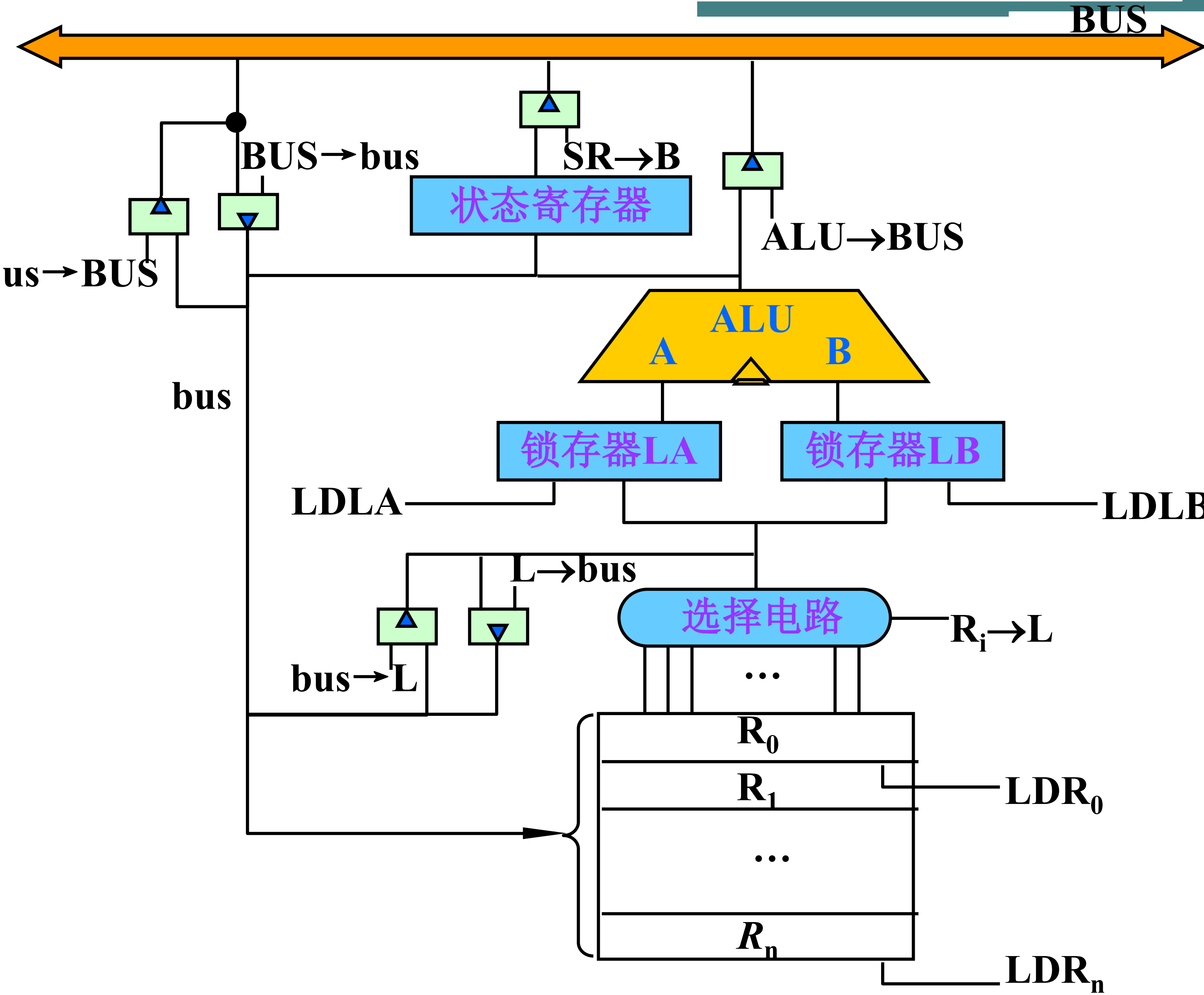


d	c
0	a
1	b

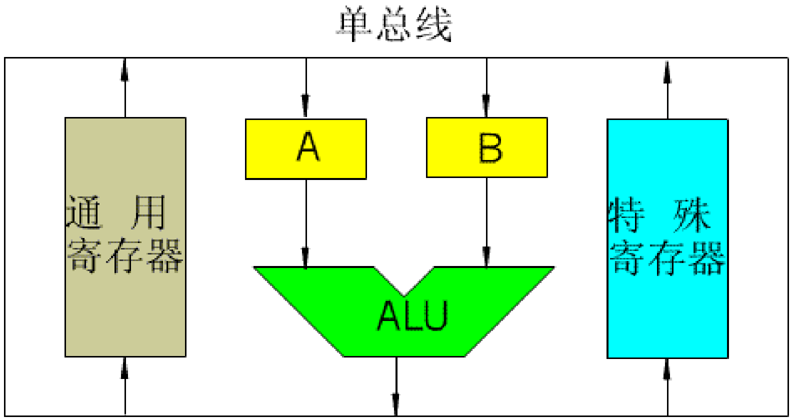
多路选择电路



单总线结构的运算器


BUS
BUS→bus
SR→B
状态寄存器
bus→BUS
ALU→BUS
ALU
A
B
bus
锁存器LA
锁存器LB
LDLA
LDLB
L→bus
选择电路
Ri→L
bus→L
…
R0
R1
…
Rn
LDR0
LDRn

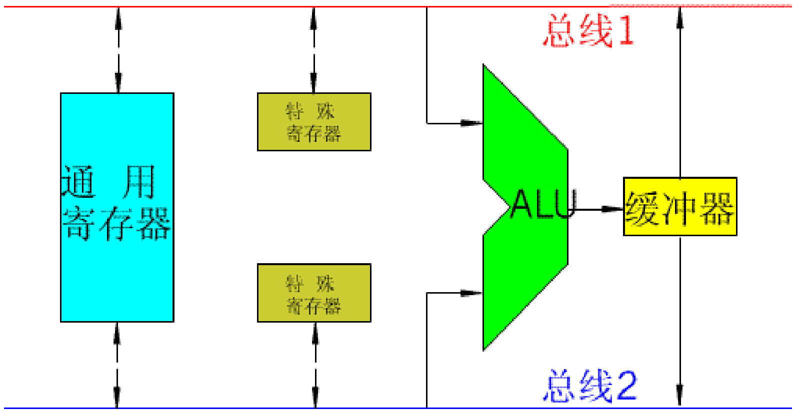
单总线结构的运算器



（a）单总线结构的运算器

- 由于所有部件都接到同一总线上,所以数据可以在任何两个寄存器之间,或者在任一个寄存器和ALU之间传送。在同一时间内,只能有一个操作数放在单总线上。为了把两个操作数输入到ALU,需要分两次来做,而且还需要A,B两个缓冲寄存器。
- 主要特点是控制电路比较简单，操作速度较慢。

