

第 6 章 计算机的运算方法

6.1 无符号数和有符号数

6.2 数的定点表示和浮点表示

6.3 定点运算

6.4 浮点四则运算

6.5 算术逻辑单元

6.1 无符号数和有符号数

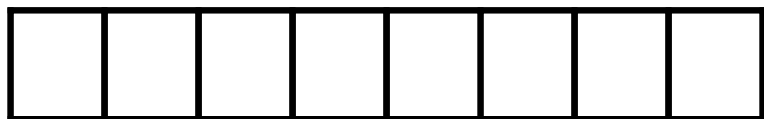
- 一、无符号数
 - ✓ 定义
- 二、有符号数
 - ✓ 特点
 - ✓ 举例
 - 机器数与真值的转换
 - 不同机器数形式之间的转化
 - 1、机器数与真值
 - 2、原码表示法
 - 3、补码表示法
 - 4、反码表示法
 - 5、移码表示法
 - ✓ 机器数表示的范围与其字长有关

6.1 无符号数和有符号数

一、无符号数

寄存器的位数

反映无符号数的表示范围



8 位

0 ~ 255



16 位

0 ~ 65535

二、有符号数

6.1

1. 机器数与真值

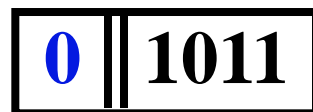
真值

机器数

带符号的数

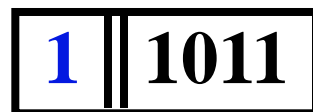
符号数字化的数

+ 0.1011



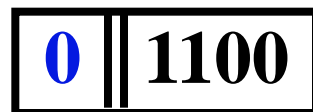
小数点的位置

- 0.1011



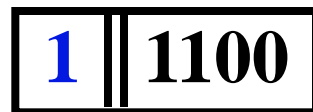
小数点的位置

+ 1100



小数点的位置

- 1100



小数点的位置

2. 原码表示法

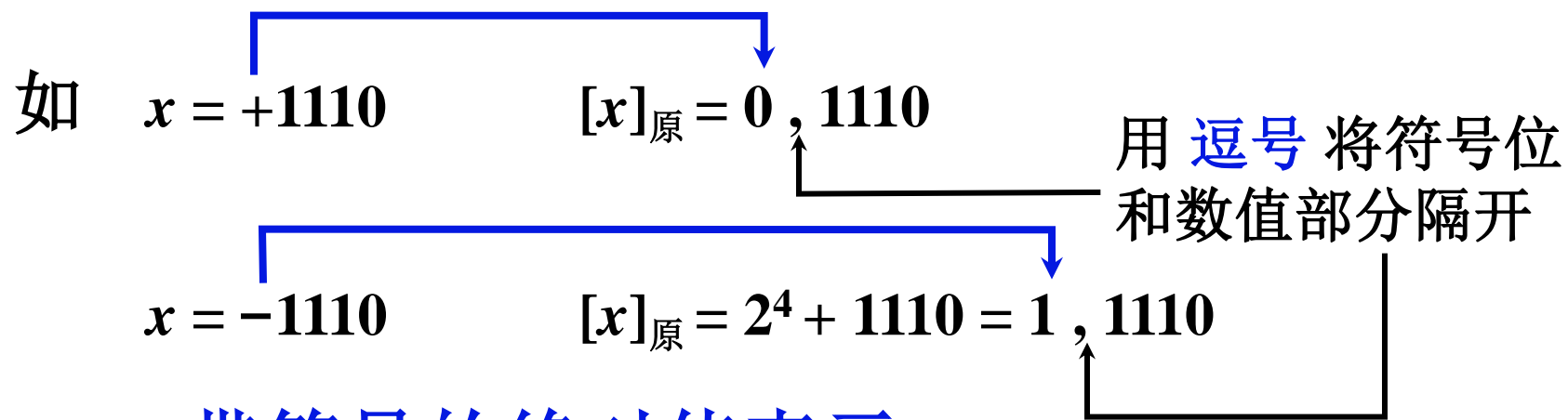
6.1

(1) 定义

整数

$$[x]_{\text{原}} = \begin{cases} 0, & x > 0 \\ 2^n - x, & x < 0 \end{cases}$$

x 为真值 n 为整数的位数



带符号的绝对值表示

小数

6.1

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

x 为真值

如

$x = +0.1101$ $[x]_{\text{原}} = 0.1101$ 用 **小数点** 将符号位和数值部分隔开

$x = -0.1101$ $[x]_{\text{原}} = 1 - (-0.1101) = 1.1101$

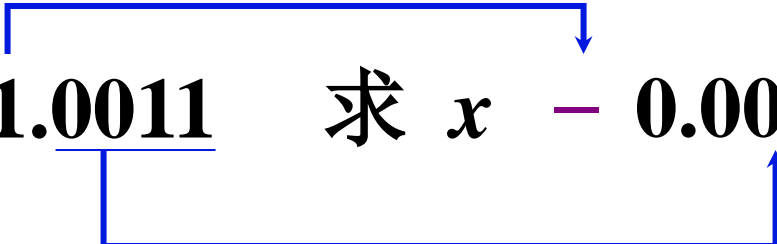
$x = +0.1000000$ $[x]_{\text{原}} = 0.1000000$ 用 **小数点** 将符号位和数值部分隔开

$x = -0.1000000$ $[x]_{\text{原}} = 1 - (-0.1000000) = 1.1000000$

(2) 举例

6.1

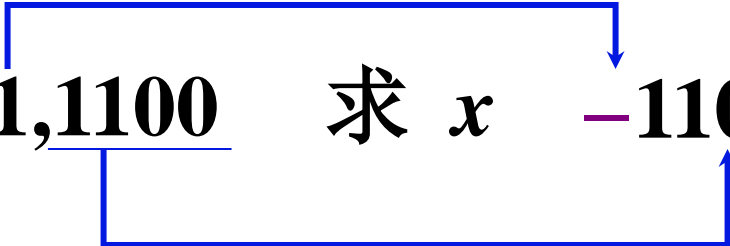
例 6.1 已知 $[x]_{\text{原}} = 1.0011$ 求 x $- 0.0011$



解：由定义得

$$x = 1 - [x]_{\text{原}} = 1 - 1.0011 = -0.0011$$

例 6.2 已知 $[x]_{\text{原}} = 1,1100$ 求 x $- 1100$



解：由定义得

$$x = 2^4 - [x]_{\text{原}} = 10000 - 1,1100 = -1100$$

6.1

例 6.3 已知 $[x]_{\text{原}} = 0.1101$ 求 x

解：根据定义 $\because [x]_{\text{原}} = 0.1101$

$$\therefore x = +0.1101$$

例 6.4 求 $x = 0$ 的原码

解：设 $x = +0.0000$ $[+0.0000]_{\text{原}} = 0.0000$

$x = -0.0000$ $[-0.0000]_{\text{原}} = 1.0000$

同理，对于整数 $[+0]_{\text{原}} = 0,0000$

$[-0]_{\text{原}} = 1,0000$

$\therefore [+0]_{\text{原}} \neq [-0]_{\text{原}}$

原码的特点：简单、直观

6.1

但是用原码作加法时，会出现如下问题：

要求	数1	数2	实际操作	结果符号
加法	正	正	加	正
加法	正	负	减	可正可负
加法	负	正	减	可正可负
加法	负	负	加	负

能否 只作加法？

找到一个与负数等价的正数 来代替这个负数

就可使 减 → 加

3. 补码表示法

6.1

(1) 补的概念

- 时钟

逆时针

$$\begin{array}{r} 6 \\ - 3 \\ \hline 3 \end{array}$$

顺时针

$$\begin{array}{r} 6 \\ + 9 \\ \hline 15 \\ - 12 \\ \hline 3 \end{array}$$

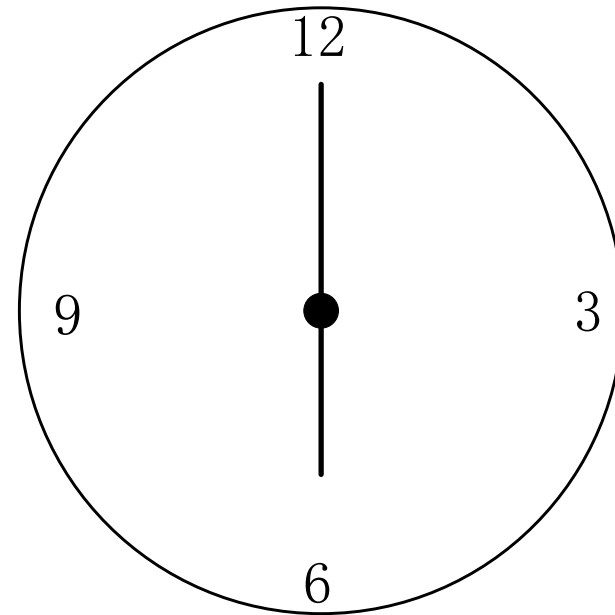
可见 -3 可用 $+9$ 代替 减法 \rightarrow 加法
称 $+9$ 是 -3 以 12 为模的 补数

记作 $-3 \equiv +9 \pmod{12}$

同理 $-4 \equiv +8 \pmod{12}$

$-5 \equiv +7 \pmod{12}$

时钟以
12为模



结论

6.1

- 一个负数加上 “模” 即得该负数的补数
- 一个正数和一个负数互为补数时
它们绝对值之和即为 模 数

• 计数器（模 16） $1011 \longrightarrow 0000$?

$$\begin{array}{r} 1011 \\ - 1011 \\ \hline 0000 \end{array}$$

$$\begin{array}{r} 1011 \\ + 0101 \\ \hline 10000 \end{array}$$

自然去掉

可见 -1011 可用 $+0101$ 代替

记作 $-1011 \equiv +0101 \pmod{2^4}$

同理 $-011 \equiv +101 \pmod{2^3}$

$-0.1001 \equiv +1.0111 \pmod{2}$

(2) 正数的补数即为其本身

6.1

两个互为补数的数

分别加上模

结果仍互为补数

$$\boxed{-1011} \equiv +0101 \pmod{2^4}$$

$$\begin{array}{r} +10000 \\ \hline \end{array}$$

$$+0101 \equiv +\boxed{10101}$$

$$\therefore +0101 \equiv +0101 \pmod{2^4}$$

丢掉

可见 $+0101 \xrightarrow{?} +0101$
 $\quad \quad \quad \downarrow ?$
 $\quad \quad \quad -1011$

? $\boxed{0},0101 \rightarrow +0101$

? $\boxed{1},0101 \rightarrow -1011$

$$2^{4+1} - 1011 = 100000$$

$$\begin{array}{r} -1011 \\ \hline 1,0101 \end{array}$$

用 逗号 将符号位
和数值部分隔开

$$\pmod{2^{4+1}}$$

(3) 补码定义

6.1

整数

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

x 为真值

n 为整数的位数

如

$$x = +1010$$

$$x = -1011000$$

$$[x]_{\text{补}} = 0,1010$$

用 逗号 将符号位
和数值部分隔开

$$[x]_{\text{补}} = 2^{7+1} + (-1011000)$$

$$= 100000000$$

$$- \quad 1011000$$

$$\hline 1,0101000$$

小数

6.1

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

x 为真值

如 $x = +0.1110$

$x = -0.1100000$

$$[x]_{\text{补}} = 0.1110$$

$$[x]_{\text{补}} = 2 + (-0.1100000)$$

$$= 10.0000000$$

$$- 0.1100000$$

$$\hline 1.0100000$$

用 小数点 将符号位
和数值部分隔开

(4) 求补码的快捷方式

6.1

设 $x = -1010$ 时

$$\begin{aligned} \text{则 } [x]_{\text{补}} &= 2^{4+1} - 1010 &= 11111 + 1 - 1010 \\ &= 100000 &= 11111 + 1 \\ &\quad - 1010 &\quad - 1010 \\ \hline &= 1,0110 &\quad \boxed{10101} + 1 \\ & &= 1,0110 \end{aligned}$$

$$\text{又 } [x]_{\text{原}} = \boxed{1,1010}$$

当真值为 负 时，补码 可用 原码除符号位外
每位取反，末位加 1 求得

(5) 举例

6.1

例 6.5 已知 $[x]_{\text{补}} = 0.0001$

求 x

解：由定义得 $x = +0.0001$

例 6.6 已知 $[x]_{\text{补}} = 1.0001$ $[x]_{\text{补}} \xrightarrow{?} [x]_{\text{原}}$

求 x

$$[x]_{\text{原}} = 1.1111$$

解：由定义得

$$\therefore x = -0.1111$$

$$x = [x]_{\text{补}} - 2$$

$$= 1.0001 - 10.0000$$

$$= -0.1111$$

例 6.7 已知 $[x]_{\text{补}} = 1,1110$

6.1

求 x

解：由定义得

$$x = [x]_{\text{补}} - 2^{4+1}$$

$$= 1,1110 - 100000$$

$$= -0010$$

$$[x]_{\text{补}} \xrightarrow{?} [x]_{\text{原}}$$

$$[x]_{\text{原}} = 1,0010$$

$$\therefore x = -0010$$

当真值为 负 时，原码 可用 补码除符号位外

每位取反，末位加 1 求得

练习 求下列真值的补码

6.1

真值	$[x]_{\text{补}}$	$[x]_{\text{原}}$
$x = +70 = 1000110$	0, 1000110	0, 1000110
$x = -70 = -1000110$	1, 0111010	1, 1000110
$x = 0.1110$	0.1110	0.1110
$x = -0.1110$	1.0010	1.1110
$x = \boxed{0.0000} \quad [+0]_{\text{补}} = [-0]_{\text{补}}$	$\boxed{0.0000}$	0.0000
$x = \boxed{-0.0000}$	$\boxed{0.0000}$	1.0000
$x = -1.0000$	1.0000	不能表示

由小数补码定义
$$[x]_{\text{补}} = \begin{cases} x & 1 > x \geq 0 \\ 2 + x & 0 > x \geq -1 \pmod{2} \end{cases}$$

$$[-1]_{\text{补}} = 2 + x = 10.0000 - 1.0000 = 1.0000$$

4. 反码表示法

6.1

(1) 定义

整数

$$[x]_{\text{反}} = \begin{cases} 0, & x & 2^n > x \geq 0 \\ (2^{n+1} - 1) + x & 0 \geq x > -2^n \pmod{2^{n+1} - 1} \end{cases}$$

x 为真值

n 为整数的位数

如 $x = +1101$

$$[x]_{\text{反}} = 0,1101$$



用 逗号 将符号位

和数值部分隔开

$x = -1101$

$$[x]_{\text{反}} = (2^{4+1} - 1) - 1101$$

$$= 11111 - 1101$$

$$= 1,0010$$



小数

6.1

$$[x]_{\text{反}} = \begin{cases} x & 1 > x \geq 0 \\ (2 - 2^{-n}) + x & 0 \geq x > -1 \pmod{2 - 2^{-n}} \end{cases}$$

x 为真值

n 为小数的位数

如

$$x = +0.1101$$

$$x = -0.1010$$

$$[x]_{\text{反}} = 0.1101$$

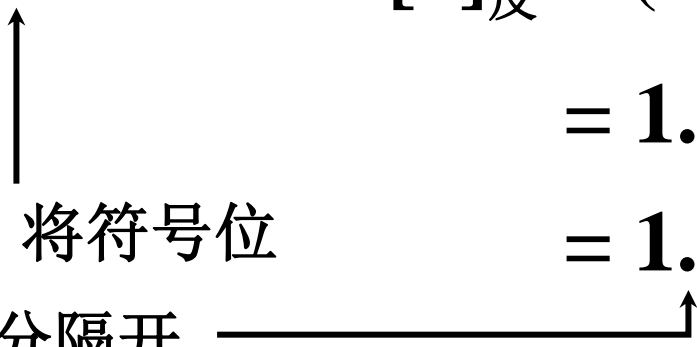
$$[x]_{\text{反}} = (2 - 2^{-4}) - 0.1010$$

$$= 1.1111 - 0.1010$$

$$= 1.0101$$

用 小数点 将符号位

和数值部分隔开



(2) 举例

6.1

例6.8 已知 $[x]_{\text{反}} = 0,1110$ 求 x

解: 由定义得 $x = +1110$

例6.9 已知 $[x]_{\text{反}} = 1,1110$ 求 x

解: 由定义得
$$\begin{aligned} x &= [x]_{\text{反}} - (2^{4+1} - 1) \\ &= 1,1110 - 11111 \\ &= -0001 \end{aligned}$$

