计算机
组成

第六章　计算机的运算方法

Contents

大 纲

(一)数制与编码

1. 进位计算制及其数据之间的相互转换
2. 定点数的编码表示

(二)运算方法和运算电路

1. 基本运算部件

   加法器，算术逻辑部件（ALU）

2. 加/减法运算

   补码加减法运算器，

   标志位的生成

3. 乘除法运算的基本原理，

   乘法电路和除法电路的基本结构

(三)整数的表示和运算

1. 无符号整数的表示和运算
2. 带符号整数的表示和运算

(四) 浮点数的表示和运算

1. 浮点数的表示

   IEEE754标准

2. 浮点数的加/减运算

数制与编码

1. 进位计算制及其相 互转换

1). 基r数制：用r个基本符号（如0,1,2,…,r-1）表示数值N。r为 基数

$$N=D_{m-1}D_{m-2}\ \ldots\ D_1\ D_0\ D_{-1}\ D_{-2}\ \ldots\ D_{-k}$$

其中， D_i（$-k \le i \le m-1$）为基本符号，小数点位置隐含在D_0 与D_{-1} 之间。

2). 有权基r数制：每个D_i的单位都赋以固定权值w_i，w_i为D_i的权。

如果该数制是逢r进位，则有

$$N=\sum_{i=-k}^{m-1} D_i \times r^i$$

其中r^i为位权，称该数制为r进位数制，简称r进制，

3). 进制转换

(1) R进制数 => 十进制数

按“权”展开 (a power of R)

例1: $(10101.01)_2=1\times2^4+1\times2^2+1\times2^0+1\times2^{-2}=(21.25)_{10}$

例2: $(307.6)_8=3\times8^2+7\times8^0+6\times8^{-1}=(199.75)_{10}$

例1: $(3A.1)_{16}=3\times16^1+10\times16^0+1\times16^{-1}=(58.0625)_{10}$

(2)十进制数 => R进制数

整数部分和小数部分分别转换

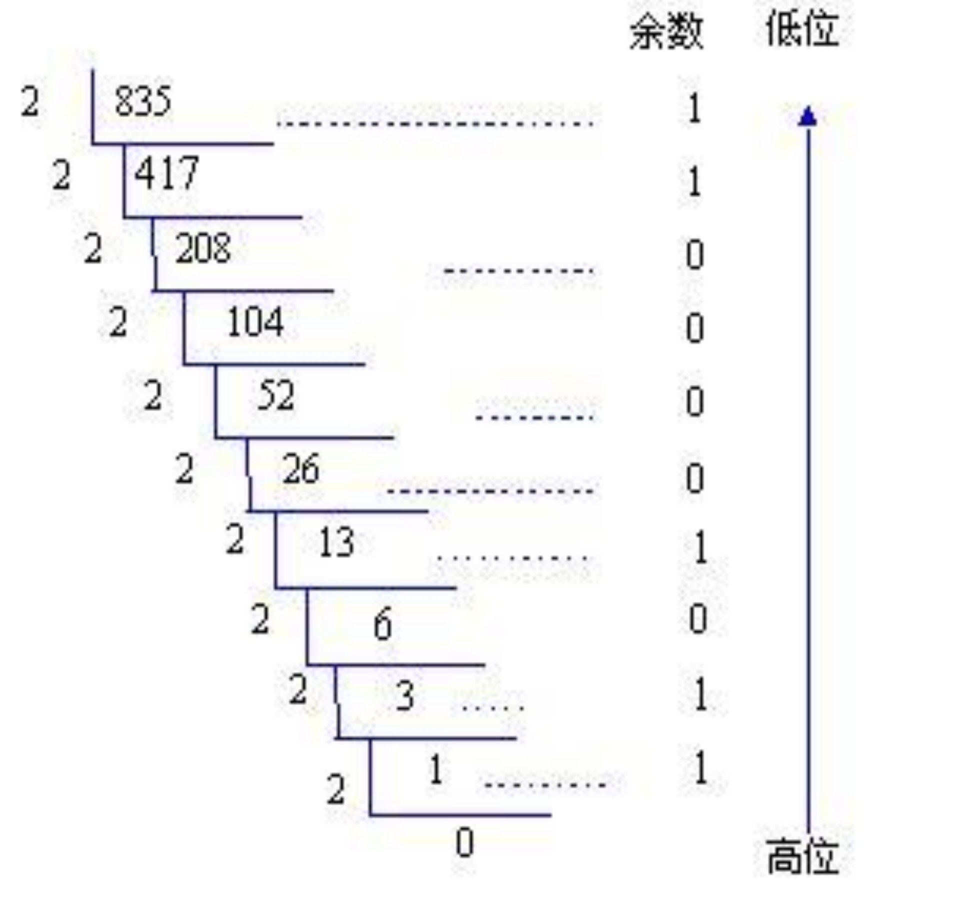
① 整数(integral part)----“除基取余，上右下左”

② 小数(fractional part)----“乘基取整，上左下右”

二、十进制转换

例1:$(835.6785)_{10}=(1101000011.1011)_2$

整数----“除基取余，上右下左”



小数----“乘基取整，上左下右”

$0.6875 \times 2 = 1.375$ 整数部分=1 (高位)

$0.375 \times 2 = 0.75$ 整数部分=0 ↓

$0.75 \times 2 = 1.5$ 整数部分=1 ↓

$0.5 \times 2 = 1.0$ 整数部分=1 (低位)

例2:$(835.63)_{10}=(1503.50243\ldots)_8$

整数----“除基取余，上右下左”小数----“乘基取整，上左下右”

有可能乘积的小数部分总得不到0 ，此时得到一个近似值。

			低位
8	835	余数	
8	104	3	
8	13	0	
8	1	5	
	0	1	高位

$0.63\times8=5.04$	整数部分=5	(高位)
$0.04\times8=0.32$	整数部分=0	
$0.32\times8=2.56$	整数部分=2	
$0.56\times8=4.48$	整数部分=4	
$0.48\times8=3.84$	整数部分=3	(低位)

