第三章 数值运算及运算器

第一节　算术逻辑运算基础

一、定点加减运算

1.原码加减运算

2.补码加减运算

两个基本关系式：

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

由[y]补求[-y]补的方法：

将[y]补连同符号位一起求反加1。

补码加减运算

注意：求一个数的补码：

正数时，补码和原码相同;

负数时, 对原码除符号位外求反加1。

例1: $y = -0.0110$

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

例2: $y = 0.0111$

$[y]_{补} = 0.0111$ $[-y]_{补} = 1.1001$

补码加、减运算规则

- 参加运算的操作数用补码表示。
- 符号位参加运算。
- 操作码为加运算时，两数直接相加；当操作码为减运算时，将减数连同符号位一起求反加1,再与被减数相加。
- 运算结果以补码表示。

补码加、减运算举例

例1: 已知: x = 0.1001, y = -0.0110;
求 x+y = ?

解: $[x]_{补}$ = 0.1001 $[y]_{补}$=1.1010

$$
\begin{array}{rr}
[x]_{补} & 0.1001 \\
+\ [y]_{补} & 1.1010 \\
\hline
[x+y]_{补} & 1\ 0.0011
\end{array}
$$

x+y = 0.0011

补码加、减运算举例

例2: 已知: x = -0.1001, y = -0.0101;
求 x+y = ?

解: $[x]_{补}$ = 1.0111 $[y]_{补}$=1.1011

	$[x]_{补}$	1.0111
+	$[y]_{补}$	1.1011
	$[x+y]_{补}$	1 1.0010

x+y = -0.1110

补码加、减运算举例

例3: 已知: x = 0.1001, y = 0.0110;
求 x-y = ?

解: $[x]_{补}$ = 0.1001
$[y]_{补}$=0.0110 $[-y]_{补}$ =1.1010

$[x]_{补}$	0.1001
$+[-y]_{补}$	1.1010
$[x-y]_{补}$	1 0.0011

x-y = 0.0011

补码加、减运算举例

例4: 已知: x = -0.1001, y = -0.0110;
求 x-y = ?

解: $[x]_{补}$ = 1.0111 $[y]_{补}$=1.1010 $[-y]_{补}$ = 0.0110

$[x]_{补}$	1.0111
$+[-y]_{补}$	0.0110
$[x-y]_{补}$	1.1101

x-y = -0.0011

3.反码加减运算

反码加减运算的规则：

- 参加运算的操作数用反码表示。
- 符号位参加运算。
- 当操作码为加运算时，两数直接相加；当操作码为减运算时，将减数连同符号位一起求反与被减数相加。
- 如果符号位产生进位，则在末位加1，即循环进位。
- 运算结果为反码表示。

二、溢出检测

1. 采用一个符号位判断

规则：

• 当两个同号数相加，若所得结果符号与两数符号不同，则表明溢出。

• 设A_n、B_n分别表示两个操作数的符号；S_n表示结果的符号，则有：

$$溢出=\overline{A}_n\overline{B}_nS_n + A_nB_n\overline{S}_n$$

63+66=129	(-63)+(-66)=-129
0,0111111	1,1000001
+0,1000010	+1,0111110
1,0000001	10,1111111

2. 采用最高有效位的进位判断

- *方法：*
- 两个正数相加，最高有效位有进位，符号位无进位，表明运算结果发生溢出；两负数相加，最高有效位无进位，符号位有进位，表明结果发生溢出。

- 设C_n表示符号位本身的进位，C_{n-1}表示最高有效位向符号位的进位；
- 得出：
- 溢出=$\overline{C}_nC_{n-1} + C_n\overline{C}_{n-1}=C_n\oplus C_{n-1}$

63+66=129	(-63)+(-66)=-129
0,0111111	1,1000001
+0,1000010	+1,0111110
1,0000001	10,1111111

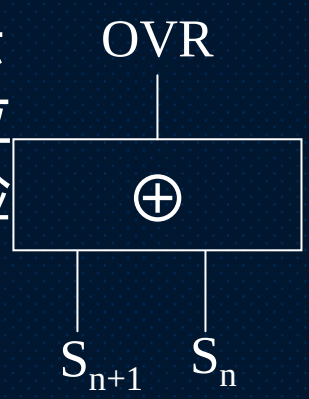
3.采用变形补码(双符号位)判溢出

- 正数：两个符号位均为0；00. $x_1x_2\ldots x_n$
- 负数：两个符号位均为1；11. $x_1x_2\ldots x_n$

•溢出判断：

•两数相加，结果符号位为00、11，表示没溢出；结果符号位为01表示正溢出，为10表示负溢出。

如果用 S_{n+1}、S_n 分别表示最高符号位和第二符号位，则采用变形补码溢出检测电路：



$$OVR = S_{n+1} \oplus S_n$$

63+66=129

00,0111111

+00,1000010

01,0000001

(-63)+(-66)=-129

11,1000001

+11,0111110

110,1111111

三、移位

- 按操作性质可分为三种类型：
- 逻辑移位、循环移位、算术移位。

1、逻辑移位

只有数码位置的变化，而无数量的变化。

左移：低位补0。　　右移：高位补0。

*例：*A寄存器的初值为　10110101

逻辑右移一位后为　01011010

逻辑左移一位后为　01101010

2、 循环移位

寄存器两端触发器有移位通路，形成闭合的移位环路。

例：A寄存器的初值为 10011001

循环右移一位后为 11001100

循环左移一位后为 00110011

3、 算术移位

数的符号不变，而数值发生变化。

- 左移一位将使数值扩大一倍（乘以2）
- 右移一位则使数值缩小一倍（乘以1/2）
- 算术移位规则：

（1）正数：原码、补码、反码左右移位时，空位均补入0（符号不变）。

例：A寄存器初值：0.0110

左移一位：0.1100 右移一位：0.0011

3、 算术移位

（2）负数：

- 原码：符号位不变（为1），空位补0。

  例：A寄存器的初值为 1.0110

  算术左移一位后为 1.1100

  算术右移一位后为 1.0011

- 补码：左移后的空位补0，右移后的空位补1。

- 例：初值：1.1011

  左移一位：1.0110

  右移一位：1.1101

- 反码：移位后的空位补**1**。
  - 例：初值：1.1011

    左移一位：1.0111

    右移一位：1.1101

四、十进制运算

1.进制转换
2.直接进行十进制运算
3. BCD码的加法运算

五、 逻辑运算

逻辑运算例:

例(1)逻辑或:

X=10100001,Y=10011011,X∨Y=?

```
      10100001  X
∨     10011011  Y
-----------------

      10111011
```

例(2)按位置“1”:

设: A=10010010,将A最低位置“1”;

设:B =00000001

10010010 A

V 00000001 B

10010011 A

例(3):按位清

设:A=10010010,将A最高位清“0”

设:B=01111111

```
    10010010   A
∧   01111111   B
-------------------
    00010010   A
```

例(4):按位测试

设:A=10010010,测A最高位是否为“1”;

设: B=10000000

$$\begin{array}{ll} & 10010010 \\ \wedge & 10000000 \\ \hline & 10000000 \end{array}$$

结果不全为“0”,表明被测码的被测位为“1”。

结果为全“0”,表明被测码的被测位为“0”。

例(5)比较

设:A＝10010010,B＝10010011,比较A,B内容相同否?

$$\begin{array}{ll} & 10010010 \\ \oplus & 10010011 \\ \hline \oplus & 00000001 \end{array}$$

结果全“0”,则A,B内容相等,否则内容不等。

第二节　基本算术运算的实现

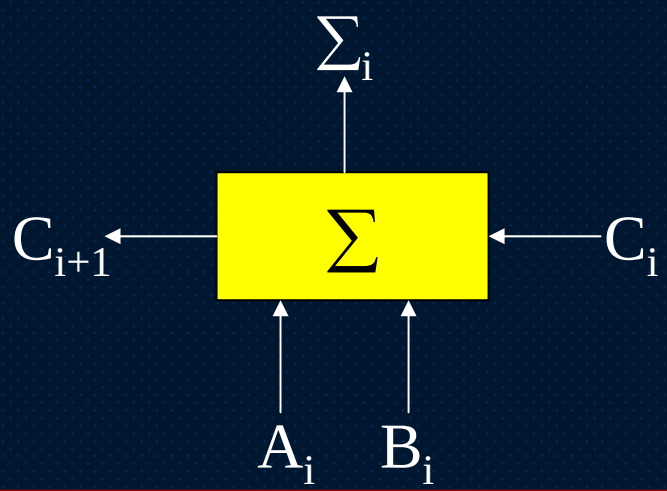
一、加法单元

全加器有三个输入量：

第 i 位的两个操作数 A_i、B_i 和低位送来的进位 $C_{i;}$

两个输出量：全加和 Σ_i 及向高位的进位C_{i+1}。

全加器框图:



全加器的功能表：

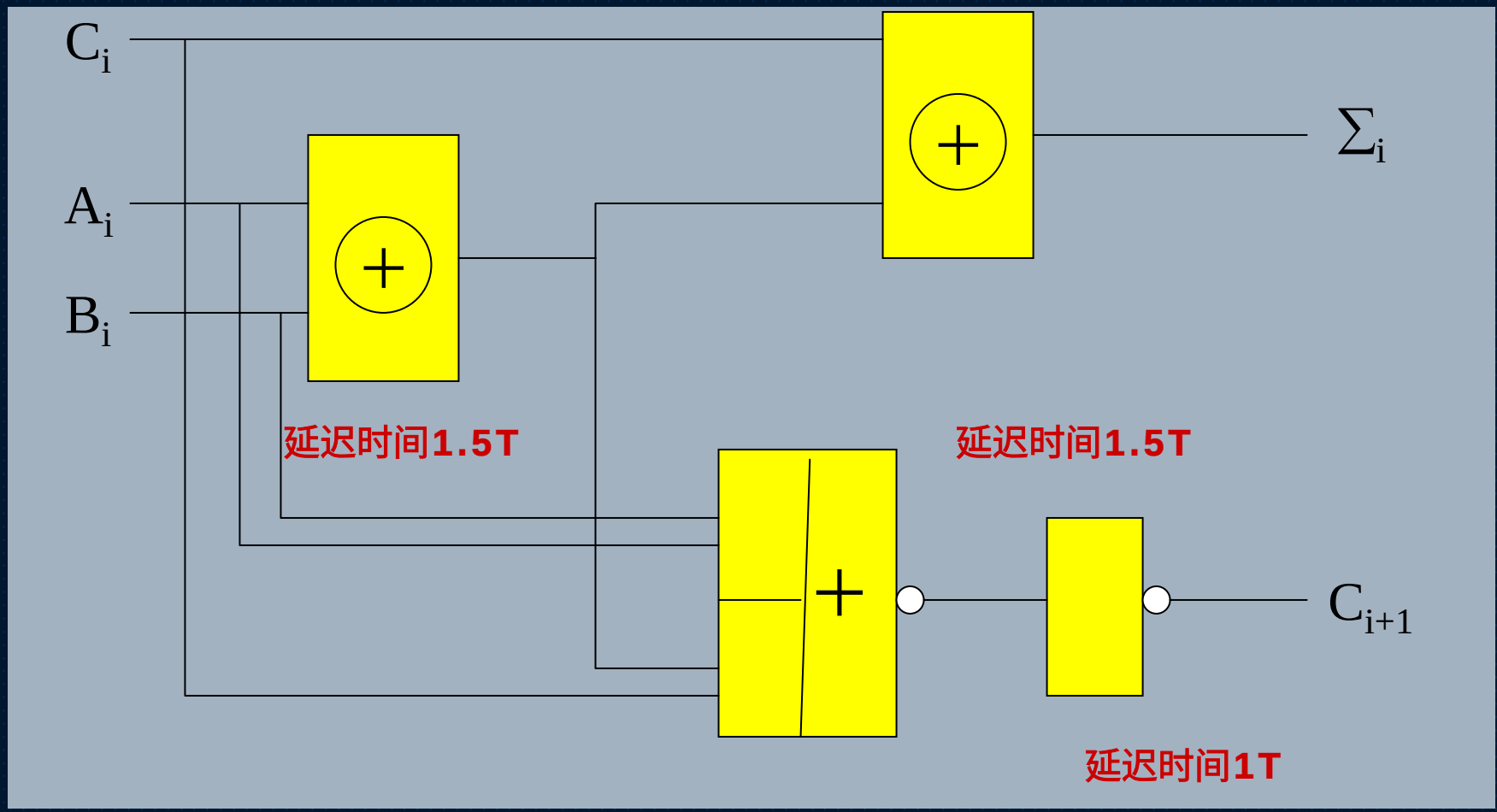
输入			输出	
A_i	B_i	C_i	$\sum_i$	C_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