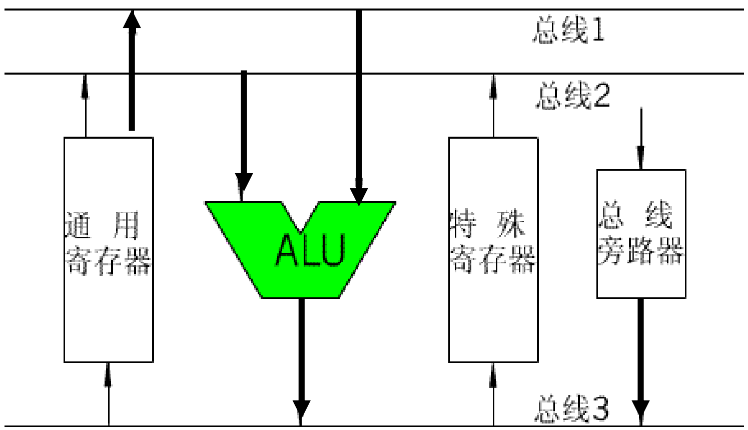
双总线结构的运算器



（b）双总线结构的运算器

- 控制要分两步完成:
  1.在ALU的两个输入端输入操作数,结果送入缓冲寄存器;
  2.把结果送入目的寄存器。

三总线结构的运算器



（c）三总线结构的运算器

- 三总线结构中,ALU的两个输入端分别由两条总线供给,而ALU的输出则与第三条总线相连。

主 要 内 容

- 2.1 数据与文字的表示方法
- 2.2 定点加法、减法运算
- 2.3 定点乘法运算
- 2.4 定点除法运算
- 2.5 定点运算器的组成
- 2.6 浮点运算和浮点运算器

浮点加减法运算

- 设有两个浮点数 x 和 y,它们分别为

$$x = 2^{E_x} \cdot M_x$$
$$y = 2^{E_y} \cdot M_y$$

其中E_x和E_y分别为数 x 和 y 的阶码,M_x和M_y为数 x 和 y 的尾数。

- 计算X+Y=?

