

计算机组成与体系结构

数据表示、计算以及指令系统

计算机科学与技术

- 数据的表示与数据结构
- 传统机器数据表示：
 - 定点数：原码、反码、补码、移码。
 - 浮点数：尾数、基数和阶码、IEEE754 标准。
 - 字符：常用ASCII码。
 - 汉字：输入码，汉字内码，汉字字模码。
- 高级数据表示：自定义的数据表示，向量数据表示、堆栈数据表示。
- 数据校验原理：奇偶校验码

第二部分主要内容



- 数据的表示
- 数据的计算
- 指令系统

掌握数值数据在计算机中实现算术运算和逻辑运算的方法，以及运算部件的基本结构和工作原理。

重点关注：

- 1、串行、并行加法器、溢出判断与检测、移位操作。**
- 2、乘法电路基本结构：BOOTH乘法器（1位）**
- 3、除法电路基本结构：加减交替除法器（补码）**
- 4、逻辑运算：与、或、非、异或运算**
- 5、运算器基本结构**

实验一

n位加法

进位产生
用 G_i 表示

进位传递函数
用 P_i 表示

- 基本加法器

1位全加器的逻辑表达式为

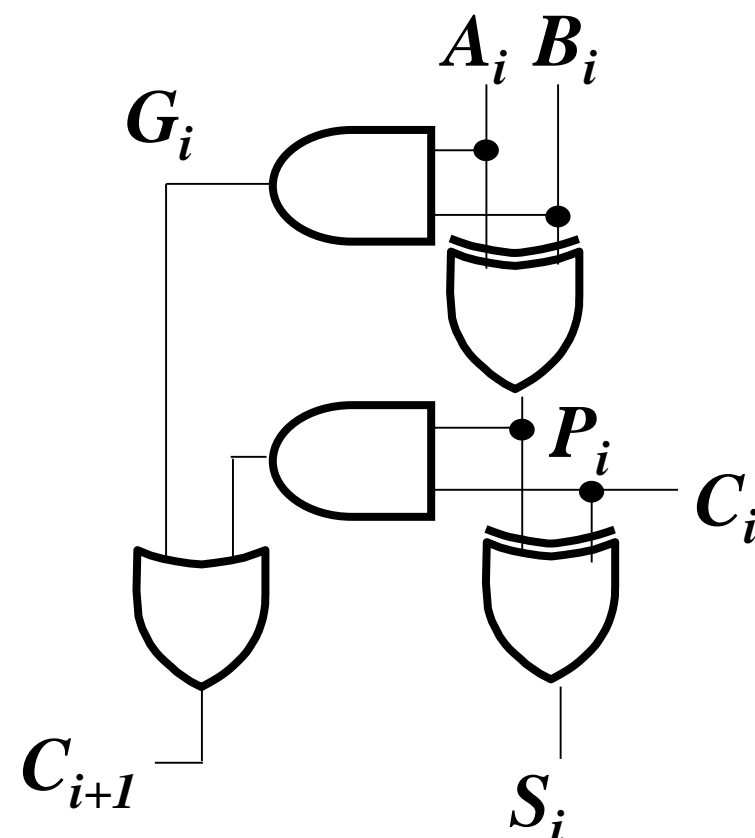
$$S_i = A_i \oplus B_i \oplus C_{i-1}$$

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

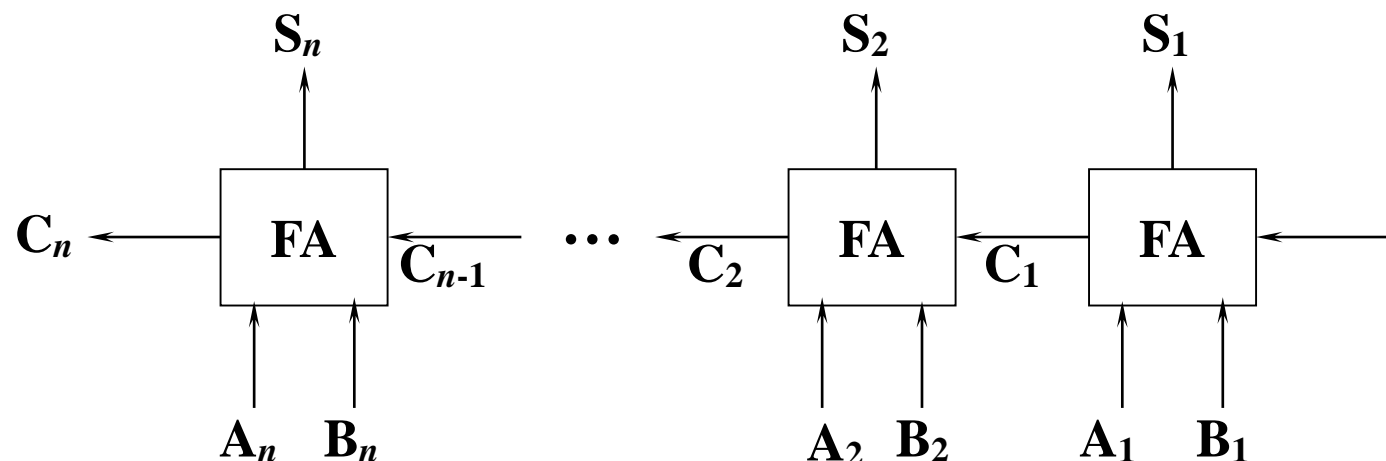
串行加法器：只有一个全加器，数据逐位串行送入加法器进行运算。如果操作数长 n 位，加法就要分 n 次进行，每次只能产生一位和。

并行加法器：并行加法器由多个全加器组成，其位数的多少取决于机器的字长，数据的各位同时运算。

Carry **G**enerating Function
Carry **P**ropagating Function



n位行波进位加法器



思考：
进位延迟与进位的关系？
能更快吗？如何改进？

每形成一级进位的延迟时间为 $2t_y$ 。在字长为 n 位的情况下，若不考虑 G_i 、 P_i 的形成时间，从 $C_0 \rightarrow C_n$ 的最长延迟时间为 $2nty$ 。

提高并行加法器速度的关键是**尽量加快进位产生和传递的速度**。

$$C_1 = G_1 + P_1 C_0$$

$$C_2 = G_2 + P_2 C_1 = G_2 + P_2 G_1 + P_2 P_1 C_0$$

$$C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 C_0$$

$$C_4 = G_4 + P_4 G_3 + P_4 P_3 G_2 + P_4 P_3 P_2 G_1 + P_4 P_3 P_2 P_1 C_0$$

.....

每个进位都不需要等待低位，直接计算可以得到，这种方法实现的加法器就被称为**超前进位加法器**（Carry_lookahead Adder, CLA）。

若不考虑 G_i 、 P_i 的形成时间，从 $C_0 \rightarrow C_n$ 的最长延迟时间仅为 $2t_y$ 。

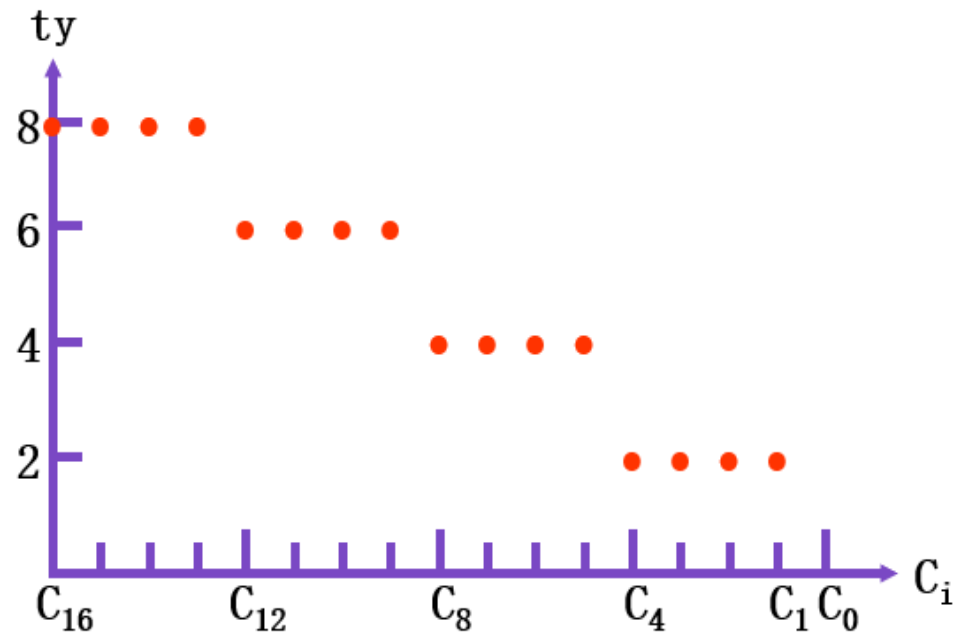
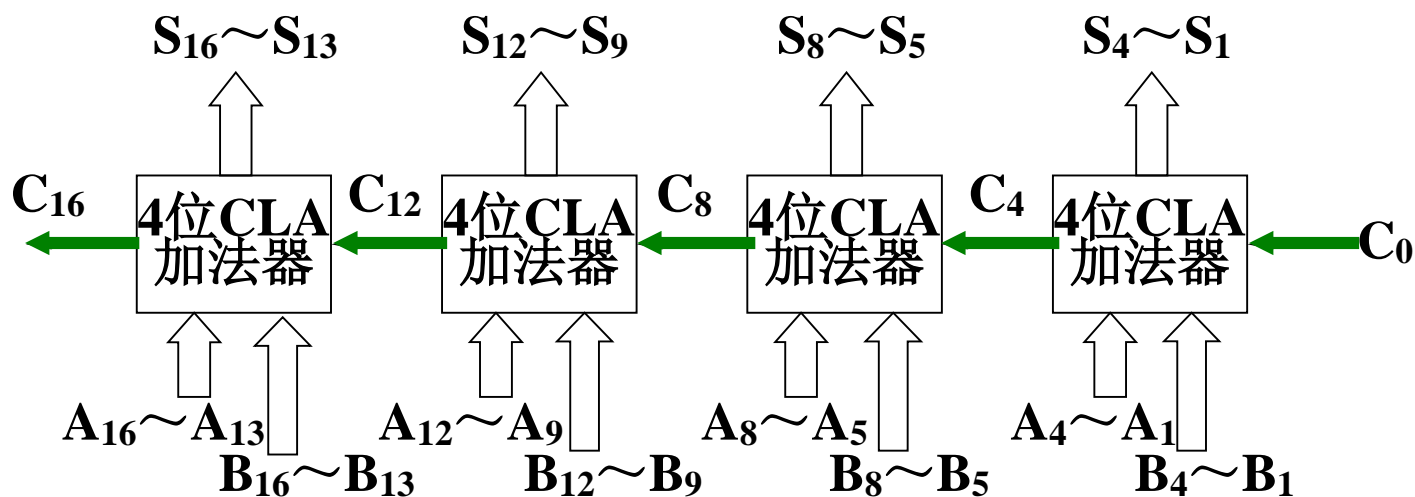
但随着加法器位数的增加， C_i 的逻辑表达式会变得越来越长，实现十分复杂，**如何解决？**

