计算机组成原理

Principle of Computer Organization

第二章 运算方法与运算器

第一部分

北京邮电大学

计算机学院

戴志涛

本章内容

- 数据的表示方法及其机器存储
- 算术运算和逻辑运算的运算方法（算法）
- 运算器的组成（逻辑实现）

数据的表示方法

- 数据
  - 数值数据
    - 定点格式：数值范围有限，处理简单
    - 浮点格式：数值范围很大，处理比较复杂
  - 符号数据
    - ASCII码
    - 汉字
- 选择数据的表示方式需要考虑的因素
  - 要表示的数的类型
  - 可能需要的数值范围
  - 数值精度
  - 数据存储和处理所需要的硬件代价

定点数（fixed-point number）的表示方法

- 定点表示：机器在运算过程中，数据的小数点位置固定不变
- 通常在设计机器时即指定好小数点的位置
- 原则上，小数点可指定在任何位置
  - 将小数点固定在最左边：数据表示成<u>纯小数</u>
  - 将小数点固定在最右边：数据表示成<u>纯整数</u>
- 为表示统一，符号也用数值表示

定点数的表示方法

- 设带符号数用n比特表示数值，1比特表示符号，则定点数 $x = x_1 x_2 \cdots x_n$ 在定点机中表示为

X_0	$X_1\ X_2\ \cdots\ X_{n-1}\ X_n$
符号	尾数（mantissa）

- 纯小数：小数点位于x_0和x_1之间
- 纯整数：小数点位于最低位x_n的右边

定点数的表示范围

- 当n固定、且小数点的位置也固定时，定点数可表示的数的范围是固定的
- 定点纯小数 $|x| = 0.x_1x_2\cdots x_n$

  $0 \le |x| \le 1-2^{-n}$

  $(0.11\ldots\ldots1+0.0\ldots\ldots01=1)$
- 定点纯整数 $|x| = x_1x_2\cdots x_n$

  $0 \le |x| \le 2^n-1 \qquad (|x|+1= 2^n)$

X_0	$X_1\ X_2\ \ldots\ X_{n-1}\ X_n$
符号	尾数（mantissa）

定点数的表示方法

- 优点：表示方法简单，便于运算
- 缺点：表示的数的范围有限，要表示很大或很小的数必须用很多比特
- 大数量级数据的表示
  - 定点计算机间接表示：在运算之前先按照一定的固定比例（比例因子）缩放，在运算之后再恢复
  - 用幂的方式运算

浮点数的表示方法

- 任意进制数N：$N = m \times R^e$
  - m：浮点数的**尾数**（纯小数）
  - e（exponont）：阶，比例因子的指数，称为浮点数的**指数**（整数）
  - R：比例因子的**基数**（常数），在二进制机器中通常规定R为2、8或16

机器浮点数的组成

- 一个机器浮点数由阶和尾数组成
  - 尾数：用定点小数或定点整数表示，给出有效数字，决定浮点数的表示精度
  - 阶：用整数形式表示，指明小数点在数据中的位置，决定浮点数的表示范围

E_S	$E_1\ E_2 \ldots\ E_m$	M_S	$M_1\ M_2 \ldots\ M_n$
阶符	阶码	尾符 数符	尾码
阶		尾数	

十进制数串的表示方法

- 字符串形式（非压缩型）：
  - 一个字节存放一个十进制的数位或符号位
  - 例：1234　“1”“2”“3”“4”
  - 优点：与ASCII码兼容
- 压缩的十进制数串形式：
  - 一个字节存放两个十进制的数位
  - 每个数值位数占用半个字节：BCD码
  - 符号位占半个字节：用四位编码中的六种冗余值

压缩十进制数串表示实例

- **约定**
  - 符号位存放在最低数字位之后
  - 12（c）表示正号，13（d）表示负号
  - 数值位数与符号位数之和必须为偶数

- **例：** +123　　　　-12

1	2	3	C
0001	0010	0011	1100

0	1	2	D
0000	0001	0010	1101

  - 既节省存储空间，又便于直接完成十进制数的算术运算

数的机器码表示

- 数据表示：数值和符号均数值化
- 数据运算：将符号位当作数值位统一参加运算
- 真值和机器码
  - 真值：日常使用的用正负号加绝对值表示数大小的原值
  - 机器码（机器数）：在计算机中使用和表示数的形式，通常将数的符号位和数值位一起编码
  - 真值通过编码转换为机器码存放和参加运算
  - 常用的定点数机器码：原码、补码、反码、移码

原码表示法

- 逻辑定义：在数值前面增加一个符号位，该位为0表示正数，该位为1表示负数
- 举例

$x = +0.1001 \longrightarrow [x]_{原} = 0.1001$

$x = -0.1001 \longrightarrow [x]_{原} = 1.1001$

原码数学定义

➤ 定点小数

- 定义

$$[x]_{原} = \begin{cases} x & 1 > x \geq 0 \\ 1 - x = 1 + |x| & 0 \geq x > -1 \end{cases}$$

- 零：$[+0]_{原}=0.000\ldots0$　　$[-0]_{原}=1.000\ldots0$

➤ 定点整数

- 定义

$$[x]_{原} = \begin{cases} x & 2^n > x \geq 0 \\ 2^n - x = 2^n + |x| & 0 \geq x > -2^n \end{cases}$$

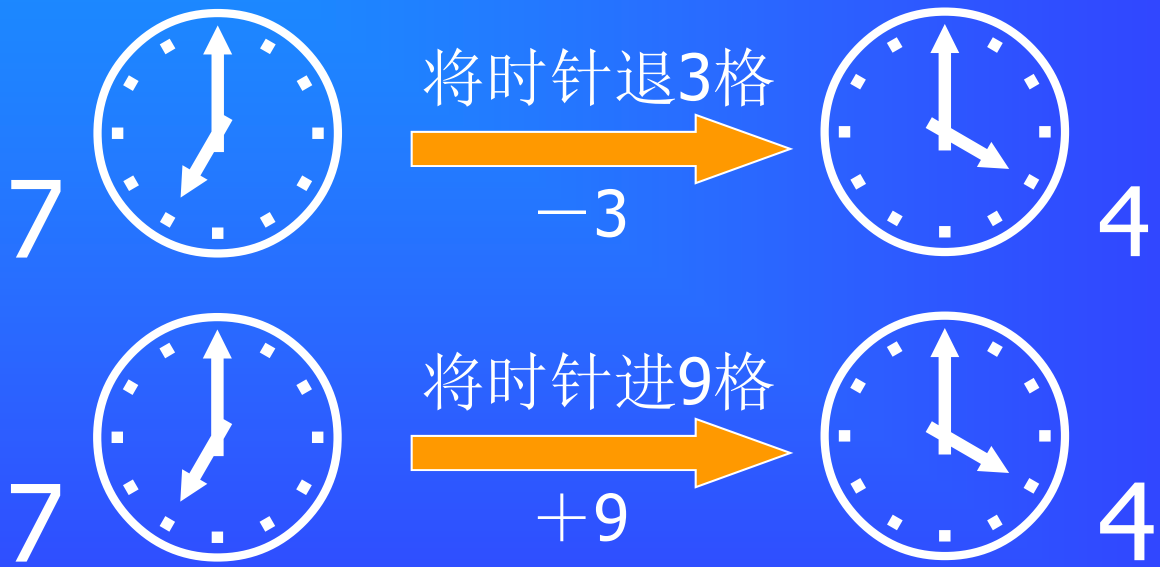
- 零：$[+0]_{原}=0000\ldots0$　　$[-0]_{原}=1000\ldots0$

原码表示的特点

- 优点：
  - 简单直观
  - 便于在真值和机器数之间转换
- 缺点：
  - 加减法运算复杂

补码（complement）表示法

- 引入补码的目的：使加减操作统一，正负数表示和运算统一
- 补码的概念与取模运算（modulus）



$-3 = +9 \pmod{12}$

$7-3 = 7+9 \pmod{12}$

补码数学定义

➢ 定点整数

- 定义

$$[x]_{补}=\begin{cases} x & 2^n > x \geq 0 \\ 2^{n+1}+x=2^{n+1}-|x| & 0 > x \geq -2^n \end{cases} \pmod{2^{n+1}}$$

- 正数的补码的范围是[0，2^n)
- 负数的补码的范围是[2^n，2^{n+1}-1]
- 举例

x=＋0111001 ⟶ $[x]_{补}$=00111001

x=－0111001 ⟶

$[x]_{补}$=100000000＋x=100000000-0111001=11000111

补码数学定义

➢定点小数

1.1111……11

□定义

$$[x]_{补}=\begin{cases} x & 1>x\geq 0 \\ 2+x=2-|x| & 0>x\geq -1 \end{cases} \quad (\mathrm{mod}\ 2)$$

□0的补码表示只有一种形式

□举例

x=＋0.1011 ⟹ $[x]_{补}$=0.1011

x=－0.1011 ⟹

$[x]_{补}$=10＋x=10－0.1011=1.0101

反码表示法

- 逻辑定义：
  - 正数的反码为其本身
  - 负数的反码等于把其相反数的各个二进制位取反

- 实例

$x=+0.1011011 \longrightarrow [x]_{反}=0.1011011$

$x=-0.1011011 \longrightarrow [x]_{反}=1.0100100$

反码的数学定义

- 正数 $x = +0.x_1x_2\dots x_n$，则

$$[x]_{反} = 0.x_1x_2\dots x_n = x$$

- 负数 $x = -0.x_1x_2\dots x_n$，则

$$[x]_{反} = 1.\overline{x_1}\,\overline{x_2}\cdots\overline{x_n}$$

- 当x<0时，

$$[x]_{反} + |x| = 1.\overline{x_1}\,\overline{x_2}\cdots\overline{x_n} + 0.x_1x_2\cdots x_n = 1.11\cdots1 = 2 - 2^{-n}$$

于是 $[x]_{反} = 2 - 2^{-n} - |x|$

反码的数学定义

- **定点小数**
  - **定义**

$$[x]_{反} = \begin{cases} x & 0 \le x < 1 \\ 2 - 2^{-n} - |x| & -1 < x \le 0 \end{cases}$$

  - **零**：$[+0]_{反} = 0.00\ldots0$　$[-0]_{反} = 1.11\ldots1$

- **定点整数**
  - **定义**

$$[x]_{反} = \begin{cases} x & 0 \le x < 2^n \\ (2^{n+1} - 1) + x & -2^n < x \le 0 \end{cases}$$

  - **零**：$[+0]_{反} = 000\ldots0$　$[-0]_{反} = 111\ldots1$

反码与补码的关系

- 比较反码与补码的定义：当x<0时
  - 以纯小数为例
    - $[x]_{反} = (2 - 2^{-n}) + x$
    - $[x]_{补} = 2 + x$
  - 可得到 $[x]_{补} = [x]_{反} + 2^{-n}$
  - 也即：同一个负数的补码和反码只在最低有效比特上差1

通过反码求补码

- 求一个负数的补码：
  - 只要先求得其反码，再在最后一位加上1即得补码
  - 即：符号位置1，其余各位按位取反，然后在最末位上加1 （**纯小数：2^{-n}**）

例：已知X=－0.1011，**求**$[X]_补$

解：

$$[X]_反=1.0100$$

$$[X]_补=1.0101$$

已知原码求补码

- **正数**　　$[X]_{补}=[X]_{原}$
- **负数**
  - 保持原码的符号位不变，其余各位按位取反，末位加1
  - 或：最右边一个非零位及其以右位不变，以左各位取反

例：（X= - 0.1011001000）

$[X]_{原}=$ 1.1011001<u>000</u>

$[X]_{补}=$ 1.0100110111+0.0000000001

$=$ 1.0100111<u>000</u>

已知补码求原码

- 正数 $[X]_{原}=[X]_{补}$
- 负数 $[X]_{原}=[[X]_{补}]_{补}$

保持补码的符号位不变，其余各位按位取反，末位加1

例： $[X]_{补}= 1.01001110$

$[X]_{原}= 1.10110001$

$+0.00000001$

$= 1.10110010$

已知$[X]_{补}$求$[-X]_{补}$

- $[X]_{补}$连同符号位一起，各位按位取反，末位加1

例： $[X]_{补}= 1.01001110$

$[-X]_{补}= 0.10110010$

机器码右移位

- 原则：移位时应保持移位前后机器码的对应关系与真值移位相同：
  - 每右移一位，真值的绝对值减为1/2
- 原码：符号位固定在最高位，左边空出的数值位补0
- 补码和反码：符号位固定在最高位，左边空出的数值位补符号位

例： $[X]_{补} = 1.01001110$

$\Rightarrow [X/2]_{补} = 1.10100111$

补码位数扩展

- 原则：扩展后真值不变
- 定点纯整数
  - 补码：符号位固定在最高位，左边空出的数值位补符号位
    - 例：

      $[-6]_{补} = [-110]_{补} = 1010$

      $[-6]_{补} = [-0000110]_{补} = 11111010$
- 定点纯小数
  - 补码：右边空出的数值位补0
    - 例：

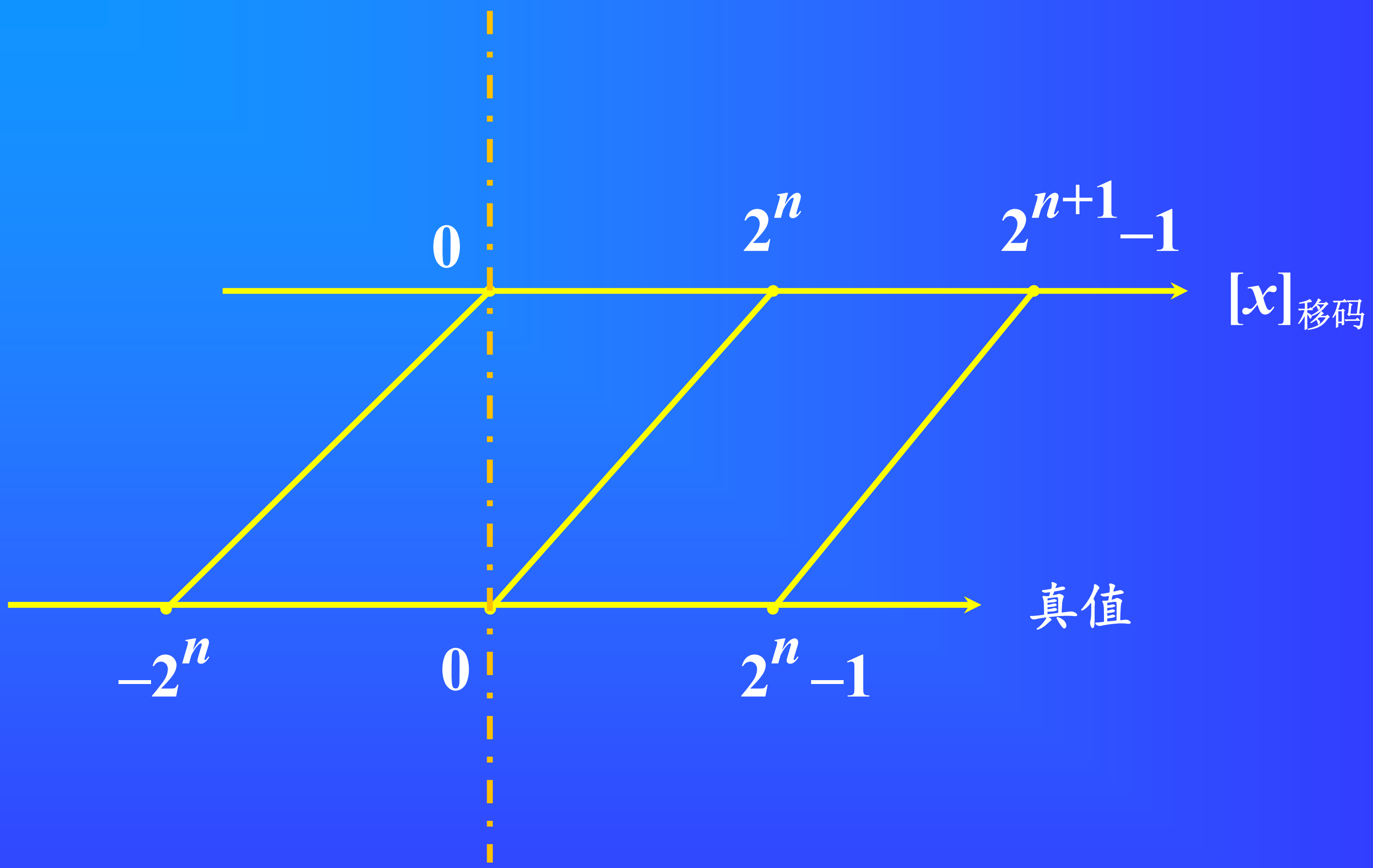
      $[-0.5]_{补} = [-0.100]_{补} = 1.100$

