计算机组成原理

第6章 计算机的运算方法

llxx@ustc.edu.cn
wjluo@ustc.edu.cn

6.4 浮点四则运算

浮点加减运算

浮点乘除法运算

浮点数的表示

- 机器中任何一个浮点数可写成

$$x = S_x \cdot r^{j_x}$$

- S_x为浮点数的尾数，一般为绝对值小于1的规格化数(补码表示时允许为-1)，机器中可用原码或补码表示。
- j_x为浮点数的阶码，一般为整数，机器中大多用补码或移码表示。
- r为浮点数的基数，常用2、4、8或16表示。以下以基数为2进行讨论。

浮点加减运算

- 设两个浮点数

$$x = S_x \cdot r^{j_x}$$

$$y = S_y \cdot r^{j_y}$$

① 尾数的加减运算规则与定点数完全相同。

② 当两浮点数阶码不等时，因两尾数小数点的实际位置不一样，尾数部分无法直接进行加减运算。

浮点加减运算的步骤

- 阶码与尾数都用变形补码表示（双符号位）。

1. 对阶：使两数的小数点位置对齐。
   - 求阶差：小向大对齐（增加），尾数右移
2. 尾数求和：按定点加减运算规则求和。
   - 不考虑溢出——溢出由阶码决定
3. 规格化：增加有效数字的位数，提高运算精度
   - 补码表示的规格化数形式
4. 舍入：要考虑尾数右移时丢失的数值位对精度的影响。
   - 避免“积累误差”：截断、“0舍1入”法、“恒置1”法
5. 溢出判断：双符号位补码
   - 阶码$[j]_{补}$=01，××…×为上溢。
   - 阶码$[j]_{补}$=10，××…×为下溢，按机器零处理

- 例：
- 设$x=2^{-101}\times(-0.101000)$,$y=2^{-100}\times(+0.111011)$，并假设阶符取2位，阶码取3位，数符取2位，尾数取6位，求x-y。
- 解：由$x=2^{-101}\times(-0.101000)$，$y=2^{-100}\times(+0.111011)$