分组控制规模

(1)单级先行进位方式



这种进位方式又称为**组内并行、组间串行**方式。以16位加法器为例，可分为四组，每组四位。第1小组组内的进位逻辑函数 C_1 、 C_2 、 C_3 、 C_4 的表达式与前述相同， $C_1 \sim C_4$ 信号是同时产生的，从 C_0 出现到产生 $C_1 \sim C_4$ 的延迟时间是 $2t_y$ 。



(2)多级先行进位方式



又称**组内并行、组间并行**进位方式。

字长为16位的两级先行进位加法器，第一小组的最高位进位 C_4 ：

$$C_4 = G_4 + P_4 G_3 + P_4 P_3 G_2 + P_4 P_3 P_2 G_1 + P_4 P_3 P_2 P_1 C_0 = G_1^* + P_1^* C_0$$

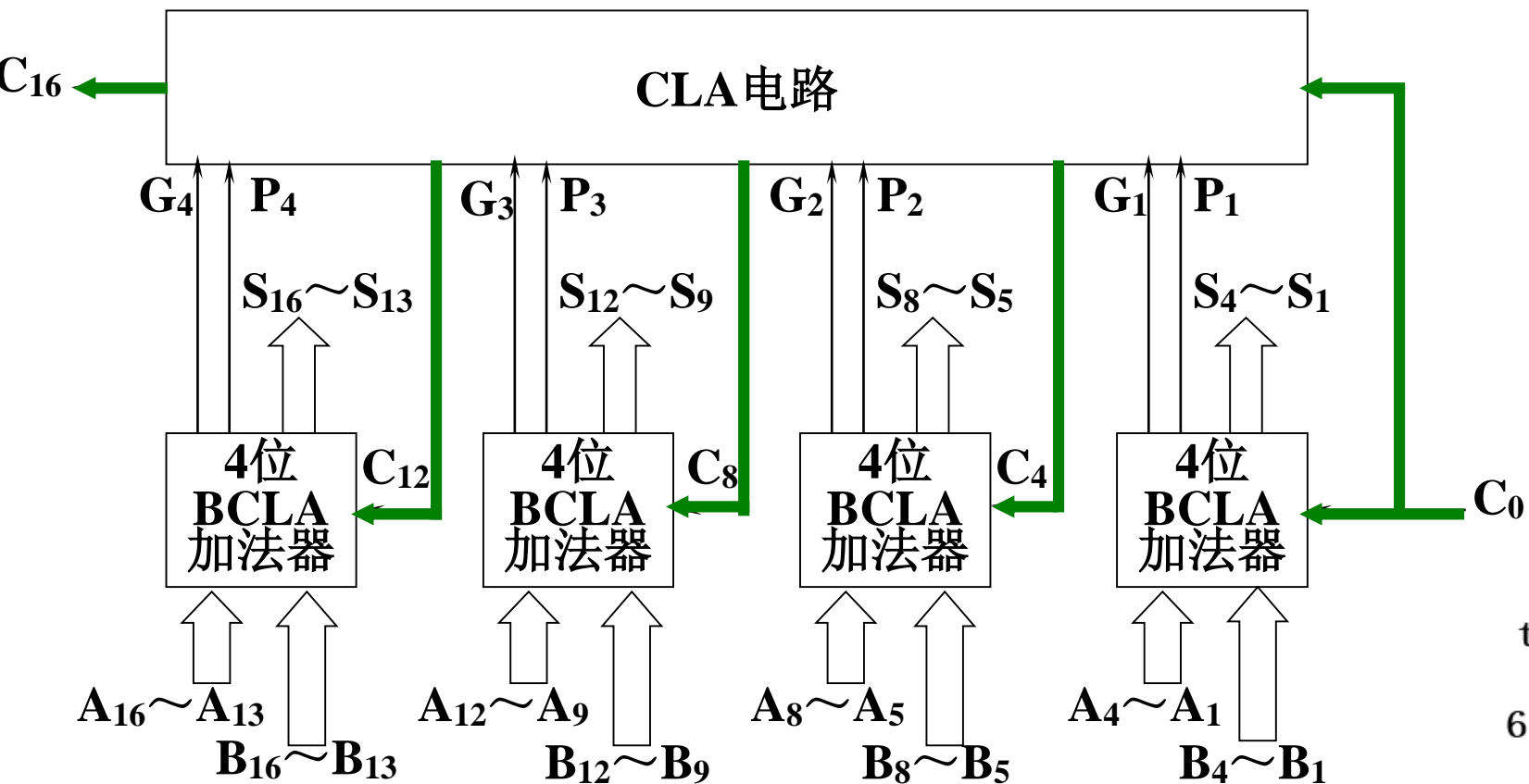
依次类推：

$$C_8 = G_2^* + P_2^* G_1^* + P_2^* P_1^* C_0$$

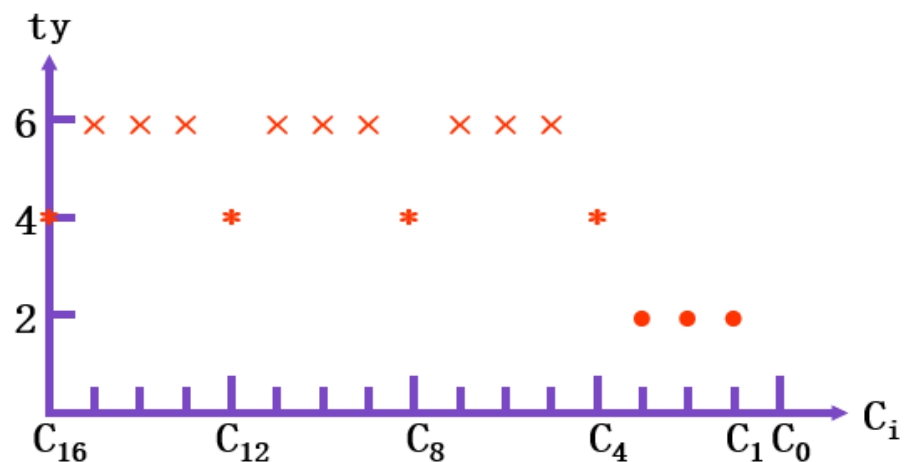
$$C_{12} = G_3^* + P_3^* G_2^* + P_3^* P_2^* G_1^* + P_3^* P_2^* P_1^* C_0$$

$$C_{16} = G_4^* + P_4^* G_3^* + P_4^* P_3^* G_2^* + P_4^* P_3^* P_2^* G_1^* + P_4^* P_3^* P_2^* P_1^* C_0$$

(2)多级先行进位方式



若不考虑 G_i 、 P_i 的形成时间， C_0 经过 $2ty$ 产生第1小组的 C_1 、 C_2 、 C_3 及所有组进位产生函数 G_i^* 和组进位传递函数 P_i^* ；再经过 $2ty$ ，产生 C_4 、 C_8 、 C_{12} 、 C_{16} ；最后经过 $2ty$ 后，才能产生第2、3、4小组内的 $C_5 \sim C_7$ 、 $C_9 \sim C_{11}$ 、 $C_{13} \sim C_{15}$ 。



