



4.1 基本算术运算的实现

4.2 定点加法运算

4.3 带符号数的移位和舍入操作

4.4 定点乘法运算


4.5 定点除法运算

4.6 规格化浮点运算

4.7 十进制整数的加法运算

4.8 逻辑运算与实现

4.9 运算器的基本组成与实例



4.6 规格化浮点运算

4.6.1 浮点加减运算

设两个非0的规格化浮点数分别为

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

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

$$A \pm B = (M_A, E_A) \pm (M_B, E_B) =$$

$$(M_A \pm M_B \times 2^{-(E_A - E_B)}, E_A) \quad E_A > E_B$$

$$(M_A \times 2^{-(E_B - E_A)} \pm M_B, E_B) \quad E_A < E_B$$

4.6 规格化浮点运算

1. 浮点数加减运算步骤

(1) 对阶

两个浮点数相加或相减，首先要把小数点的位置对齐，而浮点数的小数点的实际位置取决于阶码的大小。因此，对齐两数的小数点，就是使两数的阶码相等，这个过程称为对阶。

要对阶，首先应求出两数阶码 E_A 和 E_B 之差，即：

$$\Delta E = E_A - E_B$$

4.6 规格化浮点运算

若 $\Delta E = 0$ ，表示两数阶码相等，即 $E_A = E_B$ ；若 $\Delta E > 0$ ，表示 $E_A > E_B$ ；若 $\Delta E < 0$ ，表示 $E_A < E_B$ 。

当 $E_A \neq E_B$ 时，要通过尾数的移位来改变 E_A 或 E_B ，使 $E_A = E_B$ 相等。

对阶的规则是：**小阶向大阶看齐**。采用这一规则的原因是当阶码小的数的尾数右移并相应增加阶码时，舍去的仅是尾数低位部分，误差比较小。要使小阶的阶码增大，则相应的尾数右移，直到两数的阶码相等为止。对于 $r=2$ ，**每右移一位，阶码加1**。

4.6 规格化浮点运算

$E_A = E_B$ ，不须对阶。

$E_A > E_B$ ，则 M_B 右移。每右移一位， $E_B + 1 \rightarrow E_B$ ，直至 $E_A = E_B$ 为止。

$E_A < E_B$ ，则 M_A 右移。每右移一位， $E_A + 1 \rightarrow E_A$ ，直至 $E_A = E_B$ 为止。

尾数右移后，应对尾数进行舍入。

(2) 尾数加/减

对阶之后，就可以进行尾数加/减，即

$$M_A \pm M_B \rightarrow M_C$$

4.6 规格化浮点运算

(3) 尾数结果规格化

尾数加/减运算之后得到的数可能不是规格化数，为了增加有效数字的位数，提高运算精度，必须进行结果规格化操作。

规格化的尾数M应满足：

$$1/2 \leq |M| < 1$$

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：

4.6 规格化浮点运算

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。

4.6 规格化浮点运算

尾数每左移一位，阶码相应减1 ($E_C - 1 \rightarrow E_C$)，直至成为规格化数为止。（左规可能需进行多次）只要满足下列条件：

$$\text{左规} = \overline{C_{s1}} \overline{C_{s2}} \overline{C_1} + C_{s1} C_{s2} C_1$$

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 ($E_C + 1 \rightarrow E_C$)。（右规最多进行一次）右规的条件如下：

$$\text{右规} = C_{s1} \oplus C_{s2}$$

4.6 规格化浮点运算

(4) 舍入

由于受到硬件的限制，在对阶和右规处理之后有可能将尾数的低位丢失，这会引起一些误差。舍入方法已在前面做过介绍，这里不再赘述。

4.6 规格化浮点运算

(5) 溢出判断

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。

4.6 规格化浮点运算

2. 浮点数加减运算举例

有两浮点数为

$$A=0.101110 \times 2^{-01}$$

$$B=-(0.101011) \times 2^{-10}$$

假设这两数的格式：阶码4位，用移码表示（偏置值为 2^3 ）；尾数8位，用补码表示，包含一位符号位，即



$$[A]_{\text{浮}} = 0111; 0.1011100$$

$$[B]_{\text{浮}} = 0110; 1.0101010$$

4.6 规格化浮点运算

(1) 对阶

求阶差: $\Delta E = E_A - E_B = -1 - (-2) = 1$

$\Delta E = 1$, 表示 $E_A > E_B$ 。按对阶规则, 将 M_B 右移一位, 其阶码加1, 得:

