《计算机组成原理》

（第三讲习题答案）

厦门大学信息学院软件工程系 曾文华

2023年3月27日

第3章　运算方法与运算器

- 3.1　计算机中的运算
- 3.2　定点加减法运算
- 3.3　定点乘法运算
- 3.4　定点除法运算
- 3.5　浮点运算
- 3.6　运算器

习题（P92–94）

- 3.1
- 3.2
- 3.4（3）
- 3.5（3）
- 3.6（2）
- 3.7（2）
- 3.8（2）
- 3.9（2）
- 3.10（2）
- 3.11

- 例3.1：x=0.1010，y=0.0101，求$[x+y]_{补}$
- 解：$[x]_{补}$=0.1010，$[y]_{补}$=0.0101；$[x+y]_{补}$= $[x]_{补}$+$[y]_{补}$

$$
\begin{array}{lr}
[x]_{补} & 0.1010 \\
+\ [y]_{补} & 0.0101 \\
\hline
[x+y]_{补} & 0.1111
\end{array}
$$

- $[x+y]_{补}$=0.1111

- 例3.2：x=-0.1010，y=-0.0100，求$[x+y]_{补}$和 x+y
- 解： $[x]_{补}$=1.0110，$[y]_{补}$=1.1100；$[x+y]_{补}$= $[x]_{补}$+$[y]_{补}$

$$
\begin{array}{rr}
[x]_{补} & 1.0110 \\
+\ [y]_{补} & 1.1100 \\
\hline
[x+y]_{补} & 11.0010
\end{array}
$$

最高位1去掉

取反加1

1010取反得到0101，加1得到0110

0100取反得到1011，加1得到1100

- $[x+y]_{补}$=1.0010　　x+y=-0.1110

取反加1

0010取反得到1101，加1得到1110

- **例3.3：$x=0.1001$，$y=0.0110$，求$[x-y]_{补}$**
- **解：$[x]_{补}=0.1001$，$[y]_{补}=0.0110$，$[-y]_{补}=1.1010$；$[x-y]_{补}= [x]_{补}+[-y]_{补}$**

$$\begin{array}{lr} [x]_{补} & 0.1001 \\ +\ [-y]_{补} & 1.1010 \\ \hline [x-y]_{补} & 10.0011 \end{array}$$

取反加1

0110取反得到1001，加1得到1010

- **$[x-y]_{补}=0.0011$**

最高位1去掉

- 例3.4：x=-0.1001，y=-0.0110，求$[x-y]_{补}$和 x-y
- 解： $[x]_{补}$=1.0111，$[y]_{补}$=1.1010，$[-y]_{补}$=0.0110；$[x-y]_{补}$= $[x]_{补}$+$[-y]_{补}$

$$
\begin{array}{rl}
[x]_{补} & 1.0111 \\
+\ [-y]_{补} & 0.0110 \\
\hline
[x-y]_{补} & 1.1101
\end{array}
$$

取反加1

1001取反得到0110，加1得到0111
0110取反得到1001，加1得到1010

- $[x-y]_{补}$=1.1101　　x-y = -0.0011

取反加1

1101取反得到0010，加1得到0011

- 例3.5：（1）设$[x]_{补}$=0.1011，$[y]_{补}$=0.1100， 求$[x+y]_{补}$

  （2）设$[x]_{补}$=1.0101，$[y]_{补}$=1.0100， 求$[x+y]_{补}$
- 解：

$$\begin{array}{lr} & [x]_{补} \quad 0.1011 \\ + & [y]_{补} \quad 0.1100 \\ \hline & [x+y]_{补} \quad 1.0111 \end{array}$$

$$\begin{array}{lr} & [x]_{补} \quad 1.0101 \\ + & [y]_{补} \quad 1.0100 \\ \hline & [x+y]_{补} \quad 10.1001 \end{array}$$

最高位1去掉

- 第（1）种情况：两个正数相加，结果为负数（1.0111），出错
- 第（2）种情况：两个负数相加，结果为正数（0.1001），出错

- 原因：
  - 第（1）种情况，x+y=11/16+12/16=23/16的值大于等于1，超出了定点小数的表示范围，发生溢出
  - 第（2）种情况，x+y=(-11/16)+(-12/16)=(-23/16)的值小于-1，也超出了定点小数的表示范围，发生溢出
  - 补码定点小数表示范围：[-1,1)

- 例3.6：x=-0.1011，y=0.1100，求$[x-y]_{补}$
- 解： $[x]_{补}$=1.0101，$[y]_{补}$=0.1100，$[-y]_{补}$=1.0100

$$
\begin{array}{lr}
 [x]_{补} & 1.0101 \\
+\ [-y]_{补} & 1.0100 \\
\hline
 [x-y]_{补} & 10.1001
\end{array}
$$

- 负数-正数=负数+负数，结果是正数，溢出了

- 例3.7：x=-111，y=110，求$[x-y]_{补}$
- 解：$[x]_{补}=1,001$，$[y]_{补}=0,110$，$[-y]_{补}=1,010$
- $C_f = 1$；$C_d = 0$
- 溢出标志V=1⊕0=1，溢出（-7-6=-13；超出-8～+7的范围）

$$V = C_f \oplus C_d$$

$$\begin{array}{rr}
[x]_{补} & 1,001 \\
+\ [-y]_{补} & 1,010 \\
\hline
[x-y]_{补} & 10,011
\end{array}$$

$C_f = 1$

$C_d = 0$

- 例3.8：$x=-011$，$y=100$，求$[x-y]_{补}$
- 解：$[x]_{补}=1,101$，$[y]_{补}=0,100$，$[-y]_{补}=1,110$

$$V = C_f \oplus C_d$$

- $C_f = 1$；$C_d = 1$
- 溢出标志$V=1 \oplus 1=0$，没有溢出（-3-4=-7；没有超出-8～+7的范围）

$$\begin{array}{rr} [x]_{补} & 1,101 \\ +\ [-y]_{补} & 1,100 \\ \hline [x-y]_{补} & 11,001 \end{array}$$

$C_f = 1$ $C_d = 1$

- 例3.9：（1）设$[x]_{补}=00.1011$，$[y]_{补}=00.0111$，求$[x+y]_{补}$
  （2）设$[x]_{补}=11.0101$，$[y]_{补}=11.0011$，求$[x+y]_{补}$
  （3）设$[x]_{补}=00.1011$，$[y]_{补}=11.0011$，求$[x+y]_{补}$

$$V = S_{f1} \oplus S_{f2}$$

- 解：

$$\begin{array}{rl} [x]_{补} & 00.1011 \\ +\ [y]_{补} & 00.0111 \\ \hline [x+y]_{补} & 01.0010 \end{array} \qquad \begin{array}{rl} [x]_{补} & 11.0101 \\ +\ [y]_{补} & 11.0011 \\ \hline [x+y]_{补} & 10.1000 \end{array} \qquad \begin{array}{rl} [x]_{补} & 00.1011 \\ +\ [y]_{补} & 11.0011 \\ \hline [x+y]_{补} & 11.1110 \end{array}$$

- 第（1）情况，运算结果的符号位为01，正溢出（11/16 + 7/16 = 18/16）
- 第（2）情况，运算结果的符号位为10，负溢出（-11/16 - 13/16 = -24/16）
- 第（3）情况，运算结果的符号位为11，没有溢出（11/16 - 13/16 = -2/16）

- 例3.10：x = 0.1101， y = -0.1011， 用原码一位乘法求$[x \cdot y]_{原}$
- 解：
  - 乘积的符号$P_0 = x_0 \oplus y_0 = 1$
  - 乘积的绝对值=0.1000 1111（具体见下面的计算过程）
  - 乘积：$[x \cdot y]_{原}$=1.1000 1111， x·y = -0.1000 1111

	部分积	乘数|y|	说明
	00.0000	1011	P=0
+	00.1101		y_n=1， +|x|
	00.1101	1011	
→	00.0110	1101	逻辑右移一位
+	00.1101		y_n=1， +|x|
	01.0011	1101	
→	00.1001	1110	逻辑右移一位
+	00.0000		y_n=0， +0
	00.1001	1110	
→	00.0100	1111	逻辑右移一位
+	00.1101		y_n=1， +|x|
	01.0001	1111	
→	00.1000	1111	右移一位

乘数|y|为y（-0.1011）的数值位（1011），部分积为0（双符号位小数）

根据|y|的最后一位是1还是0，执行+|x|或+0的操作

- 例3.11：$[x]_{补}$=1.0111，$[y]_{补}$=1.0011，用补码一位乘法求$[x \cdot y]_{补}$
- 解：
  - $[-x]_{补}$ = 0.1000 + 0.0001 = 0.1001（取反加1）
  - $[x \cdot y]_{补}$ = 0.0111 0101（具体见下面的计算过程）
  - 验证：x = -0.1001，y = -0.1101，x · y = 0.0111 0101

Booth算法

	部分积	乘数y	说明
	00.0000	100110	P=0　y_{n+1}=0
+	00.1001		$y_n y_{n+1}$=10，$+[-x]_{补}$
	00.1001	100110	
→	00.0100	110011	算术右移一位
+	00.0000		$y_n y_{n+1}$=11，+0
	00.0100	110011	
→	00.0010	011001	算术右移一位
+	11.0111		$y_n y_{n+1}$=01，$+[x]_{补}$
	11.1001	011001	
→	11.1100	101100	算术右移一位
+	00.0000		$y_n y_{n+1}$=00，+0
	11.1100	101100	
→	11.1110	010110	算术右移一位
+	00.1001		$y_n y_{n+1}$=10，$+[-x]_{补}$
	00.0111	0101	最后一步不移位

乘数y为补码值（1.0011），去掉小数点（10011），最后增加一位y_{n+1}=0（100110）；部分积为0（双符号位小数）

根据y的最后二位是01、10、00或11，执行$+[x]_{补}$、$+[-x]_{补}$、+0的操作

- **例3.12：$[x]_{原}$=1.1001，$[y]_{原}$=0.1011，求$[x÷y]_{原}$**

- **解：**
  - **|x|=0.1001，|y|= 0.1011，$[-|y|]_{补}$ = 1.0101**
  - 计算过程见下页
  - 商的符号=x的符号⊕y的符号=1⊕0=1
  - 余数的符号=x的符号=1
  - 商|Q|=0.1101，$[Q]_{原}$=1.1101
  - 余数|R|=0.0000 0001，$[R]_{原}$=1.0000 0001
  - 验证：
    - » **x = -9/16，y = 11/16**
    - » **商Q = -0.1101 = -13/16，余数R = -0.0000 0001 = -1/256**
    - » **x = Q·y+R = (-13/16)·(11/16)+(-1/256) = -144/256 = - 9/16**