(3) 二/八/十六进制数的相互转换

① 八进制数转换成二进制数

$(13.724)_8 = (\ 001\ 011\ .\ 111\ \ 010\ \ 100\ \ )_2 = (1011.1110101)_2$

② 十六进制数转换成二进制数

$(2B.5E)_{16} = (00101011\ .\ 01011110)_2 = (101011.0101111)_2$

③ 二进制数转换成八进制数

$(0.10101)_2 = (\ \ 000\ .\ 101\ \ 010\ \ )_2 = (\ 0.52)_8$

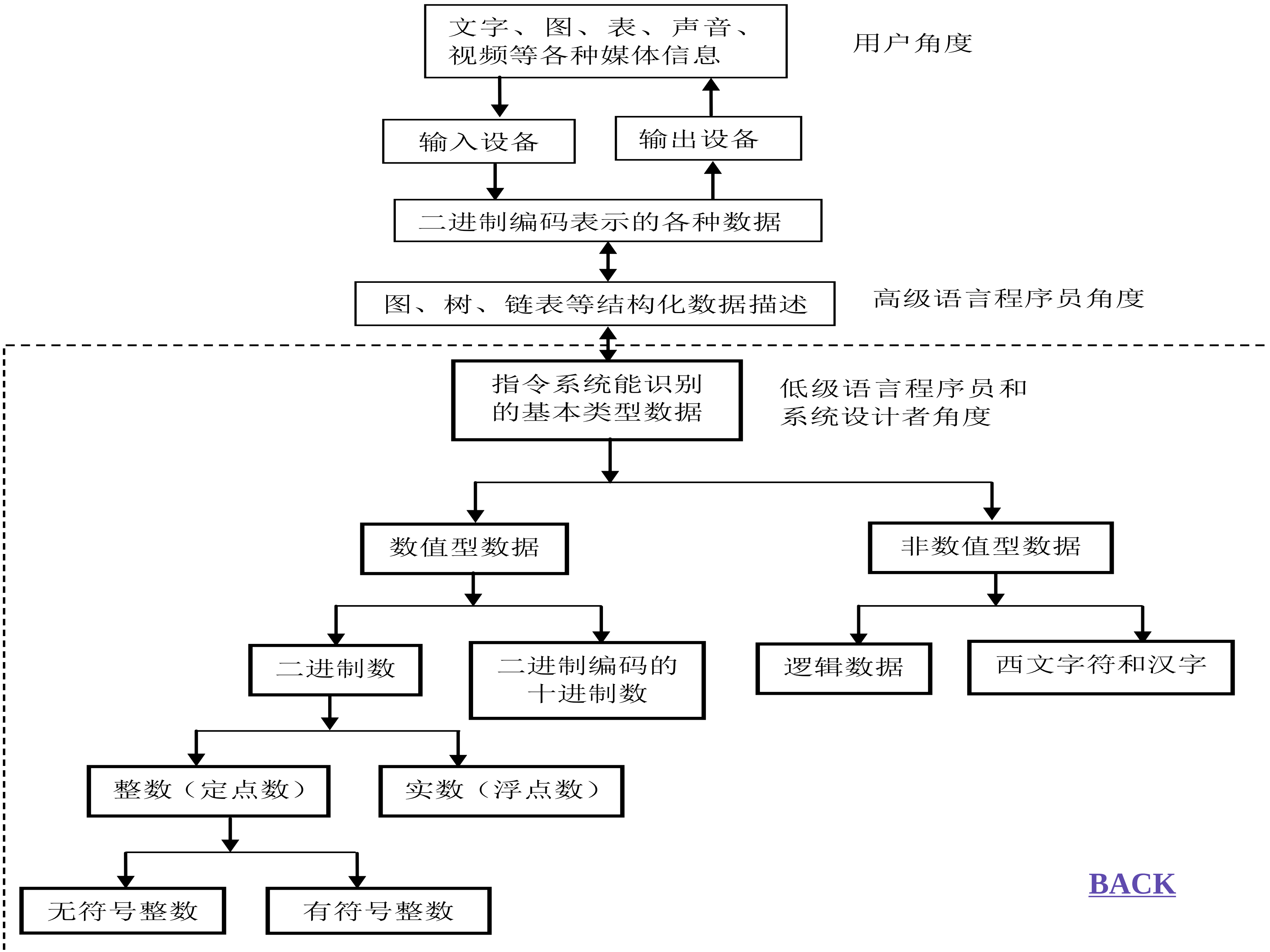
④ 二进制数转换成十六进制数

$(11001.11)_2 = (\ \ 0001\ \ 1001\ .\ 1100\ \ )_2 = (\ 19.C\ )_{16}$

信息的二进制编码

- 计算机的外部信息与内部机器级数据
- 机器级数据分两大类：
  - 数值数据：无符号整数、带符号整数、浮点数（实数）、十进制数
  - 非数值数据：逻辑数（包括位串）、西文字符和汉字
- 计算机内部所有信息都用二进制（即：0和1）进行编码
- 用二进制编码的原因：
  - 制造二个稳定态的物理器件容易
  - 二进制编码、计数、运算规则简单
  - 正好与逻辑命题对应，便于逻辑运算，并可方便地用逻辑电路实现算术运算

数值数据的表示


文字、图、表、声音、视频等各种媒体信息
用户角度
输入设备
输出设备
二进制编码表示的各种数据
图、树、链表等结构化数据描述
高级语言程序员角度
指令系统能识别的基本类型数据
低级语言程序员和系统设计者角度
数值型数据
非数值型数据
二进制数
二进制编码的十进制数
逻辑数据
西文字符和汉字
整数（定点数）
实数（浮点数）
无符号整数
有符号整数
BACK

数值数据的表示

- **数值数据表示的三要素**
  - 进位计数制
  - 定、浮点表示
  - 如何用二进制编码

  即：要确定一个数值数据的值必须先确定这三个要素。

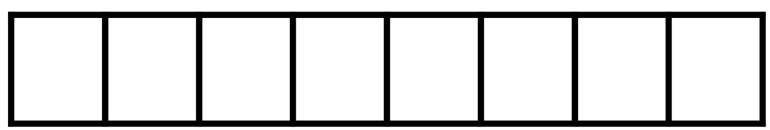
  例如，机器数 01011001的值是多少？ **答案是：不知道！**
- **进位计数制**
  - 十进制、二进制、十六进制、八进制数及其相互转换
- **定/浮点表示（解决小数点问题）**
  - 定点整数、定点小数
  - 浮点数（可用一个定点小数和一个定点整数来表示）
- **定点数的编码（解决正负号问题）**
  - 原码、补码、反码、移码 （反码很少用）

6.1 无符号数和有符号数

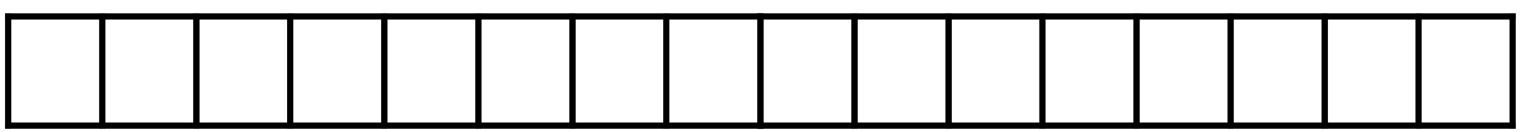
一、无符号数

寄存器的位数（机器字长）

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	0 ‖ 1011 小数点的位置
– 0.1011	1 ‖ 1011 小数点的位置
+ 1100	0 ‖ 1100 小数点的位置
– 1100	1 ‖ 1100 小数点的位置

2. 原码表示法

(1) 定义

整数

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

x 为真值　　n 为整数的位数

如　$x = +1110$　　$[x]_{原} = 0，1110$

$x = -1110$　　$[x]_{原} = 2^4 + 1110 = 1，1110$

用 逗号 将符号位和数值位隔开

带符号的绝对值表示

小数

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

x 为真值

如

$x = +0.1101$ $\quad [x]_{原} = 0\,.\,1101$

用 小数点 将符号位和数值部分隔开

$x = -0.1101$ $\quad [x]_{原} = 1 - (-0.1101) = 1\,.\,1101$

$x = +0.1000000$ $\quad [x]_{原} = 0\,.\,1000000$

用 小数点 将符号位和数值部分隔开

$x = -0.1000000$ $\quad [x]_{原} = 1 - (-0.1000000) = 1\,.\,1000000$

例 6.1 已知 $[x]_{原} = 1.\underline{0011}$ 求 x -0.0011

解：由定义得

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

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

解：由定义得

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

(2) 举例

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

解：由定义得

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

例 6.2　已知 $[x]_{原} = 1,1100$　求 x　-1100

解：由定义得

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

原码的特点：简单、直观

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

3. 补码表示法

(1) 补的概念

- 时钟　　逆时针　　顺时针

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

时钟以12为模

可见 -3 可用 $+9$ 代替　减法 ⟶ 加法

称 $+9$ 是 -3 以 12 为模的补数

记作 $-3 \equiv +9 \pmod{12}$

同理 $-4 \equiv +8 \pmod{12}$

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

结论

- 一个负数加上 "模" 即得该负数的补数
- 两个互为补数的数 它们绝对值之和即为 模 数

• 计数器（模 16） 1011 ⟶ 0000 ？

11

$$\begin{array}{r} 1011 \\ -\ 1011 \\ \hline 0000 \end{array} \qquad \begin{array}{r} 1011 \\ +\ 0101 \\ \hline \boxed{1}0000 \end{array}$$

自然去掉

可见 -1011 可用 $+0101$ 代替

记作 $-1011 \equiv +0101 \quad (\mathrm{mod}\ 2^4)$

同理 $-011 \equiv +101 \quad (\mathrm{mod}\ 2^3)$

$-0.1001 \equiv +1.0111 \quad (\mathrm{mod}\ 2)$

(2) 正数的补数即为其本身 6.1

两个互为补数的数 $\boxed{-1011 \equiv +0101}$ $(\mathrm{mod}\,2^4)$

分别加上模 $+10000$ $+10000$

结果仍互为补数 $+0101 \equiv +\boxed{1}0101$ 丢掉

$\therefore +0101 \equiv +0101 \quad (\mathrm{mod}\,2^4)$

可见 $+0101 \rightarrow +0101$

? $\rightarrow -1011$

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

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

$2^{4+1}-1011=100000 \quad (\mathrm{mod}\,2^{4+1})$

$$\begin{array}{r} 100000 \\ -1011 \\ \hline 1,0101 \end{array}$$

用 逗号 将符号位和数值位隔开

(3) 补码定义 6.1

整数

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

x 为真值　　　　n 为整数的位数

如　　$x = +1010$　　　　$x = -1011000$

$[x]_{补} = 0,1010$

$$[x]_{补} = 2^{7+1} + (-1011000) = 100000000 - 1011000 = 1,0101000$$

用 **逗号** 将符号位和数值位隔开

小数

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

x 为真值

如 $x = +0.1110$ $\qquad$ $x = -0.1100000$

$[x]_{补} = 0.1110$ $\qquad$ $[x]_{补} = 2 + (-0.1100000)$

$$\begin{array}{r} = 10.0000000 \\ -\ 0.1100000 \\ \hline 1.0100000 \end{array}$$

用 小数点 将符号位和数值位隔开

补码说明

- 补码最高一位是符号位，符号 0 正 1 负
- 补码表示为：2×符号位 + 数的真值
- 零的补码只有一个，故补码能表示 -1
- 补码能很好地用于加减运算，运算时，符号位与数值位一样参加运算。

求特殊数的补码

假定机器数有n位

① $[-2^{n-1}]_{补} = 2^n - 2^{n-1} = 10\ldots0$（n-1个0）　（mod 2^n）

② $[-1]_{补} = 2^n - 0\ldots01 = 11\ldots1$（n个1）　（mod 2^n）　整数补码

③ $[-1.0]_{补} = 2 - 1.0 = 1.00\ldots0$（n-1个0）　（mod 2）　小数补码

④ $[+0]_{补} = [-0]_{补} = 00\ldots0$（n个0）

特殊数的补码（续）

- 当补码的位数为n位时，其模为2^n，所以

$[-2^{n-1}]_{补} = 2^n - 2^{n-1} = 10\ldots0$（n-1个0）（mod 2^n）

- 当补码的位数为n+1位时，其模为2^{n+1}，所以

$[-2^{n-1}]_{补} = 2^{n+1} - 2^{n-1} = 2^n + 2^{n-1} = 1\ 10\ldots0$（n-1个0）(mod 2^{n+1})