$[B]_{\text{浮}}' = 0111; 1.1010101$

(2) 尾数求和

$$\begin{array}{r} 00.1011100 \\ + 11.1010101 \\ \hline 00.0110001 \end{array}$$

4.6 规格化浮点运算

(3) 尾数结果规格化

由于结果的尾数是非规格化的数，故应左规。尾数每左移一位，阶码减1，直至尾数成为规格化数为止。

$$[A+B]_{\text{尾补}} = 00.0110001$$

$$[A+B]_{\text{尾补}}' = 00.1100010 \times 2^{-01}$$

最后结果为

$$[A+B]_{\text{浮}}' = 0110; 0.1100010$$

$$\therefore A+B = (0.110001) \times 2^{-10}$$

4.6 规格化浮点运算

4.6.2 浮点乘除运算

设两个非0的规格化浮点数分别为

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

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

则浮点乘法和除法为

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

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

4.6 规格化浮点运算

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 。

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

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

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

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。

4.6 规格化浮点运算

(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。

4.6 规格化浮点运算

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。



4.1 基本算术运算的实现

4.2 定点加法运算

4.3 带符号数的移位和舍入操作

4.4 定点乘法运算


4.5 定点除法运算

4.6 规格化浮点运算

4.7 十进制整数的加法运算

4.8 逻辑运算与实现

4.9 运算器的基本组成与实例



4.7 十进制整数的加减运算

4.7.1 一位十进制加法运算

1.8421码加法运算

因为一位8421码用四位二进制数表示，所以8421码十位数的“1”是个位数的进位。按四位二进制数而言，这个进位的值是16，而不是8421码的10。因此，必须+6校正，才能使该进位正确。8421码的加法规则：

- ①两个8421码相加时，“逢二进一”；
- ②当和 ≤ 9 ，无需校正；
- ③当和 > 9 ，则+6校正；
- ④在做+6校正的同时，将产生向上一位的进位。

4.7 十进制整数的加减运算

| 十进制数 | 8421码 C4S4S3S2S1 | 校正前的二进制数 C4'S4'S3'S2'S1' | 校正与否 |
|-------------|-----------------------|-----------------------------|------|
| 0 9 | 0 0000 0 1001 | 0 0000 0 1001 | 不校正 |
| 10 | 1 0000 | 0 1010 | +6校正 |
| 11 | 1 0001 | 0 1011 | |
| 12 | 1 0010 | 0 1100 | |
| 13 | 1 0011 | 0 1101 | |
| 14 | 1 0100 | 0 1110 | |
| 15 | 1 0101 | 0 1111 | |
| 16 | 1 0110 | 1 0000 | |
| 17 | 1 0111 | 1 0001 | |
| 18 | 1 1000 | 1 0010 | |
| 19 | 1 1001 | 1 0011 | |

+6校正函数 = $C4' + S4'S3' + S4'S2'$

4.7 十进制整数的加减运算

2. 余3码加法运算

十进制余3码加法规则：

- ①两个余3码相加，“逢二进一”；
- ②若其和没有进位，则减3（即+1101）校正；
- ③若其和有进位，则加3（即+0011）校正。

4.7 十进制整数的加减运算

| 十进制数 | 余3码 C4S4S3S2S1 | 校正前的二进制数 C4'S4'S3'S2'S1' | 校正与否 |
|------|-------------------|-----------------------------|------|
| 0 | 0 0011 | 0 0110 | -3校正 |
| 1 | 0 0100 | 0 0111 | |
| | | | |
| 8 | 0 1011 | 0 1110 | |
| 9 | 0 1100 | 0 1111 | |
| 10 | 1 0011 | 1 0000 | +3校正 |
| 11 | 1 0100 | 1 0001 | |
| | | | |
| 18 | 1 1011 | 1 1000 | |
| 19 | 1 1100 | 1 1001 | |