原码恢复余数法

	余数R	商Q	说明
	00.1001	0.0000	初始余数R=$\vert x\vert$，商Q=0
$+[-\vert y\vert]_{补}$	11.0101		减$\vert y\vert$
	11.1110	0.0000	余数为负，商0
$+\vert y\vert$	00.1011		加$\vert y\vert$恢复余数
	00.1001		
←	01.0010	0.000	左移1位
$+[-\vert y\vert]_{补}$	11.0101		减$\vert y\vert$
	00.0111	0.0001	余数为正，商1
←	00.1110	0.001	左移1位
$+[-\vert y\vert]_{补}$	11.0101		减$\vert y\vert$
	00.0011	0.0011	余数为正，商1
←	01.0110	0.011	左移1位
$+[-\vert y\vert]_{补}$	11.0101		减$\vert y\vert$
	11.1011	0.0110	余数为负，商0
$+\vert y\vert$	00.1011		加$\vert y\vert$恢复余数
	00.0110		
←	00.1100	0.110	左移1位
$+[-\vert y\vert]_{补}$	11.0101		减$\vert y\vert$
	00.0001	0.1101	余数为正，商1

余数|R|=0.0000 0001　　商|Q|=0.1101

余数R的初始值为x的绝对值（双符号位小数）

余数为负，商0，加|y|恢复余数，左移1位，减|y|

余数为正，商1，左移1位，减|y|

- **例3.13： $[x]_{原}=1.1001$， $[y]_{原}=0.1011$， 求$[x \div y]_{原}$**
  - 计算过程见下页
  - 运算结果中的**C**为进位标志，并且商位与进位相同，**q=C**
  - 运算结果同例**3.12**

- 不恢复余数法的规则：
  - 余数为正，商**1**，余数左移**1**位，减去除数
  - 余数为负，商**0**，余数左移**1**位，加上除数

- 一会儿做加法运算，一会儿做减法运算，也称为加减交替法

原码不恢复余数法

	C	余数R	商Q	说明
		0 0. 1 0 0 1	0. 0 0 0 0	初始余数R=|x|，商Q=0
+[-|y|]补		1 1. 0 1 0 1		减|y|
	0	1 1. 1 1 1 0	0. 0 0 0 0	余数为负，商0
←		1 1. 1 1 0 0	0. 0 0 0	左移1位
+|y|		0 0. 1 0 1 1		加|y|
	1	0 0. 0 1 1 1	0. 0 0 0 1	余数为正，商1
←		0 0. 1 1 1 0	0. 0 0 1	左移1位
+[-|y|]补		1 1. 0 1 0 1		减|y|
	1	0 0. 0 0 1 1	0. 0 0 1 1	余数为正，商1
←		0 0. 0 1 1 0	0. 0 1 1	左移1位
+[-|y|]补		1 1. 0 1 0 1		减|y|
	0	1 1. 1 0 1 1	0. 0 1 1 0	余数为负，商0
←		1 1. 0 1 1 0	0. 1 1 0	左移1位
+|y|		0 0. 1 0 1 1		加|y|
	1	0 0. 0 0 0 1	0. 1 1 0 1	余数为正，商1

余数|R|=0.0000 0001　　商|Q|=0.1101

余数R的初始值为x的绝对值（双符号位小数）

余数为负，商0，左移1位，加|y|

余数为正，商1，左移1位，减|y|

- 例3.14：设$x=2^{-101}\cdot(-0.101011)$，$y= 2^{-010}\cdot(0.001110)$，数的阶码为3位，尾数为6位（均不含符号位），且都用补码表示，按照补码浮点数运算步骤计算x+y
- 解：
  - 首先用补码形式表示浮点数x和y（采用双符号位）
    - $[x]_{补}$=11 011，11.010101　　　　$[y]_{补}$=11 110，00.001110
  - （1）对阶：小阶向大阶对齐，x（阶码=-5）向y（阶码=-2）对齐，x的尾数右移3位
    - 对阶后的$[x]_{补}$=11 110，11.111010(101)
  - （2）尾数运算（求和）：

$[x]_{补}$	11 110，	11.111010 (101)
+ $[y]_{补}$	11 110，	00.001110
$[x+y]_{补}$	11 110，	00.001000 (101)

  - （3）尾数规格化处理：运算结果的尾数=00.001000(101)，为非规格化数（尾数的绝对值太小），需要左规，左移2次，变为规格化数00.100010(1)，阶码减2，阶码变为11 100
  - （4）舍入处理：舍入处理后的尾数为：00.100011（0舍1入法，末位恒置1法）
  - （5）溢出判断：阶码的双符号位相同（11），没有溢出
  - 运算结果：$[x+y]_{补}$ =11 100，00.100011，$x+y= 2^{-100}\cdot(0.100011)$
  - 验证：
    - x=(1/32)*(-43/64)，y=(1/4)*(14/64)，x+y=(-43/2048)+(112/2048)=69/2048
    - $x+y= 2^{-100}\cdot(0.100011)$=(1/16)*(35/64)=70/2048，相差1/2048，原因是舍入处理引起的误差

- 例3.15：设$x=2^7\cdot(25/32)$，$y= 2^6\cdot(-23/32)$，数的阶码为5位，尾数为7位（均含2位符号位），按照补码浮点数运算步骤计算x-y（采用0舍1入法）
- 解：
  - 首先用补码形式表示浮点数x和y（采用双符号位）
    - $x= 2^{111}\cdot(0.11001)$　　$[x]_{补}$ =00 111，00.11001
    - $y= 2^{110}\cdot(-0.10111)$　　$[-y]_{补}$=00 110，00.10111
  - （1）对阶：小阶向大阶对齐，y（阶码=6）向x（阶码=7）对齐，y的尾数右移1位
    - 对阶后的$[-y]_{补}$=00 111，00.01011(1)
  - （2）尾数运算（求和）：

$[x]_{补}$	00 111，00.11001
+ $[-y]_{补}$	00 111，00.01011(1)
$[x-y]_{补}$	00 111，01.00100(1)

  - （3）尾数规格化处理：运算结果的尾数=01.00100(1)，为非规格化数（尾数的绝对值太大），需要右规，右移1次，变为规格化数00.10010(01)，阶码加1，阶码变为01 000
  - （4）舍入处理：舍入处理后的尾数为：00.10010（0舍1入法）
  - （5）溢出判断：阶码的双符号位为01，故发生溢出
  - 运算结果：$[x-y]_{补}$ =01 000，00.10010
  - 验证：
    - x=(128)*(25/32)=100，y=(64)*(-23/32)=-46，x-y=100-(-46)=146（超出-128～+127范围）
    - $x-y= 2^8\cdot(18/32)=144$，误差为2，是舍入处理导致的，误差为2^8*(0.0000001)=2

- 例3.16：两个IEEE754单精度浮点数机器码X=00C0 0000H，Y=0080 0000H，求X-Y
- 解：
  - IEEE754单精度浮点数：32位

32浮点数 *float*

$S_{(1bit)}$	$E_{(23\sim30共8bit)}$	$M_{(0\sim22共23bit)}$

  - X=0000 0000 1100 0000 0000 0000 0000 0000=0 00000001 10000000000000000000000=$1.1*2^{-126}$
  - Y=0000 0000 1000 0000 0000 0000 0000 0000=0 00000001 00000000000000000000000=$1.0*2^{-126}$
  - E=阶码的真值+127，阶码的真值=E-127=1-127=-126；尾数=1.M
  - （1）对阶：X、Y的阶码相同，不需要对阶
  - （2）尾数相减：1.1-1.0=0.1
  - （3）结果规格化：尾数相减后的0.1为非规格化数，需要左移1次，尾数变为1.0，阶码减1，阶码变为00000000（全0）
  - （4）舍入处理：没有舍入处理
  - （5）溢出判断：阶码全0时为非规格化数据，表示发生了规格化下溢
  - IEEE754单精度非规格化浮点数：$(-1)^S*0.M*2^{-126}$，这里：S=0，E=0，M=1
  - X-Y=$0.1*2^{-126}=2^{-127}$=0 00000000 10000000000000000000000=0040 0000H

习题 3

3.1　解释下列名词。

全加器　半加器　进位生成函数　进位传递函数　算术移位　逻辑移位　阵列乘法器　原码恢复余数除法　原码不恢复余数法　阵列除法　串行进位　先行进位　对阶　规格化　保留附加位

3.2　选择题（考研真题）。

（1）[2009] 一个 C 语言程序在一台 32 位机器上运行，程序中定义了 3 个变量 x、y、z，其中 x 和 z 是 int 型，y 为 short 型。当 $x=127$，$y=-9$ 时，执行赋值语句 $z=x+y$ 后，x、y、z 的值分别是 ______。

A. x = 0000007FH，y = FFF9H，z = 00000076H

B. x = 0000007FH，y = FFF9H，z = FFFF0076H

C. x = 0000007FH，y = FFF7H，z = FFFF0076H

D. x = 0000007FH，y = FFF7H，z = 00000076H

（2）[2010] 假定有 4 个整数用 8 位补码分别表示 r_1= FEH，r_2= F2H，r_3= 90H，r_4= F8H，若将运算结果存放在一个 8 位的寄存器中，则下列运算会发生溢出的是 _______。

A. $r_1 \times r_2$　　B. $r_2 \times r_3$　　C. $r_1 \times r_4$　　D. $r_2 \times r_4$

（3）[2013] 某字长为 8 位的计算机中，已知整型变量 x、y 的机器数分别为 $[x]_{补}=11110100$，$[y]_{补}=10110000$。若整型变量 $z=2\times x+y/2$，则 z 的机器数为 _______。

A. 11000000　　B. 00100100　　C. 10101010　　D. 溢出

（4）[2018] 假定带符号整数采用补码表示，若 int 型变量 x 和 y 的机器数分别是 FFFF FFDFH 和 0000 0041H，则 x、y 的值以及 $x-y$ 的机器数分别是 _______。

A. $x=-65$，$y=41$，$x-y$ 的机器数溢出

B. $x=-33$，$y=65$，$x-y$ 的机器数为 FFFF FF9DH

C. $x=-33$，$y=65$，$x-y$ 的机器数为 FFFF FF9EH

D. $x=-65$，$y=41$，$x-y$ 的机器数为 FFFF FF96H

（5）[2018] 整数 x 的机器数为 1101 1000，分别对 x 进行逻辑右移 1 位和算术右移 1 位操作，得到的机器数各是 _______。

A. 1110 1100、1110 1100　　B. 0110 1100、1110 1100

C. 1110 1100、0110 1100　　D. 0110 1100、0110 1100

（6）[2009] 浮点数加减运算过程一般包括对阶、尾数运算、规格化、舍入和判断溢出等步骤。设浮点数的阶码和尾数均采用补码表示，且位数分别为 5 位和 7 位（均含 2 位符号位）。若有两个数 $X=2^7\times 29/32$，$Y=2^5\times 5/8$，则用浮点加法计算 $X+Y$ 的最终结果是 _______。