例 6.10 求 0 的反码

解: 设 $x = +0.0000$ $[+0.0000]_{\text{反}} = 0.0000$

$x = -0.0000$ $[-0.0000]_{\text{反}} = 1.1111$

同理, 对于整数 $[+0]_{\text{反}} = 0,0000$ $[-0]_{\text{反}} = 1,1111$

$\therefore [+0]_{\text{反}} \neq [-0]_{\text{反}}$

三种机器数的小结

- 最高位为符号位，书写上用 “,”（整数）或 “.”（小数）将数值部分和符号位隔开
- 对于正数，原码 = 补码 = 反码
- 对于负数，符号位为 1，其数值部分
原码除符号位外每位取反末位加 1 → 补码
原码除符号位外每位取反 → 反码

例6.11 设机器数字长为8位（其中1位为符号位）6.1

对于整数，当其分别代表无符号数、原码、补码和反码时，对应的真值范围各为多少？

二进制代码	无符号数 对应的真值	原码对应 的真值	补码对应 的真值	反码对应 的真值
00000000	0	+0	± 0	+0
00000001	1	+1	+1	+1
00000010	2	+2	+2	+2
⋮	⋮	⋮	⋮	⋮
01111111	127	+127	+127	+127
10000000	128	-0	-128	-127
10000001	129	-1	-127	-126
⋮	⋮	⋮	⋮	⋮
11111101	253	-125	-3	-2
11111110	254	-126	-2	-1
11111111	255	-127	-1	-0

例6.12 已知 $[y]_{\text{补}}$ 求 $[-y]_{\text{补}}$

解： 设 $[y]_{\text{补}} = y_0 \cdot y_1 y_2 \cdots y_n$

< I > $[y]_{\text{补}} = 0 \cdot y_1 y_2 \cdots y_n$

$[y]_{\text{补}}$ 连同符号位在内， 每位取反， 末位加 1

即得 $[-y]_{\text{补}}$

$$[-y]_{\text{补}} = 1 \cdot \overline{y_1} \overline{y_2} \cdots \overline{y_n} + 2^{-n}$$

< II > $[y]_{\text{补}} = 1 \cdot y_1 y_2 \cdots y_n$

$[y]_{\text{补}}$ 连同符号位在内， 每位取反， 末位加 1

即得 $[-y]_{\text{补}}$

$$[-y]_{\text{补}} = 0 \cdot \overline{y_1} \overline{y_2} \cdots \overline{y_n} + 2^{-n}$$



5. 移码表示法

6.1

补码表示很难直接判断其真值大小

如	十进制	二进制	补码	
	$x = +21$	$+10101$	$0,10101$	 错 大
	$x = -21$	-10101	$1,01011$	
	$x = +31$	$+11111$	$0,11111$	 错 大
	$x = -31$	-11111	$1,00001$	

$x + 2^5$

$+10101 + 100000 = 110101$	 大 正确
$-10101 + 100000 = 001011$	
$+11111 + 100000 = 111111$	 大 正确
$-11111 + 100000 = 000001$	

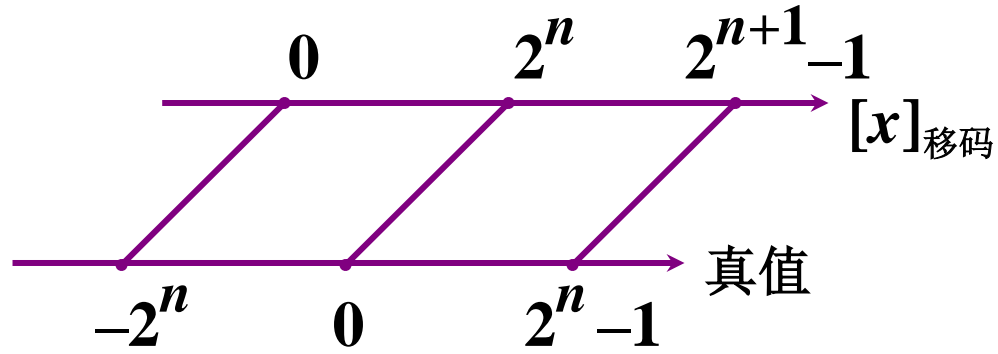
(1) 移码定义

6.1

$$[x]_{\text{移}} = 2^n + x \quad (2^n > x \geq -2^n)$$

x 为真值, n 为 整数的位数

移码在数轴上的表示



如 $x = 10100$

$$[x]_{\text{移}} = 2^5 + 10100 = 1,10100$$

$x = -10100$

$$[x]_{\text{移}} = 2^5 - 10100 = 0,01100$$

用 逗号 将符号位
和数值部分隔开

(2) 移码和补码的比较

设 $x = +1100100$

$$[x]_{\text{移}} = 2^7 + 1100100 = \mathbf{1},1100100$$

$$[x]_{\text{补}} = \mathbf{0},1100100$$

设 $x = -1100100$

$$[x]_{\text{移}} = 2^7 - 1100100 = \mathbf{0},0011100$$

$$[x]_{\text{补}} = \mathbf{1},0011100$$

补码与移码只差一个符号位

(3) 真值、补码和移码的对照表

6.1

真值 x ($n=5$)	$[x]_{\text{补}}$	$[x]_{\text{移}}$	$[x]_{\text{移}}$ 对应的 十进制整数
- 1 0 0 0 0	1 0 0 0 0	0 0 0 0 0	0
- 1 1 1 1 1	1 0 0 0 1	0 0 0 0 1	1
- 1 1 1 1 0	1 0 0 0 1 0	0 0 0 0 1 0	2
⋮	⋮	⋮	⋮
- 0 0 0 0 1	1 1 1 1 1 1	0 1 1 1 1 1	31
± 0 0 0 0 0	0 0 0 0 0 0	1 0 0 0 0 0	32
+ 0 0 0 0 1	0 0 0 0 0 1	1 0 0 0 0 1	33
+ 0 0 0 1 0	0 0 0 0 1 0	1 0 0 0 1 0	34
⋮	⋮	⋮	⋮
+ 1 1 1 1 0	0 1 1 1 1 0	1 1 1 1 1 0	62
+ 1 1 1 1 1	0 1 1 1 1 1	1 1 1 1 1 1	63

(4) 移码的特点

6.1

➤ 当 $x = 0$ 时 $[+0]_{\text{移}} = 2^5 + 0 = 1,00000$

$$[-0]_{\text{移}} = 2^5 - 0 = 1,00000$$

$$\therefore [+0]_{\text{移}} = [-0]_{\text{移}}$$

➤ 当 $n = 5$ 时 最小的真值为 $-2^5 = -100000$

$$[-100000]_{\text{移}} = 2^5 - 100000 = 000000$$

可见，最小真值的移码为全 0

用移码表示浮点数的阶码

能方便地判断浮点数的阶码大小

6.2 数的定点表示和浮点表示

- 一、定点表示
- 二、浮点表示
 - 1. 浮点数的表示形式
 - 2. 浮点数的表示范围
 - 3. 浮点数的规格化形式
 - 4. 浮点数的规格化
- 三、举例
- 四、IEEE 754 标准

二、浮点表示

- 为什么在计算机中要引入浮点数表示？
- 浮点表示的格式是什么？
- 尾数和阶码的基值必须是2吗？基值的影响？
- 表数范围与精度和哪些因素有关？
- 为什么要引入规格化表示？
- 目前浮点数表示格式的标准是什么？

二、浮点表示

- 为什么要引入浮点数表示
 - 编程困难，程序员要调节小数点的位置；
 - 数的表示范围小，为了能表示两个大小相差很大的数据，需要很长的机器字长；
 - 例如：太阳的质量是 0.2×10^{34} 克，一个电子的质量大约为 0.9×10^{-27} 克，两者的差距为 10^{61} 以上，若用定点数据表示： $2^x > 10^{61}$ ，解的， $x > 203$ 位。
 - 数据存储单元的利用率往往很低。

二、浮点表示

6.2

$N = S \times r^j$ 浮点数的一般形式

S 尾数 j 阶码 r 尾数的基值

计算机中 r 取 2、4、8、16 等

当 $r = 2$ $N = 11.0101$ 二进制表示

✓ $= 0.110101 \times 2^{10}$ 规格化数

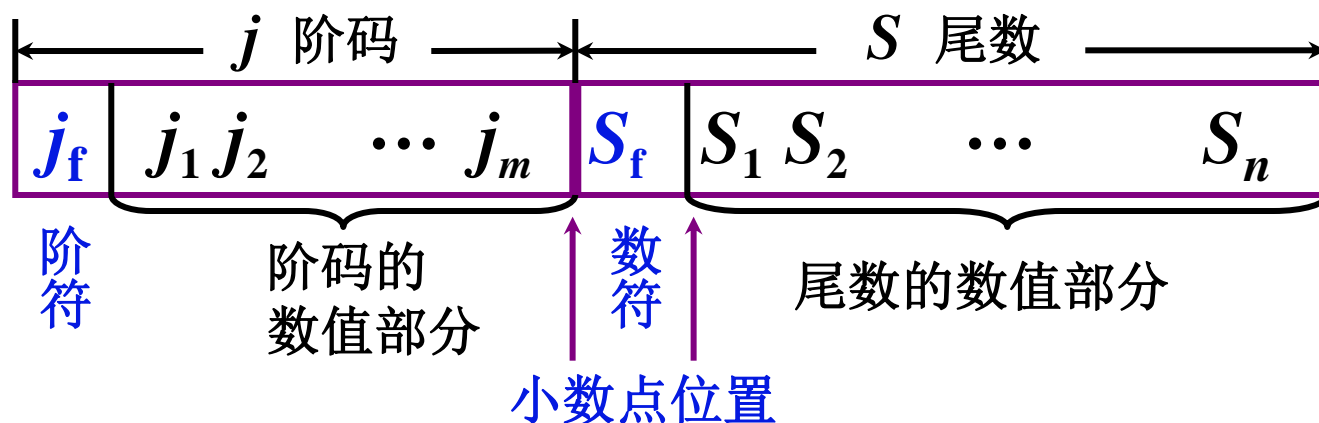
$$= 1.10101 \times 2^1$$

$$= 1101.01 \times 2^{-10}$$

$$✓ = 0.00110101 \times 2^{100}$$

计算机中 S 小数、可正可负
 j 整数、可正可负

1. 浮点数的表示形式



S_f 代表浮点数的符号

n 其位数反映浮点数的精度

m 其位数反映浮点数的表示范围

j_f 和 m 共同表示小数点的实际位置

2. 浮点数的表示范围

6.2

上溢 阶码 > 最大阶码

下溢 阶码 < 最小阶码 按 机器零 处理

上溢

上溢

负数区

下溢

正数区

最小负数

$$\begin{aligned} & -2^{(2^m-1)} \times (1-2^{-n}) \\ & -2^{15} \times (1-2^{-10}) \end{aligned}$$

最大正数

$$\begin{aligned} & 2^{(2^m-1)} \times (1-2^{-n}) \\ & 2^{15} \times (1-2^{-10}) \end{aligned}$$

最小正数

$$\begin{aligned} & 2^{-(2^m-1)} \times 2^{-n} \\ & 2^{-15} \times 2^{-10} \end{aligned}$$

最大负数

$$\begin{aligned} & -2^{-(2^m-1)} \times 2^{-n} \\ & -2^{-15} \times 2^{-10} \end{aligned}$$

设 $m = 4$
 $n = 10$

设机器数字长为 24 位，欲表示 ± 3 万的十进制数，试问在保证数的最大精度的前提下，除阶符、数符各取 1 位外，阶码、尾数各取几位？

解： $\because 2^{14} = 16384 \quad 2^{15} = 32768$

\therefore 如果是定点数 15 位二进制数可反映 ± 3 万之间的十进制数

$$2^{\boxed{15}} \times 0.\underbrace{\times \times \times \dots \times \times \times}_{n\text{位}}$$

$m = 4, 5, 6, \dots$

满足 最大精度 可取 $m = 4, n = 18$

3. 浮点数的规格化形式

6.2

$r = 2$ 尾数最高位为 1

$r = 4$ 尾数最高 2 位不全为 0

$r = 8$ 尾数最高 3 位不全为 0

基数不同，浮点数的
规格化形式不同

4. 浮点数的规格化

$r = 2$ 左规 尾数左移 1 位，阶码减 1

右规 尾数右移 1 位，阶码加 1

$r = 4$ 左规 尾数左移 2 位，阶码减 1

右规 尾数右移 2 位，阶码加 1

$r = 8$ 左规 尾数左移 3 位，阶码减 1

右规 尾数右移 3 位，阶码加 1

基数 r 越大，可表示的浮点数的范围越大

基数 r 越大，浮点数的精度降低

例如：设 $m = 4$, $n = 10$, $r = 2$

6.2

尾数规格化后的浮点数表示范围

最大正数 $2^{+1111} \times 0.\underbrace{1111111111}_{10 \text{ 个 } 1} = 2^{15} \times (1 - 2^{-10})$

最小正数 $2^{-1111} \times 0.1\underbrace{0000000000}_{9 \text{ 个 } 0} = 2^{-15} \times 2^{-1} = 2^{-16}$

最大负数 $2^{-1111} \times (-0.1\underbrace{0000000000}_{9 \text{ 个 } 0}) = -2^{-15} \times 2^{-1} = -2^{-16}$

最小负数 $2^{+1111} \times (-0.\underbrace{1111111111}_{10 \text{ 个 } 1}) = -2^{15} \times (1 - 2^{-10})$

三、举例

6.2

例 6.13 将 $+\frac{19}{128}$ 写成二进制定点数、浮点数及在定点机和浮点机中的机器数形式。其中数值部分均取 10 位，数符取 1 位，浮点数阶码取 5 位（含 1 位阶符），尾数规格化。

解： 设 $x = +\frac{19}{128}$

二进制形式 $x = 0.0010011$

定点表示 $x = 0.0010011\ 000$

浮点规格化形式 $x = 0.1001100000 \times 2^{-10}$

定点机中 $[x]_{\text{原}} = [x]_{\text{补}} = [x]_{\text{反}} = 0.0010011000$

浮点机中 $[x]_{\text{原}} = 1, 0010; 0. 1001100000$

$[x]_{\text{补}} = 1, 1110; 0. 1001100000$

$[x]_{\text{反}} = 1, 1101; 0. 1001100000$

例 6.14 将 -58 表示成二进制定点数和浮点数，**6.2**
并写出它在定点机和浮点机中的三种机器数及阶码
为移码、尾数为补码的形式（其他要求同上例）。

解： 设 $x = -58$

二进制形式 $x = -111010$

定点表示 $x = -0000111010$

浮点规格化形式 $x = -(0.1110100000) \times 2^{110}$

定点机中

$[x]_{\text{原}} = 1, 0000111010$

$[x]_{\text{补}} = 1, 1111000110$

$[x]_{\text{反}} = 1, 1111000101$

浮点机中

$[x]_{\text{原}} = 0, 0110; 1. 1110100000$

$[x]_{\text{补}} = 0, 0110; 1. 0001100000$

$[x]_{\text{反}} = 0, 0110; 1. 0001011111$

$[x]_{\text{阶移、尾补}} = 1, 0110; 1. 0001100000$

机器零

6.2

- 当浮点数 **尾数为 0** 时，不论其阶码为何值
按机器零处理
- 当浮点数 **阶码等于或小于它所表示的最小
数** 时，不论尾数为何值，按机器零处理

如 $m = 4$ $n = 10$

当阶码和尾数都用补码表示时，机器零为

$\times, \times \times \times \times; \quad \mathbf{0.00 \dots 0}$

(阶码 = -16) $\mathbf{1, 0000}; \quad \times.\times\times \dots \times$

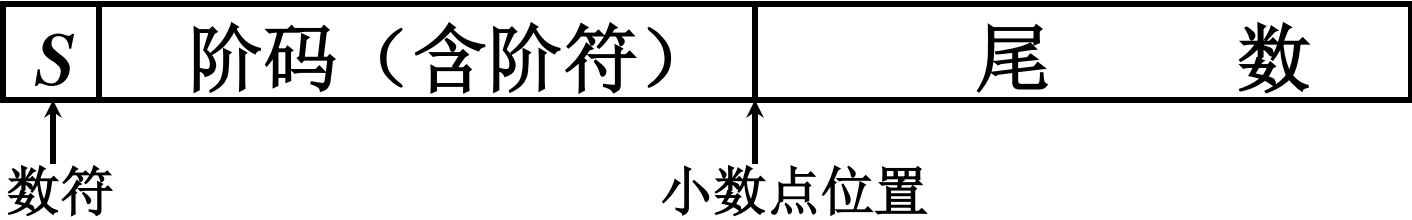
当阶码用移码，尾数用补码表示时，机器零为

$\mathbf{0, 0000; 0.00 \dots 0}$

有利于机器中“判 0”电路的实现

四、IEEE 754 标准

6.2



尾数为规格化表示

非 “0” 的有效位最高位为 “1” (隐含)