求解:

如: $E_x=E_y$ $\quad S=2^{E_x} \times (M_x+M_y)$

如: $E_x \neq E_y$ $\quad$?????

对 阶

- 对阶(使得小数部分可以按位权值按位相加)
- 大阶对小阶还是小阶对大阶？ ？ ？

$2^{10}*(0.11000)+2^{8}*(0.00110)$

大阶对小阶 $2^{10}*(0.11000) \dashrightarrow 2^{8}*(11.000)$

11.000+0.00110 ?????????

小阶对大阶 $2^{8}*(0.00110) \dashrightarrow 2^{10}*(0.00001)$

0.00001+0.11000=0.11001（尾数运算）

- 对阶过程应该是小阶对大阶，尾数右移

运算结果规格化

- $2^{10}*(0.11000)$ 也可表示为 $2^{11}*(0.01100)$
- 同一个浮点数的编码唯一，为提高精度，尾数不为零的时，要求其绝对值大于1/2，即尾数最高有效位为1，否则要以修改阶码的方式同时左右移小数点，使其变成这一要求的表示形式，这个过程称为浮点数的规格化。
- 将运算结果右移以实现规格化表示称为向右规格化，将运算结果左移以实现规格化表示称为向左规格化。
- Tips：绝对值大于1,向左破坏了规格化。此时将运算结果右移以实现规格化表示,称为**向右规格化**。当尾数不是1.M时需**向左规格化**。

规格化形式

- 规格化数形式 0.1XXXX -0.1XXXX
- 补码规格化形式 00.1XXXX 11.0XXXX

符号位和小数点后的第一位不等，则是规格化数

- 补码非规格化数 00.0XXXX 11.1XXXX

符号位和小数点后的第一位相等，则向左规格化

- 补码非规格化数 01.XXXXX 10.XXXXX

（两个符号位不等，称溢出，表明尾数求和绝对值大于1，向左破坏了规格化，将尾数运算结果右移，向右规格化。）

规格化规则小结

- 运算结果产生溢出时，必须进行右归
  - 如变形补码结果出现10.XX或者01.XXX
- 如运算结果出现0.0XXX或1.1XX必须左归
- 左归时最低数据有效位补0
- 右归时连同符号位一起右移
- 左归时，阶码作减法，右归时，阶码作加法