A. 001111100010　　B. 001110100010　　C. 010000010001　　D. 发生溢出

（7）[2015] 下列有关浮点数加减运算的叙述中，正确的是 _______。

Ⅰ. 对阶操作不会引起阶码上溢或下溢　　Ⅱ. 右规和尾数舍入都可能引起阶码上溢

Ⅲ. 左规时可能引起阶码下溢　　Ⅳ. 尾数溢出时结果不一定溢出

A. 仅Ⅱ、Ⅲ　　B. 仅Ⅰ、Ⅱ、Ⅳ

C. 仅Ⅰ、Ⅲ、Ⅳ　　D. Ⅰ、Ⅱ、Ⅲ、Ⅳ

3.3　回答下列问题。

（1）为什么采用并行进位能提高加法器的运算速度？

（2）如何判断浮点数运算结果是否发生溢出？

（3）如何判断浮点数运算结果是否为规格化数？如果不是规格化数，如何进行规格化？

（4）为什么阵列除法器中能用 CAS 的进位 / 借位控制端作为上商的控制信号？

（5）移位运算和乘法及除法运算有何关系？

3.4　已知 x 和 y，用变形补码计算 $x+y$，并判断结果是否溢出。

（1）$x=0.11010$，$y=0.10111$。

（2）$x=0.11101$，$y=-0.10100$。

（3）$x=-0.10111$，$y=-0.11000$。

3.5　已知 x 和 y，用变形补码计算 $x-y$，并判断结果是否溢出。

（1）$x=0.11011$，$y=0.11101$。

（2）$x=0.10111$，$y=0.11110$。

（3）$x=-0.11111$，$y=-0.11001$。

3.6　用原码一位乘法计算 $x\times y$。

（1）$x=-0.11111$，$y=0.11101$。

（2）$x=-0.11010$，$y=-0.01011$。

3.7　用补码一位乘法计算 $x\times y$。

（1）$x=0.10110$，$y=-0.00011$。

（2）$x=-0.011010$，$y=-0.011101$。

3.8　用原码不恢复余数法计算 $x\div y$。

（1）$x=0.10101$，$y=0.11011$。

（2）$x=-0.10101$，$y=0.11000$。

3.9　设数的阶码为 3 位，尾数为 6 位（均不包括符号位），按机器补码浮点运算规则完成下列 $[x+y]_{补}$ 运算。

（1）$x=2^{011}\times0.100100$，$y=2^{010}\times(-0.011010)$。

（2）$x=2^{-101}\times(-0.100010)$，$y=2^{-100}\times(-0.010110)$。

3.10　采用 IEEE754 单精度浮点数格式计算下列表达式的值。

（1）$0.625+(-12.25)$　　（2）$0.625-(-12.25)$

3.11　假定在一个 8 位字长的计算机中运行如下 C 语言程序段。

```
unsigned int x=134;
unsigned int y=246;
int m=x; int n=y;
unsigned int z1=x-y;
unsigned int z2=x+y;
int k1=m-n;
int k2=m+n;
```

若编译器编译时将 8 个 8 位寄存器 R1 ～ R8 分别分配给变量 x、y、m、n、$z1$、$z2$、$k1$ 和 $k2$。请回答下列问题（提示：带符号整数用补码表示）。

（1）执行上述程序段后，寄存器 R1、R5 和 R6 中的内容分别是什么？（用十六进制表示）

（2）执行上述程序段后，变量 m 和 $k1$ 的值分别是多少？（用十进制表示）

（3）上述程序段涉及带符号整数加减、无符号整数加减运算，这 4 种运算能否利用同一个加法器及辅助电路实现？简述理由。

（4）计算机内部如何判断带符号整数加减运算的结果是否发生溢出？上述程序段中，哪些带符号整数运算语句的执行结果会发生溢出？

3.12　如果全加器采用图 3.2（b）所示的方案实现，尝试分析图 3.3 ～图 3.8 所示电路的时间延迟和成本开销，你认为与图 3.2（a）所示方案相比哪个方案更好，为什么？

实践训练

（1）在 Logisim 中构建 8 位可控加减法电路、4 位先行进位电路、4 位快速加法器、16 位组内并行、组间并行加法器。

（2）在 Logisim 中设计 5 位无符号阵列乘法器，并利用该乘法器构造补码乘法器。

（3）在 Logisim 中设计 8 位原码一位乘法和补码一位乘法器。

（4）在 Logisim 中设计能满足 MIPS 指令系统功能的 32 位多功能运算器。

习题答案（P92-94）

- **3.1：解释下列名词：**

  1. 全加器：带进位的一位加法器

  2. 半加器：没有进位输入的一位加法器

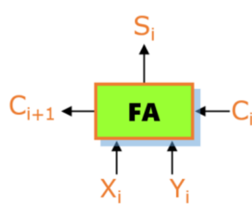
– 1、全加器（**Full Adder，FA**）

加数X_i	加数Yi	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

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

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

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



– 半加器（**Half Adder，HA**）

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

X_i	Y_i	S_i	C_{i+1}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

- **3.1：解释下列名词（续）：**

3. 进位生成函数：$G_i = X_iY_i$ 当Gi=1，Ci+1一定为1，所以称之为进位生成函数。

4. 进位传递函数 ：$P_i = X_i \oplus Y_i$, 当Pi=1时，进位输入信号Ci才能传递到进位输出Ci+1处，所以称之为进位传递函数。

5. 算术移位：对有符号数的移位称之为算术移位，包括算术左移、算术右移。

6. 逻辑移位：对于无符号数的移位称之为逻辑移位，包括逻辑左移、逻辑右移。逻辑左移和算术左移是一样的。

- **3.1：解释下列名词（续）：**

7. 阵列乘法器：采用类似手动乘法运算的方法，用大量与门阵列同时产生手动乘法中的各乘积项，同时将大量一位全加器按照手动乘法运算的需要构成全加器阵列。

8. 原码恢复余数除法：原码除法中当余数为负时，需加上除数，将其恢复成原来的余数，即为原码恢复余数法。

9. 原码不恢复余数除法：又称为加减交替法。

   当$\mathrm{R}_{\boldsymbol{i}} > \mathbf{0}$, 商上**1**，做**2** $\mathrm{R}_{\boldsymbol{i}} - \boldsymbol{y}^{*}$ 的运算。

   当$\mathrm{R}_{\boldsymbol{i}} < \mathbf{0}$, 商上**0**，做**2** $\mathrm{R}_{\boldsymbol{i}} + \boldsymbol{y}^{*}$ 的运算。

10. 阵列除法器：利用一个可控加减CAS单元组成的流水阵列，有四个输出，四个输入。P=0时，做加法，P=1时，做减法。

- **3.1：解释下列名词（续）：**

11.串行进位：并行加法器中的进位采用串行传递。

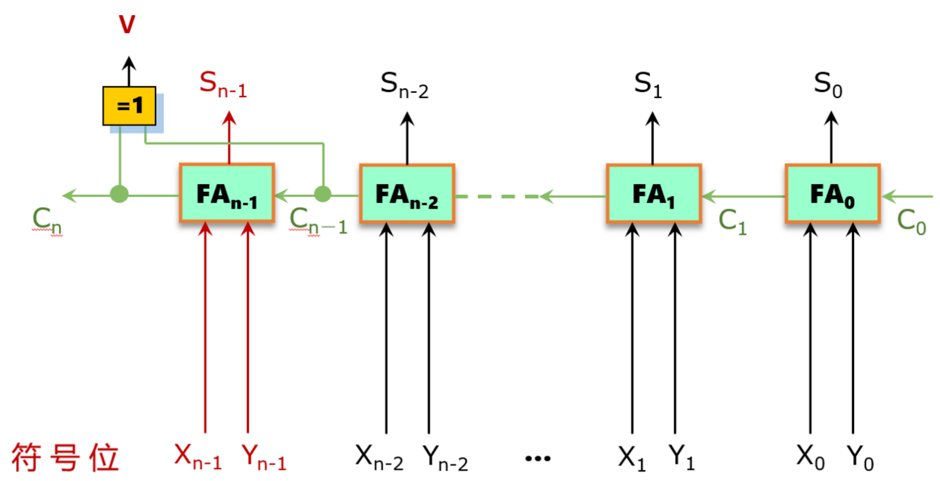


图3.3 多位串行加法器

3.1：解释下列名词（续）：

12. 先行进位：并行加法器中的进位信号是同时产生的，也称跳跃进位、并行进位。

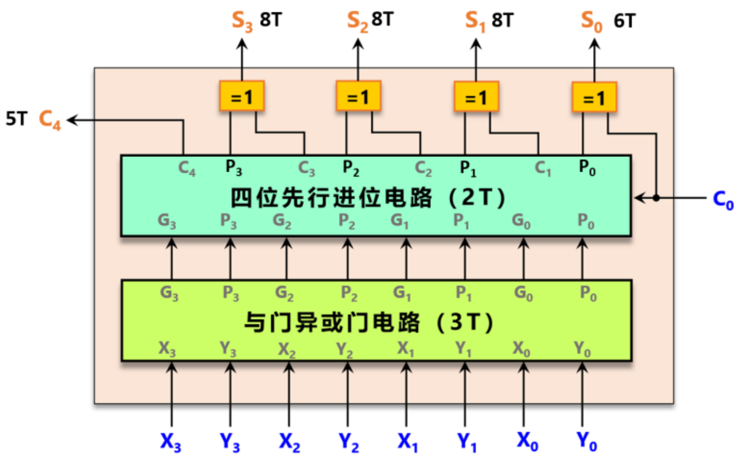


图3.6 4位先行进位加法器

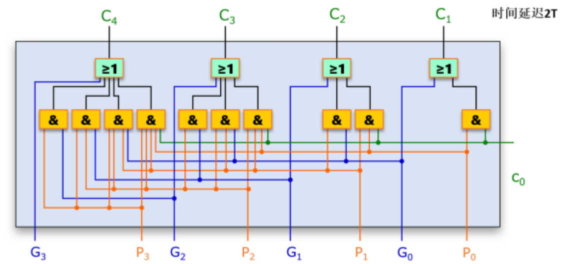


图3.5 4位先行进位电路

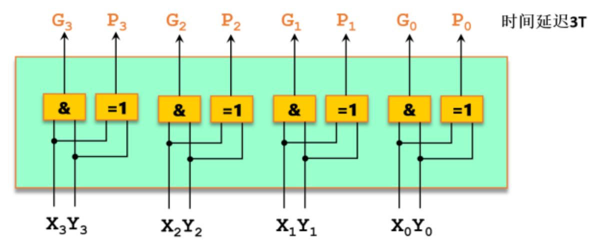


图3.6（下） 4位进位生成/进位传递电路

13.对阶：浮点加减运算的第一步，使参与运算的两个浮点数的阶码相同（小阶向大阶对齐）。

14.规格化：为增加有效数字的位数，提高运算精度，需将运算结果的尾数规格化。浮点加减运算、浮点乘除运算，都需要对结果进行规格化。

15.保留附加位：浮点加减运算时，对阶操作要对阶码小的尾数进行右移，保留附加位即保留右移后的若干位，这样在进行尾数求和与规格化处理时不会损失精度，只有在舍入处理时才丢掉附加位。

