第2章 运算方法和运算器b


BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS
北京邮电大学

2.3 定点乘法运算

乘除法运算的机器实现方法

1.完全软件实现

不设乘除法运算的硬件电路，而是由软件利用运算器中的加法器和移位寄存器实现乘、除法运算

2.加法器加上硬件辅助电路实现

利用运算器中的加法器硬件电路和移位寄存器，再设计必要的扩展电路，可以用硬件通过加法和移位操作实现乘、除法运算

3.专用乘、除法器实现

在运算器中除了设置加法器之外，采用专用硬件电路实现高速乘、除法部件，直接完成乘、除法运算

定点乘法运算算法

- 原码一位乘法运算
- 补码一位乘法运算（不要求）
- 原码两位乘法运算（不要求）
- 无符号的阵列乘法
- 有符号的阵列乘法
- 直接补码并行乘法（不要求）

原码一位乘法运算

令：$[X]_{原} = X_f.X_1\ X_2\ X_3\cdots\cdots X_n$

$[Y]_{原} = Y_f.Y_1\ Y_2\ Y_3\cdots\cdots Y_n$

则

$$[X*Y]_{原} = (X_f \oplus Y_f) + (|X|*|Y|)$$

$$= (X_f \oplus Y_f) + (0.X_1X_2\cdots X_n \cdot 0.Y_1Y_2\cdots Y_n)$$

积的符号：被乘数与乘数两符号的异或值

积的数值：被乘数与乘数两数的绝对值之积

二进制乘法

例．0.1101×1.1011

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

加符号：1.10001111

- 基本算法描述：
  - 若乘数的当前位为1， 则将被乘数和部分积求和。
  - 若乘数的当前位为0，则跳过。
  - 将部分积移位。
  - 所有位都完成后，部分积即为最终结果。
- N 位乘数 × M 位被乘数 => N+M 位的积
- 乘法显然比加法更复杂…
  - 但比十进制乘法要简单

部分积

令：$y=0.y_1y_2\cdots\cdots y_n$

则：$xy = x\times(y_1\times2^{-1}+ y_2\times2^{-2}+\cdots\cdots + y_n\times2^{-n})$

$= 2^{-1}\times(xy_1+ xy_2\times2^{-1}+\cdots\cdots + xy_n\times2^{-n+1})$

$= 2^{-1}\times(xy_1+ 2^{-1}(xy_2+ 2^{-1}(xy_3+ \cdots\cdots+$

$2^{-1}(xy_n+0))$

令：$z_0=0$

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

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

……

$z_i=2^{-1}(xy_{n-i+1}+z_{i-1})$ ……　　部分积

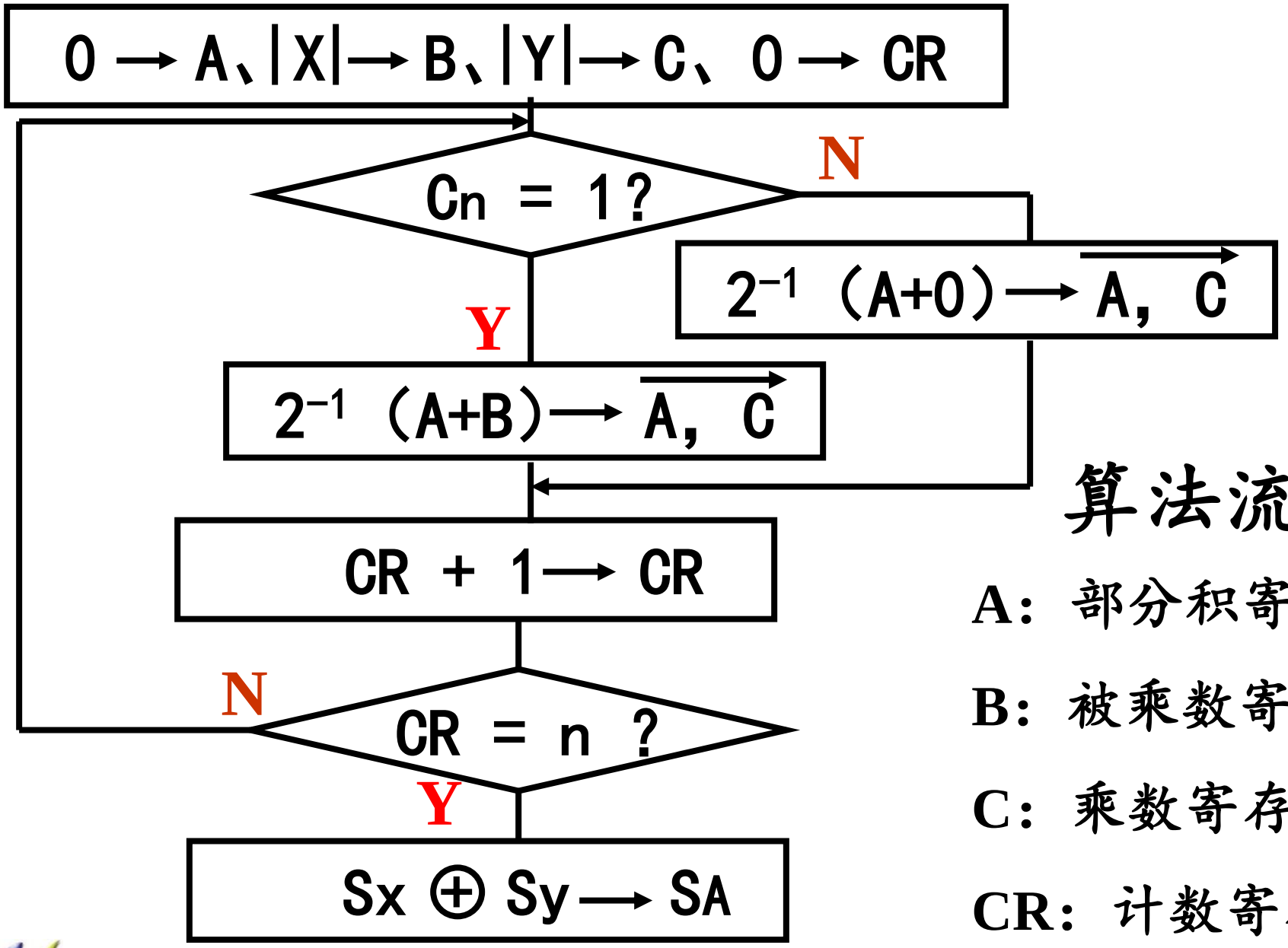
$z_n=2^{-1}(xy_1+z_{n-1})=xy$

A 部分积Z	乘数Y	C	B 00.1101 被乘数X
00 0000	1 0 1 1		$z_0=0, y_4=1$
+x 00 1101			xy_4+z_0
00 1101			
右移一位→ 00 0110	1 1 0 1	1(丢弃)	$z_1=2^{-1}(xy_4+z_0)$
+x 00 1101			
01 0011			
右移一位→ 00 1001	1 1 1 0	1(丢弃)	
+0 00 0000			
00 1001			
右移一位→ 00 0100	1 1 1 1	0(丢弃)	
+x 00 1101			
01 0001			
右移一位→ 00 1000	1 1 1 1	1(丢弃)	
乘积高位	乘积低位		