舍入处理

0	1	1	0	0	0	1	1

0	0	1	1	0	0	0	1	1

右规后低位部分丢失了一位，这会对数产生一定的误差。

0舍1入法

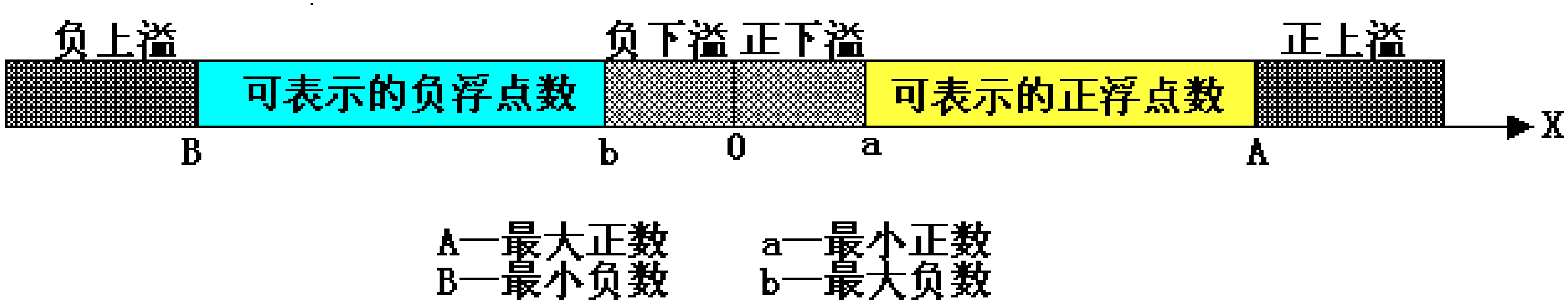
如被丢的最高数位为0，舍去，如为1，则将尾数末位加一。

截去法

"恒置1"法

溢出处理

- 尾数上溢　右归
- 尾数下溢　右归，尾数的最低有效位从尾数域右端流出,要进行舍入处理。
- 阶码正上溢　|X|→∞
- 阶码负上溢　|X|→0



浮点数加减法五个基本步骤

两浮点数进行加法和减法的运算规则是

$$x \pm y = (M_x 2^{E_x - E_y} \pm M_y) 2^{E_y}, \quad E_x <= E_y \quad (2.39)$$

- 对阶
- 尾数求和
- 规格化（左规，右规）
- 舍入（截去、0舍1入）
- 检查溢出

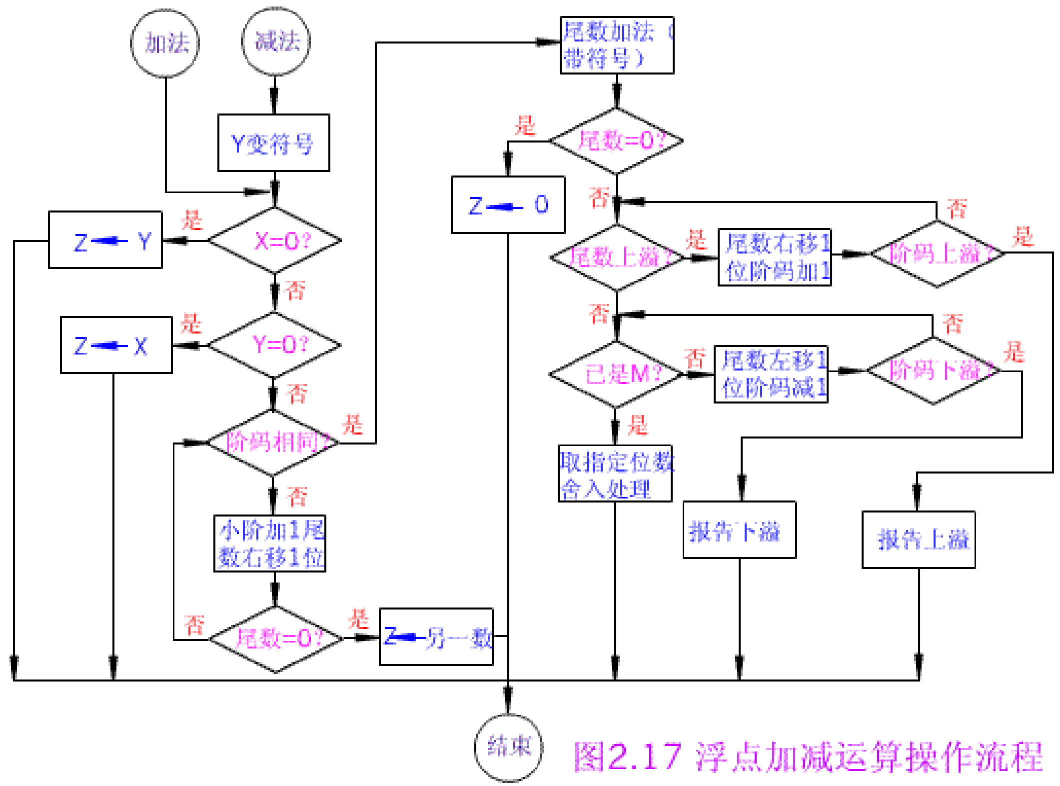


图2.17 浮点加减运算操作流程

例 子28//25

设 $x=2^{010}\times 0.11011011$, $y=2^{100}\times(-0.10101100)$,求 $x+y$。

[解:]将x,y转换成浮点数据格式

$[x]_{浮}=00\ 010,\quad 00.11011011$

$[y]_{浮}=00\ 100,\quad 11.01010100$

<1> 求阶差并对阶

$\triangle E=E_x-E_y=[E_x]_{补}+[-E_y]_{补}=00\ 010+11\ 100=11\ 110$

即$\triangle E$为-2, x的阶码小,应使M_x右移两位,E_x加2,

$[x]_{浮}=00\ 100,\quad 00.00110110(11)$

其中(11)表示M_x右移2位后移出的最低两位数。

<2> 尾数求和

$$\begin{array}{rl} & 00.00110110\,(11) \\ + & 11.01010100 \\ \hline & 11.10001010\,(11) \end{array}$$