	符号位 <i>S</i>	阶码	尾数	总位数
短实数	1	8	23	32
长实数	1	11	52	64
临时实数	1	15	64	80

第 6 章 计算机的运算方法

6.1 无符号数和有符号数

6.2 数的定点表示和浮点表示

6.3 定点运算

6.4 浮点四则运算

6.5 算术逻辑单元

6.3 定点运算

- 一、移位运算
- 二、加减法运算
- 三、乘法运算
- 四、除法运算

6.3 定点运算

- 一、移位运算
 - 1、移位运算的数学意义
 - 2、算术移位规则
 - 3、算术移位的硬件实现
 - 4、算术移位与逻辑移位的区别

6.3 定点运算

一、移位运算

1. 移位的意义

$$15.\text{m} = 1500.\text{cm}$$

小数点右移 2 位

机器用语 15 相对于小数点 左移 2 位

（ 小数点不动 ）

左移 绝对值扩大

右移 绝对值缩小

在计算机中，移位与加减配合，能够实现乘除运算

2. 算术移位规则

6.3

$$x = -0.x_1x_2...x_k100...000$$

$$[x]_{补} = 1.\bar{x}_1\bar{x}_2...\bar{x}_k100...000$$

符号位不变

	码 制	添补代码
正数	原码、补码、反码	0
负数	原 码	0
	补 码	左移 添 0
		右移 添 1
	反 码	1

例6.16

6.3

设机器数字长为 8 位（含 1 位符号位），写出 $A = +26$ 时，三种机器数左、右移一位和两位后的表示形式及对应的真值，并分析结果的正确性。

解： $A = +26 = +11010$
则 $[A]_{原} = [A]_{补} = [A]_{反} = 0,0011010$

移位操作	机 器 数	对应的真值
	$[A]_{原}=[A]_{补}=[A]_{反}$	
移位前	0,0011010	+26
左移一位	0,0110100	+52
左移两位	0,1101000	+104
右移一位	0,0001101	+13
右移两位	0,0000110	+6

例6.17

6.3

设机器数字长为 8 位（含 1 位符号位），写出 $A = -26$ 时，三种机器数左、右移一位和两位后的表示形式及对应的真值，并分析结果的正确性。

解： $A = -26 = -11010$

原码	移位操作	机 器 数	对应的真值
	移位前	1,0011010	- 26
	左移一位	1,0110100	- 52
	左移两位	1,1101000	- 104
	右移一位	1,0001101	- 13
	右移两位	1,0000110	- 6

6.3

补码

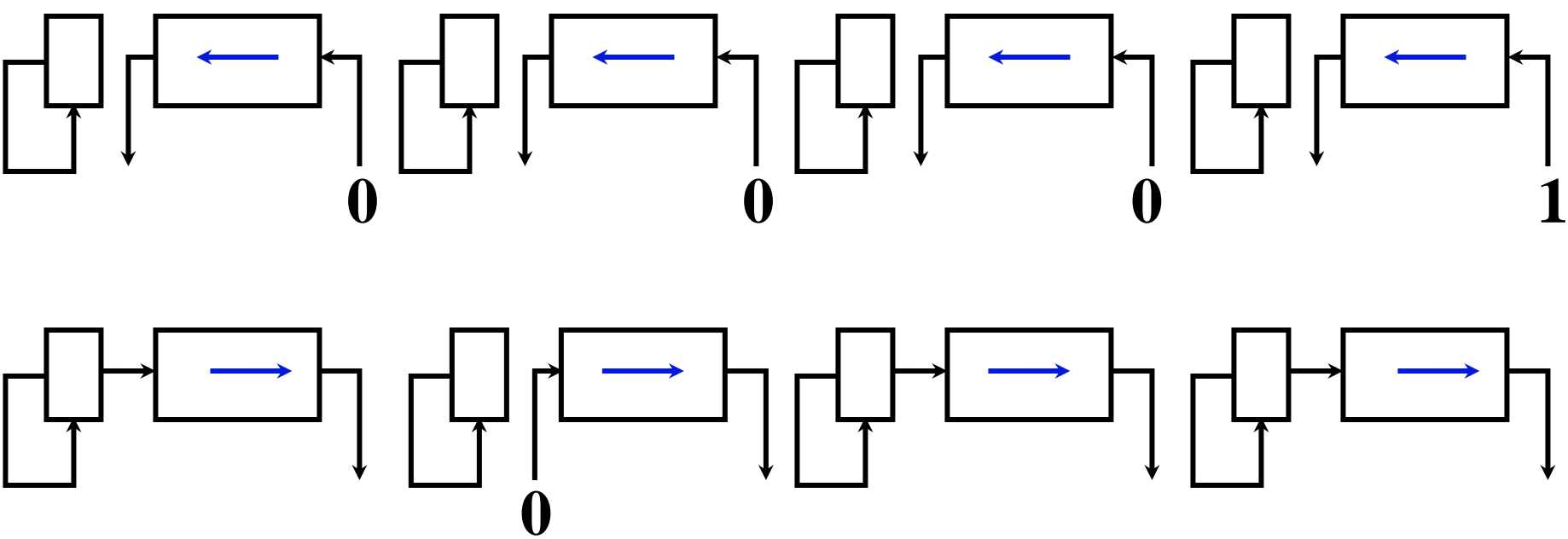
移位操作	机 器 数	对应的真值
移位前	1,1100110	– 26
左移一位	1,1001100	– 52
左移两位	1,0011000	– 104
右移一位	1,1110011	– 13
右移两位	1,111001	– 7

反码

移位操作	机 器 数	对应的真值
移位前	1,1100101	– 26
左移一位	1,1001011	– 52
左移两位	1,0010111	– 104
右移一位	1,1110010	– 13
右移两位	1,111001	– 6

3. 算术移位的硬件实现

6.3



(a) 真值为正 (b) 负数的原码 (c) 负数的补码 (d) 负数的反码

← 丢 1 出错 出错 正确 正确

→ 丢 1 影响精度 影响精度 影响精度 正确

4. 算术移位和逻辑移位的区别

6.3

算术移位 有符号数的移位

逻辑移位 无符号数的移位

逻辑左移 低位添 0，高位移丢

逻辑右移 高位添 0，低位移丢

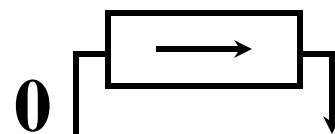
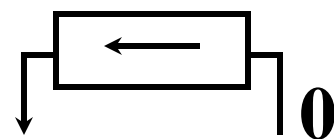
例如 **01010011**

逻辑左移 **10100110**

算术左移 **00100110**

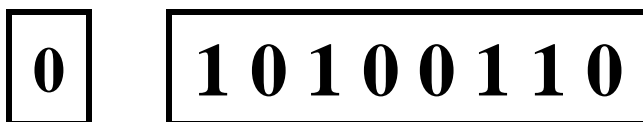
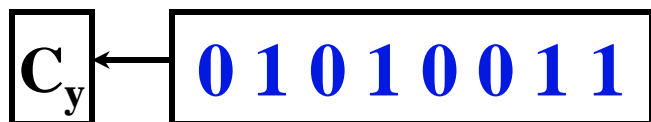
逻辑右移 **01011001**

算术右移 **11011001** (补码)



10110010

高位 1 移丢



6.3 定点运算

- 一、移位运算
 - 1、移位运算的数学意义
 - 2、算术移位规则
 - 3、算术移位的硬件实现
 - 4、算术移位与逻辑移位的区别

6.3 定点运算

- 一、移位运算
- 二、加减法运算
- 三、乘法运算
- 四、除法运算

但是用原码作加法时，会出现如下问题：

要求	数1	数2	实际操作	结果符号
加法	正	正	加	正
加法	正	负	减	可正可负
加法	负	正	减	可正可负
加法	负	负	加	负

能否 只作加法？

找到一个与负数等价的正数 来代替这个负数

就可使 减 \longrightarrow 加

6.3 定点运算

- 二、加减法运算
 - 1、补码加减法运算的公式
 - 2、举例
 - 3、溢出的判断
 - 4、补码加减法的硬件配置

二、加减法运算

6.3

1. 补码加减运算公式

(1) 加法

整数 $[A]_{\text{补}} + [B]_{\text{补}} = [A+B]_{\text{补}} \pmod{2^{n+1}}$

小数 $[A]_{\text{补}} + [B]_{\text{补}} = [A+B]_{\text{补}} \pmod{2}$

(2) 减法

$$A-B = A+(-B)$$

整数 $[A-B]_{\text{补}} = [A+(-B)]_{\text{补}} = [A]_{\text{补}} + [-B]_{\text{补}} \pmod{2^{n+1}}$

小数 $[A-B]_{\text{补}} = [A+(-B)]_{\text{补}} = [A]_{\text{补}} + [-B]_{\text{补}} \pmod{2}$

连同符号位一起相加，符号位产生的进位自然丢掉

2. 举例

6.3

例 6.18 设 $A = 0.1011$, $B = -0.0101$

求 $[A + B]_{\text{补}}$

验证

解: $[A]_{\text{补}} = 0.1011$

$+ [B]_{\text{补}} = 1.1011$

$[A]_{\text{补}} + [B]_{\text{补}} = 10.0110 = [A + B]_{\text{补}}$

$\therefore A + B = 0.0110$

$$\begin{array}{r} 0.1011 \\ - 0.0101 \\ \hline 0.0110 \end{array}$$

例 6.19 设 $A = -9$, $B = -5$

求 $[A+B]_{\text{补}}$

验证

解: $[A]_{\text{补}} = 1, 0111$

$+ [B]_{\text{补}} = 1, 1011$

$[A]_{\text{补}} + [B]_{\text{补}} = 11, 0010 = [A + B]_{\text{补}}$

$\therefore A + B = -1110$

$$\begin{array}{r} -1001 \\ + -0101 \\ \hline -1110 \end{array}$$

例 6.20 设机器数字长为 8 位（含 1 位符号位） **6.3**
 且 $A = 15$, $B = 24$, 用补码求 $A - B$

解: $A = 15 = 0001111$

$B = 24 = 0011000$

$[A]_{\text{补}} = 0, 0001111$ $[B]_{\text{补}} = 0, 0011000$

$+ [-B]_{\text{补}} = 1, 1101000$

$[A]_{\text{补}} + [-B]_{\text{补}} = 1, 1110111 = [A - B]_{\text{补}}$

$\therefore A - B = -1001 = -9$

练习 1 设 $x = \frac{9}{16}$ $y = \frac{11}{16}$, 用补码求 $x+y$

$x + y = -0.1100 = -\frac{12}{16}$ **错**

练习 2 设机器数字长为 8 位（含 1 位符号位）
 且 $A = -97$, $B = +41$, 用补码求 $A - B$

$A - B = +1110110 = +118$ **错**

3. 溢出判断

6.3

(1) 一位符号位判溢出

参加操作的 **两个数**（减法时即为被减数和“求补”以后的减数）**符号相同，其结果的符号与原操作数的符号不同，即为溢出**

硬件实现

最高有效位的进位 \oplus 符号位的进位 = 1 溢出

如

$1 \oplus 0 = 1$	} 有 溢出
$0 \oplus 1 = 1$	
$0 \oplus 0 = 0$	} 无 溢出
$1 \oplus 1 = 0$	

(2) 两位符号位判溢出

6.3

$$[x]_{\text{补}'} = \begin{cases} x & 1 > x \geq 0 \\ 4 + x & 0 > x \geq -1 \pmod{4} \end{cases}$$

$$[x]_{\text{补}'} + [y]_{\text{补}'} = [x + y]_{\text{补}'} \pmod{4}$$

$$[x - y]_{\text{补}'} = [x]_{\text{补}'} + [-y]_{\text{补}'} \pmod{4}$$

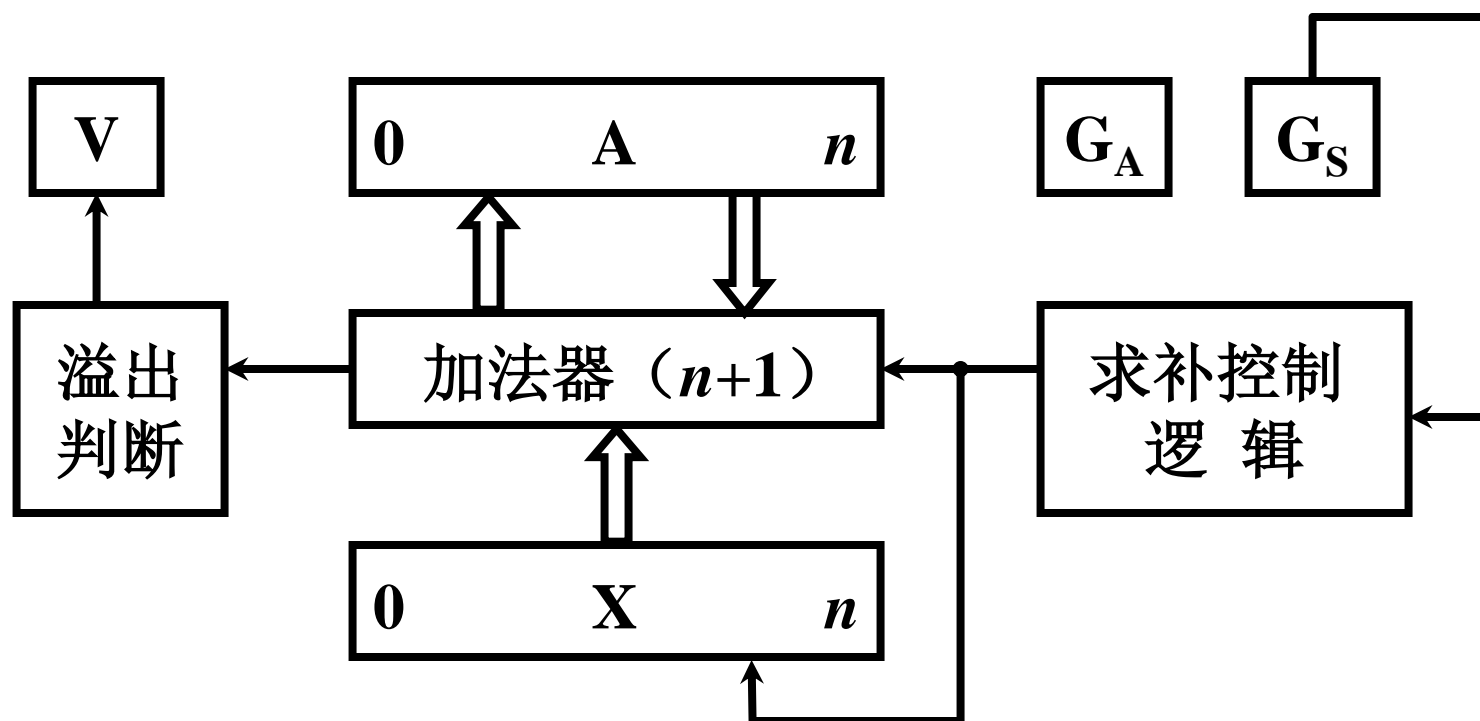
结果的双符号位 **相同** **未溢出** **00**, ×××××
11, ×××××