例:设X=0.1101,
Y=0.1011,
求X×Y

X×Y=0.10001111

移位4次，运算完成



算法流程

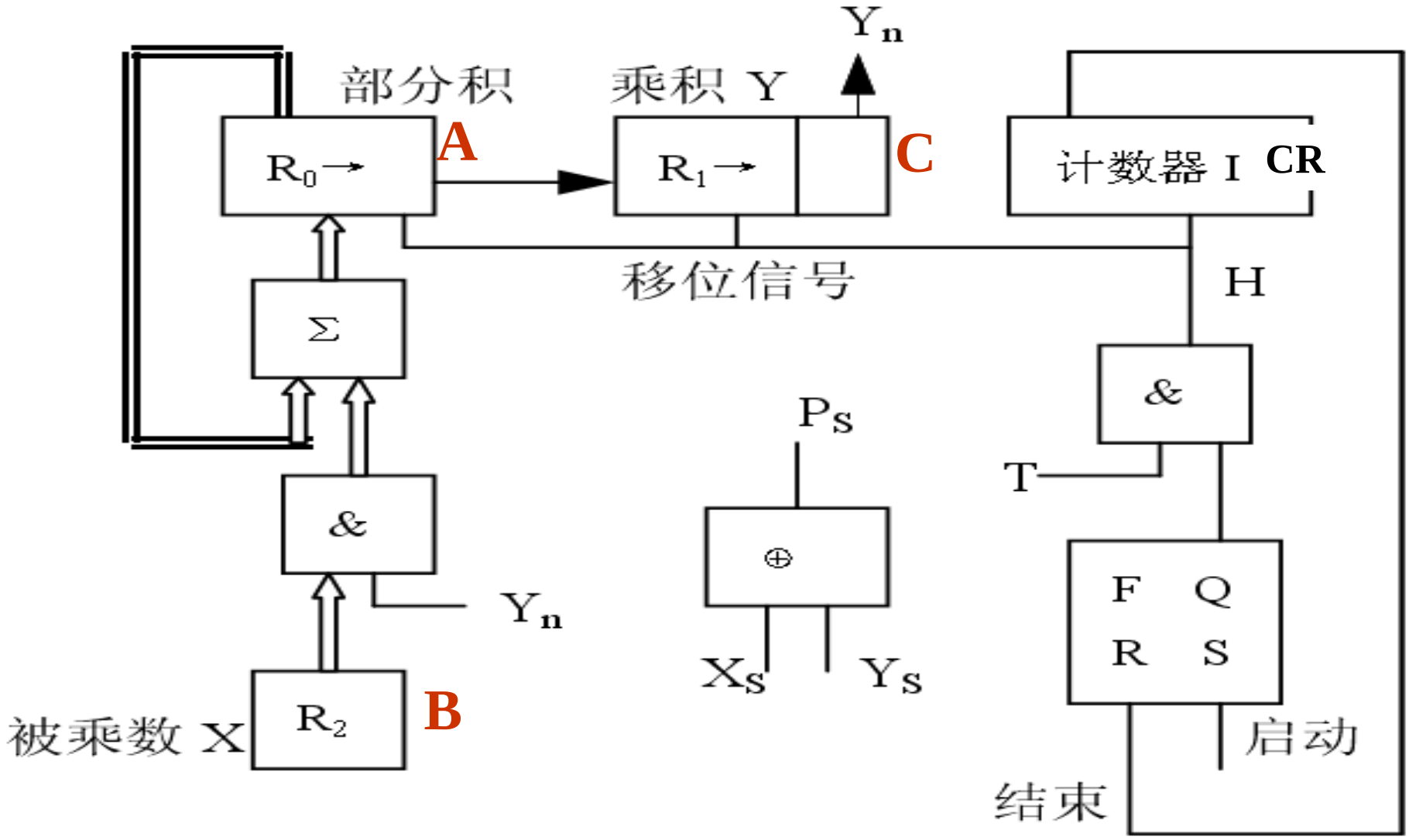
A：部分积寄存器

B：被乘数寄存器

C：乘数寄存器

CR：计数寄存器

原码一位乘法的硬件电路



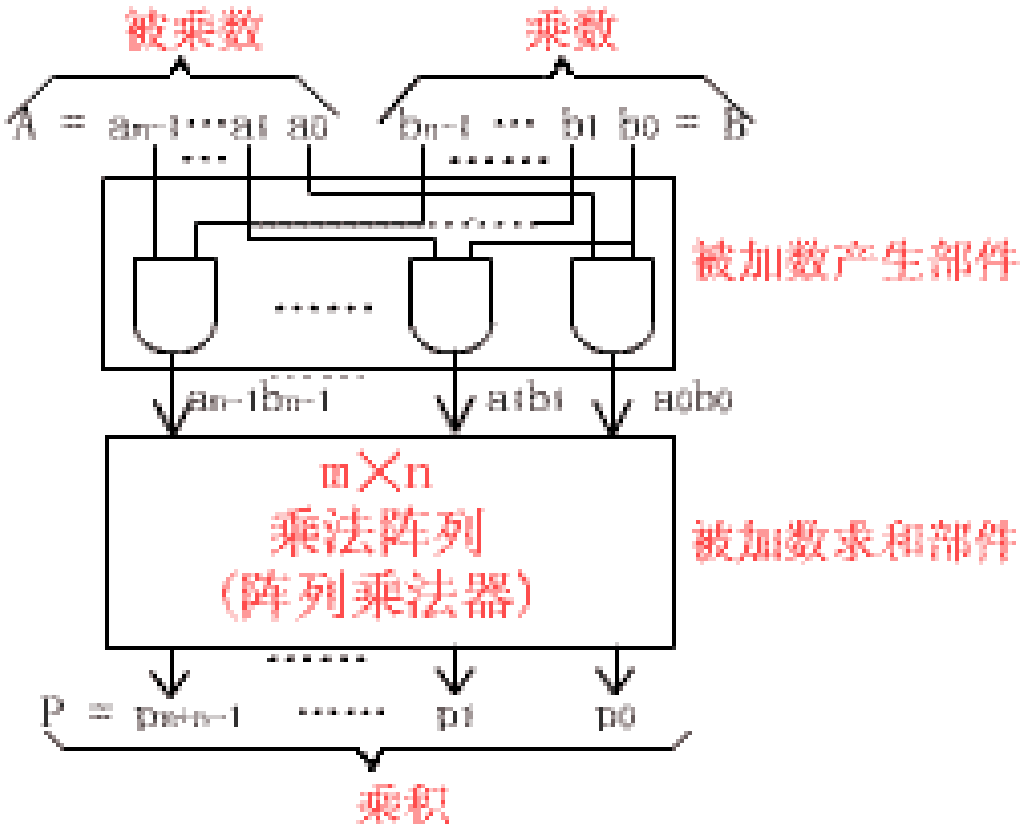
无符号数阵列乘法器

$A=a_{m-1}\cdots a_1a_0$
$B=b_{n-1}\cdots b_1b_0$

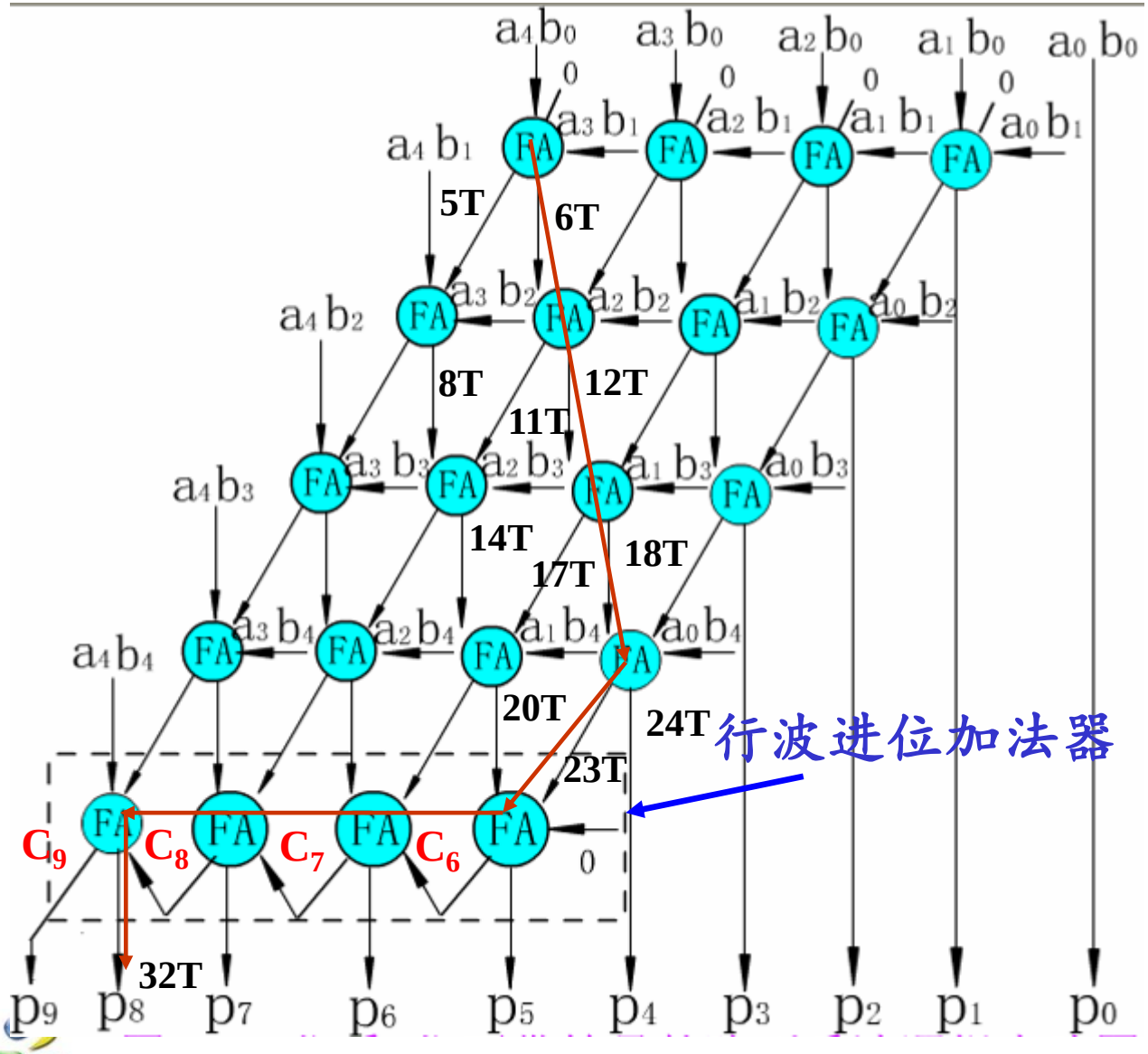
$P=A*B=p_{m+n-1}\cdots p_1p_0$

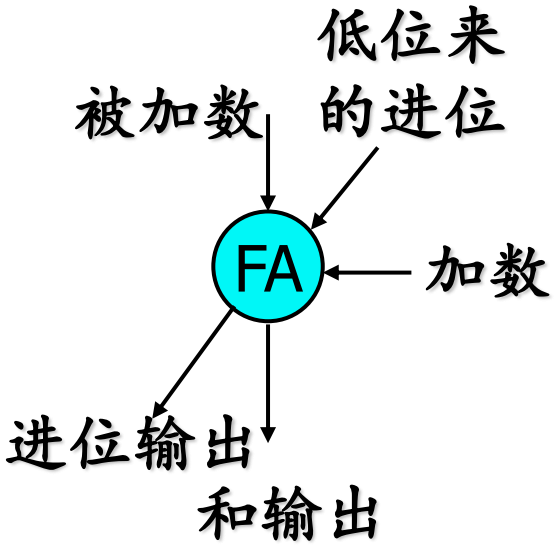
$$
\begin{array}{rccccccc}
 & a_{m-1} & a_{m-2} & \cdots & a_1 & a_0 & = & A \\
\times) & & b_{n-1} & \cdots & b_1 & b_0 & = & B \\
\hline
 & a_{m-1}b_0 & a_{m-2}b_0 & \cdots & a_1b_0 & a_0b_0 & & \\
 & a_{m-1}b_1 & a_{m-2}b_1 & \cdots & a_1b_1 & a_0b_1 & & \\
 & \vdots & \vdots & & \vdots & \vdots & & \\
+) & a_{m-1}b_{n-1} & a_{m-2}b_{n-1} & \cdots & a_1b_{n-1} & a_0b_{n-1} & & \\
\hline
p_{m+n-1} & p_{m+n-2} & p_{m+n-3} & \cdots & p_{n-1} & \cdots & p_1 & p_0 = P
\end{array}
$$

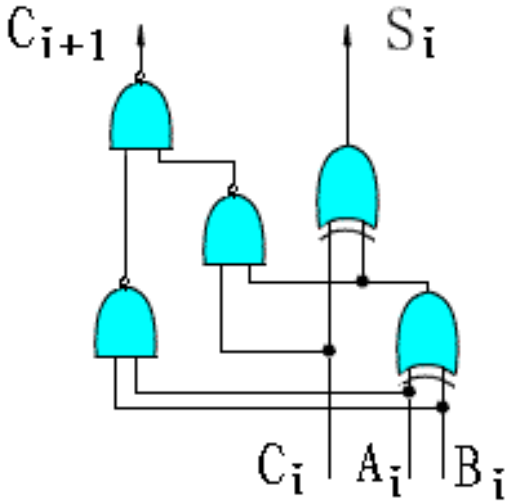
m×n位不带符号的阵列乘法器逻辑图



5位×5位不带符号的阵列乘法器







时间计算

- 实现n位×n位不带符号的原码阵列乘法器，需要n(n-1)个全加器和n^2个“与”门
- n位×n位不带符号的原码阵列乘法器总的乘法时间估算为：

$$t_m = T_a + (n-1)\times 6T + (n-1)\times T_f$$
$$= T + (n-1)\times 6T + (n-1)\times T_f$$
$$= (8n-7)T$$

  - T_a是计算aibi的时间，为1T
  - T_f是全加器的进位延迟，为2T

例19

$a4b0=1$ $a3b0=1$ $a2b0=0$ $a1b0=1$ $a0b0=1$

$a4b1=0$ $a3b1=0$ $a2b1=0$ $a1b1=0$ $a0b1=0$

$a4b2=1$ $a3b2=1$ $a2b2=0$ $a1b2=1$ $a0b2=0$

$a4b3=0$ $a3b3=0$ $a2b3=0$ $a1b3=0$ $a0b3=0$

$a4b4=1$ $a3b4=1$ $a2b4=0$ $a1b4=1$ $a0b4=1$

```
                    1 1 0 1 1 = A(27₁₀)
  ×                 1 0 1 0 1 = B(21₁₀)
-------------------------------------------
                    1 1 0 1 1
                  0 0 0 0 0
                1 1 0 1 1
              0 0 0 0 0
  +       1 1 0 1 1
-------------------------------------------
      1 0 0 0 1 1 0 1 1 1 = P
```

$P=p9p8p7p6p5p4p3p2p1p0=1000110111\ (567_{10})$

有符号数的并行乘法运算

- 原码（即用原码表示的机器）
  - 尾数参加无符号数乘法器运算。
  - 符号位单独处理，即通过异或门得到乘积的符号。
- 补码（用补码表示的机器）
  - 间接补码乘法
    - 正数：尾数参加无符号数乘法器运算
    - 负数：由补码求得其绝对值后参加无符号数乘法器运算
    - 符号位通过异或门得到乘积的符号
    - 若乘积为负数，需将乘积的绝对值经过求补电路得到积的补码
  - 直接补码乘法（不要求）

求补器电路图

- 例：求补

1 010101 ~~求补~~→ 1 101011；　1 101000 ~~求补~~→ 1 011000

  - 从数的最右端开始，从右向左直到找出第一个“1”，该“1”保持不变，以左的每一个位都求反

E是符号位

E＝0时输出与输入相等

E＝1时对其求补

$C_{-1}=0$

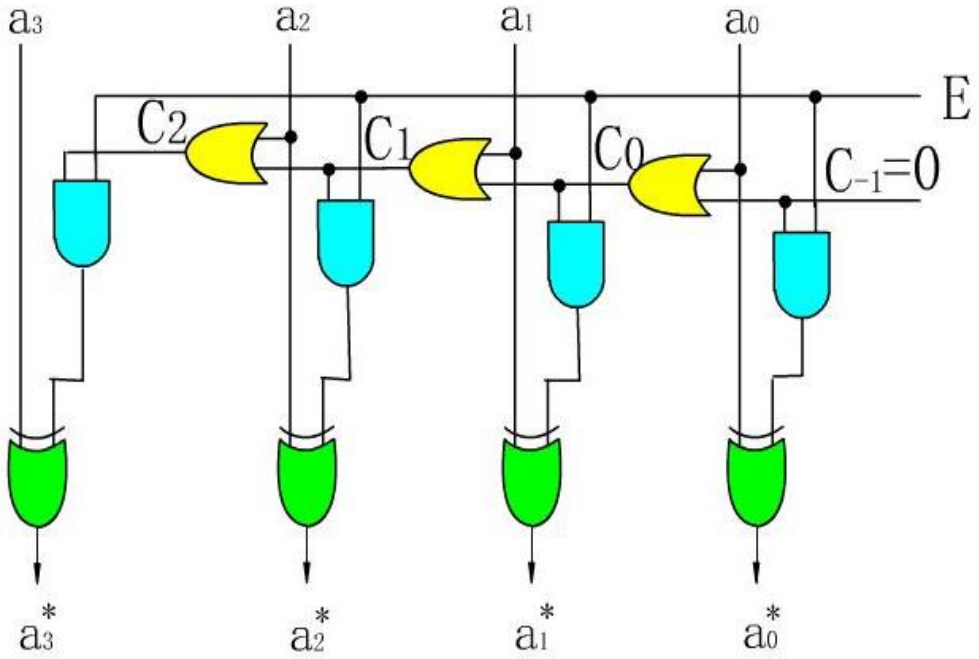
$C_i=a_i+C_{i-1}$

$a_i^*=a_i \oplus EC_{i-1}$

n+1位(不含符号位)的时间：

$t_{TC}=n*2T+5T$

$=(2n+5)T$



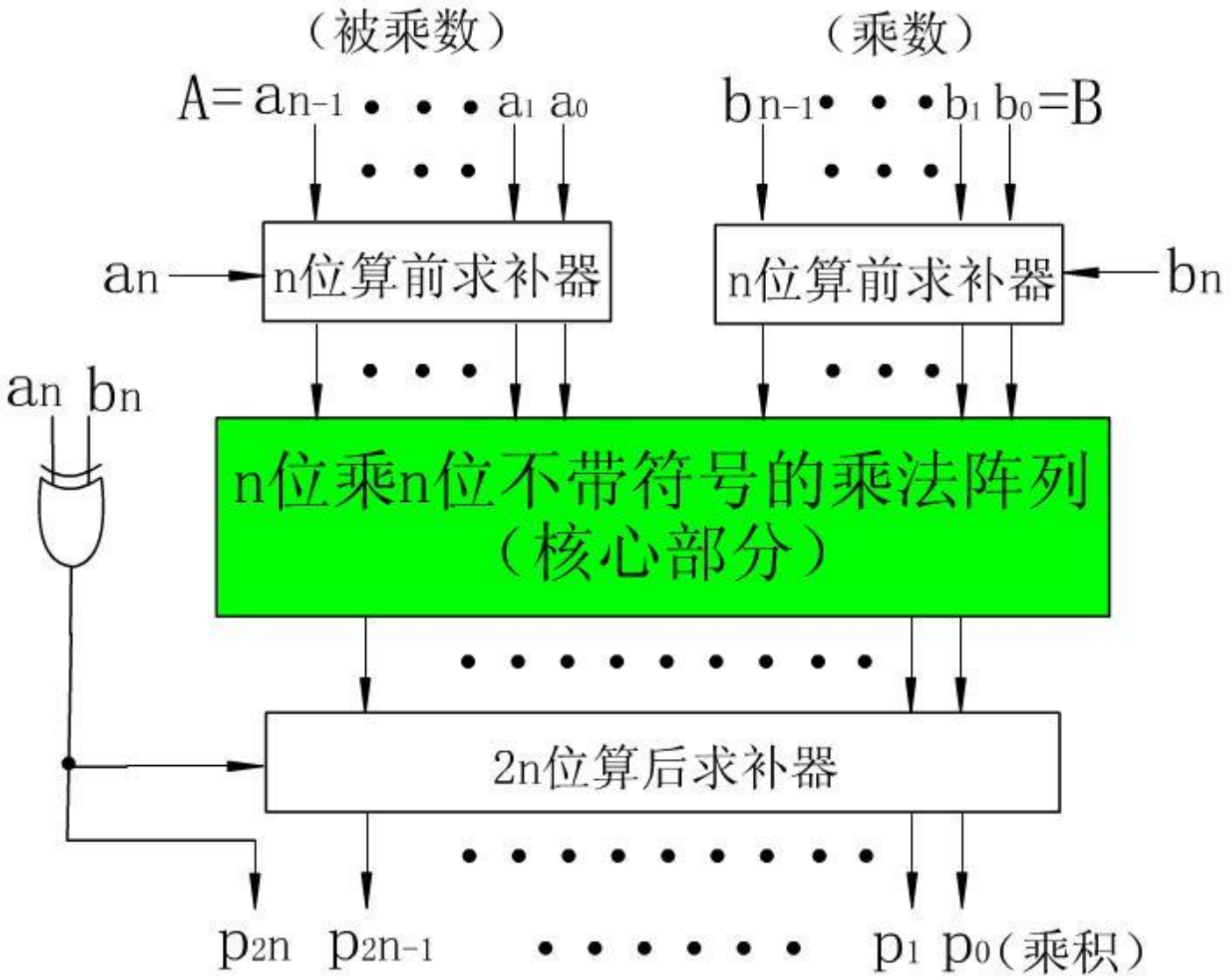
假设：与门、或门延迟为2T，异或门延迟为3T

n+1位间接补码阵列乘法器

三个求补器:

- 两个n位算前求补器
- 一个2n位算后求补器

n×n位不带符号的乘法阵列



例

设 x ＝＋15， y ＝ －13， 已知机器补码表示有符号数，请用间接补码阵列乘法器求出乘积 x × y ＝？

$[x]_{补}$ ＝01111　　$[y]_{补}$ ＝ 10011

算前求补器输出 | x |＝1111，| y |＝1101

```
              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
```

算后求补级输出并加上乘积符号位1，得

$[x \times y]_{补}$ ＝ 100111101

二进制数真值：$x \times y = (-1000011)_2 = (-195)_{10}$

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


BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS
北京邮电大学

2.4 定点除法运算

原码除法运算原理

- 设有n位定点小数（定点整数也适用）

  被除数x，其原码为$[x]_{原}=x_f.x_{n-1}\cdots x_1x_0$

  除数y，其原码为$[y]_{原}=y_f.y_{n-1}\cdots y_1y_0$

  则其商q＝x÷y，其原码为

  $[q]_{原}=(x_f\oplus y_f)+(0.x_{n-1}\cdots x_1x_0\div 0.y_{n-1}\cdots y_1y_0)$

手工除法运算过程

x÷y？，x=0.1001，y=0.1011

	0.1101	商q
0.1011)	0.10010	x (r_0)　被除数小于除数，商0
−0.01011		$2^{-1}y$　除数右移1位，减除数，商1
	0.001110	r_1　得余数r_1
−0.001011		$2^{-2}y$　除数右移1位，减除数，商1
	0.000110	r_2　得余数r_2
−0.000**1**011		$2^{-3}y$　除数右移1位，不减除数，商0
	0.0001100	r_3　得余数r_3
−0.0000**1**011		$2^{-4}y$　除数右移1位，减除数，商1
−0.000000001		r_4　得余数r_4

- 商0还是商1？人比较后才可以确定，但计算机如何确定？
- 余数末位补0后，减去除数右移后的值，导致加法器尾数逐渐增多，最后要求加法器的位数必须为被除数的2倍

加减交替法（不恢复余数法）

- 判断是否够减： 被除数或部分余数左移减去除数得新余数 $r'_i = 2r_{i-1} - y$
  - 若$r'_i > 0$，则i位上商1，且在下一步，r_i左移减除数。因为：
    - $r_i = r'_i$
    - $r_{i+1} = 2r_i - y = 2r'_i - y$
  - 若$r'_i < 0$，则i位上商0，且在下一步，r_i左移加除数。因为：
    - $r_i = r'_i + y$
    - $r_{i+1} = 2r_i - y = 2(r'_i + y) - y = 2r'_i + y$

即：将恢复余数与下一步部分余数左移减去除数合并为加余数。

$2r'_i + y$也可理解为：商0后，$2r'_i$中已多减了2y，计算r_{i+1}时加y实际取得减除数的作用

- 如果最终的余数是负数，需“纠余”得到除法运算的余数：$r_n = r'_n + y$

例：

X=0.01011
Y=0.01101

被除数(余数)		商	
0 0 1 0 1 1		0 0 0 0 0	开始情形
+) 1 1 0 0 1 1	补码		-Y
0 1 1 1 1 0		0 0 0 0 0	<0, 商0
1 1 1 1 0 0		0 0 0 0 0	左移1位
+) 0 0 1 1 0 1			+Y
1 0 0 1 0 0 1		0 0 0 0 1	>0, 商1
0 1 0 0 10		0 0 0 1 0	左移1位
+) 1 1 0 0 1 1			-Y
1 0 0 0 1 0 1		0 0 0 1 1	>0, 商1
0 0 1 0 1 0		0 0 1 1 0	左移1位
+) 1 1 0 0 1 1			-Y
0 1 1 1 0 1		0 0 1 1 0	<0, 商0
1 1 1 0 1 0		0 1 1 0 0	左移1位
+) 0 0 1 1 01			+Y
1 0 0 0 1 1 1		0 1 1 0 1	>0, 商1