      $[-0.5]_{补} = [-0.1000000]_{补} = 1.1000000$

移码（增码）表示法

- 通常用于表示浮点数的阶
- 为简化操作，使所有阶码均为正数，对所有阶均加上一个固定的正常数（偏置常数）
  - 通常选择偏置常数的值为最负阶的绝对值
  - 若用n位整数表示阶（不含符号位），则偏置常数为2^n
- 对定点整数$x_1x_2\ldots x_n$，定义：
  - $[x]_{移}=2^n+x \qquad 2^n > x \geq -2^n$

移码与真值的关系



移码表示法

- 例：若阶码数值部分为5位，以 x 表示真值，则
  - $[x]_{移} = 2^5 + x \qquad 2^5 > x \geq -2^5$
- 例
  - 正数 x = +10101，则

    $[x]_{移} = 1,10101$

    $[x]_{补} = 010101$
  - 负数 x = −10101，则

    $[x]_{移} = 2^5 + x = 2^5 - 10101 = 0,01011$

    $[x]_{补} = 2^6 + x = 2^6 - 10101 = 101011$

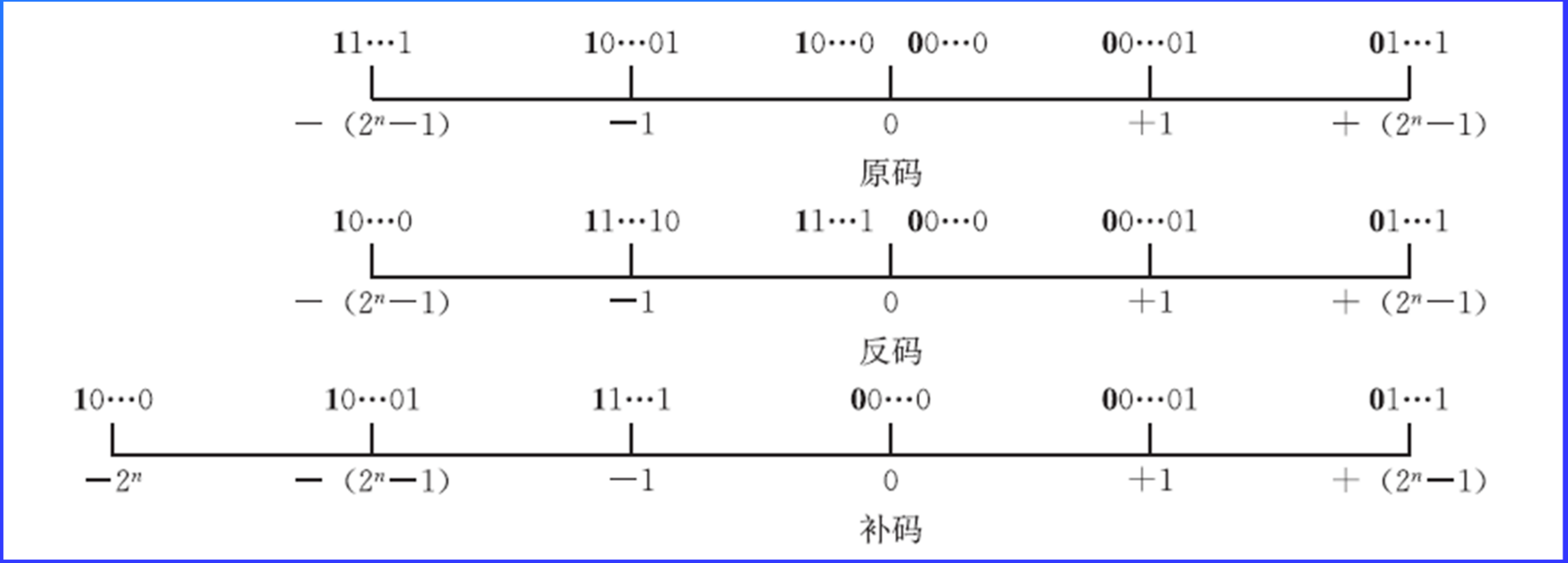
- 移码符号位 x_0 表示的规律与原码、补码、反码相反
- 移码与补码仅差符号位

【例6】机器码的表示范围

以定点整数为例，用数轴形式说明原码、反码、补码的表示范围和可能的数码组合情况。

[解:] 0：原码、反码不同，补码只有一种形式

正负数的表示范围：原码和反码对称，补码多一个负数



【例8】机器码的最值

设机器字长16位，定点表示，尾数15位，数符1位。问：

1）定点原码纯整数表示时，最大正数和最小负数是多少？

2）定点原码纯小数表示时，最大正数和最小负数是多少？

[解:]

1）定点原码整数表示

最大正数值：0111 1111 1111 1111

$$(2^{15}-1)_{10}=(+32767)_{10}$$

最小负数值：1111 1111 1111 1111

$$-(2^{15}-1)_{10}=(-32767)_{10}$$

2）定点原码小数表示

最大正数值 $=(1-2^{-15})_{10}=(+0.111...11)_2$

最小负数值 $=-(1-2^{-15})_{10}=(-0.111..11)_2$

原码、反码、补码和移码的比较

- 解决负数在机器中的表示与运算问题
- 若X为正数，则$[X]_{原}=[X]_{反}=[X]_{补}=X$
- 最高位可以看作符号位：
  - $[X]_{原}$、$[X]_{反}$、$[X]_{补}$用“0”表示正号，用“1”表示负号
  - $[X]_{移}$用“1”表示正号，用“0”表示负号

原码、反码、补码和移码的比较

- 移码与补码的尾码相同，符号位相反
- $[0]_{补}$、$[0]_{移}$有唯一编码，
  $[0]_{原}$、$[0]_{反}$ 有两种编码
- 补码、反码和移码的符号位在加减运算时可以作为数值看待，原码的符号位必须单独处理

课堂练习

一个8位的二进制整数，若采用补码表示，且由3个“1”和5个“0”组成，则最小值为（　　）。

A.-127　　　　B. -32

C.-125　　　　D. +3

【解析】

$[x]_{补} = 1\ 0000011$

$[x]_{原} = 1\ 1111101$

$x = -125$

【解】　C

北京邮电大学

【浮点数的机器表示】（第二章第二部分）

定点加减法运算

- 机器码的存储和运算
  - 实例1：数据用原码存储，用补码运算
  - 实例2：加减法运算用补码，乘除法用原码
- 加减法运算
  - 原码、反码和补码均可进行运算，但算法不同
  - 原码和反码便于与真值相互转换，补码便于加减法运算
  - 补码和反码的符号位与数值位可等同看待
  - 补码应用最广

定点加减法运算

- **原码定点数的加/减运算**
  - 方法与人工运算类似：
    - 同号相加
    - 异号相减：绝对值大的数减绝对值小的数

补码加法

- 含义：从两个加数的补码直接求得和的补码
- 运算公式
  - $[x+y]_{补} = [x]_{补} + [y]_{补}$ (mod 2或2^{n+1})

$$f(x+y) \overset{?}{=} f(x)+f(y)$$

$$f(x)=[x]_{补}$$

补码加法

- 运算公式
  - $[x+y]_{补}=[x]_{补}+[y]_{补}$ (mod 2或2^{n+1})
- 证明：
  - 以定点小数证明
  - 先决条件：
    - $|x|<1$
    - $|y|<1$
    - $|x+y|<1$
    - mod 2
  - 分四种情况证明

补码加法

- **证明：**

(1) $x >= 0$，$y >= 0$：

必有 $x + y >= 0$，

故 $[x + y]_{补} = x + y \pmod 2$

$[x]_{补} = x$，$[y]_{补} = y$，

故 $[x]_{补} + [y]_{补} = x + y \pmod 2$

所以：$[x + y]_{补} = [x]_{补} + [y]_{补}$

补码加法

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

由补码定义，有$[x]_{补} = x$，$[y]_{补} = 2 + y$

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

定点纯小数补码的编码值：在[0,2)之内

- 当$0 =< x + y < 1$时：

  $2 + (x + y) > 2$，进位2被舍弃

  故$[x]_{补} + [y]_{补} = x + y$

  又由定义：$[x + y]_{补} = x + y$

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

- 当$-1 < x + y < 0$时：

  $1 < 2 + (x + y) < 2$

  由定义：$[x + y]_{补} = 2 + x + y$

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

补码加法

(3) $x < 0,\ y >= 0$

由真值相加的交换率：

$[y+x]_{补}=[x+y]_{补}$

由补码相加的交换率：

$[x]_{补}+[y]_{补}=[y]_{补}+[x]_{补}$

由(2)：

$[y+x]_{补}=[y]_{补}+[x]_{补}$

故$[x+y]_{补}=[y+x]_{补}=[y]_{补}+[x]_{补}$

$=[x]_{补}+[y]_{补} \pmod 2$

补码加法

(4) $x<0$， $y<0$

$\because [x]_{补}=2+x$, $[y]_{补}=2+y$

$\therefore [x]_{补}+[y]_{补}=2+x+2+y$

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

由 $-1<x+y<0$

有 $1<2+x+y<2$，故进位2被舍弃

所以 $[x]_{补}+[y]_{补}=2+(x+y) \pmod 2$

由定义：$[x+y]_{补}=2+x+y$

故 $[x]_{补}+[y]_{补}=[x+y]_{补} \pmod 2$

补码加法

例11

已知： $x = 0.1001, y = 0.0101$

求： $x + y$

解：

$[x]_{补} = 0.1001, [y]_{补} = 0.0101$

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

所以 $x + y = +0.1110$

补码加法

例12

已知： $x = +0.1011$, $y = -0.0101$

求： $x + y$

解：

$[x]_{补} = 0.1011$, $[y]_{补} = 1.1011$

$$
\begin{array}{rr}
[x]_{补} & 0.1011 \\
+[y]_{补} & 1.1011 \\
\hline
[x+y]_{补} & 10.0110
\end{array}
$$

所以 $x + y = +0.0110$

补码减法

- 含义：由$[x]_补$和$[y]_补$求得$[x-y]_补$
- 由加法公式推导出补码减法运算公式：
  - $[x-y]_补=[x+(-y)]_补=[x]_补+[-y]_补$
- 从$[y]_补$求$[-y]_补$：
  - 对$[y]_补$包括符号位在内，各位取反，最末位加1
  - $[-y]_补=\neg[y]_补+2^{-n}$
  - 若$[y]_补=y_0.y_1y_2\cdots y_n$，则

$$[-y]_补=\overline{y_0}.\overline{y_1}\cdots\overline{y_n}+2^{-n}$$

补码减法

例11

已知： $x = +0.1101$, $y = +0.0110$

求： $x - y$

解：

$[x]_{补} = 0.1101 \quad [y]_{补} = 0.0110$

$[-y]_{补} = 1.1010$

$$
\begin{array}{rr}
 [x]_{补} & 0.1101 \\
+\ [-y]_{补} & 1.1010 \\
\hline
 [x-y]_{补} & 10.0111
\end{array}
$$

所以 $x - y = +0.0111$

溢出的概念

- 对给定的字长及数据格式，系统所能表示的数据范围是确定的
- 一旦运算结果超出了所能表示的数的范围，就会产生“溢出”（overflow）
  - 8位二进制数纯整数补码：－128～＋127
  - 8位二进制数纯小数补码：－1～＋127/128
- 溢出发生时，计算结果整体而言是不正确的
- 运算器应能发现何时产生了溢出，并使计算机做出相应的反应

溢出如何避免？

溢出的概念

例15

已知： $x = +0.1011$, $y = +0.1001$

求： $x + y$

解:

$[x]_{补} = 0.1011$, $[y]_{补} = 0.1001$

$$
\begin{array}{rr}
[x]_{补} & 0.1011 \\
+\ [y]_{补} & 0.1001 \\
\hline
[x+y]_{补} & 1.0100
\end{array}
$$

两个正数相加的结果成为负数

溢出的概念

例16

已知：$x = -0.1101$，$y = -0.1011$

求：$x + y$

解:

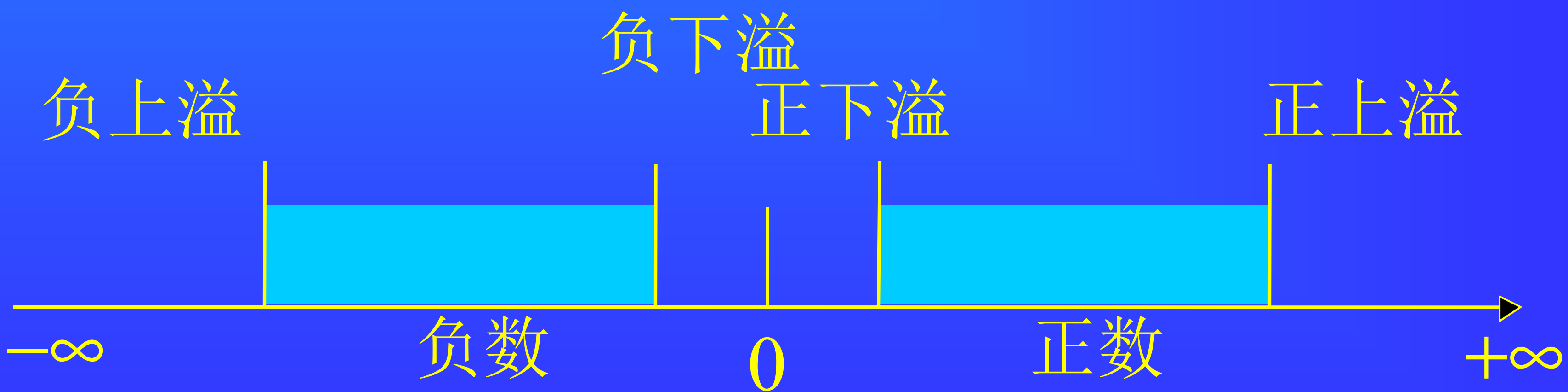
$[x]_{补} = 1.0011$，$[y]_{补} = 1.0101$

$$
\begin{array}{rr}
[x]_{补} & 1.0011 \\
+[y]_{补} & 1.0101 \\
\hline
[x+y]_{补} & 10.1000
\end{array}
$$

两个负数相加的结果成为正数

溢 出

- 定点数：正溢和负溢
  - ❑ 正溢：运算结果为正，且超出所能表示的范围
  - ❑ 负溢：运算结果为负，且超出所能表示的范围



定点数运算溢出检测方法

- 补码甲方运算先决条件：$|x+y|<1$
- 运算之前无法判定，运算器应能检测出运算过程中的溢出
- 溢出检测方法：
  - 原理：两正数相加得负数；两负数相加得正数
  - 单符号位法
  - 双符号位法（变形补码法）

定点数运算溢出检测方法

➢单符号位法

- ❑正溢：最高有效位有进位而符号位无进位
- ❑负溢：最高有效位无进位而符号位有进位
- ❑溢出检测的逻辑表达式：$V=C_f \oplus C_0$
  - ⌧C_f：符号位产生的进位
  - ⌧C_0：最高有效位产生的进位

$$
\begin{array}{rr}
[x]_{补} & 0.1011 \\
+\ [y]_{补} & 0.1001 \\
\hline
[x+y]_{补} & 1.0100
\end{array}
$$

溢出检测方法

- **变形补码法**
  - 采用双符号位“变形补码”（模4补码）
  - 2^1和2^0均为符号位
  - 变形补码的数学定义

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

    - 同余式 $[\,x\,]_{补}=4+x \quad (\mathrm{mod}\ 4)$
  - 运算法则：$[\,x\,]_{补}+[\,y\,]_{补}=[\,x+y\,]_{补} \quad (\mathrm{mod}\ 4)$
    - 两个符号位均当作数值一样参加运算
    - 以4为模的加法