结果的双符号位 **不同** **溢出** **10**, ×××××
01, ×××××

最高符号位 代表其 **真正的符号**

4. 补码加减法的硬件配置

6.3



A、**X** 均 $n+1$ 位

用减法标记 **G_S** 控制求补逻辑

6.3 定点运算

- 一、移位运算
- 二、加减法运算
- 三、乘法运算
- 四、除法运算

6.3 定点运算

- 三、乘法运算
 - 计算机中怎么做二进制的乘法运算呢
 - 可以分析一下笔算乘法是怎么做的
 - 笔算乘法的分析
 - 笔算乘法的改进
 - 原码的乘法运算
 - 补码的乘法运算

三、乘法运算

6.3

1. 分析笔算乘法

$$A = -0.1101 \quad B = 0.1011$$

$$A \times B = -0.10001111 \quad \text{乘积的符号心算求得}$$

0.1101	
× 0.1011	✓ 符号位单独处理
<hr/>	
1101	✓ 乘数的某一位决定是否加被乘数
1101	
0000	? 4个位积一起相加
1101	
<hr/>	
0.10001111	✓ 乘积的位数扩大一倍

2. 笔算乘法改进

6.3

$$A \cdot B = A \cdot 0.1011$$

$$= 0.1A + 0.00A + 0.001A + 0.0001A$$

$$= 0.1A + 0.00A + 0.001(A + 0.1A)$$

$$= 0.1A + 0.01[0 \cdot A + 0.1(A + 0.1A)]$$

$$= 0.1\{A + 0.1[0 \cdot A + 0.1(A + 0.1A)]\}$$

右移一位

$$= 2^{-1}\{1 \cdot A + 2^{-1}[0 \cdot A + 2^{-1}(1 \cdot A + 2^{-1}(1 \cdot A + 0))]\}$$

第一步 被乘数 $A + 0$

第二步 右移一位，得新的部分积

第三步 部分积 + 被乘数

⋮

第八步 右移一位，得结果

⑧

①

②

③

3. 改进后的笔算乘法过程（竖式） 6.3

部 分 积	乘 数	说 明
<div>0.0000</div> <div>+ 0.1101</div>	<div>1011</div> <div>=</div>	初态，部分积 = 0 乘数为 1，加被乘数
<div>0.1101</div> <div>0.0110</div> <div>+ 0.1101</div>	<div>1101</div> <div>=</div>	→1，形成新的部分积 乘数为 1，加被乘数
<div>1.0011</div> <div>0.1001</div> <div>+ 0.0000</div>	<div>1</div> <div>1110</div> <div>=</div>	→1，形成新的部分积 乘数为 0，加 0
<div>0.1001</div> <div>0.0100</div> <div>+ 0.1101</div>	<div>11</div> <div>1111</div> <div>=</div>	→1，形成新的部分积 乘数为 1，加被乘数
<div>1.0001</div> <div>0.1000</div>	<div>111</div> <div>1111</div>	→1，得结果

小结

6.3

- 乘法 运算可用 加和移位实现
 $n = 4$, 加 4 次, 移 4 次
 - 由乘数的末位决定被乘数是否与原部分积相加,
然后 \rightarrow 1 位形成新的部分积, 同时 乘数 \rightarrow 1 位
(末位移丢), 空出高位存放部分积的低位。
 - 被乘数只与部分积的高位相加
- 硬件 3 个寄存器, 其中2个具有移位功能
 1 个全加器

6.3 定点运算

- 三、乘法运算

- 计算机中怎么做二进制的乘法运算呢

- 可以分析一下笔算乘法是怎么做的

- 笔算乘法的分析

- 笔算乘法的改进

- 原码的乘法运算

- 补码的乘法运算

运算规则

递推公式

举例

硬件配置

4. 原码乘法

6.3

(1) 原码一位乘运算规则

以小数为例

$$\text{设}[x]_{\text{原}} = x_0.x_1x_2 \cdots x_n$$

$$[y]_{\text{原}} = y_0.y_1y_2 \cdots y_n$$

$$\begin{aligned}[x \cdot y]_{\text{原}} &= (x_0 \oplus y_0).(0.x_1x_2 \cdots x_n)(0.y_1y_2 \cdots y_n) \\ &= (x_0 \oplus y_0).x^*y^*\end{aligned}$$

式中 $x^* = 0.x_1x_2 \cdots x_n$ 为 x 的绝对值

$y^* = 0.y_1y_2 \cdots y_n$ 为 y 的绝对值

乘积的符号位单独处理 $x_0 \oplus y_0$

数值部分为绝对值相乘 $x^* \cdot y^*$

(2) 原码一位乘递推公式

6.3

$$x^* \cdot y^* = x^*(0.y_1y_2 \dots y_n)$$

$$= x^*(y_12^{-1} + y_22^{-2} + \dots + y_n2^{-n})$$

$$= 2^{-1}(y_1x^* + \underbrace{2^{-1}(y_2x^* + \dots 2^{-1}(y_nx^* + \underbrace{0}_{z_0}) \dots)}_{z_1} \dots)_{z_n}$$

$$z_0 = 0$$

$$z_1 = 2^{-1}(y_nx^* + z_0)$$

$$z_2 = 2^{-1}(y_{n-1}x^* + z_1)$$

\vdots

$$z_n = 2^{-1}(y_1x^* + z_{n-1})$$

例6.21 已知 $x = -0.1110$ $y = 0.1101$ 求 $[x \cdot y]_{\text{原}}$ 6.3

解：数值部分的运算

部分积		乘数	说明
0.0000		110 <u>1</u>	部分积 初态 $z_0 = 0$
+ 0.1110			+ x^*
逻辑右移	0.1110	011 <u>0</u>	$\rightarrow 1$, 得 z_1
	0.0111		
+ 0.0000			+ 0
逻辑右移	0.0111	0	$\rightarrow 1$, 得 z_2
	0.0011		
+ 0.1110		101 <u>1</u>	+ x^*
逻辑右移	1.0001	10	$\rightarrow 1$, 得 z_3
	0.1000		
+ 0.1110		110 <u>1</u>	+ x^*
逻辑右移	1.0110	110	$\rightarrow 1$, 得 z_4
	0.1011		

例6.21 结果

6.3

① 乘积的符号位 $x_0 \oplus y_0 = 1 \oplus 0 = 1$

② 数值部分按绝对值相乘

$$x^* \cdot y^* = 0.10110110$$

$$\text{则 } [x \cdot y]_{\text{原}} = 1.10110110$$

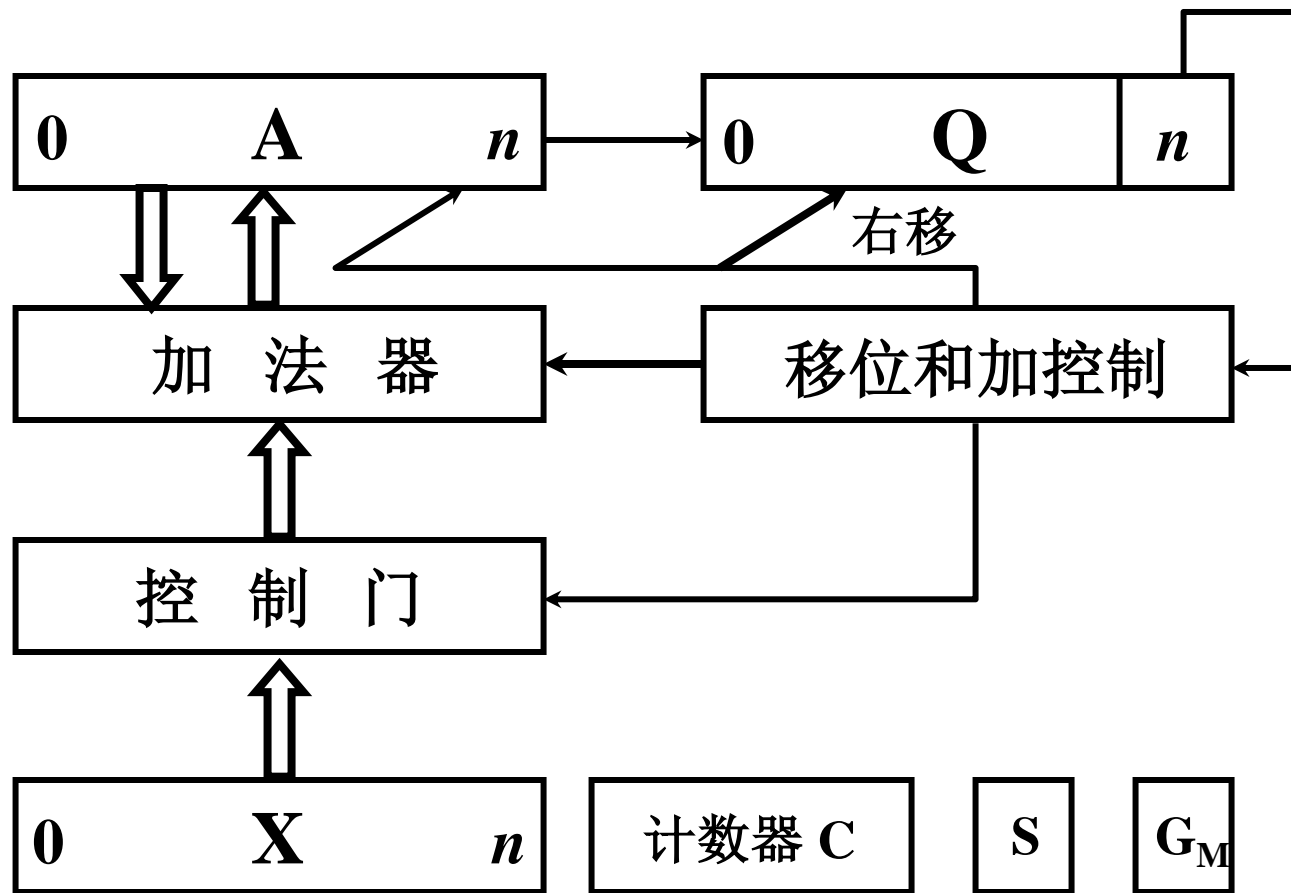
特点 绝对值运算

用移位的次数判断乘法是否结束

逻辑移位

(3) 原码一位乘的硬件配置

6.3



A、X、Q 均 $n+1$ 位

移位和加受末位乘数控制

6.3 定点运算

- 一、移位运算
- 二、加减法运算
- 三、乘法运算
- 四、除法运算

6.3 定点运算

- 三、乘法运算
 - 1. 分析笔算乘法
 - 2. 笔算乘法的改进
 - 3. 改进后的乘法笔算过程
 - 4. 原码一位乘
 - 5. 补码一位乘

5. 补码乘法

6.3

(1) 补码一位乘运算规则

以小数为例 设 被乘数 $[x]_{\text{补}} = x_0 \cdot x_1 x_2 \cdots x_n$
乘数 $[y]_{\text{补}} = y_0 \cdot y_1 y_2 \cdots y_n$

① 被乘数任意，乘数为正

与原码乘相似 但 加 和 移位 按 补码规则 运算
乘积的符号自然形成

② 被乘数任意，乘数为负

乘数 $[y]_{\text{补}}$ ，去掉符号位，操作同 ①

最后 加 $[-x]_{\text{补}}$ ，校正

③ Booth 算法（被乘数、乘数符号任意） 6.3

设 $[x]_{\text{补}} = x_0 \cdot x_1 x_2 \cdots x_n$ $[y]_{\text{补}} = y_0 \cdot y_1 y_2 \cdots y_n$

$[x \cdot y]_{\text{补}}$

$$-[x]_{\text{补}} = +[-x]_{\text{补}}$$

$$= [x]_{\text{补}} (0 \cdot y_1 \cdots y_n) - [x]_{\text{补}} \cdot y_0$$

$$= [x]_{\text{补}} (y_1 2^{-1} + y_2 2^{-2} + \cdots + y_n 2^{-n}) - [x]_{\text{补}} \cdot y_0$$

$$2^{-1} = 2^0 - 2^{-1}$$

$$= [x]_{\text{补}} (-y_0 + y_1 2^{-1} + y_2 2^{-2} + \cdots + y_n 2^{-n})$$

$$2^{-2} = 2^{-1} - 2^{-2}$$

$$= [x]_{\text{补}} [-y_0 + (y_1 - y_1 2^{-1}) + (y_2 2^{-1} - y_2 2^{-2}) + \cdots + (y_n 2^{-(n-1)} - y_n 2^{-n})]$$

$$= [x]_{\text{补}} [(y_1 - y_0) + (y_2 - y_1) 2^{-1} + \cdots + (y_n - y_{n-1}) 2^{-(n-1)} + (0 - y_n) 2^{-n}]$$

$$= [x]_{\text{补}} [(y_1 - y_0) + (y_2 - y_1) 2^{-1} + \cdots + (y_{n+1} - y_n) 2^{-n}]$$

附加位 y_{n+1}

$$y'_1 2^{-1} + \cdots + y'_n 2^{-n}$$

$$= y'_0 [x]_{\text{补}} + 2^{-1} (y'_1 [x]_{\text{补}} + 2^{-1} (y'_2 [x]_{\text{补}} + \cdots 2^{-1} (y'_n [x]_{\text{补}} + 0) \cdots))$$

④ Booth 算法递推公式

6.3

$$[z_0]_{\text{补}} = 0$$

$$[z_1]_{\text{补}} = 2^{-1}\{(y_{n+1}-y_n)[x]_{\text{补}} + [z_0]_{\text{补}}\} \quad y_{n+1} = 0$$

⋮

$$[z_n]_{\text{补}} = 2^{-1}\{(y_2-y_1)[x]_{\text{补}} + [z_{n-1}]_{\text{补}}\}$$

$$[x \cdot y]_{\text{补}} = (y_1-y_0)[x]_{\text{补}} + [z_n]_{\text{补}} \qquad \text{最后一步不移位}$$

如何实现

$+(y_{i+1}-y_i)[x]_{\text{补}} \text{ ?}$

y_i	y_{i+1}	$y_{i+1}-y_i$	操作
0	0	0	$\rightarrow 1$
0	1	1	$+ [x]_{\text{补}} \rightarrow 1$
1	0	-1	$+ [-x]_{\text{补}} \rightarrow 1$
1	1	0	$\rightarrow 1$

例6.23 已知 $x = +0.0011$ $y = -0.1011$ 求 $[x\ y]_{\text{补}}$ 6.3

解:

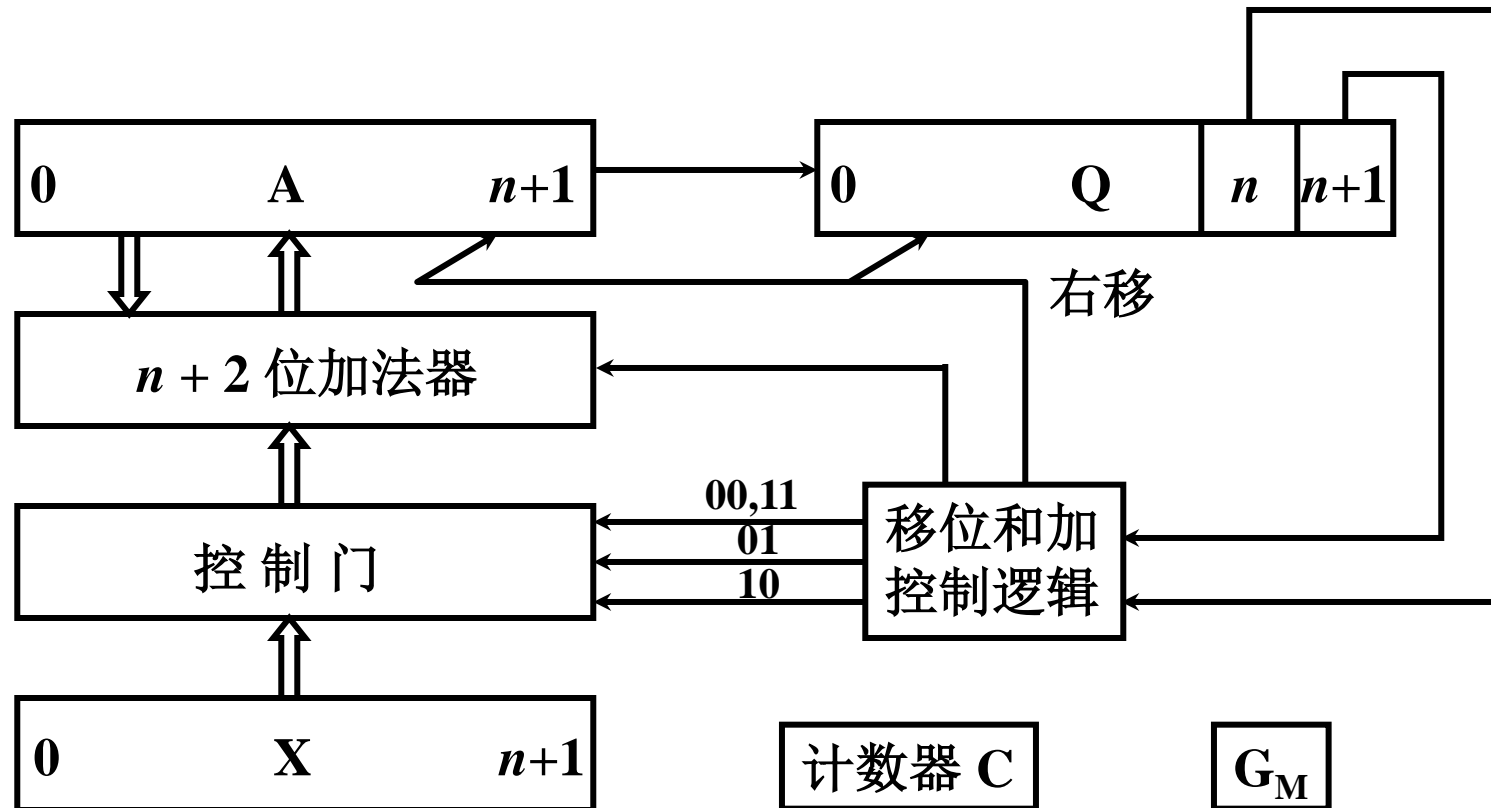
0 0 . 0 0 0 0	1 . 0 1 0 <u>1</u> <u>0</u>	0	$+[-x]_{\text{补}}$
+ 1 1 . 1 1 0 1			
1 1 . 1 1 0 1			
补码右移 1 1 . <u>1</u> 1 1 0	1	1 0 1 0 <u>1</u>	$\rightarrow 1$
+ 0 0 . 0 0 1 1			$+ [x]_{\text{补}}$
0 0 . 0 0 0 1	1		
补码右移 0 0 . <u>0</u> 0 0 0	1 1	1 0 <u>1</u> <u>0</u>	$\rightarrow 1$
+ 1 1 . 1 1 0 1			$+ [-x]_{\text{补}}$
1 1 . 1 1 0 1	1 1		
补码右移 1 1 . <u>1</u> 1 1 0	1 1 1	1 <u>0</u> <u>1</u>	$\rightarrow 1$
+ 0 0 . 0 0 1 1			$+ [x]_{\text{补}}$
0 0 . 0 0 0 1	1 1 1		
补码右移 0 0 . <u>0</u> 0 0 0	1 1 1 1	<u>1</u> <u>0</u>	$\rightarrow 1$
+ 1 1 . 1 1 0 1			$+ [-x]_{\text{补}}$
1 1 . 1 1 0 1	1 1 1 1		最后一步不移位

$[x]_{\text{补}} = 0.0011$
 $[y]_{\text{补}} = 1.0101$
 $[-x]_{\text{补}} = 1.1101$

$\therefore [x\ y]_{\text{补}} = 1.11011111$

(2) Booth 算法的硬件配置

6.3



A、X、Q 均 $n+2$ 位

移位和加法操作受乘数末两位控制

乘法小结

6.3

- 整数乘法与小数乘法过程完全相同
可用 逗号 代替小数点
- 原码乘 符号位 单独处理
补码乘 符号位 自然形成
- 原码乘去掉符号位运算 即为无符号数乘法
- 不同的乘法运算需有不同的硬件支持

6.3 定点运算

- 一、移位运算
- 二、加减法运算
- 三、乘法运算
- 四、除法运算

6.3 定点运算

- 四、除法运算
 - 1. 笔算除法是怎么做的
 - 2. 如何用计算机硬件来模拟笔算除法的过程
 - 恢复余数法
 - 加减交替法

四、除法运算

6.3

1. 分析笔算除法

$$x = -0.1011 \quad y = 0.1101 \quad \text{求 } x \div y$$

$$\begin{array}{r} 0.1101 \overline{) 0.1101} \\ 0.1101 \\ \hline 0.0000 \\ 0.0000 \\ \hline 0.0000 \\ 0.0000 \\ \hline 0.0000 \\ 0.0000 \\ \hline 0.0000 \end{array}$$

✓ 商符单独处理

? 心算上商

? 余数不动低位补“0”
减右移一位的除数

? 上商位置不固定

$$x \div y = -0.1101 \quad \text{商符心算求得}$$

$$\text{余数 } 0.00000111$$

2. 笔算除法和机器除法的比较

6.3

笔算除法

商符单独处理

心算上商

余数 **不动** 低位补“0”
减右移一位 的除数

2 倍字长加法器

上商位置 **不固定**

机器除法

符号位异或形成

$|x| - |y| > 0$ 上商 1

$|x| - |y| < 0$ 上商 0

余数 **左移一位** 低位补“0”
减 除数

1 倍字长加法器

在寄存器 **最末位**上商

3. 原码除法

6.3

以小数为例

$$[x]_{\text{原}} = x_0.x_1x_2 \dots x_n$$

$$[y]_{\text{原}} = y_0.y_1y_2 \dots y_n$$

$$[\frac{x}{y}]_{\text{原}} = (x_0 \oplus y_0). \frac{x^*}{y^*}$$

式中 $x^* = 0.x_1x_2 \dots x_n$ 为 x 的绝对值
 $y^* = 0.y_1y_2 \dots y_n$ 为 y 的绝对值

商的符号位单独处理 $x_0 \oplus y_0$

数值部分为绝对值相除 $\frac{x^*}{y^*}$

约定 小数定点除法 $x^* < y^*$ 整数定点除法 $x^* > y^*$

被除数不等于 0

除数不能为 0

(1) 恢复余数法

6.3

例6.24 $x = -0.1011$ $y = -0.1101$ 求 $[\frac{x}{y}]_{\text{原}}$

解: $[x]_{\text{原}} = 1.1011$ $[y]_{\text{原}} = 1.1101$ $[y^*]_{\text{补}} = 0.1101$ $[-y^*]_{\text{补}} = 1.0011$

① $x_0 \oplus y_0 = 1 \oplus 1 = 0$

② 被除数 (余数)	商	说 明
0.1011	0.0000	
+ 1.0011		$+[-y^*]_{\text{补}}$
1.1110	0	余数为负, 上商 0
+ 0.1101		恢复余数 $+ [y^*]_{\text{补}}$
0.1011	0	恢复后的余数
逻辑左移 1.0110	0	$\leftarrow 1$
+ 1.0011		$+ [-y^*]_{\text{补}}$
0.1001	01	余数为正, 上商 1
逻辑左移 1.0010	01	$\leftarrow 1$
+ 1.0011		$+ [-y^*]_{\text{补}}$

6.3

被除数 (余数)	商	说 明
0.0101	011	余数为正, 上商 1
逻辑左移 0.1010	011	← 1
+ 1.0011		+ $[-y^*]_{\text{补}}$
1.1101	0110	余数为负, 上商 0
+ 0.1101		恢复余数 + $[y^*]_{\text{补}}$
逻辑左移 0.1010	0110	恢复后的余数
1.0100	0110	← 1
+ 1.0011		+ $[-y^*]_{\text{补}}$
0.0111	01101	余数为正, 上商 1

$$\therefore \left[\frac{x}{y} \right]_{\text{原}} = 0.1101$$

余数为正 上商 1

余数为负 上商 0, 恢复余数

上商 5 次

第一次上商判溢出

移 4 次

(2) 不恢复余数法（加减交替法）

6.3

- 恢复余数法运算规则

余数 $R_i > 0$ 上商 “1”， $2R_i - y^*$

余数 $R_i < 0$ 上商 “0”， $R_i + y^*$ 恢复余数

$$2(R_i + y^*) - y^* = 2R_i + y^*$$

- 不恢复余数法运算规则

上商 “1” $2R_i - y^*$

上商 “0” $2R_i + y^*$

加减交替

例6.25

$x = -0.1011$
 $y = -0.1101$

求 $[\frac{x}{y}]_{\text{原}}$

6.3

解:	0.1011	0.0000	
	+1.0011		$+[-y^*]_{\text{补}}$
逻辑左移	1.1110	0	余数为负, 上商 0
	1.1100	0	$\leftarrow 1$
	+0.1101		$+ [y^*]_{\text{补}}$
逻辑左移	0.1001	01	余数为正, 上商 1
	1.0010	01	$\leftarrow 1$
	+1.0011		$+ [-y^*]_{\text{补}}$
逻辑左移	0.0101	011	余数为正, 上商 1
	0.1010	011	$\leftarrow 1$
	+1.0011		$+ [-y^*]_{\text{补}}$
逻辑左移	1.1101	0110	余数为负, 上商 0
	1.1010	0110	$\leftarrow 1$
	+0.1101		$+ [y^*]_{\text{补}}$
	0.0111	01101	余数为正, 上商 1

$[x]_{\text{原}} = 1.1011$
 $[y]_{\text{原}} = 1.1101$
 $[x^*]_{\text{补}} = 0.1011$
 $[y^*]_{\text{补}} = 0.1101$
 $[-y^*]_{\text{补}} = 1.0011$

例6.25 结果

6.3

$$\textcircled{1} x_0 \oplus y_0 = 1 \oplus 1 = 0$$

$$\textcircled{2} \frac{x^*}{y^*} = 0.1101$$

$$\therefore \left[\frac{x}{y}\right]_{\text{原}} = 0.1101$$

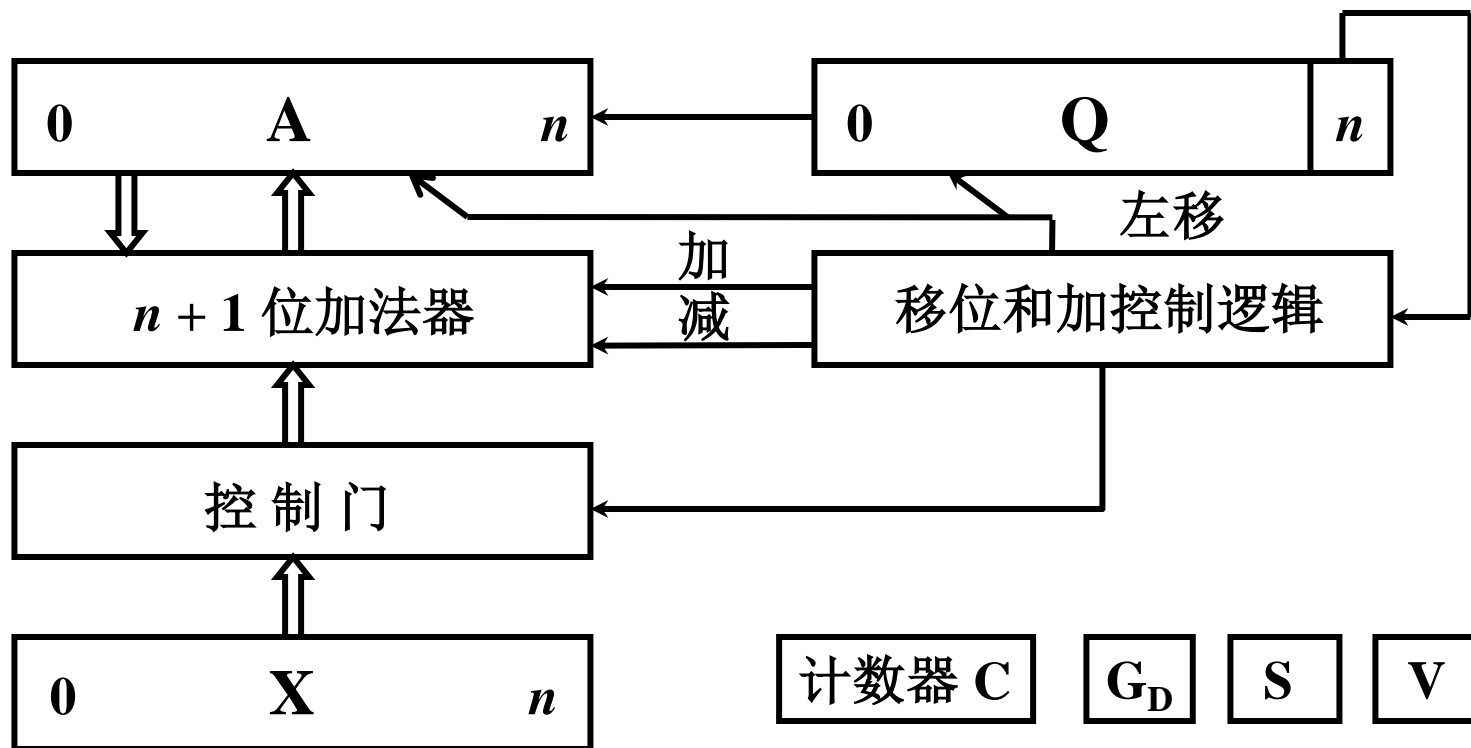
特点 上商 $n+1$ 次

第一次上商判溢出

移 n 次，加 $n+1$ 次

用移位的次数判断除法是否结束

(3) 原码加减交替除法硬件配置



A、X、Q 均 $n+1$ 位

用 Q_n 控制加减交替

第 6 章 计算机的运算方法

6.1 无符号数和有符号数

6.2 数的定点表示和浮点表示

6.3 定点运算

6.4 浮点四则运算

6.5 算术逻辑单元

6.4 浮点四则运算

- 浮点数的加减运算
 - 对阶
 - 尾数求和
 - 规格化
 - 舍入
 - 溢出判断
 - 举例
- 浮点的乘法运算
 -

6.4 浮点四则运算

一、浮点加减运算

$$x = S_x \cdot 2^{j_x} \quad y = S_y \cdot 2^{j_y}$$

1. 对阶

(1) 求阶差

$$\Delta j = j_x - j_y = \begin{cases} = 0 & j_x = j_y & \text{已对齐} \\ > 0 & j_x > j_y & \begin{cases} x \text{ 向 } y \text{ 看齐} & S_x \leftarrow 1, j_x - 1 \\ y \text{ 向 } x \text{ 看齐} & \checkmark S_y \rightarrow 1, j_y + 1 \end{cases} \\ < 0 & j_x < j_y & \begin{cases} x \text{ 向 } y \text{ 看齐} & \checkmark S_x \rightarrow 1, j_x + 1 \\ y \text{ 向 } x \text{ 看齐} & S_y \leftarrow 1, j_y - 1 \end{cases} \end{cases}$$

(2) 对阶原则

小阶向大阶看齐

例如 $x = 0.1101 \times 2^{01}$ $y = (-0.1010) \times 2^{11}$ **6.4**

求 $x + y$

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

1. 对阶

$$\textcircled{1} \text{ 求阶差 } [\Delta j]_{\text{补}} = [j_x]_{\text{补}} - [j_y]_{\text{补}} = 00, 01$$

$$\begin{array}{r} + \quad 11, 01 \\ \hline 11, 10 \end{array}$$

阶差为负 (-2) $\therefore S_x \rightarrow 2 \quad j_x + 2$

$$\textcircled{2} \text{ 对阶 } [x]_{\text{补}}' = 00, 11; 00.0011$$

2. 尾数求和

$$\begin{array}{r} [S_x]_{\text{补}}' = 00.0011 \quad \text{对阶后的}[S_x]_{\text{补}}' \\ + \quad [S_y]_{\text{补}} = 11.0110 \\ \hline 11.1001 \\ \therefore [x+y]_{\text{补}} = 00, 11; 11. 1001 \end{array}$$