  得$[x]_{补}$=11,011;11.011000，$[y]_{补}$=11,100;00.111011

  ①对阶

  $[\Delta_j]_{补}=[j_x]_{补}-[j_y]_{补}$=11,011+00,100=11,111

  即Δ_j=-1,则x的尾数向右移一位，阶码相应加1，即

  $[x]'_{补}$=11,100;11.101100

  ②求和

  $[S_x]'_{补}-[S_y]_{补}=[S_x]_{补}+[-S_y]_{补}$

  =11.101100+11.000101

  =10.110001

  即 $[x-y]_{补}$=11,100;10.110001

  尾数符号位出现“10”，需右规。

- 解（续）：

即 $[x-y]_{补}$=11,100;10.110001， 尾数符号位出现“10”，需右规。

③规格化

右规后得 $[x-y]_{补}$=11,101;11.0110001

④舍入处理

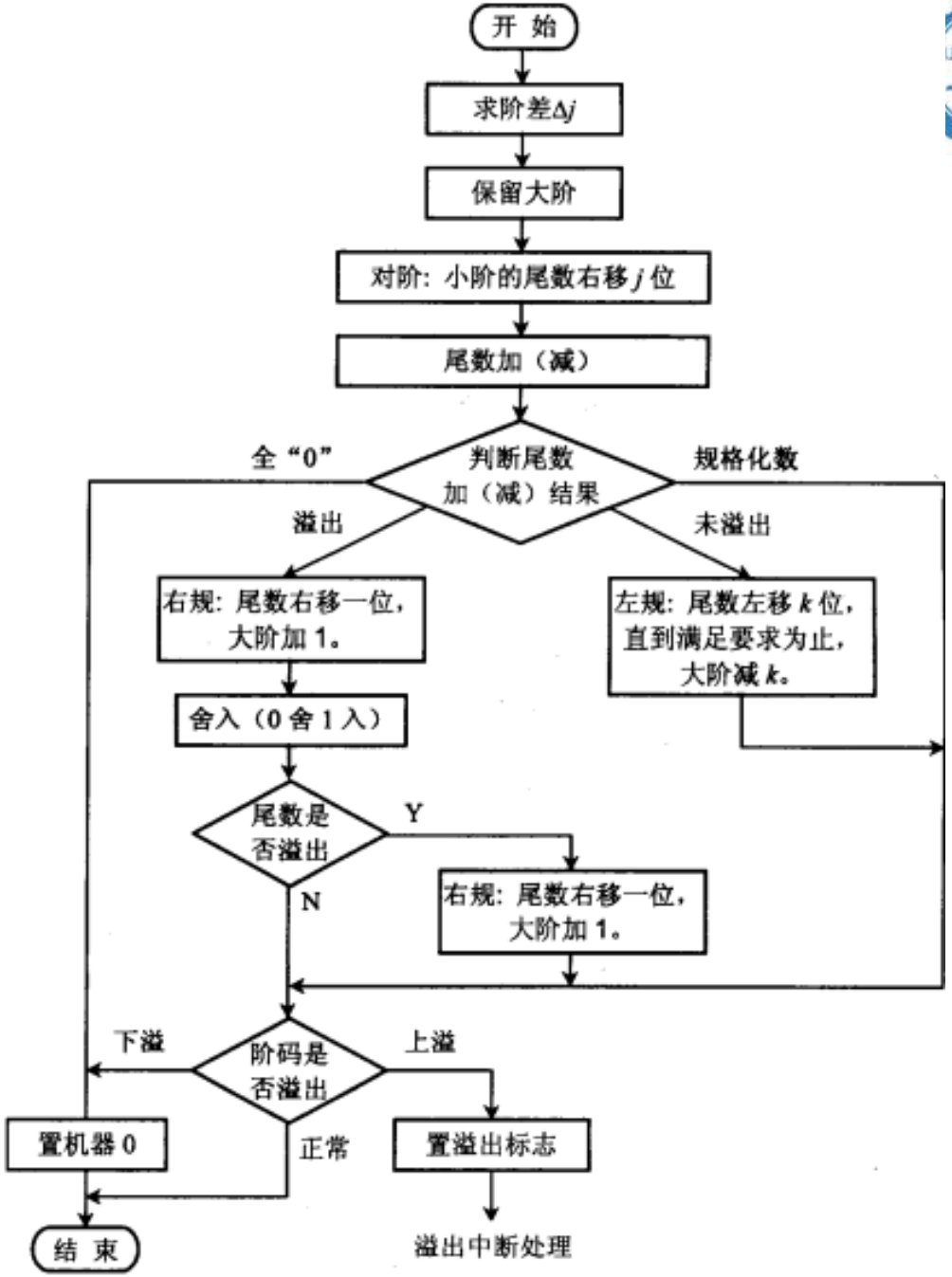
采用0舍1入法，其尾数右规时末位丢1，则

$[x-y]_{补}$=11,101;11.011001

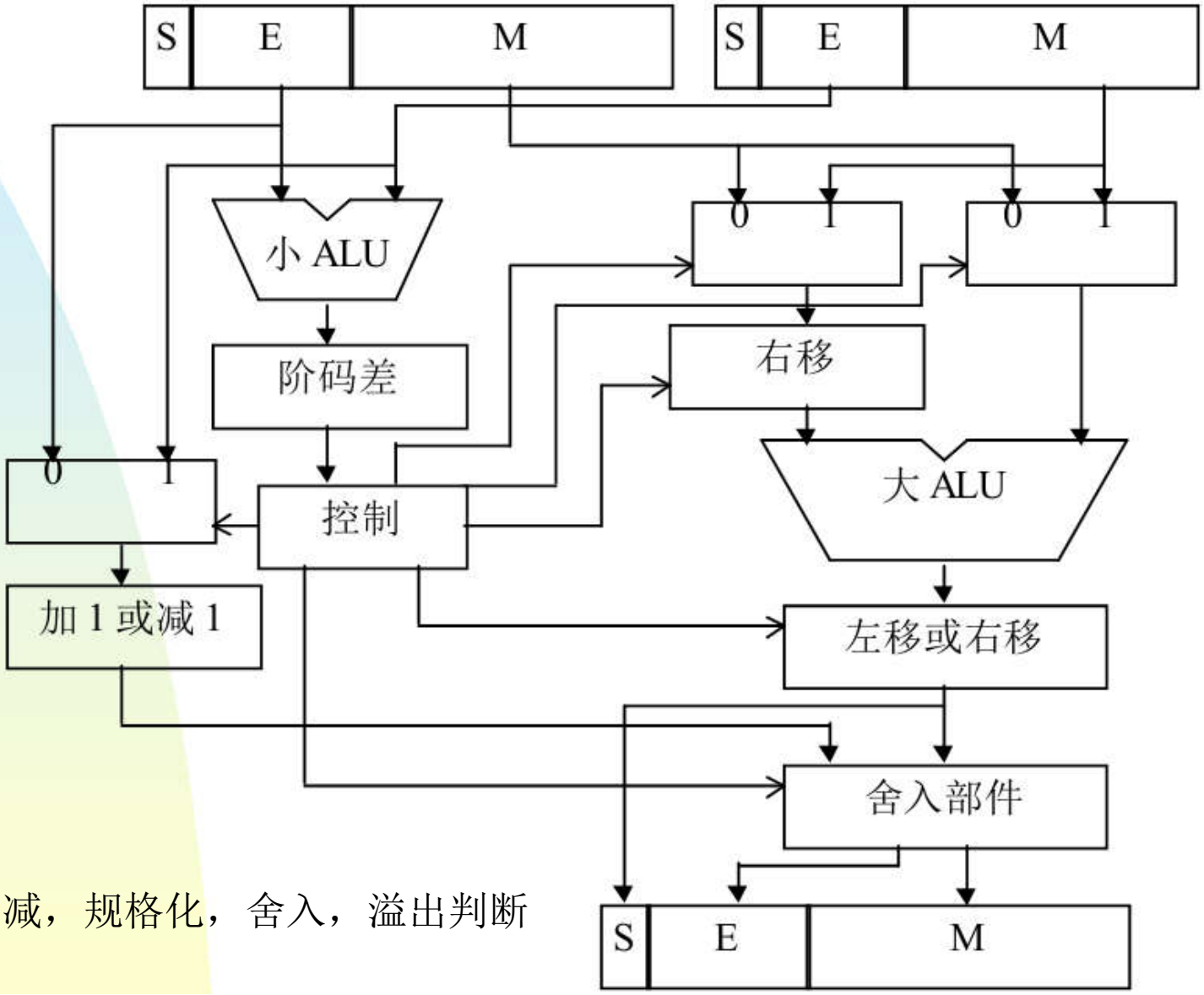
⑤溢出判断

经舍入处理后阶符为“11”，不溢出，故最终结果： $x-y= 2^{-011}\times(-0.100111)$

浮点数加减运算流程


开 始
求阶差Δj
保留大阶
对阶：小阶的尾数右移 j 位
尾数加（减）
判断尾数加（减）结果
全“0”
规格化数
溢出
未溢出
右规：尾数右移一位，大阶加 1。
左规：尾数左移 k 位，直到满足要求为止，大阶减 k。
舍入（0 舍 1 入）
尾数是否溢出
Y
N
右规：尾数右移一位，大阶加 1。
下溢
阶码是否溢出
上溢
置机器 0
正常
置溢出标志
结 束
溢出中断处理

浮点加减运算电路



对阶，尾数加减，规格化，舍入，溢出判断

浮点乘除法运算

- 设两浮点数 $x = S_x \cdot r^{j_x}$ $y = S_y \cdot r^{j_y}$

- 则 $x \cdot y = (S_x \cdot S_y) \cdot r^{j_x + j_y}$ $\frac{x}{y} = \frac{S_x}{S_y} \cdot r^{j_x - j_y}$

1. 阶码加减
2. 尾数乘除
3. 规格化
4. 舍入
5. 溢出判断

1. 阶码运算

- **若阶码用补码运算**，乘积的阶码为$[j_x]_{补}+[j_y]_{补}$，商的阶码为$[j_x]_{补}-[j_y]_{补}$。
- **若阶码用移码运算**，则

$$[j_x]_{移}=2^n+ j_x，-2^n\le j_x<2^n \text{ (n为整数的位数)}$$

$$[j_y]_{移}=2^n+ j_y，-2^n\le j_y<2^n \text{ (n为整数的位数)}$$

所以$[j_x]_{移}+[j_y]_{移}= 2^n+ j_x +2^n+ j_y = 2^n+ (2^n +(j_x + j_y))= 2^n+[j_x + j_y]_{移}$

可见，直接用移码求阶码和时，其最高位多加了一个2^n，要得到移码形式的结果，**必须减去2^n**。

1. 阶码运算(续)

由于同一个真值的移码和补码其数值部分完全相同，而符号位正好相反，即 $[j_y]_{补}=2^{n+1}+ j_y \pmod{2^{n+1}}$

- 因此如果求阶码和可用下式完成：

$$[j_x]_{移}+[j_y]_{补}= 2^n+ j_x +2^{n+1}+ j_y = 2^n+ [2^n +(j_x + j_y)] = [j_x + j_y]_{移} \pmod{2^{n+1}}$$

则直接可得移码形式。

- 同理，当作除法运算时，商的阶码用下式完成：

$$[j_x]_{移}+[-j_y]_{补} = [j_x - j_y]_{移}$$

阶码运算规则

- **进行移码加减运算：结果为移码**
  - **只需将移码表示的加数或减数的符号位取反(即变为补码)，然后进行运算，就可得阶和(或阶差)的移码。**
  - **双符号位移码**
    - 被加数或被减数：双符号位移码，符号位高位为“0”。
    - 加数或减数：双符号位补码，两位符号位保持一致。
- **溢出判断：双符号位移码**
  - **第一位指示溢出，第二位指示符号（注意：与变形补码不同！）**
    - 如果运算结果移码的最高符号位为0，即表明没溢出。此时:
      - 若低位符号位为1，表明结果为正；
      - 低位符号位为0，表示结果为负。
    - 如果运算结果移码的最高符号位为1，出现溢出。此时:
      - 若低位符号位为0，表示上溢；
      - 低位符号位为1，表示下溢。

阶码运算 - 溢出判断举例

- 设阶码取三位（不含符号位），

  当j_x=+101，j_y=+100时，有

  $[j_x]_{移}$=01,101， $[j_y]_{补}$=00,100

  则：

  $[j_x+j_y]_{移}=[j_x]_{移}+[j_y]_{补}$=01,101+00,100=10,001 结果上溢

  $[j_x-j_y]_{移}=[j_x]_{移}+[-j_y]_{补}$=01,101+11,100=01,001 结果为“+1”

2. 尾数运算

- **浮点乘法尾数运算**
  - **预处理：检测乘数或被乘数是否为0**
  - **相乘**
  - **规格化**
    - 溢出处理：根据阶码
  - **尾数截断：对于双倍字长结果，如只取一字，需截断**
    - 舍入处理
- **浮点除法尾数运算**
  - **预处理**
    - 检测被除数或除数是否为0
    - 比较被除数与除数的绝对值，保证是小数运算
      - 被除数≥除数，被除数右移
        - » 如果操作数是规格化数，则商必然是规格化数
  - **小数除法**

（1）浮点乘法尾数运算

- 预处理：检测两个尾数中是否有一个为0，若有一个为0，乘积必为0，不再作其他操作；如果两尾数均不为0，则可进行乘法运算。
- 相乘：两个浮点数的尾数相乘可以采用定点小数的任何一种乘法运算来完成。
- **规格化：**相乘结果可能要进行**左规**，左规时调整阶码后如果发生阶下溢，则作机器零处理；如果发生阶上溢，则作溢出处理。

- 尾数截断：尾数相乘会得到一个双倍字长的结果，若限定只取1倍字长，则乘积的若干低位将会丢失。
- 如何处理丢失的各位值，有两种办法：
  - 截断处理：无条件的丢掉正常尾数最低位之后的全部数值。
  - 舍入处理：按浮点加减运算讨论的舍入原则进行舍入处理。

（1）浮点乘法尾数运算

- **舍入处理**
  - 对于原码，采用0舍1入法时，不论其值是正数或负数，“舍”使数的绝对值变小，“入”使数的绝对值变大。
  - 对于补码，采用0舍1入法时，若丢失的位不是全0，对正数来说，“舍”、“入”的结果与原码正好相同；对负数来说，“舍”、“入”的结果与原码分析正好相反，即“舍”使绝对值变大，“入”使绝对值变小。
    - 为了使原码、补码舍入处理后的结果相同，对负数的补码可采用如下规则进行舍入处理。
      - ①当丢失的各位均为0时，不必舍入；
      - ②当丢失的各位数中的最高位为0时，且以下各位不全为0；或丢失的各位数中的最高位为1，且以下各位均为0时，则舍去被丢失的各位；
      - ③当丢失的各位数中的最高位为1，且以下各位又不全为0时；则在保留尾数的最末位加1修正。

（1）浮点乘法尾数运算

- 舍入操作实例

$[x]_{补}$舍入前	舍入后	对应的真值
1.01110000	1.0111（不舍不入）	-0.1001
1.01111000	1.0111（舍）	-0.1001
1.01110101	1.0111（舍）	-0.1001
1.01111100	1.1000（入）	-0.1000

- 对负数补码可采用如下规则进行舍入处理。
  - ①当丢失的各位均为0时，不必舍入；
  - ②当丢失的各位数中的最高位为0时，且以下各位不全为0；或丢失的各位数中的最高位为1，且以下各位均为0时，则舍去被丢失的各位；
  - ③当丢失的各位数中的最高位为1，且以下各位又不全为0时，则在保留尾数的最末位加1修正。

$[x]_{补}$舍入前	舍入后	对应的真值
1.10010000	1.1001（不舍不入）	-0.1001
1.10001000	1.1001（舍）	-0.1001
1.10001011	1.1001（入）	-0.1001
1.10000100	1.1000（舍）	-0.1000

浮点乘法运算举例

- 例：设机器数阶码取3位(不含阶符)，尾数取7位(不含数符)，要求阶码用移码运算，尾数用补码运算，最后结果保留1倍字长。
- 设$x=2^{-101}\times0.0110011)$，$y=2^{011}\times(-0.1110010)$

  求： $x\bullet y$。

- 解：$[x]_{补}$=11，011；00.0110011

  $[y]_{补}$=00，011；11.0001110

  ①阶码运算

  $[j_x]_{移}$=00，011, $[j_y]_{补}$=00，011

  $[j_x+j_y]_{移}=[j_x]_{移}+[j_y]_{补}$=00，011+00，011=00，110

  对应真值-2

浮点乘法运算举例（续）

- **②尾数相乘（采用Booth算法）其过程如下表所示。**

部分积	乘 数	y_{n+1}	说 明
00.0000000 00.0000000 + 11.1001101	1.0001110 0 1000111	0 0	→1位 +$[-S_x]_{补}$
11.1001101 11.1100110 11.1110011 11.1111001 + 00.0110011	0 10100011 01010001 10101000	1 1 1	→1位 →1位 →1位 +$[S_x]_{补}$
00.0101100 00.0010110 00.0001011 00.0000101 + 11.1001101	1010 01010100 00101010 10010101	0 0 0	→1位 →1位 →1位+ + $[-S_x]_{补}$
11.1010010	1001010		