这说明同一个真值在不同位数的补码表示中，对应的机器数不同，因此给定编码表示时，一定要明确编码的位数，在机器内部编码的位数就是机器中运算部件的位数。

(4) 求补码的快捷方式 6.1

设 $x = -1010$ 时

则 $[x]_{补} = 2^{4+1} - 1010 = 11111 + 1 - 1010$

$$
\begin{array}{r}
= 100000 \\
-\ 1010 \\
\hline
= 1,0110
\end{array}
\qquad
\begin{array}{r}
= 11111 + 1 \\
-\ 1010 \\
\hline
\boxed{10101} + 1 \\
= 1,0110
\end{array}
$$

又 $[x]_{原} = \boxed{1,1010}$

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

例 6.5　已知 $[x]_{补} = 0.0001$

求 x

解：由定义得　$x = +0.0001$

例 6.6　已知 $[x]_{补} = 1.0001$

求 x

解：由定义得

$x = [x]_{补} - 2$

$= 1.0001 - 10.0000$

$= -0.1111$

$[x]_{补} \xrightarrow{?} [x]_{原}$

$[x]_{原} = 1.1111$

$\therefore x = -0.1111$

例 6.7　已知 $[x]_{补} = 1{,}1110$　6.1

求 x

解：由定义得　　$[x]_{补} \xrightarrow{?} [x]_{原}$

$x = [x]_{补} - 2^{4+1}$　　$[x]_{原} = 1{,}0010$

$= 1{,}1110 - 100000$　　$\therefore x = -0010$

$= -0010$

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

真值	$[x]_{补}$	$[x]_{原}$
$x = +70 = 1000110$	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
$x = \boxed{0.0000}$ $[+0]_{补} = [-0]_{补}$	$\boxed{0.0000}$	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
$$[x]_{补} = \begin{cases} x & 1 > x \geq 0 \\ 2 + x & 0 > x \geq -1 \pmod 2 \end{cases}$$

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

4. 反码表示法 6.1

(1) 定义

整数

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

x 为真值　　　　n 为整数的位数

如　　$x = +1101$　　　　$x = -1101$

$[x]_{反} = 0,1101$　　　　$[x]_{反} = (2^{4+1} - 1) - 1101$

$= 11111 - 1101$

$= 1,0010$

用 逗号 将符号位和数值位隔开

小数 6.1

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

x 为真值

如

$x = +0.1101$ $\qquad$ $x = -0.1010$

$[x]_{反} = 0.1101$ $\qquad$ $[x]_{反} = (2-2^{-4}) - 0.1010$

$= 1.1111 - 0.1010$

$= 1.0101$

用 小数点 将符号位

和数值位隔开

(2) 举例

例6.8　已知 $[x]_{反} = 0,1110$　求 x

解：　由定义得　$x = +1110$

例6.9　已知 $[x]_{反} = 1,1110$　求 x

解：　由定义得　$x = [x]_{反} - (2^{4+1} - 1)$

$= 1,1110 - 11111$

$= -0001$

例 6.10　求 0 的反码

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

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

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

$\therefore [+0]_{反} \neq [-0]_{反}$

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（其中一位为符号位）

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 $[y]_{补}$　　求 $[-y]_{补}$　　6.1

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

<I>　　$\boxed{[y]_{补} = 0.\, y_1 y_2 \cdots y_n}$

$[y]_{补}$连同符号位在内，　每位取反，　末位加 1

即得$[-y]_{补}$

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

<II>　　$\boxed{[y]_{补} = 1.\, y_1 y_2 \cdots y_n}$

$[y]_{补}$连同符号位在内，　每位取反，　末位加 1

即得$[-y]_{补}$

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

5. 移码表示法

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
$x = -21$	-10101	$1,01011$	大
$x = +31$	$+11111$	$0,11111$	错
$x = -31$	-11111	$1,00001$	大

$x + 2^5$

$+10101 + 100000 = 110101$ 大

$-10101 + 100000 = 001011$ 正确

$+11111 + 100000 = 111111$ 大

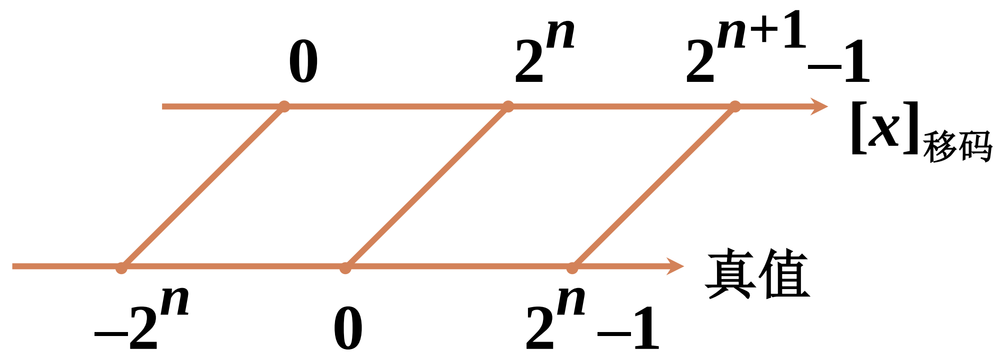
$-11111 + 100000 = 000001$ 正确

(1) 移码定义

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

x 为真值，n 为 整数的位数

移码在数轴上的表示



如 $x = 10100$

$[x]_{移} = 2^5 + 10100 = 1,10100$

$x = -10100$

$[x]_{移} = 2^5 - 10100 = 0,01100$

用 逗号 将符号位和数值位隔开

(2) 移码和补码的比较

设 $x = +1100100$

$[x]_{移} = 2^7 + 1100100 = 1,1100100$

$[x]_{补} = 0,1100100$

设 $x = -1100100$

$[x]_{移} = 2^7 - 1100100 = 0,0011100$

$[x]_{补} = 1,0011100$

补码与移码只差一个符号位

(3) 真值、补码和移码的对照表

真值 x (n=5)	$[x]_{补}$	$[x]_{移}$	$[x]_{移}$对应的十进制整数
- 1 0 0 0 0 0	1 0 0 0 0 0	0 0 0 0 0 0	0
- 1 1 1 1 1	1 0 0 0 0 1	0 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

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

  $[-0]_{移} = 2^5 - 0 = 1,00000$

  $\therefore [+0]_{移} = [-0]_{移}$

- 当 $n = 5$ 时 最小的真值为 $-2^5 = -100000$

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

  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

例：无符号数在C语言中对应**unsigned short**、**unsigned int(unsigned)**、**unsigned long**，带符号整数表示为**short**、**int**、**long**类型，求以下程序段在一个**32**位机器上运行时输出的结果，并说明为什么。

```
1 int x=-1;
2 unsigned u=2 147 483 648;
3 printf ("x=%u=%d\n", x , x);
4 printf ("u=%u=%d\n", u , u);
```

说明：

•其中**printf**为输出函数，指示符**%u**、**%d**分别表示以无符号整数和 有符号整数形式输出十进制数的值，

• 2 147 483 648=2^{31}

解：因为现代计算机中带符号整数都是用补码表示的， **-1**的补码整数表示为“**11…1**”（共**32**位），当作**32**位无符号数时，因此，十进制表示为 **2^{32}-1=4 294 967 296-1= 4 294 967 295**

2^{31}的无符号整数在计算机中表示为“**10…0**”，当作为有符号整数输出时，其值为最小负数**-2^{32-1}=-** 2 147 483 648

输出结果：

x= 4 294 967 295=-1

u=2 147 483 648=- 2 147 483 648

例：以下为C语言程序，用来计算一个数组a中每个元素的和，当参数len为0时，返回值应该为0，却发生了存储器访问异常。请问这是什么原因造成的？说明如何修改。

```
float  sum_elements  ( float a[ ], unsigned len)
  {
        int     i;
        float  result=0;
         for ( i=0; i<=len-1; i++ )
              result+=a[ i ] ;
         return result;
  }
```

解： 存储器访问异常是由于对数组a访问时产生了越界错误造成的。循环变量i是int型，而len是unsigned型，当len为0时，执行len-1的结果为32个1，是最大可表示的32位无符号数，任何无符号数都比它小，使循环体不断被执行，导致数组访问越界，因而发生存储器访问异常，应当将len声明为int型。

6.2 数的定点表示和浮点表示

小数点按约定方式标出

一、定点表示

S_f	$S_1S_2\cdots S_n$
数符	数值部分

小数点位置

或

S_f	$S_1S_2\cdots S_n$
数符	数值部分

小数点位置

定点机	小数定点机	整数定点机
原码	$-(1-2^{-n}) \sim +(1-2^{-n})$	$-(2^n-1) \sim +(2^n-1)$
补码	$-1 \sim +(1-2^{-n})$	$-2^n \sim +(2^n-1)$
反码	$-(1-2^{-n}) \sim +(1-2^{-n})$	$-(2^n-1) \sim +(2^n-1)$