可控加法/减法（CAS）单元

- 当输入线P=0时，CAS做加法运算
- 当输入线P=1时，CAS做减法运算

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

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

当P=0时：

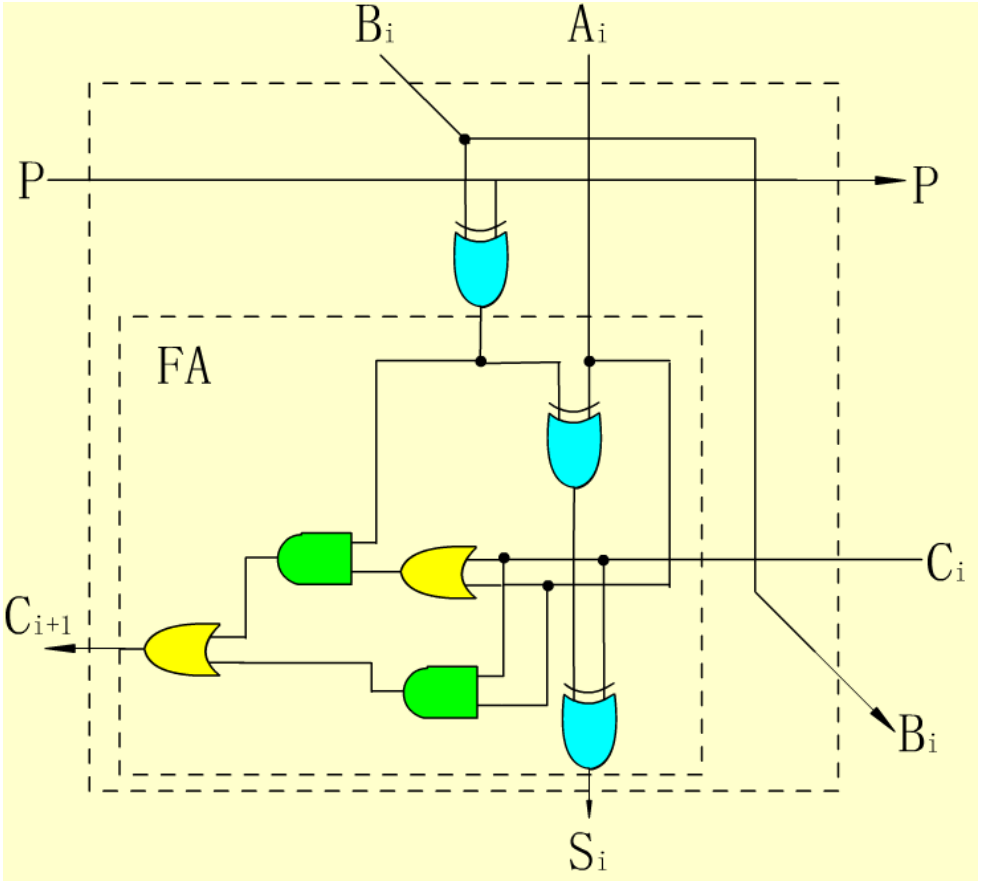
$S_i=A_i \oplus B_i \oplus C_i$

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

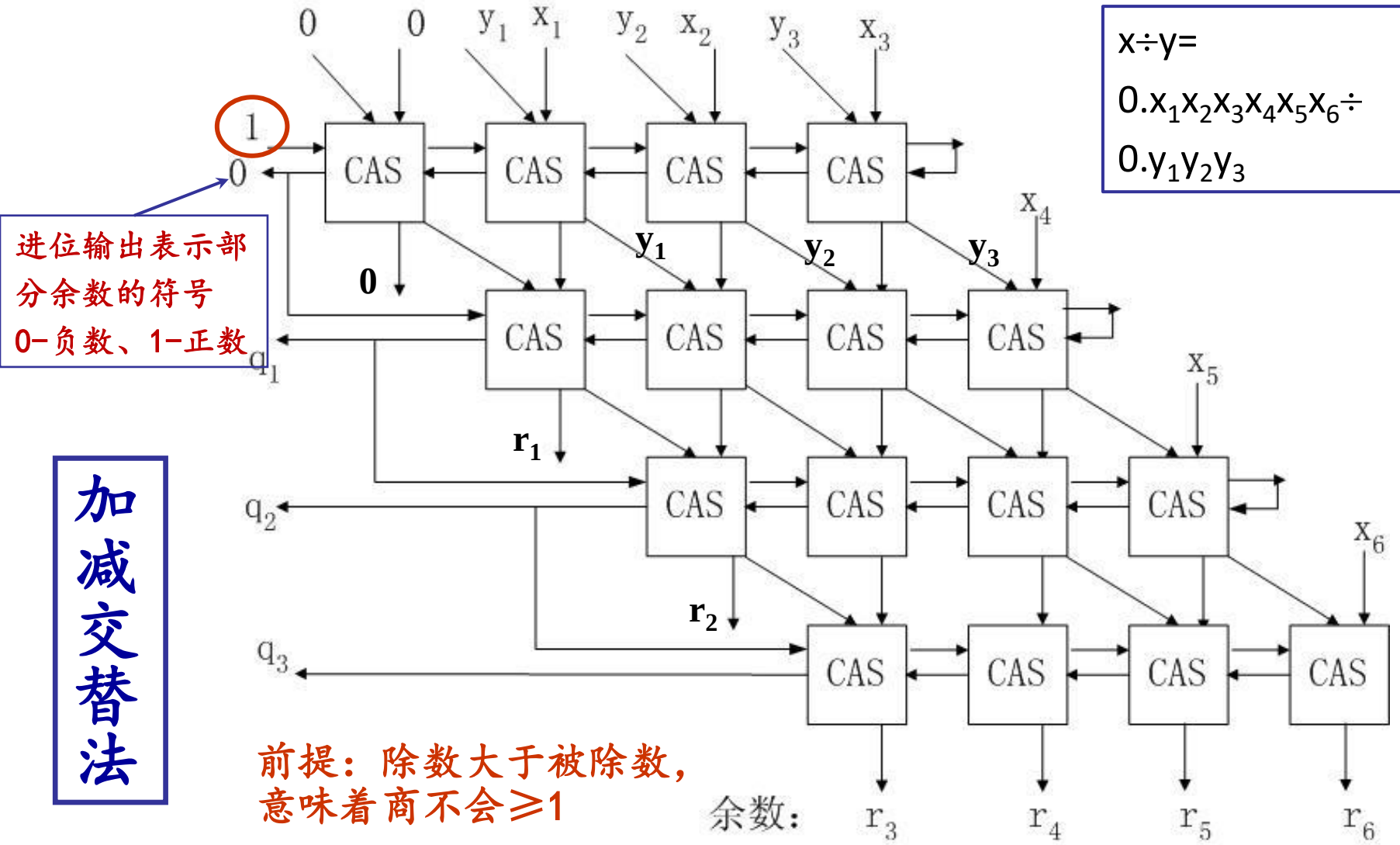
当P=1时：

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

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



不恢复余数的阵列除法器



x÷y=

$0.x_1x_2x_3x_4x_5x_6 \div 0.y_1y_2y_3$

加减交替法

前提：除数大于被除数，意味着商不会≥1

时间延迟

对一个2n位（小数部分）除以n位的不恢复余数阵列除法器

- $(n+1)^2$个CAS单元
- CAS单元的进位延迟为3T

除法的执行时间 $t= (n+1)^2 3T$

$=3(n+1)^2 T$


BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS
北京邮电大学

2.5 定点运算器的组成

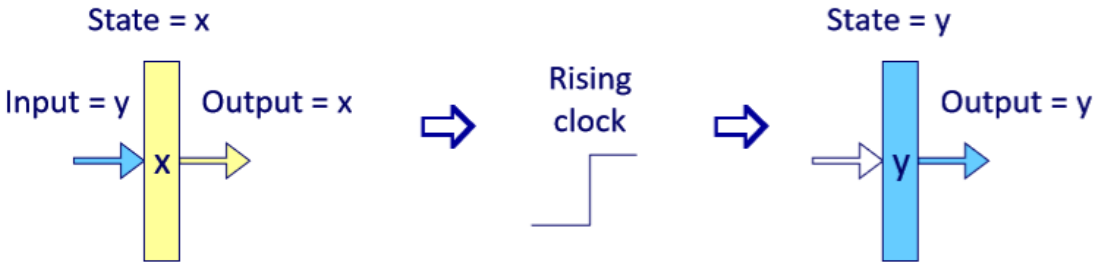
运算器的基本结构

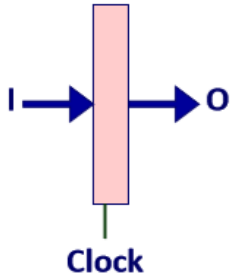
典型运算器包括：

- 算术逻辑运算单元ALU
- 阵列乘除法器
- 寄存器组
- 多路选择器
- 三态门
- 数据总线

寄存器

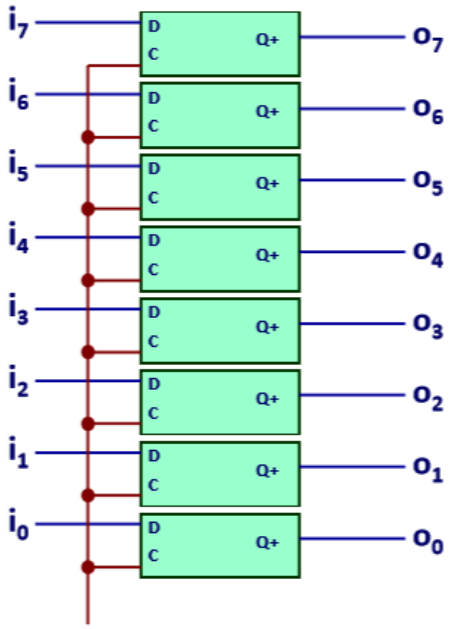
Registers





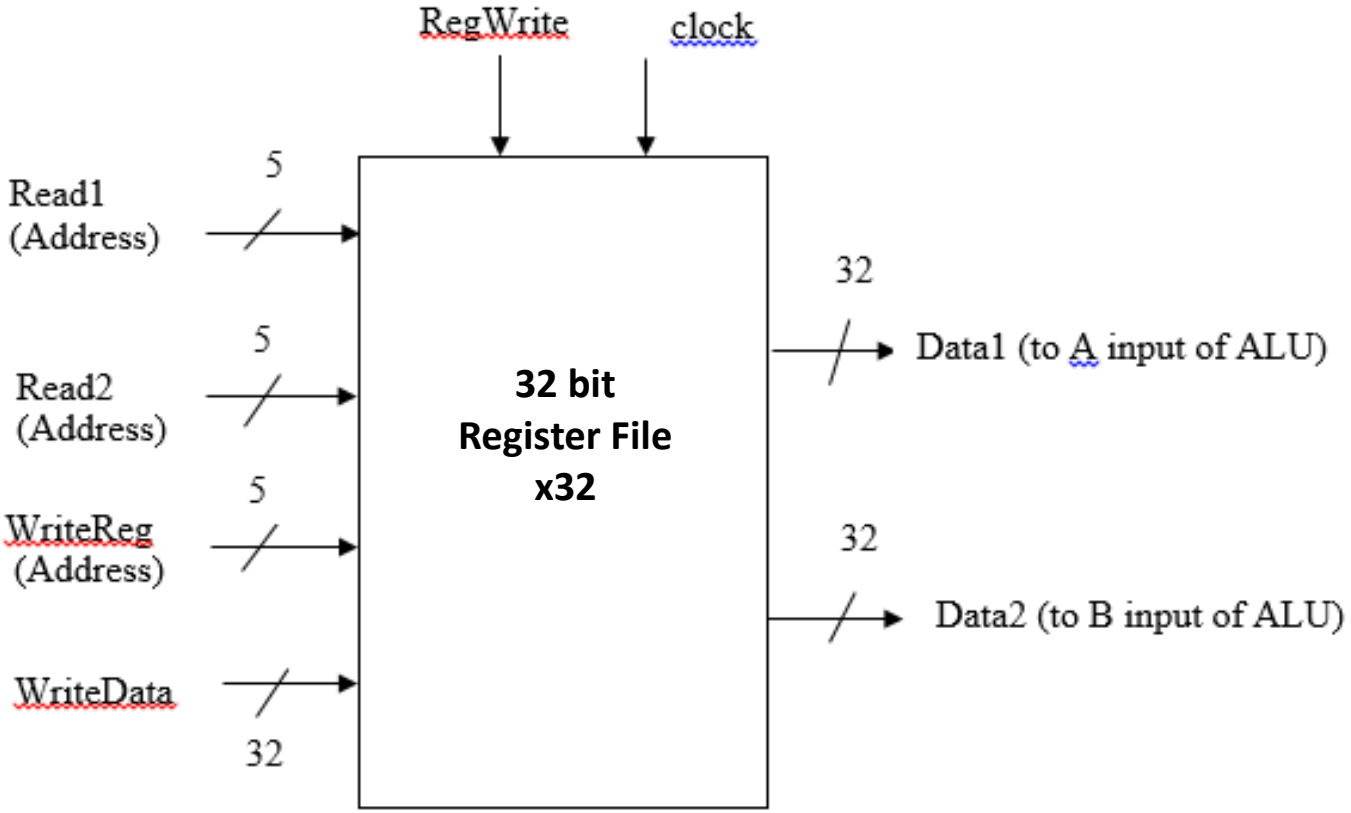
- Stores data bits
- For most of time acts as barrier between input and output
- As clock rises, loads input

Structure



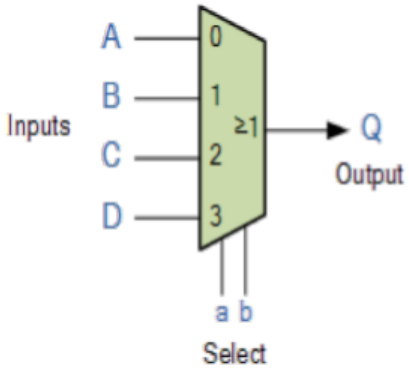
- Stores word of data
  - Different from *program registers* seen in assembly code
- Collection of edge-triggered latches
- Loads input on rising edge of clock

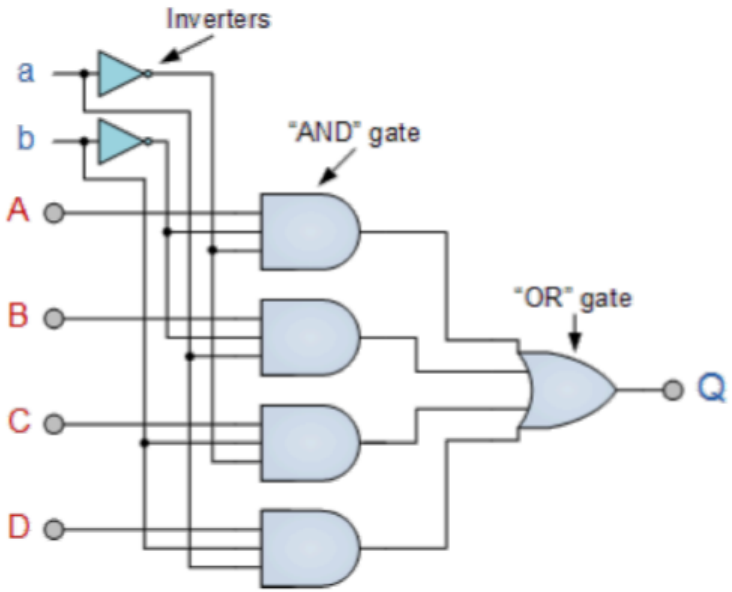
寄存器组

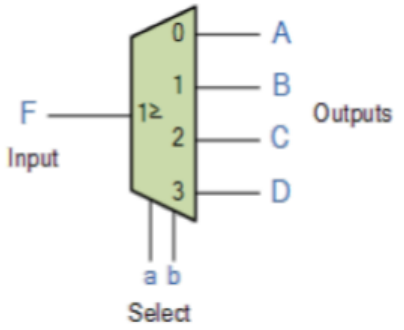

RegWrite
clock
5
Read1
(Address)
5
Read2
(Address)
5
WriteReg
(Address)
WriteData
32
32 bit
Register File
x32
32
Data1 (to A input of ALU)
32
Data2 (to B input of ALU)