- **相乘的结果为：$[S_x \cdot S_y]_{补}$=11.10100101001010**

浮点乘法运算举例（续）

即$[x \cdot y]_{补}$=11,110;11.10100101001010

- ③ 规格化。左规后$[x \cdot y]_{补}$=11,101;11.01001010010100

- ④舍入处理。尾数为负，按负数的补码的舍入规则，取1倍字长，丢失的7位为0010100,应“舍”。

  故最终的结果为：$[x \cdot y]_{补}$=11,101;11.0100101

  即：$xy = 2^{-011} \times (-0.1011011)$

（2）浮点除法尾数运算

- 步骤：
  - ① 检测被除数是否为0，若为0，则商为0；再检测除数是否为0，若为0，则商为无穷大，另作处理。若两数均不为0，则可进行除法运算。
  - ② 两浮点数尾数相除同样可采取定点小数的任何一种除法运算来完成。
    - 对已规格化的尾数，为了防止除法结果溢出，可先比较被除数和除数的绝对值，如果被除数的绝对值大于除数的绝对值，则先将被除数右移一位，其阶码加1，再作尾数相除。**此时所得结果必然是规格化的定点小数。**

浮点除法尾数运算—例题

- 例： $x=2^{101}\times0.1001, y=2^{011}\times(-0.1101)$，按补码浮点运算方法求x / y。

- 解：$[x]_{补}=00,101;00.1001$, $[y]_{补}=00,011;11.0011$，
  y的尾数 $[-S_y]_{补}=00.1101$

  ①阶码相减。

  $[j_x]_{补}-[j_y]_{补}=00,101-00,011=00,101+11,101=00,010$

  ②尾数相除（采用补码除法）。

浮点除法尾数运算—例题（续）

- ②尾数相除（采用补码除法）。

被除数（余数）	商	说 明
00.1001 + 11.0011		$[S_x]_{补}$与$[S_y]_{补}$异号，$+[S_y]_{补}$
11.1100 11.1000 + 00.1101	1 1	$[R]_{补}$与$[S_y]_{补}$同号，上商1 ←1位 $+[-S_y]_{补}$
00.0101 00.1010 + 11.0011	1 0 1 0	$[R]_{补}$与$[S_y]_{补}$异号，上商0 ←1位 $+[S_y]_{补}$
11.1101 11.1010 + 00.1101	1 0 1 1 0 1	$[R]_{补}$与$[S_y]_{补}$同号，上商1 ←1位 $+[-S_y]_{补}$
00.0111 + 00.1110	1 0 1 0 1 0 1 0 1	$[R]_{补}$与$[S_y]_{补}$异号，上商0 ←1位，末位商恒置1

结果为$[S_x / S_y]$=1.0101

③规格化。尾数相除结果已为规格化数。

所以$[x / y]_{补}$=00,010;11.0101，则$[x / y]=2^{010}\times(-0.1011)$

浮点乘除法运算小结

- 两浮点数相**乘**其乘积的阶码为相乘两数阶码之和,其尾数应为相乘两数的尾数之积。

  $[j_x + j_y]_{移} = [j_x]_{移} + [j_y]_{补}$

-

- 两个浮点数相**除**，商的阶码为被除数的阶码减去除数的阶码得到的差，尾数为被除数的尾数除以除数的尾数所得的商。

  $[j_x - j_y]_{移} = [j_x]_{移} + [-j_y]_{补}$

- 参加运算的两个数都为规格化浮点数，乘除运算都可能出现结果不满足规格化要求的问题，因此也必须进行**规格化、舍入和溢出判断**等操作。
  - 规格化时要修改阶码。

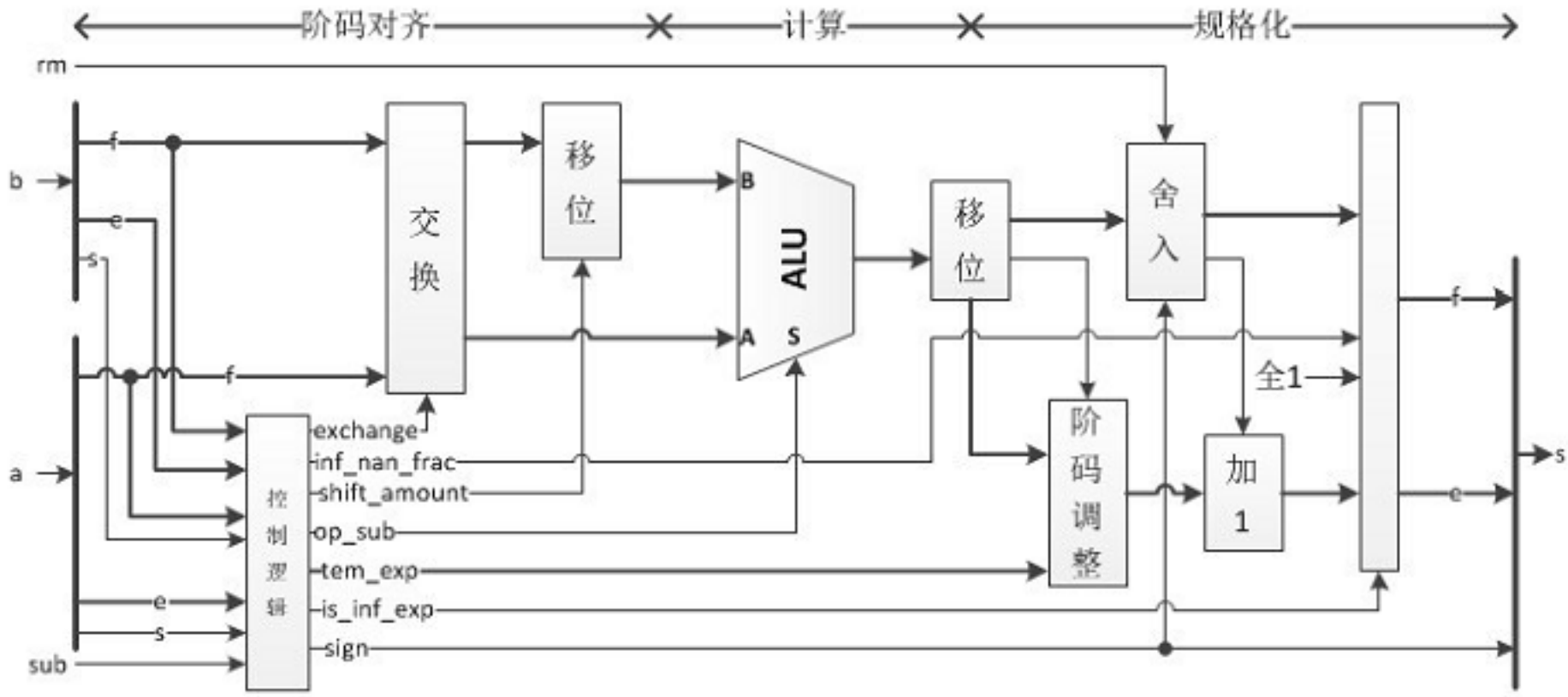
浮点运算所需的硬件配置

- 浮点运算器主要由两个定点运算部件组成：
  - 阶码运算部件：用来完成阶码加、减，以及控制对阶时小阶的尾数右移次数和规格化时对阶码的调整。
  - 尾数运算部件：用来完成尾数的四则运算以及判断尾数是否已规格化。
  - 此外，还需有判断运算结果是否溢出的电路等。

- 现代计算机可把浮点运算部件做成独立的选件，称为协处理器。
  - 浮点协处理器Inte l80287可与Intel 80286或80386微处理器配合处理浮点数的算术运算和多种函数计算。

- 也可用编程的办法来完成浮点运算，不过这会影响机器的运算速度。

浮点流水线


阶码对齐
计算
规格化
rm
b
a
sub
f
e
s
交换
移位
B
A
ALU
S
移位
舍入
全1
加1
阶码调整
控制逻辑
exchange
inf_nan_frac
shift_amount
op_sub
tem_exp
is_inf_exp
sign
f
e
s

浮点四则运算小结

- **加减运算：阶码与尾数都用变形补码表示**
  1. **对阶：小阶向大阶对齐**
  2. **尾数加减**
  3. **规格化：双符号位补码形式**
  4. **舍入：截断，0舍1入，恒置1**
  5. **溢出判断：双符号位补码**
- **乘除运算：阶可补可移（双符号），尾数为变形补码**
  1. **阶码加减**
  2. **尾数乘除（补码：Booth法，加减交替法）**
  3. **规格化：双符号位补码形式**
     - 乘法：左规
     - 除法：无需规格化（商自然是规格化数）
  4. **舍入：补码舍入规则与原码不同！**
  5. **溢出判断：按双符号补码或双符号移码**