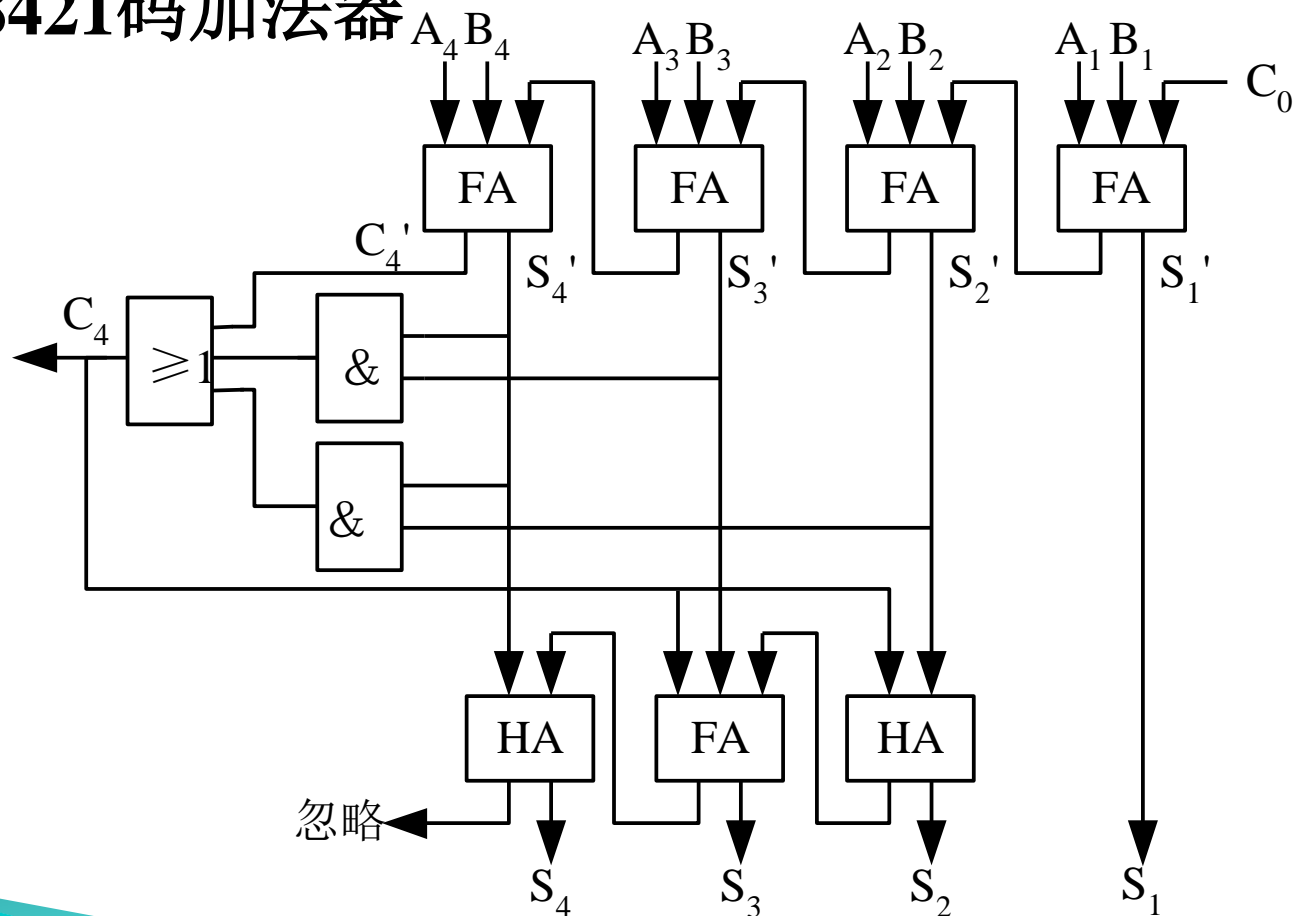
-3校正函数= $\overline{C4'}$

+3校正函数= $C4'$

4.7 十进制整数的加减运算

4.7.2 十进制加法器

1. 一位8421码加法器





4.1 基本算术运算的实现

4.2 定点加法运算

4.3 带符号数的移位和舍入操作

4.4 定点乘法运算


4.5 定点除法运算

4.6 规格化浮点运算

4.7 十进制整数的加法运算

4.8 逻辑运算与实现

4.9 运算器的基本组成与实例



4.8 逻辑运算与实现

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。