- 全加和Σ_i及进位C_{i+1}的逻辑表达式：

$$\Sigma_i = \overline{A}_i\overline{B}_iC_i + \overline{A}_iB_i\overline{C}_i + A_i\overline{B}_i\overline{C}_i + A_iB_iC_i = A_i \oplus B_i \oplus C_i$$

$$C_{i+1} = \overline{A}_iB_iC_i + A_i\overline{B}_iC_i + A_iB_i\overline{C}_i + A_iB_iC_i = A_iB_i + (A_i \oplus B_i)C_i$$

用半加器构成的全加器



二、串行加法器和并行加法器

- 加法器有两种形式：串行加法器和并行加法器。

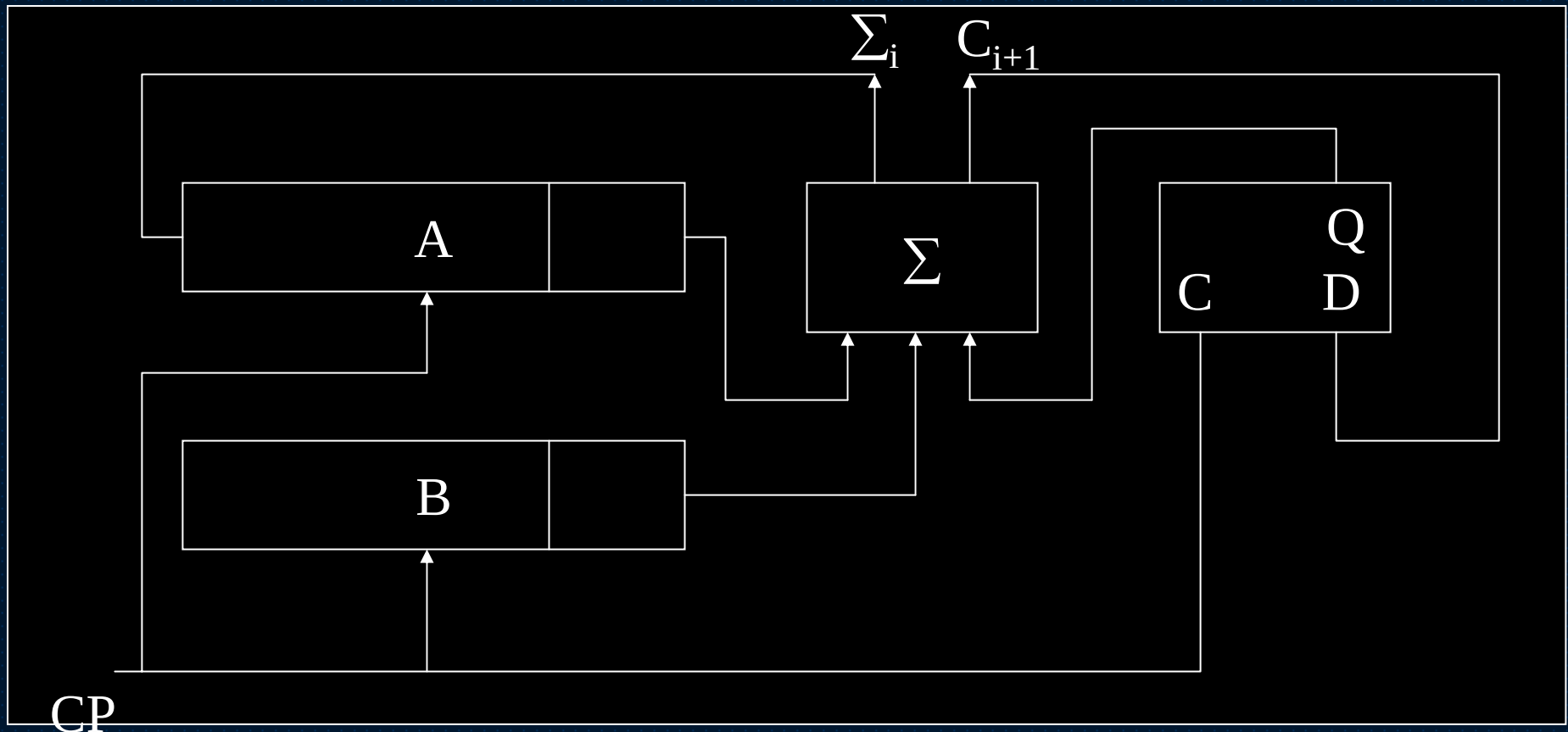
（一）串行加法器

- n 位字长的加法器仅有一位全加器，使用移位寄存器从低位到高位串行地提供操作数，分n步进行相加。

（二）并行加法器

全加器位数和操作数位数相同，同时对所有位进行求和。

二、串行加法器和并行加法器



串行加法器逻辑图

三、并行加法器的进位结构

- 并行加法器中传递进位信号的逻辑线路称为进位链

- 进位线路结构分为：串行进位、并行进位，将整个加法器分组（分级），对组内、组间（级间）分别采用串行或并行进位。

（一）对进位公式的分析

设相加的两个n位操作数为：

$A=A_{n-1}A_{n-2}\cdots A_i..A_0$　$B=B_{n-1}B_{n-2}\cdots B_i..B_0$

$C_{i+1} = A_iB_i + (A_i \oplus B_i)\, C_i$ —— 进位逻辑表达式

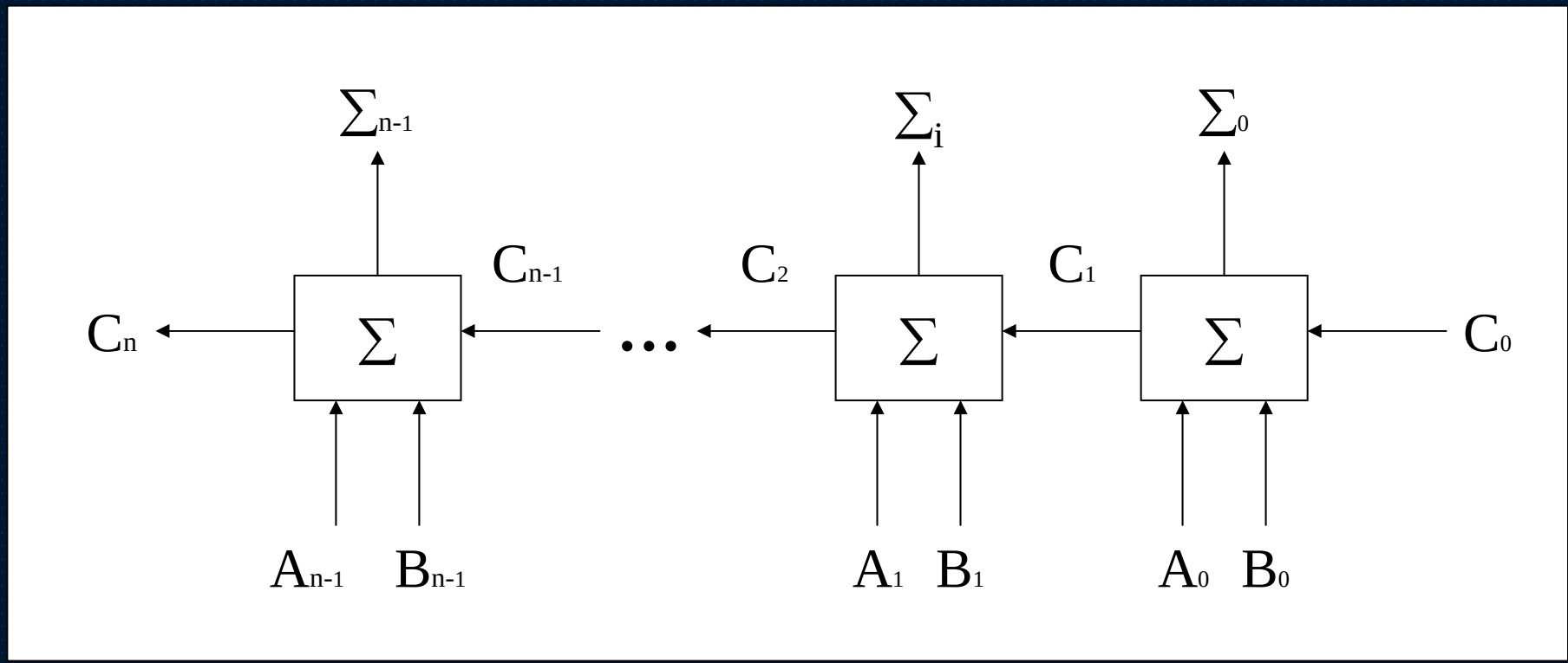
- 设：$G_i = A_iB_i$ —— 进位产生函数(Carry Generate Function)

$P_i = A_i \oplus B_i$ —— 进位传递函数(Carry Propagate Function)

- 当$P_i=1$时，如果低位有进位，本位将产生进位。

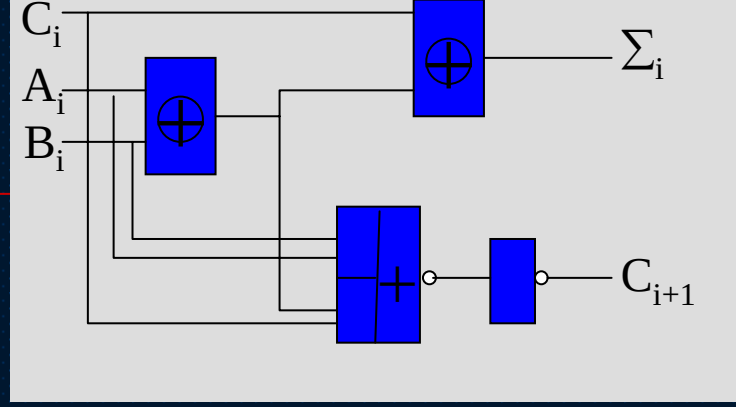
则：$C_{i+1} = G_i + P_iC_i$

（二）串行进位（行波进位）



N位串行进位的并行加法器

（二）串行进位（行波进位）



- 串行进位的逻辑表达式：

$$C_1 = G_0 + P_0C_0 = A_0B_0 + (A_0 \oplus B_0)C_0$$

$$C_2 = G_1 + P_1C_1 = A_1B_1 + (A_1 \oplus B_1)C_1$$

$$C_3 = G_2 + P_2C_2 = A_2B_2 + (A_2 \oplus B_2)C_2$$

$$\cdots$$

$$C_n = G_{n-1} + P_{n-1}C_{n-1} = A_{n-1}B_{n-1} + (A_{n-1} \oplus B_{n-1})C_{n-1}$$