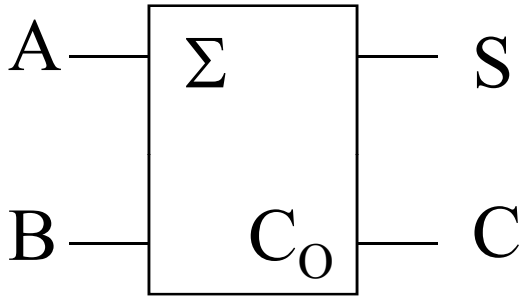
6.5 算术逻辑单元

ALU电路、快速进位链

半加器（half adder）

不考虑低位的进位，将两个一位二进制数A和B相加。

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



$$S = A\overline{B} + \overline{A}B = A \oplus B$$

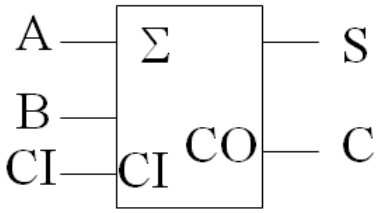
$$C_O = AB$$

全加器（full adder）

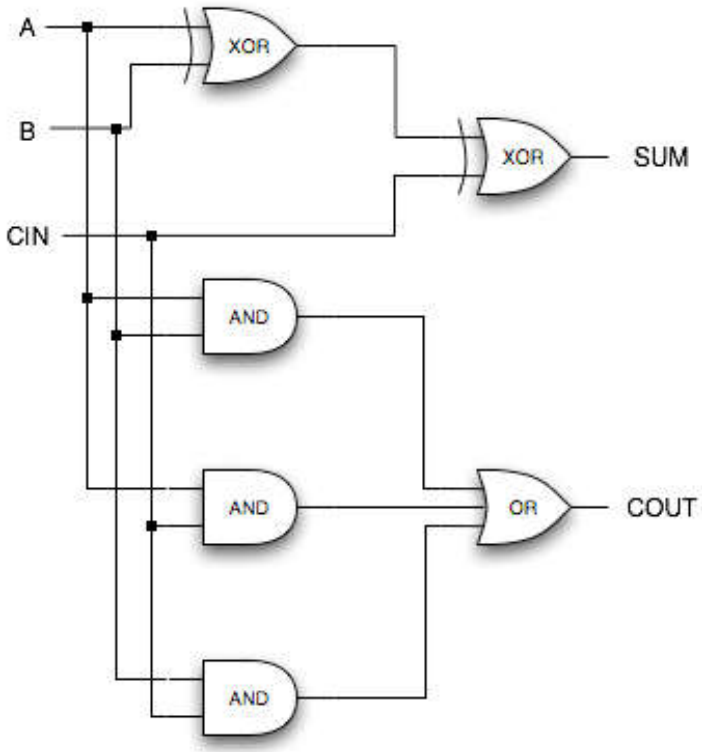
- 两个1位二进制数相加，考虑低位的进位。

$$\begin{cases} S = \overline{A}\overline{B}CI + \overline{A}B\overline{CI} + A\overline{B}\overline{CI} + ABCI \\ CO = \overline{A}BCI + A\overline{B}CI + AB\overline{CI} + ABCI \\ = AB + (A+B)CI \end{cases}$$