4.8 逻辑运算与实现

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。



4.1 基本算术运算的实现

4.2 定点加法运算

4.3 带符号数的移位和舍入操作

4.4 定点乘法运算

4.5 定点除法运算

4.6 规格化浮点运算

4.7 十进制整数的加法运算

4.8 逻辑运算与实现

4.9 运算器的基本组成与实例



4.9 运算器的基本组成与实例

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。

4.9 运算器的基本组成与实例

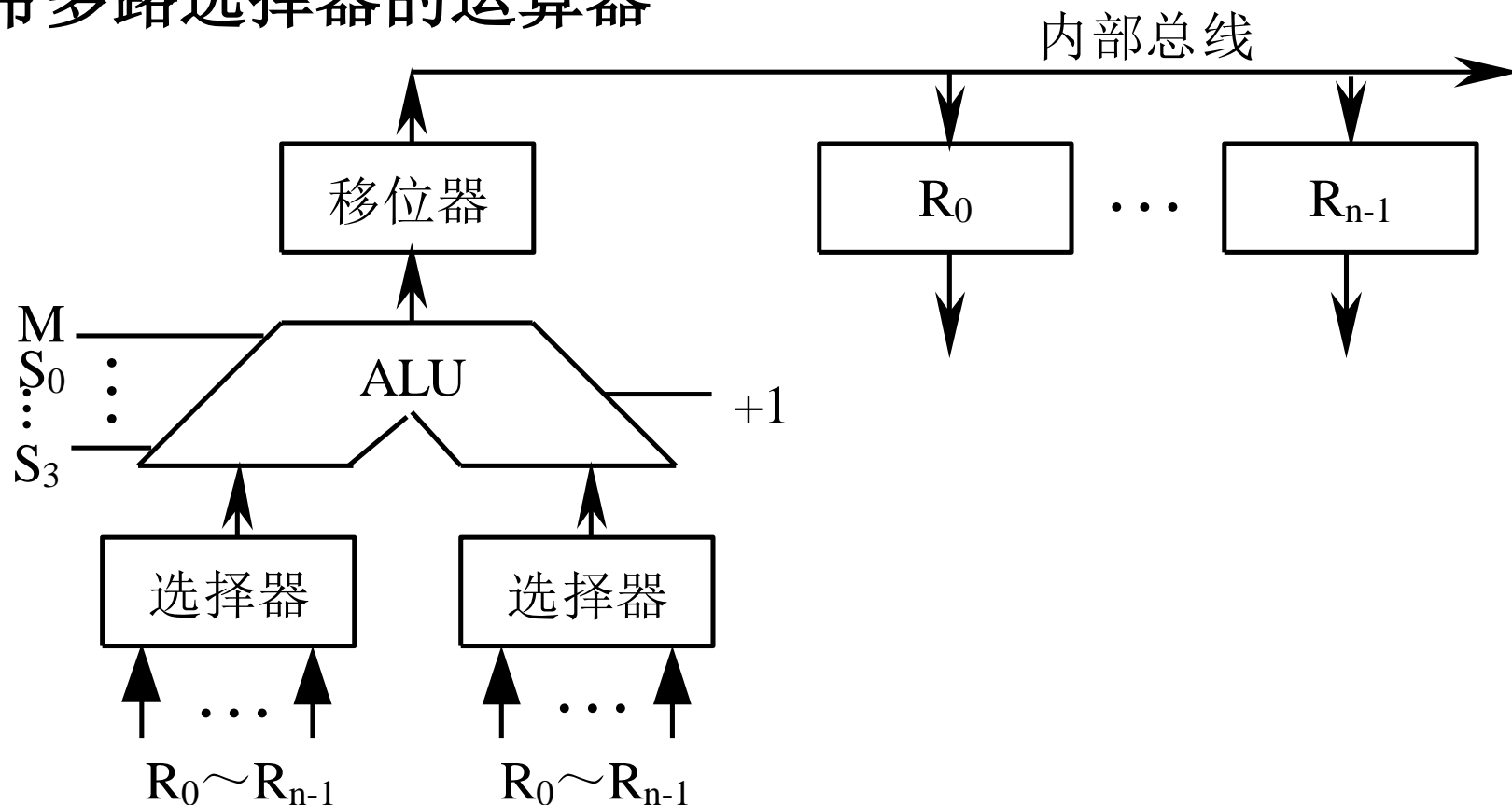
5.9.1 运算器结构

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。

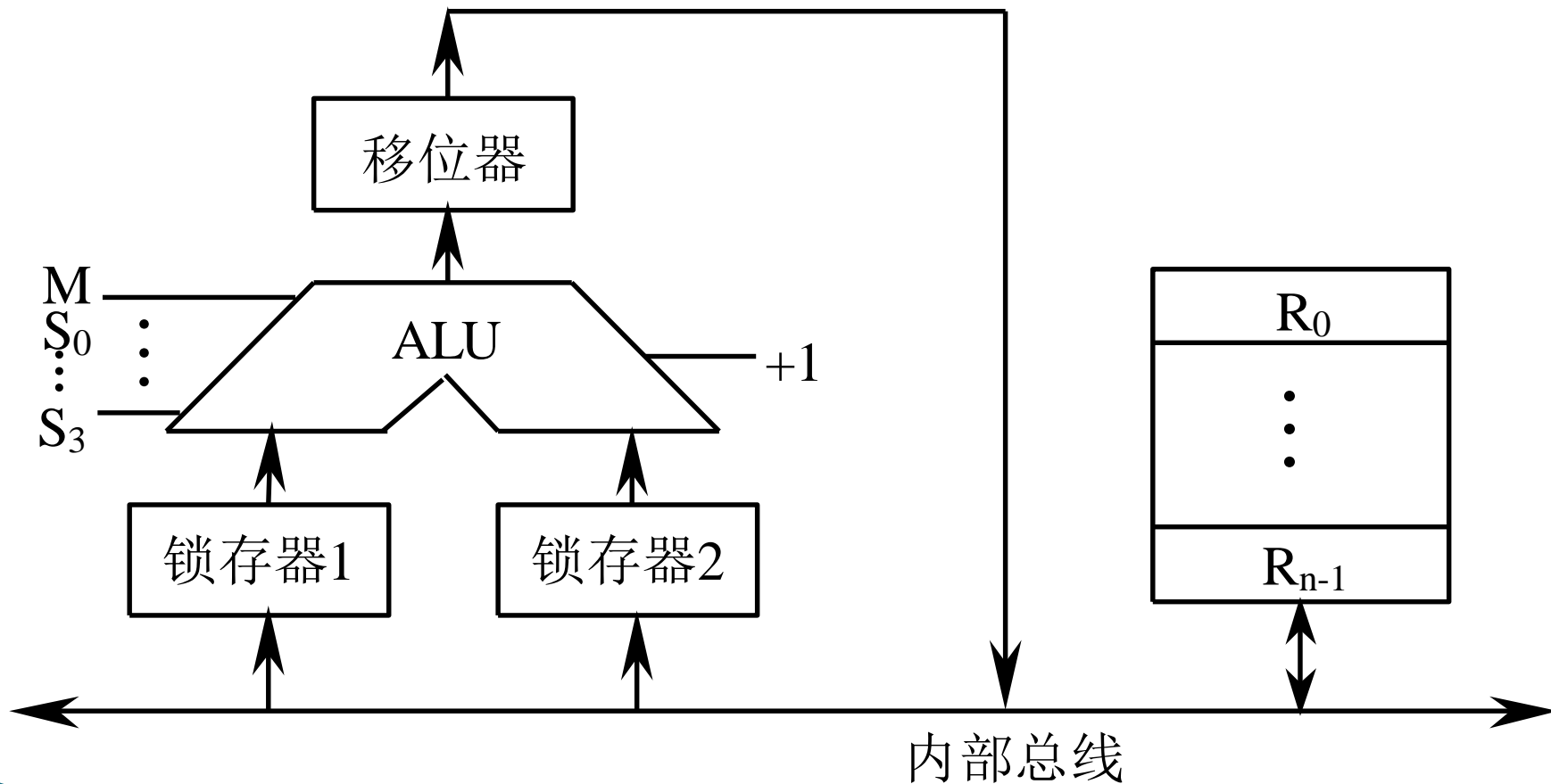
4.9 运算器的基本组成与实例

(1) 带多路选择器的运算器



4.9 运算器的基本组成与实例

(2) 带输入锁存器的运算器



4.9 运算器的基本组成与实例

2.运算器的内部总线结构

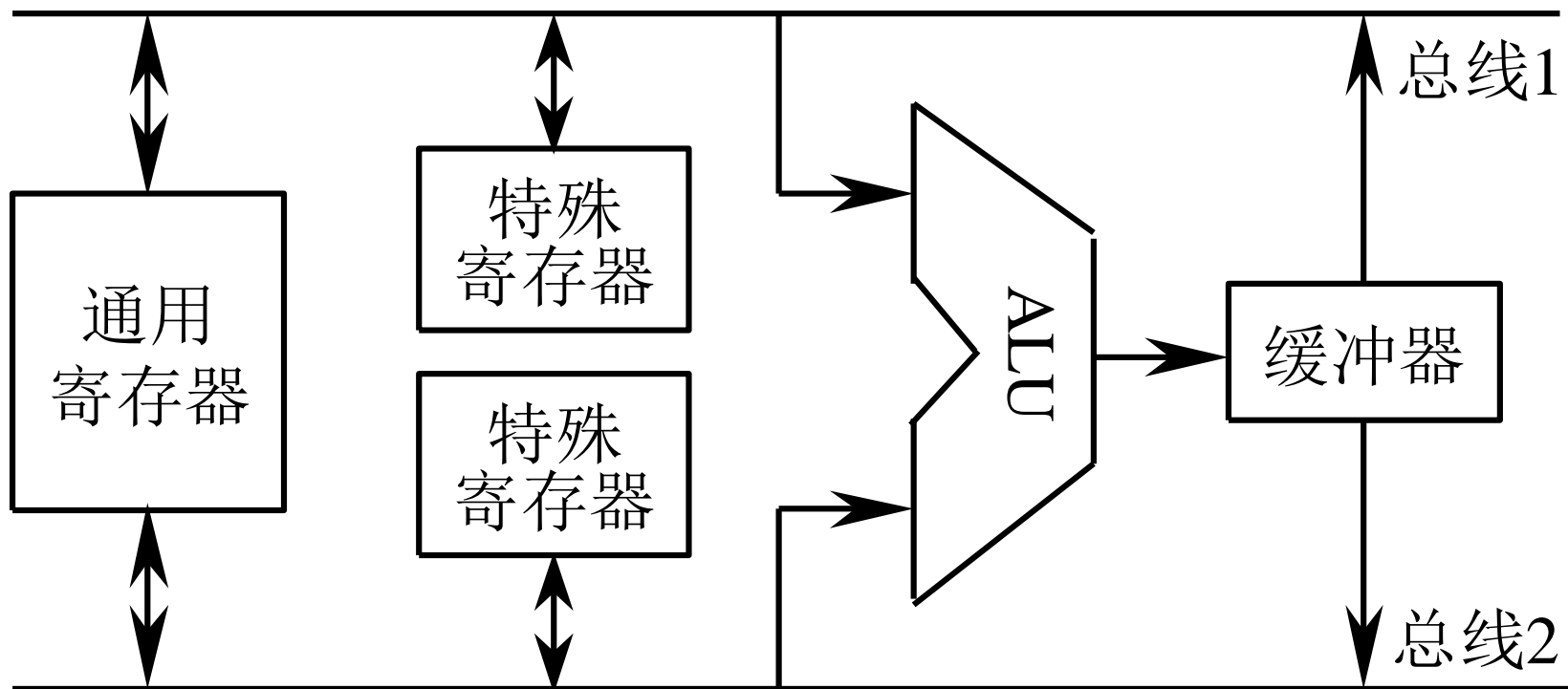
(1)单总线结构运算器