例续

11. 1 0 0 0 1 0 1 0 (11)

<3>规格化处理

尾数运算结果的符号位与最高数值位同值,应执行左规处理,结果为1.00010101(10),阶码为 00 011。

<4>舍入处理

采用0舍1入法处理,则有

$$
\begin{array}{r}
11.00010101 \\
+\qquad\qquad 1 \\
\hline
11.00010110
\end{array}
$$

<5>判溢出

阶码符号位为00,不溢出,故得最终结果为

$$x + y = 2^{011} \times (-0.11101010)$$

浮点数乘法运算

- 设有两个浮点数 x 和 y： $x = 2^{E_x} \cdot M_x$

  $y = 2^{E_y} \cdot M_y$

浮点乘法运算的规则是

$$x \times y = 2^{(E_x + E_y)} \cdot (M_x \times M_y) \qquad \textbf{(2.40)}$$

浮点数乘法运算

- (1) 阶码相加

  阶码相加可能产生溢出，若产生溢出，则给出溢出指示，计算机进行溢出处理。

- (2) 尾数相乘

  尾数部分相乘可得积的尾数，尾数相乘可按定点乘法运算的方法进行运算。

- (3) 结果规格化

  当运算结果需要进行规格化操作时，可按浮点加/减法运算规格化方式处理，舍入方式也与加/减法方式中的相同。

浮点数乘法运算

(1) 浮点数的阶码运算

移码的定义为

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

按此定义,则有

$$\begin{aligned}[x]_{移}+[y]_{移}&=2^n+x+2^n+y\\&=2^n+(2^n+(x+y))\\&=2^n+[x+y]_{移}\end{aligned}$$

- 即直接用移码实现求阶码之和时,结果的最高位多加了个1,要得到正确的移码形式结果,必须对结果的符号再执行一次求反。

浮点数乘法运算

- 混合使用移码和补码时,考虑到移码和补码的关系:对同一个数值,其数值位完全相同,而符号位正好完全相反。而$[y]_补$的定义为$[y]_补=2^{n+1}+y$,

则求阶码和:$[x]_移+[y]_补=2^n+x+2^{n+1}+y$
$=2^{n+1}+(2^n+(x+y))$

即,$[x+y]_移=[x]_移+[y]_补 \pmod{2^{n+1}}$ **(2.42)**

同理$[x-y]_移=[x]_移+[-y]_补$ **(2.43)**

- 表明执行阶码加减时,对加数或减数 y 来说,应送移码符号位正常值的反码。
- 如果阶码运算的结果溢出,上述条件则不成立。

浮点数乘法运算

- (2) 尾数处理
- 浮点加减法对结果的规格化及舍入处理也适用于浮点乘除法。**截断法**，**舍入法**，**恒置1法**。
- 舍入法规则：

当丢失的各位均为0时,不必舍入；

当丢失的最高位为0 时,以下各位不全为0 时,或者丢失的最高位为1,以下各位均为0时,则舍去丢失位上的值；

当丢失的最高位为1,以下各位不全为0 时,则执行在尾数最低位入1的修正操作。

例 子28(3rd) - 略

- 设有浮点数 $x=2^{-5}\times 0.0110011$, $y=2^{3}\times(-0.1110010)$，阶码用4位移码表示，尾数（含符号位）用8位补码表示。求$[x\times y]_{浮}$。要求用补码完成尾数乘法运算，运算结果尾数保留高8位（含符号位），并用尾数低位字长值处理舍入操作。[解:]移码采用双符号位，尾数补码采用单符号位，则有

$$[M_x]_{补}=0.0110011,\quad [M_y]_{补}=1.0001110,$$

$$[E_x]_{移}=00\ 011,\ [E_y]_{移}=01\ 011,\quad [E_y]_{补}=00\ 011,$$

$$[x]_{浮}=00\ 011,\ 0.0110011,\quad [y]_{浮}=01\ 011,\ 1.0001110$$

(1) 求阶码和 $[E_x+E_y]_{移}=[E_x]_{移}+[E_y]_{补}=00\ 011+00\ 011=00\ 110$，值为移码形式$-2$。

(2) 尾数乘法运算可采用补码阵列乘法器实现，即有

$$[M_x]_{补}\times[M_y]_{补}=[0.0110011]_{补}\times[1.0001110]_{补}$$
$$=[1.1010010,1001010]_{补}$$