计算机内定点数一般用**补码**表示；**思考为什么？**

补码加减运算规则如下：

- (1) 参加运算的两个操作数均用补码表示；
- (2) 符号位作为数的一部分参加运算；
- (3) 若做加法，则两数直接相加；若做减法，则将被减数与减数的机器负数相加；
- (4) 运算结果用补码表示。

结论公式：

$$[X + Y]_{\text{补}} = [X]_{\text{补}} + [Y]_{\text{补}} \quad (1)$$

$$[X - Y]_{\text{补}} = [X]_{\text{补}} + [-Y]_{\text{补}} \quad (2)$$

其中， $[-Y]_{\text{补}}$ 称为 $[Y]_{\text{补}}$ 的机器负数， $[-Y]_{\text{补}} = [[Y]_{\text{补}}]_{\text{变补}}$

$[Y]_{\text{补}}$ 连同符号一起变反、末尾+1。



例1: $A=0.1011$, $B=-0.1110$,

求: $A+B$

$$\because [A]_{\text{补}}=0.1011, [B]_{\text{补}}=1.0010$$

$$\begin{array}{r} 0.1011 \\ + 1.0010 \\ \hline 1.1101 \end{array}$$

$$\therefore [A+B]_{\text{补}}=1.1101, A+B=-0.0011$$

例2: $A=0.1011$, $B=-0.0010$,

求: $A-B$

$$\because [A]_{\text{补}}=0.1011, [B]_{\text{补}}=1.1110, \\ [-B]_{\text{补}}=0.0010$$

$$\begin{array}{r} 0.1011 \\ + 0.0010 \\ \hline 0.1101 \end{array}$$

$$\therefore [A-B]_{\text{补}}=0.1101, A-B=0.1101$$

两数补码加减计算结果一定可用吗?

思考：什么时候会溢出？

溢出检测方法

设：被操作数为： $[X]_{\text{补}} = X_s, X_1 X_2 \dots X_n$

其和（差）为： $[S]_{\text{补}} = S_s, S_1 S_2 \dots S_n$

操作数为： $[Y]_{\text{补}} = Y_s, Y_1 Y_2 \dots Y_n$

产生的进位为 $C_s, C_1 C_2 \dots C_n$

(1) 硬件检测方法一：采用一个符号位

$$\text{溢出} = \overline{X_s} \overline{Y_s} S_s + X_s Y_s \overline{S_s}$$

(2) 硬件检测方法二：采用进位位

$$\text{溢出} = C_s \overline{C_1} + \overline{C_s} C_1 = C_s \oplus C_1$$

(3) 硬件检测方法三：采用变形补码（双符号位）

$$\text{溢出} = S_{s1} \oplus S_{s2}$$

在双符号位的情况下，把左边的符号位 S_{s1} 叫做真符，因为它代表了该数真正的符号，两个符号位都作为数的一部分参加运算。这种编码又称为变形补码。变形补码存储时仍然保存单符号位，运算时扩充成双符号位。

双符号位的含义如下：

$S_{s1} S_{s2} = 00$ 结果为正数，无溢出

$S_{s1} S_{s2} = 01$ 结果正溢

$S_{s1} S_{s2} = 10$ 结果负溢

$S_{s1} S_{s2} = 11$ 结果为负数，无溢出

变形补码的本质是扩大了模，对于定点小数来说，模为4，对于字长为 $n+2$ 位的整数来说，模为 2^{n+2}

原码一位乘法类似于手工计算（自学）。

$$\text{乘积 } P = |X| \times |Y| \quad \text{符号 } P_s = X_s \oplus Y_s$$

补码一位乘法：Booth乘法
乘法运算需要3个寄存器：

A寄存器：部分积与最后乘积的高位部分，初值为0。

B寄存器：被乘数X。

C寄存器：乘数Y，运算后C寄存器中不再需要保留乘数，改为存放乘积的低位部分。

Booth乘法规则如下：

- 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_{n+1} ，其初值为0；
- ④ 由于每求一次部分积要右移一位，所以乘数的最低两位 Y_n 、 Y_{n+1} 的值决定了每次应执行的操作；

判断位 Y_n Y_{n+1} 操 作

0 0	原部分积右移一位
0 1	原部分积加 $[X]_{\text{补}}$ 后右移一位
1 0	原部分积加 $[-X]_{\text{补}}$ 后右移一位
1 1	原部分积右移一位

- ⑤ 移位按补码右移规则进行；
- ⑥ 共需做 $n+1$ 次累加， n 次移位，第 $n+1$ 次不移位。

例：已知 $X=-0.1101$ ， $Y=0.1011$ ；求 $X\times Y$ 。

$[X]_{\text{补}}=1.0011\rightarrow B$ ， $[Y]_{\text{补}}=0.1011\rightarrow C$ ， $0\rightarrow A$ $[-X]_{\text{补}}=0.1101$

		A	C	附加位	说明
		0.1011	10		
$+ [-X]_{\text{补}}$	$\begin{array}{r} 000000 \\ 001101 \\ \hline 001101 \end{array}$	0.1011	10		$C_4C_5=10$ ， $+ [-X]_{\text{补}}$
\rightarrow	$\begin{array}{r} 001101 \\ 000110 \\ \hline 000110 \end{array}$	1010	11		部分积右移一位
$+0$	$\begin{array}{r} 000110 \\ 000000 \\ \hline 000110 \end{array}$	1010	11		$C_4C_5=11$ ， $+0$
\rightarrow	$\begin{array}{r} 000110 \\ 000011 \\ \hline 000011 \end{array}$	0101	01		部分积右移一位
$+ [X]_{\text{补}}$	$\begin{array}{r} 000011 \\ 110011 \\ \hline 110011 \end{array}$	0101	01		$C_4C_5=01$ ， $+ [X]_{\text{补}}$
\rightarrow	$\begin{array}{r} 110011 \\ 111011 \\ \hline 001011 \end{array}$	0010	10		部分积右移一位
$+ [-X]_{\text{补}}$	$\begin{array}{r} 001011 \\ 001101 \\ \hline 001000 \end{array}$	0010	10		$C_4C_5=10$ ， $+ [-X]_{\text{补}}$
\rightarrow	$\begin{array}{r} 001000 \\ 001000 \\ \hline 001000 \end{array}$	0001	01		部分积右移一位
$+ [X]_{\text{补}}$	$\begin{array}{r} 001000 \\ 110011 \\ \hline 111011 \end{array}$	1110	11		$C_4C_5=01$ ， $+ [X]_{\text{补}}$
	111011				
$\therefore [X\times Y]_{\text{补}}=1.01110001$					
$\therefore X\times Y=-0.10001111$					

思考：还可以更快吗？

自学：补码两位乘法

1.原码比较法、恢复余数法和不恢复余数法（原码加减交替法）（自学）

2.补码加减交替除法规则，求新余数公式： $[r_{i+1}]_{\text{补}} = 2[r_i]_{\text{补}} + (1 - 2Q_i) \times [Y]_{\text{补}}$

$[X]_{\text{补}}$ 与 $[Y]_{\text{补}}$	第一次操作	$[r_i]_{\text{补}}$ 与 $[Y]_{\text{补}}$	上商	求新余数 $[r_{i+1}]_{\text{补}}$ 的操作
同号	$[X]_{\text{补}} - [Y]_{\text{补}}$	①同号 (够减)	1	$[r_{i+1}]_{\text{补}} = 2[r_i]_{\text{补}} - [Y]_{\text{补}}$
		②异号 (不够减)	0	$[r_{i+1}]_{\text{补}} = 2[r_i]_{\text{补}} + [Y]_{\text{补}}$
异号	$[X]_{\text{补}} + [Y]_{\text{补}}$	①同号 (不够减)	1	$[r_{i+1}]_{\text{补}} = 2[r_i]_{\text{补}} - [Y]_{\text{补}}$
		②异号 (够减)	0	$[r_{i+1}]_{\text{补}} = 2[r_i]_{\text{补}} + [Y]_{\text{补}}$

已知: $X=0.1000$, $Y=-0.1010$; 求 $X\div Y$

$[X]_{\text{补}}=0.1000\rightarrow A$, $[Y]_{\text{补}}=1.0110\rightarrow B$, $0\rightarrow C$, $[-Y]_{\text{补}}=0.1010$

	0	0	1	0	0	0
$+ [Y]_{\text{补}}$	1	1	0	1	1	0
	1	1	1	1	1	0
←	1	1	1	1	0	0
$+ [-Y]_{\text{补}}$	0	0	1	0	1	0
	0	0	0	1	1	0
←	0	0	1	1	0	0
$+ [Y]_{\text{补}}$	1	1	0	1	1	0
	0	0	0	0	1	0
←	0	0	0	1	0	0
$+ [Y]_{\text{补}}$	1	1	0	1	1	0
	1	1	1	0	1	0
←	1	1	0	1	0	0
$+ [-Y]_{\text{补}}$	0	0	1	0	1	0
	1	1	1	1	1	0