- 最长进位延迟时间为 4+2.5(n-1)T ，与 n 成正比。

（三）并行进位（同时进位、先行进位）

$C_1 = G_0 + P_0C_0$

$C_2 = G_1 + P_1C_1 = G_1 + P_1(G_0 + P_0C_0)$

$C_3 = G_2 + P_2C_2 = G_2 + P_2(G_1 + P_1(G_0 + P_0C_0))$

$C_4 = G_3 + P_3C_3 = G_3 + P_3(G_2 + P_2(G_1 + P_1(G_0 + P_0C_0)))$

- 展开整理：

$C_1 = G_0 + P_0C_0$

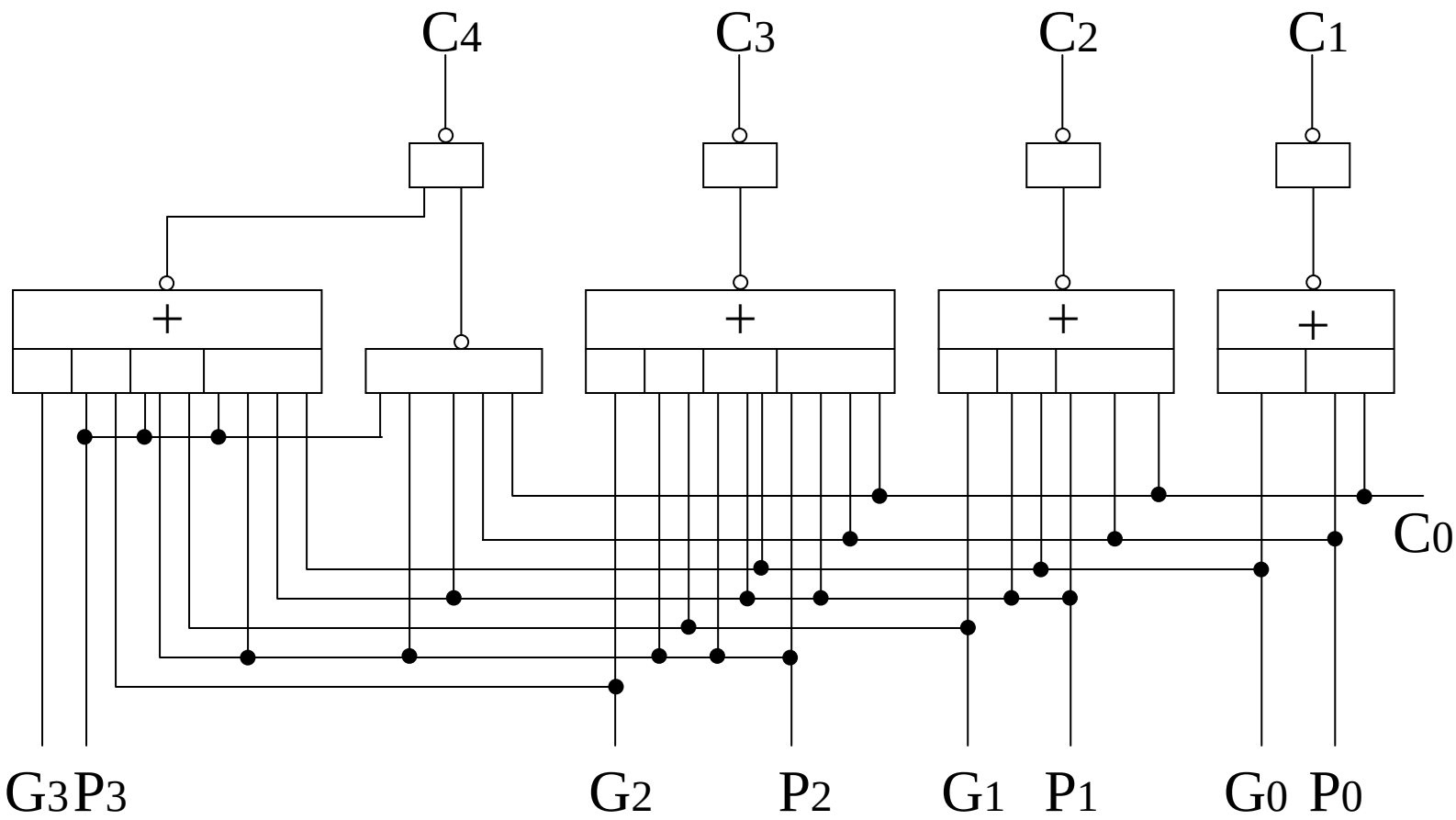
$C_2 = G_1 + P_1G_0 + P_1P_0C_0$

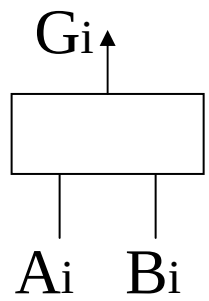
$C_3 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$

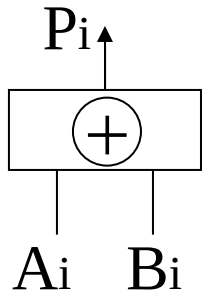
$C_4 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0$

- 全部进位输出信号仅由进位产生函数 G_i，进位传递函数 P_i 以及最低位进位 C_0 决定，与低位进位无关。

四位并行进位链线路

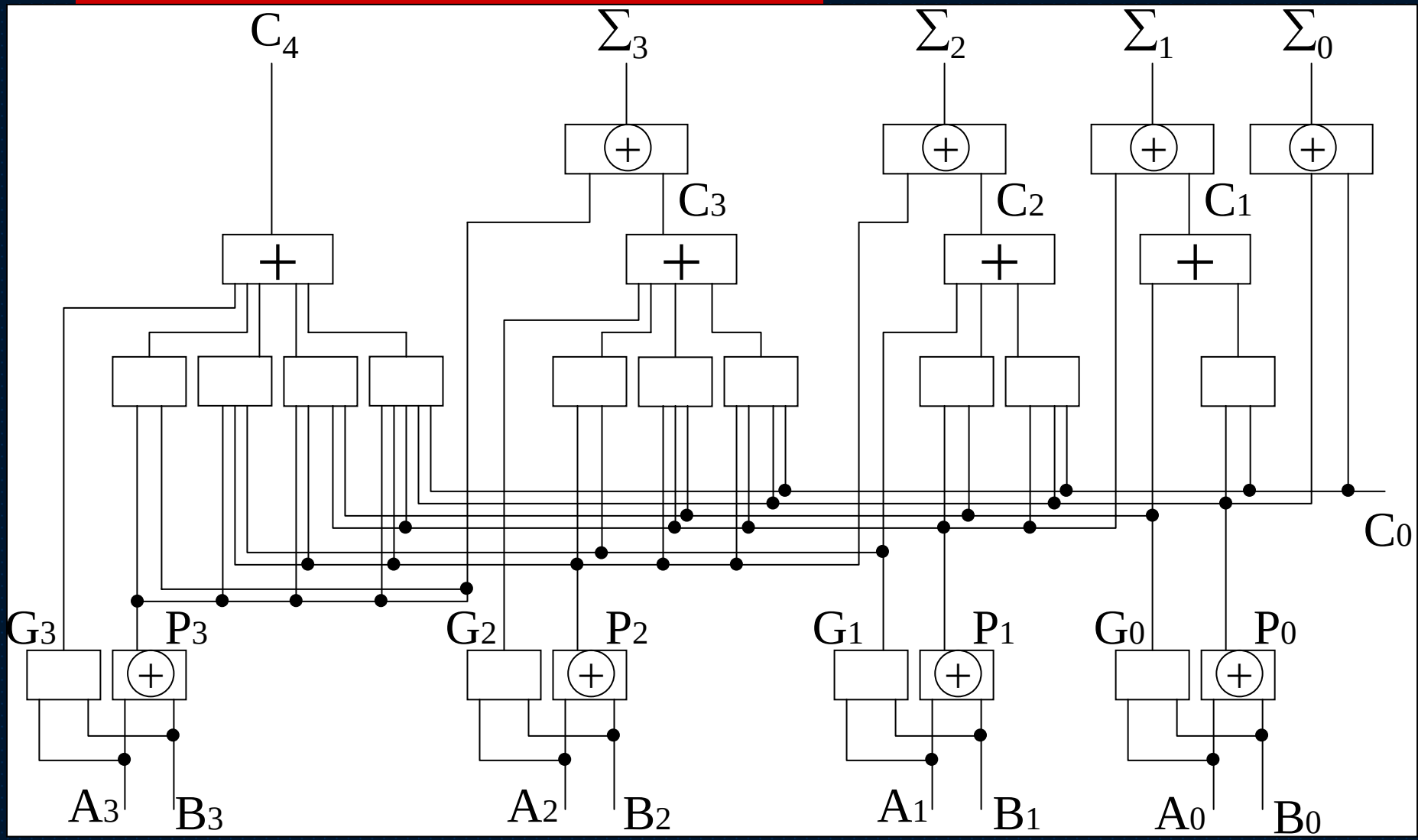






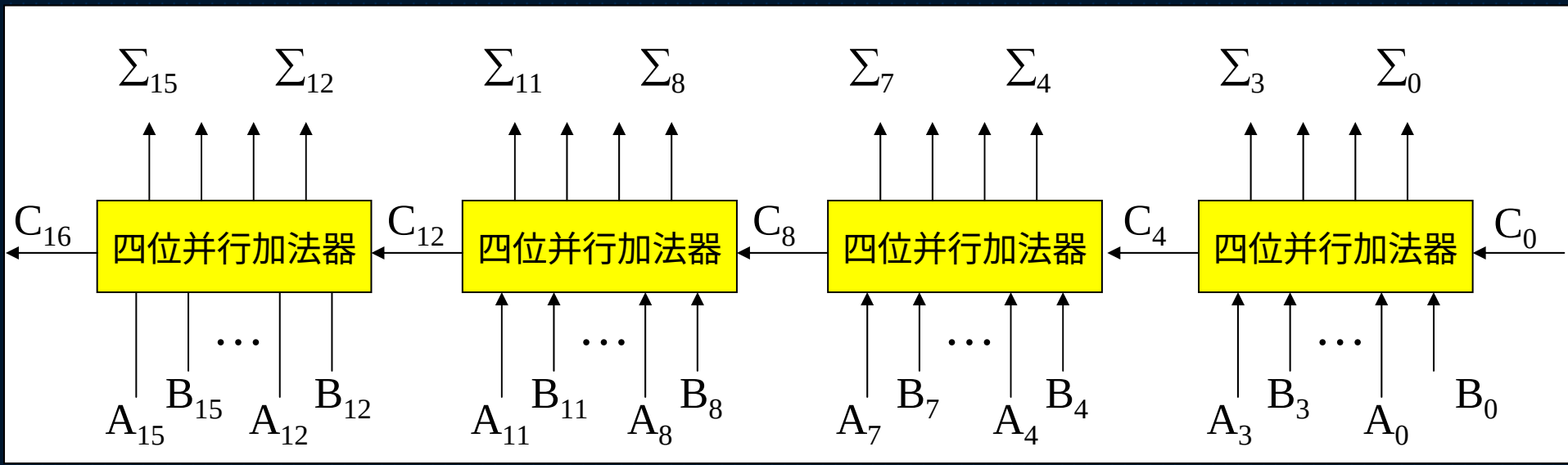
每位进位延迟时间为4T

四位并行进位加法器


C4
Σ3
Σ2
Σ1
Σ0
C3
C2
C1
C0
G3
P3
G2
P2
G1
P1
G0
P0
A3
B3
A2
B2
A1
B1
A0
B0

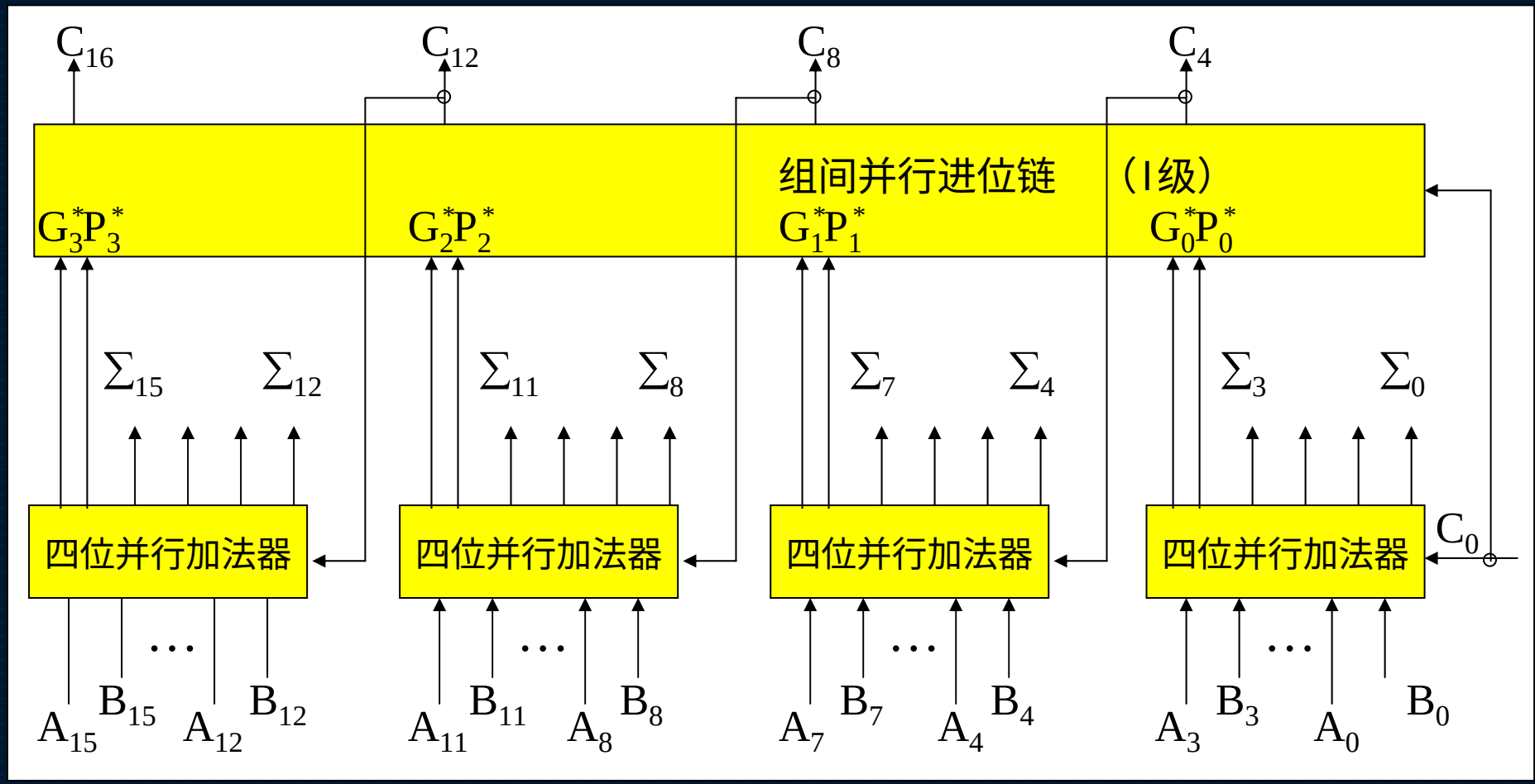
（四）组内并行、组间串行的进位结构

- 将n位加法器分成若干个小组，小组采用并行、组间采用串行的进位结构。
- 例：将16位加法器分成4组，每组4位，组内采用并行进位结构，组间采用串行进位结构。



- 最高进位的形成时间为 (4+3×2)T = 10 T
- 如果采用串行进位，最高进位的形成时间为 (4+15×2.5)T = 41.5 T

（五）组内并行、组间并行的进位结构



组内并行、组间并行的16位加法器

（五）组内并行、组间并行的进位结构

- 将加法器分成几个小组，每一小组包括几位，采用并行进位结构，小组间也采用并行进位。
- 再引入两个辅助函数 G_i^*和P_i^*；分别称为组进位产生函数和传递函数。
- G_i^*为本小组产生的进位（与低位小组来的进位无关）。
- Pi*为小组进位的传递条件（决定于低位小组进位能否传送至高位小组）。