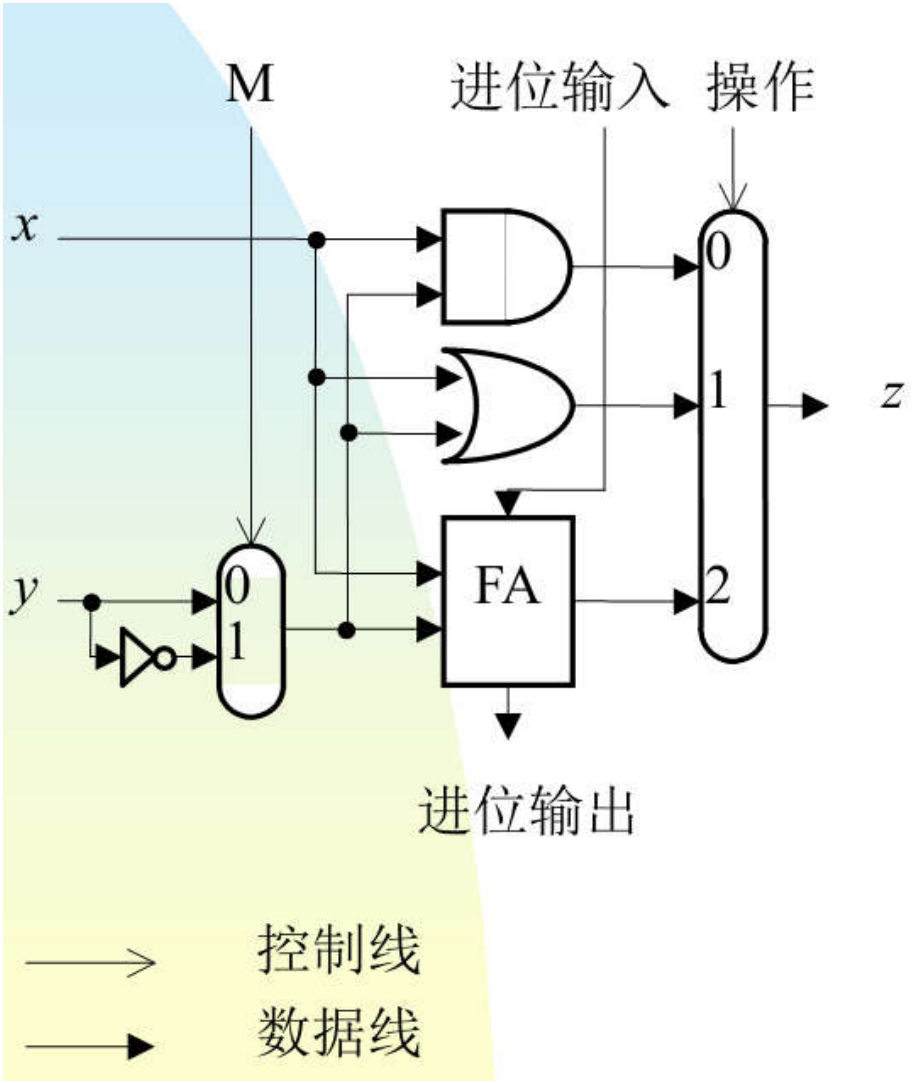
$$S = A_i \oplus B_i \oplus C_i$$

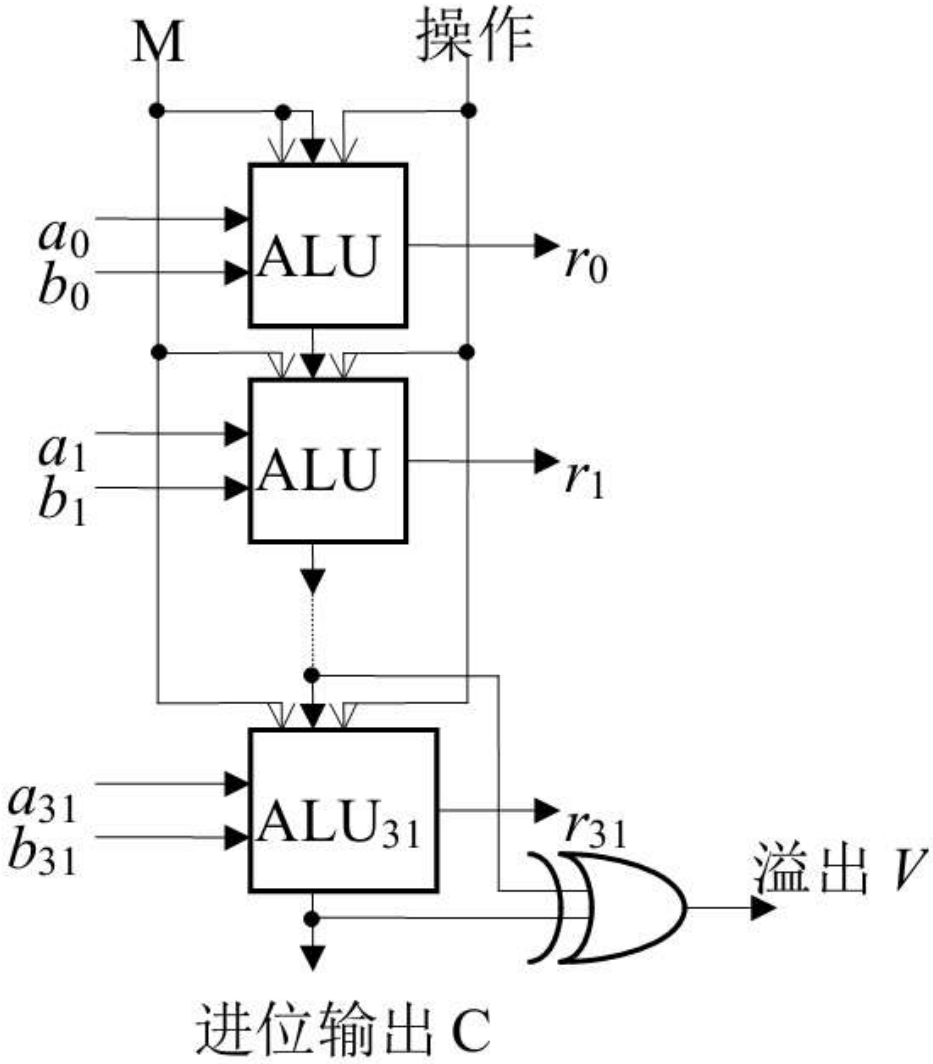


A_i	B_i	C_i	S_i	C_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



算术逻辑运算单元ALU：与/或/非/FA

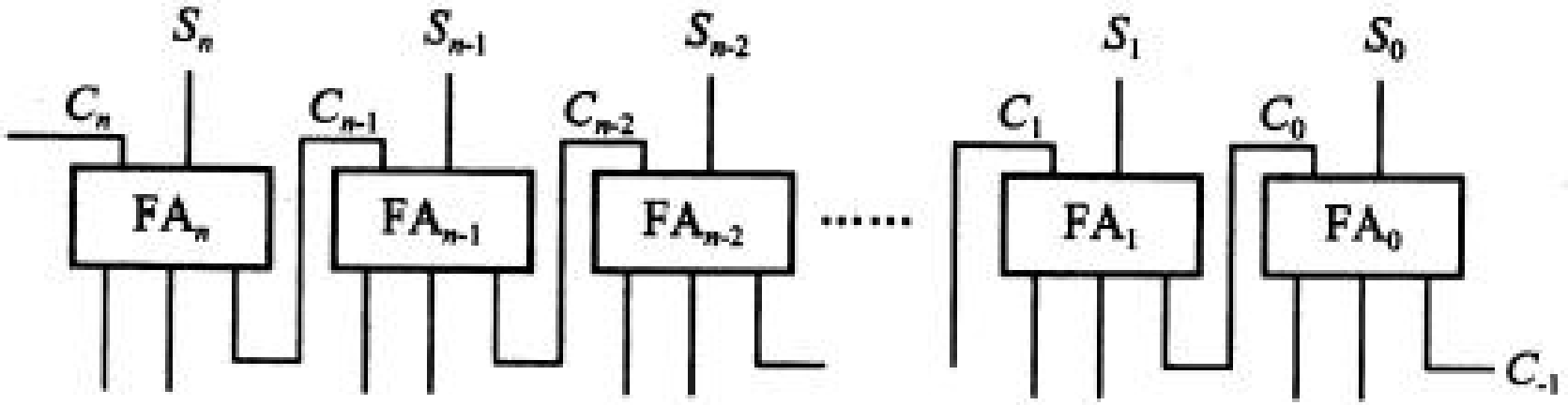




加法运算完成时间？

多位加法器

- *n*+1个全加器级联，就组成了一个*n*+1位的并行加法器。



- 串行进位链—行波进位加法器
  - 由于每位全加器的进位输出是高一位全加器的进位输入，因此当全加器有进位时，一级一级传递进位的过程，将会严重影响运算速度。

多位加法器

- **分析：由全加器的逻辑表达式可知，**

  和：$S_i = \overline{A}_i\overline{B}_iC_{i-1} + \overline{A}_iB_i\overline{C}_{i-1} + A_i\overline{B}_i\overline{C}_{i-1} + A_iB_iC_{i-1}$

  进位：$C_i = \overline{A}_iB_iC_{i-1} + A_i\overline{B}_iC_{i-1} + A_iB_i\overline{C}_{i-1} + A_iB_iC_{i-1}$

- **C_i进位由两部分组成：$= A_iB_i + (A_i + B_i)C_{i-1}$**
  - **本地进位：A_iB_i，可记作d_i，与低位无关；**
  - **传递进位：$(A_i + B_i)C_{i-1}$，与低位有关，称$(A_i + B_i)$为传递条件，记作t_i，则：**

$$C_i = d_i + t_iC_{i-1}$$

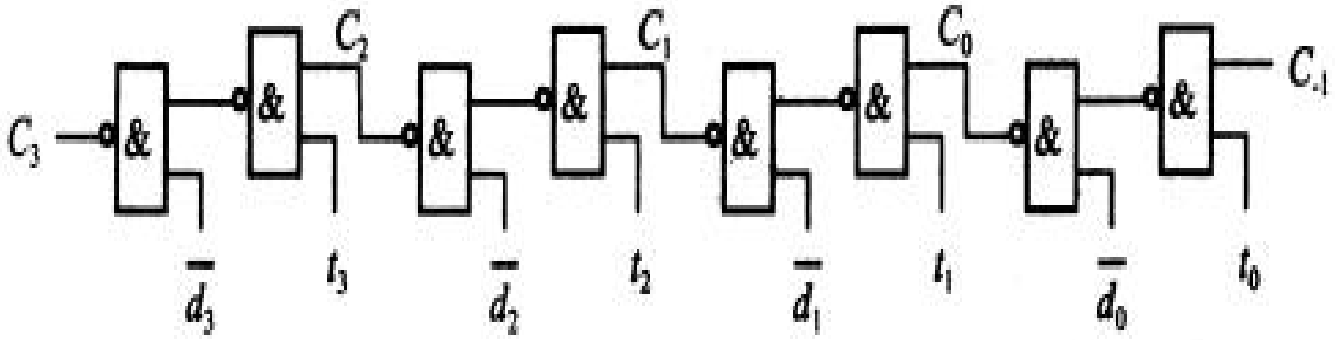
- **由C_i的组成可以将逐级传递进位的结构转换为以进位链的方式实现快速进位。**

串行进位链（行波进位加法器）

- **串行进位链**是指并行加法器中的进位信号采用串行传递。
- 以四位并行加法器为例，每一位的进位表达式可示为：

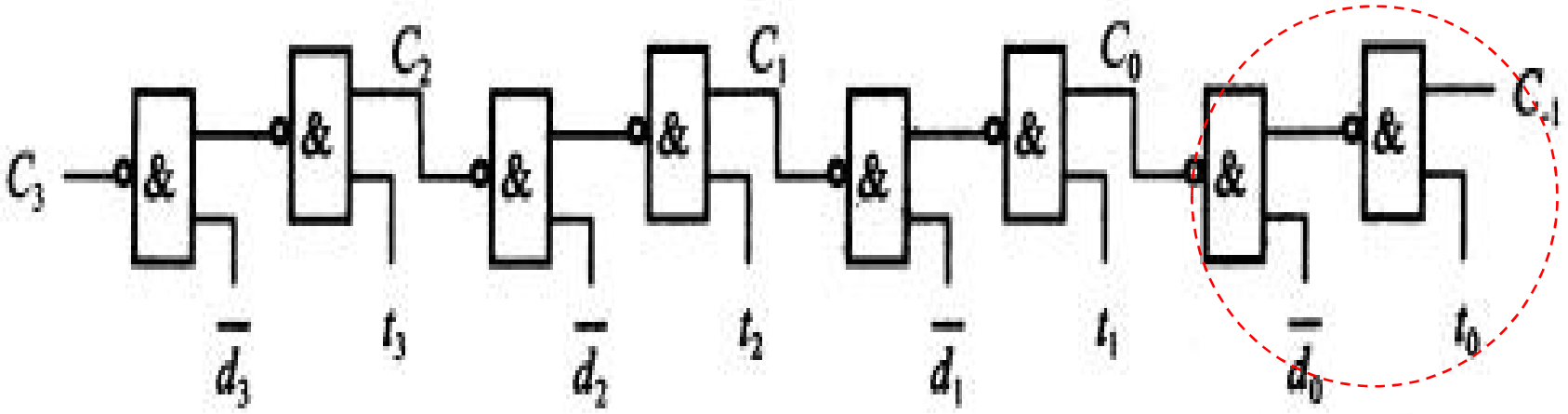
$$C_0 = d_0 + t_0 C_{-1}$$
$$C_1 = d_1 + t_1 C_0$$
$$C_2 = d_2 + t_2 C_1$$
$$C_3 = d_3 + t_3 C_2$$

- 由上式可见，采用与非逻辑电路可方便地实现进位传递，如下图。



串行进位链延迟时间分析

- 若设与非门的级延迟时间为t_y，那么当d_i、t_i形成后，共需$8t_y$便可产生最高位的进位。
- 实际上每增加一位全加器，进位时间就会增加$2t_y$。n位全加器的最长进位时间为$2nt_y$。



快速进位链

- **串行进位链——行波进位加法器：慢！**
  - **随着操作数位数的增加，产生进位的速度对运算时间的影响也越大。**

- **并行进位链——先行进位加法器：快！**
  - **又称“超前进位”、“跳跃进位”**
    - Carry-lookahead addition
  - **单重分组跳跃进位**
    - 即：单级分组
  - **双重分组跳跃进位**

并行进位链（超前进位加法器）

- **并行进位链**是指并行加法器中的进位信号是同时产生的，又称**先行进位**、**跳跃进位**等。

- 理想的并行进位链是n位全加器的n位进位同时产生，但实际实现有困难。

- 通常并行进位链有**单重分组**和**双重分组**两种实现方案。

单重分组跳跃进位

- **单重分组跳跃进位**：将M位全加器分成若干小组，小组内的进位同时产生，小组与小组之间采用串行进位。
  - 又称为“组内并行、组间串行”进位。
- 以四位并行加法器为例，对其进位表示式稍作变换，便可获得并行进位表达式：

$$
\begin{aligned}
C_0 &= d_0 + t_0 C_{-1} \\
C_1 &= d_1 + t_1 C_0 \\
C_2 &= d_2 + t_2 C_1 \\
C_3 &= d_3 + t_3 C_2
\end{aligned}
\Rightarrow
\begin{aligned}
C_0 &= d_0 + t_0 C_{-1} \\
C_1 &= d_1 + t_1 C_0 = d_1 + t_1 d_0 + t_1 t_0 C_{-1} \\
C_2 &= d_2 + t_2 C_1 = d_2 + t_2 d_1 + t_2 t_1 d_0 + t_2 t_1 t_0 C_{-1} \\
C_3 &= d_3 + t_3 C_2 = d_3 + t_3 d_2 + t_3 t_2 d_1 + t_3 t_2 t_1 d_0 + t_3 t_2 t_1 t_0 C_{-1}
\end{aligned}
$$