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。

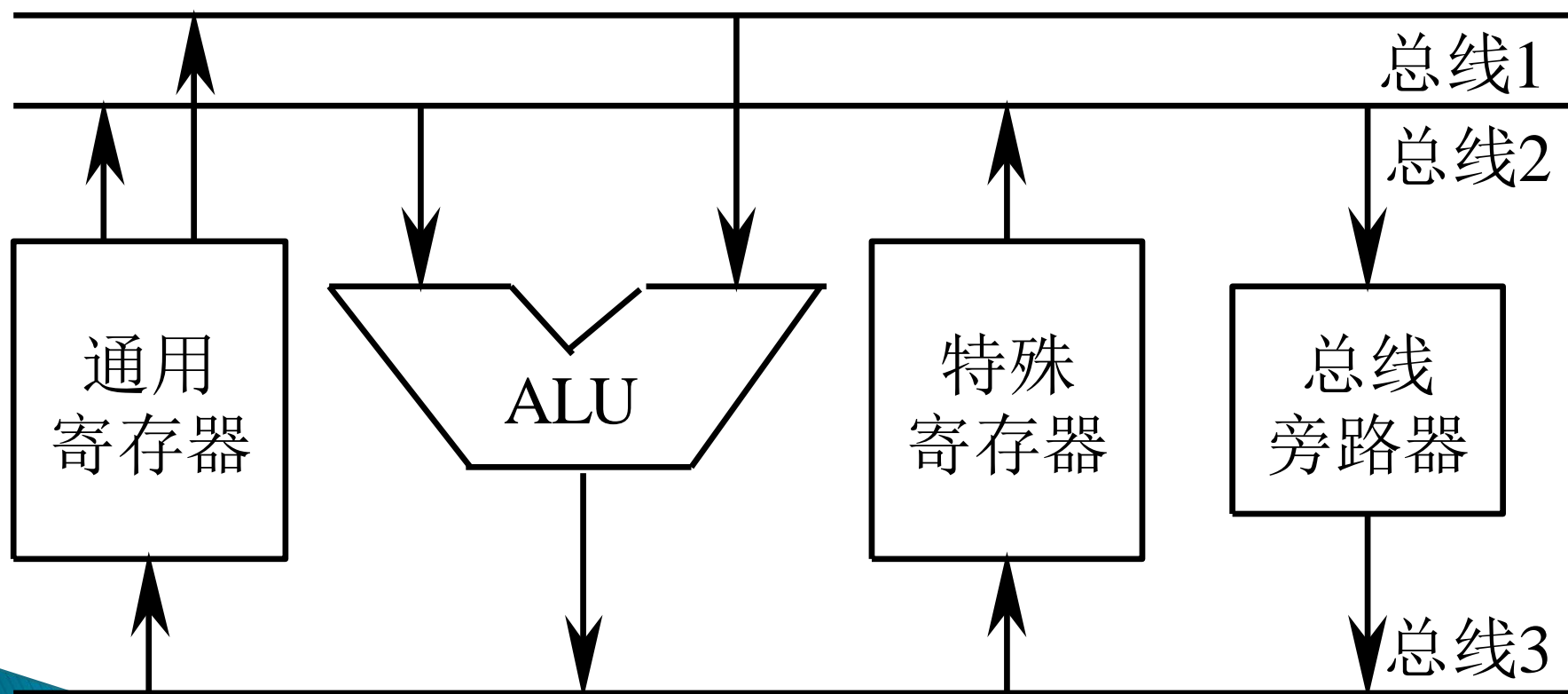
4.9 运算器的基本组成与实例



4.9 运算器的基本组成与实例

(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。



4.9 运算器的基本组成与实例

4.9.2 ALU举例

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。

4.9 运算器的基本组成与实例