二、浮点表示

$N = S \times r^j$　　浮点数的一般形式

S 尾数　j 阶码　r 基数（基值）

计算机中 r 取 2、4、8、16 等

当 $r = 2$　　N = 11.0101　　二进制表示

$= 0.110101 \times 2^{\boxed{10}}$　规格化数

$= 1.10101 \times 2^{1}$

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

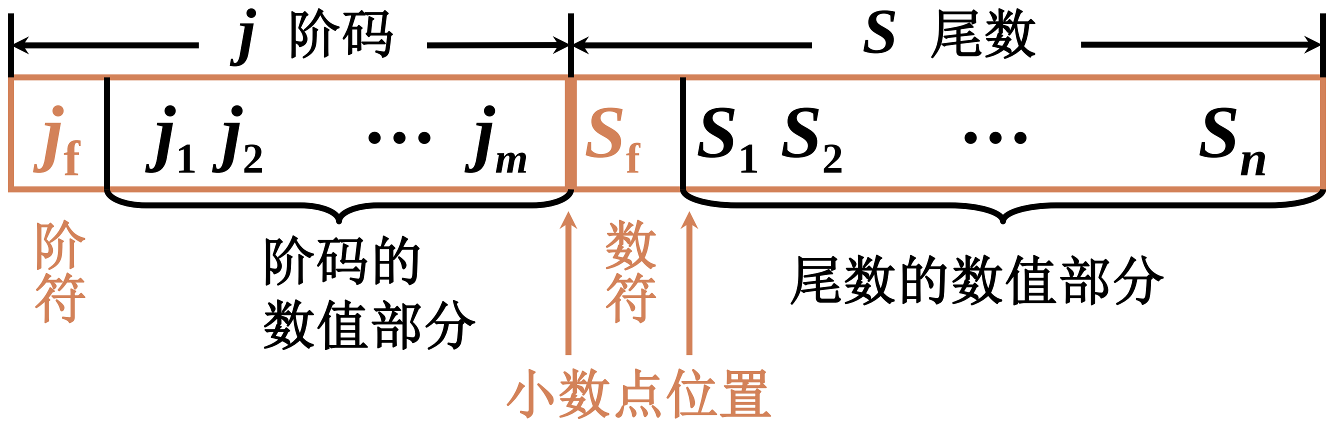
$= 0.00110101 \times 2^{100}$

尾数为纯小数

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

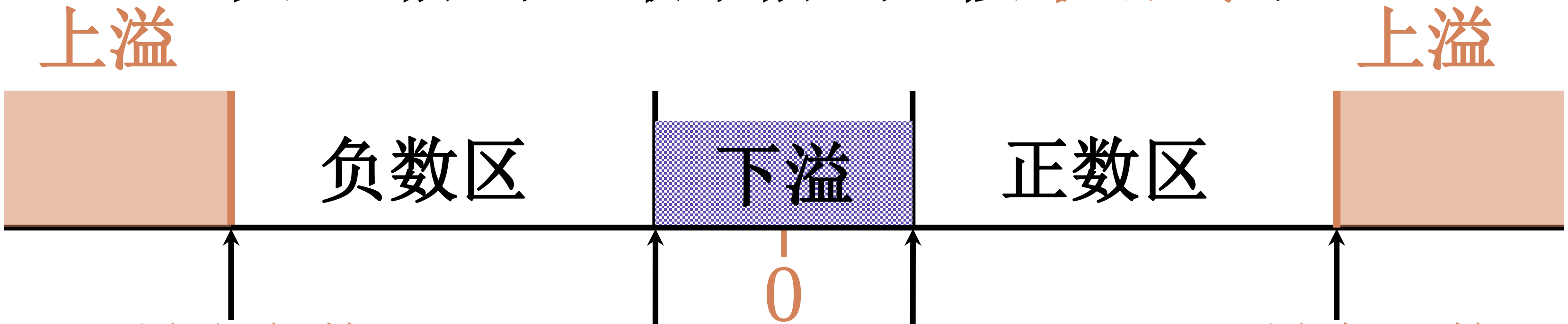
m 其位数反映浮点数的表示范围

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 > 最大阶玛

下溢　阶码 < 最小阶码　按 机器零 处理



最小负数

$$-2^{(2^m-1)}\times(1-2^{-n})$$

$$-2^{15}\times(1-2^{-10})$$

最大负数

$$-2^{-(2^m-1)}\times 2^{-n}$$

$$-2^{-15}\times 2^{-10}$$

最小正数

$$2^{-(2^m-1)}\times 2^{-n}$$

$$2^{-15}\times 2^{-10}$$

最大正数

$$2^{(2^m-1)}\times(1-2^{-n})$$

$$2^{15}\times(1-2^{-10})$$

设 $m = 4$

$n = 10$

练习 6.2

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

$\therefore$ 15 位二进制数可反映 ±3 万之间的十进制数

$$2^{\boxed{15}} \times 0.\underbrace{\times\times\times\ \dots\dots\ \times\times}_{18\text{位}}$$

$2^4 = 16$ $m = 4、5、6 \cdots$

满足 最大精度 可取 $m = 4$，$n = 18$

3. 浮点数的规格化形式 6.2

$|s| \geqslant 1/r$

$r = 2$	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

$$0\ .\ \times\ \times\ \times\ \times\ \times\ \times\ \times\ \times$$

$$2^{-1}\,2^{-2}\,2^{-3}\,2^{-4}\ldots$$

$r = 4$

$r = 8$

√ 0. 0101110 √

√ √ 0. 1000101 √

0. 0010101 √

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 $m = 4$，$n = 10$

尾数规格化后的浮点数表示范围

最大正数 $2^{+1111} \times 0.1111111111 = 2^{15} \times (1 - 2^{-10})$

$2^{-15} \times 2^{-10} = 2^{-25}$

最小正数 $2^{-1111} \times 0.1\underbrace{000000000} = 2^{-15} \times 2^{-1} = 2^{-16}$

$-2^{-15} \times 2^{-10} = -2^{-25}$

最大负数 $2^{-1111} \times (-0.1\underbrace{000000000}_{9\text{ 个 }0}) = -2^{-15} \times 2^{-1} = -2^{-16}$

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

三、举例

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）。

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　$[x]_{原} = [x]_{补} = [x]_{反} = 0.0010011000$

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

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

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

例 6.14　将 –58 表示成二进制定点数和浮点数，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　　$x = -0000\,111010$

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

定点机中

$[x]_{原} = 1, 0000111010$

$[x]_{补} = 1, 1111000110$

$[x]_{反} = 1, 1111000101$

浮点机中

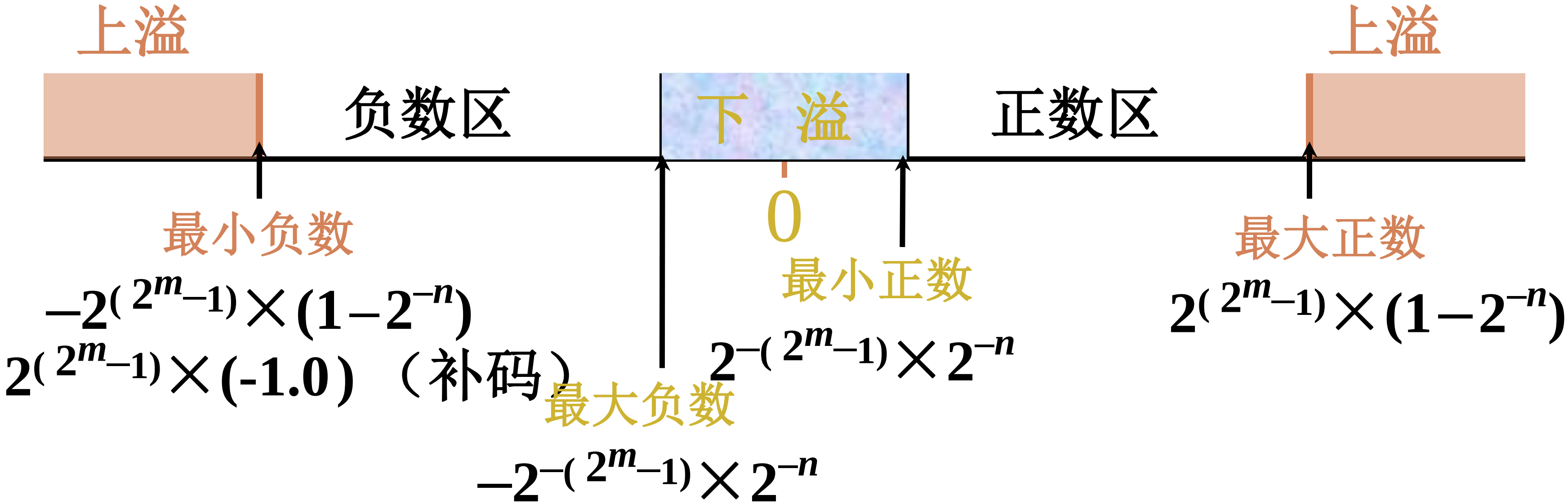
$[x]_{原} = 0, 0110; 1.\ 1110100000$

$[x]_{补} = 0, 0110; 1.\ 0001100000$

$[x]_{反} = 0, 0110; 1.\ 0001011111$

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

例6.15　写出对应下图所示的浮点数的补码形式。设 $n = 10$，$m = 4$，阶符、数符各取 1位。



解:

	真值	补码
最大正数	$2^{15} \times (1-2^{-10})$	0,1111; 0.1111111111
最小正数	$2^{-15} \times 2^{-10}$	1,0001; 0.0000000001
最大负数	$-2^{-15} \times 2^{-10}$	1,0001; 1.1111111111
最小负数	$-2^{15} \times (1-2^{-10})$	0,1111; 1.0000000001
负数边界	$2^{15} \times (-1.0)$	0,1111; 1.0000000000

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 $m = 4 \qquad n = 10$

当阶码和尾数都用补码表示时，机器零为

×, × × × ×；　0. 0 0 …… 0

（阶码 = −16）　1, 0 0 0 0；　×.× × …… ×

当阶码用移码，尾数用补码表示时，机器零为

0, 0 0 0 0；　0. 0 0 …… 0

有利于机器中“ 判 0 ” 电路的实现

“Father” of the IEEE 754 standard

直到80年代初，各个机器内部的浮点数表示格式还没有统一
因而相互不兼容，机器之间传送数据时，带来麻烦

1970年代后期，IEEE成立委员会着手制定浮点数标准

1985年完成浮点数标准IEEE754的制定
现在所有计算机都采用IEEE754来表示浮点数

This standard was primarily the work of one person, UC Berkeley math professor William Kahan.

www.cs.berkeley.edu/~wkahan/ieee754status/754story.html

Prof. William Kahan

IEEE 754 标准

现代计算机中，浮点数一般采用IEEE制定的国际标准IEEE754标准

符号	阶码（含阶符）	尾数

符号位： 0--正数，1--负数。

尾数: 1. xxx……x

在实际表示中整数位的1省略，称隐藏位（临时浮点数不采用隐藏位方案）

阶码: 移码

由于尾数形式的变化，阶码部分也与一般移码不同, 对短实数而言， $[X]_{移}=2^7-1+X = 127+X$

IEEE 754 Floating Point Standard

Single Precision：（Double Precision is similar）

1 bit S	8 bits Exponent	23 bits Significand

- Sign bit: 1 表示negative ; 0表示 positive
- Exponent（阶码 / 指数）：

全0和全1用来表示特殊值！

  - SP规格化数阶码范围为0000 0001 (真值：-126) ~ 1111 1110 (真值：127)
  - bias为127 (single), 1023 (double)

为什么用127？若用128, 则阶码范围为多少？

- Significand（尾数）：
  - 规格化尾数最高位总是1，所以隐含表示，省1位
  - 1 + 23 bits （ single），1 + 52 bits （double）

SP: $(-1)^S \times (1 + \text{Significand}) \times 2^{(\text{Exponent}-127)}$

DP: $(-1)^S \times (1 + \text{Significand}) \times 2^{(\text{Exponent}-1023)}$

0000 0001 (-127) ～ 1111 1110 (126)

IEEE制

主要用在浮点运算单元内部，用于减少舍入误差

	单精度数	双精度数	临时实数
字长	32	64	80 128
符号	1	1	1
阶码	8	11	15
偏移量	127	1023	16383
尾数	23+(1)	52+(1)	64 112
最大正数	$2^{127}(2-2^{-23})=2^{128}(1-2^{-24})\approx 1.7*10^{38}$	$2^{1023}(2-2^{-52})=2^{1024}(1-2^{-53})\approx 9*10^{307}$	
最小规格化正数	$2^{-126}\approx 1.2*10^{-38}$	$2^{-1022}\approx 2.2*10^{-308}$	
精度	$2^{-23}\approx 10^{-7}$		

Ex: Converting Binary FP to Decimal

BEE00000H is the hex. Rep. Of an IEEE 754 SP FP number

1	0111 1101	110 0000 0000 0000 0000 0000

$(-1)^S \times (1 + \text{Significand}) \times 2^{(\text{Exponent}-127)}$

- **Sign**: 1 => negative
- **Exponent**:
  - $0111\ 1101_{two} = 125_{ten}$
  - Bias adjustment: 125 - 127 = -2
- **Significand**:

  $1 + 1\times2^{-1} + 1\times2^{-2} + 0\times2^{-3} + 0\times2^{-4} + 0\times2^{-5} + \ldots$

  $=1+2^{-1} +2^{-2} = 1+0.5 +0.25 = 1.75$
- **Represents**: $-1.75_{ten}\times2^{-2} = -0.4375 \quad (= -4.375\times10^{-1})$

Ex: Converting Decimal to FP

$$-1.275 \times 10^1$$

1. Denormalize: -12. 75
2. Convert integer part:

   $12 = 8 + 4 = 1100_2$

3. Convert fractional part:

   $.75 = .5 + .25 = .11_2$

4. Put parts together and normalize:

   $1100.11 = 1.10011 \times 2^3$

5. Convert exponent: $127 + 3 = 128 + 2 = 1000\ 0010_2$

1	1000 0010	100 1100 0000 0000 0000 0000

The Hex rep. is C14C0000H

0的表示

How to represent 0?

exponent: all zeros

significand: all zeros

What about sign? Both cases valid.

+0: 0 00000000 00000000000000000000000

-0: 1 00000000 00000000000000000000000

如何表示+∞/-∞

In FP, 除数为0的结果是 +/- ∞, 不是溢出异常.　∞：无穷(infinity)

为什么要这样处理?

- 可以利用+∞/-∞作比较。 例如：X/0>Y可作为有效比较

How to represent +∞/-∞?

- Exponent : all ones (11111111B = 255)
- Significand: all zeros

+∞ : 0 11111111 00000000000000000000000

-∞ : 1 11111111 00000000000000000000000

Operations

5 / 0 = +∞,　　-5 / 0 = -∞

5+(+∞) = +∞,　　(+∞)+(+∞) = +∞

5 - (+∞) = -∞,　　(-∞) - (+∞) = -∞　　etc

用BCD码表示十进制数

- **BCD码:**用**4**位二进制代码的不同组合来表示一个十进制数码的编码方法。
- **编码思想： 每个十进数位至少有4位二进制表示。而4位二进制位可组合成16种状态，去掉10种状态后还有6种冗余状态。**

用BCD码表示十进制数

- 编码方案

1. 十进制有权码
   - 每个十进制数位的4个二进制位（称为基2码）都有一个确定的权。8421码是最常用的十进制有权码。也称自然BCD（NBCD）码。
2. 十进制无权码
   - 每个十进制数位的4个基2码没有确定的权。在无权码方案中，用的较多的是余3码和格雷码。
     - 余3码：**8421码+0011**
     - 格雷码：任何两个相邻代码只有一个二进制位的状态不同

十进制数	8421
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

用BCD码表示十进制数

- 编码方案

  1. 十进制有权码
  2. 十进制无权码
  3. 其他编码方案 （5中取2码、独热码等）

- 符号位的表示：
  - “+”：1100 ； “-”：1101
  - 例：+236=(1100 0010 0011 0110)$_{8421}$ (占2个字节)

    - 2369=(1101 0000 0010 0011 0110 1001)$_{8421}$

    (占3个字节)

补0以使数占满一个字节

BCD码

十进制数	8421	5421	2421	余3码	余3循环码
0	0000	0000	0000	0011	0010
1	0001	0001	0001	0100	0110
2	0010	0010	0010	0101	0111
3	0011	0011	0011	0110	0101
4	0100	0100	0100	0111	0100
5	0101	1000	1011	1000	1100
6	0110	1001	1100	1001	1101
7	0111	1010	1101	1010	1111
8	1000	1011	1110	1011	1110
9	1001	1100	1111	1100	1010

逻辑数据的编码表示

- 表示
  - 用一位表示 真：1 / 假：0
  - N位二进制数可表示N个逻辑数据，或一个位串
- 运算
  - 按位进行
  - 如:按位与 / 按位或 / 逻辑左移 / 逻辑右移 等
- 识别
  - 逻辑数据和数值数据在形式上并无差别，也是一串0/1序列，机器靠指令来识别。

西文字符的编码表示

- 特点
  - 是一种拼音文字，用有限几个字母可拼写出所有单词
  - 只对有限个字母和数学符号、标点符号等辅助字符编码
  - 所有字符总数不超过256个，使用7或8个二进位可表示
- 表示（常用编码为7位ASCII码）

  要求必须熟悉数字、字母和空格(SP)的表示
  - 十进制数字：0/1/2…/9
  - 英文字母：A/B/…/Z/a/b/…/z
  - 专用符号：+/-/%/*/&/……
  - 控制字符（不可打印或显示）
- 操作
  - 字符串操作，如:传送/比较　等