溢出检测方法

变形补码法

变形补码求和结果

- 任何小于1的正数，符号位为00
- 任何大于－1的负数，符号位为11
- 结果的符号位出现“01”或“10”，表示发生溢出
- 最高符号位永远表示结果的正确符号

溢出检测方法

例17

已知： $x = +0.1100$， $y = +0.1000$

求： $x + y$

解：

$$[x]_{补} = 00.1100, [y]_{补} = 00.1000$$

$$\begin{array}{rr} [x]_{补} & 00.1100 \\ +[y]_{补} & 00.1000 \\ \hline [x+y]_{补} & 01.0100 \end{array}$$

两个符号位为“01”，表示已经溢出

溢出检测方法

例18

已知：$x = -0.1100$, $y = -0.1000$

求：$x + y$

解：

$$[x]_{补} = 11.0100,\ [y]_{补} = 11.1000$$

$$\begin{array}{rr} [x]_{补} & 11.0100 \\ +[y]_{补} & 11.1000 \\ \hline [x+y]_{补} & 110.1100 \end{array}$$

两个符号位为“10”，表示已经溢出

溢出检测方法

➢ 变形补码法

❑ 溢出检测规则

- 运算结果的两个符号位相异时，表示溢出
- 运算结果的两个符号位相同时，表示未溢出
- 溢出逻辑表达式： $V = S_{f1} \oplus S_{f2}$
  - ❑ S_{f1}和S_{f2}分别为最高符号位和第二符号位

2011年考研综合应用题

43．（11 分）假定在一个8位字长的计算机中运行如下类C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，这四种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？

2011年考研综合应用题

43．（11分）假定在一个8位字长的计算机中运行如下类C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）……

【解析】：考查溢出概念、机器码与真值之间的转换，以及C语言中强制类型转换操作对数据的处理方式等知识点。

- 注意：
  - 无符号数没有溢出的概念，超出最大值的进位将被丢弃。
  - C语言规定在无符号整数和带符号整数之间进行强制类型转换时，机器码并不改变，改变的是对机器码的解释方式。

2011年考研综合应用题

43．（11分）假定在一个8位字长的计算机中运行如下类C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，R1、R5和R6的内容分别是什么？（用十六进制表示）

答：（1）各寄存器和变量的对应关系如下表所示。

寄存器	R1	R2	R3	R4	R5	R6	R7	R8
变量	x	y	m=x	n=y	z1=x-y	z2=x+y	k1=m-n	k2==m+n
性质	无符号	无符号	带符号补码	带符号补码	无符号	无符号	带符号补码	带符号补码

R1=x=134=10000110b=86h

y=246=11110110b

R5=z1=x-y=134-246=10000110b-11110110b =10000110b+00001010b =10010000b=90h (134-246+256=144=90h)

R6=z2=x+y=134+246=10000110b+11110110b=(1)01111100b=7ch (134+246 - 256=124=7ch)

2011年考研综合应用题

43．（11分）假定在一个8位字长的计算机中运行如下类C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）

（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）

答：（2）各寄存器和变量的对应关系如下表所示。

寄存器	R1	R2	R3	R4	R5	R6	R7	R8
变量	x	y	m=x	n=y	z1=x-y	z2=x+y	k1=m-n	k2==m+n
性质	无符号	无符号	带符号补码	带符号补码	无符号	无符号	带符号补码	带符号补码

$m_{补}$=x=10000110b，m= -1111010b= -7ah= -122

$n_{补}$=y=11110110b，n= -0001010b= -10

$k1_{补}=m_{补}-n_{补}$=10000110b-11110110b

=10000110b+00001010b=10010000b，

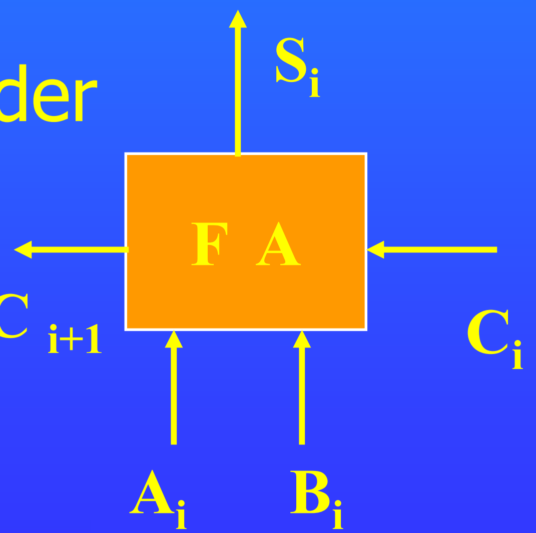
k1= -1110000b= -70h= -112

运算器的实现：
基本二进制加减法器

一位二进制全加器

- 对两个操作数的一个二进制位求和
  - A_i、B_i：第i位加数
  - C_i：进位输入
  - S_i：第i位全加和
  - C_{i+1}：第i位向第i+1位的进位

FA： Full Adder



输入			输出	
A_i	B_i	C_i	S_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

一位二进制全加器

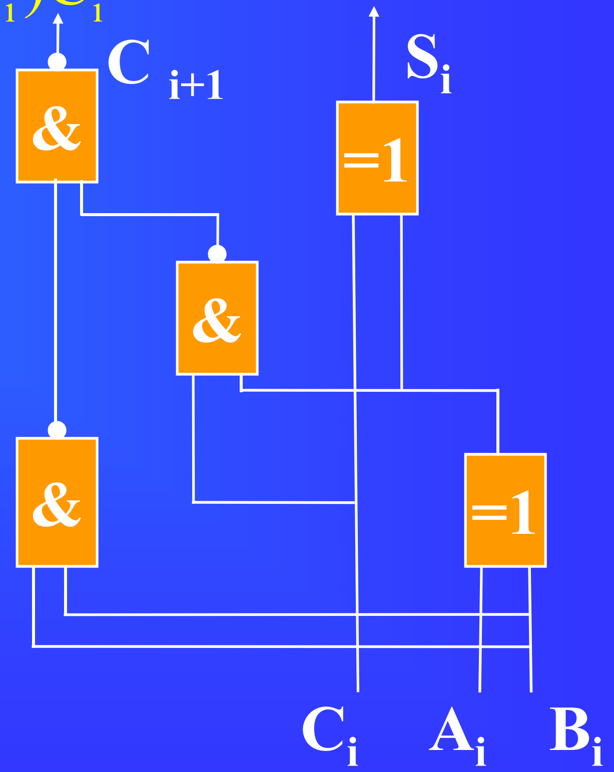
- **一位全加器逻辑表达式**

$$S_i = A_i \oplus B_i \oplus C_i = A_iB_iC_i + A_i\overline{B_i}\,\overline{C_i} + \overline{A_i}B_i\overline{C_i} + \overline{A_i}\,\overline{B_i}C_i$$

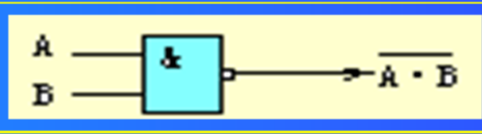
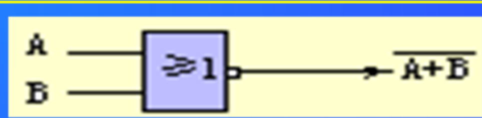
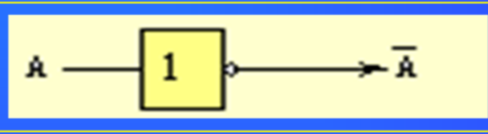
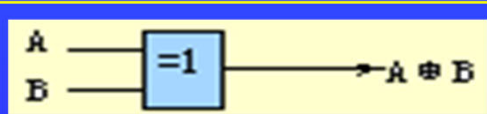
$$C_{i+1} = A_iB_i + B_iC_i + A_iC_i = A_iB_i + (A_i \oplus B_i)C_i$$

$$= A_iB_i + (A_i + B_i)C_i = \overline{\overline{A_iB_i} \bullet \overline{(A_i \oplus B_i)C_i}}$$

- **延迟时间：从输入数据信号有效到输出数据信号有效的时间**



典型门电路的逻辑符号和时间延迟

门的名称	门的功能	逻辑符号(正逻辑)	时间延迟
与非	NAND		T
或非	NOR		T
非	NOT		T
与	AND		2T
或	OR		2T
异或	XOR		3T

一位二进制全加器

➢ **一位全加器逻辑表达式**

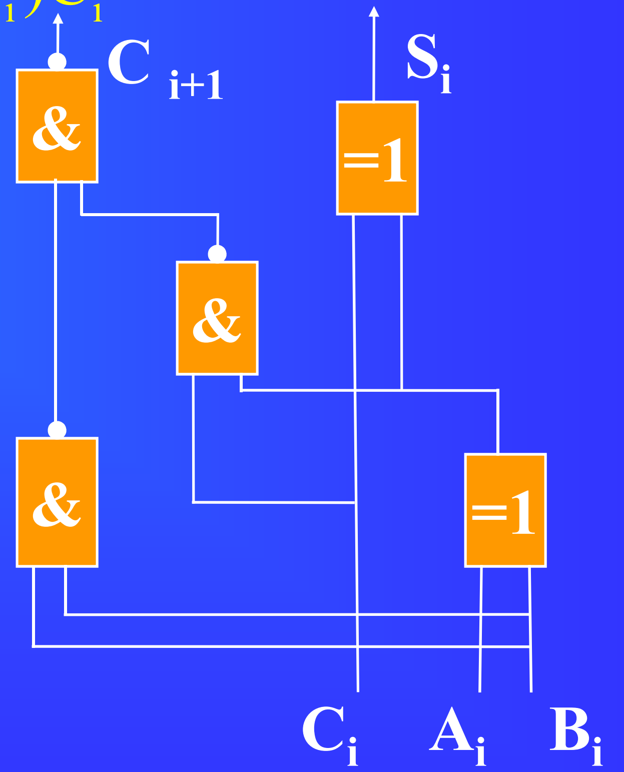
$$S_i = A_i \oplus B_i \oplus C_i = A_iB_iC_i + A_i\overline{B_i}\,\overline{C_i} + \overline{A_i}B_i\overline{C_i} + \overline{A_i}\,\overline{B_i}C_i$$

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

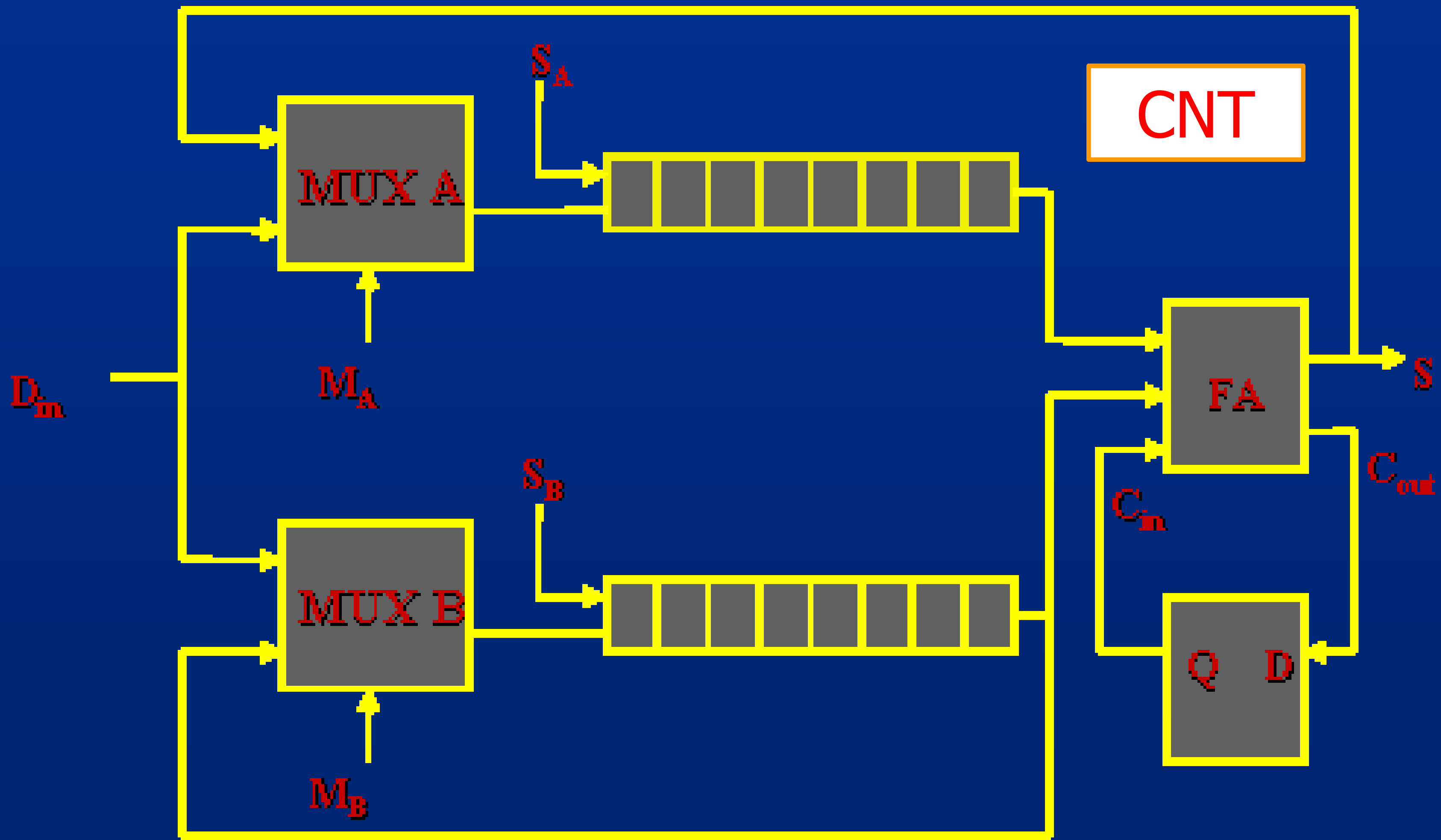
$$= A_iB_i + (A_i + B_i)C_i = \overline{\overline{A_iB_i} \bullet \overline{(A_i \oplus B_i)C_i}}$$

➢ **延迟时间：从输入数据信号有效到输出数据信号有效的时间**

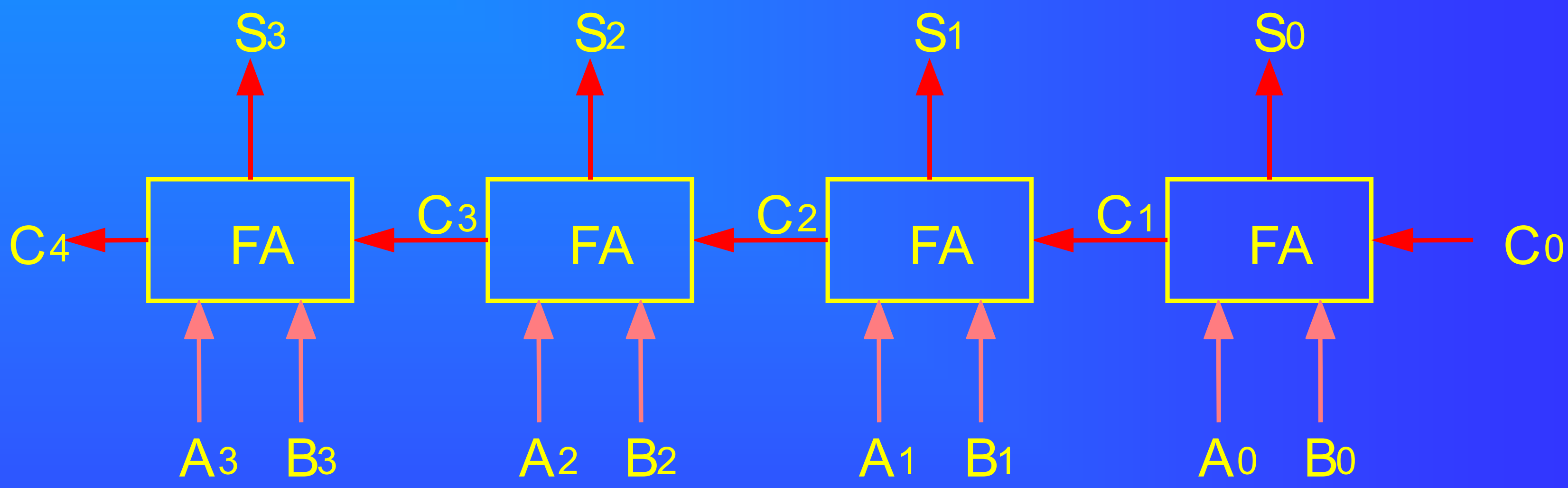
- ❑ **设**
  - ⊠ 单位门（与非门、或非门）延迟为T
  - ⊠ 每级异或门延迟3T
- ❑ **一位全加器**
  - ⊠ S_i的时间延迟：6T
  - ⊠ C_{i+1}的时间延迟：5T