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；

4.9 运算器的基本组成与实例

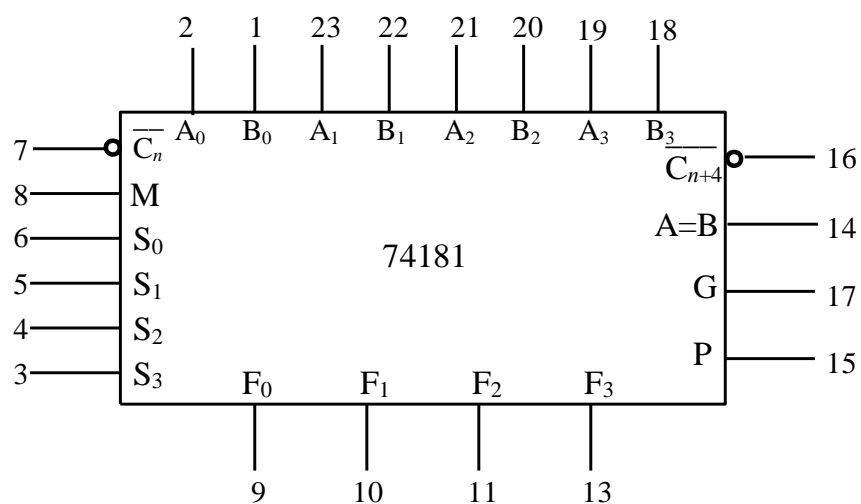
M: 工作方式, **M=0**为算术操作, **M=1**为逻辑操作;

S₀~S₃: 功能选择线。

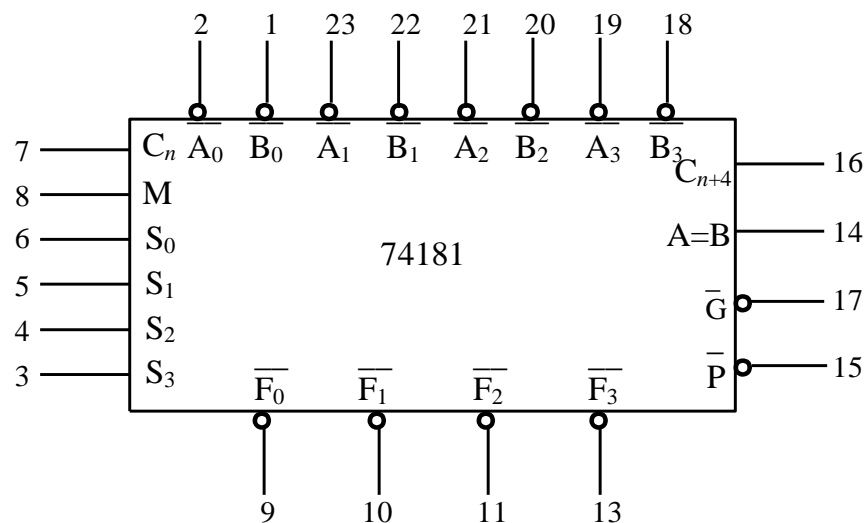
74181的4位作为一个小组, 组间既可以采用串行进位, 也可以采用并行进位。

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。

4.9 运算器的基本组成与实例



(a)