字符串的表示与存储

字符串是指连续的一串字符，它们占据主存中连续的多个字节，每个字节存放一个字符，对一个主存字的多个字节，有按从低位到高位字节次序存放的，也有按从高位到低位字节次序存放的。表示字符串数据要给出串存放的主存起始地址和串的长度。如：

IF A>B THEN READ(C) 就可以有如下不同存放方式：

I	F		A
>	B		T
H	E	N	
R	E	A	D
(	C	)	

A		F	I
T		B	>
	N	E	H
D	A	E	R
	)	C	(

假定每个字

由 4 个字节组成

字符串的表示与存储

按从低位到高位字节次序存放

I	F		A
>	B		T
H	E	N	
R	E	A	D
(	C	)	

31 24	23 16	15 8	7 0
49	**46**	**20**	**41**
3e	**42**	**20**	**54**
48	**45**	**4e**	**20**
52	**45**	**41**	**44**
28	**43**	**29**	**20**

16进制数据

汉字及国际字符的编码表示

- 特点
  - 汉字是表意文字，一个字就是一个方块图形。
  - 汉字数量巨大，总数超过6万字，给汉字在计算机内部的表示、汉字的传输与交换、汉字的输入和输出等带来了一系列问题。
- 编码形式
  - 有以下几种汉字代码：
    - 输入码：对汉字用相应按键进行编码表示，用于输入
    - 内码：用于在系统中进行存储、查找、传送等处理
    - 字模点阵或轮廓描述：描述汉字字模点阵或轮廓，用于显示/打印

汉字的输入码

向计算机输入汉字的方式：

① 手写汉字联机识别输入，或者是印刷汉字扫描输入后自动识别，这两种方法现均已达到实用水平。

② 用语音输入汉字，虽然简单易操作，但离实用阶段还有距离。

③ 利用英文键盘输入汉字：每个汉字用一个或几个键表示，这种对每个汉字用相应按键进行的编码称为汉字“输入码”，又称外码。输入码的码元为按键。是最简便、最广泛的汉字输入方法。

常用的方法有：搜狗拼音、五笔字型、智能ABC、微软拼音等

使用汉字输入码的原因：

① 键盘面向西文设计，一个或两个西文字符对应一个按键，非常方便。

② 汉字是大字符集，专门的汉字输入键盘由于键多、查找不便、成本高等原因而几乎无法采用。

字符集与汉字的内码

问题：西文字符常用的内码是什么？其内码就是ASCII码。

对于汉字内码的选择，必须考虑以下几个因素：

① 不能有二义性，即不能和ASCII码有相同的编码。

② 尽量与汉字在字库中的位置有关，便于汉字查找和处理。

③ 编码应尽量短。

国标码（国标交换码）

1981年我国颁布了《信息交换用汉字编码字符集·基本集》(GB2312—80)。该标准选出6763个常用汉字，为每个汉字规定了标准代码，以供汉字信息在不同计算机系统间交换使用

可在汉字国标码的基础上产生汉字机内码

GB2312-80字符集

- 由三部分组成：
  ① 字母、数字和各种符号，包括英文、俄文、日文平假名与片假名、罗马字母、汉语拼音等共687个
  ② 一级常用汉字，共3755个，按汉语拼音排列
  ③ 二级常用汉字，共3008个，不太常用，按偏旁部首排列
- 汉字的区位码
  - 码表由94行、94列组成，行号为区号，列号为位号，各占7位
  - 指出汉字在码表中的位置，共14位，区号在左、位号在右
- 汉字的国标码
  - 每个汉字的区号和位号各自加上32（20H），得到其“国标码”
  - 国标码中区号和位号各占7位。在计算机内部，为方便处理与存储，前面添一个0，构成一个字节

汉字内码

- 至少需2个字节才能表示一个汉字内码。
  - 由汉字的总数决定！
- 可在GB2312国标码的基础上产生汉字内码
  - 为与ASCII码区别，将国标码的两个字节的第一位置“1”后得到一种汉字内码

例如，汉字“大”在码表中位于第20行、第83列。因此区位码为0010100 1010011，国标码为00110100 01110011，即3473H。前面的34H和字符“4”的ACSII码相同，后面的73H和字符“s”的ACSII码相同，将每个字节的最高位各设为“1”后，就得到其内码：B4F3H (1011 0100 1111 0011B)，因而不会和ASCII码混淆。

国际字符集

国际字符集的必要性

- 不同地区使用不同字符集内码，如中文GB2312 / Big5、日文Shift-JIS / EUC-JP等。在安装中文系统的计算机中打开日文文件，会出现乱码。
- 为使所有国际字符都能互换，必须创建一种涵盖全部字符的多字符集。

国际多字符集

- 通过对各种地区性字符集规定使用范围来唯一定义各字符的编码。
- 国际标准ISO/IEC 10646提出了一种包括全世界现代书面语言文字所使用的所有字符的标准编码，有4个字节编码(UCS-4)和2字节编码(UCS-2) 。
- 我国（包括香港、台湾地区）与日本、韩国联合制订了一个统一的汉字字符集（CJK编码），共收集了上述不同国家和地区共约2万多汉字及符号，采用2字节编码 (即：UCS-2)，已被批准为国家标准(GB13000)。
- Windows操作系统(中文版)已采用中西文统一编码，收集了中、日、韩三国常用的约2万汉字，称为“Unicode”，采用2字节编码，与UCS-2一致 。

UNICODE编码

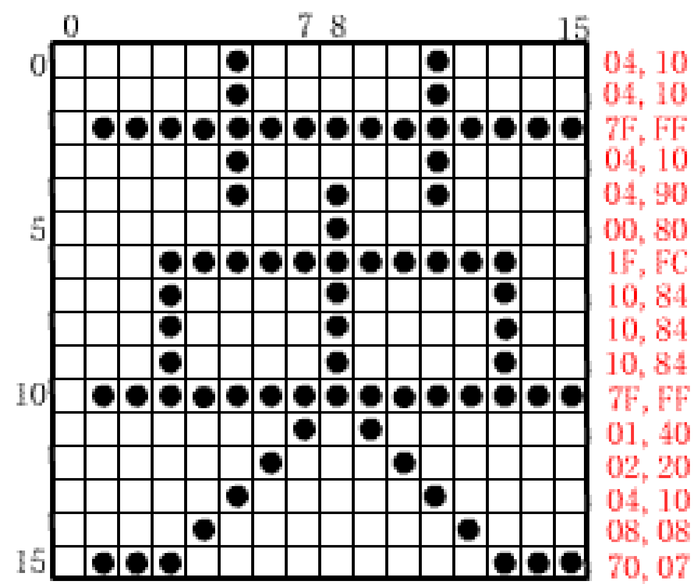
Unicode是完全双字节表示的多国文字编码体系，编码空间 0x0000-0xFFFF。可以表示65536 个字符

汉字的字模点阵码和轮廓描述

- 为便于打印、显示汉字，汉字字形必须预先存在机内
  - 字库 (font)：所有汉字形状的描述信息集合
  - 不同字体 (如宋体、仿宋、楷体、黑体等) 对应不同字库
  - 从字库中找到字形描述信息，然后送设备输出
- 字形主要有两种描述方法：
  - 字模点阵描述（图像方式）
  - 轮廓描述（图形方式）
    - 直线向量轮廓
    - 曲线轮廓（True Type字形）

字模点阵描述

- 字模编码
  - 字模编码是以点阵方式用来描述汉字字形的代码，它是汉字的输出形式。



数据的基本宽度

- 比特（bit）是计算机中处理、存储、传输信息的最小单位
- 二进制信息的计量单位是“字节”(Byte)，也称“位组”
  - 现代计算机中，存储器按字节编址
  - 字节是最小可寻址单位 *(addressable* unit *)*
- 除比特和字节外，还经常使用“字”(word)作为单位
- “字”和“字长”的概念不同

数据的基本宽度

- “字”和 “字长”的概念不同
  - “字长”指数据通路的宽度。

    (数据通路指CPU内部数据流经的路径以及路径上的部件，主要是CPU内部进行数据运算、存储和传送的部件，这些部件的宽度基本上要一致，才能相互匹配。因此， “字长”等于CPU内部总线的宽度、运算器的位数、通用寄存器的宽度等。)
  - “字”表示被处理信息的单位，用来度量数据类型的宽度。
  - 字和字长的宽度可以一样，也可不同。

    例如，x86体系结构定义“字”的宽度为16位，但从386开始字长就是32位了。

6.3 定点运算

一、移位运算

1. 移位的意义

15. 米 = 1500. 厘米

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

符号位不变

<table>
<tr><th></th><th>码　　制</th><th>添补代码</th></tr>
<tr><td>正数</td><td>原码、补码、反码</td><td>0</td></tr>
<tr><td rowspan="4">负数</td><td>原　　码</td><td>0</td></tr>
<tr><td rowspan="2">补　　码</td><td>左移 添 0</td></tr>
<tr><td>右移 添 1</td></tr>
<tr><td>反　　码</td><td>1</td></tr>
</table>