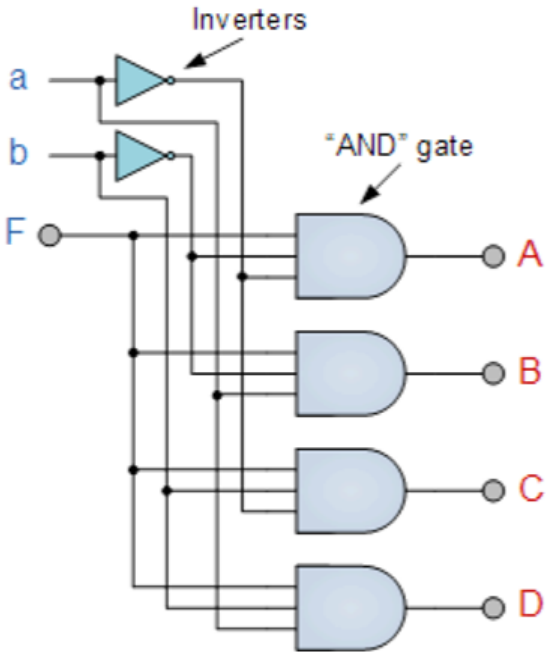
多路复用器/多路分配器

Multiplexor/Demultiplexor

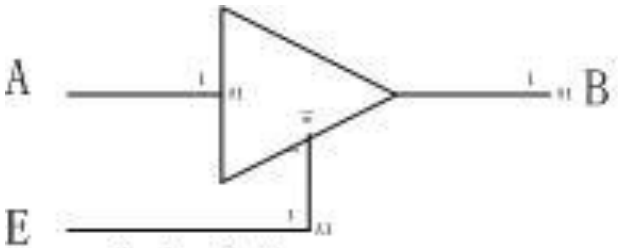




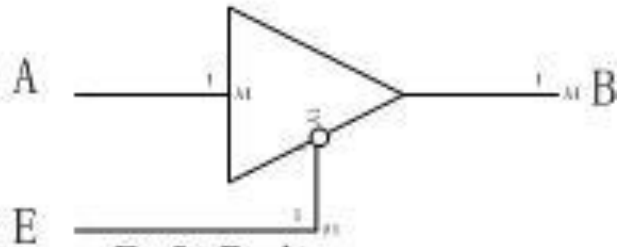




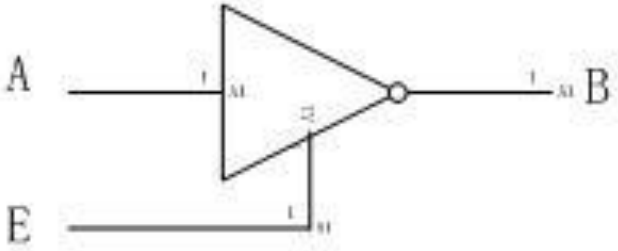
三态门



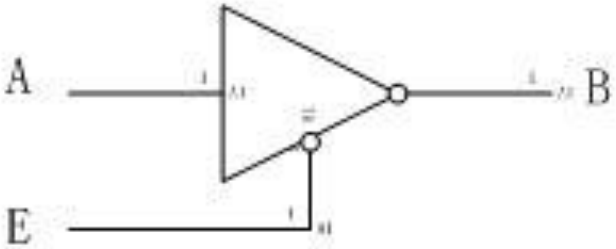
E=1, B=A
E=0, B=高阻
74LS241



E=0, B=A
E=1, B=高阻
74LS244



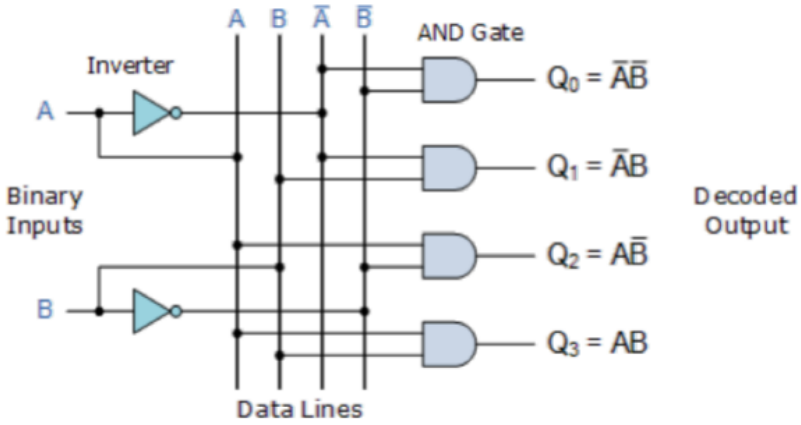
E=1, B=$\overline{A}$
E=0, B=高阻
74LS240

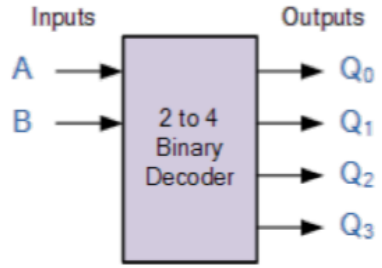


E=0, B=$\overline{A}$
E=1, B=高阻
74LS245

译码器/编码器

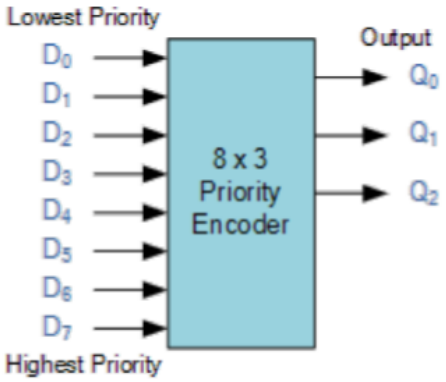
Decoder/Encoder





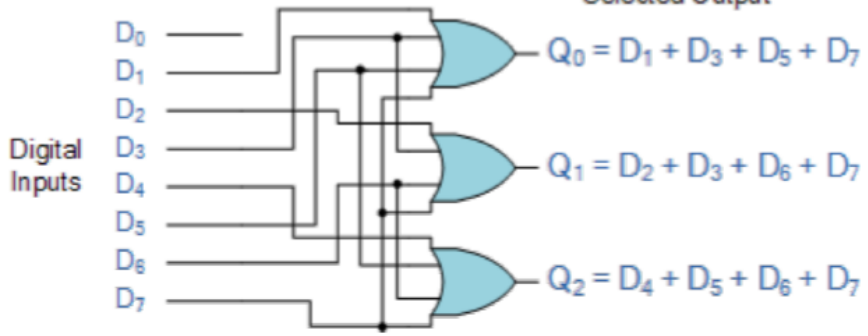
Truth Table

A	B	Q_0	Q_1	Q_2	Q_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

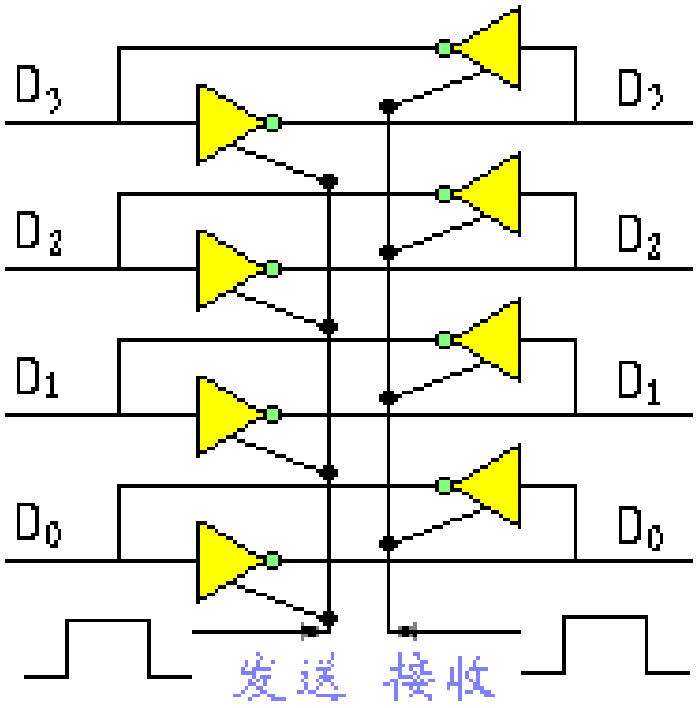


Inputs								Outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Q_2	Q_1	Q_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	x	0	0	1
0	0	0	0	0	1	x	x	0	1	0
0	0	0	0	1	x	x	x	0	1	1
0	0	0	1	x	x	x	x	1	0	0
0	0	1	x	x	x	x	x	1	0	1
0	1	x	x	x	x	x	x	1	1	0
1	x	x	x	x	x	x	x	1	1	1

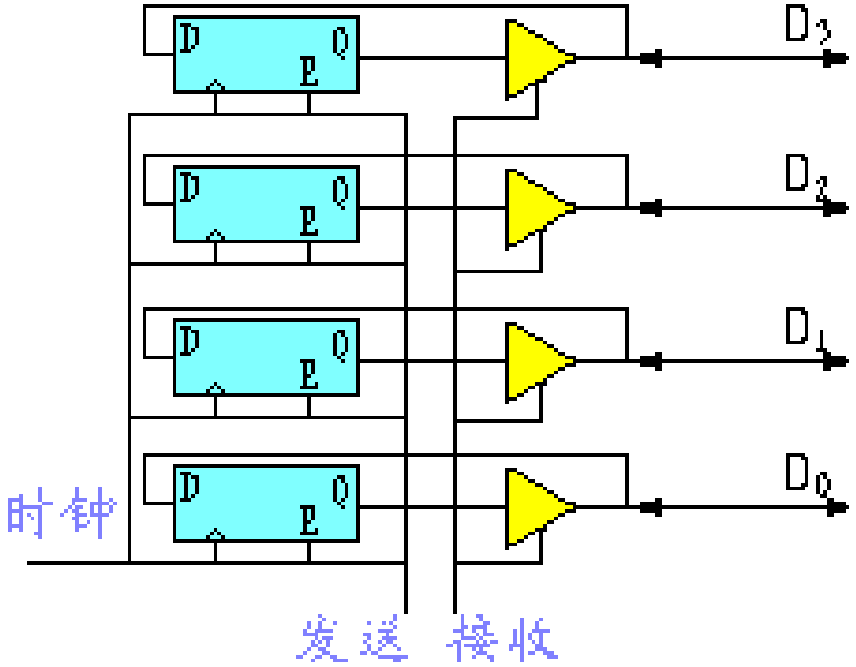
X = don't care



总线驱动器



(a) 带有缓冲器的双向数据总线



(b) 带有锁存器的4位双向数据总线

内部总线

- 机器内部各部件数据传送频繁，可以把寄存器间的数据传送通路加以归并，组成总线结构
- 总线分类
  - 所处位置
    - 内部总线（CPU内）
    - 外部总线（系统总线）
  - 逻辑结构
    - 单向传送总线
    - 双向传送总线

逻辑运算的应用需求

- 逻辑或： 1∨x=1， 0∨x=x

  将一个数的某些指定比特置1，其他比特保持不变

  例如：a∨01000010 ↔ 将a的第1和第6位置1
- 逻辑与： 1∧x=x， 0∧x=0

  将一个数的某些指定比特清0，其他比特保持不变

  例如：a∧10111101 ↔ 将a的第1和第6位清0
- 逻辑异或： 1⊕x=/x， 0⊕x=x

  将一个数的某些指定比特取反，其他比特保持不变

  例如：a⊕01000010 ↔ 将a的第1和第6位取反

一位全加器FA

- 逻辑表达式

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

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

只能完成加、减运算

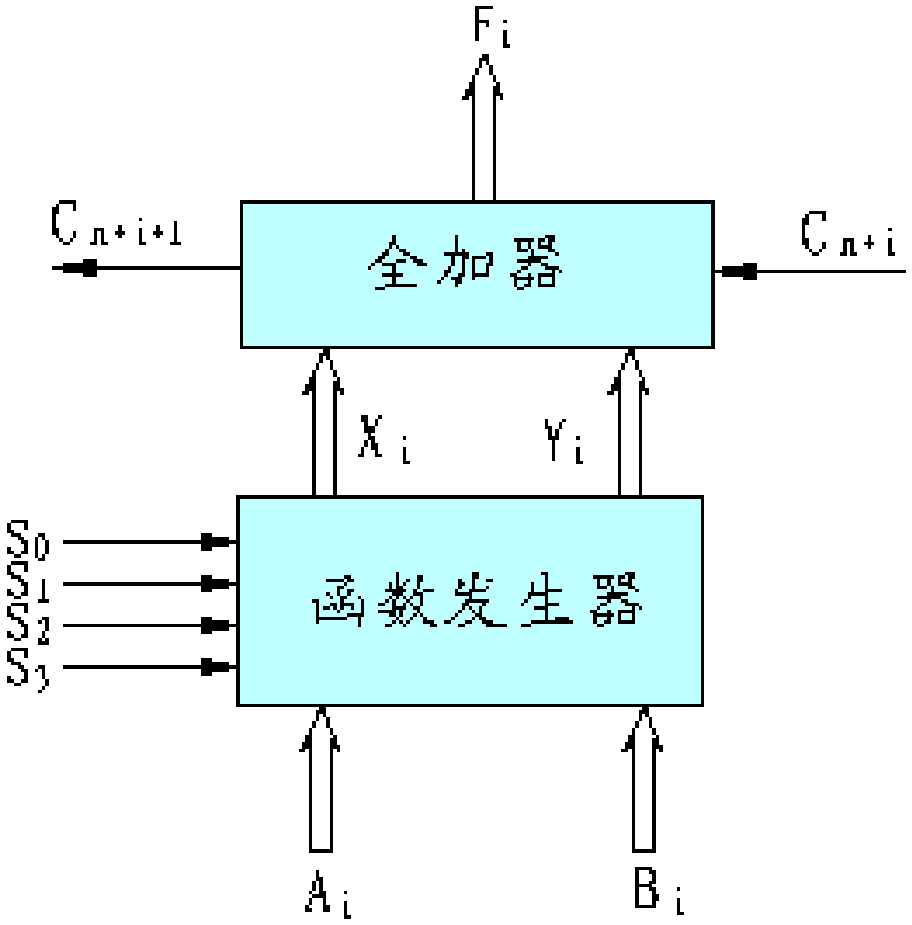
- 一种解决方法

为了实现多种算术逻辑运算，可将A_i和B_i输入一个函数发生器得到输出X_i和Y_i，再作为一位全加器的输入

算术/逻辑运算单元ALU（1）

基本思想

- 先将原全加器的输入A_i、B_i送至函数发生器进行处理
- S_0、S_1、S_2、S_3为函数发生器的控制信号
- 函数发生器的输出X_i、Y_i至全加器进行全加
- 由S_0、S_1、S_2、S_3来控制ALU所完成的运算

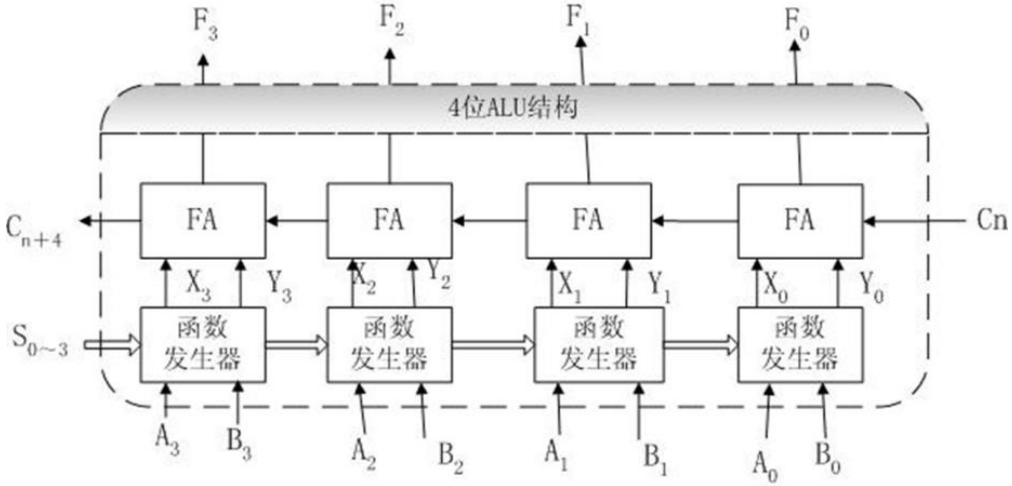


算术/逻辑运算单元ALU（2）

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

  $C_{n+i+1} = X_i Y_i + Y_i C_{n+i} + X_i C_{n+i}$

  - 用n+i代替原来一位全加器C_i中的i，i表示在一片ALU芯片上二进制位数。对于一片4位ALU，i＝0、1、2、3；n表示用多片ALU组成更多位数运算器时每片电路的进位输入。如果用4片4位ALU组成一个16位运算器，n＝0、4、8、12



算术/逻辑运算单元ALU逻辑表达式

- 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	$\bar{A}_i B_i$	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	$\bar{A}_i$

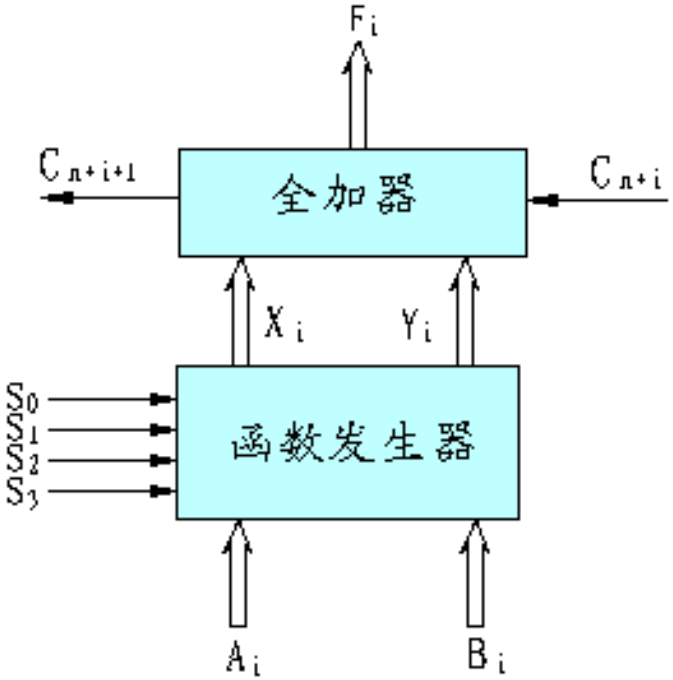
$$X_i=\bar{S}_2\bar{S}_3+\bar{S}_2S_3(\bar{A}_i+\bar{B}_i)+S_2\bar{S}_3(\bar{A}_i+B_i)+S_2S_3\bar{A}_i$$
$$=\overline{S_3A_iB_i+S_2A_iB_i}$$

$$Y_i=\bar{S}_0\bar{S}_1\bar{A}_i+\bar{S}_0S_1\bar{A}_iB_i+S_0\bar{S}_1\bar{A}_iB_i$$
$$=\overline{A_i+S_0B_i+S_i\bar{B}_i}$$

$$X_i\ Y_i=Y_i \Rightarrow C_{n+i+1}=Y_i+X_iC_{n+i}$$

ALU的第i位逻辑表达式

$$\begin{cases} X_i = \overline{S_3 A_i B_i + S_2 A_i \overline{B_i}} \\ Y_i = \overline{A_i + S_0 B_i + S_1 \overline{B_i}} \\ F_i = X_i \oplus Y_i \oplus C_{n+i} \\ C_{n+i+1} = Y_i + X_i C_{n+i} \end{cases}$$