（五）组内并行、组间并行的进位结构

- Gi*和Pi*的逻辑表达式：

$$\begin{cases} G_0^* = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 \\ P_0^* = P_3P_2P_1P_0 \end{cases}$$

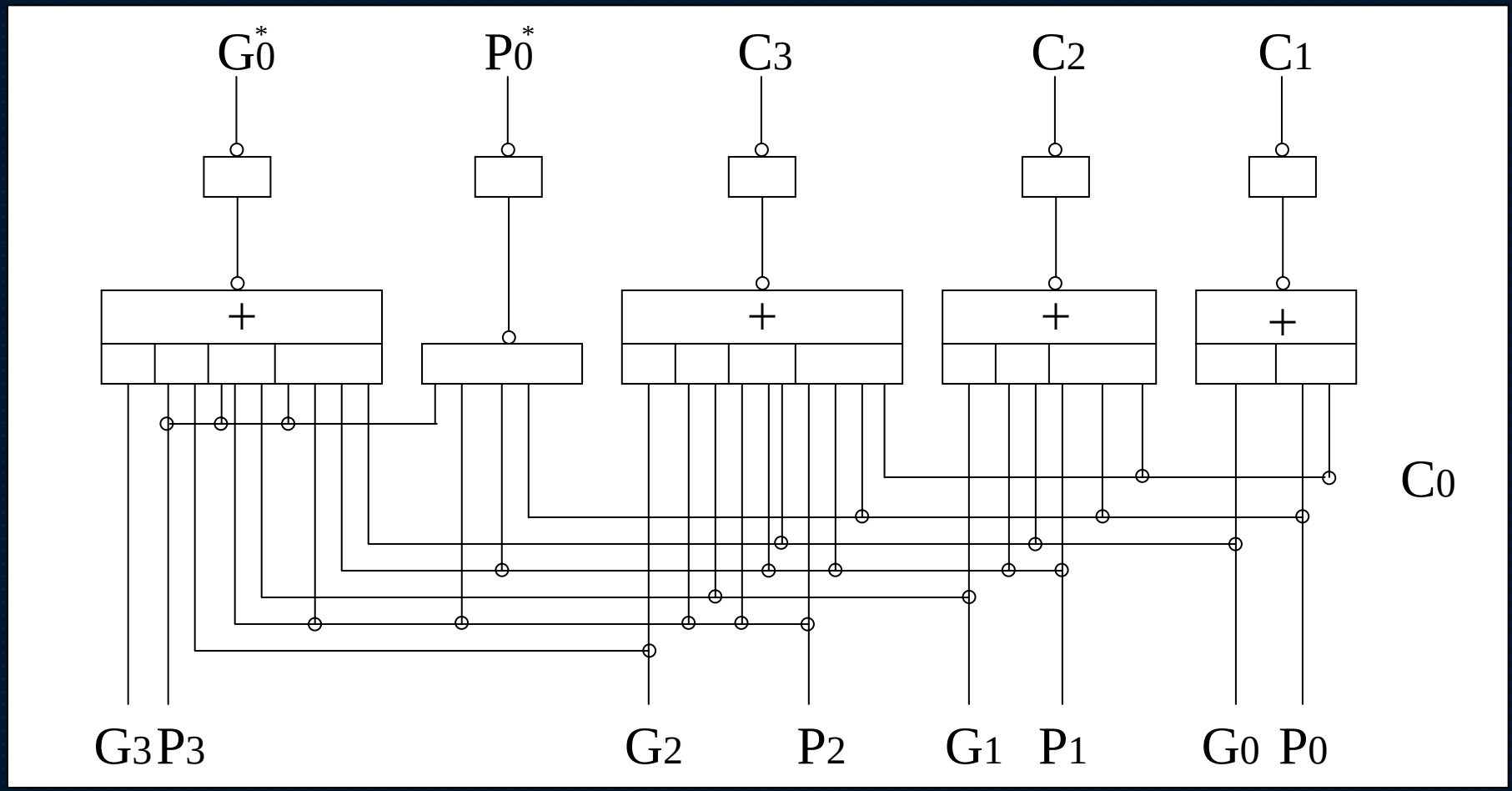
$$\begin{cases} G_1^* = G_7 + P_7G_6 + P_7P_6G_5 + P_7P_6P_5G_4 \\ P_1^* = P_7P_6P_5P_4 \end{cases}$$

$$\begin{cases} G_2^* = G_{11} + P_{11}G_{10} + P_{11}P_{10}G_9 + P_{11}P_{10}P_9G_8 \\ P_2^* = P_{11}P_{10}P_9P_8 \end{cases}$$

$$\begin{cases} G_3^* = G_{15} + P_{15}G_{14} + P_{15}P_{14}G_{13} + P_{15}P_{14}P_{13}G_{12} \\ P_3^* = P_{15}P_{14}P_{13}P_{12} \end{cases}$$