- 例3.14：设$x=2^{-101}\cdot(-0.101011)$，$y=2^{-010}\cdot(0.001110)$，数的阶码为3位，尾数为6位（均不含符号位），且都用补码表示，按照补码浮点数运算步骤计算x+y
- 解：
  - 首先用补码形式表示浮点数x和y（采用双符号位）
    - $[x]_补$=11 011，11.010101　　$[y]_补$=11 110，00.001110
  - （1）对阶：小阶向大阶对齐，x（阶码=-5）向y（阶码=-2）对齐，x的尾数右移3位
    - 对阶后的$[X]_补$=11 110，11.111010(101)
  - （2）尾数运算（求和）：

$[x]_补$	11 110，	11.111010 (101)
+ $[y]_补$	11 110，	00.001110
$[x+y]_补$	11 110，	00.001000 (101)

  - （3）尾数规格化处理：运算结果的尾数=00.001000(101)，为非规格化数（尾数的绝对值太小），需要左规，左移2次，变为规格化数00.100010(1)，阶码减2，阶码变为11 100
  - （4）舍入处理：舍入处理后的尾数为：00.100011（0舍1入法，末位恒置1法）
  - （5）溢出判断：阶码的双符号位相同（11），没有溢出
  - 运算结果：$[x+y]_补$ =11 100，00.100011，$x+y=2^{-100}\cdot(0.100011)$
  - 验证：
    - x=(1/32)*(-43/64)，y=(1/4)*(14/64)，x+y=(-43/2048)+(112/2048)=69/2048
    - $x+y=2^{-100}\cdot(0.100011)$=(1/16)*(35/64)=70/2048，相差1/2048，原因是舍入处理引起的误差

- **3.2 选择题**

① **D：（x=127=0000 007FH；y=-9=FFF7H；z=x+y=118=0000 0076H）。**

② **B：（r1=FEH=-2；r2=F2H=-14；r3=90H=-112；r4=F8H=-8；r2xr3=1568，**溢出了**）。**

③ **A：（x=1,0001011+1=1,0001100=-12；y=1,1001111+1=1,1010000=-80；z=-24+(-40)=-64=1100 0000）。**

④ **C：（x=FFFF FFDFH=-33；y=0000 0041H=65；x-y=-98=FFFF FF9EH）。**

⑤ **B：（**逻辑右移**1**位**：x=0110 1100；**算术右移**1**位**：x=1110 1100）。**

⑥ **D：（X+Y=2^7*((29/32)+(5/32))=2^7*(01.00010B)；**规格化：右移并且尾数舍入，尾数**00.10001B ，**阶码**8，**补码的表示范围：**-8～+7，11,000～00,111，**因此阶码发生溢出**）。**

⑦ **D：**

I：正确。

II：正确：右规时，阶码增大，尾数舍入过程需要右规调整，因此可能出现阶码上溢。

III：正确：左规时，尾数增大，阶码减小，有可能下溢。

IV：正确：因为浮点数中以阶码作为溢出判断的标准。

- **3.3 回答下列问题**

① 为什么采用并行进位能提高加法器的运算速度？

– 答：并行进位也就是先行进位，可以同时产生各个进位；而串行进位，高位的进位必须等到低位的进位得到后，才能产生；因此并行进位能够提高加法器的运算速度。例如，**4**位并行进位加法器，时间延迟为**8T**，而**4**位串行加法器的时间延迟为**(2n+4)T=(2*4+4)T=12T**。

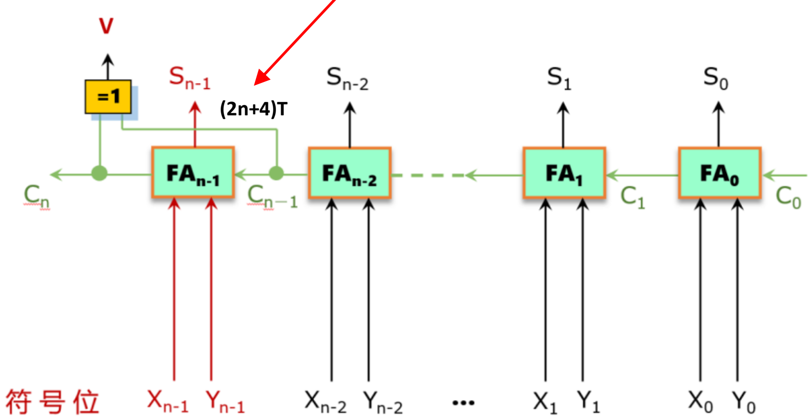


图3.3 多位串行加法器

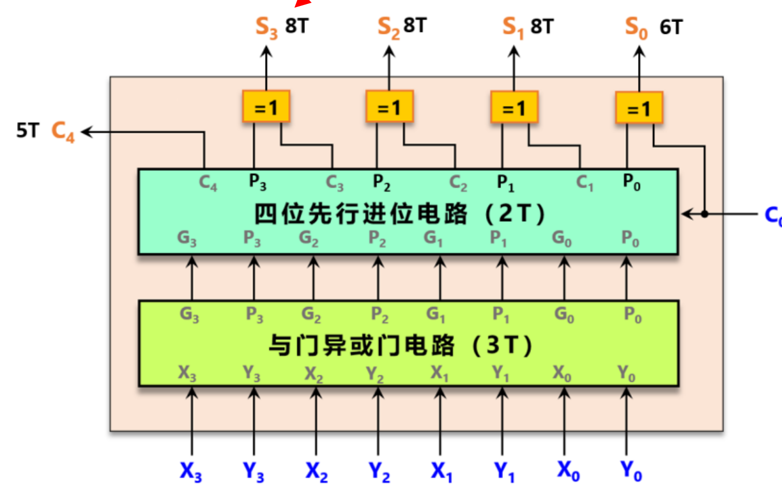


图3.6 4位先行进位加法器

② 如何判断浮点数运算结果是否发生溢出？

– 答：浮点数运算结果是否溢出是由阶码决定的，阶码采用双符号位时，如果符号位=01或10，则表示浮点数溢出了。

– 1、阶码和尾数采用补码表示的浮点加减运算

- （1）对阶：小阶向大阶对齐
- （2）尾数运算：定点数的补码加减运算
- （3）结果规格化：尾数的绝对值大于1，需要进行右规，尾数只需要右移1次，阶码加1；尾数的绝对值小于0.5，需要进行左规，尾数需要左移若干次，直到尾数的绝对值大于0.5（尾数每左移1次，阶码减1）
- （4）舍入：尾数进行右规时（尾数右移），尾数的末尾会被丢掉，从而产生误差。舍入方法：末位恒置1法；0舍1入法
- （5）溢出判断：阶码的符号位为01或10时，表示浮点数溢出了

③ 如何判断浮点数运算结果是否为规格化数？如果不是规格化数，如何进行规格化？

- 答：如果浮点数的尾数采用原码表示，当尾数的最高有效位=1，则为规格化数。如果浮点数的尾数采用补码表示，当尾数为正数且尾数的最高有效位=1，则为规格化数；当尾数为负数且尾数的最高有效位=0，则为规格化数。

- 如果不是规格化数，则可以通过左规（浮点数的绝对值小于0.5）或右规（浮点数的绝对值大于1），使浮点数的尾数变为规格化的数。

- 1、阶码和尾数采用补码表示的浮点加减运算
  - （1）对阶：小阶向大阶对齐
  - （2）尾数运算：定点数的补码加减运算
  - （3）结果规格化：尾数的绝对值大于1，需要进行右规，尾数只需要右移1次，阶码加1；尾数的绝对值小于0.5，需要进行左规，尾数需要左移若干次，直到尾数的绝对值大于0.5（尾数每左移1次，阶码减1）
  - （4）舍入：尾数进行右规时（尾数右移），尾数的末尾会被丢掉，从而产生误差。舍入方法：末位恒置1法；0舍1入法
  - （5）溢出判断：阶码的符号位为01或10时，表示浮点数溢出了

④　为什么阵列除法器中能用CAS的进位/借位控制端作为上商的控制信号？

– 答：上商位决定了下一步是进行加法还是进行减法，因此可用上一步的商（最左侧CAS的进位/借位输出）控制下一行串行进位加减法电路的运算，即商上1，下一步减除数；而商上0，下一步加除数；故将上一步的商与下一行的CAS电路的P输入相连。

– 2、阵列除法器的连接与运算

- 被除数$X=0.X_1X_2X_3X_4X_5X_6X_7X_8$
- 除数$Y=0.Y_1Y_2Y_3Y_4$
- 商$Q=0.Q_1Q_2Q_3Q_4$
- 余数$=0.000R_4R_5R_6R_7R_8$
- P=1（CAS做减法运算）