串行加法器（Serial Adder）

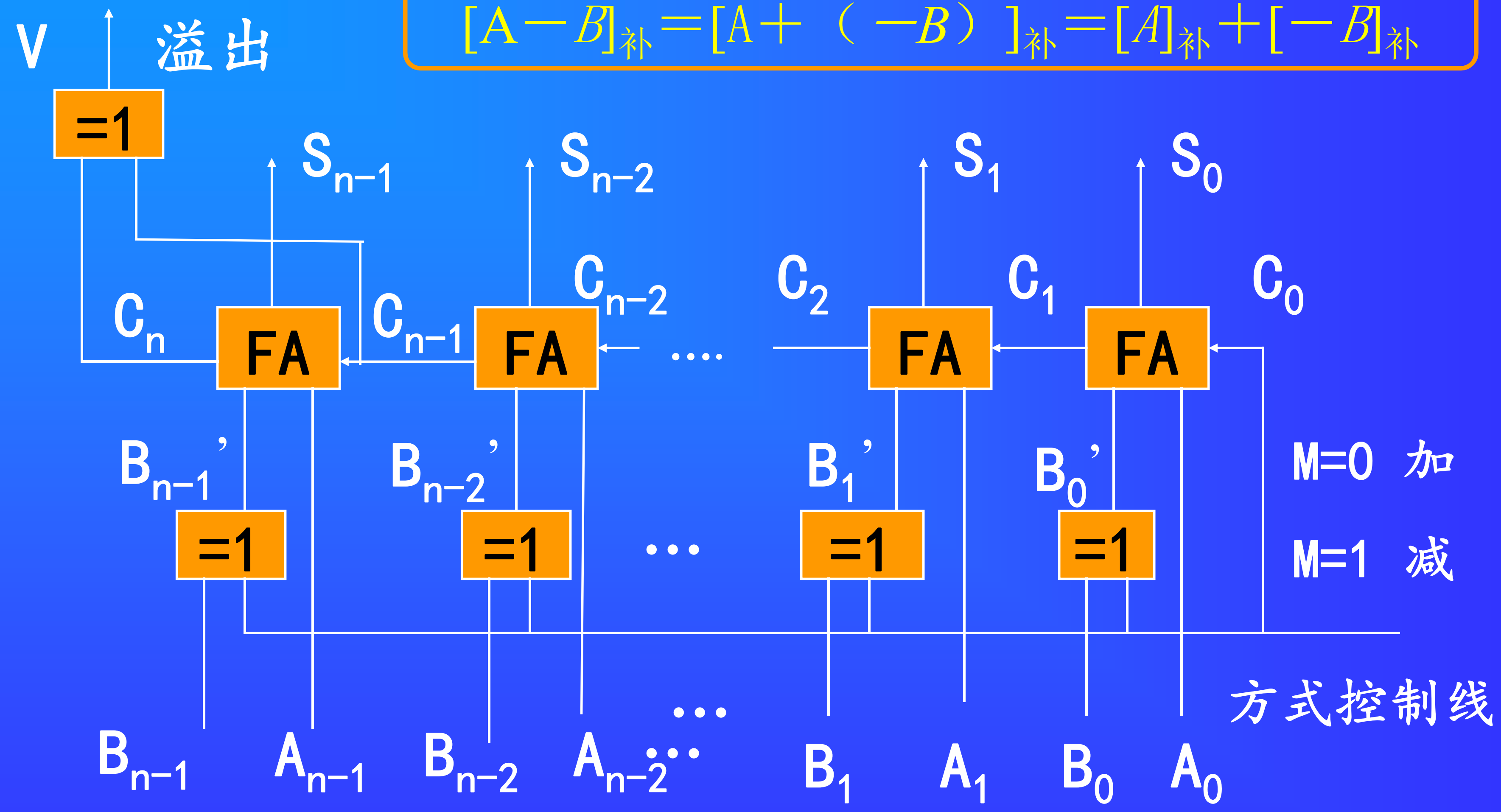


串行进位并行加法器

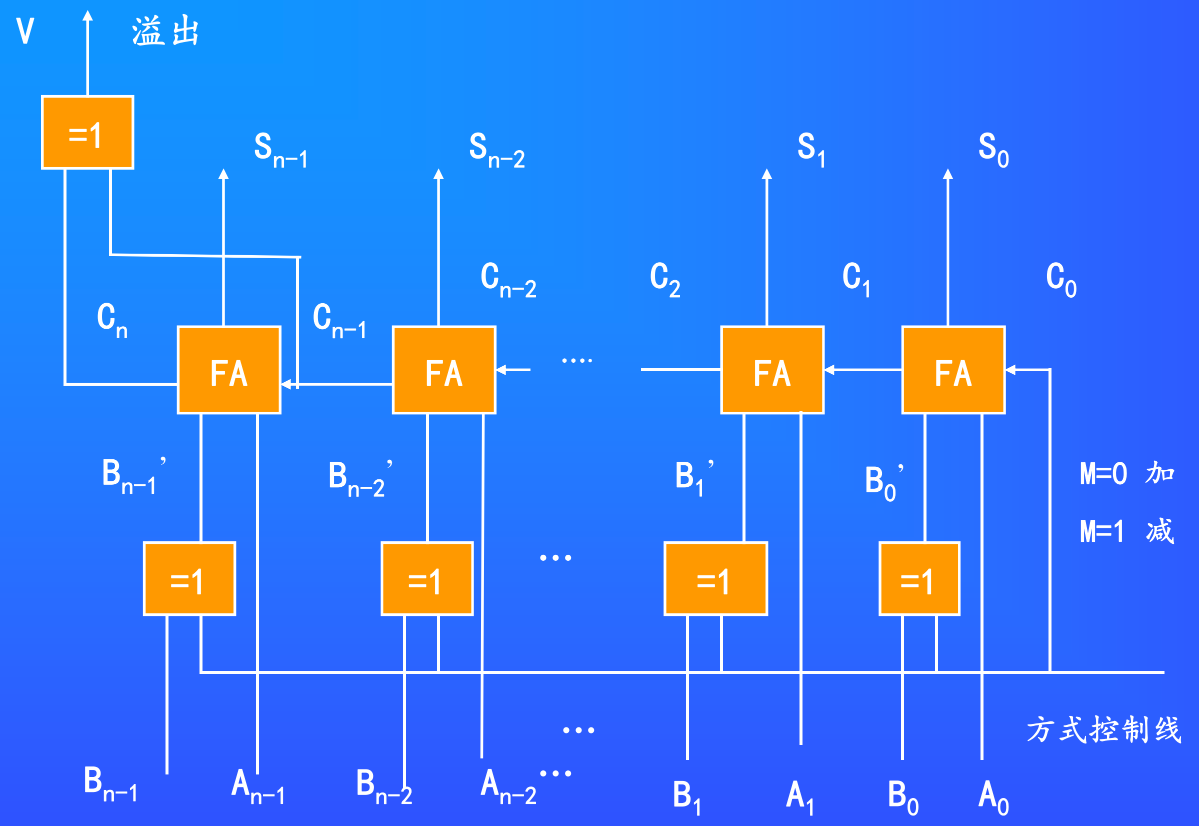


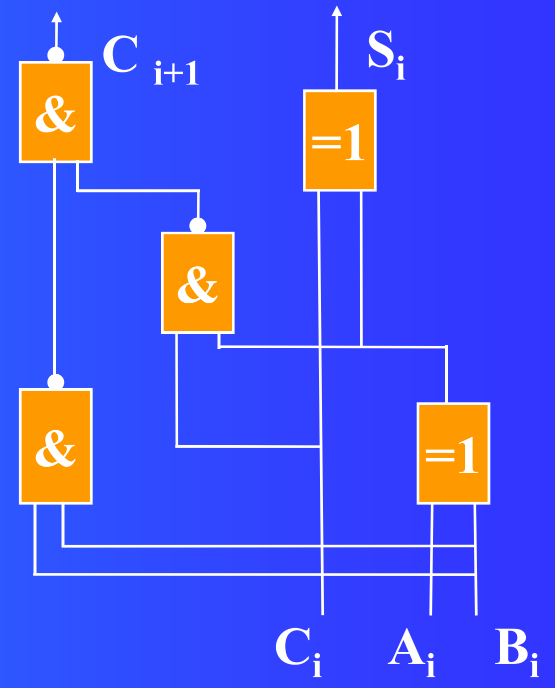
n位串行（行波）进位并行补码加减法器

$$[A-B]_{补}=[A+(-B)]_{补}=[A]_{补}+[-B]_{补}$$



串行进位并行加法器

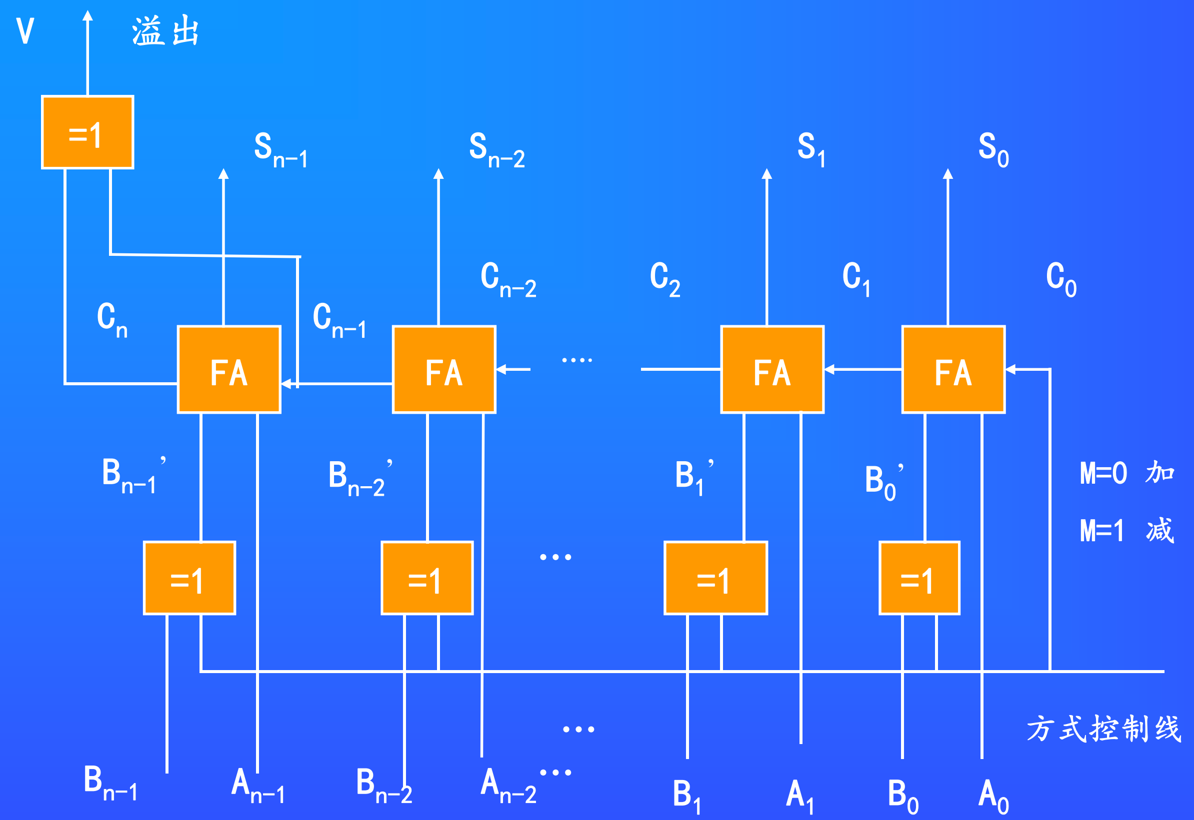


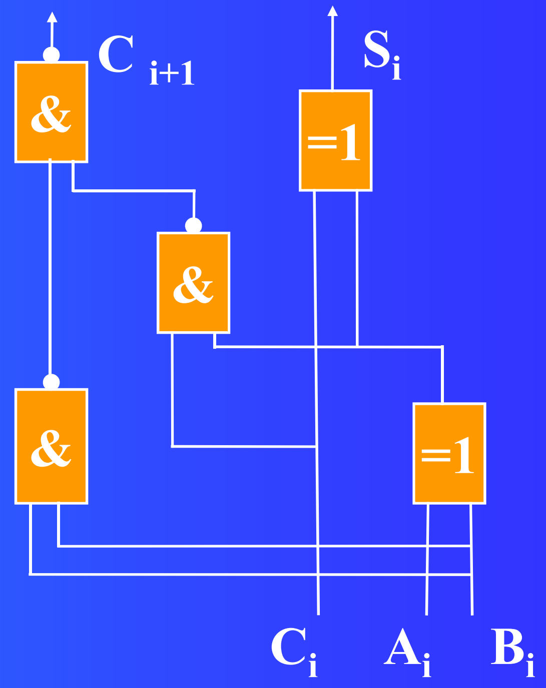


B_i与M异或：3T（n位并行）　A_i与B_i'异或：3T（n位并行）

C_i经两级与非门延迟：2T（n位串行）

串行进位并行加法器





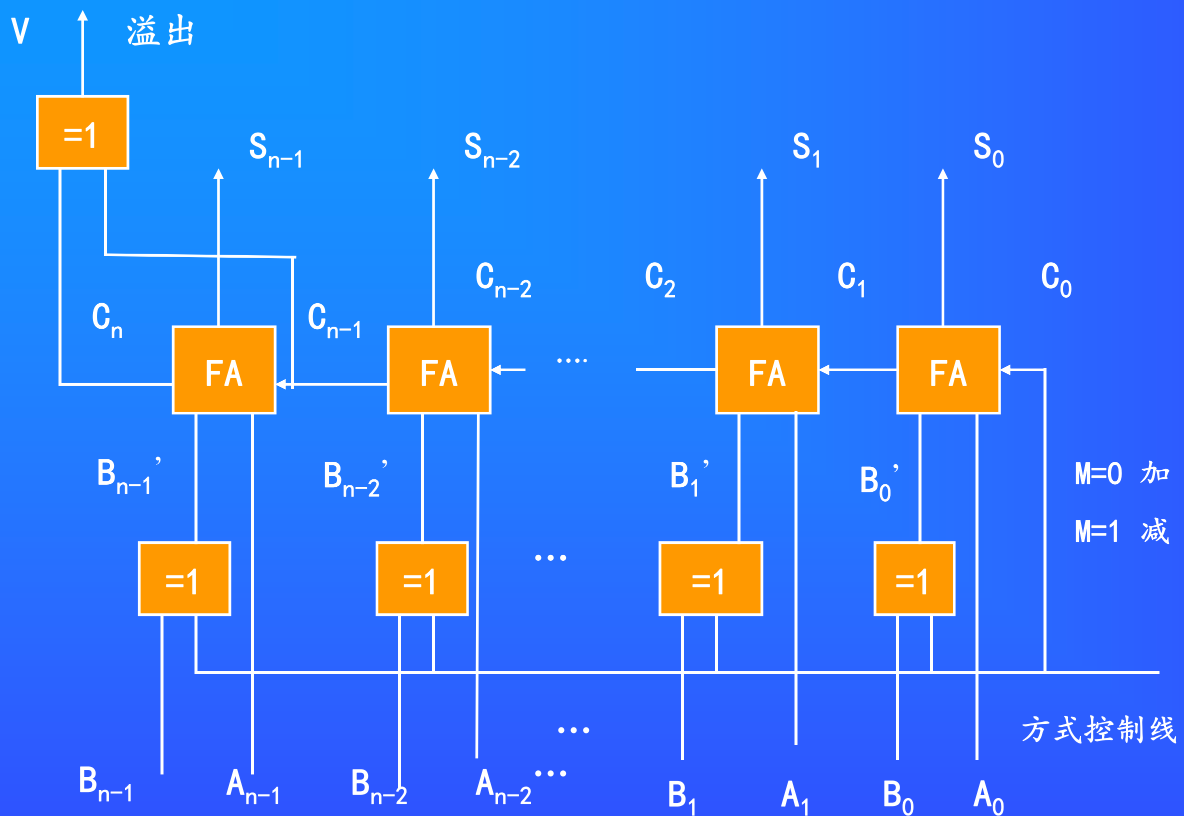
t=0: A_0~A_{n-1}, B_0~B_{n-1}, C_0=M有效

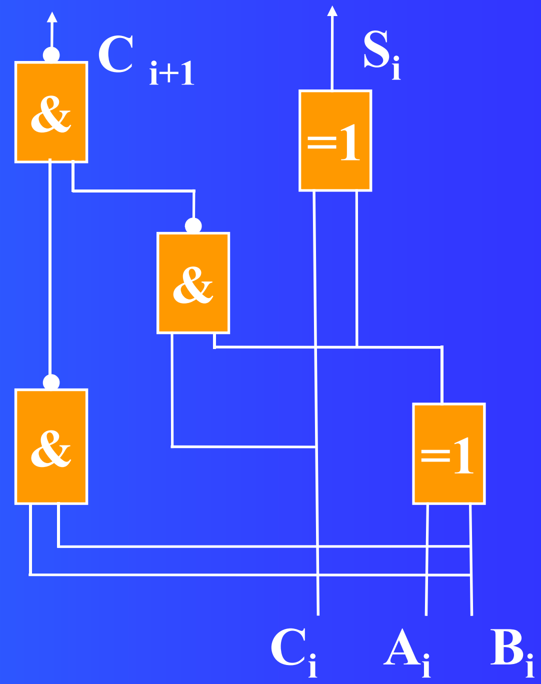
t=3T: B_i'=(B_i xor M)有效

B_i与M异或：3T（n位并行）

A_i与B_i'异或：3T（n位并行）

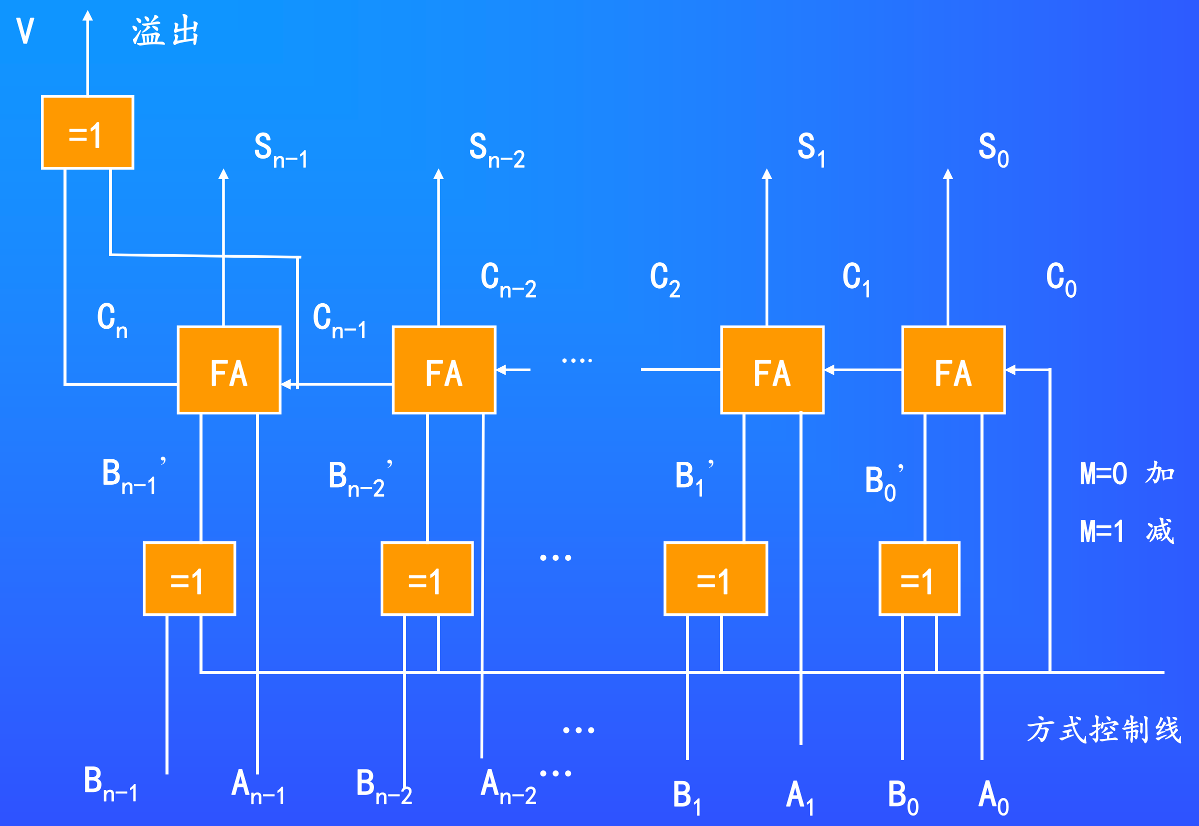
串行进位并行加法器

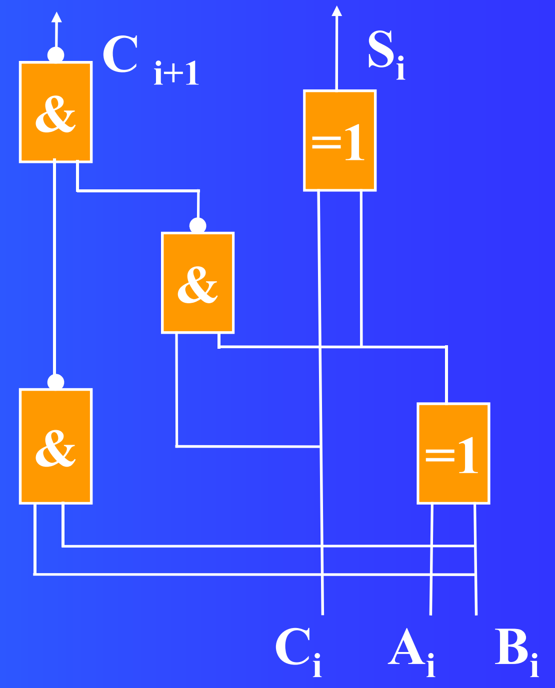




- $t=0$: $A_0 \sim A_{n-1}$, $B_0 \sim B_{n-1}$, $C_0=M$有效
- $t=3T$: $B_i'=(Bi \text{ xor } M)$有效
- $t=6T$: B_i' xor Ai有效

串行进位并行加法器





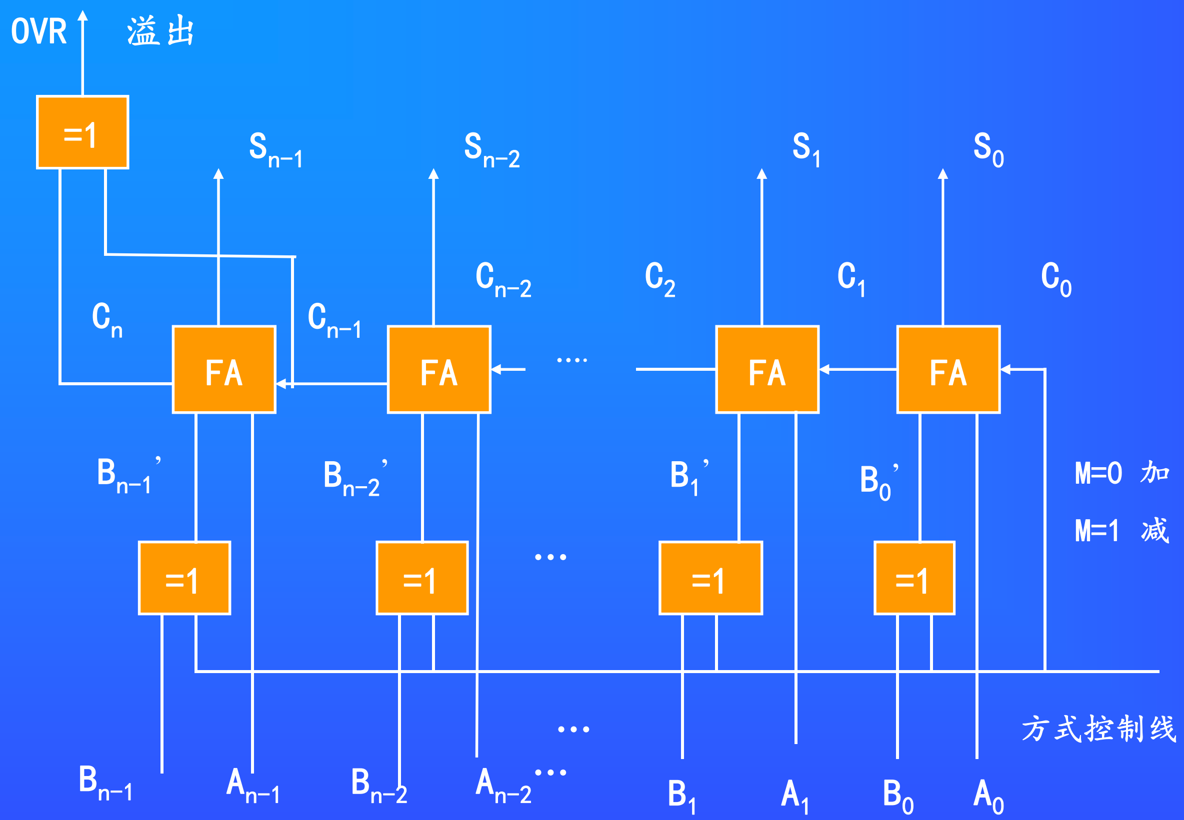
t=6T: B_i' xor A_i有效

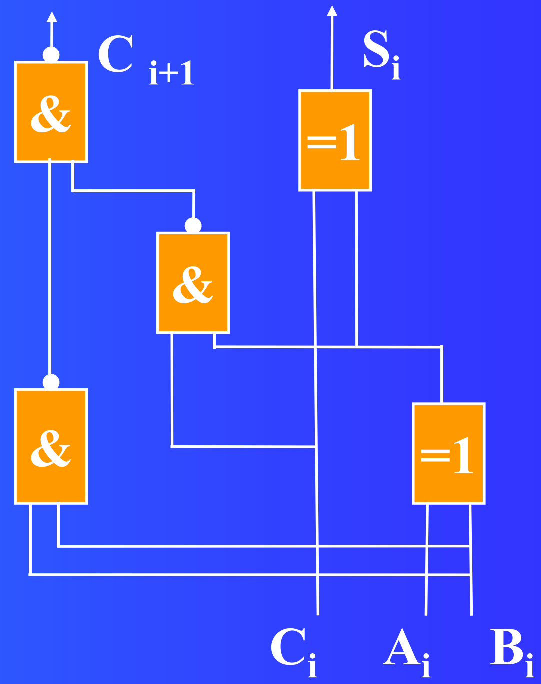
t=6T+2T: C_1有效

t=6T+4T: C_2有效

t=6T+6T: C_3有效

串行进位并行加法器





$t=6T+2(n-1)T$: C_{n-1}有效

$t=6T+2(n-1)T+3T=7T+2nT$: S_{n-1}有效

$t=6T+2nT$: C_n有效