四位一组并行进位

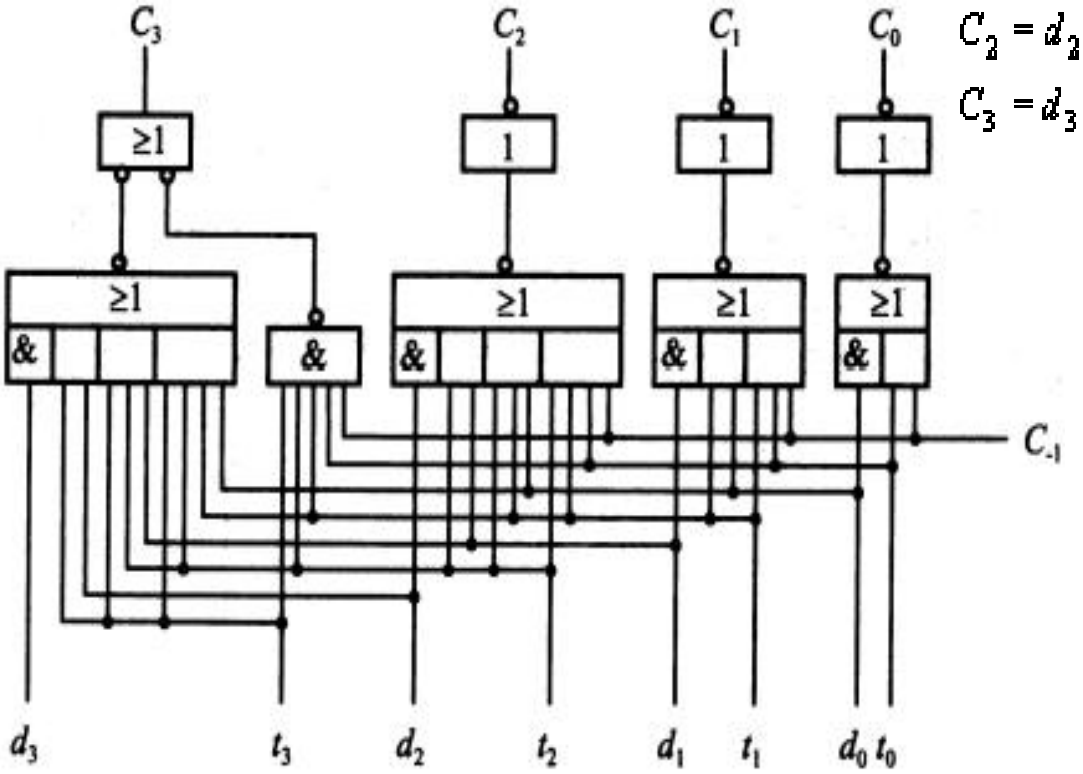
- 对应的逻辑图为：

$$C_0 = d_0 + t_0 C_{-1}$$

$$C_1 = d_1 + t_1 C_0 = d_1 + t_1 d_0 + t_1 t_0 C_{-1}$$

$$C_2 = d_2 + t_2 C_1 = d_2 + t_2 d_1 + t_2 t_1 d_0 + t_2 t_1 t_0 C_{-1}$$

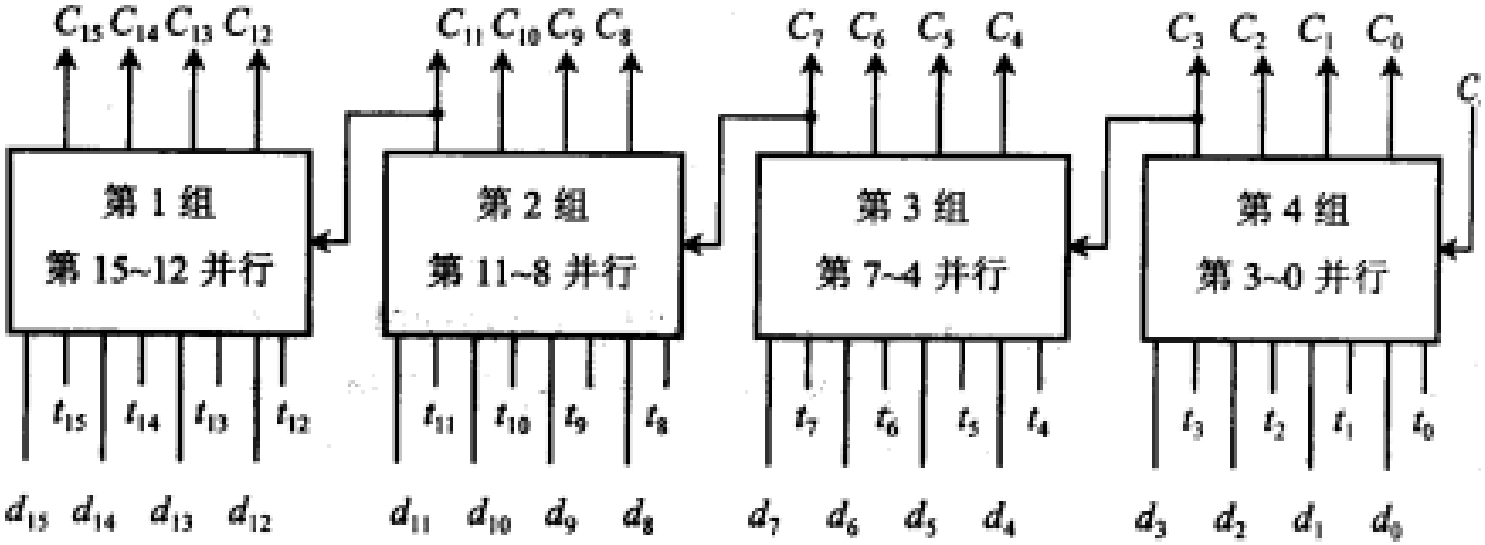
$$C_3 = d_3 + t_3 C_2 = d_3 + t_3 d_2 + t_3 t_2 d_1 + t_3 t_2 t_1 d_0 + t_3 t_2 t_1 t_0 C_{-1}$$



- 设“与或非门”的级延迟时间为1.5t_y，与非门的级延迟时间仍为1t_y，则d_i、t_i形成后，只需2.5t_y就可产生全部进位。

单重分组跳跃进位

- 如果将16位的全加器按四位一组分组，便可得单重分组跳跃进位链框图



- 在d_i、t_i形成后，经2.5t_y可产生C_3、C_2、C_1、C_0四个进位信息，经10t_y就可产生全部进位。
  - 如前所示，n=16的串行进位链的全部进位时间为32t_y，则16位全加器的单重分组方案进位时间仅约为串行进位链的三分之一。

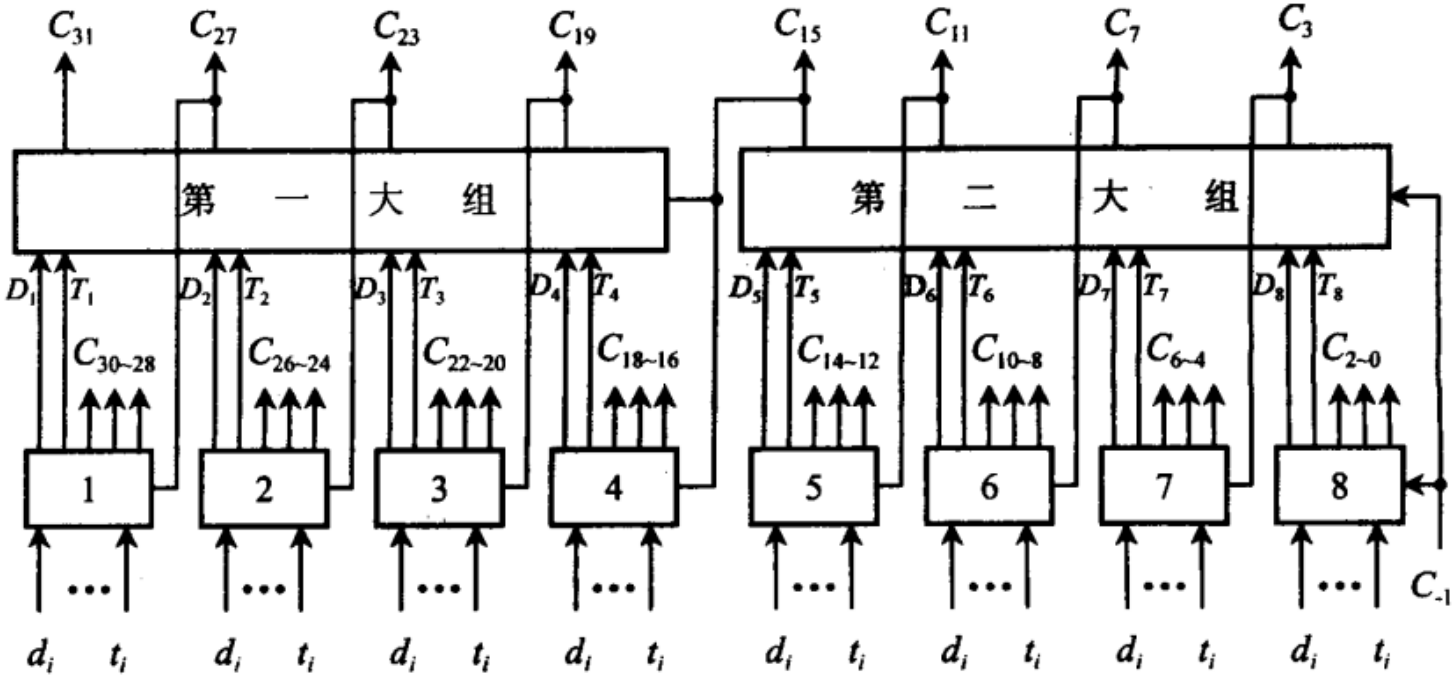
单重分组跳跃进位

- 缺点：但随着n的增大，其优势便很快减弱。
  - 例如，n=64，4位分组，共为16组：组间有16位串行进位，在d_i、t_i形成后，还需经16×2.5 = 40t_y才能产生全部进位，显然进位时间太长。