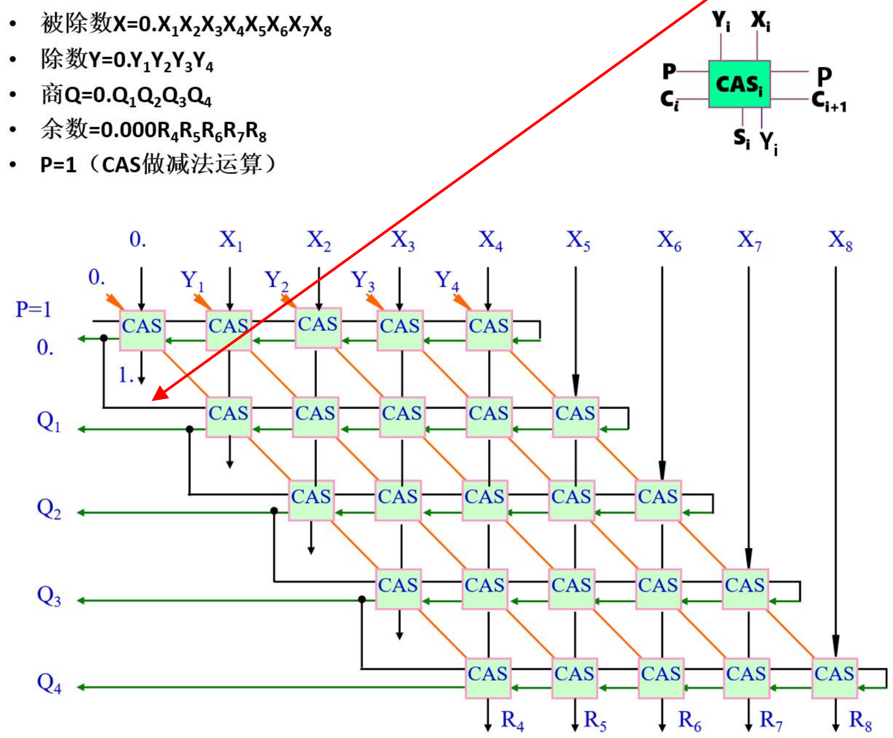


图3.18　阵列除法器

⑤ 移位运算和乘法及除法运算有何关系？

– 答：乘法是由加法和右移实现的；除法是由减法（减法也将通过加法实现）和左移实现的。因此计算机中只有加法器和移位运算部件，就可以完成加减乘除四则运算。

- 例3.10：x = 0.1101，y = -0.1011，用原码一位乘法求x·y
- 解：
  - 乘积的符号 $P_0 = x_0 \oplus y_0 = 1$
  - 乘积的绝对值=0.1000 1111（具体见下面的计算过程）
  - 乘积：$[x \cdot y]_{原}$=1.1000 1111，x·y = -0.1000 1111

	部分积	乘数 $\lvert y\rvert$	说明
	00.0000	1011	P=0
+	00.1101		$y_n=1$，$+\lvert x\rvert$
	00.1101	1011	
→	00.0110	1101	逻辑右移一位
+	00.1101		$y_n=1$，$+\lvert x\rvert$
	01.0011	1101	
→	00.1001	1110	逻辑右移一位
+	00.0000		$y_n=0$，+0
	00.1001	1110	
→	00.0100	1111	逻辑右移一位
+	00.1101		$y_n=1$，$+\lvert x\rvert$
	01.0001	1111	
→	00.1000	1111	右移一位

乘数|y|为y（-0.1011）的数值位（1011），部分积为0（双符号位小数）

根据|y|的最后一位是1还是0，执行+|x|或+0的操作

原码恢复余数法

	余数R	商Q	说明
	00.1001	0.0000	初始余数R=$\lvert x\rvert$，商Q=0
$+[-\lvert y\rvert]_{补}$	11.0101		减$\lvert y\rvert$
	11.1110	0.0000	余数为负，商0
$+\lvert y\rvert$	00.1011		加$\lvert y\rvert$恢复余数
	00.1001		
←	01.0010	0.000	左移1位
$+[-\lvert y\rvert]_{补}$	11.0101		减$\lvert y\rvert$
	00.0111	0.0001	余数为正，商1
←	00.1110	0.001	左移1位
$+[-\lvert y\rvert]_{补}$	11.0101		减$\lvert y\rvert$
	00.0011	0.0011	余数为正，商1
←	01.0110	0.011	左移1位
$+[-\lvert y\rvert]_{补}$	11.0101		减$\lvert y\rvert$
	11.1011	0.0110	余数为负，商0
$+\lvert y\rvert$	00.1011		加$\lvert y\rvert$恢复余数
	00.0110		
←	00.1100	0.110	左移1位
$+[-\lvert y\rvert]_{补}$	11.0101		减$\lvert y\rvert$
	00.0001	0.1101	余数为正，商1

余数|R|=0.0000 0001　商|Q|=0.1101

余数R的初始值为x的绝对值（双符号位小数）

余数为负，商0，加|y|恢复余数，左移1位，减|y|

余数为正，商1，左移1位，减|y|

- **3.4 已知x和y，用变形补码计算x+y，并判断结果是否溢出。**

- **答：**
- **（1）x=0.11010， y=0.10111**
  - $[x+y]_{补}=[x]_{补}+[y]_{补}$
  - 变形补码： $[x]_{补}$=00.11010， $[y]_{补}$=00.10111
  - $[x+y]_{补}$=01.10001，符号位为01，表示正溢出

$[x]_{补}$	00.11010
+ $[y]_{补}$	00.10111
$[x+y]_{补}$	01.10001

$[x+y]_{补}$＝01.10001， 正溢出

- **（2）x=0.11101， y=-0.10100**

$[x+y]_{补}$＝00.01001， 未溢出

- **（3）x=-0.10111， y=-0.11000**

$[x+y]_{补}$＝10.10001， 负溢出

- **3.5 已知x和y，用变形补码计算x-y，并判断结果是否溢出。**

- **答：**
- **（1）x=0.11011， y=0.11101**
  - $[x-y]_{\text{补}}=[x]_{\text{补}}+[-y]_{\text{补}}$
  - 变形补码： $[x]_{\text{补}}=00.11011$，$[-y]_{\text{补}}=11.00011$
  - $[x-y]_{\text{补}}=11.11101$，符号位为11，没有溢出

$$
\begin{array}{rr}
[x]_{\text{补}} & 00.11011 \\
+\ [-y]_{\text{补}} & 11.00011 \\
\hline
[x-y]_{\text{补}} & 11.11110
\end{array}
$$

$[x-y]$补=11.11101， 未溢出

- **（2）x=0.10111， y=0.11110**

$[x-y]_{\text{补}}=11.11001$，未溢出

- **（3）x=-0.11111，y=-0.11001**

$[x-y]_{\text{补}}=11.11010$，未溢出

- **3.6 用原码一位乘法计算x·y。**
- 答：
- （1）x=-0.11111， y=0.11101
  - 乘积的符号$P_0 = x_0 \oplus y_0 = 1 \oplus 0 = 1$
  - 乘积的绝对值=0.11100 00011（具体见下面的计算过程）
  - 乘积：$[x \cdot y]_原$=1. 11100 00011，x·y = -0. 11100 00011

$x \times y = -0.1110000011$

	部分积	乘数|y|	说明
	00.00000	11101	P=0
+	00.11111		y_n=1，+|x|
	00.11111	11101	
→	00.01111	11110	逻辑右移一位
+	00.00000		y_n=0，+0
	00.01111	11110	
→	00.00111	11111	逻辑右移一位
+	00.11111		y_n=1，+|x|
	01.00110	11111	
→	00.10011	01111	逻辑右移一位
+	00.11111		y_n=1，+|x|
	01.10010	01111	
→	00.11001	00111	逻辑右移一位
+	00.11111		y_n=1，+|x|
	01.11000	00111	
→	00.11100	00011	逻辑右移一位

乘数|y|为y（0.11101）的数值位（11101），部分积P为0（双符号位小数）

根据|y|的最后一位是1还是0，执行+|x|或+0的操作

- **3.6** 用原码一位乘法计算**x·y**。
- 答：
- （**2**）**x=-0.11010**，**y=-0.01011**
  - 乘积的符号$P_0 = x_0 \oplus y_0 = 1 \oplus 1 = 0$
  - 乘积的绝对值=**0.01000 11110**（具体计算过程参考上一题）
  - 乘积：$[x \cdot y]_{原}$=**0. 01000 11110**，**x·y = 0. 01000 11110**

$$\boxed{x \times y = 0.0100011110}$$

- **3.7 用补码一位乘法计算x·y。**
- 答：
- （1）x=0.10110，y=-0.00011
  - $[x]_补$ = 0.10110； $[y]_补$ = 1.11101
  - 双符号位：$[x]_补$ = 00.10110；$[-x]_补$ = 11.01010
  - $[x \cdot y]_补$ = 11.11101 11110（具体见下面的计算过程）
  - 验证：x = 0.10110，y = -0.00011，x · y = -0.00010 00010， $[x \cdot y]_补$ = 1.11101 11110

$[x \times y]_补 = 1.1110111110$

	部分积	乘数y	说明
	00.00000	1111010	P=0 y_{n+1}=0
+	11.01010		$y_n y_{n+1}$=10，$+[-x]_补$
	11.01010	1111010	
→	11.10101	0111101	算术右移一位
+	00.10110		$y_n y_{n+1}$=01，$+[x]_补$
	00.01011	0111101	
→	00.00101	1011110	算术右移一位
+	11.01010		$y_n y_{n+1}$=10，$+[-x]_补$
	11.01111	1011110	
→	11.10111	1101111	算术右移一位
+	00.00000		$y_n y_{n+1}$=11，+0
	11.10111	1101111	
→	11.11011	1110111	算术右移一位
+	00.00000		$y_n y_{n+1}$=11，+0
	11.11011	1110111	
→	11.11101	1111011	算术右移一位
+	00.00000		$y_n y_{n+1}$=11，+0
	11.11101	11110	最后一步不移位

乘数y为补码值（1.11101），去掉小数点（111101），最后增加一位y_{n+1}=0（1111010）；部分积为0（双符号位小数）

根据y的最后二位是01、10、00或11，执行$+[x]_补$、$+[-x]_补$、+0的操作

- **3.7 用补码一位乘法计算x·y。**
- 答：
- （2）x=-0.011010，y=-0.011101
  - $[x]_{补}$ = 1.100110； $[y]_{补}$ = 1.100011
  - 双符号位：$[x]_{补}$ = 11.100110；$[-x]_{补}$ = 00.011010
  - $[x \cdot y]_{补}$ = 00.001011 110010（具体计算过程参考上一题）
  - 验证： x=-0.011010，y=-0.011101， x · y = 0.001011 110010， $[x \cdot y]_{补}$ = 0. 001011 110010

$$\boxed{[x \times y]_{补} = 0.001011110010}$$

- **3.8 用原码不恢复余数法计算x÷y。**
- 答：
- （1）x=0.10101，y=0.11011
  - $|x|$=00.10101，$|y|$= 00.11011，$[-|y|]_{补}$ = 11.00101