$$C_4 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0$$

组内并行、组间并行进位结构



- 小组间产生四个进位，C_4、C_8、C_{12}和C_{16}。

$$C_4 = G_0^* + P_0^* C_0$$

$$C_8 = G_1^* + P_1^* C_4 = G_1^* + P_1^* G_0^* + P_1^* P_0^* C_0$$

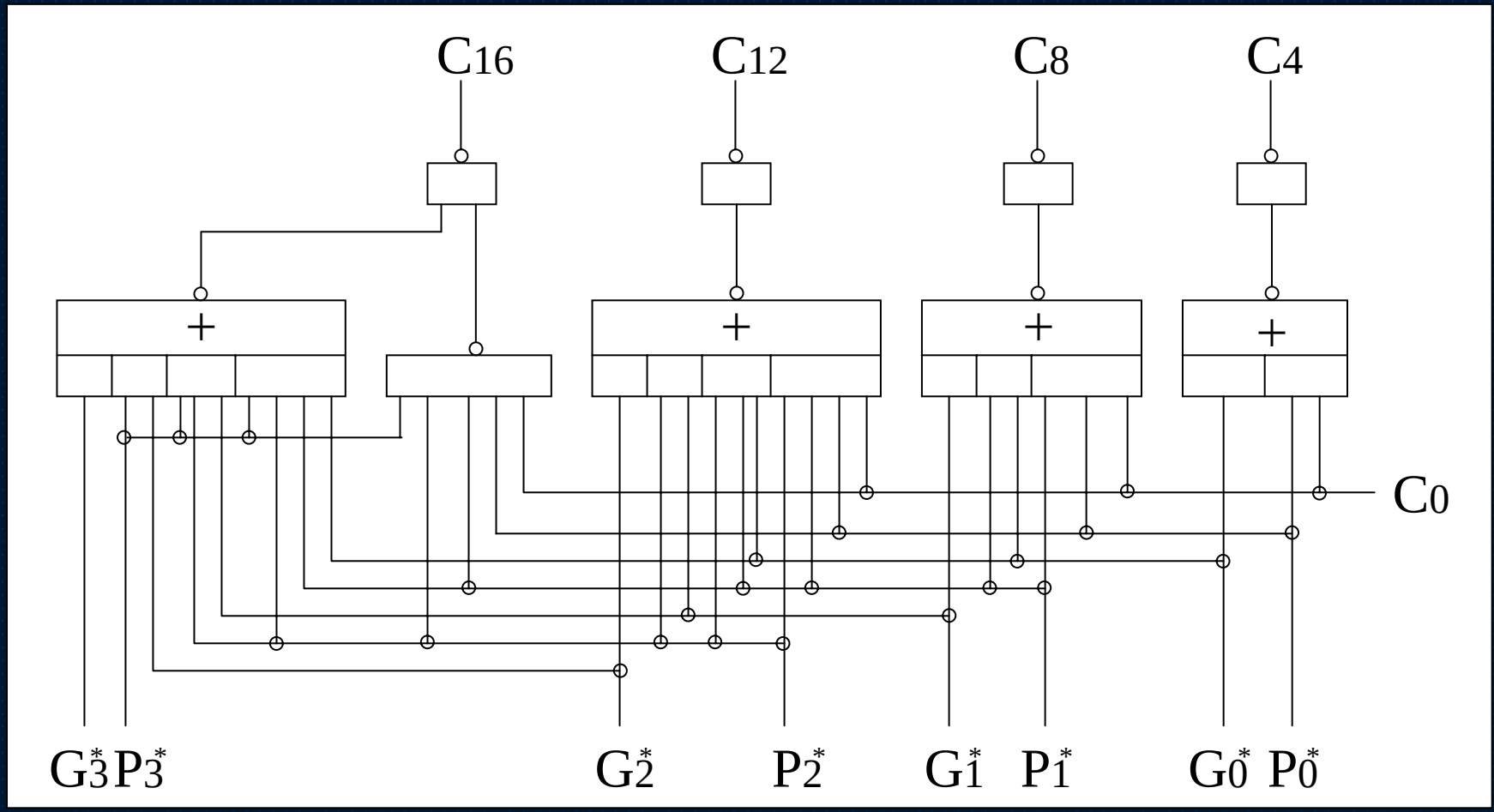
$$C_{12} = G_2^* + P_2^* C_8$$

$$= G_2^* + P_2^* G_1^* + P_2^* P_1^* G_0^* + P_2^* P_1^* P_0^* C_0$$

$$C_{16} = G_3^* + P_3^* C_{12}$$

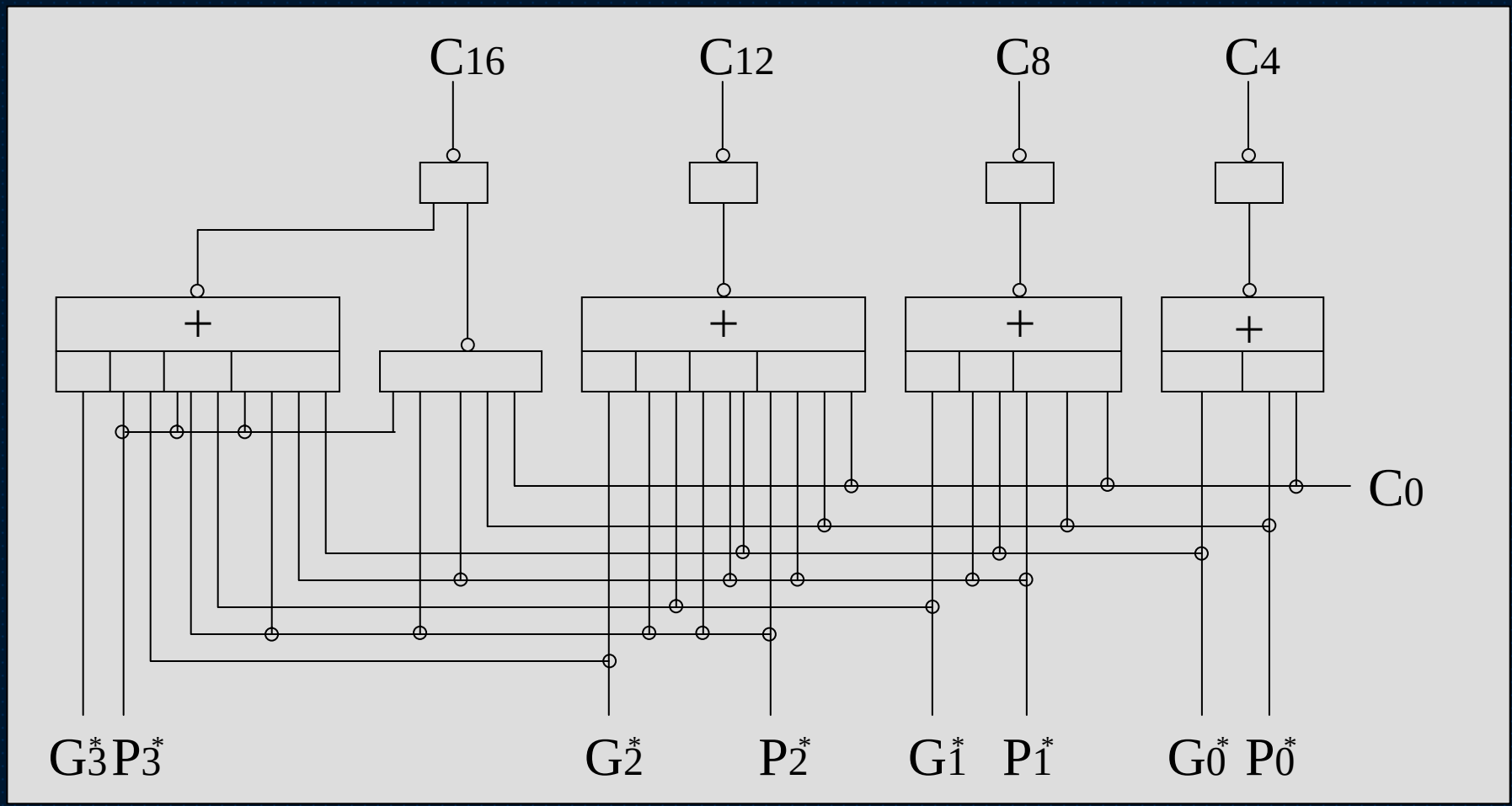
$$= G_3^* + P_3^* G_2^* + P_3^* P_2^* G_1^* + P_3^* P_2^* P_1^* G_0^* + P_3^* P_2^* P_1^* P_0^* C_0$$

- 当G_i^*、P_i^*及C_0形成后，C_4、C_8、C_{12}和C_{16}便可同时产生。



- C4、C8、C12和C16已由组间进位线路产生，组内并行进位线路不需要再产生这些进位，将其作适当修改，便可产生小组的本地进位Gi*和小组的传送条件Pi*以及小组内的低3位进位。

- 例：16位加法器采用组内并行、组间并行进位结构的框图。
- 进位形成过程如下：从A_i、B_i、C_0输入开始；经过4T形成C_1、C_2、C_3及全部G_i^*、P_i^*；又经过2.5T形成C_4、C_8、C_{12}、和C_{16}；最后再经 2.5T 形成二、三、四、小组内的其余进位C7~5、C11~9、C15~13。

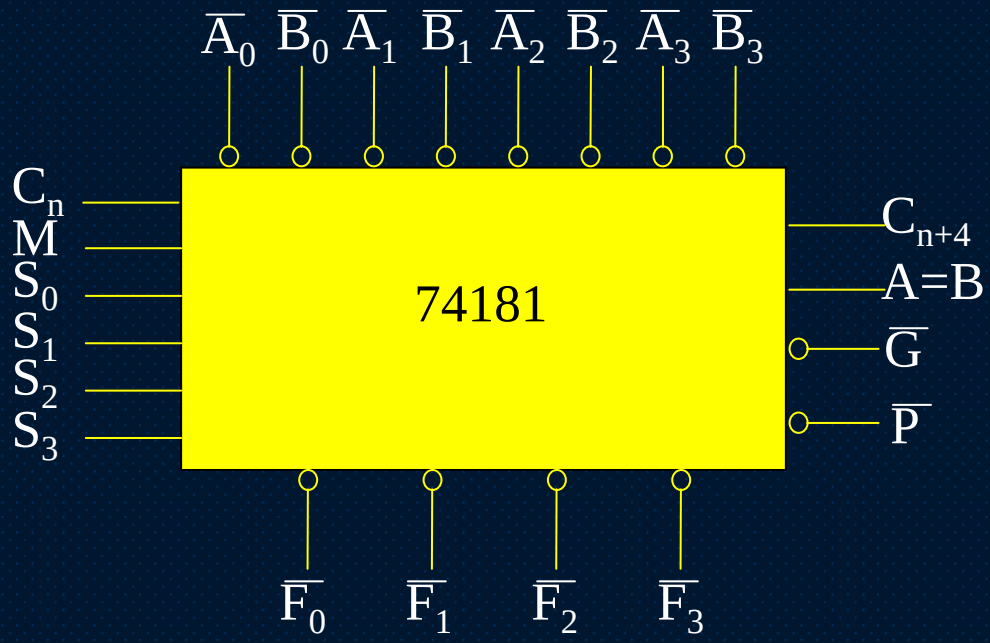


最长进位形成时间：(4+2.5+2.5)T = 9T

四、运算器举例

1、74181

- 算术逻辑单元，简称ALU，具有组内并行进位链，提供了辅助函数G，P供组间进位链使用。



四、运算器举例

2、74182（先行进位发生器）

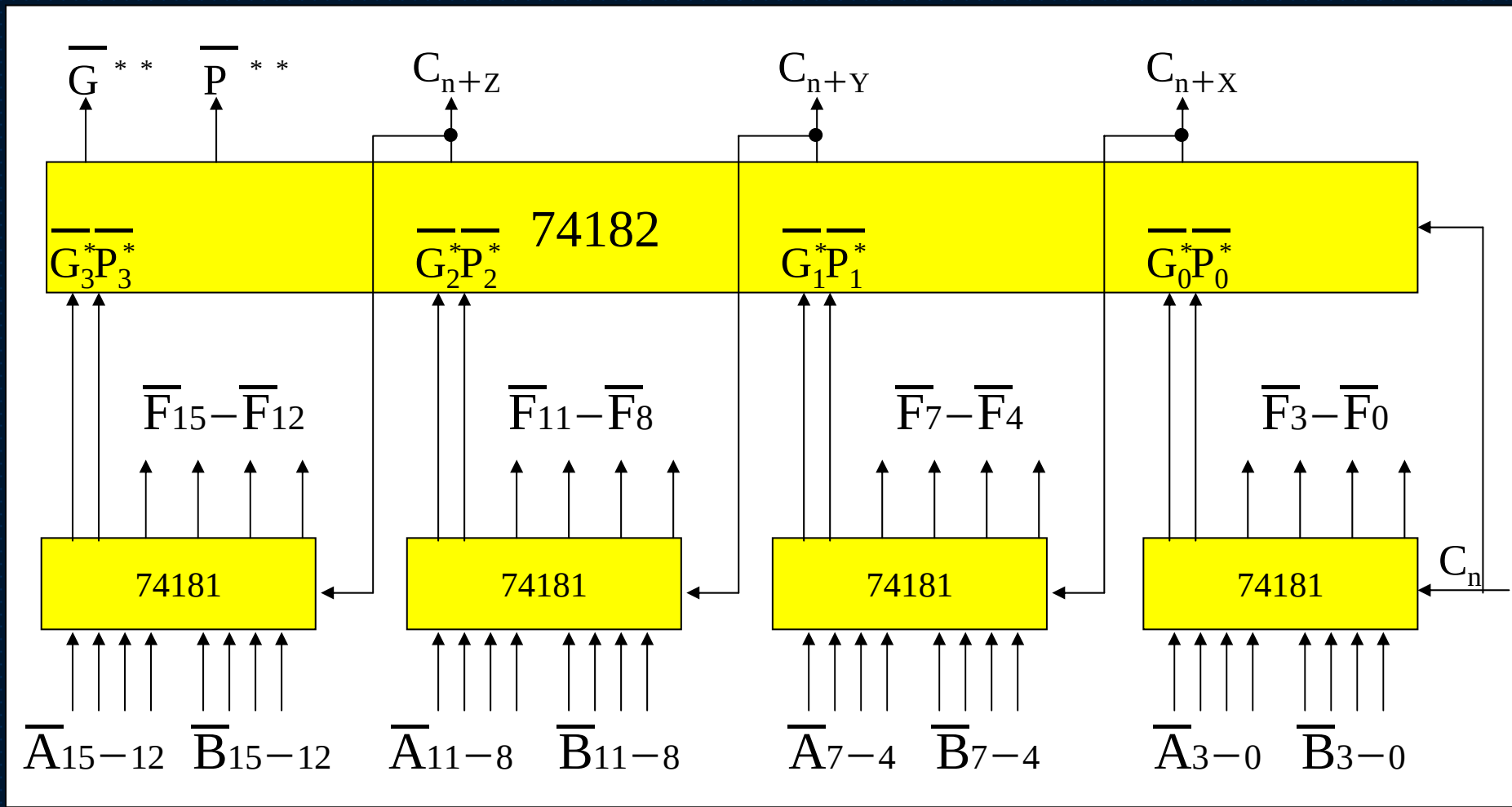
提供：组间并行进位信号C_{n+x}，C_{n+y}，C_{n+z}

小组辅助函数：$\overline{P}$，$\overline{G}$

例：用74181和74182组成16位分二级同时进位的加法器。

利用并行进位链74182可产生向高一级进位链提供辅助函数$\overline{G}^{**}$、$\overline{P}^{**}$，用于位数更长时，组成第三级并行进位链。