(b)

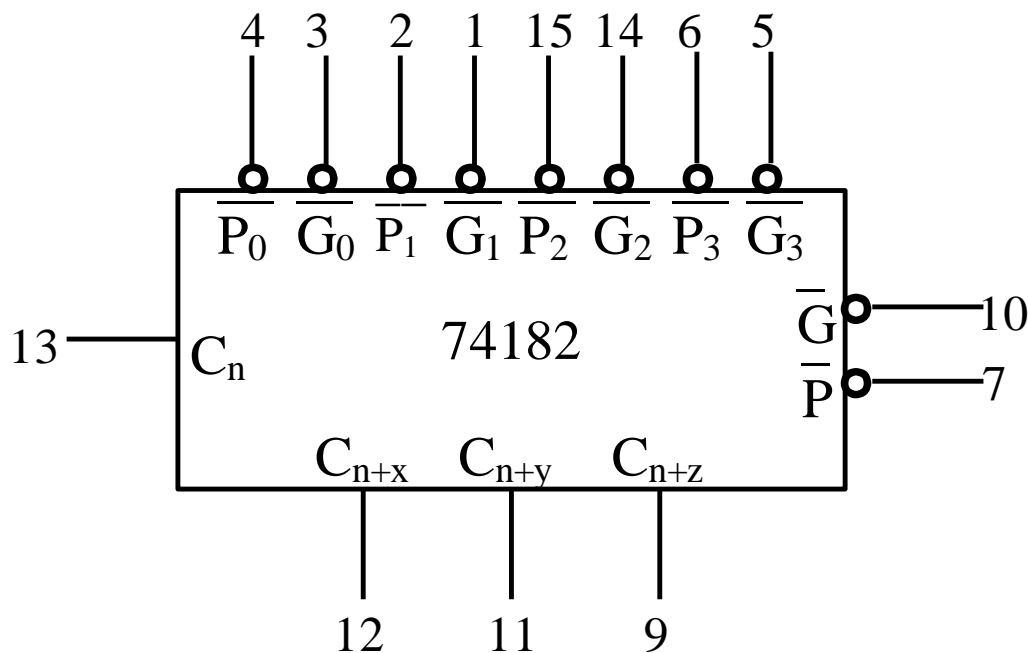
4.9 运算器的基本组成与实例

| 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 | $F=\overline{AB}$ | $F=AB$ 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 | $F=\overline{A}+\overline{B}$ | $F=A$ 加 $(A+\overline{B})$ | $F=A$ 加 $(A+\overline{B})$ 加 1 | $F=\overline{AB}$ | $F=A$ 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 | $F=A$ |

4.9 运算器的基本组成与实例

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）。



4.9 运算器的基本组成与实例

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。

$$C_{16} = \underbrace{G_4^* + P_4^* G_3^* + P_4^* P_3^* G_2^* + P_4^* P_3^* P_2^* G_1^*}_{\text{大组进位产生函数 } G_1^{**}} + \underbrace{P_4^* P_3^* P_2^* P_1^*}_{\text{大组进位传递函数 } P_1^{**}} C_0$$

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

大组进位
产生函数 G_1^{**}

大组进位
传递函数 P_1^{**}

4.9 运算器的基本组成与实例

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+x} 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。

第4章 小结

4.1 基本运算的实现

▶ 加法器

串行加法器与并行加法器

▶ 进位的产生和传递

▶ 并行加法器快速进位

第4章 小结

4.2 定点加减运算

- ▶ 补码加法运算

- ▶ 补码减法运算

已知 $[Y]_{\text{补}}$ 求 $[-Y]_{\text{补}}$ 的方法

- ▶ 补码的溢出判断与检测方法

一位符号位，进位位，双符号位补码

第4章 小结

4.3 带符号数的移位运算和舍入操作

- ▶ 补码的移位运算
- ▶ 舍入操作

4.4 定点乘法运算

- ▶ 原码一位乘法
- ▶ 补码一位乘法

第4章 小结

4.5 定点除法运算

- ▶ 原码加减交替除法
- ▶ 补码加减交替除法

4.6 规格化浮点运算算法

- ▶ 浮点加减运算
- ▶ 浮点乘除运算

第4章 小结

4.7 十进制整数的加减运算

- ▶ 十进制加法运算的校正

4.8 逻辑运算与实现

4.9 运算器的基本组成与实例

- ▶ 运算器的基本结构
 - ▶ ALU举例
- 