3. 规格化

6.4

(1) 规格化数的定义

$$r = 2 \quad \frac{1}{2} \leq |S| < 1$$

(2) 规格化数的判断

$S > 0$	规格化形式	$S < 0$	规格化形式
真值	$0.1 \times \times \dots \times$	真值	$-0.1 \times \times \dots \times$
原码	$0.\boxed{1} \times \times \dots \times$	原码	$1.\boxed{1} \times \times \dots \times$
补码	$\boxed{0.1} \times \times \dots \times$	补码	$\boxed{1.0} \times \times \dots \times$
反码	$0.1 \times \times \dots \times$	反码	$1.0 \times \times \dots \times$

原码 不论正数、负数，第一数位为1

补码 符号位和第一数位不同

特例

6.4

$$S = -\frac{1}{2} = -0.100 \cdots 0$$

$$[S]_{\text{原}} = 1.100 \cdots 0$$

$$[S]_{\text{补}} = \boxed{1.1}00 \cdots 0$$

$\therefore [-\frac{1}{2}]_{\text{补}}$ 不是规格化的数

$$S = -1$$

$$[S]_{\text{补}} = \boxed{1.0}00 \cdots 0$$

$\therefore [-1]_{\text{补}}$ 是规格化的数

(3) 左规

尾数左移一位，阶码减 1，直到数符和第一数位不同为止

上例 $[x+y]_{\text{补}} = 00, 11; 11. 1001$

左规后 $[x+y]_{\text{补}} = 00, 10; 11. 0010$

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

(4) 右规

当 尾数溢出 (>1) 时，需 右规

即尾数出现 $01. \times \times \dots \times$ 或 $10. \times \times \dots \times$ 时

尾数右移一位，阶码加 1

例6.27 $x = 0.1101 \times 2^{10}$ $y = 0.1011 \times 2^{01}$ 6.4

求 $x+y$ (除阶符、数符外, 阶码取 3 位, 尾数取 6 位)

解: $[x]_{\text{补}} = 00, 010; 00. 110100$
 $[y]_{\text{补}} = 00, 001; 00. 101100$

① 对阶

$$[\Delta j]_{\text{补}} = [j_x]_{\text{补}} - [j_y]_{\text{补}} = \begin{array}{r} 00, 010 \\ + 11, 111 \\ \hline 100, 001 \end{array}$$

阶差为 +1 $\therefore S_y \rightarrow 1, j_y+1$

$\therefore [y]_{\text{补}}' = 00, 010; 00. 010110$

② 尾数求和

$$\begin{array}{r} [S_x]_{\text{补}} = 00. 110100 \\ + [S_y]_{\text{补}}' = 00. 010110 \\ \hline 01. 001010 \end{array} \quad \begin{array}{l} \text{对阶后的 } [S_y]_{\text{补}}' \\ \text{尾数溢出需右规} \end{array}$$

③ 右规

6.4

$$[x+y]_{\text{补}} = 00, 010; 01. 001010$$

右规后

$$[x+y]_{\text{补}} = 00, 011; 00. 100101$$

$$\therefore x+y = 0. 100101 \times 2^{11}$$

4. 舍入

在 对阶 和 右规 过程中，可能出现 尾数末位丢失
引起误差，需考虑舍入

(1) 0 舍 1 入法

(2) 恒置 “1” 法

6.4

例 6.28 $x = (-\frac{5}{8}) \times 2^{-5}$ $y = (-\frac{7}{8}) \times 2^{-4}$

求 $x-y$ (除阶符、数符外, 阶码取 3 位, 尾数取 6 位)

解: $x = (-0.101000) \times 2^{-101}$ $y = (0.111000) \times 2^{-100}$

$[x]_{\text{补}} = 11, 011; 11. 011000$ $[y]_{\text{补}} = 11, 100; 00. 111000$

① 对阶

$$\begin{array}{rcl} [\Delta j]_{\text{补}} = [j_x]_{\text{补}} - [j_y]_{\text{补}} & = & 11, 011 \\ & + & 00, 100 \\ \hline & & 11, 111 \end{array}$$

阶差为 -1 $\therefore S_x \longrightarrow 1, j_x + 1$

$\therefore [x]_{\text{补}}' = 11, 100; 11. 101100$

6.4

② 尾数求和

$$\begin{array}{r} [S_x]_{\text{补}} = 11.101100 \\ + [-S_y]_{\text{补}} = 11.001000 \\ \hline 110.110100 \end{array}$$

③ 右规

$$[x - y]_{\text{补}} = 11, 100; 10.110100$$

右规后

$$[x - y]_{\text{补}} = 11, 101; 11.011010$$

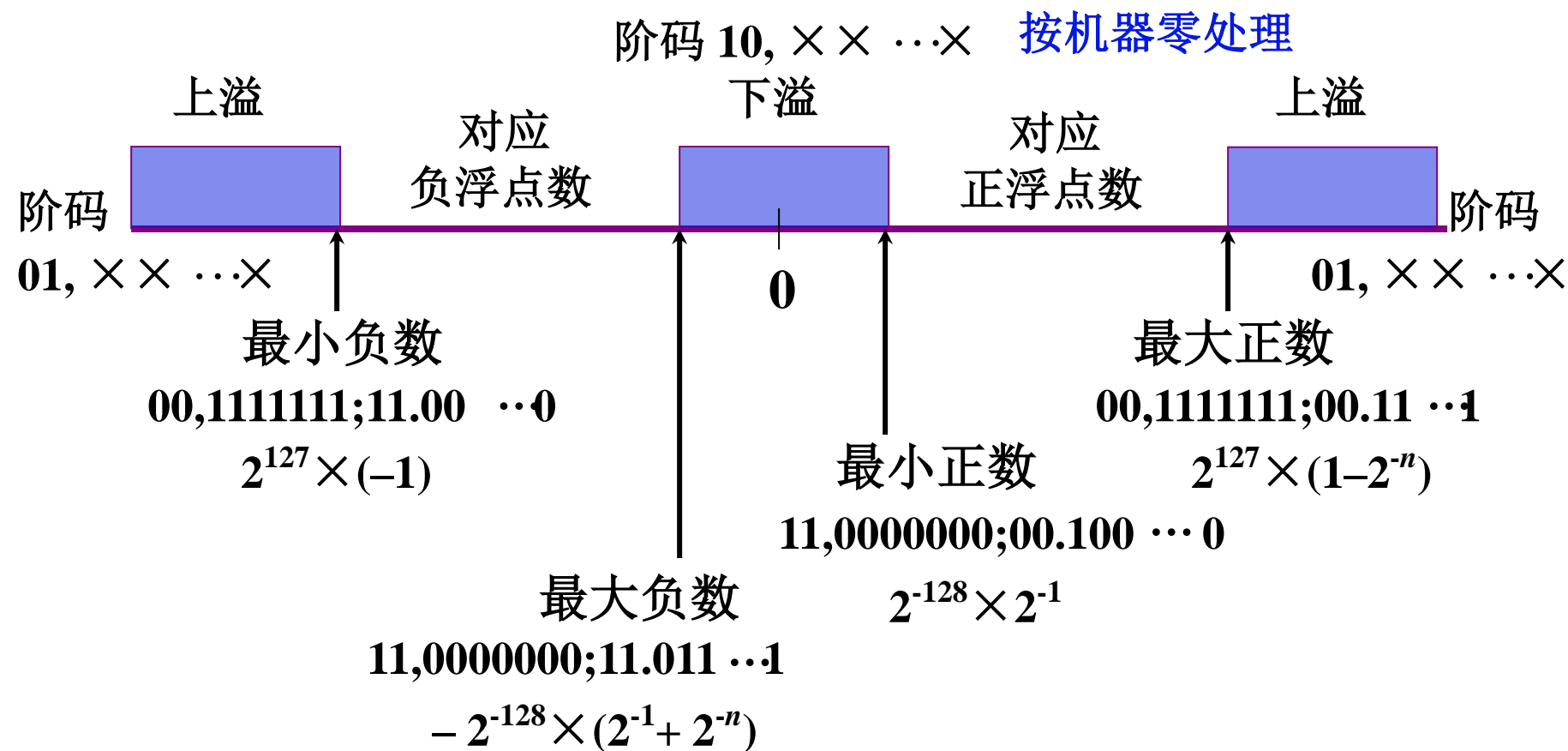
$$\therefore x - y = (-0.100110) \times 2^{-11}$$

$$= \left(-\frac{19}{32}\right) \times 2^{-3}$$

5. 溢出判断

6.4

设机器数为补码，尾数为规格化形式，并假设阶符取 2 位，阶码的数值部分取 7 位，数符取 2 位，尾数取 n 位，则该补码在数轴上的表示为



第 6 章 计算机的运算方法

6.1 无符号数和有符号数

6.2 数的定点表示和浮点表示

6.3 定点运算

6.4 浮点四则运算

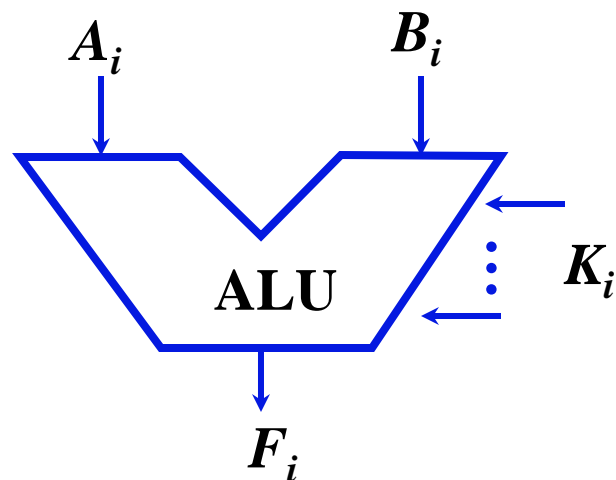
6.5 算术逻辑单元

6.5 算术逻辑单元

- 一、ALU 电路
- 二、快速进位链
 - 1. 并行加法器
 - 2. 串行进位链
 - 3. 并行进位链
 - (1) 单重分组跳跃进位链
 - (2) 双重分组跳跃进位链

6.5 算术逻辑单元

一、ALU 电路



组合逻辑电路

K_i 不同取值

F_i 不同

四位 ALU 74181

$M = 0$ 算术运算

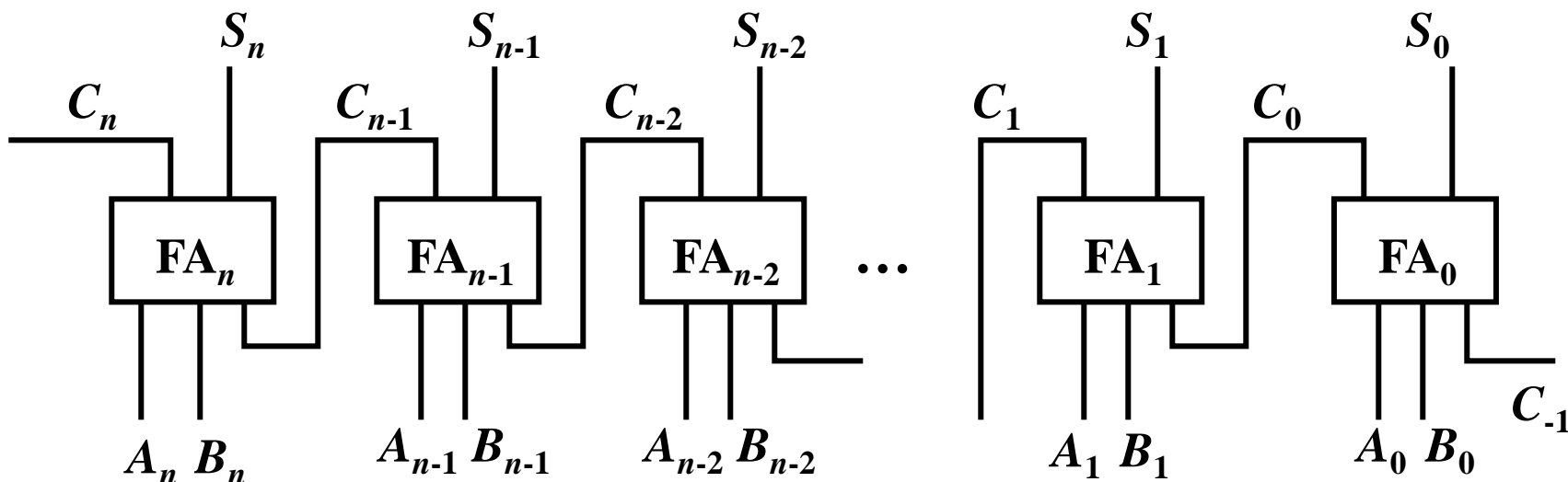
$M = 1$ 逻辑运算

$S_3 \sim S_0$ 不同取值，可做不同运算

二、快速进位链

6.5

1. 并行加法器



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

$$C_i = \overline{A_i} B_i C_{i-1} + A_i \overline{B_i} C_{i-1} + A_i B_i \overline{C_{i-1}} + A_i B_i C_{i-1}$$

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

$$d_i = A_i B_i \quad \text{本地进位} \quad t_i = A_i + B_i \quad \text{传送条件}$$

$$\text{则 } C_i = d_i + t_i C_{i-1}$$

2. 串行进位链

6.5

进位链

传送进位的电路

串行进位链

进位串行传送

以 4 位全加器为例，每一位的进位表达式为

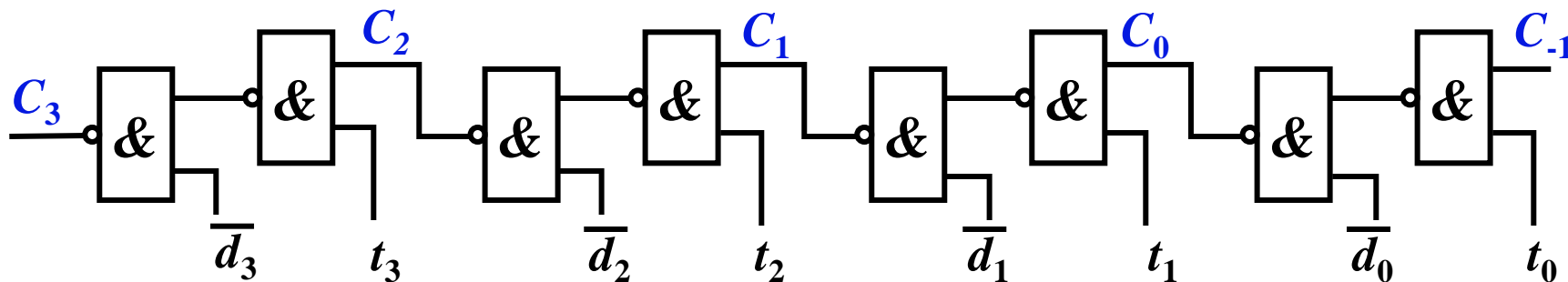
$$C_0 = d_0 + t_0 C_{-1} = \overline{\overline{d_0} \cdot \overline{t_0 C_{-1}}}$$