(3) 规格化处理 乘积的尾数符号位与最高数值位符号相同，不是规格化的数，需要左规，阶码变为00 101(-3)，尾数变为 1.0100101,0010100。

(4) 舍入处理 尾数为负数，取尾数高位字长，按舍入规则，舍去低位字长，故尾数为1.0100101 。

最终相乘结果为 $[x\times y]_{浮}=00\ 101,1.0100101$

其真值为 $x\times y=2^{-3}\times(-0.1011011)$

例 子30(4nd)

- 设有浮点数 $x=2^{-5}\times0.0110011$, $y=2^{3}\times(-0.1110010)$, 阶码用4位补码表示, 尾数（含符号位)用8位原码表示。求$[x\times y]_{浮}$。要求用原码完成尾数乘法运算, 运算结果尾数保留高8位(含符号位), 并用尾数低位字长值处理舍入操作。[解:]阶码采用双符号位, 尾数原码采用单符号位, 则有

$$[M_x]_{原}=0.0110011,\quad [M_y]_{原}=1.1110010,$$

$$[E_x]_{补}=11\ 011, [E_y]_{补}=00\ 011,$$

$$[x]_{浮}=11\ 011,\ 0.0110011,\quad [y]_{浮}=00\ 011,\ 1.\ 1110010$$

(1) 求阶码和 $[E_x+E_y]_{补}=[E_x]_{补}+[E_y]_{补}=11\ 011+00\ 011=11\ 110$, 值为补码形式$-2$。

(2) 尾数乘法运算可采用原码阵列乘法器实现, 即有

$$[M_x]_{原}\times[M_y]_{原}=[0.0110011]_{原}\times[1.1110010]_{原}$$
$$=[1.0101101,0110110]_{原}$$