16位并行进位ALU结构



第三节　定点乘法运算

在计算机中实现乘除法运算的三种方式：

- 软件实现；
- 在原有ALU的基础上增加一些逻辑线路以实现乘除运算；
- 设置专用的乘除法器。

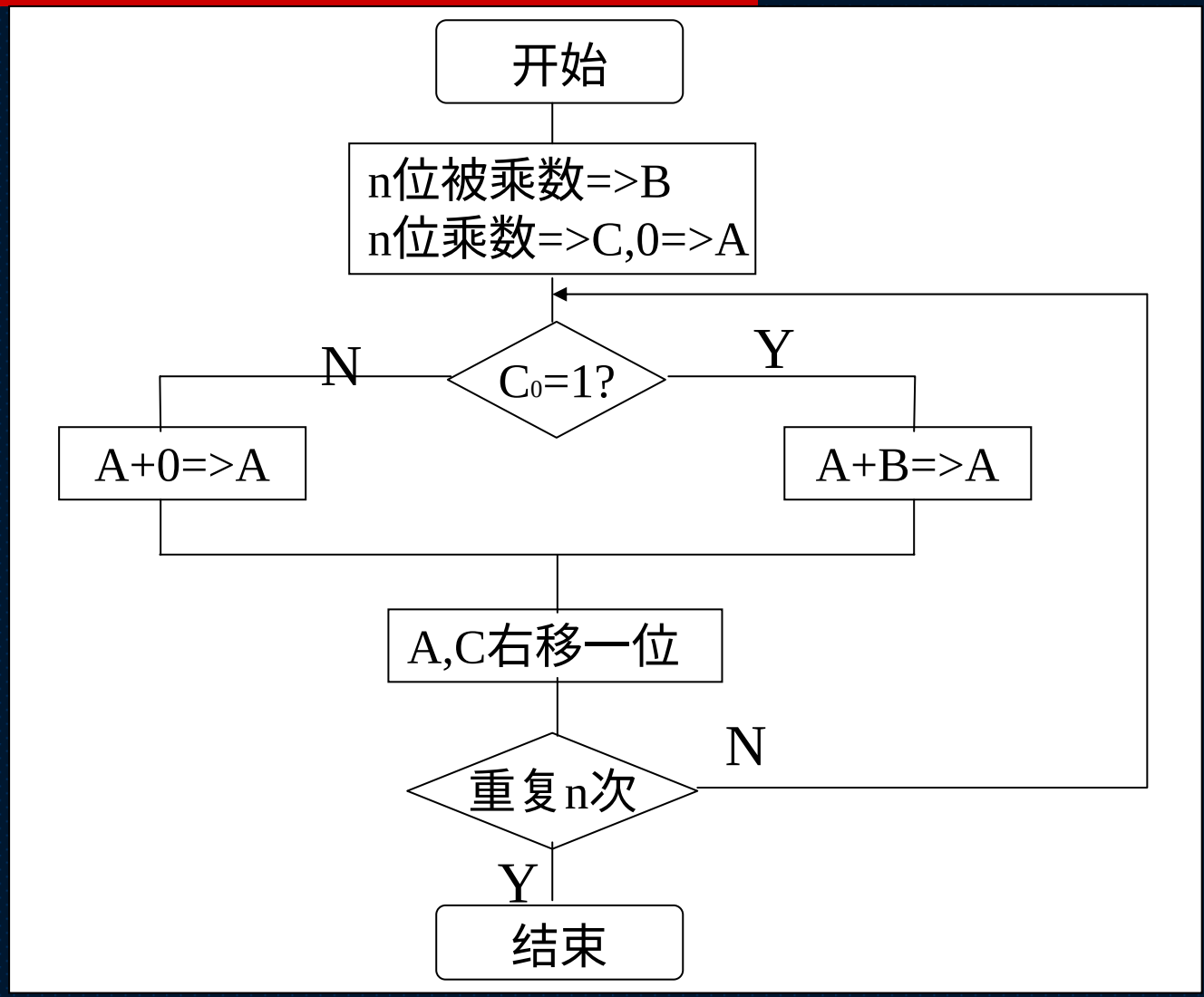
一、无符号数一位乘

例： x=0.1101　　y=0.1011

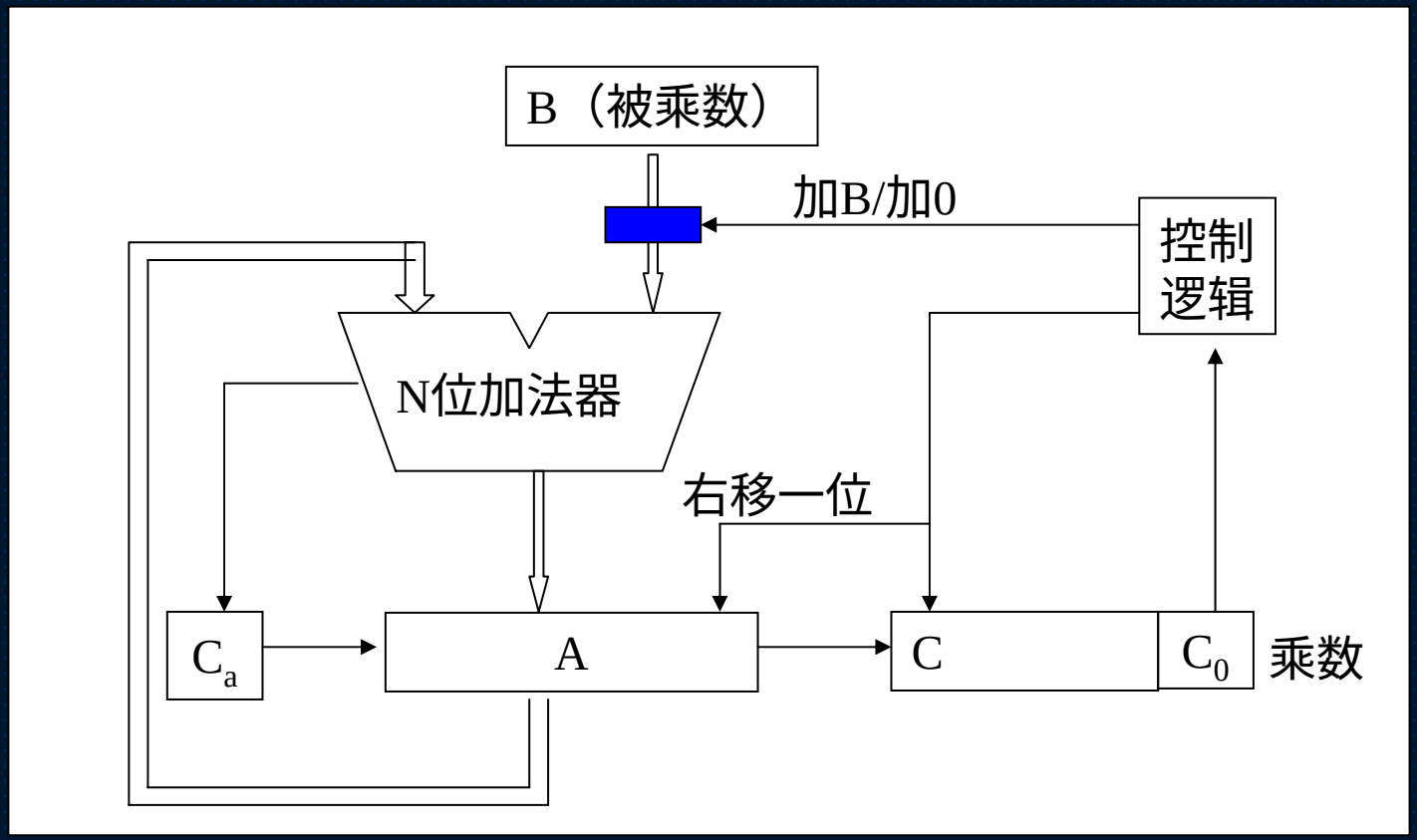
$$
\begin{array}{r}
0.1101 \\
\times\ 0.1011 \\
\hline
1101 \\
1101 \\
0000 \\
1101 \\
\hline
0.10001111
\end{array}
$$

计算机计算：将n位乘转化为n次“累加与移位”。每一步只求一位乘数所对应的新部分积，并与原部分积作一次累加，然后移位一次。

无符号数一位乘算法流程图



硬件实现无符号数一位乘



例：1101×1011的运算过程如下：

B=1101, C=1011

	C_a	A	C
	0	0 0 0 0	1 0 1 1
C_0=1		+B 1 1 0 1	1 0 1 1
	0	1 1 0 1	
→	0	0 1 1 0	1 1 0 1
C_0=1		+B 1 1 0 1	
	1	0 0 1 1	
→	0	1 0 0 1	1 1 1 0
C_0=0		+0 0 0 0 0	
	0	1 0 0 1	
→	0	0 1 0 0	1 1 1 1
C_0=1		+B 1 1 0 1	
	1	0 0 0 1	
→	0	1 0 0 0	1 1 1 1

1101 × 1011=10001111

软件实现无符号数一位乘

```
DATA   SEGMENT
NUM1   DW XXXXH
NUM2   DW XXXXH
DATA   ENDS


MULT   PROC
          MOV     BX，NUM1        ；取被乘数
          MOV     AX，NUM2        ；取乘数
          MOV     CX，16          ；加和移位次数送计数器CX
   LOP:   TEST    AX，01H         ；测试一个乘数位
          JE       NEXT           ；乘数位＝0，转移
          ADD     DX，BX          ；乘数位＝1，被乘数加在部分积上
NEXT:     RCR     DX，1
          RCR     AX，1           ；部分积、乘数联合右移一位
          LOOP    LOP             ；CX－1＝>CX，若CX!=0继续
          RET                     ；CX＝0，已求得乘积返回
MULT   ENDP
```

二、带符号数一位乘法

1.校正法（补码乘法算法的推导）

（1）被乘数x符号任意，乘数y符号为正

设：$[x]_{补} = x_n.x_{n-1}\cdots x_1 .x_0$

$[y]_{补} = 0.y_{n-1}y_{n-2}\cdots y_1 y_0$

根据补码定义：

$[x]_{补} = 2+x \pmod 2$

$[y]_{补} = y = 0.y_{n-1}y_{n-2}\cdots y_1 y_0$

$[x]_{补}[y]_{补} = 2\cdot y + x\cdot y = 2(y_{n-1}\cdots y_1 y_0)+x\cdot y \pmod 2$

$\because 2(y_{n-1}\cdots y_1 y_0) = 2 \pmod 2$

$\therefore [x]_{补}[y]_{补} = 2+x\cdot y = [x\cdot y]_{补} \pmod 2$

即：$[x\cdot y]_{补} = [x]_{补}[y]_{补} = [x]_{补}\cdot y = [x]_{补}\cdot(0.\ y_{n-1}\cdots y_1 y_0)$

（2）被乘数x符号任意，乘数y为负

$[x]_{补} = x_n.x_{n-1}x_{n-2}\cdots x_0$

$[y]_{补} = 1.\ y_{n-1}\cdots y_1\ y_0 = 2+y \pmod 2$

$y = [y]_{补}-2 = 1.\ y_{n-1}\cdots y_1\ y_0 - 2 = 0.\ y_{n-1}\cdots y_1\ y_0 - 1$

$x\cdot y = x(0.\ y_{n-1}\cdots y_1\ y_0)-x$

$[x\cdot y]_{补} = [x(0.\ y_{n-1}\cdots y_1\ y_0)-x]_{补}$

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

$= [x(0.\ y_{n-1}\cdots y_1\ y_0)]_{补} - [x]_{补}$

$= [x]_{补}\cdot(0.\ y_{n-1}\cdots y_1\ y_0) - [x]_{补}$

（3）当被乘数x和乘数y符号任意，以补码表示：

$[x \cdot y]_{补} = [x]_{补}(0.\ y_{n-1} \cdots y_1\ y_0) - [x]_{补} \cdot y_n$

- $y \geq 0$ ：$y_n = 0$ 不需校正
- $y < 0$ ：$y_n = 1$ 需要校正（$-[x]_{补}$）

2.补码乘法比较法——布斯（Booth）乘法

$$[x\cdot y]_{补}=[x]_{补}(0.\ y_{n-1}\cdots y_1\ y_0)-[x]_{补}\cdot y_n$$

- 运算法则

$$[x\cdot y]_{补}=[x]_{补}[-y_n+y_{n-1}2^{-1}+y_{n-2}2^{-2}+\cdots+y_0 2^{-n}]$$

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

$$+(y_0 2^{-(n-1)}-y_0 2^{-n})]$$

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

$$(0-y_0)2^{-n}]$$

$$=[x]_{补}(y_{n-1}-y_n)+2^{-1}(\ [x]_{补}(y_{n-2}-y_{n-1})\ +\ 2^{-1}(\ [x]_{补}(y_{n-3}-$$

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

设：$(y_{-1}=0)$

- 递推公式：

$$[p_0]_{补}=0$$

$$[p_1]_{补}=2^{-1}([p_0]_{补}+(y_{-1}-y_0)[x]_{补})$$

$$[p_2]_{补}=2^{-1}([p_1]_{补}+(y_0-y_1)[x]_{补})$$

$$\cdots\cdots$$

$$[p_i]_{补}=2^{-1}([p_{i-1}]_{补}+(y_{i-2}-y_{i-1})\ [x]_{补})$$

$$\cdots\cdots$$

$$[p_n]_{补}=2^{-1}([p_{n-1}]_{补}+(y_{n-2}-y_{n-1})\ [x]_{补})$$

$$[p_{n+1}]_{补}=[p_n]_{补}+(y_{n-1}-y_n)[x]_{补}=[x\cdot y]_{补}$$

- 每一步乘法在前次部分积的基础上，根据 $y_{i-2}-y_{i-1}$ (i=1,2…n) 的值决定对 $[x]_{补}$ 进行什么操作，然后右移一位，得到新的部分积。重复n步。第n+1步由$(y_{n-1}-y_n)$的值决定对$[x]_{补}$的操作但不移位。

Booth算法：

① 参加运算的数用补码表示

② 符号位参加运算

③ 乘数最低位后面增加一位附加位y_{-1}（初值为0），逐次比较相邻两位并按下列规则运算：

y_i	y_{i-1}	$y_{i-1}-y_i$	操 作
0	0	0	部分积加 0，右移一位
0	1	1	部分积加 $[x]_{补}$，右移一位
1	0	-1	部分积加 $[-x]_{补}$，右移一位
1	1	0	部分积加 0，右移一位