$t=9T+2nT$: OVR有效

串行进位并行加法器的延迟

- 考虑溢出检测时的延迟：

$$t_a = n \cdot 2T + 9T = (2n+9)T$$

- 不考虑溢出检测时的延迟：

$$t_a = (n-1)2T + 9T = (2n+7)T$$

- 提高运算速度：
  - 选择高速器件
  - 改进全加器的组织
  - 改为并行进位

进位产生和传递函数

- 并行加法器的进位链的基本逻辑关系

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

- 令 $G_i = A_iB_i$ 为进位产生函数（本地进位）

  $P_i = A_i \oplus B_i$ 为进位传递函数（进位条件）

  P_iC_i 为传送进位（条件进位 ）

于是 $C_{i+1} = G_i + P_iC_i$

- **只有**$A_i = B_i = 1$**时，本位才向高位进位**
- **当**$A_i \neq B_i$**时，低一位的进位将向更高位传送**
- **传送进位和本地进位不可能同时为1**

进位产生和传递函数

$$C_{i+1}=G_i+P_iC_i=G_i+P_i(G_{i-1}+P_{i-1}C_{i-1})=G_i+P_iG_{i-1}+P_iP_{i-1}C_{i-1}$$

依此公式递推，所有进位均可从最低进位C_0**直接求得而无需等待低一位进位**

$C_1 = G_0 + P_0C_0$

$C_2 = G_1 + P_1G_0 + P_1P_0C_0$

$C_3 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$

…………

- 对任意i，均可得到只含A_k、B_k（k=0~i）以及C_0的C_{i+1}的表达式
- **先行进位逻辑的总时间延迟：5T**

$$C_1=G_0+P_0C_0=\overline{\overline{G_0}\bullet\overline{P_0C_0}}$$

4位先行进位部件

$C_1 = G_0 + P_0C_0$

$C_2 = G_1 + P_1G_0 + P_1P_0C_0$

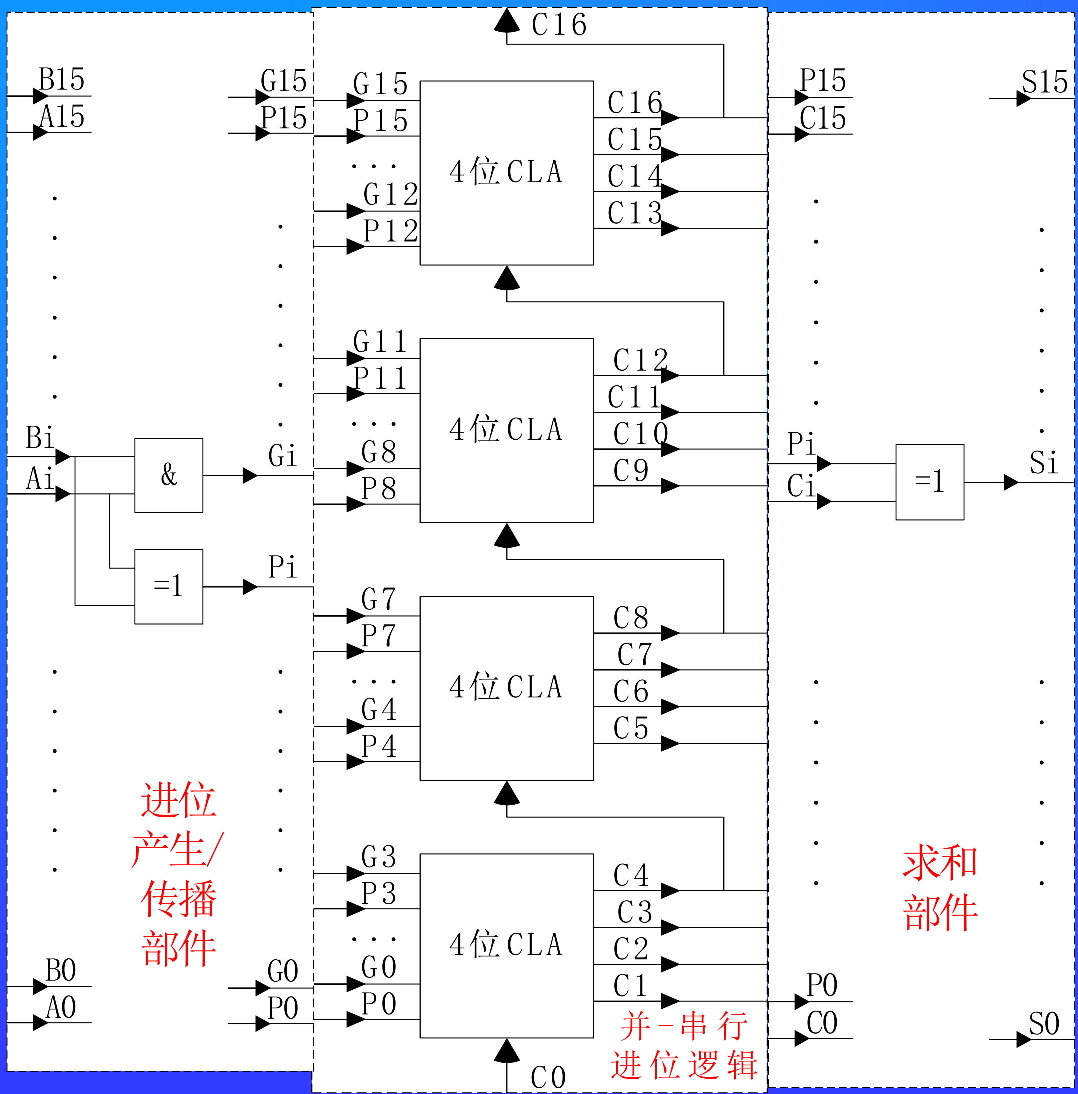
$C_3 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$