例如：$S_3S_2S_0S_1=0000$

代入：

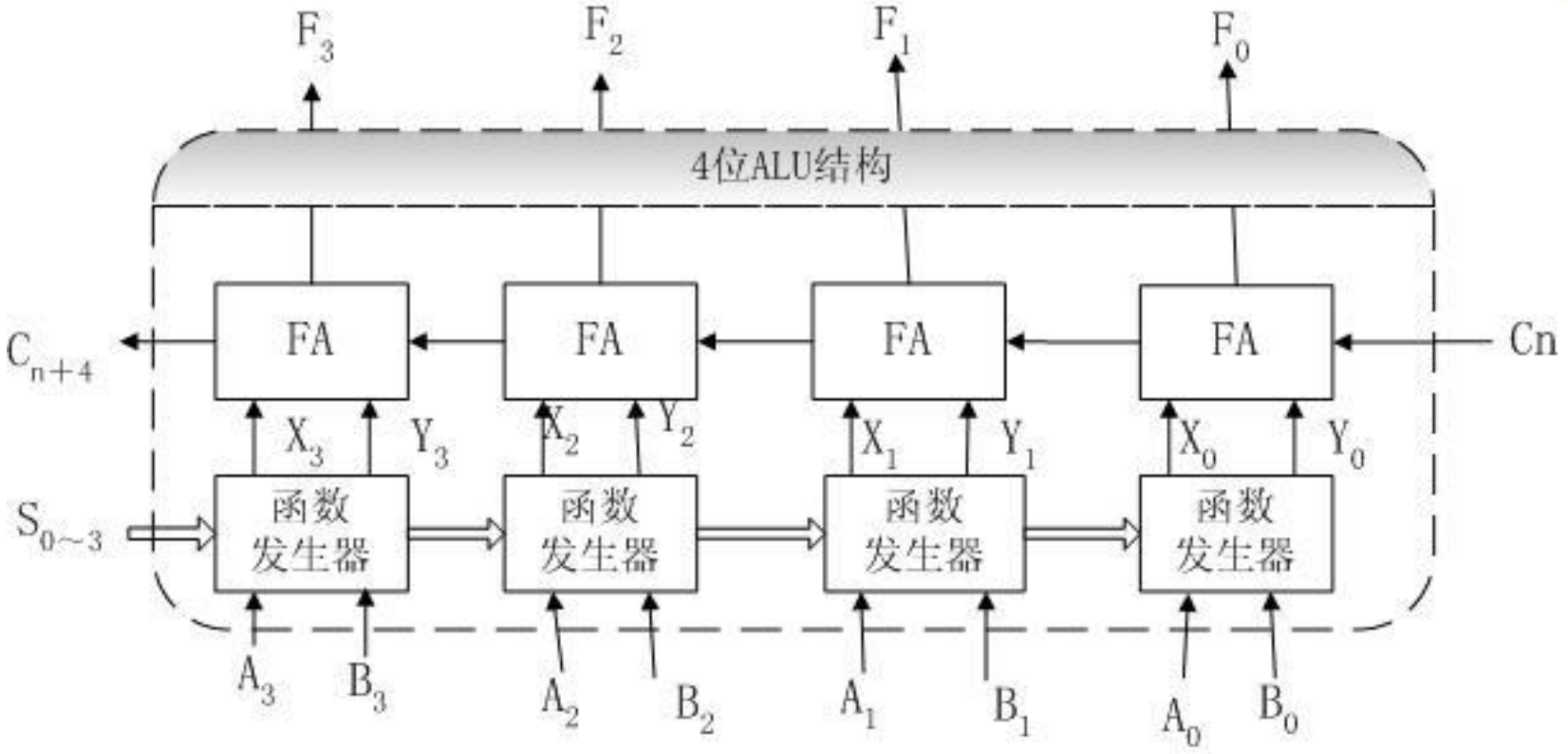
$$X_i = \overline{S_3 A_i B_i + S_2 A_i \overline{B_i}} = \overline{0+0} = 1$$

$$Y_i = \overline{A_i + S_0 B_i + S_1 \overline{B_i}} = \overline{A_i}$$

$$F_i = Y_i \oplus X_i \oplus C_{n+1} = \overline{A_i} \oplus 1 \oplus C_{n+1} = A_i \text{(设} C_{n+1}=0\text{)}$$

所以，在$S_3 \sim S_0=0000$的时候，输出结果为A_i

4位ALU结构



- 片内是串行进位还是并行进位？

串行进位，速度慢，可以改进

$$C_{n+1} = Y_0 + X_0 C_n$$
$$C_{n+2} = Y_1 + X_1 C_{n+1}$$
$$C_{n+3} = Y_2 + X_2 C_{n+2}$$
$$C_{n+4} = Y_3 + X_3 C_{n+3}$$

采用先行进位

- 每一位的进位公式：

$C_{n+1}=Y_0+X_0C_n$

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

$C_{n+3}=Y_2+X_2C_{n+2}=Y_2+Y_1X_2+Y_0X_1X_2+X_0X_1X_2C_n$

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

$=G+PC_n$

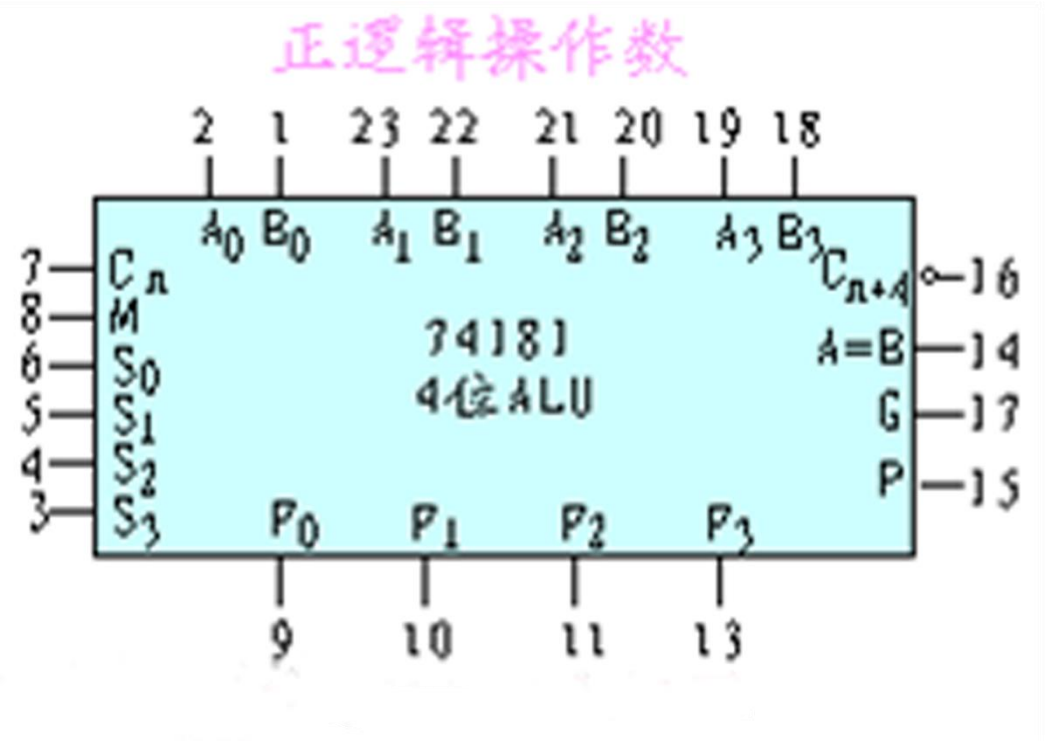
其中，$G=Y_3+Y_2X_3+Y_1X_2X_3+Y_0X_1X_2X_3$，$P=X_0X_1X_2X_3$

- **G为进位发生输出　P为进位传送输出**
- **P、G是为了实现多片ALU之间先行进位而设置的**

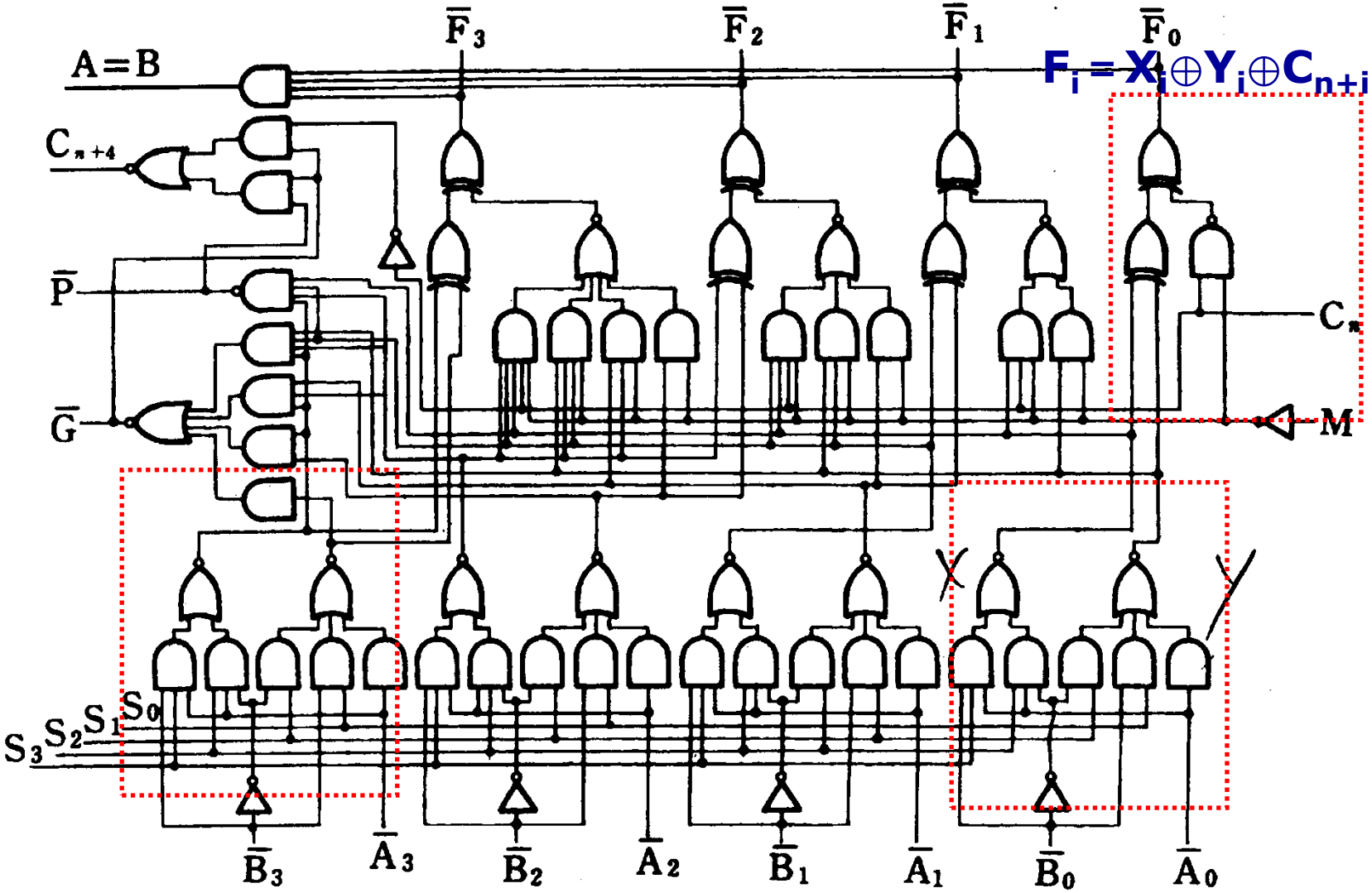
从上式可以看出：第0位的进位输入C_n可以直接传送到最高进位位上去，从而实现高速运算

4位算术/逻辑运算单元实例

- 74181



74181ALU逻辑图



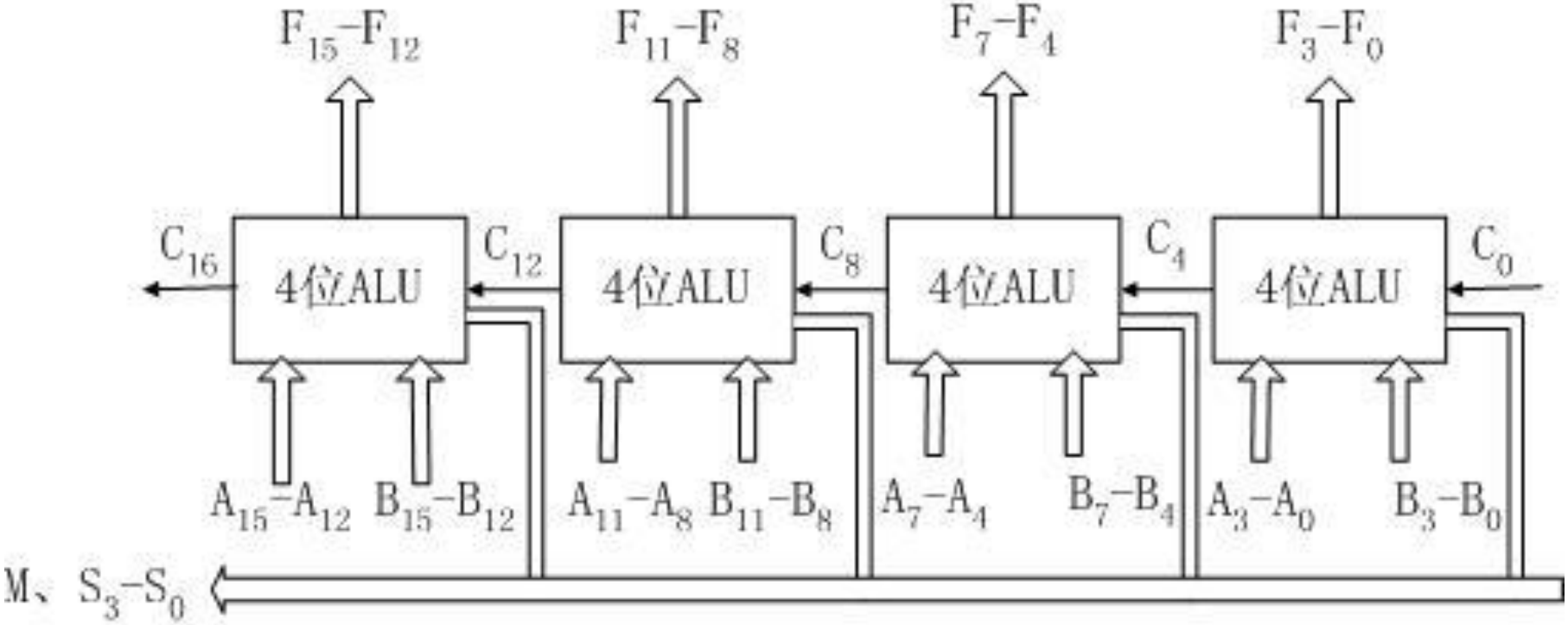
算术逻辑运算的实现

- M=0时，对进位信号没有影响，做算术运算
- M=1时，进位位门被封锁，做逻辑运算
- 说明：
  - A=B端可以判断两个数是否相等

- 74181ALU算数/逻辑运算功能表（见书表2.4）
  - 可以实现16中逻辑运算和16种算术运算

设计一个16位ALU

- 片内先行进位，片间串行进位



片间串行进位是否可以进一步改进？

改进 两级先行进位16位ALU

- 74181ALU已设置了P和G两个本组先行进位输出端
- 将4片74181的P、G输出端送入到74182先行进位部件（CLA），实现第二级的先行进位，即组与组之间的先行进位