按上述算法进行n＋1步操作（n是不包括符号位在内的字长），第n＋1步不移位。

④ 移位要按补码的移位规则进行

例： 已知X=1011, Y= -1101, 用比较法求$[X \cdot Y]_{补}$

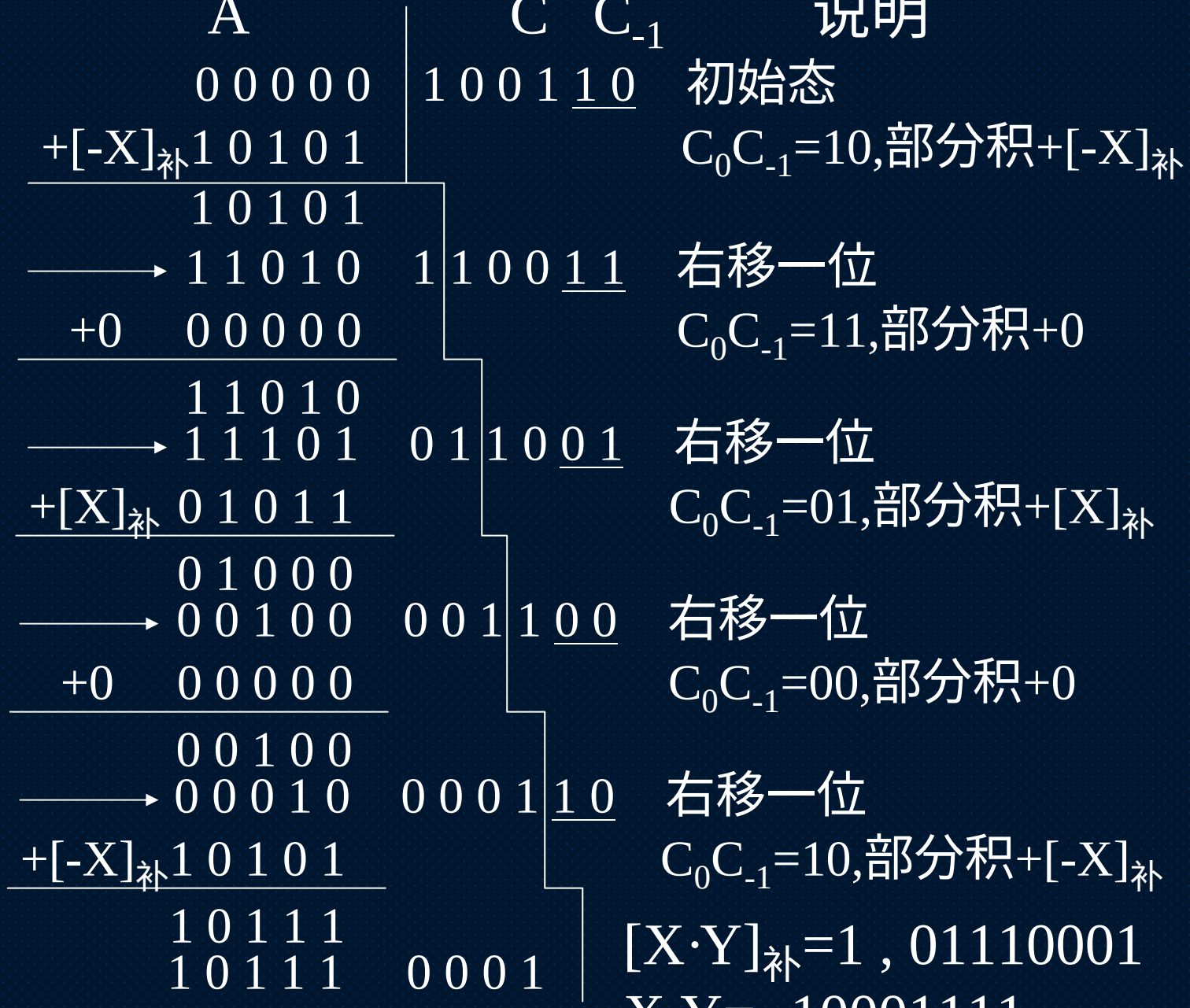
解：部分积存放于A寄存器中，初值为0。

$[X]_{补}$=0，1011，存放于B寄存器中。

$[-X]_{补}$=1，0101

$[Y]_{补}$=1，0011，存放于C寄存器中；

附加位C_{-1} (Y_{-1})置0。


A
C C-1
说明
00000 100110 初始态
+[-X]补 10101 C0C-1=10,部分积+[-X]补
10101
11010 110011 右移一位
+0 00000 C0C-1=11,部分积+0
11010
11101 011001 右移一位
+[X]补 01011 C0C-1=01,部分积+[X]补
01000
00100 001100 右移一位
+0 00000 C0C-1=00,部分积+0
00100
00010 000110 右移一位
+[-X]补 10101 C0C-1=10,部分积+[-X]补
10111
10111 0001
[X·Y]补=1 , 01110001
X·Y= -10001111

三、两位乘简介

补码两位乘法的算法，运算规则如下：

Y_{i+1}	Y_i	Y_{i-1}	操作
0	0	0	部分积 + 0，右移两位
0	0	1	部分积 + $[X]_{补}$，右移两位
0	1	0	部分积 + $[X]_{补}$，右移两位
0	1	1	部分积 + $2[X]_{补}$，右移两位
1	0	0	部分积 + $2[-X]_{补}$，右移两位
1	0	1	部分积 + $[-X]_{补}$，右移两位
1	1	0	部分积 + $[-X]_{补}$，右移两位
1	1	1	部分积 + 0，右移两位

由布斯乘法推出补码两位乘

- $[p_{i+1}]_{补} = 2^{-1}([p_i]_{补} + (y_{i-1} - y_i)[x]_{补})$
- $[p_{i+2}]_{补} = 2^{-1}([p_{i+1}]_{补} + (y_i - y_{i+1})[x]_{补})$
- $[p_{i+2}]_{补} = 2^{-1}\,(2^{-1}([p_i]_{补} + (y_{i-1} - y_i)[x]_{补})$

  $+ (y_i - y_{i+1})[x]_{补})$

  $= 2^{-2}([pi]_{补} + (-2y_{i+1} + y_i + y_{i-1})[x]_{补})$

2.带符号数两位乘

- 乘数数值位数为偶数n，采用双符号位，做n/2+1步加法，n/2步移位(即：最后一步不移位)。
- 乘数数值位数为奇数n，采用一个符号位，做（n+1）/2步加法及移位，最后一步移一位。
- 部分积和被乘数采用三个符号位。

例：已知X=10110, Y= -10101, 用补码两位乘求$[X\cdot Y]_{补}$。

$[X]_{补}$=0,10110，$[Y]_{补}$=1,01011, $[-X]_{补}$=1,01010

$[X]_{补}\Rightarrow B$, $[Y]_{补}\Rightarrow C$, $0\Rightarrow A$, $0\Rightarrow C_{-1}$ （Y_{-1}）

	A	C	C_{-1}	说明
	0 0 0	0 0 0 0 0	1 0 1 0 1 1 0	初始态
$+[-X]_{补}$	1 1 1	0 1 0 1 0		$C_1C_0C_{-1}$=110, $+[-X]_{补}$
	1 1 1	0 1 0 1 0		
2 →	1 1 1	1 1 0 1 0	1 0 1 0 1 0 1	右移二位，
$+[-X]_{补}$	1 1 1	0 1 0 1 0		$C_1C_0C_{-1}$=101, $+[-X]_{补}$
	1 1 1	0 0 1 0 0		
2 →	1 1 1	1 1 0 0 1	0 0 1 0 1 0 1	右移二位，
$+[-X]_{补}$	1 1 1	0 1 0 1 0		$C_1C_0C_{-1}$=101, $+[-X]_{补}$
	1 1 1	0 0 0 1 1		
1 →	1 1 1 ,	1 0 0 0 1	1 0 0 1 0 1	

$[X\cdot Y]_{补}$= 1,1000110010　　$X\cdot Y$= -0111001110

第四节　定点除法运算

一、无符号数一位除

1.无符号数恢复余数法

例：1001010÷1000=1001+10/1000

```
          1001
      ------------
1000 ) 1001010
     - 1000
     ---------
          10
          101
          1010
        - 1000
        --------
          10
```

无符号数恢复余数法算法：

（1）做减法试探 X-Y

若：余数符号为0,被除数>除数，

调整比例因子。

若： 余数符号为1，被除数<除数，

商“ 0 ”，恢复余数。

(2) 被除数（余数）寄存器（A）与商寄存器（C）联合左移一位。

无符号数恢复余数法算法：

(3) 做减法试探，

新余数为正，上一次余数>除数 ,够减,

商“1”，

余数为负，上一次余数<除数，不够减，

商“0”。恢复原来的余数。

(4) 重复(2)(3)步骤，直到商的位数=操作数位数。

余数的符号与权的处理：(小数除法)

(1)让加符号前的商和余数都保持正值,当最后一步不够减时,应恢复余数(即:当余数为负数时,要加上除数,得到真正的余数)

(2)进行了n步除之后,形式上的余数,应乘以2^{-n}才为真正的余数的值

被除数(余数)采用2位符号位 (变形补码)

例: x=0.1011, y=0.1101,求x/y

x=0.1011, y=0.1101, $[-y]_{补}$=1.0011

被除数 (余数):A	商 (C)	
00.1011	0.0000 |	初态(0)作减法
+) 11.0011		$x+[-y]_{补}$余数
11.1110		为负，商0，恢复余
+) 00.1101	0.000|0	数+y
00.1011		(1)余数与商左
01.0110	0.00|0	移一位，减除数
+) 11.0011		$+[-y]_{补}$，余数为
00.1001	0.00|01	正,商“1”
01.0010	0.0|01	(2)余数与商左
+) 11.0011		移一位，减除数
00.0101	0.0|011	$+[-y]_{补}$
00.1010	0.|011	余数为正,商“1”

00.1010	0.|011	(3)余数与商左移一位，减除数
+) 11.0011		$+[-y]_{补}$ 余数
11.1101	0.|0110	为负,商“0”，
+) 00.1101		恢复余数+y
00.1010		(4)余数与商左移一位，减除数$+[-y]_{补}$
01.0100	|0.110	
+) 11.0011		
00.0111	0.1101	余数为正,商“1”