0	0	0	0	0
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	1	0	0	1
1	0	0	1	1

$[X]_{\text{补}}$ 、 $[Y]_{\text{补}}$ 异号, $+ [Y]_{\text{补}}$
 $[r_i]_{\text{补}}$ 、 $[Y]_{\text{补}}$ 同号, 商1
左移一位
 $+ [-Y]_{\text{补}}$
 $[r_i]_{\text{补}}$ 、 $[Y]_{\text{补}}$ 异号, 商0
左移一位
 $+ [Y]_{\text{补}}$
 $[r_i]_{\text{补}}$ 、 $[Y]_{\text{补}}$ 异号, 商0
左移一位
 $+ [Y]_{\text{补}}$
 $[r_i]_{\text{补}}$ 、 $[Y]_{\text{补}}$ 同号, 商1
左移一位
 $+ [-Y]_{\text{补}}$
末位恒置1



除法运算需要3个寄存器：

A寄存器：存放被除数X，最后A寄存器中剩下的是扩大了若干倍的余数。运算过程中A寄存器的内容将不断地发生变化。

B寄存器：存放除数Y。

C寄存器：存放商Q，它的初值为0。

$$[\text{商}]_{\text{补}} = 1.0011$$

$$[\text{余数}]_{\text{补}} = 1.1110 \times 2^{-4}$$

$$\begin{aligned} [X \div Y] \\ &= 1.0011 + \\ \therefore \text{商} &= -0.1101 \end{aligned}$$

$$\begin{array}{r} 1.1110 \times 2^{-4} \\ \text{补 } 1.0110 \end{array}$$

$$\text{余数} = -0.0010 \times 2^{-4}$$

$$X \div Y = -0.1101 + \frac{-0.0010 \times 2^{-4}}{-0.1010}$$

浮点加减运算

设两个非0的规格化浮点数分别为

$$A = M_A \times 2^{E_A}$$

$$B = M_B \times 2^{E_B}$$

$$A \pm B = (M_A, E_A) \pm (M_B, E_B) = \begin{cases} M_A \pm M_B \times 2^{-(E_A - E_B)}, E_A & E_A > E_B \\ (M_A \times 2^{-(E_B - E_A)} \pm M_B, E_B) & E_A < E_B \end{cases}$$

(1)对阶： 规则是： **小阶向大阶看齐**，每右移一位，阶码加1。

(2)尾数加/减 $M_A \pm M_B \rightarrow M_C$

(3)尾数结果规格化，设尾数用双符号位补码表示，经过加/减运算之后，可能出现以下六种情况：

① 00.1 x x ... x

② 11.0 x x ... x

③ 00.0 x x ... x

④ 11.1 x x ... x

⑤ 01.x x x ... x

⑥ 10.x x x ... x

第①、②种情况，已是规格化数。

第③、④种情况需要使尾数左移以实现规格化，这个过程称为左规。尾数每左移一位，阶码相应减1，直至成为规格化数为止。（左规可能需进行多次）

第⑤、⑥种情况在定点加减运算中称为溢出；但在浮点加减运算中，只表明此时尾数的绝对值大于1，而并非真正的溢出。这种情况应将尾数右移以实现规格化。这个过程称为右规。尾数每右移一位，阶码相应加1。（右规最多进行一次）

当尾数之和（差）出现 $10.x \ x \ x \ \dots \ x$ 或 $01.x \ x \ x \ \dots \ x$ 时，并不表示溢出，只有将此数右规后，再根据阶码来判断浮点运算结果是否溢出。

浮点数的溢出情况由阶码的符号决定，若阶码也用双符号位补码表示，

当： $[E_C]_{\text{补}}=01, x \ x \ x \ \dots \ x$ ，表示上溢。此时，浮点数真正溢出，机器需停止运算，做溢出中断处理。

$[E_C]_{\text{补}}=10, x \ x \ x \ \dots \ x$ ，表示下溢。浮点数值趋于零，机器不做溢出处理，而是按机器零处理。

设两个非0的规格化浮点数分别为

$$A = M_A \times 2^{E_A}$$

$$B = M_B \times 2^{E_B}$$

则浮点乘法和除法为

$$A \times B = (M_A \times M_B) \times 2^{(E_A + E_B)}$$

$$A \div B = (M_A \div M_B) \times 2^{(E_A - E_B)}$$

1.乘法步骤

(1)阶码相加

两个浮点数的阶码相加，当阶码用移码表示的时候，应注意要减去一个偏置值 2^n 。

因为 $[E_A]_{\text{移}} = 2^n + E_A$ ， $[E_B]_{\text{移}} = 2^n + E_B$

$[E_A + E_B]_{\text{移}} = 2^n + (E_A + E_B)$

而 $[E_A]_{\text{移}} + [E_B]_{\text{移}} = 2^n + E_A + 2^n + E_B$

显然，此时阶码和中多余了一个偏置值 2^n ，应将它减去。
另外，阶码相加后有可能产生溢出，此时应另作处理。

(2)尾数相乘

与定点小数乘法算法相同。

(3)尾数结果规格化

因为 $1/2 < |M_A| < 1$, $1/2 < |M_B| < 1$, 所以 $1/4 < |M_A \times M_B| < 1$ 。

当 $1/2 < |M_A \times M_B| < 1$ 时, 乘积已是规格化数, 不须再进行规格化操作; 当 $1/4 < |M_A \times M_B| < 1/2$ 时, 则需要左规一次。

2.除法步骤

(1)尾数调整

首先须要检测 $|M_A| < |M_B|$ 。如果不小于，则 M_A 右移一位， $E_A + 1 \rightarrow E_A$ ，称为尾数调整。因为A、B都是规格化数，所以最多调整一次。

(2)阶码相减

两浮点数的阶码相减，当阶码用移码表示时，应注意要加上一个偏置值 2^n 。

(3)尾数相除

与定点小数除法算法相同。

逻辑运算比算术运算要简单得多，这是因为**逻辑运算是按位进行的，位与位之间没有进位/借位的关系。**

1. 逻辑非

逻辑非又称求反操作，它对某个寄存器或主存单元中各位代码按位取反。

2. 逻辑乘

逻辑乘就是将两个寄存器或主存单元中的每一相应位的代码进行“与”操作。

3.逻辑加

逻辑加就是将两个寄存器或主存单元中的每一相应位的代码进行“或”操作。

4.按位异或

按位异或是计算机中一个特定的逻辑操作，它对寄存器或主存单元中各位的代码求模2和，又称模2加或半加，也叫异或。

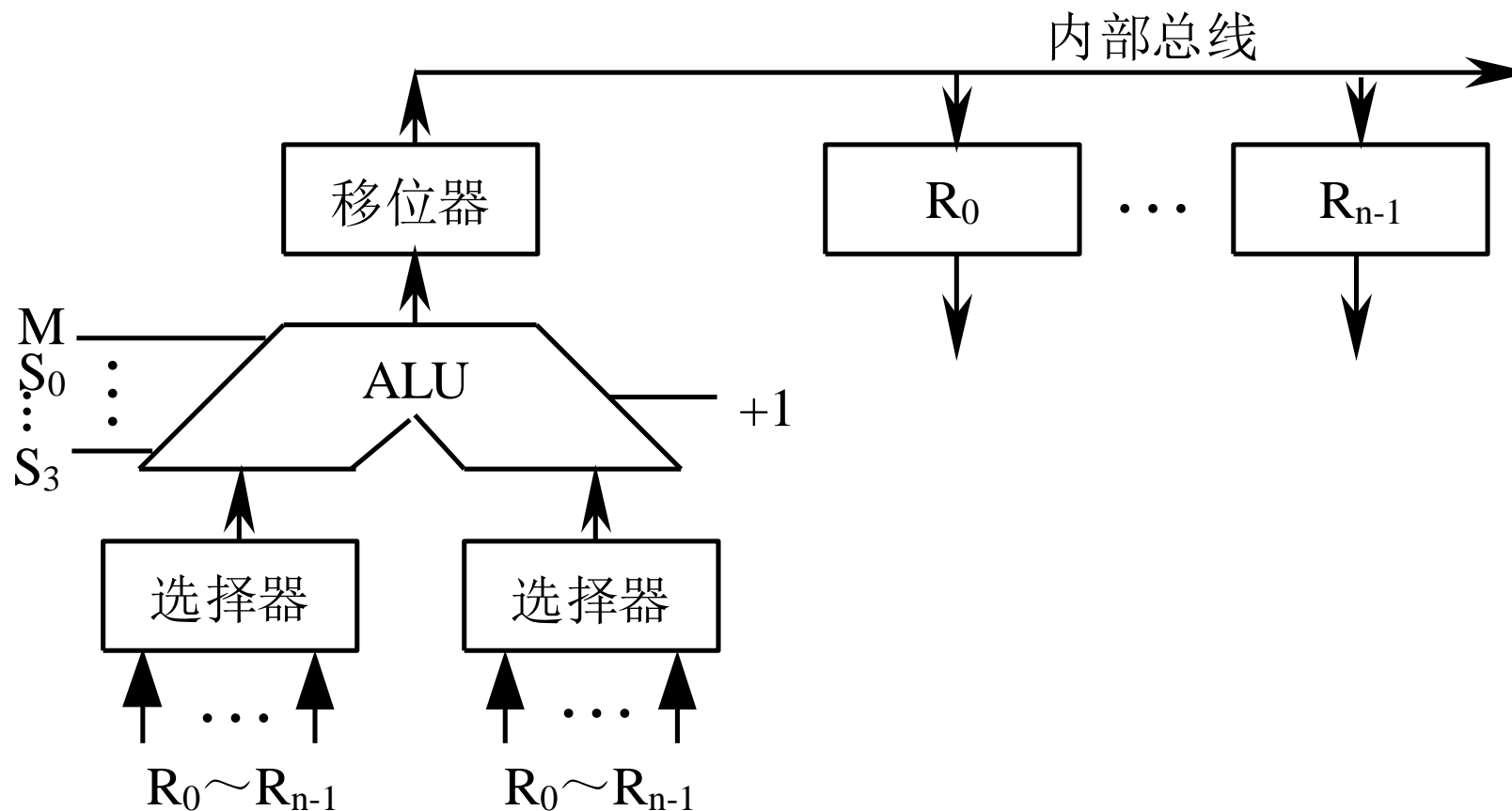
运算器是在控制器的控制下实现其功能的，运算器不仅可以完成数据信息的算逻运算，还可以作为数据信息的传送通路。