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

CLA：Carry Look Ahead

延迟时间：2T（不含求P_i的时间）

16位单级分组先行进位并行加法器



延迟时间：

- 先行进位部件：

$2T \times 4 = 8T$

- 进位产生/传播部件：3T
- 求和部件：3T
- 加法器的总延迟时间：

$t=3T+4 \times 2T+3T=14T$

- *串行进位*16位加法器的

总延迟时间：

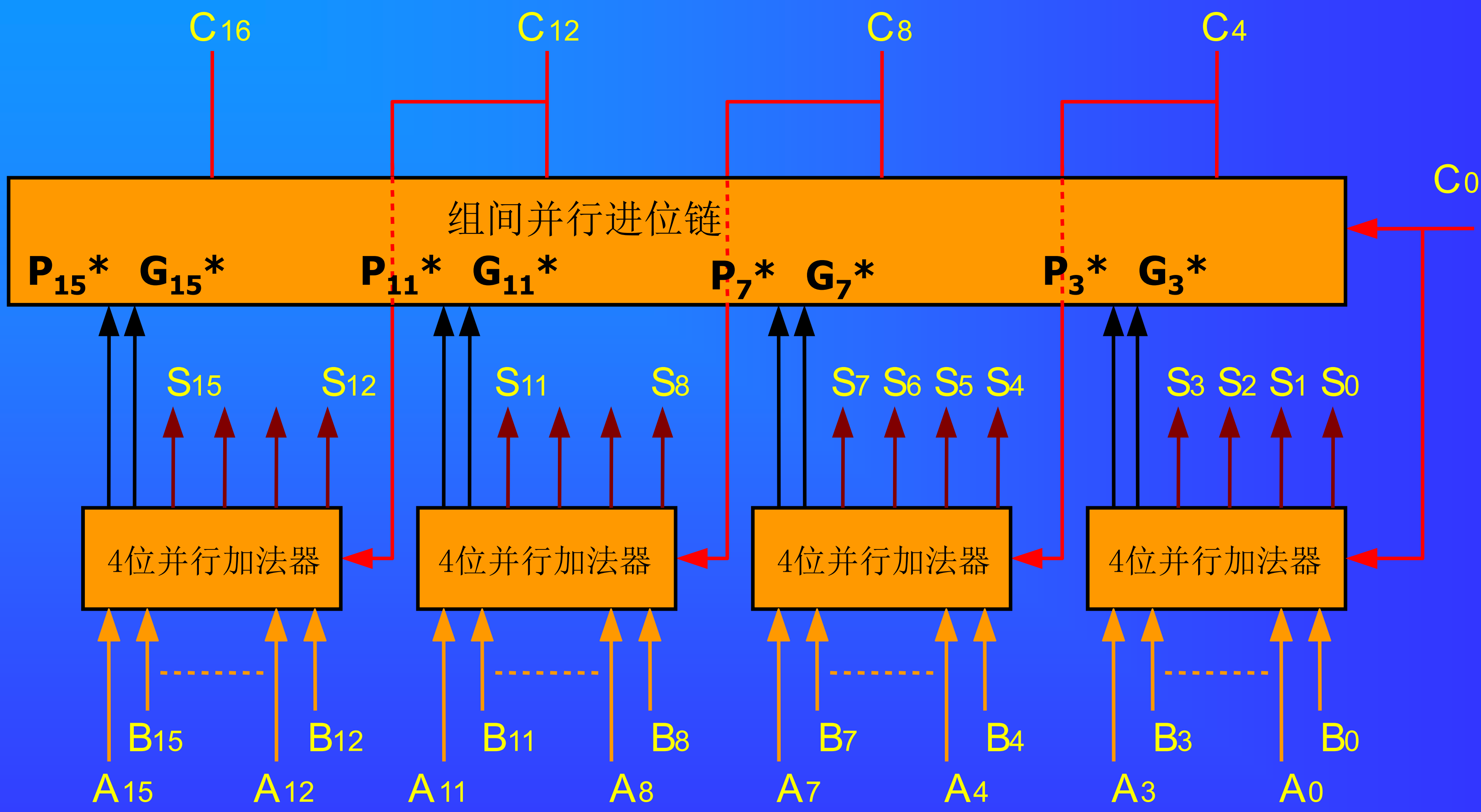
$t=9T+2nT$

$=9T+2 \times 16T=41T$

多级分组先行进位方式

- 将若干小组编成一大组
  - 小组内为先行进位
  - 同一大组内的各小组之间也采用并行进位
- 一个加法器有一个或多个大组
  - 若有多个大组，各大组间既可以采用串行进位，也可采用并行进位

两级分组16位先行进位加法器



组内并行，组间并行——并-并行进位方式

组间先行进位逻辑公式

第0小组：

$$C_4=G_3+P_3G_2+P_3P_2G_1+P_3P_2P_1G_0+P_3P_2P_1\ P_0C_0$$

$$= G_3^* + P_3^*C_0$$

其中：

$G_3^*= G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0$ ——本组进位

$P_3^* = P_3P_2P_1P_0$ ——组间进位传递函数

$P_3^*C_0$ ——组间传送进位

组间先行进位逻辑公式

$C_4 = G_3^* + P_3^* C_0$

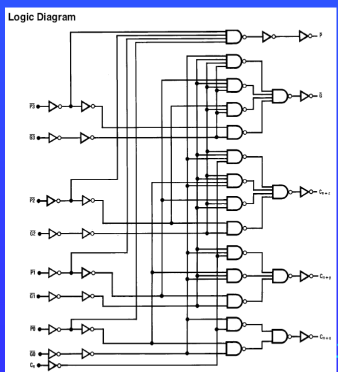
$C_8 = G_7^* + P_7^* C_4 = G_7^* + P_7^* G_3^* + P_7^* P_3^* C_0$

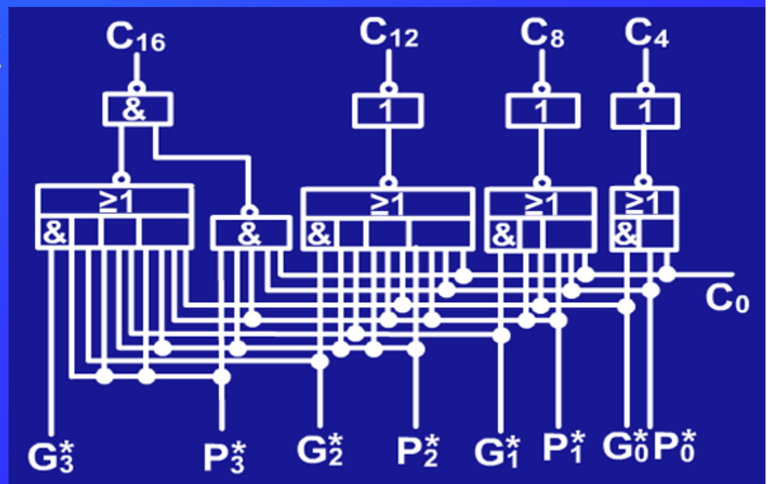
$C_{12} = G_{11}^* + P_{11}^* G_7^* + P_{11}^* P_7^* G_3^* + P_{11}^* P_7^* P_3^* C_0$

$C_{16} = G_{15}^* + P_{15}^* G_{11}^* + P_{15}^* P_{11}^* G_7^* + P_{15}^* P_{11}^* P_7^* G_3^* + P_{15}^* P_{11}^* P_7^* P_3^* C_0$

组间先行进位逻辑BCLA：Block Carry Look Ahead

实例：**74182组间先行进位逻辑**





4位组内先行进位部件CLA

$$C_1 = G_0 + P_0C_0$$

$$C_2 = G_1 + P_1G_0 + P_1P_0C_0$$

$$C_3 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$$

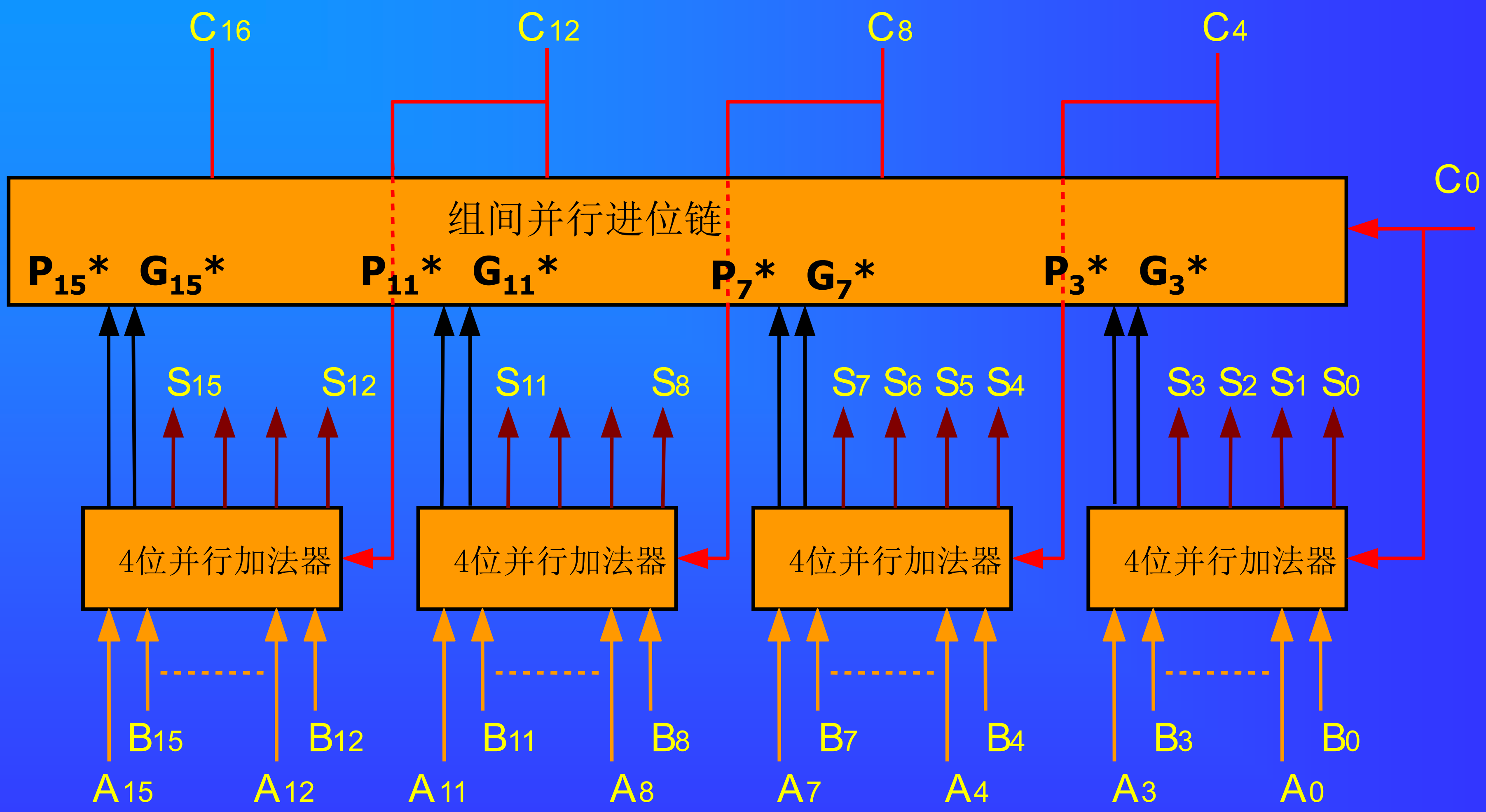
$$C_4 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0$$

修正四位CLA的输出：C_1、C_2、C_3、P_3*和G_3*（无C_4）

CLA：Carry Look Ahead

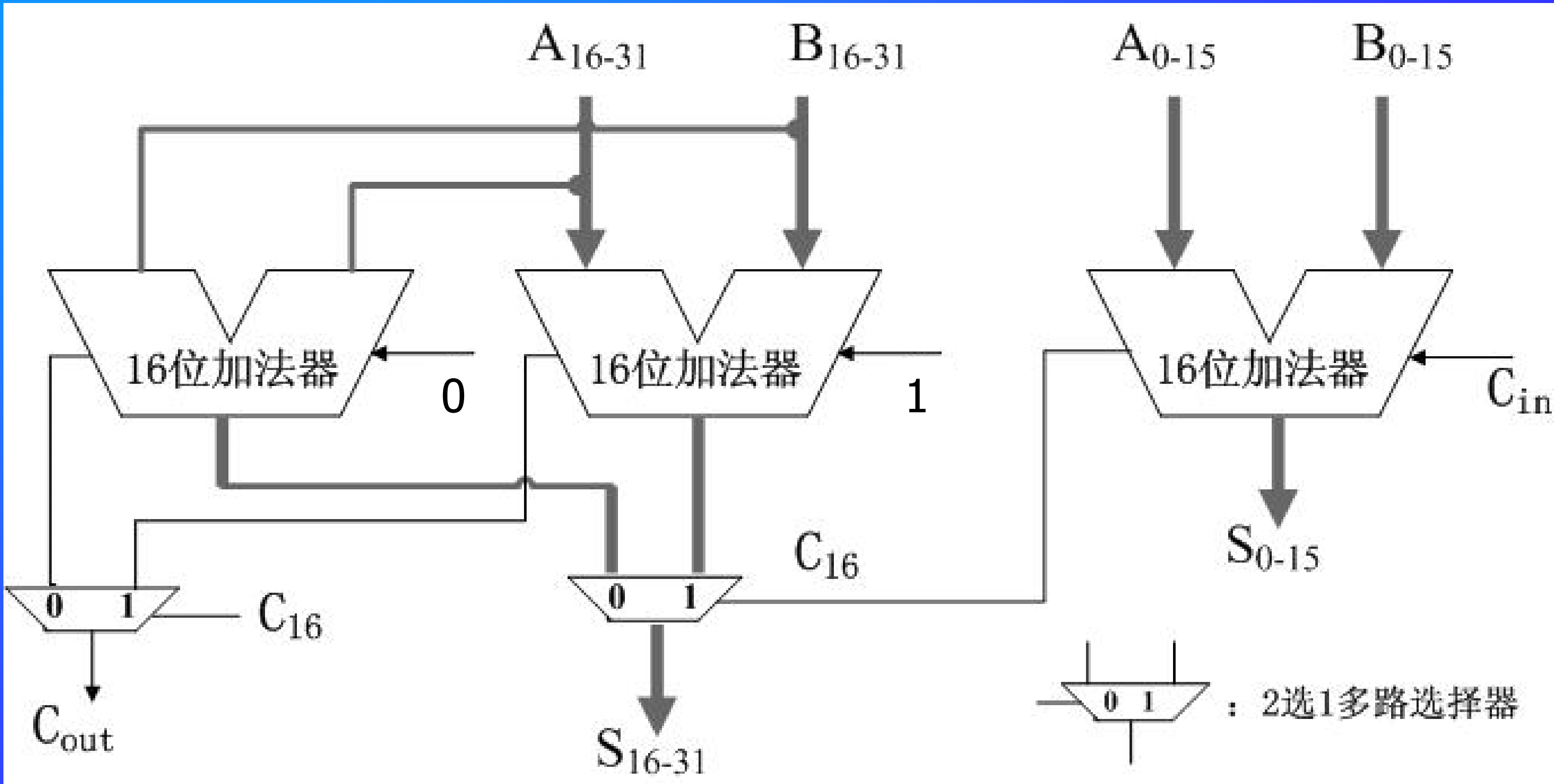
延迟时间：2T（不含求P_i的时间）

两级分组16位先行进位加法器



组内并行，组间并行——并-并行进位方式

进位选择加法器



32位进位选择加法器

北京邮電大学

【定点乘除法运算】
（第二章第二部分）

逻辑运算

- 逻辑运算仅在无符号数之间进行
- 逻辑运算与算术运算的区别：
  - 逻辑运算只在对应位之间进行
  - 各位之间无进位/借位关系
- 计算机中的逻辑运算（四种基本运算）
  - 逻辑非（逻辑反）
  - 逻辑加（逻辑或）
  - 逻辑乘（逻辑与）
  - 逻辑异（逻辑异或，按位加）