原码不恢复余数法

<table>
<tr><th></th><th>余数R</th><th>商Q</th><th>说明</th></tr>
<tr><td></td><td>00.10101</td><td>0.00000</td><td>初始余数R=|x|，商Q=0</td></tr>
<tr><td>$+[-|y|]_{补}$</td><td>11.00101</td><td></td><td>减|y|</td></tr>
<tr><td></td><td>11.11010</td><td>0.00000</td><td>余数为负，商0</td></tr>
<tr><td>←</td><td>11.10100</td><td>0.0000</td><td>左移1位</td></tr>
<tr><td>+|y|</td><td>00.11011</td><td></td><td>加|y|</td></tr>
<tr><td></td><td>00.01111</td><td>0.00001</td><td>余数为正，商1</td></tr>
<tr><td>←</td><td>00.11110</td><td>0.0001</td><td>左移1位</td></tr>
<tr><td>$+[-|y|]_{补}$</td><td>11.00101</td><td></td><td>减|y|</td></tr>
<tr><td></td><td>00.00011</td><td>0.00011</td><td>余数为正，商1</td></tr>
<tr><td>←</td><td>00.00110</td><td>0.0011</td><td>左移1位</td></tr>
<tr><td>$+[-|y|]_{补}$</td><td>11.00101</td><td></td><td>减|y|</td></tr>
<tr><td></td><td>11.01011</td><td>0.00110</td><td>余数为负，商0</td></tr>
<tr><td>←</td><td>10.10110</td><td>0.0110</td><td>左移1位</td></tr>
<tr><td>+|y|</td><td>00.11011</td><td></td><td>加|y|</td></tr>
<tr><td></td><td>11.10001</td><td>0.01100</td><td>余数为负，商0</td></tr>
<tr><td>←</td><td>11.00010</td><td>0.1100</td><td>左移1位</td></tr>
<tr><td>+|y|</td><td>00.11011</td><td></td><td>加|y|</td></tr>
<tr><td></td><td>11.11101</td><td>0.11000</td><td>余数为负，商0</td></tr>
<tr><td>+|y|</td><td>00.11011</td><td></td><td>加|y|</td></tr>
<tr><td></td><td>00.11000</td><td>0.11000</td><td></td></tr>
</table>

商Q=0.11000；余数R=0.00000 11000
因为余数为负数，需要加|y|恢复为正余数（余数的符号必须与x的符号相同）

余数R的初始值为x的绝对值（双符号位小数）

余数为负，商0，左移1位，加|y|

余数为正，商1，左移1位，减|y|

验证：x=0.10101=21/32，
y=0.11011=27/32
商Q=0.11000=24/32，余数R=0.000000 11000=24/1024

x=Q*y+R=(24/32)*(27/32)+24/1024
=672/1024=21/32

- **3.8** 用原码不恢复余数法计算**x÷y**。
- 答：
- （2）**x=-0.10101，y=0.11011**
  - **|x|=00.10101，|y|= 00.11011，** $[-|y|]_{补}$ **= 11.00101**
  - 具体计算过程请参考上一题
  - 商**Q = -0.11000=-24/32**（商的符号为**x**和**y**符号的异或）；余数**R = -0.00000 11000**
  - **=-24/1024**（余数的符号与**x**相同）
  - 验证：**x =-0.10101=-21/32，y =0.11011=27/32**
  - **x=Q*y+R=(-24/32)*(27/32)+(-24/1024)=-672/1024=-21/32**

- **3.9 浮点数加法运算$[x+y]_{补}$。**
- 答：
- （1）**$x=2^{011}$x0.100100，$y=2^{010}$x(-0.011010)**

- 第0步：阶码用4位移码表示（含1位符号位），尾数用7位补码表示（含1位符号位）
- **x=1,011 0.100100 y=1,010 1.100110**

- 第1步：对阶，小阶向大阶对齐，y向x看齐
- 对阶后的**y=0,011 1.110011**

$$\begin{array}{rr} [x]_{补} & 0.100100 \\ +\ [y]_{补} & 1.110011 \\ \hline [x+y]_{补} & 0.010111 \end{array}$$

- 第2步：尾数求和
- **$[x+y]_{补}$=0.010111**

- 第3步：尾数规格化，**0.010111**为非规格化的尾数（绝对值小于**0.5**），需要左规，即左移**1**次，尾数变为**0.101110**，阶码减**1**，阶码**=0,010**

- **$[x+y]_{补}$=0,010 0.101110**

- 验证：**$x=2^3\cdot(36/64)=4.5$，$y=2^2\cdot(-26/64)=-1.625$；$x+y=2.875$**
- **$[x+y]_{补}$=0,010 0.101110 = $2^2\cdot(46/64)=2.875$**

- 3.9 浮点数加法运算$[x+y]_{补}$。
- 答：
- （2）$x=2^{-101}x(-0.100010)$， $y=2^{-100}x(-0.010110)$
- 阶码用4位移码表示（含1位符号位），尾数用7位补码表示（含1位符号位）
- $[x+y]_{补}$=1,100　1.011001
- 验证：$x=2^{-5}\cdot(-34/64)=-17/1024$， $y=2^{-4}\cdot(-22/64)=-22/1024$； $x+y=-39/1024$
- $[x+y]_{补}$=1,100　1.011001 = $2^{-100}\cdot(-0.100111)=2^{-4}\cdot(-39/64)=-39/1024$

- **3.10 采用IEEE754单精度浮点数计算：**
- **答：**
- **（1）0.625+(-12.25)**

- **$0.625=0.101B=1.01\times2^{-1}$**
- **E=-1+127=126=0111 1110**
- **M=010 0000 0000 0000 0000 0000**
- **IEEE754单精度浮点数=0 0111 1110 010 0000 0000 0000 0000 0000=3F20 0000H**

- **$-12.25=-1100.01=-1.10001\times2^{11}$**
- **E=3+127=130=1000 0010**
- **M=100 0100 0000 0000 0000 0000**
- **IEEE754单精度浮点数=1 1000 0010 100 0100 0000 0000 0000 0000=C144 0000H**

- **$0.625+(-12.25)=-11.625=-1011.101=-1.011101\times2^{11}$**
- **E=3+127=130=1000 0010**
- **M=011 1010 0000 0000 0000 0000**
- **IEEE754单精度浮点数=1 1000 0010 011 1010 0000 0000 0000 0000=C13A 0000H**

- **3.10 采用IEEE754单精度浮点数计算：**
- **答：**
- **（2）0.625-(-12.25)**

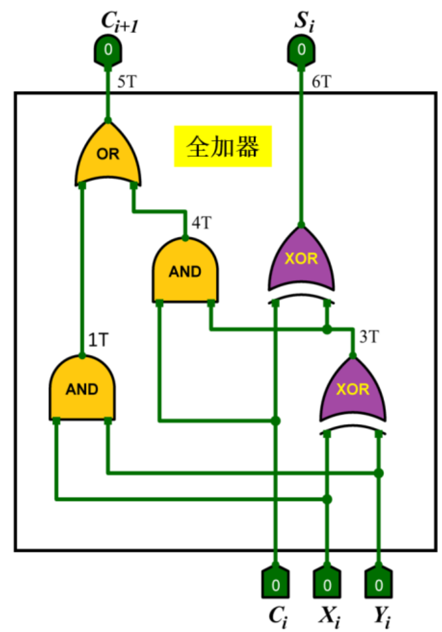
- **0.625=0.101B=1.01x2^{-1}**
- **E=-1+127=126=0111 1110**
- **M=010 0000 0000 0000 0000 0000**
- **IEEE754单精度浮点数=0 0111 1110 010 0000 0000 0000 0000 0000=3F20 0000H**

- **12.25=1100.01=1.10001x2^{11}**
- **E=3+127=130=1000 0010**
- **M=100 0100 0000 0000 0000 0000**
- **IEEE754单精度浮点数=0 1000 0010 100 0100 0000 0000 0000 0000=4144 0000H**

- **0.625-(-12.25)=12.875=1100.111=1.100111x2^{11}**
- **E=3+127=130=1000 0010**
- **M=100 1110 0000 0000 0000 0000**
- **IEEE754单精度浮点数=0 1000 0010 100 1110 0000 0000 0000 0000=414E 0000H**

- **3.11 运行C语言程序段。请回答：采用IEEE754单精度浮点数计算：**
- 答：
- （1）寄存器R1、R5、R6中的内容分别是什么？
  - x=134=86H（R1=86H）
  - y=246=F6H（R2=F6H）
  - m=x=86H=1,000 0110= -111 1010= -122（R3=86H）
  - n=y=F6H=1,111 0110= -000 1010= -10（R4=F6H）
  - z1=x-y=86H-F6H=90H（R5=90H）
  - z2=x+y=86H+F6H=7CH（R6=7CH）
  - k1=m-n=-122-(-10)= -112（R7=90H）
  - k2=m+n=-122+(-10)=-132（R8=7CH）
  - 所以，R1=86H，R5=90H，R6=7CH
- （2）m=-122，k1=-112
- （3）能。因为可控加减法器既可以完成无符号数的加减运算，也可以完成有符号数的加减运算。
- （4）判断带符号整数加减运算有3种方法，第2种方法是“根据运算过程中最高数据位的进位与符号位的进位是否一致进行检测”。上述程序中“int k2=m+n”会发生溢出，因为运算结果=-132，超出了8位二进制数补码的表示范围-128～+127。