1.运算器的基本组成

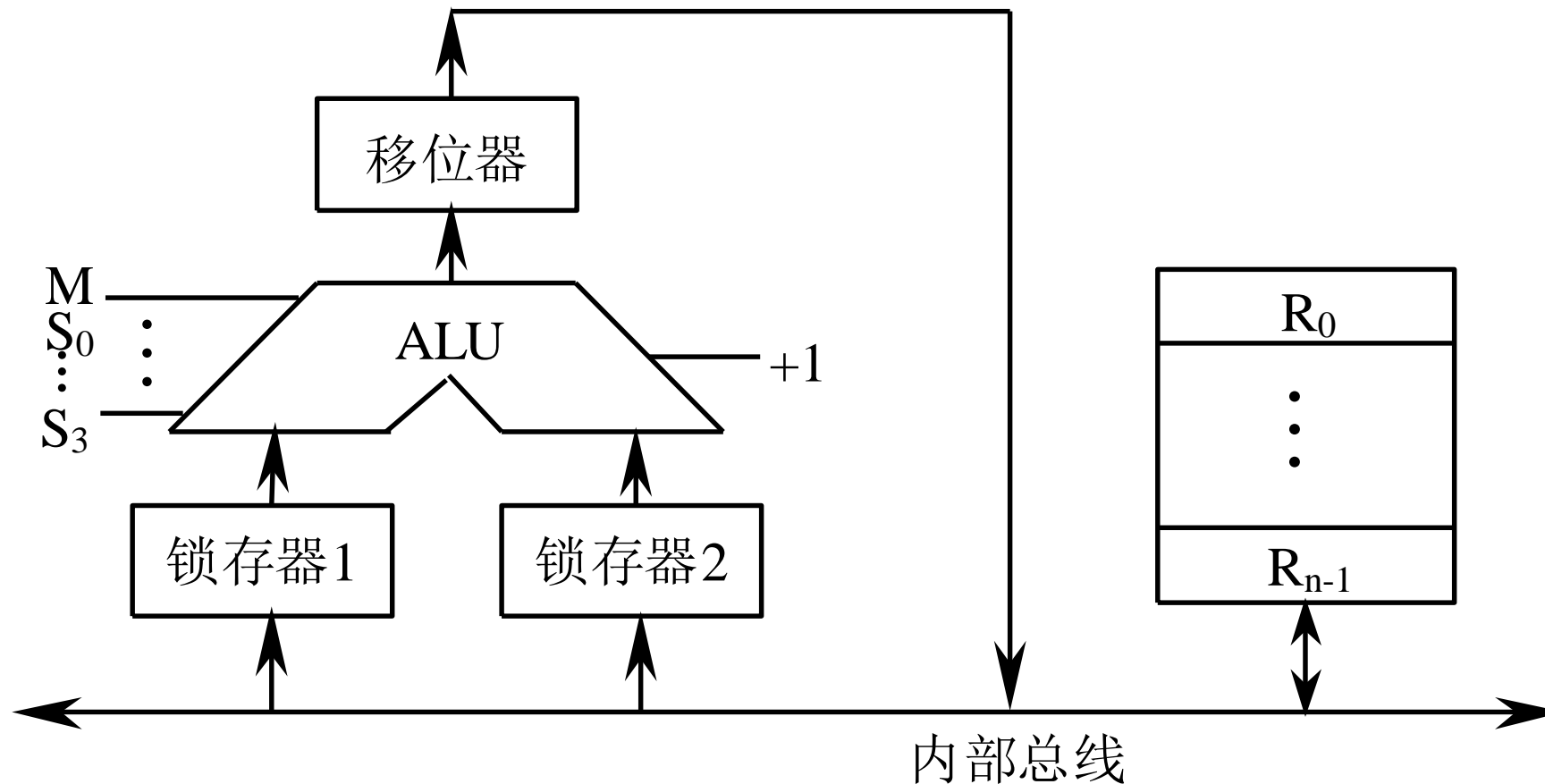
基本的运算器包含以下几个部分：

**实现基本算术、逻辑运算功能的ALU，
提供操作数与暂存结果的寄存器组，
有关的判别逻辑和控制电路等。**

(1) 带多路选择器的运算器



(2)带输入锁存器的运算器



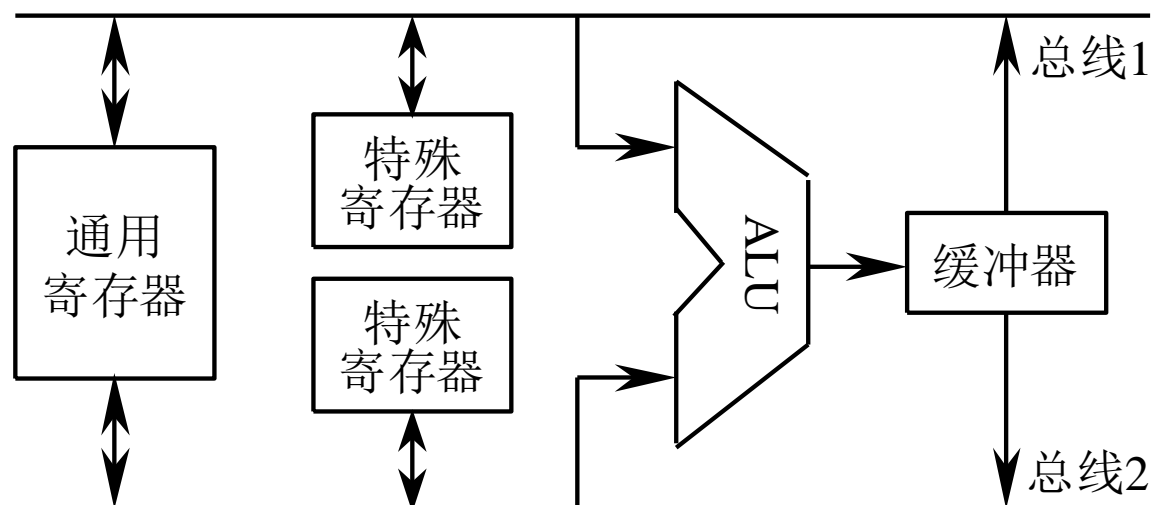
2.运算器的内部总线结构

(1)单总线结构运算器

运算器实现一次双操作数的运算需要分成三步。

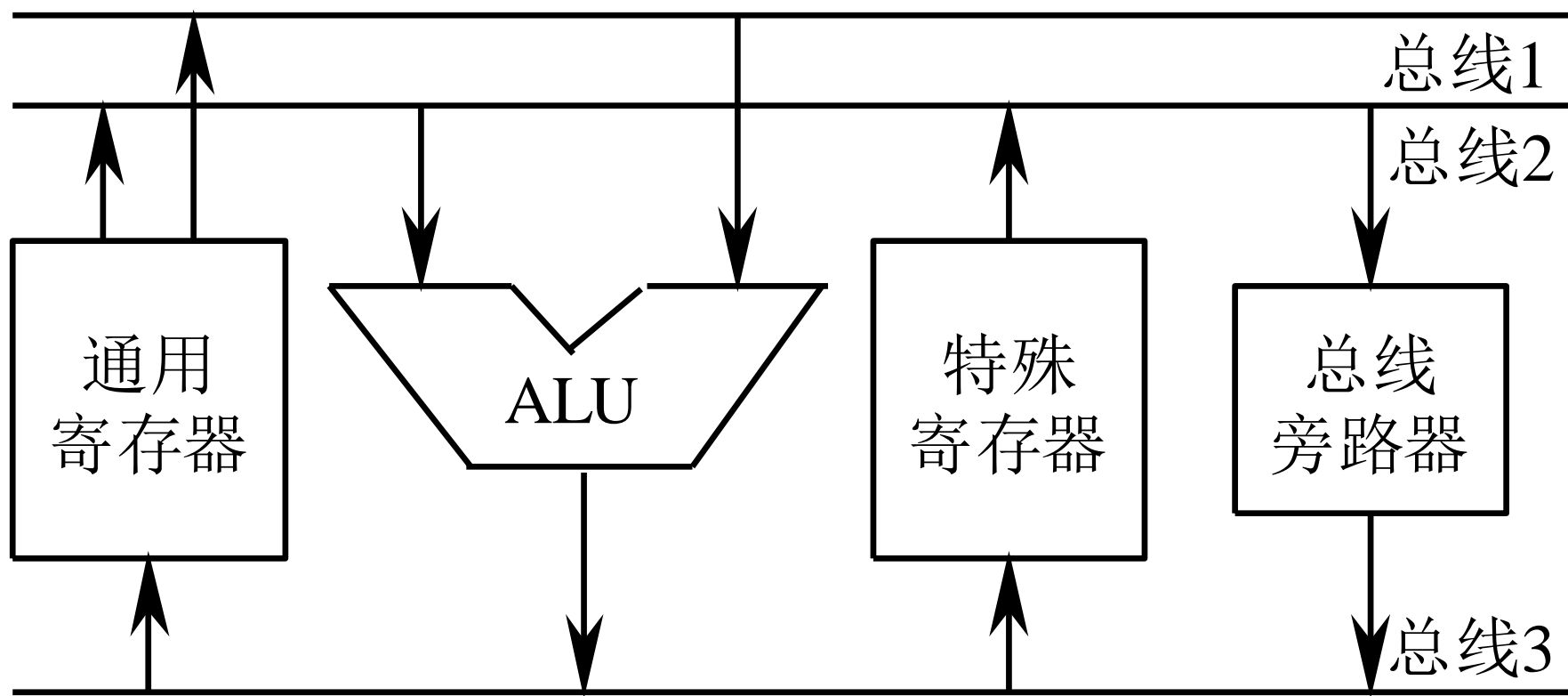
(2)双总线结构运算器

运算器实现一次双操作数的运算需要两步。



(3) 三总线结构运算器

实现一次双操作数的运算仅需要一步。



1. ALU电路

ALU即算术逻辑单元，它是既能完成算术运算又能完成逻辑运算的部件。前面已经讨论过，无论是加、减、乘、除运算，最终都能归结为加法运算。因此，ALU的核心首先应当是一个并行加法器，同时也能执行像“与”、“或”、“非”、“异或”这样的逻辑运算。由于ALU能完成多种功能，所以ALU又称多功能函数发生器。