逻辑运算的应用

- **逻辑或：** $1 \vee x=1$，$0 \vee x=x$

  将一个数的某些指定比特置1，其他比特保持不变

  例：$a \vee 01000010$——将a的第1和第6位置1
- **逻辑与：** $1 \wedge x=x$，$0 \wedge x=0$

  将一个数的某些指定比特清0，其他比特保持不变

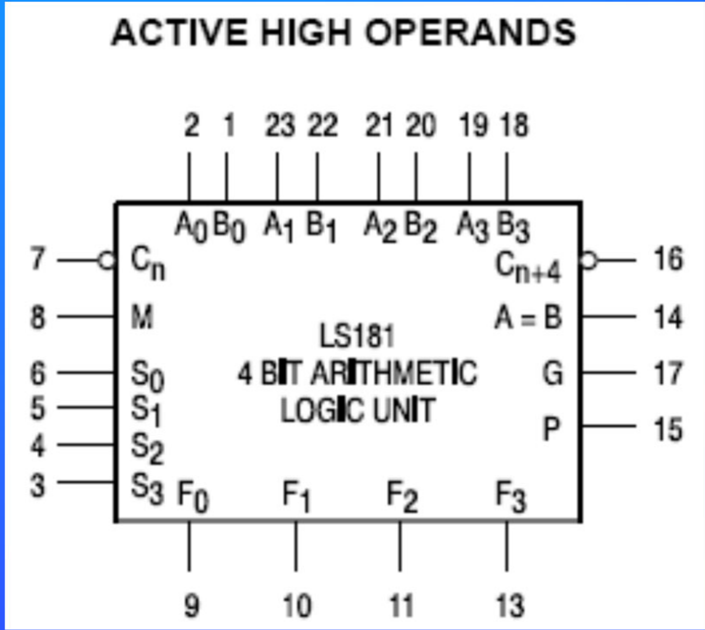
  例：$a \wedge 10111101$——将a的第1和第6位清0
- **逻辑异或：** $1 \oplus x=\bar{x}$，$0 \oplus x=x$

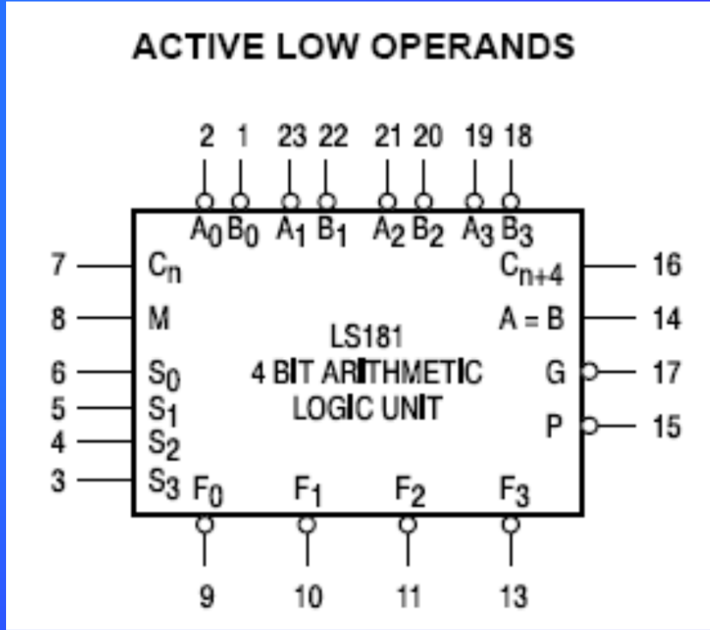
  将一个数的某些指定比特取反，其他比特保持不变

  例：$a \oplus 01000010$——将a的第1和第6位取反

多功能算术/逻辑运算单元

- ALU ：Arithmetic Logic Unit
- **4位多功能ALU——74181**





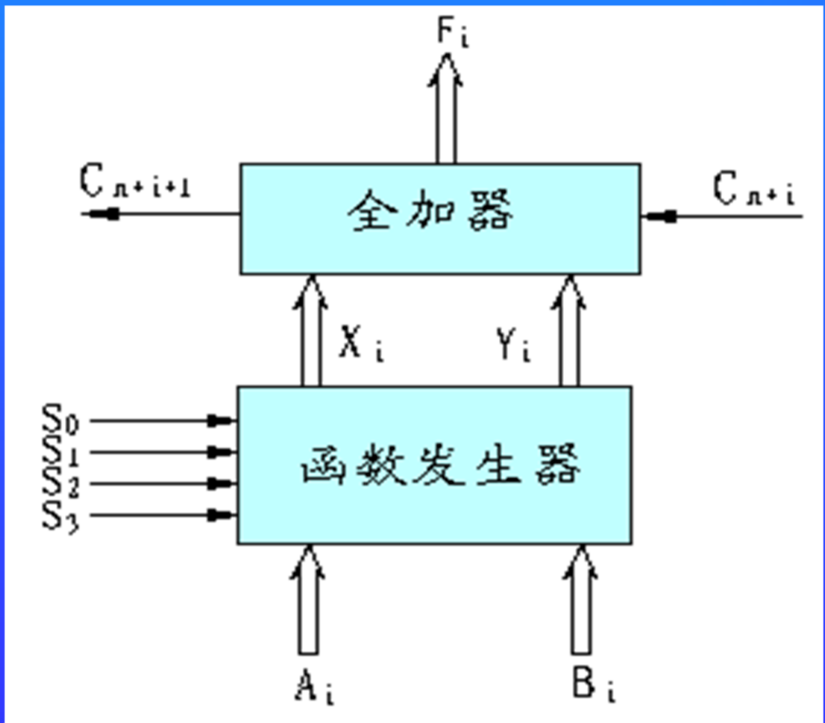
- **算术运算：4位先行进位**
- **输出：**C_{n+4}、P、G：**支持组间串行进位 或 组间先行进位**

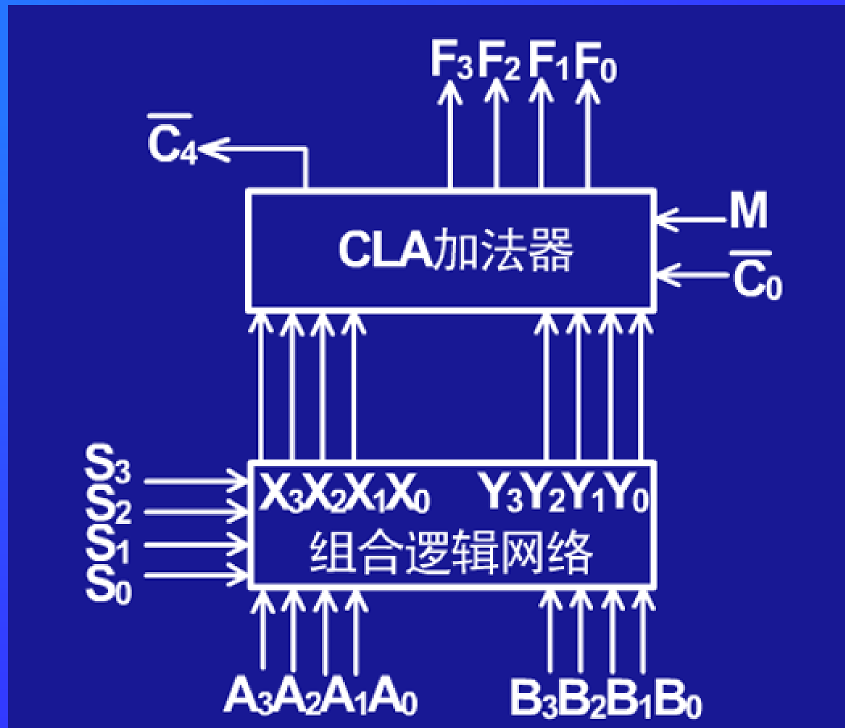
4位多功能ALU——74181

一位算术/逻辑运算单元的逻辑表达式

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

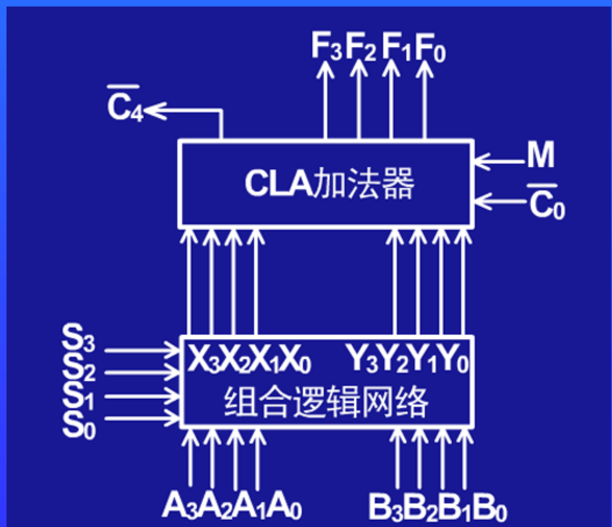
$$C_{n+i+1} = X_iY_i + Y_iC_{n+i} + C_{n+i}X_i$$

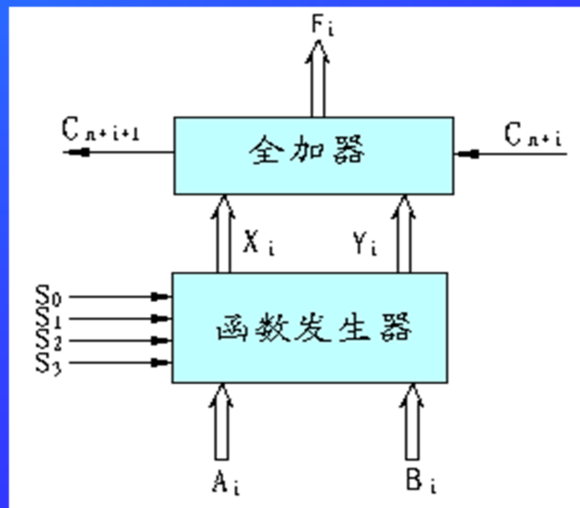




4位多功能ALU——74181

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	$\bar{A}_i$





4位多功能ALU——74181

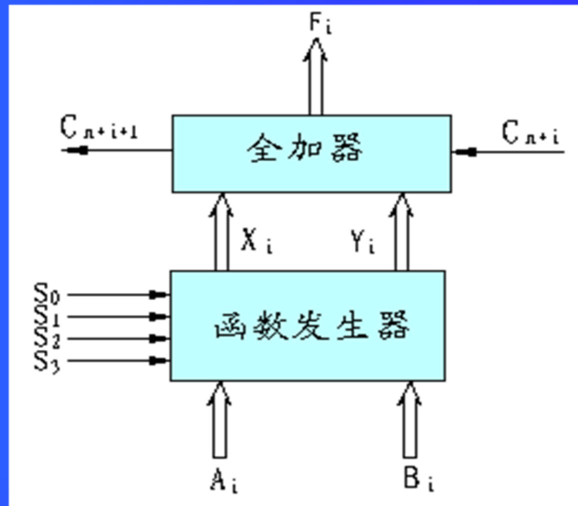
➢ **逻辑表达式**

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

$$=\overline{S_3A_iB_i+S_2A_i\bar{B}_i}$$

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

$$=\overline{A_i+S_0B_i+S_1\bar{B}_i}$$



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

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

$$=X_iY_i+（Y_i+X_i）C_{n+i}=Y_i+X_iC_{n+i}$$

◆其中：$X_iY_i=Y_i$, $Y_i+X_i=X_i$

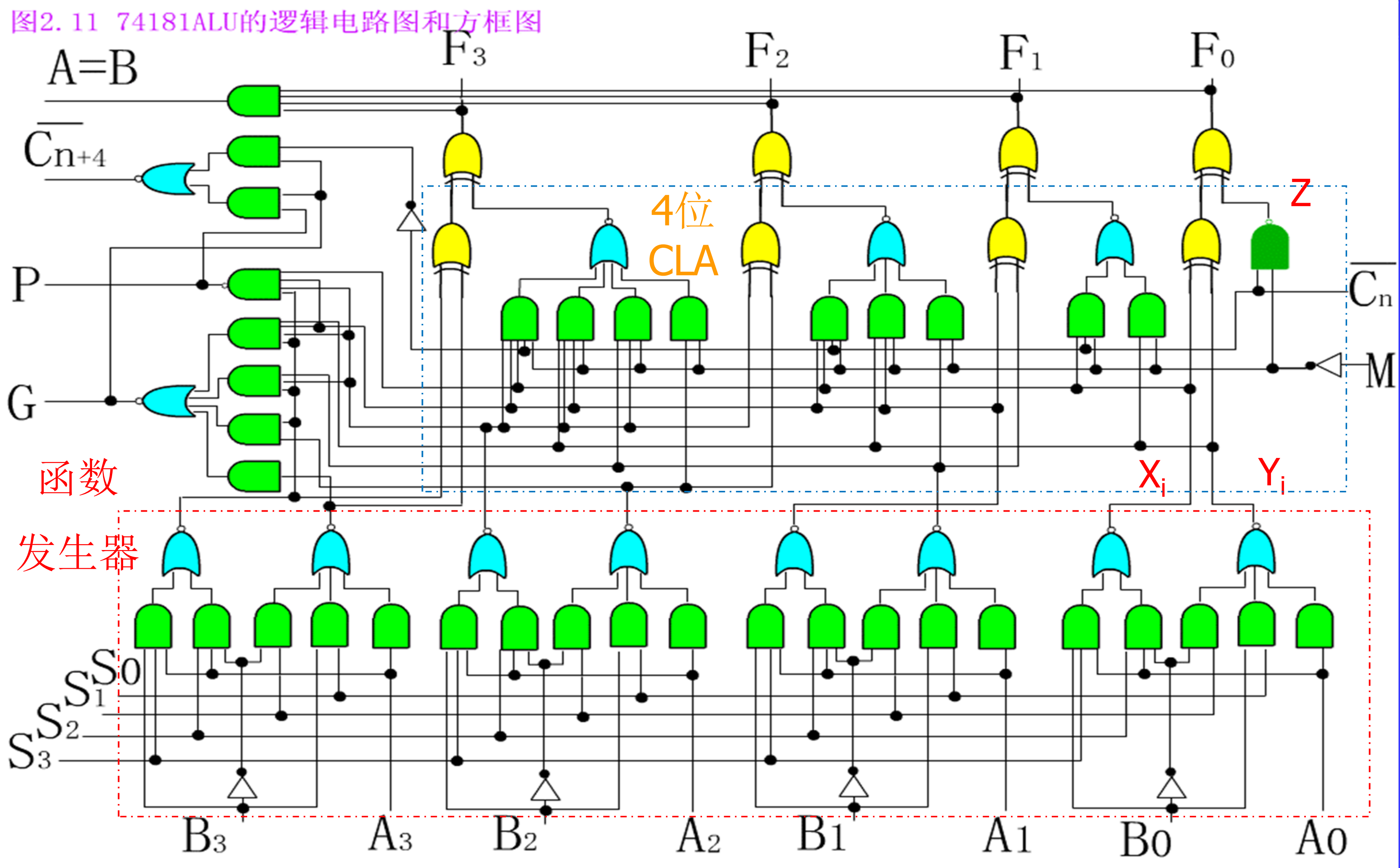
74181功能表

S_3	S_2	S_1	S_0	正 逻 辑		
				M=H 逻辑运算	M=L算术运算	
					Cn=1（无进位）	Cn=0（有进位）
L	L	L	L	$\overline{A}$	A	A+1
L	L	L	H	$\overline{A+B}$	$A+\overline{B}$	(A+B)加1
L	L	H	L	$\overline{A}\cdot B$	A+B	$(A+\overline{B})$加1
L	L	H	H	“0”	减1	“0”
L	H	L	L	$\overline{A\cdot B}$	A加$(A\cdot\overline{B})$	A加$(A\cdot\overline{B})$加1
L	H	L	H	$\overline{B}$	$(A\cdot\overline{B})$加(A+B)	$(A\cdot\overline{B})$加(A+B)加1
L	H	H	L	$A\oplus B$	A减B减1	A减B
L	H	H	H	$A\cdot\overline{B}$	$(A\cdot\overline{B})$减1	$A\cdot\overline{B}$
H	L	L	L	$\overline{A}+B$	A加$(A\cdot B)$	A加$(A\cdot B)$加1
H	L	L	H	$\overline{A\oplus B}$	A加B	A加B加1
H	L	H	L	B	$(A\cdot B)$加$(A+\overline{B})$	$(A\cdot B)$加$(A+\overline{B})$加1
H	L	H	H	$A\cdot B$	$(A\cdot B)$减1	$(A\cdot B)$
H	H	L	L	“1”	A加A	A加A加1
H	H	L	H	$A+\overline{B}$	A加(A+B)	A加(A+B)加1
H	H	H	L	A+B	A加$(A+\overline{B})$	A加$(A+\overline{B})$加1
H	H	H	H	A	A减1	A