74182CLA的进位逻辑关系

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

$C_{n+y}=G_1+P_1C_{n+x}=G_1+G_0P_1+P_0P_1C_n$

$C_{n+z}=G_2+P_2C_{n+y}$

$=G_2+G_1P_2+G_0P_1P_2+P_0P_1P_2C_n$

$C_{n+4}=G_3+P_3C_{n+z}$

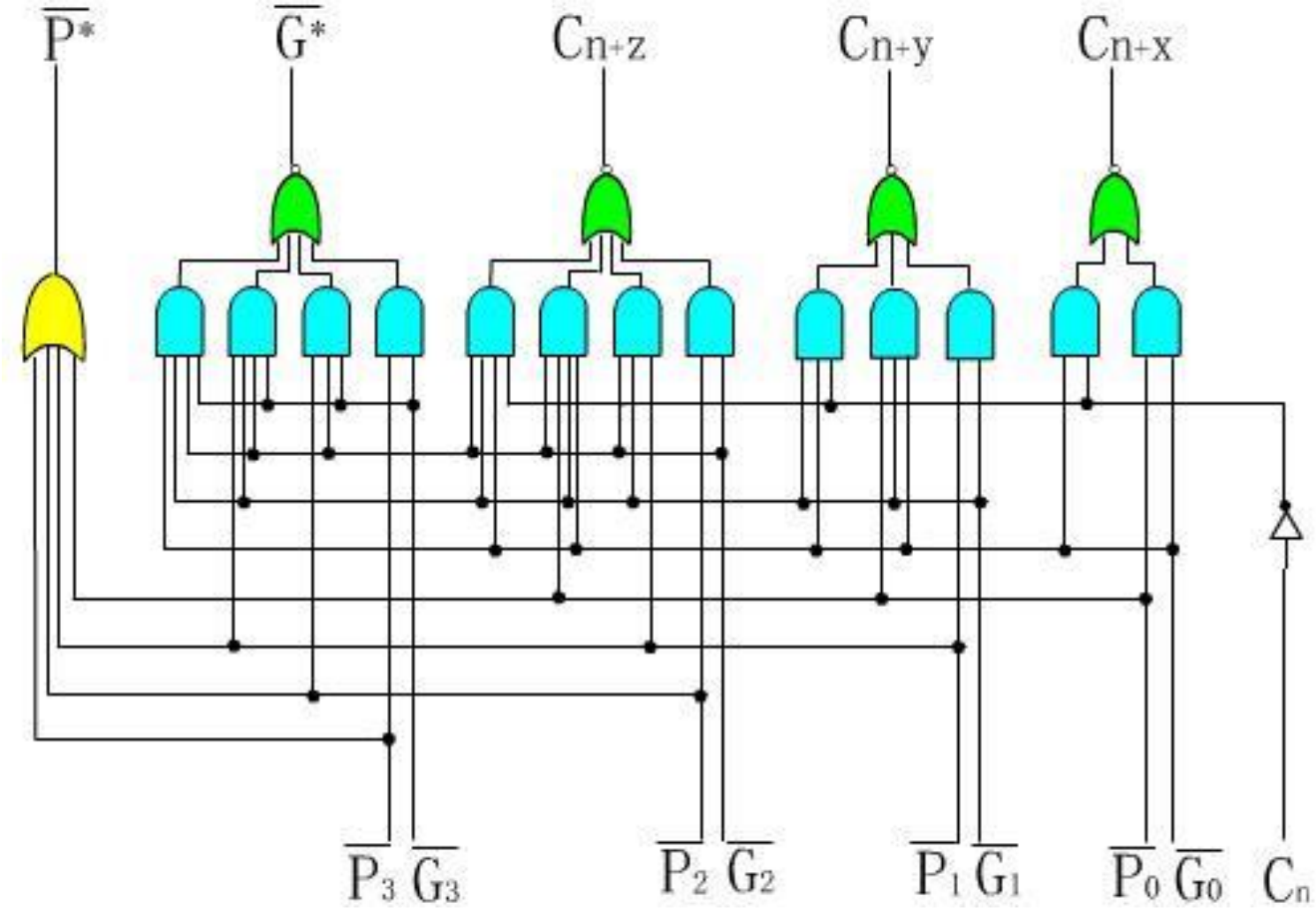
$=G_3+G_2P_3+G_1P_2P_3+G_0P_1P_2P_3+P_0P_1P_2P_3C_n$

$=G^*+P^*C_n$

G*为成组先行进位发生输出

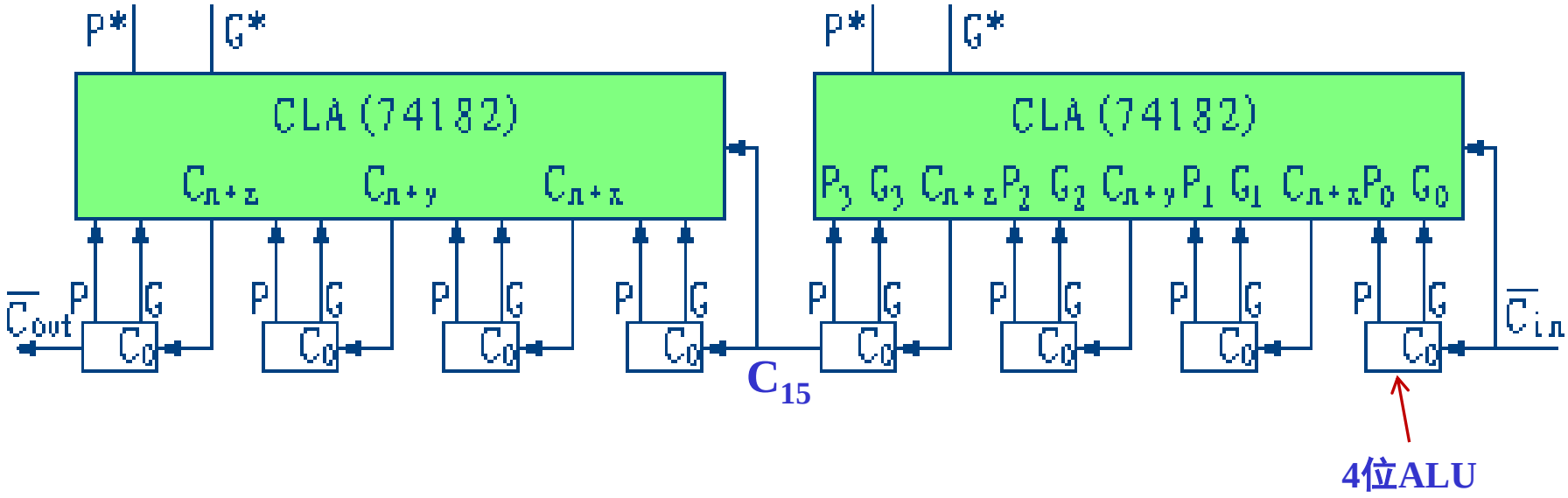
P*为成组先行进位传送输出

74182CLA的逻辑电路图



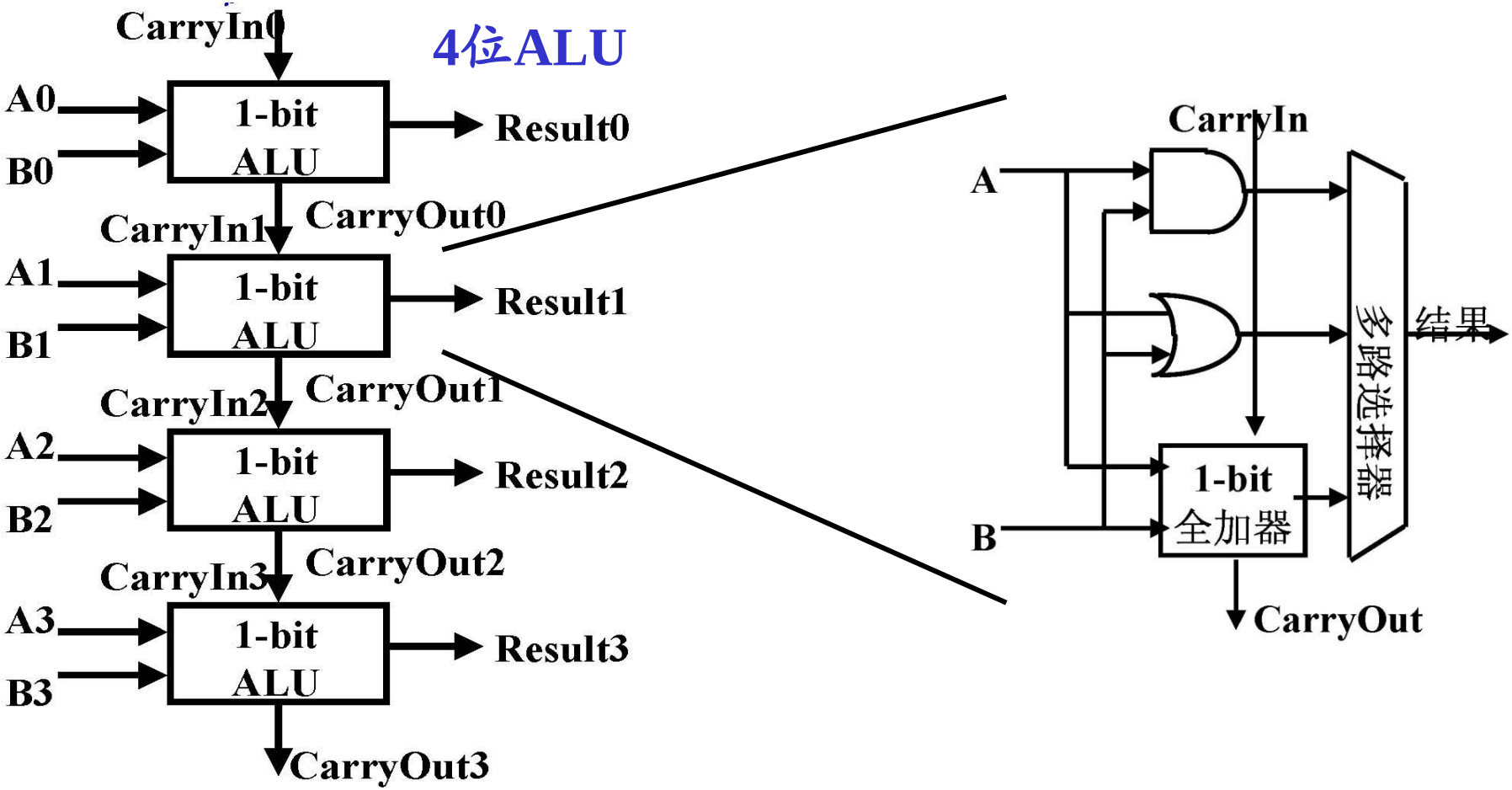
设计一个32位ALU

- 用两个16位全先行进位逻辑级联组成的32位ALU



算术/逻辑运算ALU的另一个方案

- 实现功能简单⇨逻辑电路简单



BCD码加法实例

- x=36, y=47, 求x+y

解：$[x]_{BCD}$=0011 0110, $[y]_{BCD}$=0100 0111

$[x]_{BCD}$ 0011 0110

$+[y]_{BCD}$ 0100 0111

0111 1101

校正 ＋ 0110

$[x+y]_{BCD}$ 1000 0011

所以，x+y=83

校正逻辑：BCD码完成十进制运算时，当和数大于9时，必须对和数进行加6修正

1位十进制（BCD）加法器设计

令X_i和Y_i为两个一位BCD码。

- 设S_i为BCD码的和，C_{i+1}为BCD码的进位输出，C_i为进位输入。
- 令S'_i为修正之前的二进制和，C'_{i+1}为修正之前的进位。

1）若$0 \le X_i + Y_i + C_i < 10$,则$C'_{i+1} = 0$，$0 \le S'_i \le 9$

$S_i = S'_i$， $C_{i+1} = 0$

2）若$10 \le X_i + Y_i + C_i \le 15$，则$C'_{i+1} = 0$，$10 \le S'_i \le 15$

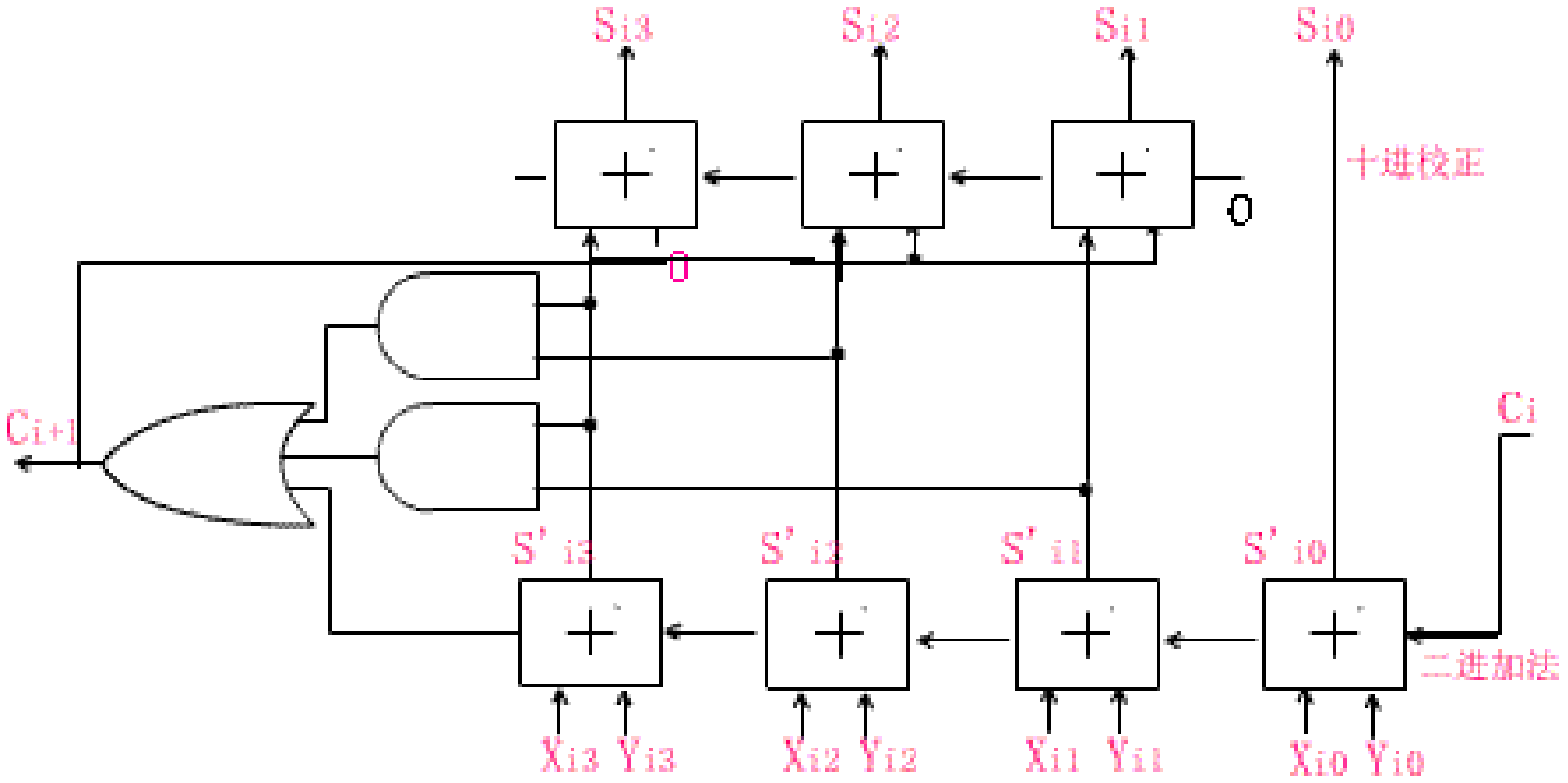
$S_i = S'_i + 6$ （mod 16），$C_{i+1} = 1$

11xx或1x1x

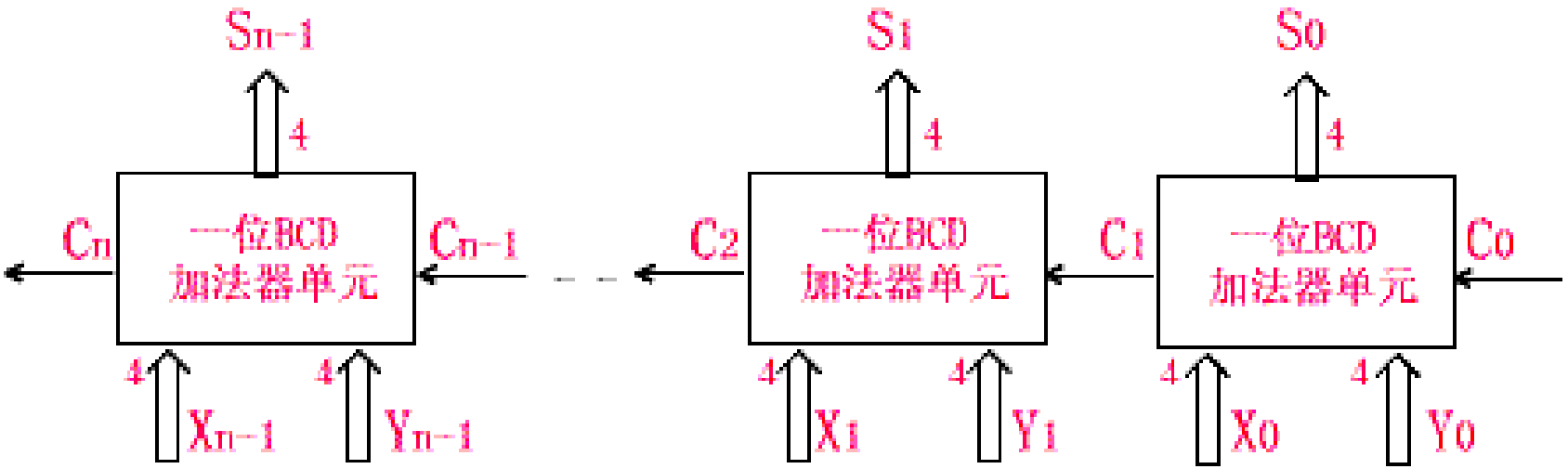
3）若$16 \le X_i + Y_i + C_i \le 19$，则$C'_{i+1} = 1$，$0 \le S'_i \le 3$