- 如果能使组间进位也同时产生，必然会更大地提高进位速度，这就是组内、组间均为并行进位的方案。

双重分组跳跃进位

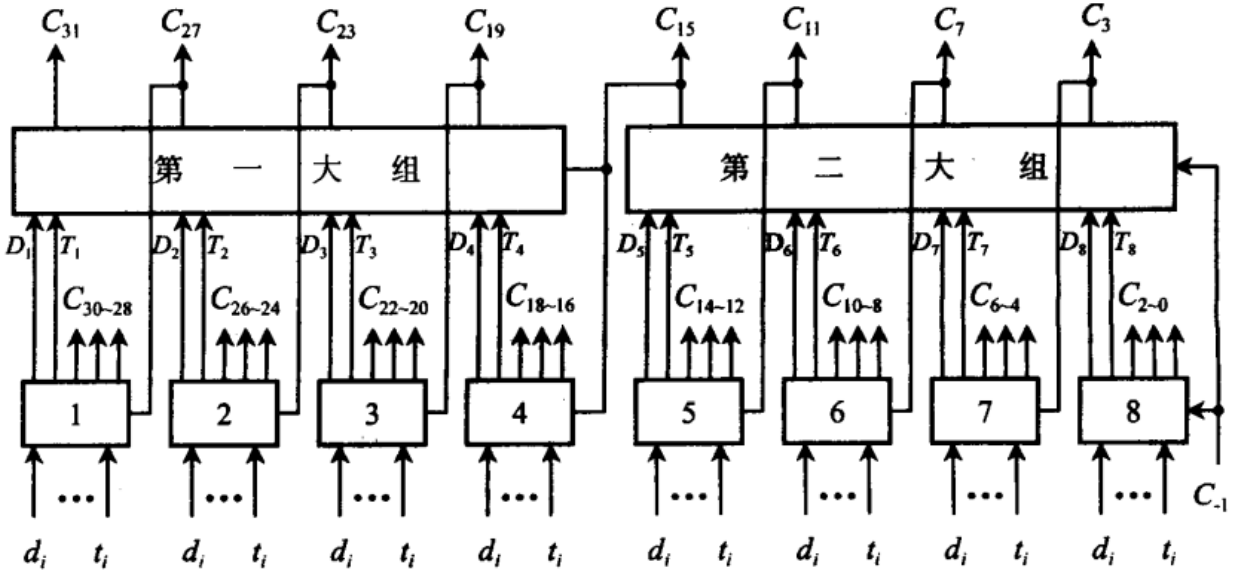
- 32位并行加法器双重分组跳跃进位链的框图



- 分两大组，每个大组内包含4个小组，第一大组内的4个小组的最高位进位C_{31}、C_{27}、C_{23}、C_{19}是同时产生的；第二大组内4个小组的最高位进位C_{15}、C_{11}、C_7、C_3也是同时产生的，而第二大组向第一大组的进位C_{15}采用串行进位方式。

双重分组跳跃进位

- 32位并行加法器双重分组跳跃进位链的框图

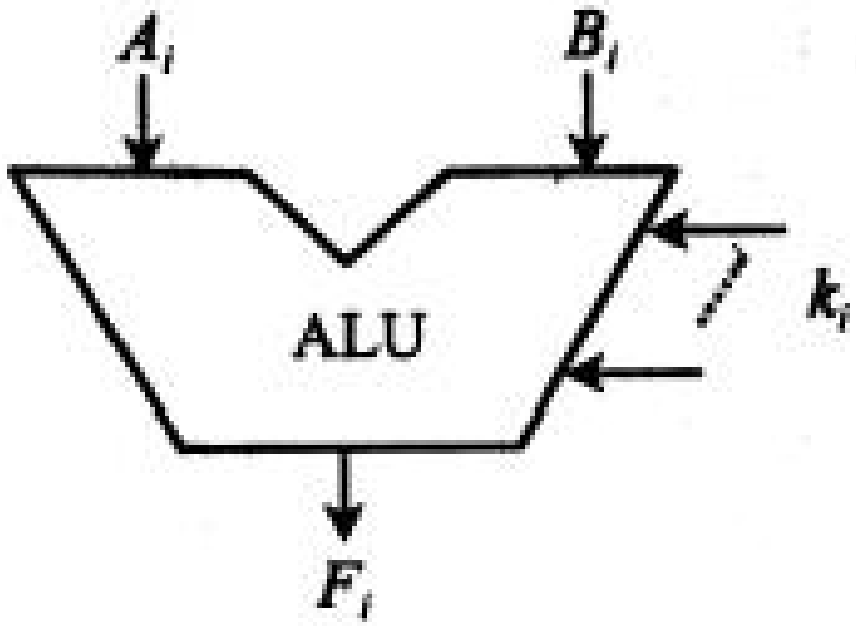


- 从D_i、T_i、及C_{-1}(外来进位)形成后开始，

① 经2.5T_y：形成C_2、C_1、C_0和全部D_i、T_i；

- 再经2.5T_y：形成第二大组内的四个进位C_{15}、C_{11}、C_7、C_3；
- 再经过2.5T_y：形成$C_{18\sim16}$、$C_{14\sim12}$、$C_{10\sim8}$、$C_{6\sim4}$、C_{31}、C_{27}、C_{23}、C_{19}；
- 再经过2.5T_y：形成$C_{30\sim28}$、$C_{26\sim24}$、$C_{22\sim20}$。

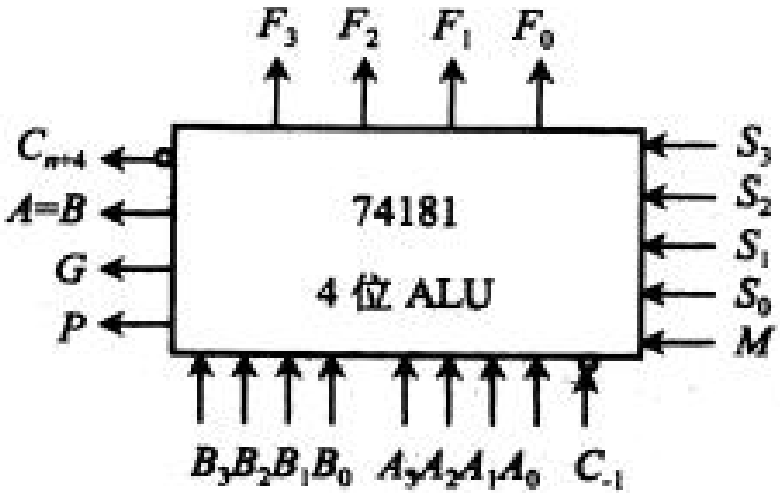
ALU电路符号



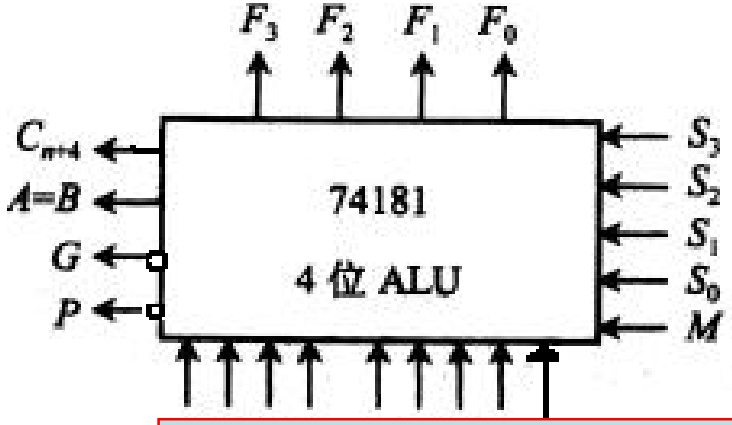
- A_i和B_i为输入变量；K_i为控制信号，K_i的不同取值可决定该电路作哪一种算术运算或哪一种逻辑运算；F_i是输出函数。

74181—ALU集成电路芯片

- 74181是能完成四位二进制代码的算逻运算部件，其外特性如下图所示。



正逻辑工作方式



负

P、G：先行进位
M：算术运算

- 正逻辑中，“1”用高电平表示，“0”用低电平表示，而负逻辑刚好相反。
- 正逻辑与负逻辑的关系为：正逻辑的“与”到负逻辑中变为“或”，即”+”、”·”互换。

74181—ALU集成电路芯片

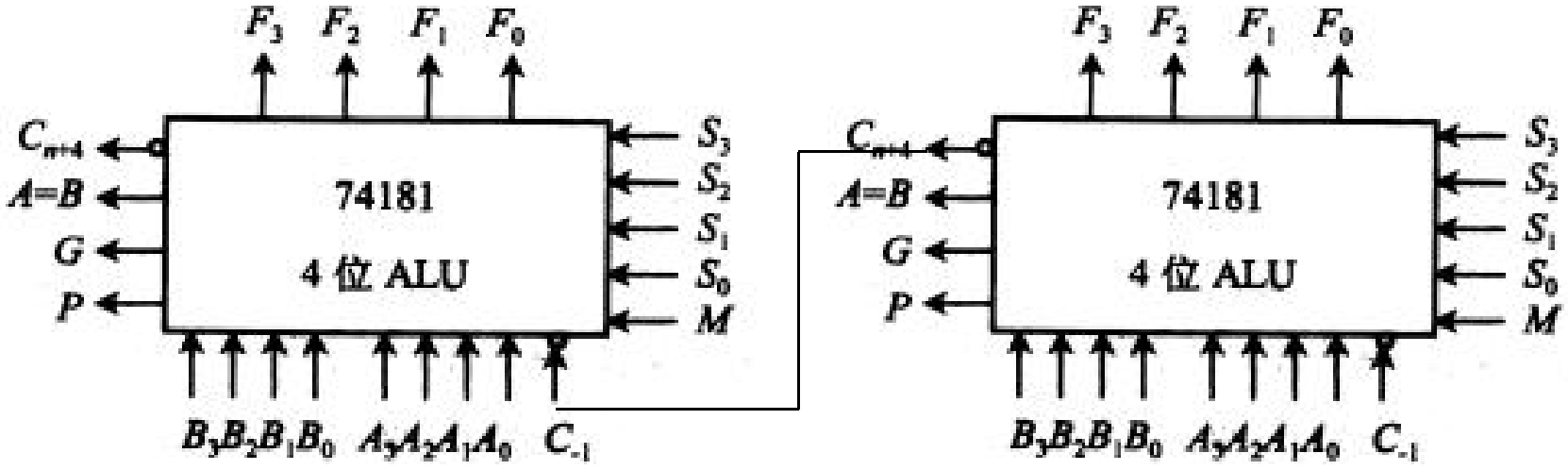
功能表				
工作方式选择输入 $S_3S_2S_1S_0$	负逻辑输入或输出		正逻辑输入或输出	
	逻辑运算 (M=1)	算术运算 (M=0)（C_{-1}=0）	逻辑运算 (M=1)	算术运算 (M=0)（C_{-1}=1）
0000	$\overline{A}$	A 减 1	$\overline{A}$	A
0001	$\overline{AB}$	AB 减 1	$\overline{A+B}$	$A+B$
0010	$\overline{A}+B$	$A\overline{B}$ 减 1	$\overline{A}B$	$A+\overline{B}$
0011	逻辑 1	减 1	逻辑 0	减 1
0100	$\overline{A+B}$	A 加（$A+\overline{B}$）	$\overline{AB}$	A 加 $A\overline{B}$
0101	$\overline{B}$	AB 加（$A+\overline{B}$）	$\overline{B}$	$(A+B)$ 加 $A\overline{B}$
0110	$\overline{A\oplus B}$	A 减 B 减 1	$A\oplus B$	A 减 B 减 1
0111	$A+\overline{B}$	$A+\overline{B}$	$A\overline{B}$	$A\overline{B}$ 减 1
1000	$\overline{A}B$	A 加（$A+B$）	$\overline{A}+B$	A 加 AB
1001	$A\oplus B$	A 加 B	$\overline{A\oplus B}$	A 加 B
1010	B	$A\overline{B}$ 加（$A+B$）	B	（$A+\overline{B}$）加 AB
1011	$A+B$	$A+B$	AB	AB 减 1
1100	逻辑 0	A 加 A^*	逻辑 1	A 加 A^*
1101	$A\overline{B}$	AB 加 A	$A+\overline{B}$	（$A+B$）加 A
1110	AB	$A\overline{B}$ 加 A	$A+B$	（$A+\overline{B}$）加 A
1111	A	A	A	A 减 1