$$C_1 = d_1 + t_1 C_0$$

$$C_2 = d_2 + t_2 C_1$$

$$C_3 = d_3 + t_3 C_2$$

设与非门的级延迟时间为 t_y



4位全加器产生进位的全部时间为 $8t_y$

n 位全加器产生进位的全部时间为 $2nt_y$

3. 并行进位链（先行进位，跳跃进位）

6.5

n 位加法器的进位同时产生 以 4 位加法器为例

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

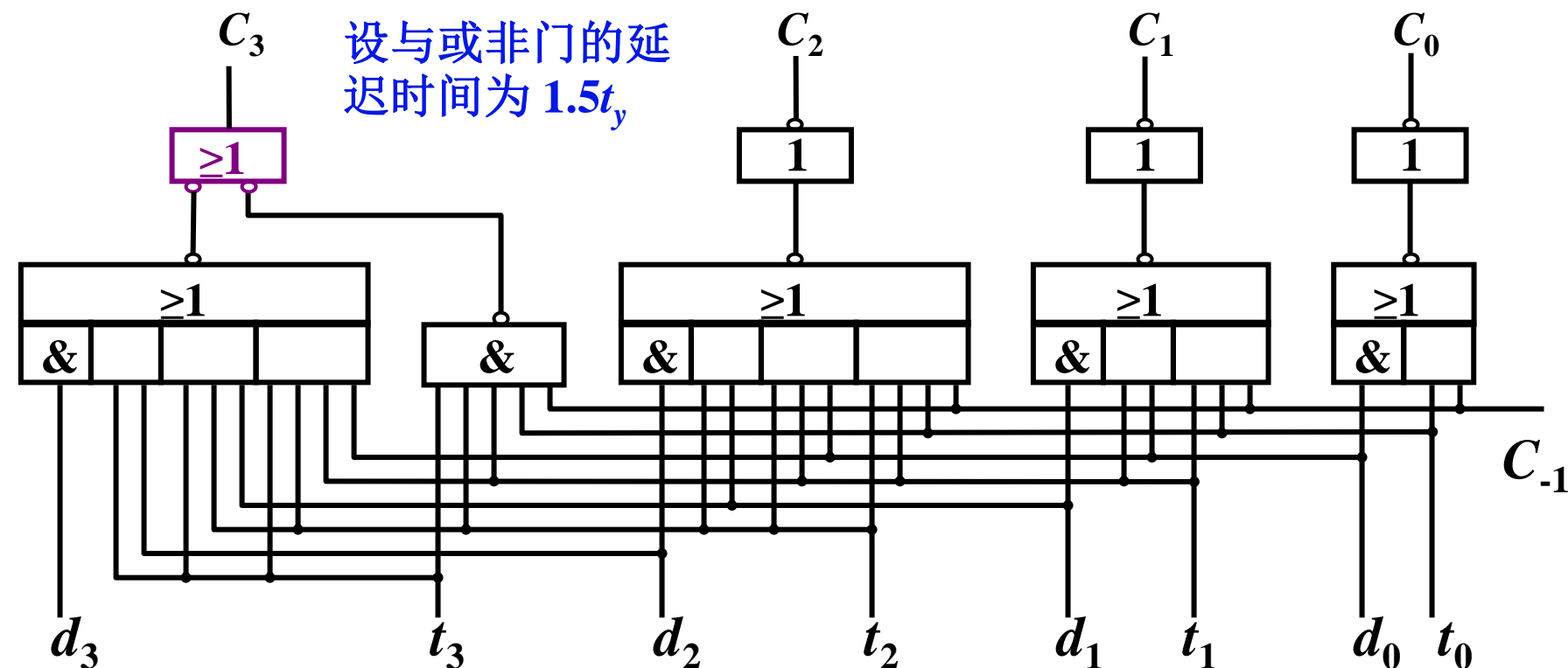
当 $d_i t_i$ 形成后，只需 $2.5t_y$

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

产生全部进位

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

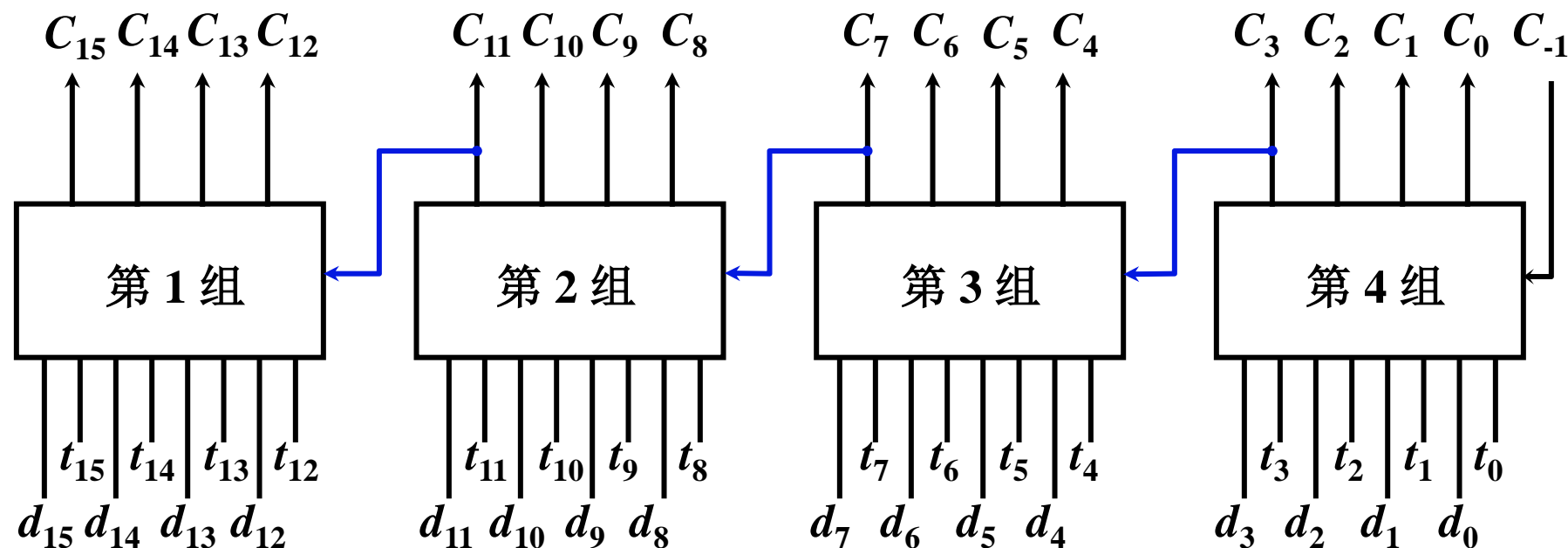
$$C_3 = d_3 + t_3 C_2 = d_3 + t_3 d_2 + t_3 t_2 d_1 + t_3 t_2 t_1 d_0 + t_3 t_2 t_1 t_0 C_{-1}$$



(1) 单重分组跳跃进位链

6.5

n 位全加器分若干小组，小组中的进位同时产生，
小组与小组之间采用串行进位 以 $n = 16$ 为例



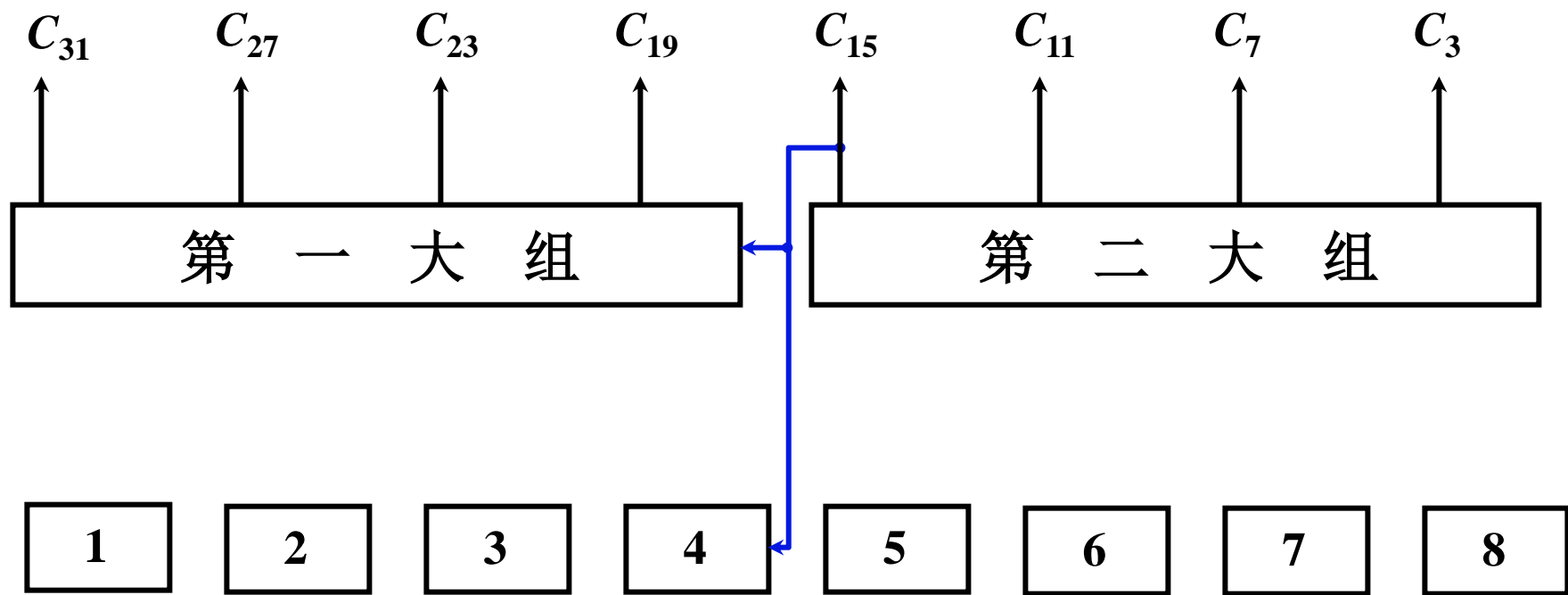
当 d_i 、 t_i 形成后	经 $2.5 t_y$	产生 $C_3 \sim C_0$
	$5 t_y$	产生 $C_7 \sim C_4$
	$7.5 t_y$	产生 $C_{11} \sim C_8$
	$10 t_y$	产生 $C_{15} \sim C_{12}$

(2) 双重分组跳跃进位链

6.5

n 位全加器分若干大组，大组中又包含若干小组。每个大组中小组的最高位进位同时产生。大组与大组之间采用串行进位。

以 $n = 32$ 为例



(3) 双重分组跳跃进位链 大组进位分析

6.5

以第 8 小组为例

$$C_3 = d_3 + t_3 C_2 = \underbrace{d_3 + t_3 d_2 + t_3 t_2 d_1 + t_3 t_2 t_1 d_0}_{D_8} + \underbrace{t_3 t_2 t_1 t_0 C_{-1}}_{T_8 C_{-1}}$$

D_8 小组的本地进位 与外来进位无关

T_8 小组的传送条件 与外来进位无关 传递外来进位

同理 第 7 小组 $C_7 = D_7 + T_7 C_3$

第 6 小组 $C_{11} = D_6 + T_6 C_7$

第 5 小组 $C_{15} = D_5 + T_5 C_{11}$

进一步展开得

$$C_3 = D_8 + T_8 C_{-1}$$

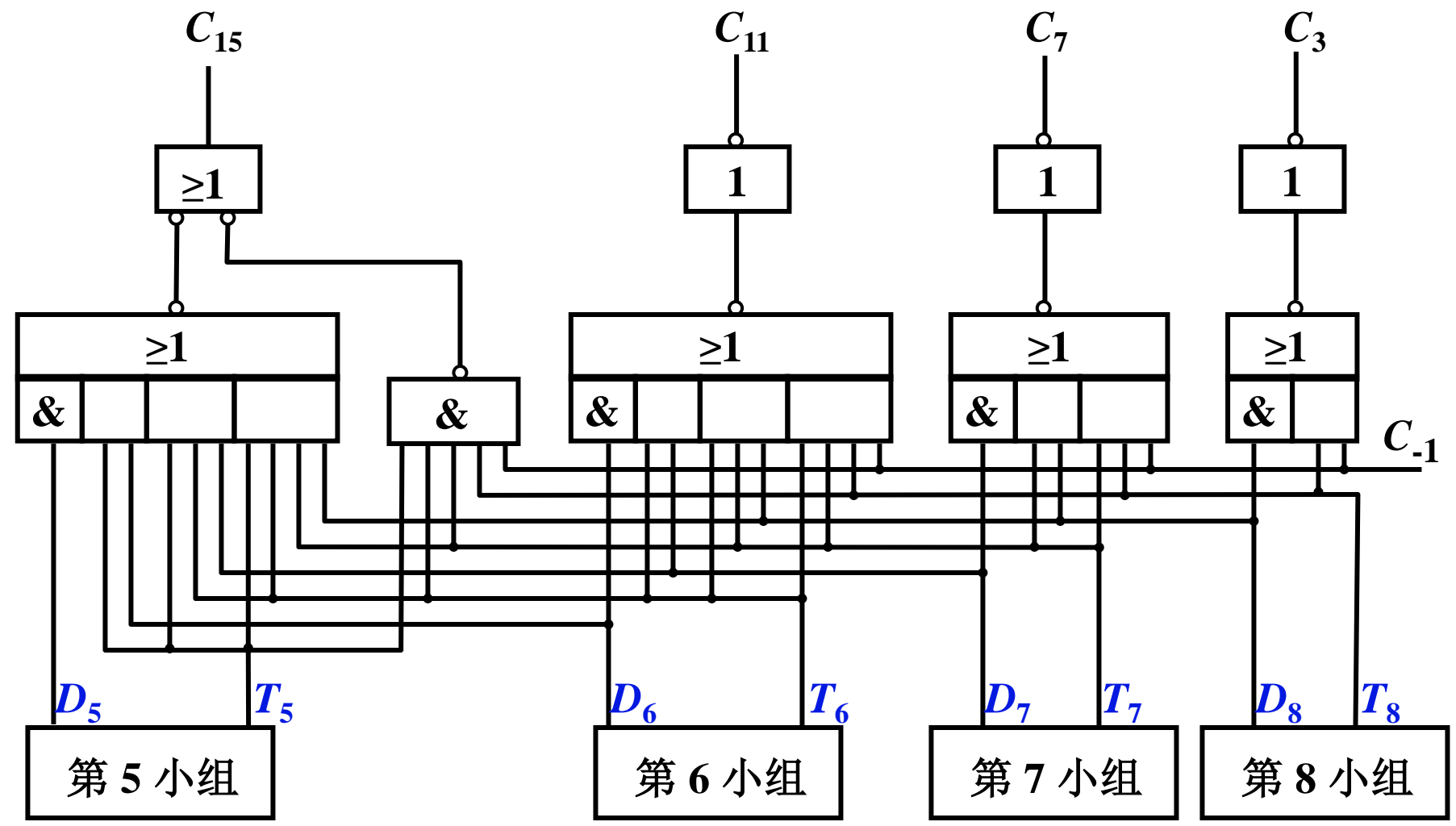
$$C_7 = D_7 + T_7 C_3 = D_7 + T_7 D_8 + T_7 T_8 C_{-1}$$