例6.16

设机器数字长为 8 位（含一位符号位），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 $[A]_{原}=[A]_{补}=[A]_{反}$	对应的真值
移位前	0,0011010	+26
←1	0,0110100	+52
←2	0,1101000	+104
→1	0,0001101	+13
→2	0,0000110	+6

例6.17

设机器数字长为 8 位（含一位符号位），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
⟵1	1,0110100	– 52
⟵2	1,1101000	– 104
⟶1	1,0001101	– 13
⟶2	1,0000110	– 6

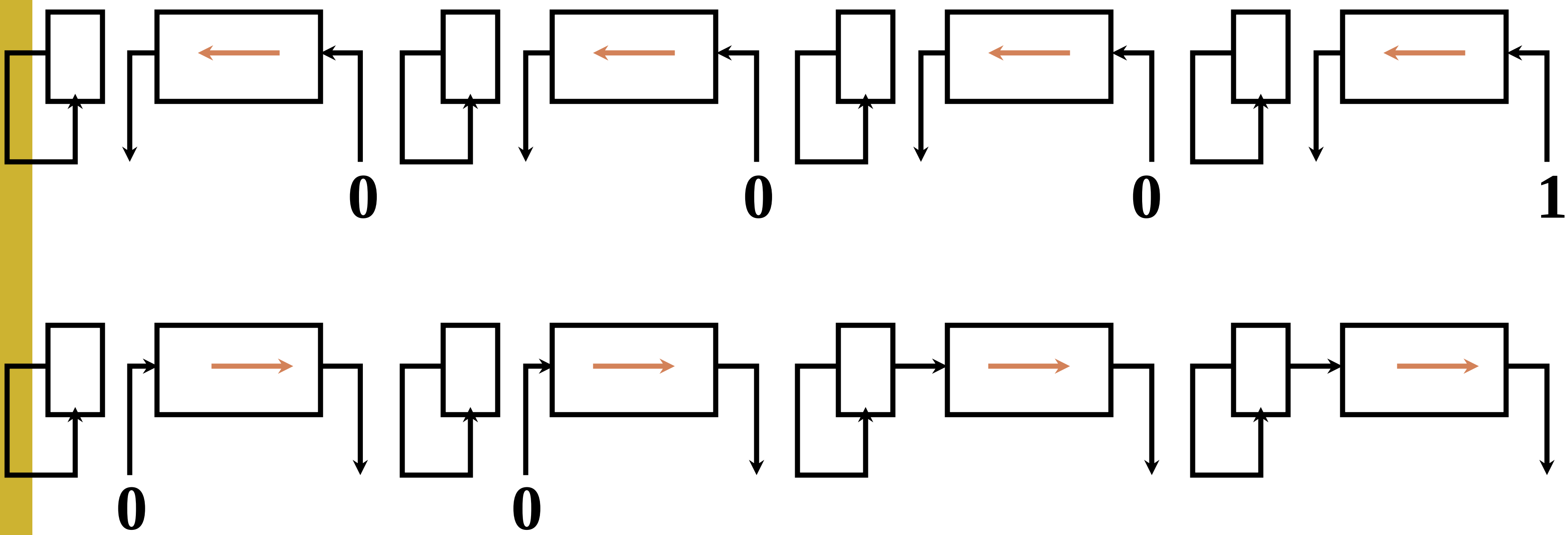
补码

移位操作	机　器　数	对应的真值
移位前	1,1100110	– 26
←1	1,1001100	– 52
←2	1,0011000	– 104
→1	1,1110011	– 13
→2	1,1111001	– 7

反码

移位操作	机　器　数	对应的真值
移位前	1,1100101	– 26
←1	1,1001011	– 52
←2	1,0010111	– 104
→1	1,1110010	– 13
→2	1,1111001	– 6

3. 算术移位的硬件实现



	（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

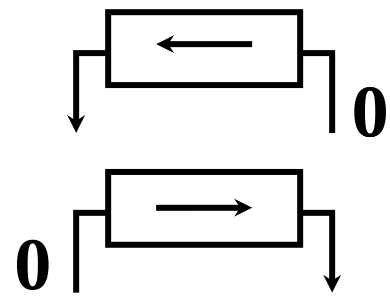
4. 算术移位和逻辑移位的区别

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　　11010011　　　　10110010

逻辑左移　　10100110　　逻辑右移　　01011001

算术左移　　10100110　　算术右移　　11011001（补码）

高位 1 移丢

C_y ← 1 1 0 1 0 0 1 1

1　　1 0 1 0 0 1 1 0

二、加减法运算 6.3

1. 补码加减运算公式

(1) 加法

整数 $[A]_{补} + [B]_{补} = [A+B]_{补}$（mod 2^{n+1}）

小数 $[A]_{补} + [B]_{补} = [A+B]_{补}$（mod 2）

(2) 减法

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

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

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

连同符号位一起相加，符号位产生的进位自然丢掉

证明

(1) 设A＞0, B＜0

① | B | ＞A (A+B ＜0)

$$[A+B]_{补}=2+A+B$$

$$=A+2+B$$

$$=[A]_{补}+[B]_{补}$$

② | B | ＜ A (A+B ＞ 0)

$$[A+B]_{补}=A+B$$

$$=A+2+B \pmod 2$$

$$=[A]_{补}+[B]_{补}$$

2. 举例

例 6.18　设 $A = 0.1011$，$B = -0.0101$

求 $[A + B]_{补}$　　验证

解：

$$
\begin{array}{rcl}
[A]_{补} & = & 0.1011 \\
+[B]_{补} & = & 1.1011 \\
\hline
[A]_{补} + [B]_{补} & = & 1\,0.0110 = [A + B]_{补}
\end{array}
$$

$\therefore A + B = 0.0110$

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

例 6.19　设 $A = -9$，$B = -5$

求 $[A+B]_{补}$　　验证

解：

$$
\begin{array}{rcl}
[A]_{补} & = & 1,0111 \\
+[B]_{补} & = & 1,1011 \\
\hline
[A]_{补} + [B]_{补} & = & 1\,1,0010 = [A + B]_{补}
\end{array}
$$

$\therefore A + B = -1110$

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

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

解：$A = 15 = 0001111$

$B = 24 = 0011000$

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

$$
\begin{array}{rl}
[A]_{补} = & 0,0001111 \\
+\ [-B]_{补} = & 1,1101000 \\
\hline
[A]_{补} + [-B]_{补} = & 1,1110111 = [A - B]_{补}
\end{array}
$$

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

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

$$\left.\begin{array}{l} 1 \oplus 0 = 1 \\ 0 \oplus 1 = 1 \end{array}\right\} \text{有 溢出}$$

$$\left.\begin{array}{l} 0 \oplus 0 = 0 \\ 1 \oplus 1 = 0 \end{array}\right\} \text{无 溢出}$$