2.4位ALU芯片

74181是四位算术逻辑运算部件（ALU），又称多功能函数发生器，能执行16种算术运算和16种逻辑运算。

A0、B0 ~ A3、B3：操作数输入端；

F0 ~ F3：输出端；

C_n' ：进位输入端；

C_{n+4}' ：进位输出端；

G^* ：组进位产生函数输出端；

P^* ：组进位传递函数输出端；

运算器的基本组成与实例

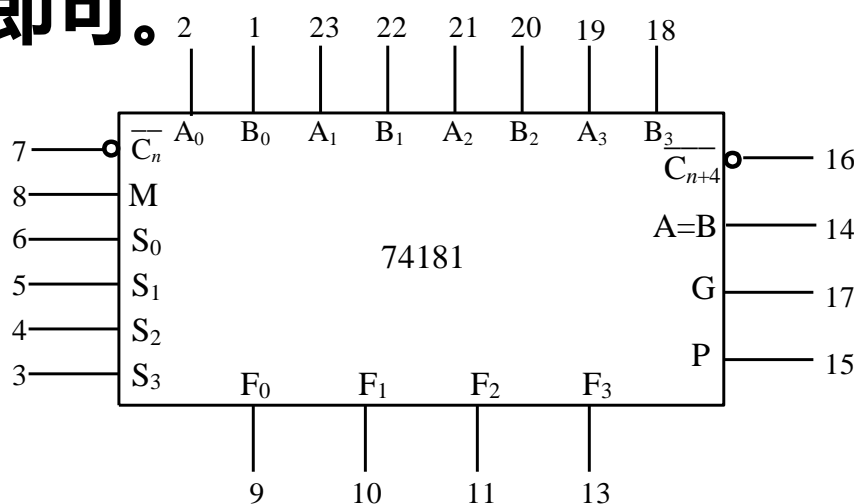


M: 工作方式, $M=0$ 为算术操作, $M=1$ 为逻辑操作;

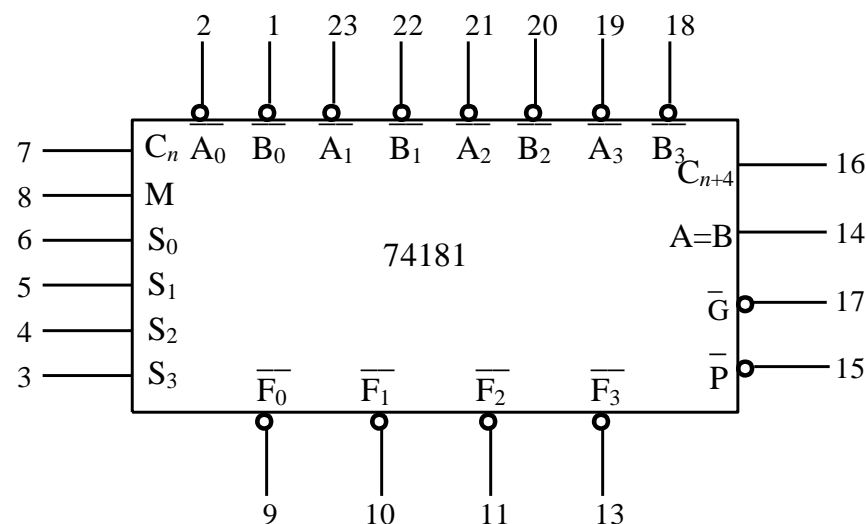
$S_0 \sim S_3$: 功能选择线。

74181的4位作为一个小组, 组间既可以采用串行进位, 也可以采用并行进位。

当采用组间串行进位时, 只要把前片的 C_{n+4} 与下一片的 C_n 相连即可。



(a)



(b)