$$C_{11} = D_6 + T_6 C_7 = D_6 + T_6 D_7 + T_6 T_7 D_8 + T_6 T_7 T_8 C_{-1}$$

$$C_{15} = D_5 + T_5 C_{11} = D_5 + T_5 D_6 + T_5 T_6 D_7 + T_5 T_6 T_7 D_8 + T_5 T_6 T_7 T_8 C_{-1}$$

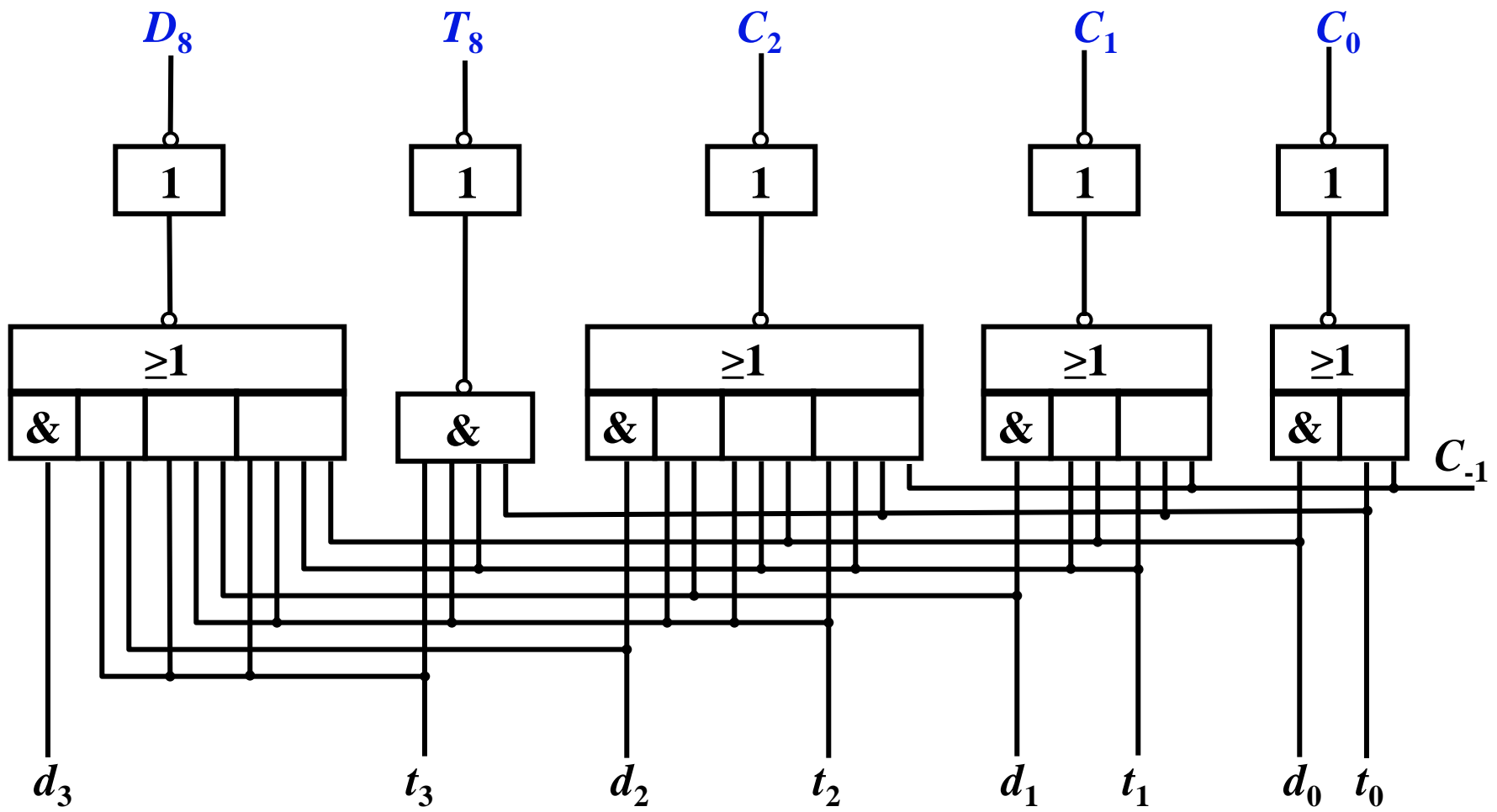
(4) 双重分组跳跃进位链的大组进位线路 6.5

以第 2 大组为例

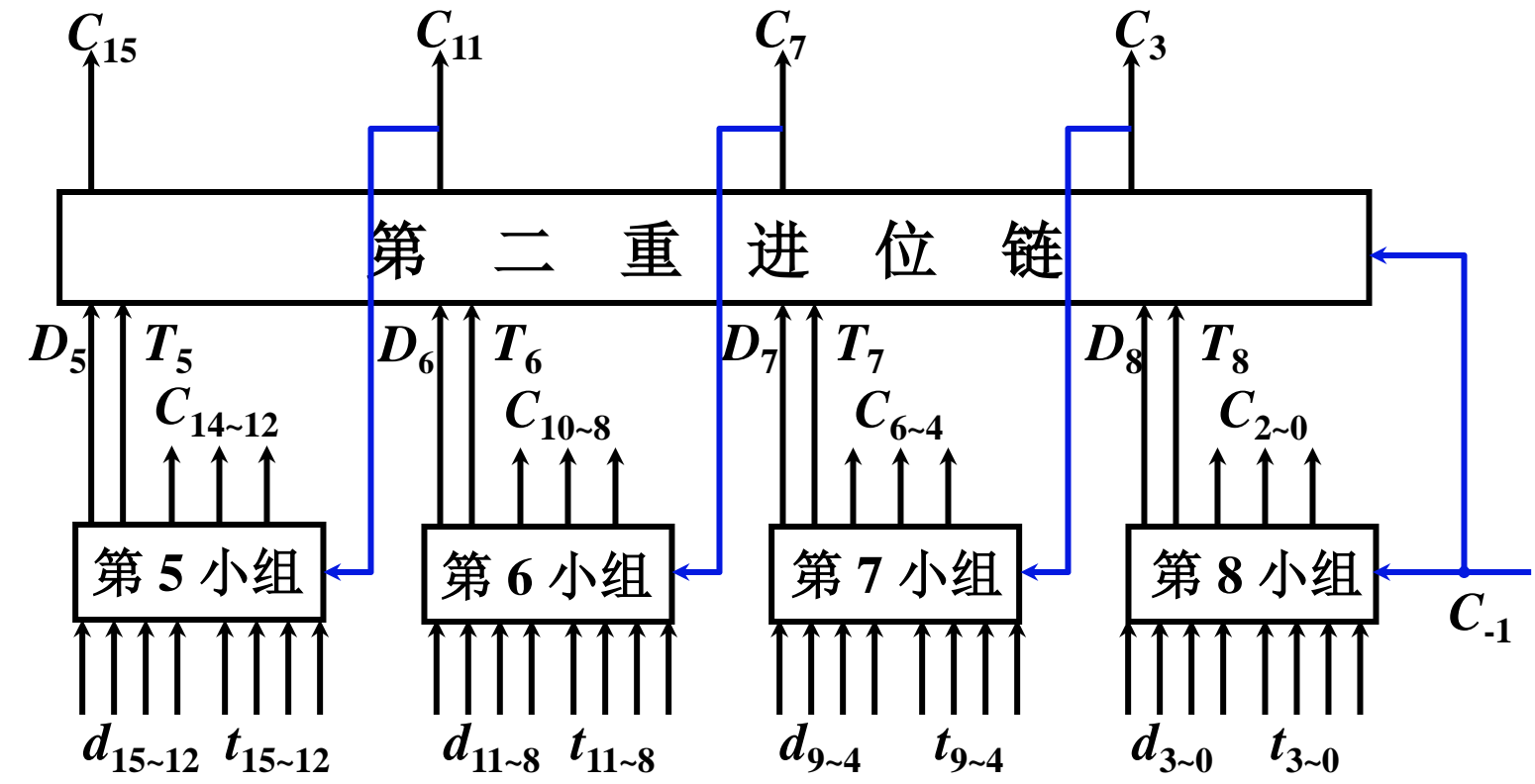


(5) 双重分组跳跃进位链的 小组 进位线路 6.5

以第 8 小组为例 只产生 低 3 位 的进位和 本小组的 $D_8 T_8$



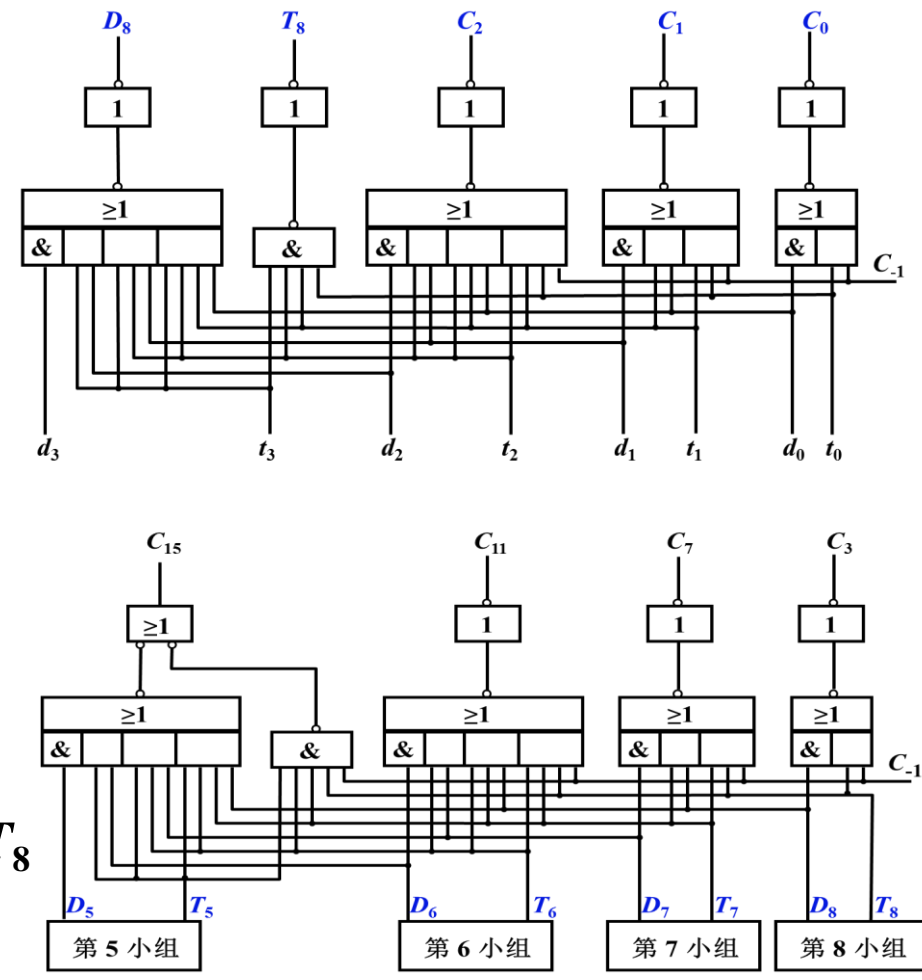
(6) $n=16$ 双重分组跳跃进位链



当 d_i 、 t_i 和 C_{-1} 形成后

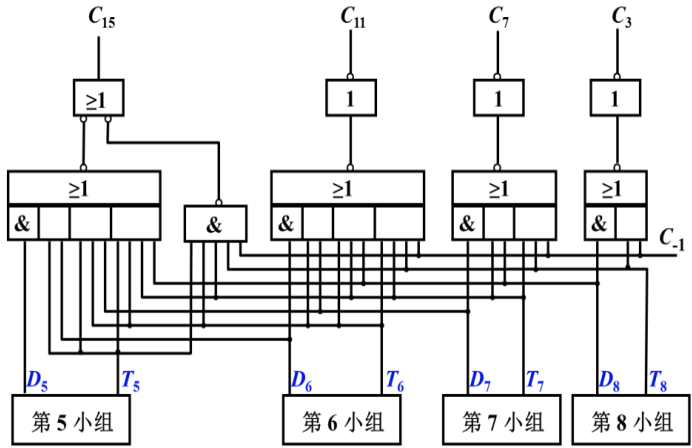
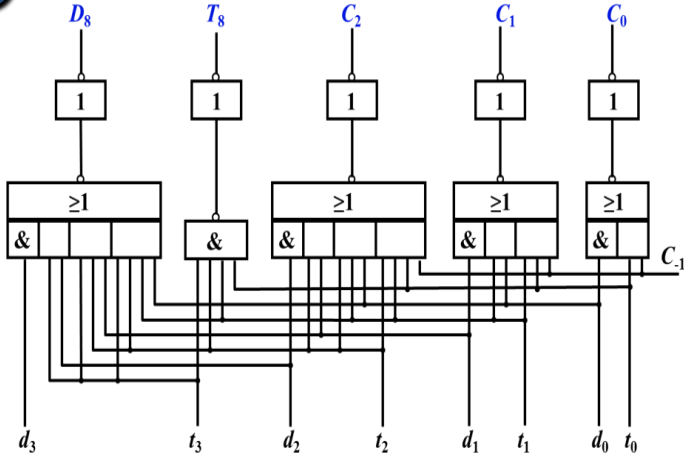
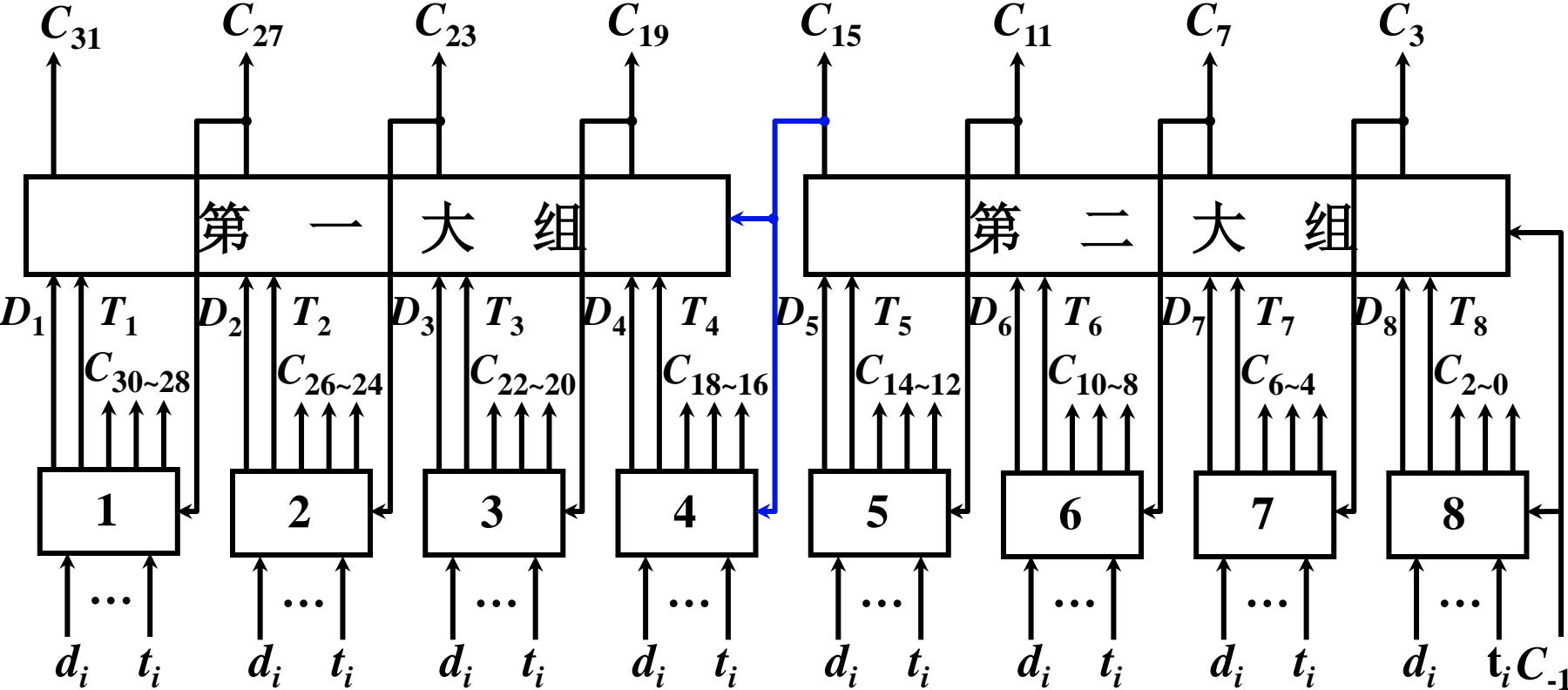
经 $2.5 t_y$	产生 C_2 、 C_1 、 C_0 、 $D_5 \sim D_8$ 、 $T_5 \sim T_8$
经 $5 t_y$	产生 C_{15} 、 C_{11} 、 C_7 、 C_3
经 $7.5 t_y$	产生 $C_{14} \sim C_{12}$ 、 $C_{10} \sim C_8$ 、 $C_6 \sim C_4$
串行进位链 经 $32 t_y$	产生 全部进位
单重分组跳跃进位链 经 $10 t_y$	产生 全部进位

6.5



(7) $n=32$ 双重分组跳跃进位链

6.5



当 d_i 、 t_i 形成后 经 $2.5 t_y$ 产生 C_2 、 C_1 、 C_0 、 $D_1 \sim D_8$ 、 $T_1 \sim T_8$
 $5 t_y$ 产生 C_{15} 、 C_{11} 、 C_7 、 C_3
 $7.5 t_y$ 产生 $C_{18} \sim C_{16}$ 、 $C_{14} \sim C_{12}$ 、 $C_{10} \sim C_8$ 、 $C_6 \sim C_4$
 C_{31} 、 C_{27} 、 C_{23} 、 C_{19}
 $10 t_y$ 产生 $C_{30} \sim C_{28}$ 、 $C_{26} \sim C_{24}$ 、 $C_{22} \sim C_{20}$