(3) 规格化处理 乘积不是规格化的数, 需要左规, 阶码变为11 101(-3), 尾数变为 1.1011010,1101100。

(4) 舍入处理 尾数为负数, 取尾数高位字长，按舍入规则, 舍去低位字长, 故尾数为1.1011011 。

最终相乘结果为 $[x\times y]_{浮}=11\ 101,1.1011011$

其真值为 $x\times y=2^{-3}\times(-0.1011011)$

浮点数除法运算

- $x \div y = 2^{(E_x - E_y)} \cdot (M_x \div M_y)$ **(2.41)**

1.尾数调整

- 如果被除数尾数大于除数的尾数(从绝对值考虑)，则将被除数尾数右移一位并相应调整阶码，由于给出运算的操作数是规格化数，一般只作一次调整便可达到要求。

2.阶码求差

- 商的阶码等于被除数的阶码减去除数的阶码，此步运算基本上获得了商的阶码。

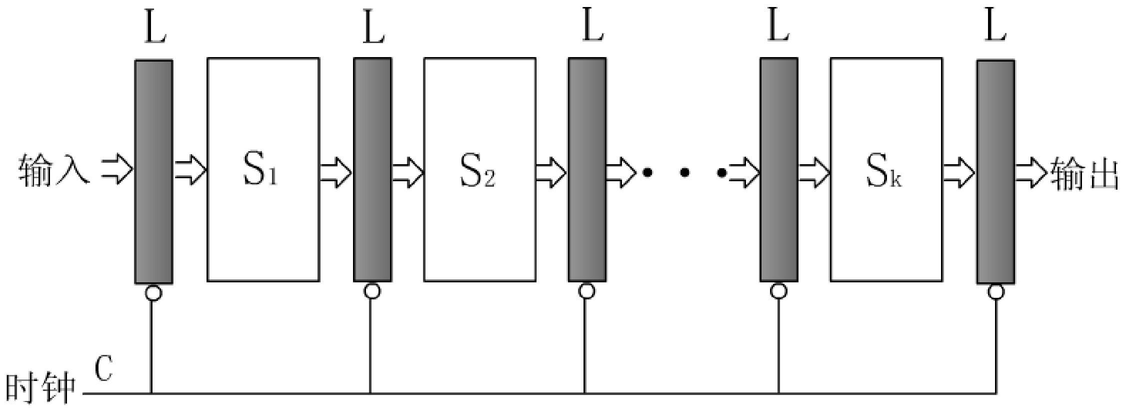
3.尾数相除

- 以被乘数的尾数除以除数的尾数以获得商的尾数，尾数相除与定点除法运算相同。

2.6.3 浮点运算流水线

- 1、提高并行性的两个渠道：
  - 空间并行性：增加冗余部件，如增加多操作部件处理机和超标量处理机
  - 时间并行性：改善操作流程如：流水线技术
- 2流水线特点：
  - 在流水线中必须是连续的任务，只有不断的提供任务才能充分发挥流水线的效率
  - 把一个任务分解为几个有联系的子任务。每个子任务由一个专门的功能部件实现
  - 在流水线中的每个功能部件之后都要有一个缓冲寄存器，或称为锁存器
  - 流水线中各段的时间应该尽量相等，否则将会引起“堵塞”和“断流”的现象
  - 流水线需要有装入时间和排空时间，只有当流水线完全充满时，才能充分发挥效率

2.6.3 浮点运算流水线



- 过程段(S_i) ,高速的缓冲寄存器(L), 共计K个过程段,时钟周期τ , 过程段 S_i所需的时间为τ_i,缓冲寄存器的延时为τ_l, n 个任务。
- $\tau = \max\{\tau_i\} + \tau_l = \tau_m + \tau_l$ **(2.44)**
- 流水处理总周期数$T_k = k + (n-1)$ **(2.45)**
- 串行处理：$T_L = n \cdot k$ **(2.46)**
- 加速比： $C_k = T_L / T_k = (n \cdot k) / (k + (n-1))$ **(2.47)**，当 n>>k 时, $C_k \to k$

例子 32

- 4级流水浮点加法器，操作数检查t1=70ns/对阶t2=60ns/相加t3=90ns/规格化时间t4=80ns。缓冲时延τ为10ns。 求：

1） 4级流水加速比；

2）若每个过程段时间都为75ns（包括缓冲寄存器时间），加速比是多少？

- 解：1）加法流水线时钟周期至少为

$$t=90+10=100ns;$$

串行方式，浮加时间t1+t2+t3+t4=300ns。

加速比=300/100=3

2)加速比=300/75=4

第二章作业汇总

- 3 (增加：条件改为IEEE 754标准，重做一遍) 字长32位，其中符号位1位，阶码8位采用移码表示，，尾数23位采用补码表示。1）最大数；2）最小数；3）规格化数的范围。
- 5(1)(2)，变形补码计算x+y。
  1) x=11011, y=00011
  2) x=11011, y=-10101
- 6(1)(2) 变形补码计算x-y。
  1) x=11011, y=-11111
  2) x=10111, y=11011

第二章作业汇总（续）

- 8(1) 不恢复余数法/加减交替法(除数右移/余数左移，各演算一遍)

  1) x=11000, y=-11111

- 9(1) 浮点数加减法

  1) $x=2^{-011}\times 0.100101$， $y=2^{-010}\times(-0.011110)$

- 10(1) 浮点数乘法

  1) $(2^{3}\times 13/16)\times[2^{4}\times(-9/16)]$

练习

1)IEEE754标准中32为浮点数采用符号位1位，阶码8位，尾数23位，所能表示的最大规格化正数是（　）

A. $+(2-2^{-23})\times 2^{+127}$　B. $+(1-2^{-23})\times 2^{+127}$

C. $+(2-2^{-23})\times 2^{+255}$　D. $+(1-2^{-23})\times 2^{+255}$

2)在机器数中，（　）的零的表示形式是唯一的。

A. 原码　B.补码　C.反码　D.原码和反码

3)下列字符码包含了奇偶校验位，没有数据错误，采用偶校验的字符码是（　）

A. 11001011　B. 11010110　C. 11000001　D.11001001