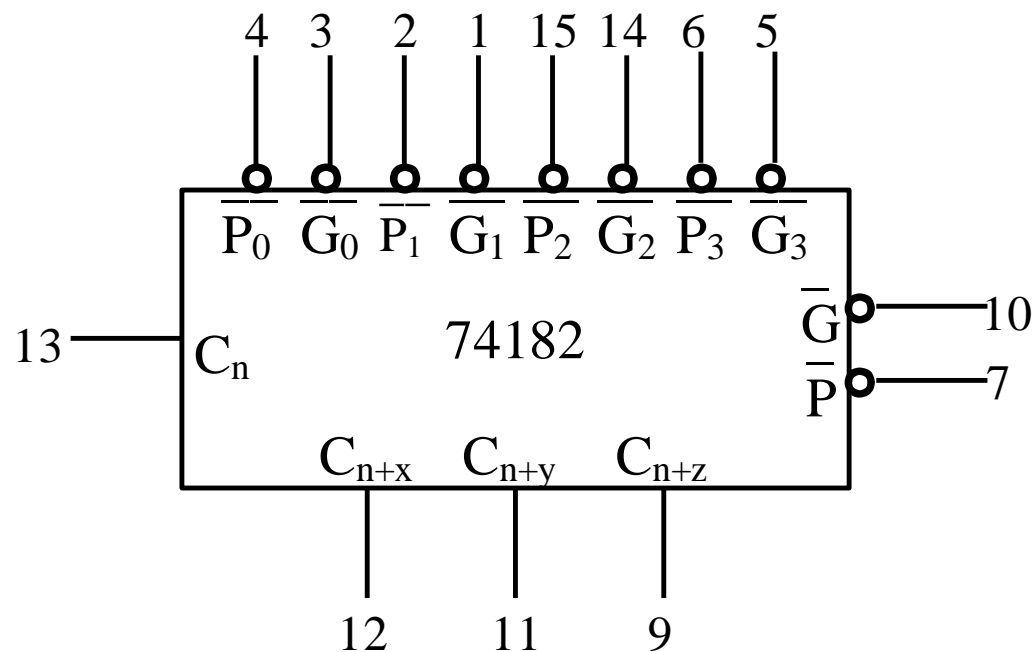
运算器的基本组成与实例



工作选择 $S_3S_2S_1S_0$	负逻辑			正逻辑		
	逻辑运算 ($M=1$)	算术运算 ($M=0$) $C_n=0$ (无进位)	算术运算 ($M=0$) $C_n=1$ 有进位)	逻辑运算 ($M=1$)	算术运算 ($M=0$) $\overline{C_n}=1$ (无进位)	算术运算 ($M=0$) $\overline{C_n}=0$ 有进位)
0000	$F=\overline{A}$	$F=A$ 减 1	$F=A$	$F=\overline{A}$	$F=A$	$F=A$ 加 1
0001	$F=\overline{A}\overline{B}$	$F=AB$ 减 1	$F=AB$	$F=\overline{A+B}$	$F=A+B$	$F=(A+B)$ 加 1
0010	$F=\overline{A}+B$	$F=A\overline{B}$ 减 1	$F=A\overline{B}$	$F=\overline{A}B$	$F=A+\overline{B}$	$F=(A+\overline{B})$ 加 1
0011	$F=1$	$F=\text{减 } 1$	$F=0$	$F=0$	$F=\text{减 } 1$	$F=0$
0100	$F=\overline{A+B}$	$F=A$ 加 $(A+\overline{B})$	$F=A$ 加 $(A+\overline{B})$ 加 1	$F=\overline{A}\overline{B}$	$F=A$ 加 $A\overline{B}$	$F=A$ 加 $A\overline{B}$ 加 1
0101	$F=\overline{B}$	$F=AB$ 加 $(A+\overline{B})$	$F=AB$ 加 $(A+\overline{B})$ 加 1	$F=\overline{B}$	$F=(A+B)$ 加 $A\overline{B}$	$F=(A+B)$ 加 $A\overline{B}$ 加 1
0110	$F=\overline{A\oplus B}$	$F=A$ 减 B 减 1	$F=A$ 减 B	$F=A\oplus B$	$F=A$ 减 B 减 1	$F=A$ 减 B
0111	$F=A+\overline{B}$	$F=A+\overline{B}$	$F=(A+\overline{B})$ 加 1	$F=A\overline{B}$	$F=A\overline{B}$ 减 1	$F=A\overline{B}$
1000	$F=\overline{A}B$	$F=A$ 加 $(A+B)$	$F=A$ 加 $(A+B)$ 加 1	$F=\overline{A}+B$	$F=A$ 加 AB	$F=A$ 加 AB 加 1
1001	$F=A\oplus B$	$F=A$ 加 B	$F=A$ 加 B 加 1	$F=\overline{A\oplus B}$	$F=A$ 加 B	$F=A$ 加 B 加 1
1010	$F=B$	$F=A\overline{B}$ 加 $(A+B)$	$F=A\overline{B}$ 加 $(A+B)$ 加 1	$F=B$	$F=(A+\overline{B})$ 加 AB	$F=(A+\overline{B})$ 加 AB 加 1
1011	$F=A+B$	$F=A+B$	$F=(A+B)$ 加 1	$F=AB$	$F=AB$ 减 1	$F=AB$
1100	$F=0$	$F=A$ 加 A^*	$F=A$ 加 A 加 1	$F=1$	$F=A$ 加 A^*	$F=A$ 加 A 加 1
1101	$F=A\overline{B}$	$F=AB$ 加 A	$F=AB$ 加 A 加 1	$F=A+\overline{B}$	$F=(A+B)$ 加 A	$F=(A+B)$ 加 A 加 1
1110	$F=AB$	$F=A\overline{B}$ 加 A	$F=A\overline{B}$ 加 A 加 1	$F=A+B$	$F=(A+\overline{B})$ 加 A	$F=(A+\overline{B})$ 加 A 加 1
1111	$F=A$	$F=A$	$F=A$ 加 1	$F=A$	$F=A$ 减 1	$F=A$

3. ALU的应用

当采用组间并行进位时，需要增加一片先行进位部件（74182）。



74182可以产生三个进位信号 C_{n+x} 、 C_{n+y} 、 C_{n+z} ，并且还产生大组进位产生函数 G^{**} 和大组进位传递函数 P^{**} ，可供组成位数更长的多级先行进位ALU时用。

$$C_{16} = \underbrace{G_4^* + P_4^* G_3^* + P_4^* P_3^* G_2^* + P_4^* P_3^* P_2^* G_1^*}_{G_1^{**}} + \underbrace{P_4^* P_3^* P_2^* P_1^*}_{P_1^{**}} C_0$$

$$= G_1^{**} + P_1^{**} C_0$$

大组进位
产生函数 G_1^{**}

大组进位
传递函数 P_1^{**}

74181和74182的结合可组成各种位数的ALU部件。

8片74181和2片74182构成的32位两级行波ALU。各片74181输出的组进位产生函数和组进位传递函数作为74182的输入，而74182输出的进位信号 C_{n+x} 、 C_{n+y} 、 C_{n+z} 作为74181的输入，74182输出的大组进位产生函数和大组进位传递函数可作为更高一级74182的输入。

实验一：设计并实现一个32位ALU



- 要求：
- 1、支持至少8种运算，自行选择，实验报告中说明；
 - 2、输出5个标志符号：SF(符号位)，CF（进位标志），ZF（零标志），（溢出标志）和PF（奇偶标志）；
 - 3、支持左右移位操作；
 - 4、可支持至少两种舍入操作。
 - 5、（可选）对ALU进行改进，写出改进目标，以及改进的量化对比结果。（例如：时间上的缩短，逻辑单元减少等）

实验环境vivado2019.2 语言：Verilog HDL。

- 乐学提交：
- 1、实验报告，姓名-学号-实验一.pdf或doc/docx格式
 - 2、带源代码的工程文件。姓名-学号-实验一.zip

- **重要认识1：**计算机中所有算术运算都基于**加法器实现**， **n 位**计算时是一种**模 2^n 运算系统**！
- **重要认识2：****无符号整数**和**带符号整数**的加、减运算电路完全一样，这个运算电路称为**整数加减运算部件**，基于**带标志加法器实现**。
- **重要认识3：****加法器不判定对错**，总是取低 **n 位**作为结果，并生成标志信息。