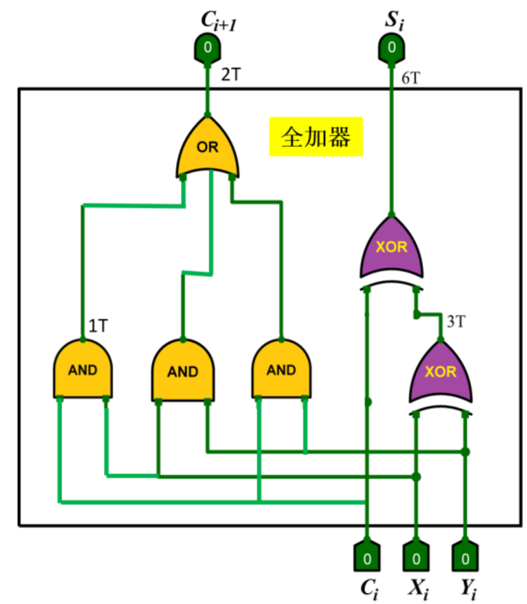
- **3.12 全加器采用教材图3.2(b)的方案实现。**
- 答：



$S_i = X_i \oplus Y_i \oplus C_i$

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

图3.2(b) 全加器方案1



$S_i = X_i \oplus Y_i \oplus C_i$

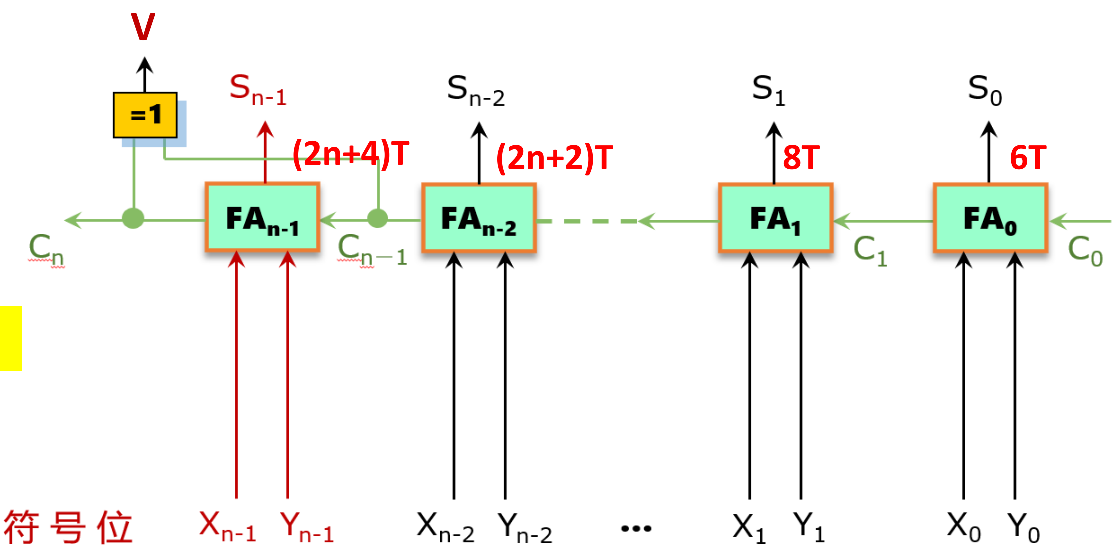
$C_{i+1} = X_i Y_i + (X_i + Y_i) C_i$

图3.2(b) 全加器方案2

产生进位信号需要5T

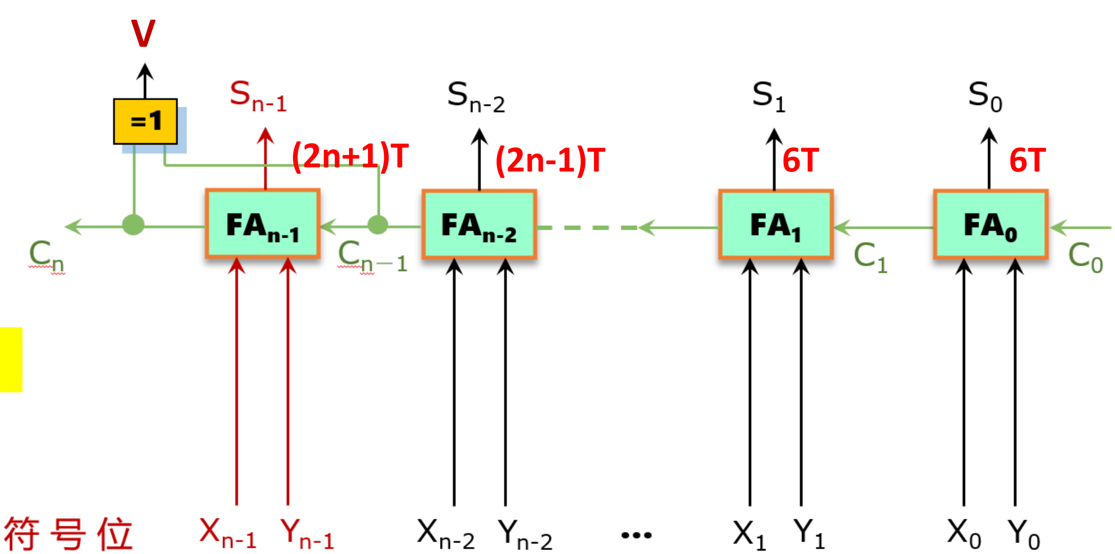
产生进位信号只需要2T

方案2与方案1相比产生进位信号的延迟少2T，但是增加1个与门



全加器方案1

图3.3



全加器方案2

少3T

全加器方案1

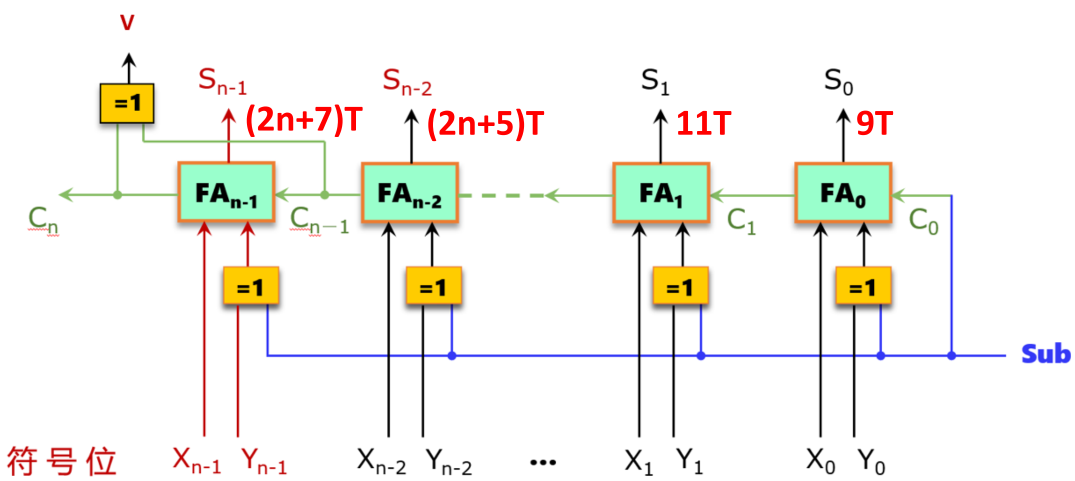
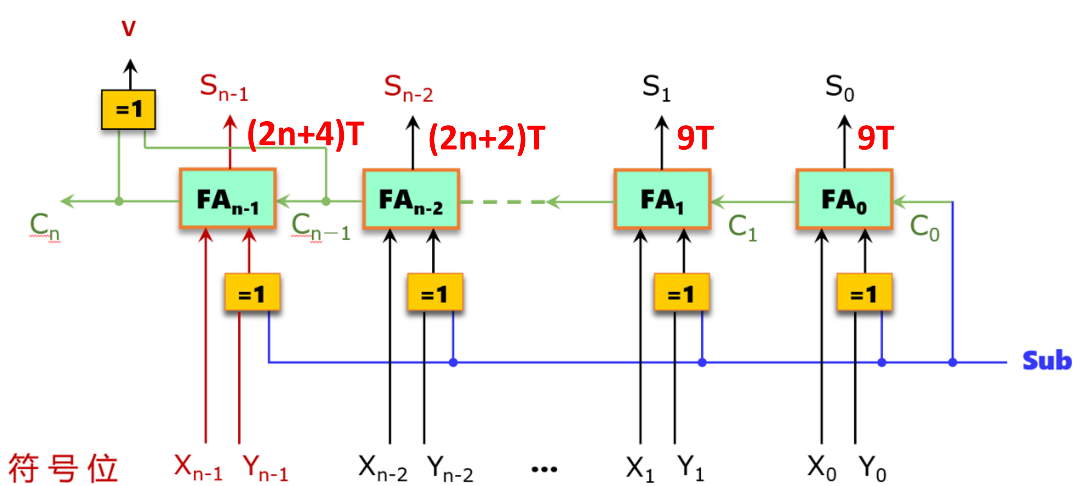


图3.4

全加器方案2



少3T

全加器方案1

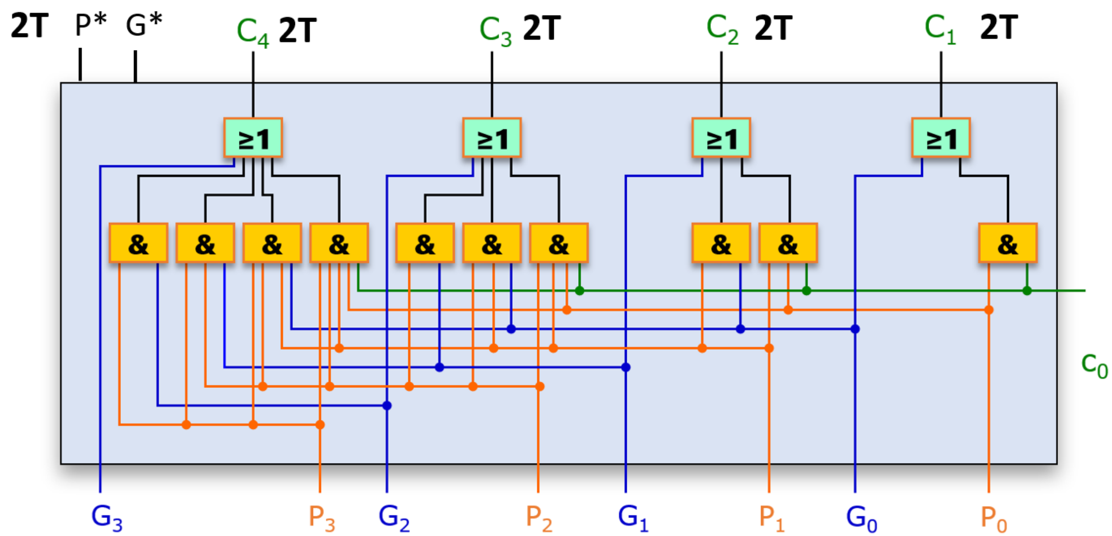
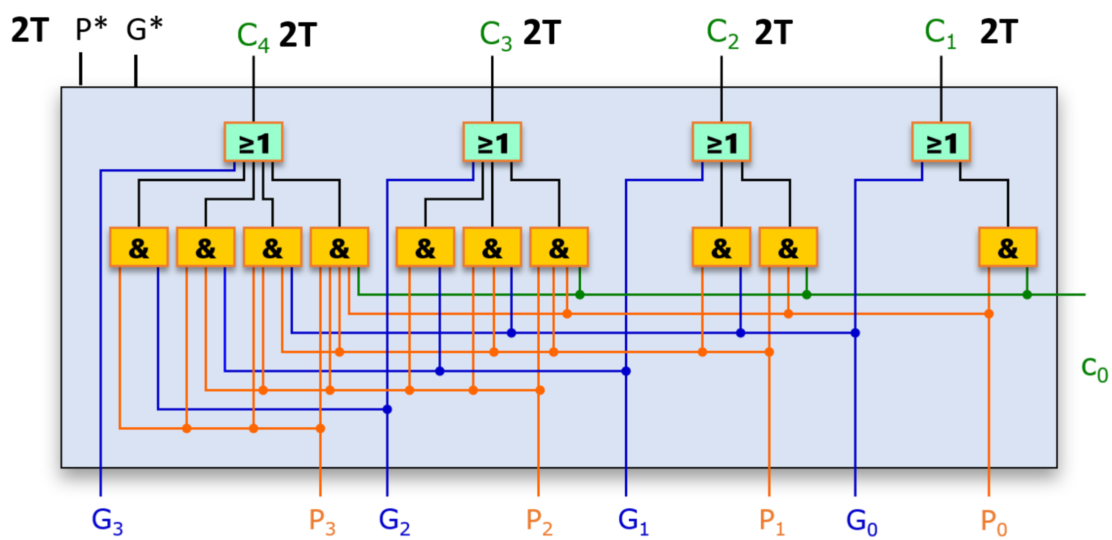