商值为:0.1101 ,

x/y的商= 0.1101　　余数为: 0.0111×2^{-4}

$x/y= 0.1101+ 0.0111/0.1101 \times 2^{-4}$

2. 无符号数不恢复余数法（加减交替法）

对恢复余数法进行修正

若$r_i \geq 0$,商“1”，下一步：$r_{i+1} = 2r_i - y$。

若$r_i < 0$,商“0”，恢复余数$r_i + y$，下一步

$$r_{i+1} = 2(r_i + y) - y$$

$$= 2r_i + 2y - y$$

$$= 2r_i + y$$

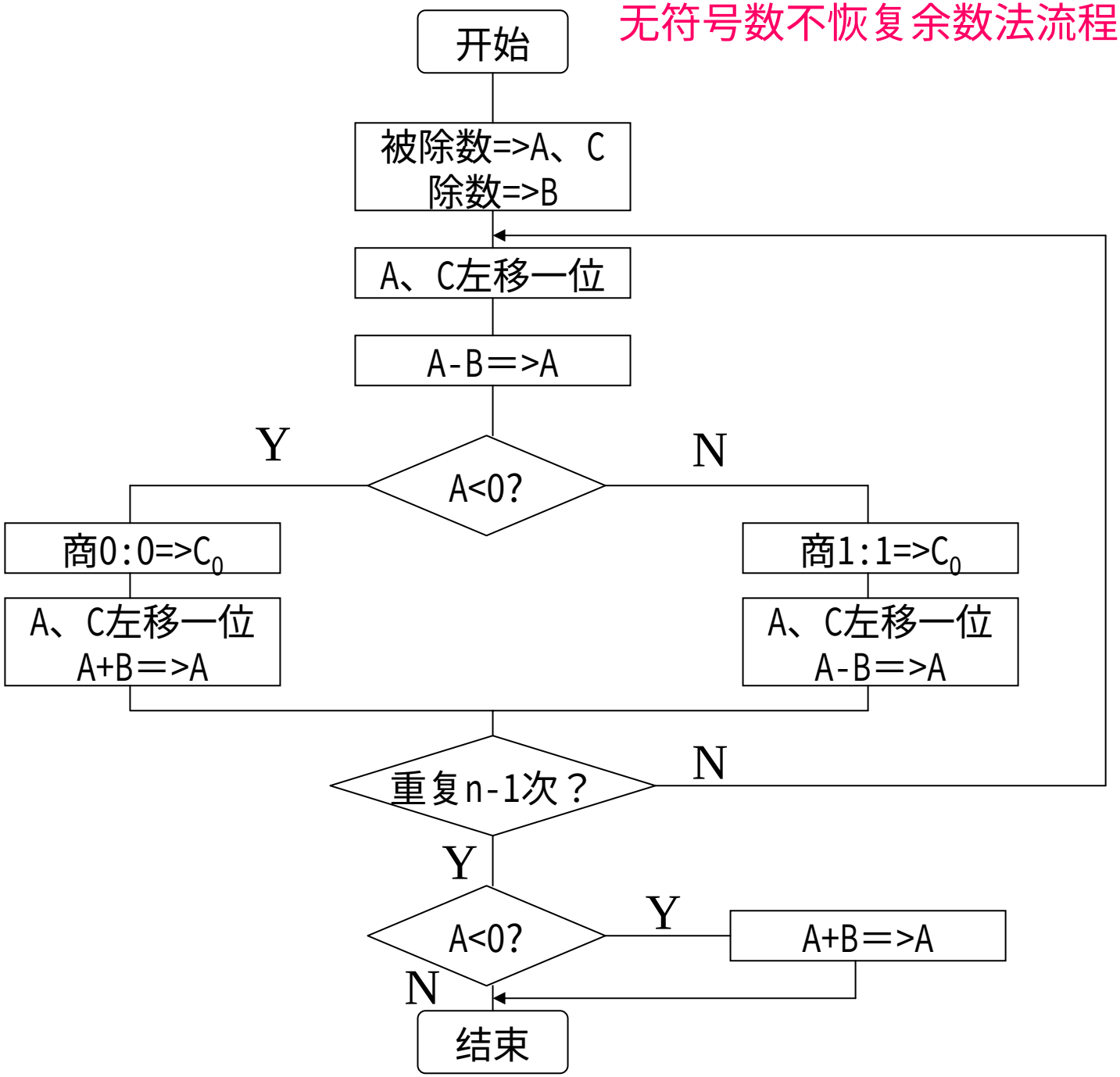
不恢复余数除法规则：

（1）当余数为正，商上“1”，做$2r_i-y$的运算。

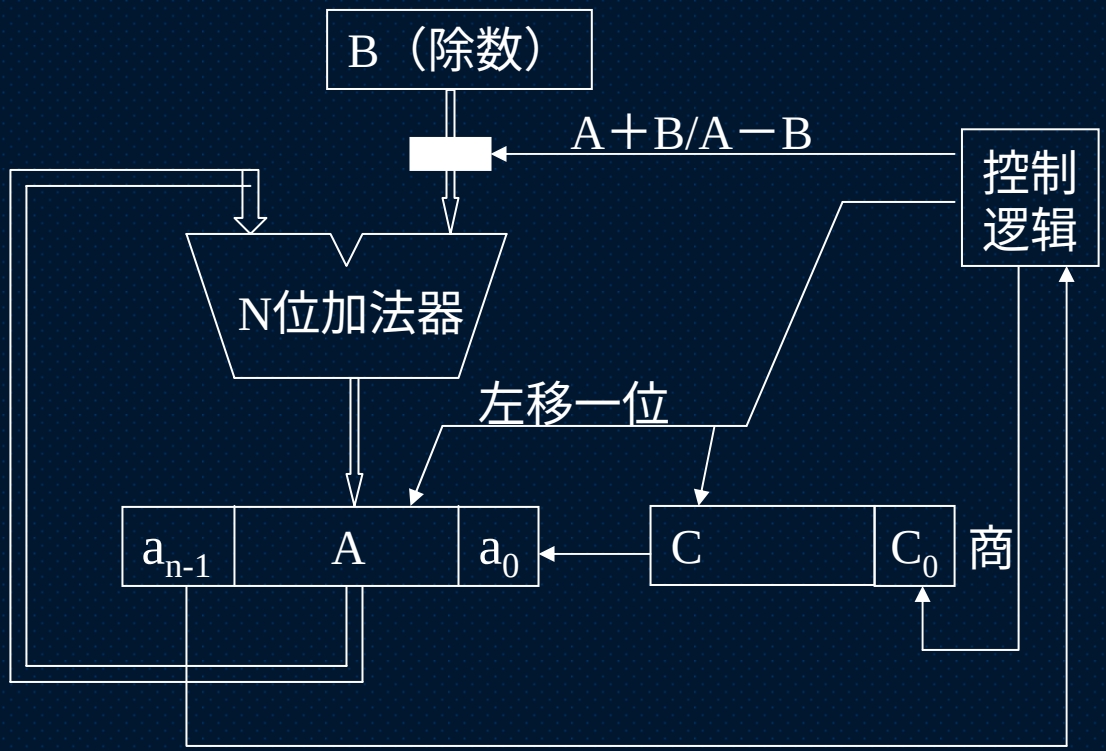
（2）当余数为负，商上“0”，做$2r_i+y$的运算。

（3）重复（1）、（2），上商n（+1）次，左移n次。

无符号数不恢复余数法流程图



无符号数不恢复余数法原理框图



例: x=1000, y=0011,求x/y

x=00001000, y=0011, $[-y]_{补}$=1101

被除数 (余数):A	商 (C)	
0 0 0 0	1 0 0 0|	初态
0 0 0 1	0 0 0|0	A、C左移一位
+) 1 1 0 1		减除数
1 1 1 0	0 0 0|0	为负，商0，下步左移后+y
1 1 0 0	0 0|0 0	A、C左移一位
+) 0 0 1 1		加除数
1 1 1 1	0 0|0 0	为负，商0，下步左移后+y
1 1 1 0	0|0 0 0	A、C左移一位
+) 0 0 1 1		加除数
0 0 0 1	0|0 0 1	为正，商1，下步左移后-y

被除数 (余数):A　　商 (C)

0 0 1 0	|0 0 1 0	A、C左移一位
+) 1 1 0 1		减除数
1 1 1 1	|0 0 1 0	为负，商0
+) 0 0 1 1		恢复余数
0 0 1 0		

商值为:0010 , 余数为：0010

x/y= 0010+ 0010/0011

第五节　浮点运算

一、浮点加减运算及实现

- 设有两个浮点数x和y

$$x = 2^{E_x} \cdot M_x$$

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

M_x和M_y分别为x和y的尾数，E_x和E_y分别为x和y的阶码。

运算步骤

1.对阶

- 阶码：反映了数的小数点位置。
- 浮点数相加减要求阶码相等，小数点的位置对齐，这个过程称为对阶。

①求阶差　　$\triangle E = E_x - E_y$

- 若$\triangle E=0$，不需对阶。若$\triangle E\neq 0$，需对阶，按$|\triangle E|$ 调整阶码。

②保留大阶

- 对阶的规则：小阶向大阶看齐。

③阶码小的尾数向右移位，每右移一位阶码加1，直至阶差为0。

例：两浮点数为 $x = 0.1101\times2^{01}$

$y = -(0.1010)\times2^{11}$，求x+y=?

解：x、y以补码表示：

$[x]_{补}$=00，01；00.1101　　$[y]_{补}$=00，11；11.0110

E_x　M_x　　E_y　M_y

1. 对阶

[△E]补=[Ex]补-[Ey]补

=00,01+11,01=11,10

即 △E = -2，x阶码 < y阶码，将x的尾数右移两位，E_x+2，|△E| = 0，对阶完毕。

$[x]_{补}'$ = 00,11;00.0011

2.求和/差

$[x]_{补}$ = 00,11;00.0011

$[y]_{补}$ = 00,11;11.0110

则 $[Mx+My]_{补}$ 为

$$\begin{array}{r} 00.0011 \\ +)\ 11.0110 \\ \hline 11.1001 \end{array}$$

3.规格化

（1）左规

- 左规条件：运算后结果尾数的符号位与尾数第一位相等。
- 左规：尾数左移，每左移一位，阶码减1，直至尾数符号与尾数第一位不相等。

$[x+y]_{补}$ = 00,11;11.1001

尾数左移一位，阶码减一

$[x+y]_{补}$ = 00,10;11.0010

$x+y = -0.1110 \times 2^{10}$

（2）右规

- 右规条件：运算后结果尾数的两位符号位不等。
- 右规：将尾数右移一位，阶码加1。

例：两浮点数：$x=0.1101\times2^{10}$，$y=0.1011\times2^{01}$。求$x+y=?$

经对阶、求和 $[x+y]_{补}=00,10;01.0010$

右规：将尾数右移一位，阶码加1，

$[x+y]_{补}=00,11;00.1001$

$x+y=0.1001\times2^{11}$

4.舍入

（1）0舍1入

- 右移时被丢掉数位的最高位为0，则舍去；被丢掉数位的最高位为1，则将尾数的末位加1。

例如：对 01.0100×2^{10} 进行右规，得：

00.1010×2^{11}；

对 01.1011×2^{01} 进行右规，得：

00.1110×2^{10}。

- 补码有一种情况例外：当负数补码被丢掉的数位的最高位为1，其它各位均为0时，此“1”应该舍去。
  - 若舍入后又造成尾数溢出，须再进行右规。

例：对补码表示的负数1.01101000进行舍入（保留小数点后四位有效数字），结果为1.0110。

4.舍入

（2）恒置1法

- 只要数位被移掉，就在尾数的末位恒置“1”。

上例1、 01.0100×2^{10} 右规：
00.1011×2^{11}

上例2、 01.1011×2^{01} 右规：
00.1101×2^{10}

5.浮点数的溢出判断

- 设有阶码m位（包括一位符号位），采用补码（或移码）表示，则表数范围为：
- $-2^{m-1} \leq E \leq 2^{m-1}-1$。

- 当 阶码$>2^{m-1}-1$时，称为阶码上溢
- 当 阶码$<-2^{m-1}$时，称为阶码下溢

二、浮点乘除运算及实现

- 设两个浮点数分别为 $x=2^{Ex}\cdot M_x$，$y = 2^{Ey}\cdot M_y$

1、乘法运算：

$$x\cdot y=2^{(Ex+Ey)}(M_x\cdot M_y)$$

- 乘积的阶码为：两数的阶码之和。
- 乘积的尾数为：两数的尾数之积。

2、除法运算：

$$x/y = 2^{(Ex-Ey)}(M_x/M_y)$$

- 商的阶码为：被除数的阶码减去除数的阶码之差。
- 商的尾数为：被除数的尾数除以除数的尾数所得的商。

- 讨论阶码运算（讨论对移码的运算规则和判定溢出的方法）。

（一）阶码加减运算（移码表示的阶码）

[x]移与[x]补：符号位相反，其数值位相同。

$[x]_{移}=2^n+x \quad -2^n \leq x < 2^n$

$[x]_{移}+[y]_{移}=2^n+x+2^n+y=2^n+(2^n+x+y)=2^n+[x+y]_{移}$

$[x]_{移}+[y]_{移}+2^n=[x+y]_{移} \quad (\mathrm{mod}\, 2^{n+1})$

$[y]_{补}=2^{n+1}+y$ (整数补码)

又∵$[y]_{移}=2^n+y$

∴$[y]_{补}=2^n+2^n+y=2^n+[y]_{移}$

$[x]_{移}+[y]_{补}=[x+y]_{移} \quad (\mathrm{mod}\, 2^{n+1})$

同理$[x]_{移}+[-y]_{补}=[x-y]_{移}$

- 执行阶码加减运算，对加数或减数的移码符号位求反再运算，结果就是正确的移码值。

（二）溢出判断

- 阶码采用双符号位，（最高符号位）恒为0。
- 运算结果符号：

01	结果为正	无溢出
00	结果为负	
10	上溢	
11	下溢	

例：阶码用四位移码表示（包括一位符号位）

$E_x = +110$　　$E_y = +011$

$[E_x]_{移}=01,110$　$[E_y]_{补}=00,011$

$[-E_y]_{补}=11,101$

①$[E_x+E_y]_{移}=[E_x]_{移}+[E_y]_{补}=10,001$　上溢

②$[E_x-E_y]_{移}=[E_x]_{移}+[-E_y]_{补}=01,011$　正确(+3)

当$E_x = -110$　　$E_y = -011$时

$[E_x]_{移}=00,010$　$[E_y]_{补}=11,101$　$[-E_y]_{补}=00,011$

③$[E_x+E_y]_{移}=[E_x]_{移}+[E_y]_{补}=11,111$　结果下溢

④$[E_x-E_y]_{移}=[E_x]_{移}+[-E_y]_{补}=00,101$　结果正确(-3)

例：浮点数相乘，4位阶码（移码表示），8位尾数（用补码表示），（均包括一位符号位）。

已知 $x=2^3\times(0.1001101)$，

$y=2^{-5}\times(-0.1110010)$，求 $x\cdot y$ = ？

解：x的浮点表示形式 1,011；0.1001101

y的浮点表示形式 0,011；1.0001110

移码采用双符号位 $[E_x]_{移}=01,011$

$[E_y]_{补}=11,011$

$[-M_x]_{补}=1.0110011$

乘法步骤：

①阶码相加，$[E_x+E_y]_{移}=[E_x]_{移}+[E_y]_{补}$

$[E_x]_{移}$　　01,011

$[E_y]_{补}$　+）　11,011

$[E_x+E_y]_{移}$　　00,110

②尾数相乘，采用补码两位乘方案。

$[M_x \cdot M_y]_{补}=1.01110110110110$

③规格化处理：

$[M_x \cdot M_y]_{补}=1.01110110110110$

④舍入

设尾数保留8位(包括符号位)，采用0舍1入法。

$[M_x \cdot M_y]_{补} = 1.01110110110110$

尾数为 $[M_x \cdot M_y]_{补} = 1.0111011$。

- x·y的浮点表示为：0,110；1.0111011
- $x \cdot y = 2^{-2} \times (-0.1000101)$。