$S_i = S'_i + 6$ （mod 16），$C_{i+1} = 1$

1位十进制加法器电路

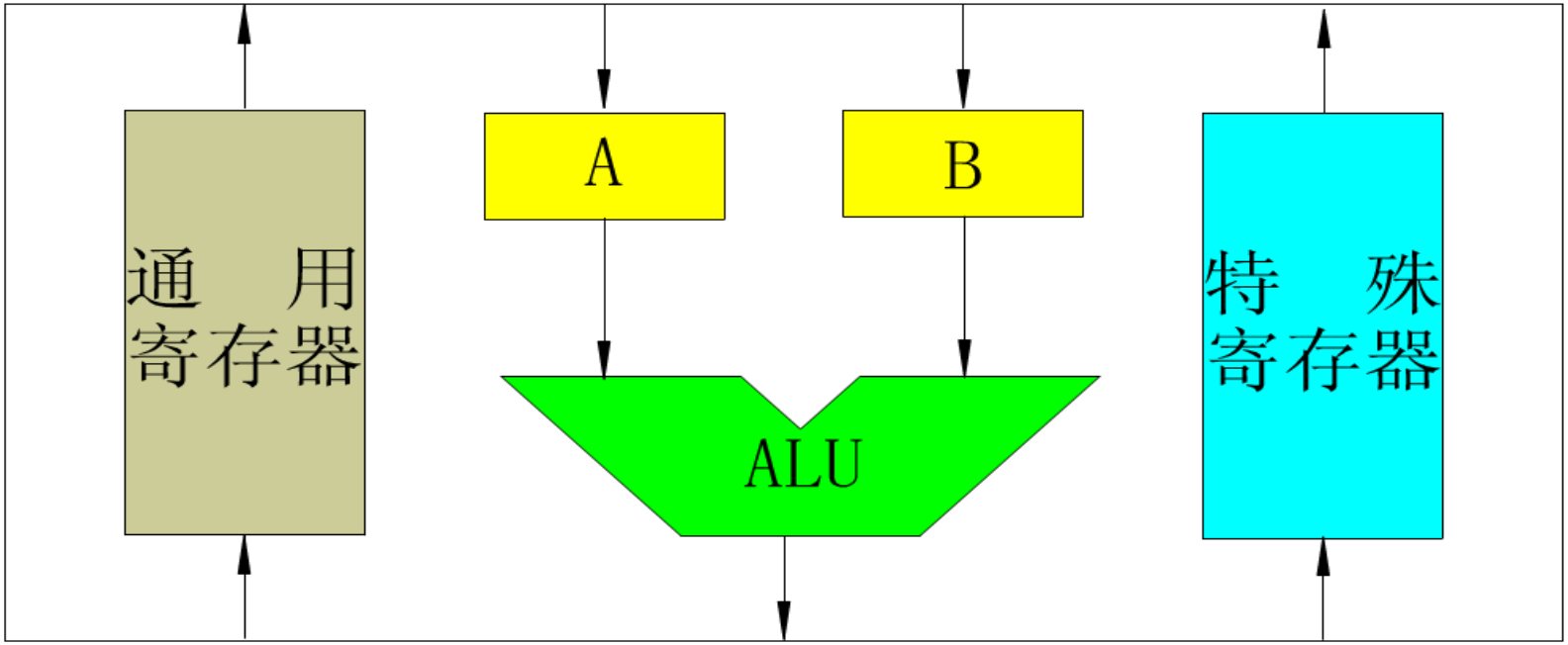

Si3
Si2
Si1
Si0
十进校正
0
0
Ci+1
Ci
S'i3
S'i2
S'i1
S'i0
二进加法
Xi3 Yi3
Xi2 Yi2
Xi1 Yi1
Xi0 Yi0

n位数的行波进位BCD加法器



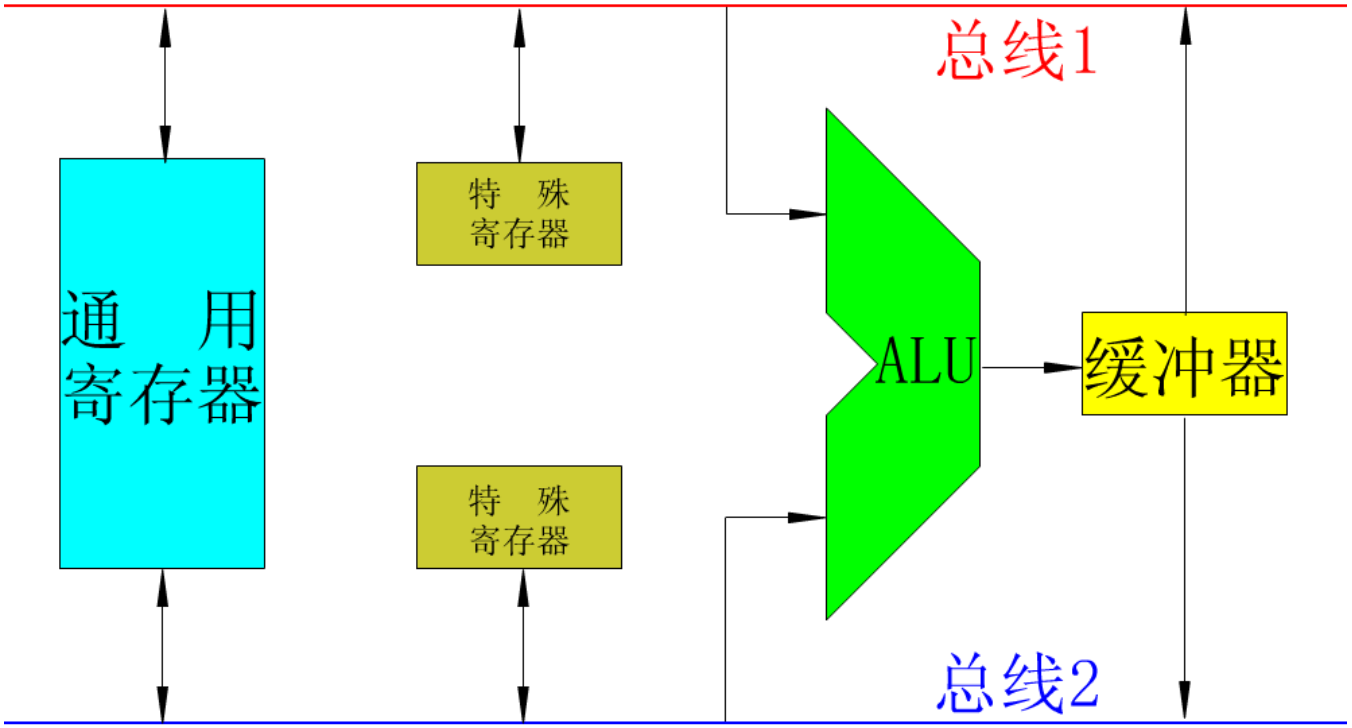
单总线结构运算器

- 所有部件都接到同一总线上
- 在某一时刻，只有一个操作数能放在总线上
- 控制电路比较简单，但操作速度较慢



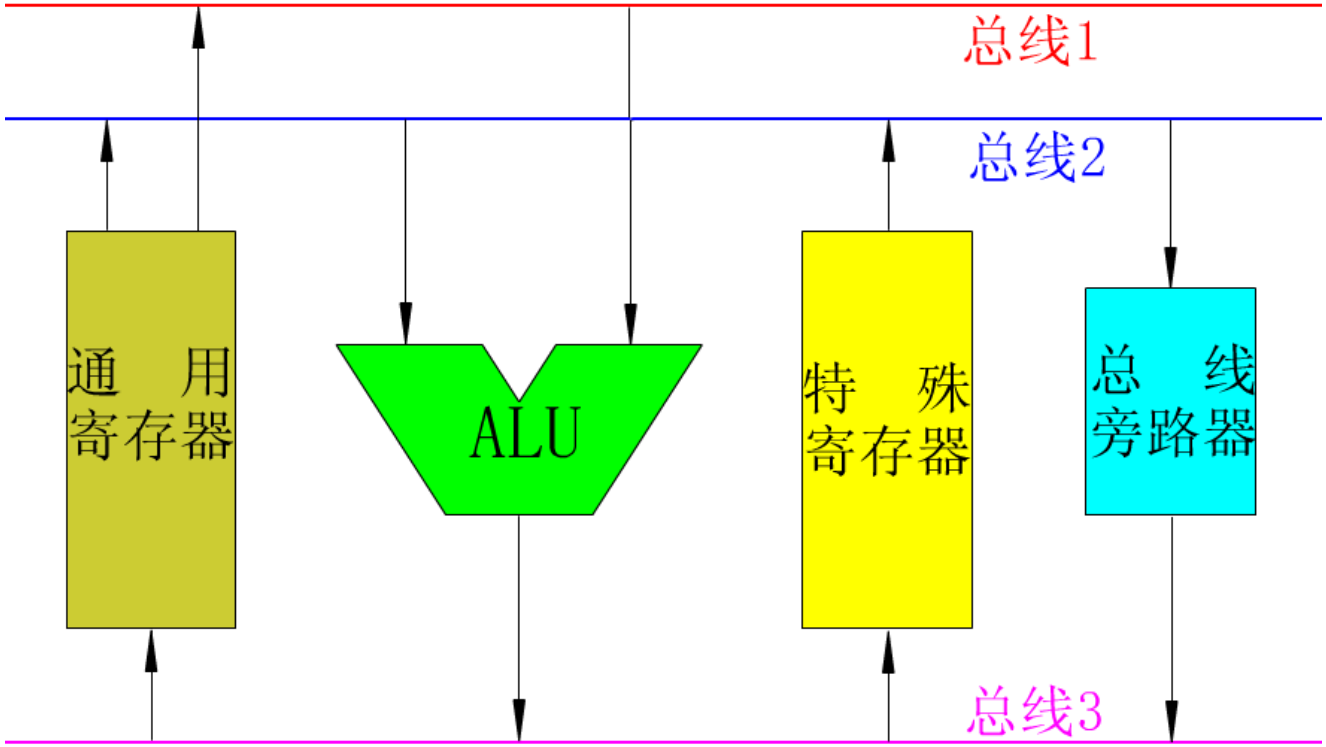
双总线结构运算器

- 两个操作数可同时加到ALU进行运算
- 特殊寄存器分为两组，它们分别与一条总线交换数据
- 控制电路稍复杂，操作速度较快



三总线结构运算器

- ALU的两个输入端各连至一条总线，而输出端连至第三条总线。算术逻辑操作可在一步控制之内完成
- 控制电路复杂，操作速度快



2.6 浮点运算方法和浮点运算器

浮点数加、减法

设有两个浮点数 $x = 2^{E_x} \cdot M_x$

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

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

- 算法
  - 检查操作数是否为0
  - 对阶，求阶差： $\Delta E = E_X - E_Y$，使阶码小的数的尾数右移 $|\Delta E|$ 位，其阶码取大的阶码值
  - 对尾数进行加、减法，求得结果
  - 规格化，舍入（可能再次规格化），进行溢出检查（阶码）

对阶

- 原则：小阶向大阶看齐（思考：为什么？）
- 方法:
  - 逐位移位比较：小阶的尾数每次右移一位,其阶码加1，直到两数的阶码相等为止
  - 求阶差:
    - 右移的位数等于阶差△E＝E x －E y
    - 哪个数移位由阶差的正负决定。
      - 若△E＝0, 表示两数阶码相等;
      - 若△E>0, 表示$E_x > E_y$； 则y移位;
      - 若△E<0, 表示$E_x < E_y$； 则x移位。

结果规格化

- 左规：向左规格化
  - 当尾数为原码时，应使结果为x.1xxxxx
  - 当尾数为补码时，应使尾数的最高位与符号位相反。即：若符号位与最高位相等，则应逐次使尾数左移，低位补0，并使阶码减1，直到符合规定。
  - 而对IEEE754浮点格式，应使尾数变为1.M形式
- 右规：向右规格化
  - 若尾数求和的结果为01.x…x或10.x…x, 应将运算结果右移以实现规格化表示
  - 规则：尾数右移1位,阶码加1

IEEE754的舍入处理

四种舍入（Rounding）方式：

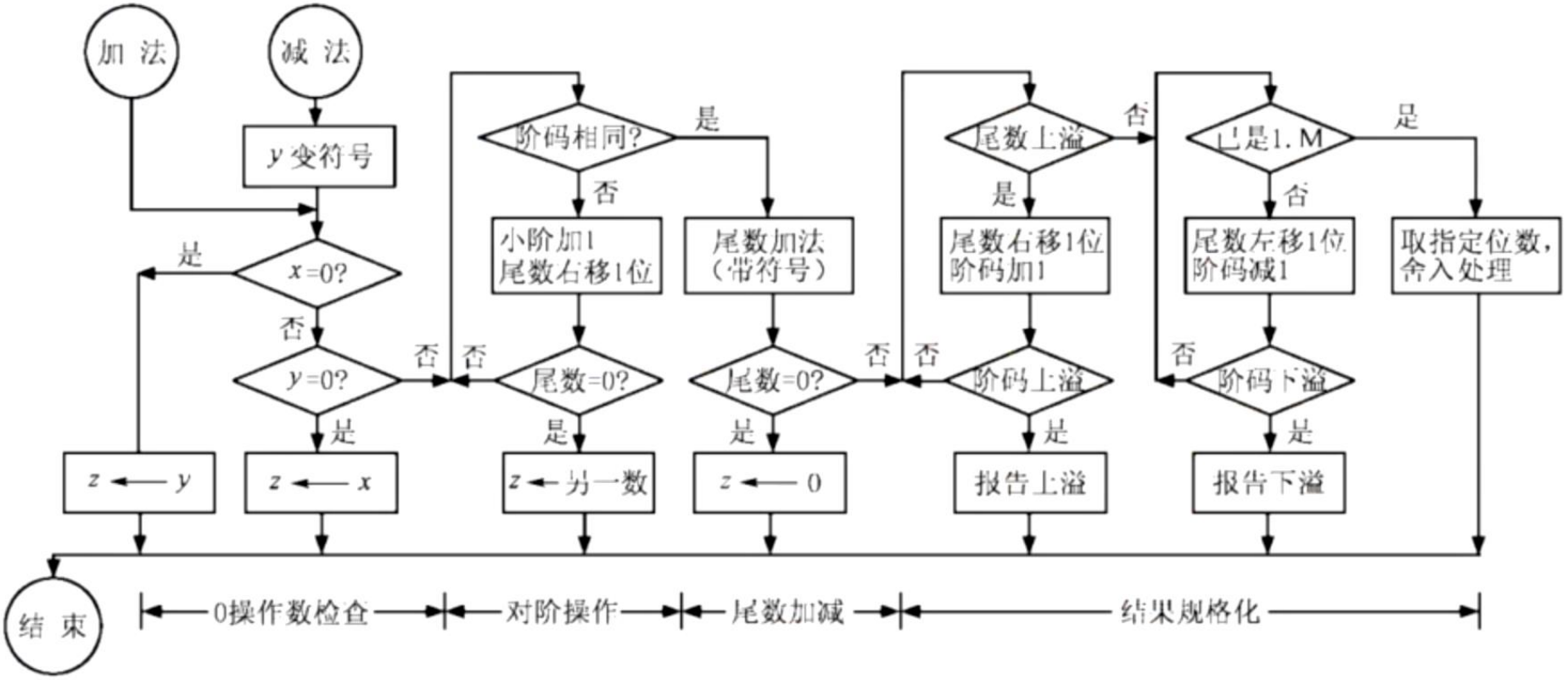
- 就近舍入（向偶数舍入）：例如，尾数超出规定的23位的多余位是10010，多余位的值超过规定的最低有效位值的一半，则最低有效为增1。若多余的是01111，则简单截尾即可。对于10000这种特殊情况，则进一步判断最低有效位，若为0，则舍去；否则，向最低有效位进位，最终结果为偶数。
- 朝0舍入：即朝数轴原点方向舍入，就是把多余的位数简单的截去。截尾使取值的绝对值比原值的绝对值小。
- 朝＋∞舍入：对于正数，当多余位不全为0则向最低有效位进1；而对负数则是简单的截尾。
- 朝－∞舍入：处理方法正好与朝＋∞舍入情况相反。对于正数，当多余位不全为0则简单截尾；而对负数，则向最低有效位进1。