示例：n位整数加/减运算部件

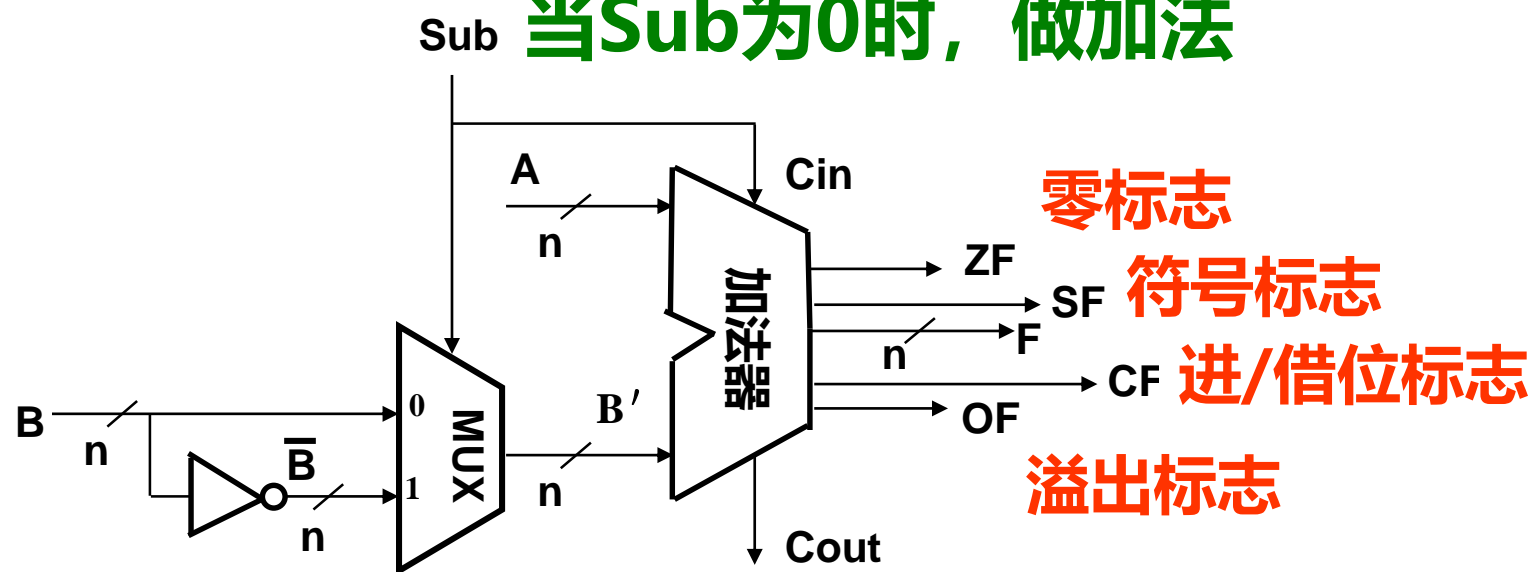
- 利用带标志加法器，可构造整数加/减运算器，进行以下运算：

无符号整数加、无符号整数减

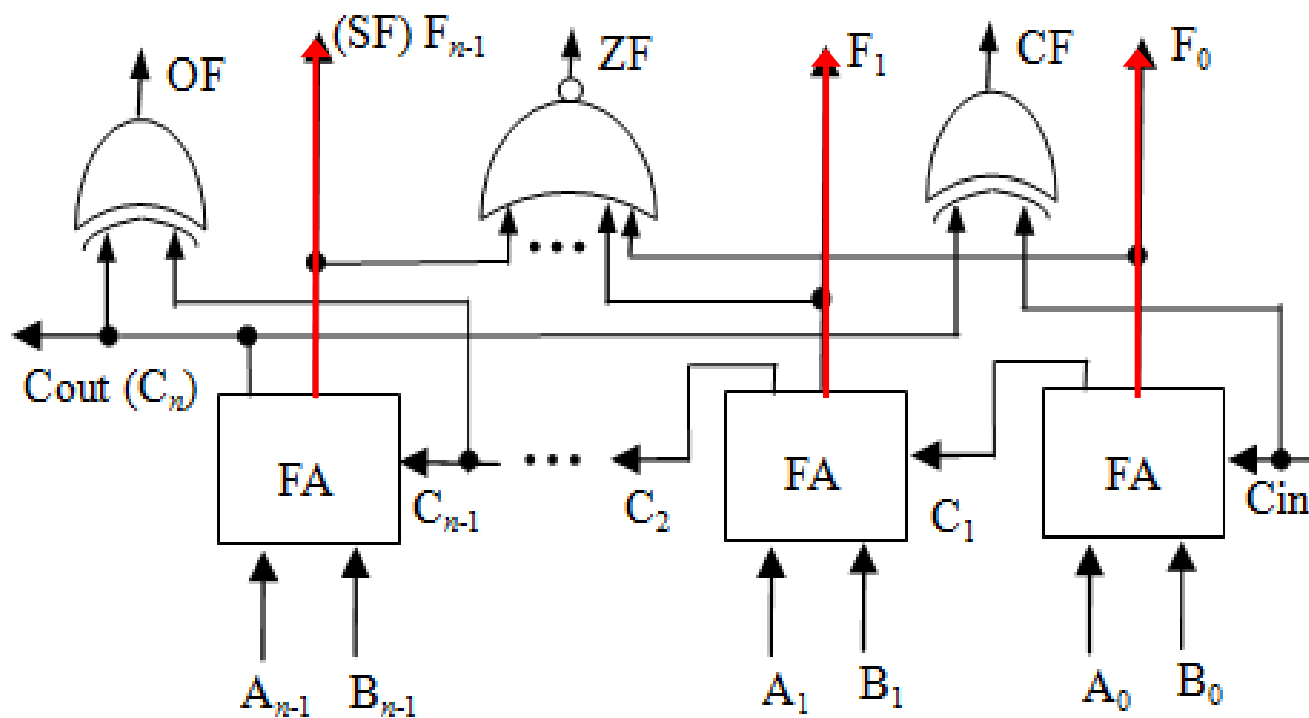
带符号整数加、带符号整数减

当Sub为1时，做减法
当Sub为0时，做加法

在整数加/减运算部件
基础上，加上寄存器、
移位器以及控制逻辑，
就可实现**ALU**、**乘/除**
运算以及**浮点**运算电路



整数加/减运算部件



带标志加法器的逻辑电路

溢出标志OF: $OF = C_n \oplus C_{n-1}$

符号标志SF: $SF = F_{n-1}$

零标志ZF=1当且仅当F=0;

进位/借位标志CF:

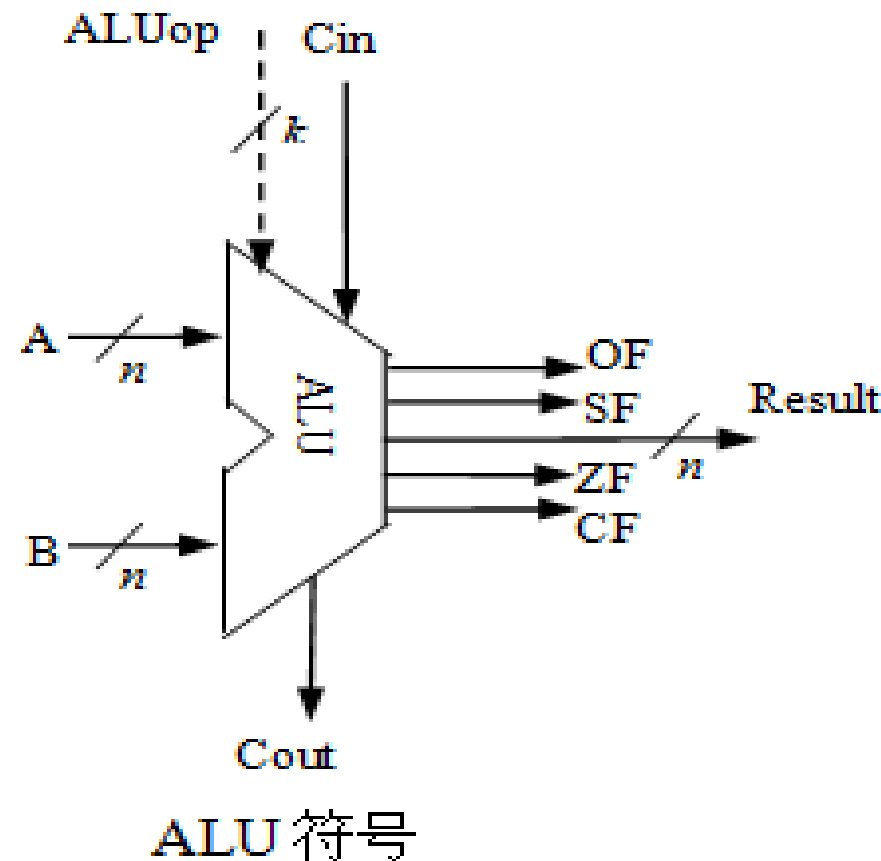
$$CF = Cout \oplus Cin$$

条件标志 (Flag) 在运算电路中产生，被记录到专门的寄存器中

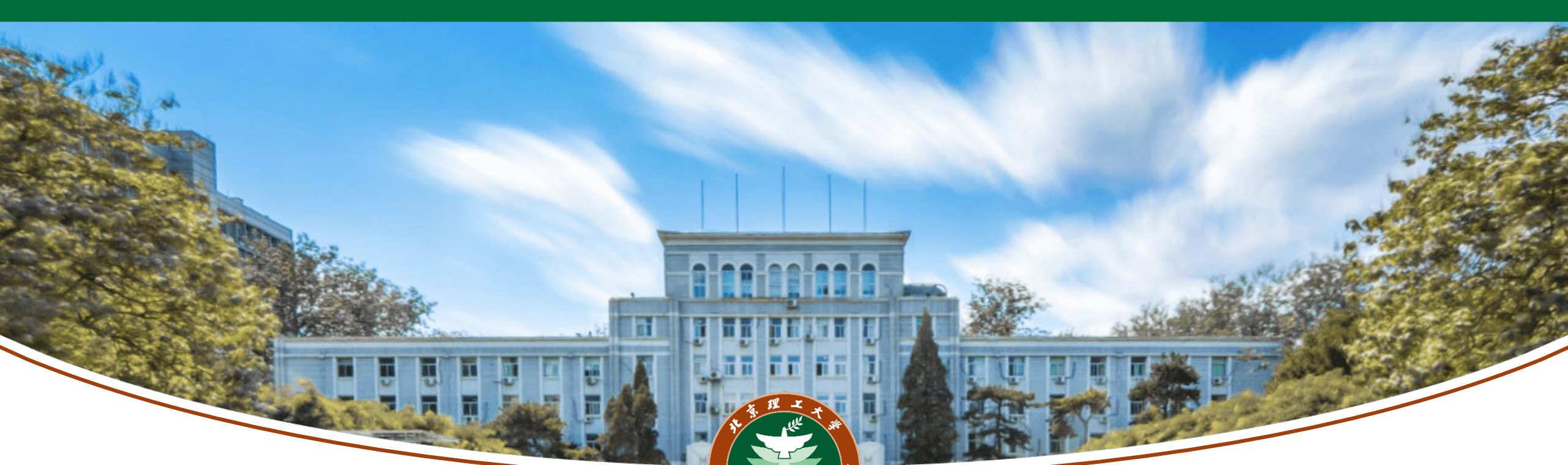
算术逻辑部件（ALU）：多功能函数发生器



- 进行**基本**算术运算与逻辑运算
 - 无符号整数加、减
 - 带符号整数加、减
 - 与、或、非、异或等逻辑运算
- 核心电路是**整数加/减运算部件**
- 输出除**和/差等**，还有**标志信息**
- 有一个**操作控制端**（ALUop），用来决定ALU所处理功能。ALUop的位数 k 决定了操作的种类，**位数 k 为3时**，ALU最多只有 $2^3=8$ 种操作。



ALUop	Result	ALUop	Result	ALUop	Result	ALUop	Result
0 0 0	A加B	0 1 0	A与B	1 0 0	A取反	1 1 0	A
0 0 1	A减B	0 1 1	A或B	1 0 1	$A \oplus B$	1 1 1	未用



感谢聆听