图3.5

全加器方案2



没有
变化

全加器方案1

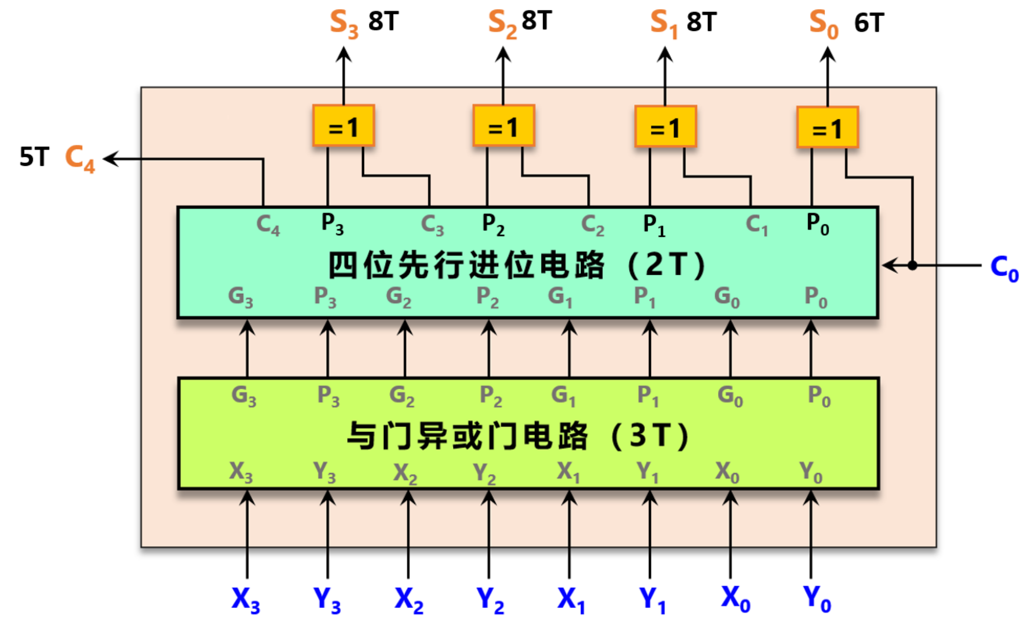
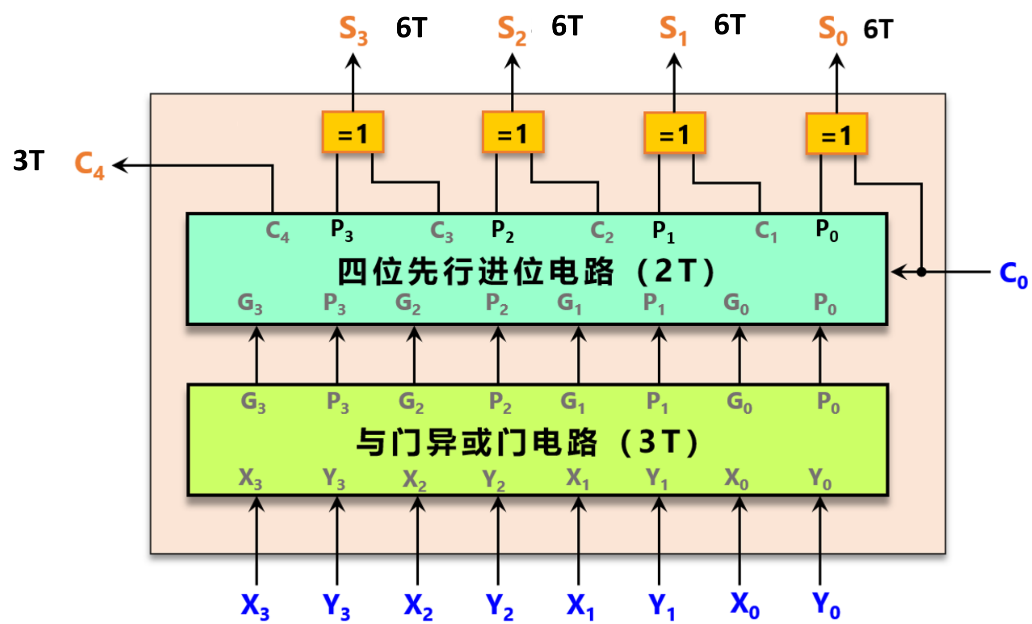


图3.6

全加器方案2



少2T

全加器方案1

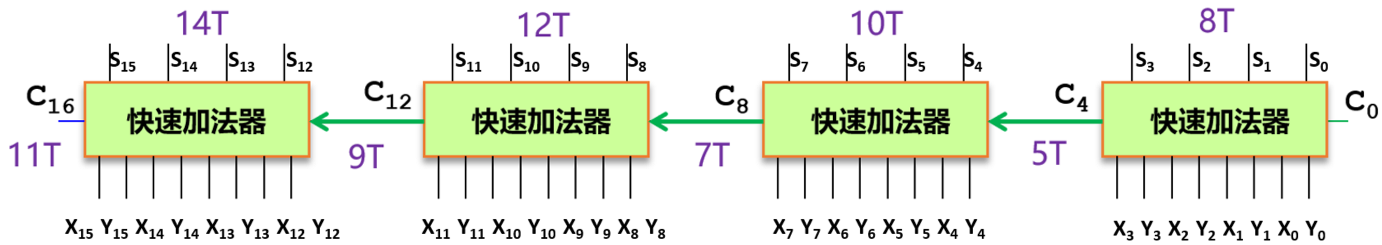
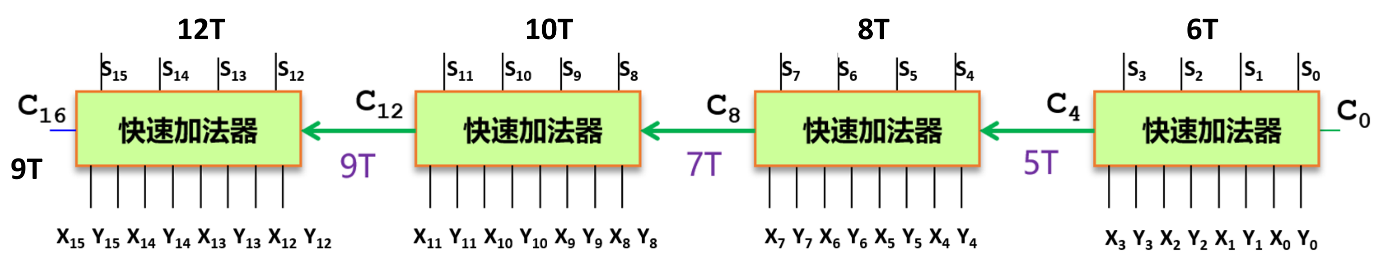


图3.7

全加器方案2



少2T

全加器方案1

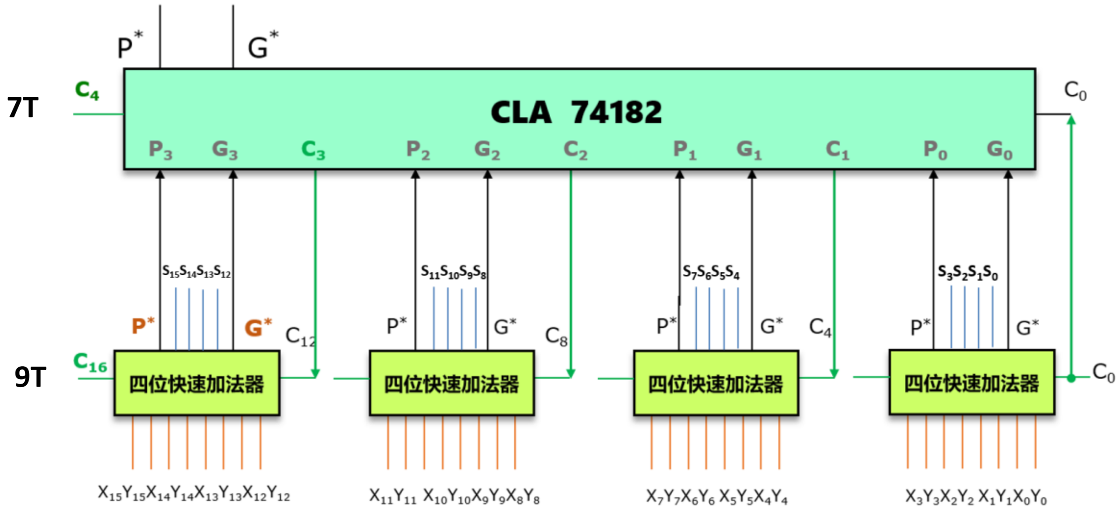
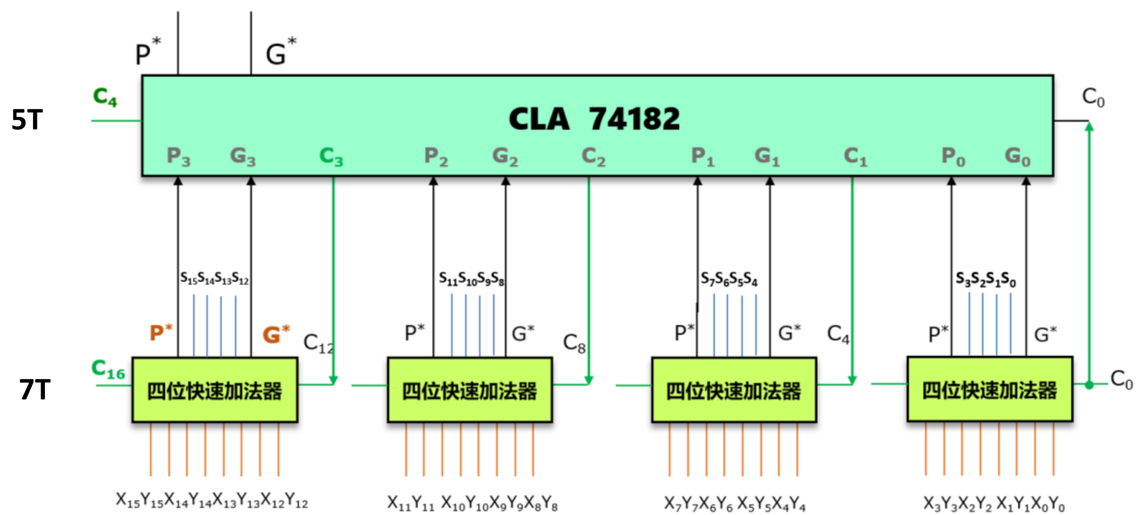


图3.8

全加器方案2



少2T

Thanks