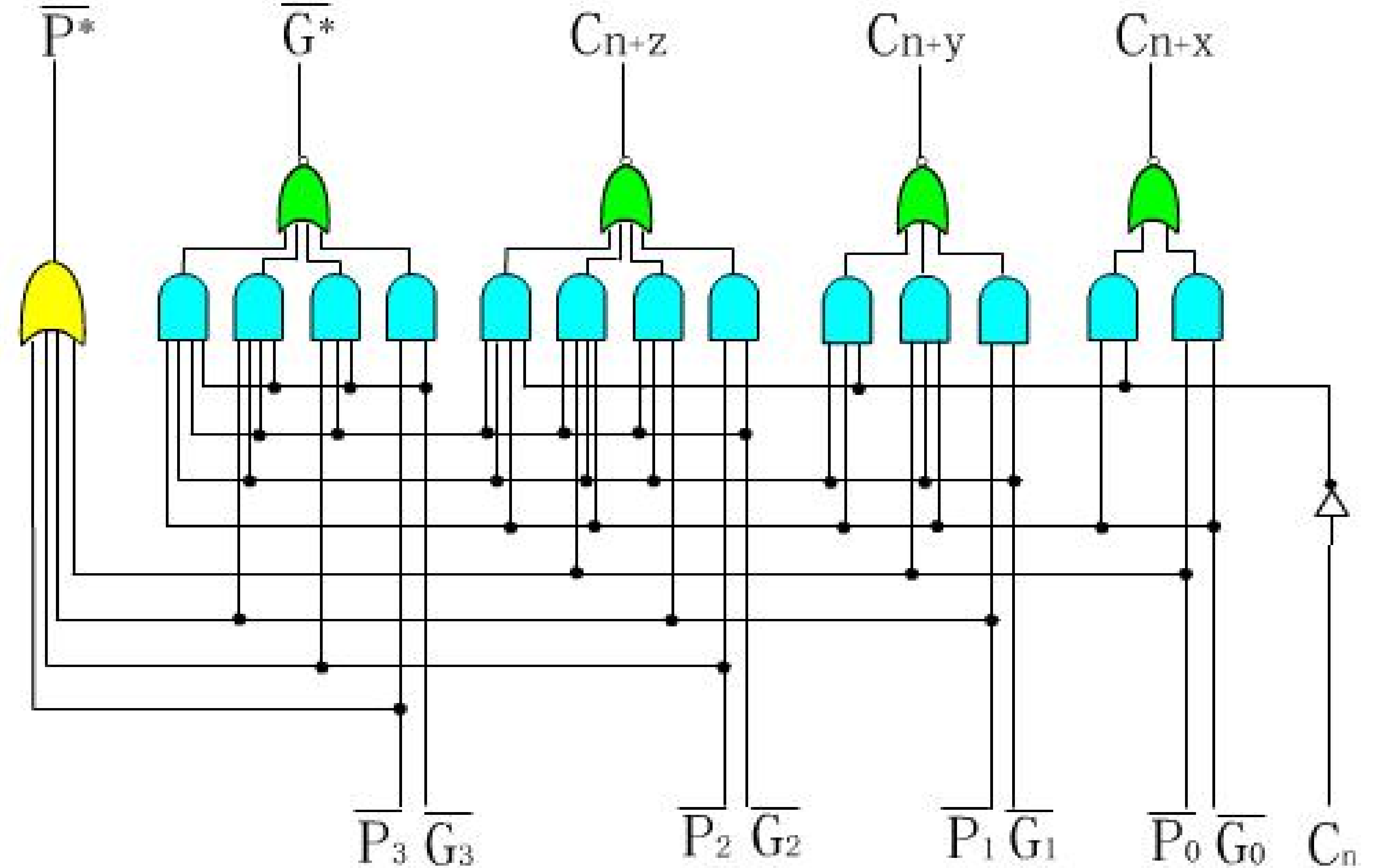
加：算术加，运算需考虑进位；+：逻辑加

74181的逻辑电路图

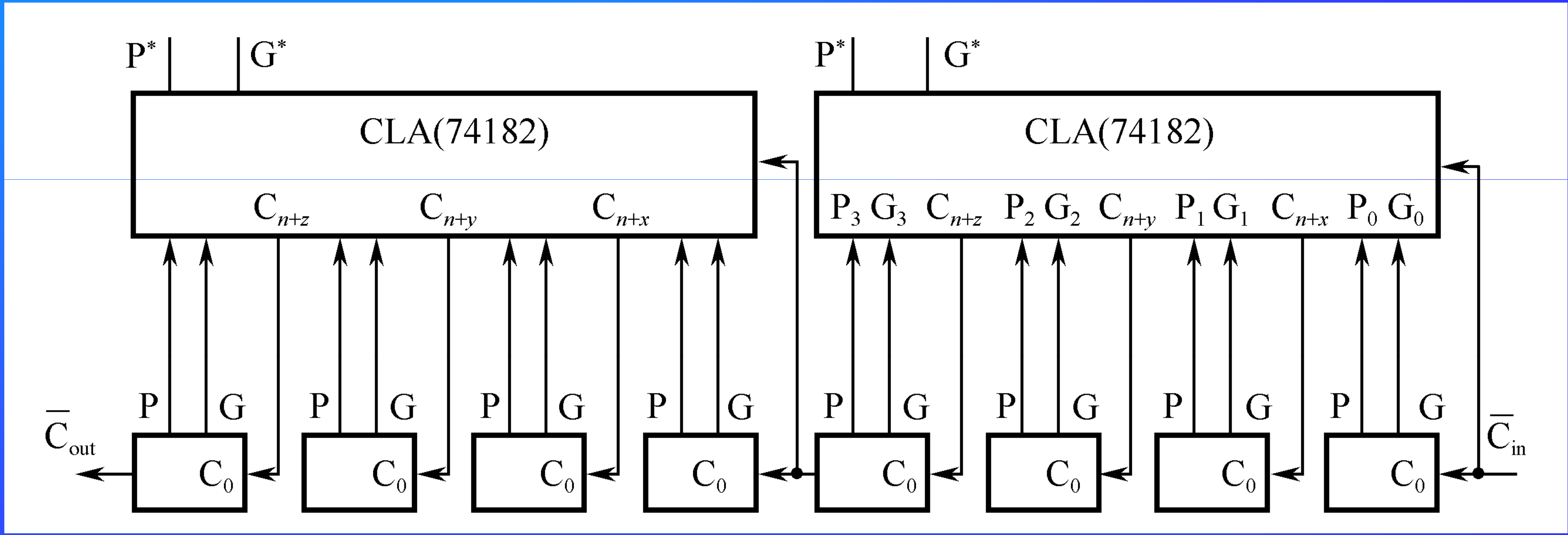
M=1：逻辑运算
M=0：算术运算



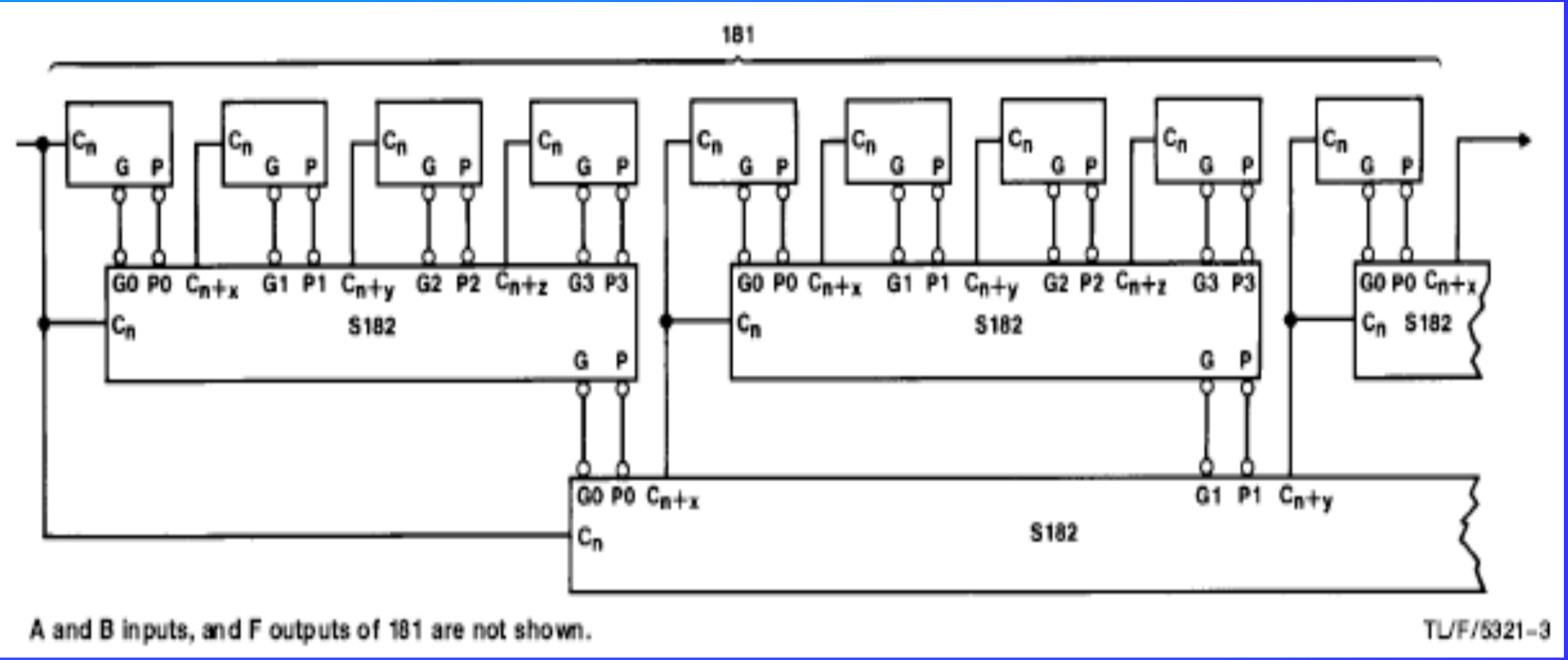
74182 CLA的逻辑电路图



三级分组“并-并-串”先行进位并行ALU

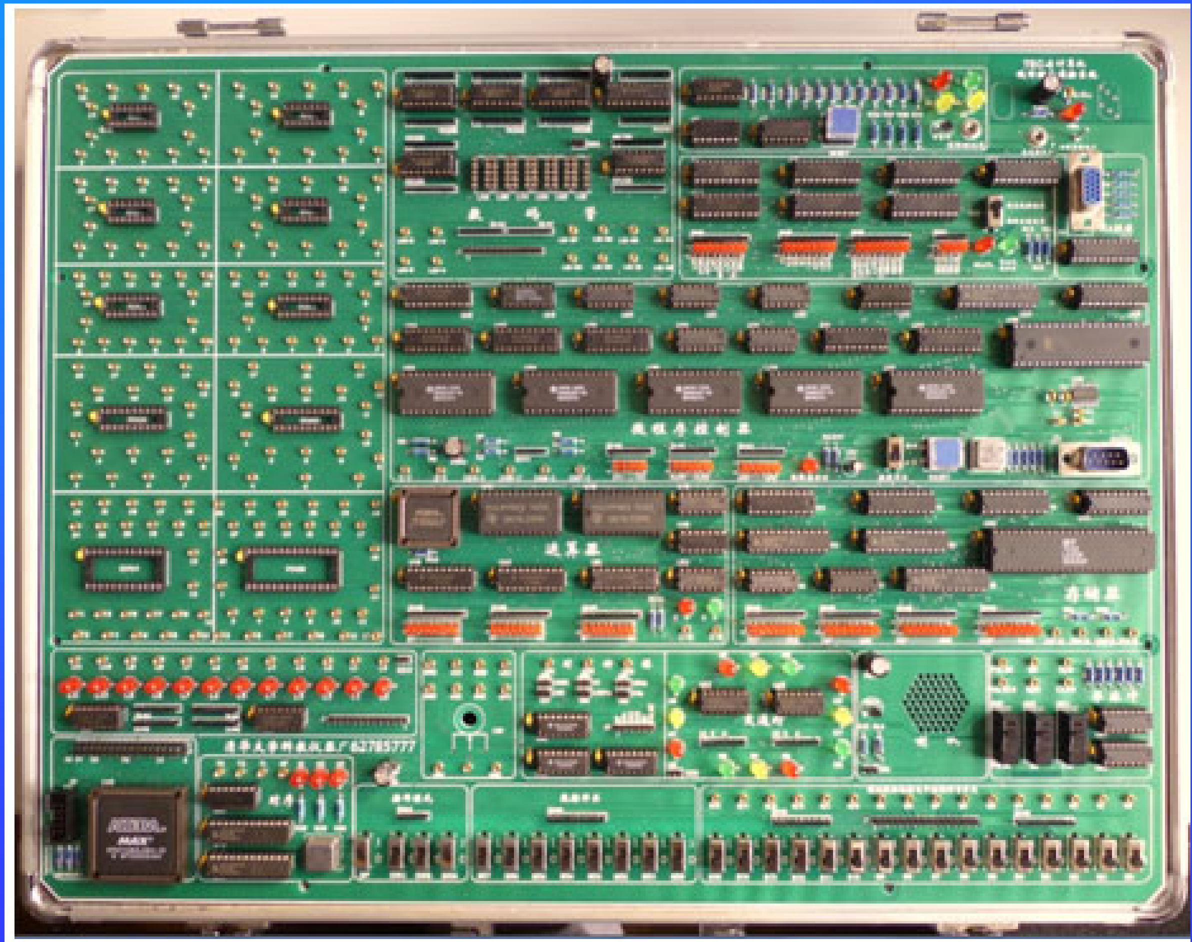


64-BIT ALU, FULL-CARRY LOOK AHEAD IN THREE LEVELS



实验一 运算器组成

TEC-8 计算机硬件综合实验系统



主楼

723室

电话：

62283297

张杰老师

第二章小结（第一部分）

- 数据的表示方法及其机器存储：
  - 便于存储、便于运算
- 算术运算和逻辑运算的运算方法
- 运算器的组成（实现）
  - 性能与成本的平衡

本章重点（第一部分）

- 数值数据的表示方法：
  - 机器数的表示范围、精度和特点
  - 定点数的机器码各种码制之间的转换
- 定点算术运算的实现
  - 溢出的概念，溢出的检测方法
  - 加减法的统一
  - 先行进位
  - 运算器的硬件复杂度和性能
  - 定点运算器的组成和结构

作业

1. 一个C语言程序运行在一台32位机器上。程序中定义了三个变量x、y和z。其中，x和z为int型，y为short型。已知x＝127，y＝－9。执行赋值语句z=x+y后，x、y和z的值分别是多少？（请用16进制数表示结果）
2. 利用74181和74182器件设计如下三种方案的64 位ALU：（1）行波进位CLA；（2）两级行波进位CLA；（3）三级CLA。试求三种方案的集成电路片数，并给出方框图。

计算机组成原理

Principle of Computer Organization

第二章 运算方法与运算器

第一部分

结束