溢出判断和处理

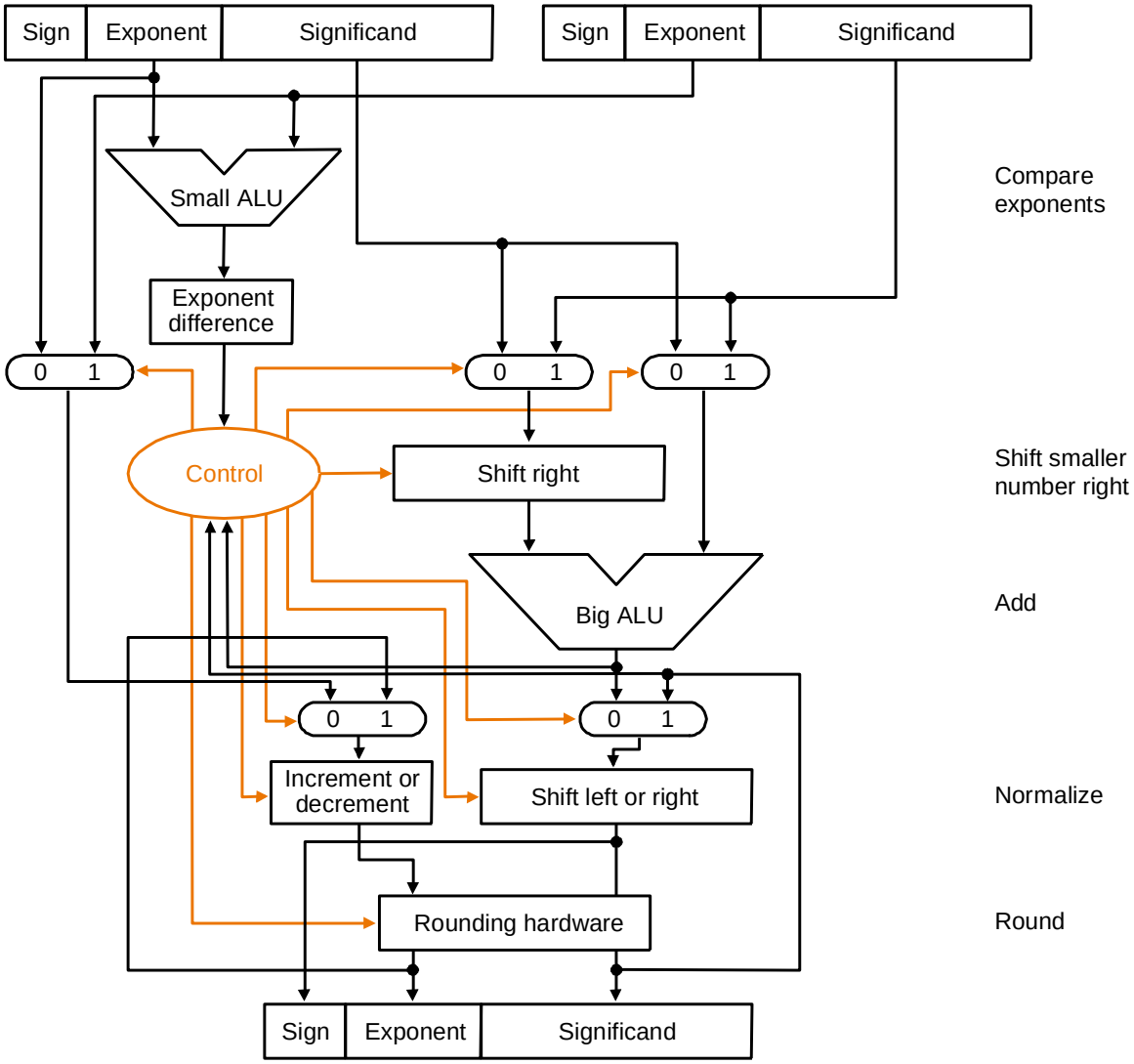
- 阶码上溢，一般将其认为是＋∞和－∞
- 阶码下溢，则数值为0
- 尾数上溢，两个同符号位的数相加。处理方法是尾数右移，阶码加1
- 尾数下溢。尾数右移时，最低位从最右端流出。要进行舍入处理

浮点加法运算操作流程

$$(a+b)+c=?\ a+(b+c)$$



浮点运算部件


Sign
Exponent
Significand
Sign
Exponent
Significand
Small ALU
Exponent difference
0 1
0 1
0 1
Control
Shift right
Big ALU
0 1
0 1
Increment or decrement
Shift left or right
Rounding hardware
Sign
Exponent
Significand
Compare exponents
Shift smaller number right
Add
Normalize
Round

例(1)

$x=2^2\times0.11011011$，$y=-2^4\times0.10101100$

1、检查操作数不为0

2、对阶：阶码对齐后才能加减。规则是阶码小的向阶码大的数对齐；

- 阶差=E_x-E_y=00 010 - 00 100=11110
- 即阶差为-2，M_x右移两位，E_x加2
- x=00100 , 0.00110110（11）

3、尾数相加

00.00110110(11)+11.01010100

=11.10001010(11)

补码

例(2)

4、结果规格化

右规，阶码加1；左规，阶码减1

左规为11.00010101（10），阶码减1为00011

5、舍入（0舍1入） 11.00010110

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

浮点数乘、除法

设有两个浮点数 $x = 2^{E_x} \times M_x$

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

- $x \times y = 2^{(E_x + E_y)} \times (M_x \times M_y)$

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

- 算法
  - 检查操作数是否为0
  - 阶码加、减：乘：$E_X + E_Y$，除：$E_X - E_Y$
  - 对尾数进行乘、除法，求得结果
  - 规格化、舍入处理（可能再次规格化）
  - 进行溢出检查（阶码）

尾数处理

方法1：截断

方法2：舍入–按某种规则进行修正

- 如果尾数用原码表示
  - ① 只要尾数最低为1或者移出位中有1数值位，使最低位置1
  - ② 0舍1入
- 如果尾数用补码表示（第五版教材中未列出）

  丢失的各位均为0，不必舍入；

  丢失的最高位为0，以后各位不全为0时；或者最高为1，以后各位全为0时，不必舍入；

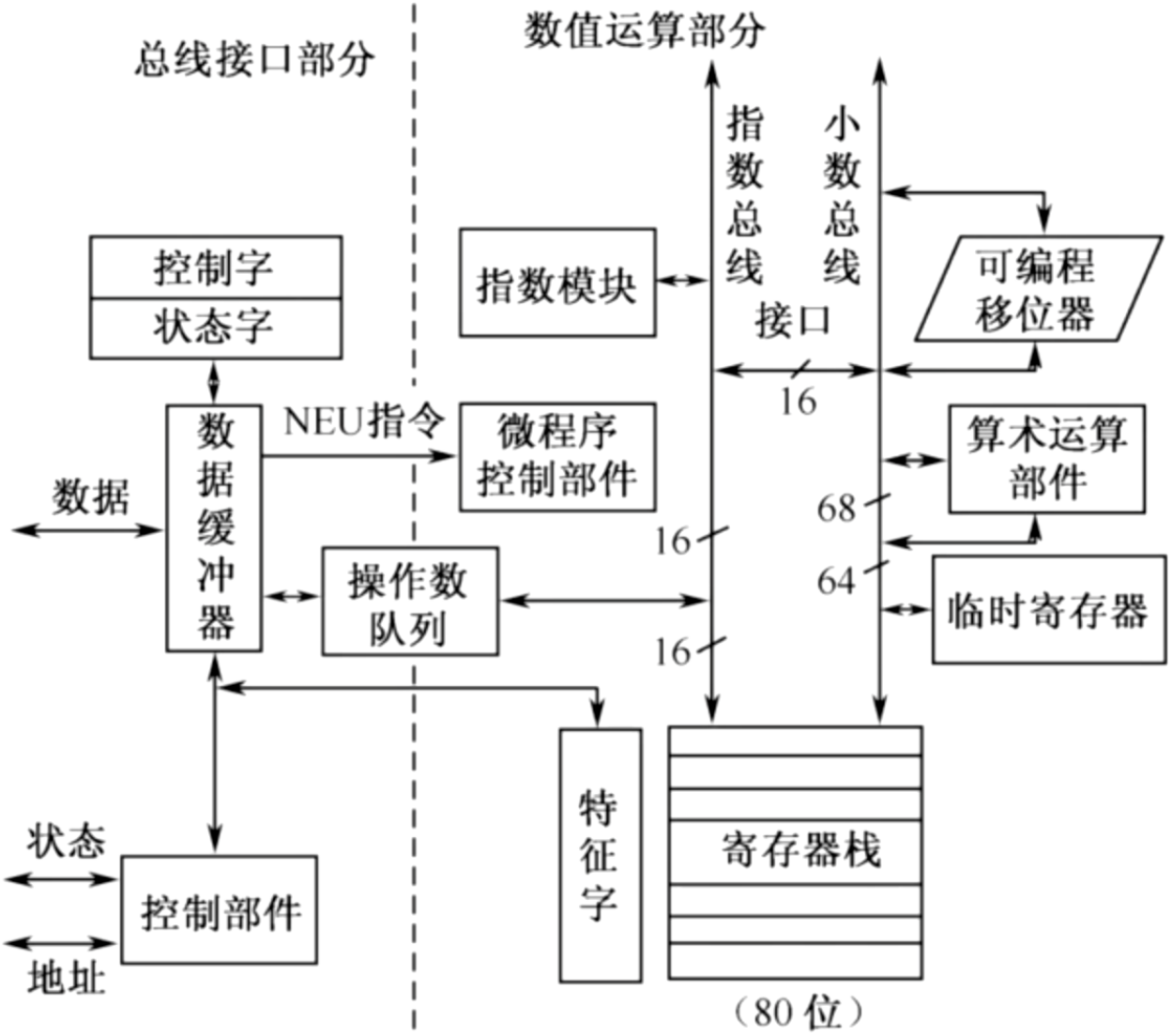
  丢失的最高位为1，以后各位不全为0时，则在尾数的最低位进1的修正操作。

浮点运算器实例

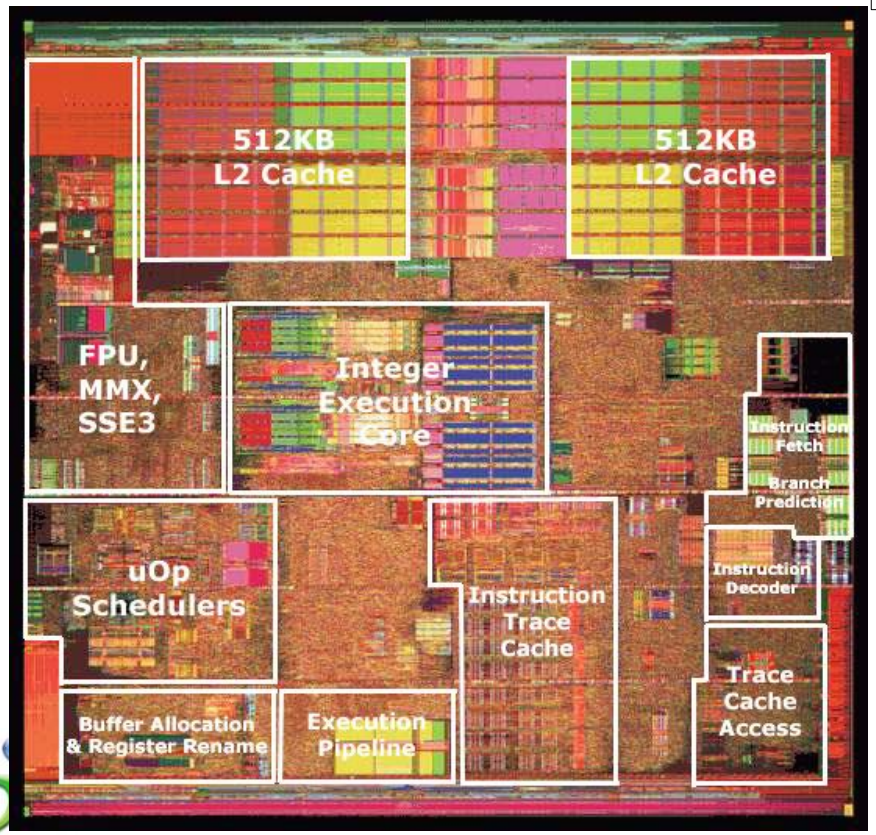
- CPU之外的浮点运算器（数学协处理器）如80x87
  - 完成浮点运算功能，不能单独使用
  - 协处理器：配合80386或80286（80x86）异步并行工作
  - 使用特殊的80位的扩展精度格式。有8个80位的寄存器组。754标准单精度和双精度数从存储器加载到浮点寄存器中时转化成这种格式
- CPU之内的浮点运算器（486DX以上）

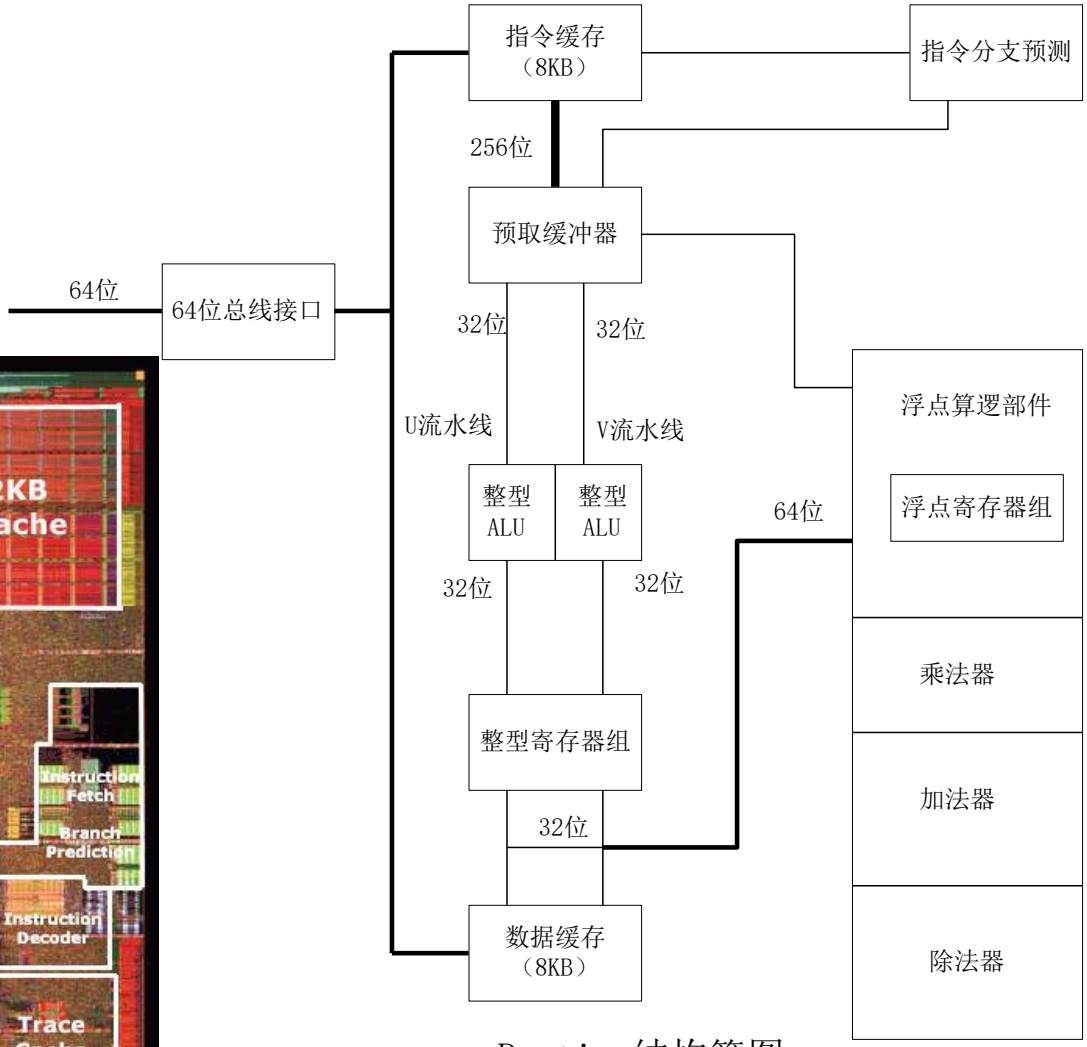
  较新的Intel处理器提供对754标准的直接硬件支持

80x87浮点运算器逻辑框图



Pentium结构简图





Pentium结构简图

本章小结

- 运算器功能：处理数据
- 数据表示：原码、反码、补码、移码
- 功能实现：加、减、乘、除
- 运算器的构成
- 浮点数的表示及运算
- 浮点运算器