（1） 1=高电平；0=低电平；（2）*表示每一位均移到下一个更高位，即 $A^*=2A$

注意：ALU为组合逻辑电路，因此实际应用ALU时，其输入端口A和B必须与锁存器相连，而且在运算的过程中锁存器的内容是不变的。其输出也必须送至寄存器中保存。

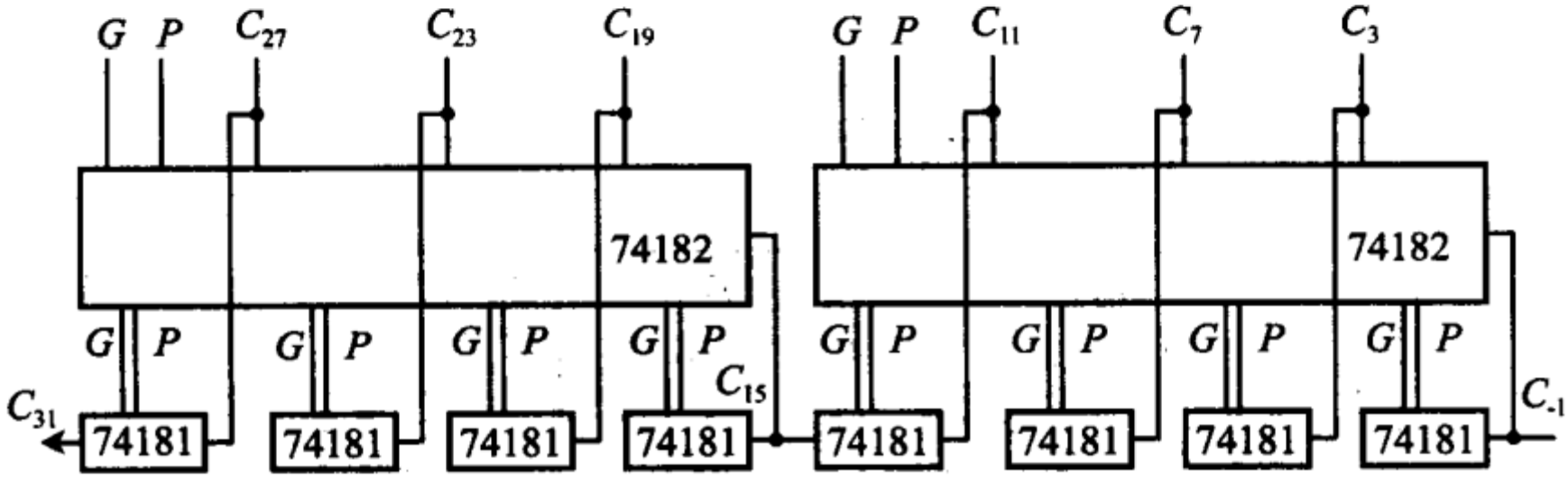
多位运算器

- 74181芯片是4位ALU电路，其四位进位是同时产生的，多片74181级联就犹如本节介绍的单重分组跳跃进位，即组内(74181片内)并行，组间(74181片间)串行。



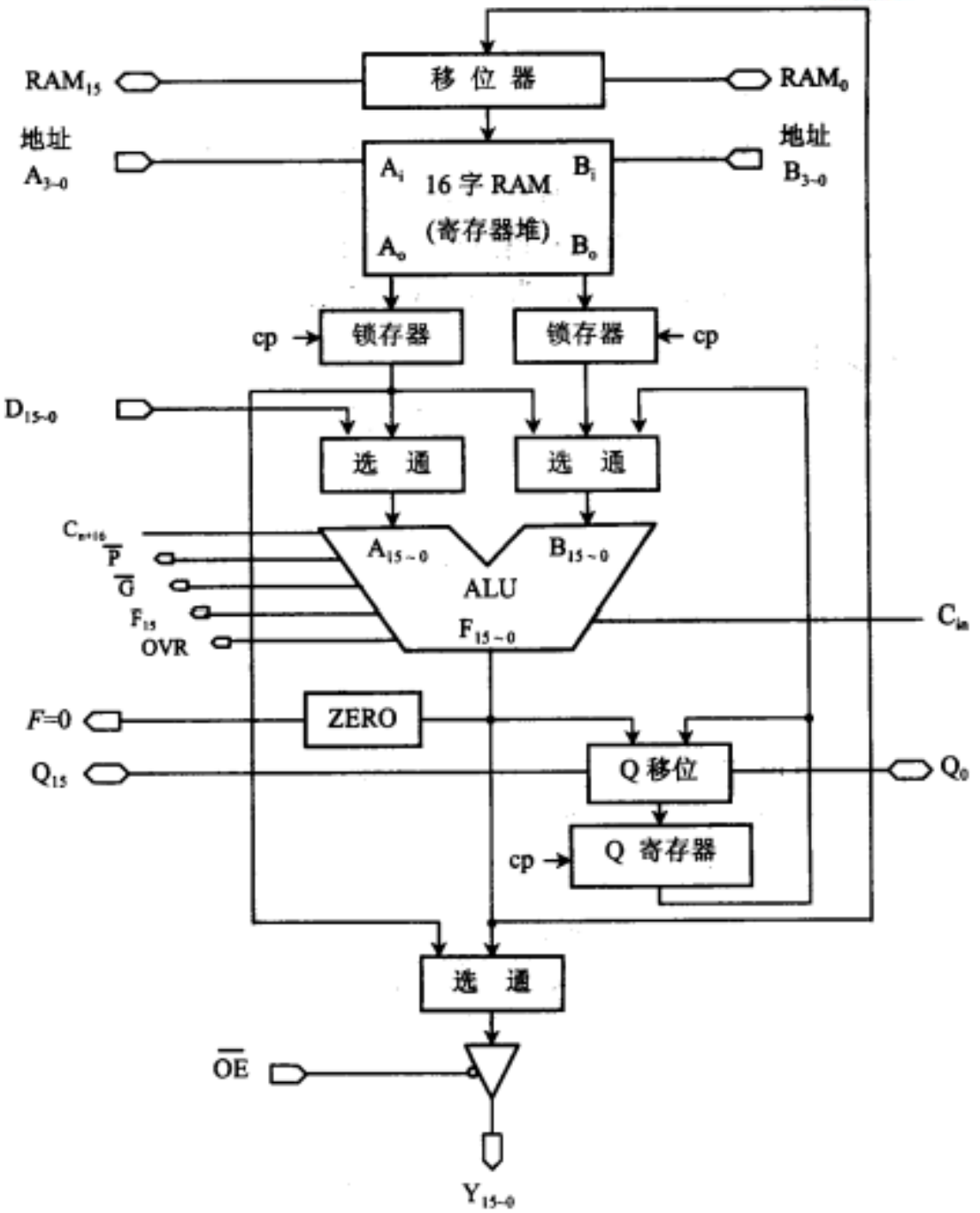
32位ALU电路

- 74182为先行进位部件，将74181与74182芯片配合，就可组成双重分组跳跃进位链。
  - 两片74182和8片74181组成32位ALU电路。



29C101芯片

- 将寄存器和ALU集成导一个芯片内。



本章小结

1. 数据的表示方法和转换
2. 无符号数和有符号数
3. 数的定点表示和浮点表示
4. 定点运算
5. 浮点四则运算
6. 算术逻辑单元ALU

作业

- 6.6、6.7、 6.12、 6.14、 6.20（1）、6.21（2）、 6.23、 6.29（1）、 6.30（2）。
- 报告(选)：编写一个完成原码、补码四则运算（定点、浮点）的仿真程序。
  - 输入：x，y
  - 输出：按流程图，演示运算每个步骤的结果。

休息是为了走更远的路！