(2) 两位符号位判溢出

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

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

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

结果的双符号位 相同　　未溢出　　00, × × × × ×

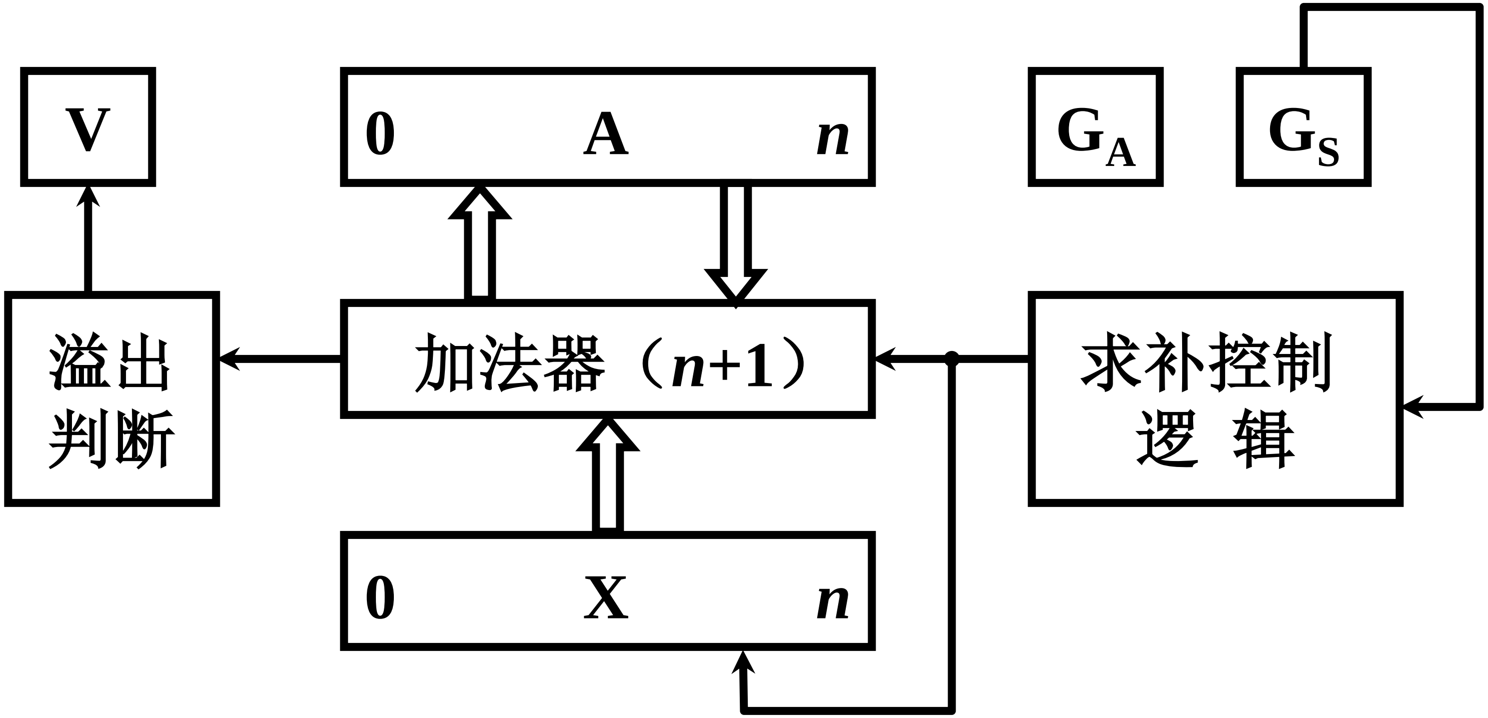
11, × × × × ×

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



A、X 均 $n+1$ 位

用减法标记 G_S 控制求补逻辑

三、乘法运算

1. 分析笔算乘法

$A = -0.1101 \qquad B = 0.1011$

$A \times B = -0.10001111$ 乘积的符号心算求得

$$
\begin{array}{r}
0.1101 \\
\times 0.1011 \\
\hline
1101 \\
1101 \\
0000 \\
1101 \\
\hline
0.10001111
\end{array}
$$

- ✓ 符号位单独处理
- ✓ 乘数的某一位决定是否加被乘数
- ? 4个位积一起相加
- ✓ 乘积的位数扩大一倍

2. 笔算乘法改进

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

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

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

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

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

右移一位

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

第一步 被乘数A + 0 ①

第二步 $\longrightarrow$ 1，得新的部分积 ②

第三步 部分积 + 被乘数 ③

⋮

第八步 $\longrightarrow$ 1，得结果 ⑧

3. 改进后的笔算乘法过程（竖式） 6.3

部分积	乘数	说明
0.0000	1011	初态，部分积 = 0
0.1101		乘数为 1，加被乘数
0.1101		
0.0110	1101	→1，形成新的部分积
0.1101		乘数为 1，加被乘数
1.0011	1	
0.1001	1110	→1，形成新的部分积
0.0000		乘数为 0，加 0
0.1001	11	
0.0100	1111	→1，形成新的部分积
0.1101		乘数为 1，加 被乘数
1.0001	111	
0.1000	1111	→1，得结果

小结 6.3

➢ 乘法 运算 ⟶ 加和移位。$n = 4$，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 ⟶ 1，形成新的部分积，同时乘数 ⟶ 1（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，具有移位功能

　　　　一个全加器

4. 原码乘法 6.3

(1) 原码一位乘运算规则

以小数为例

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

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

$$[x \cdot y]_{原} = (x_0 \oplus y_0).(0.x_1x_2 \cdots x_n)(0.y_1y_2 \cdots y_n)$$

$$= (x_0 \oplus y_0).\ x^*y^*$$

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

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

$$= x^*(y_1 2^{-1}+y_2 2^{-2}+ \ \ \cdots + y_n 2^{-n})$$

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

$$z_0 = 0$$
$$z_1 = 2^{-1}(y_nx^*+z_0)$$
$$z_2 = 2^{-1}(y_{n-1}x^*+z_1)$$
$$\vdots$$
$$z_n = 2^{-1}(y_1x^*+z_{n-1})$$

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

解： 数值部分的运算

	部分积	乘数	说明
	0 . 0 0 0 0	1 1 0 1	部分积 初态 $z_0 = 0$
	0 . 1 1 1 0		
	0 . 1 1 1 0		
	0 . 0 1 1 1	0 1 1 0	$\longrightarrow$ 1， 得 z_1
	0 . 0 0 0 0		
	0 . 0 1 1 1	0	
	0 . 0 0 1 1	1 0 1 1	$\longrightarrow$ 1， 得 z_2
	0 . 1 1 1 0		
	1 . 0 0 0 1	1 0	
逻辑右移	0 . 1 0 0 0	1 1 0 1	$\longrightarrow$ 1， 得 z_3
	0 . 1 1 1 0		
	1 . 0 1 1 0	1 1 0	
逻辑右移	0 . 1 0 1 1	0 1 1 0	$\longrightarrow$ 1， 得 z_4

例6.21 结果

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

$$x^* \cdot y^* = 0.\,1\,0\,1\,1\,0\,1\,1\,0$$

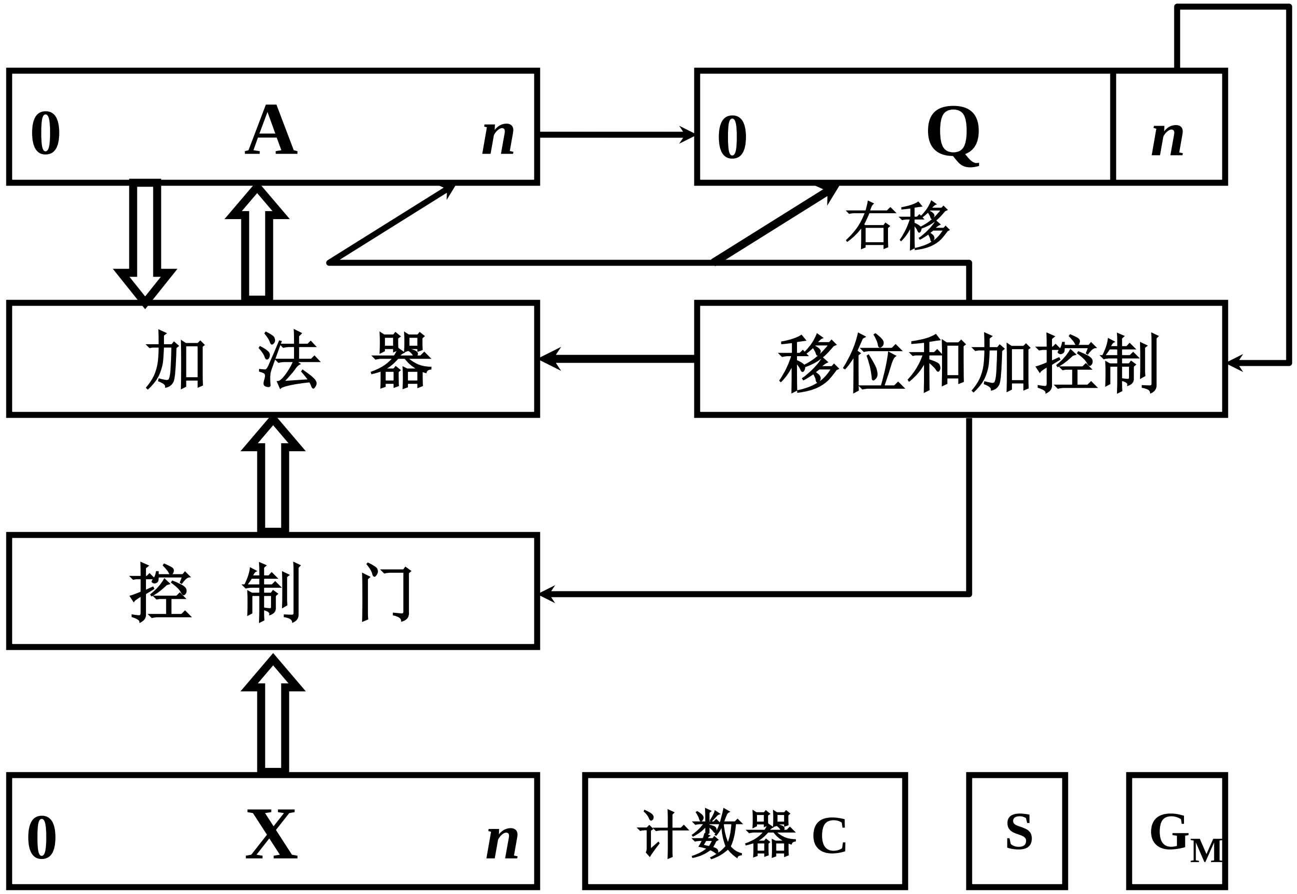
$$\text{则 } [x \cdot y]_{原} = 1.\,1\,0\,1\,1\,0\,1\,1\,0$$

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

(4) 原码两位乘

原码乘　　符号位 和 数值位 部分 分开运算

两位乘　　每次用 乘数的 2 位判断 原部分积是否加 和 如何加 被乘数

乘数 $y_{n-1}\ y_n$	新的部分积
0 0	加 “0” ⟶ 2
0 1	加 1 倍的被乘数 ⟶ 2
1 0	加 2 倍的被乘数 ⟶ 2
1 1	加 3 倍的被乘数 ⟶ 2

3 ?

$$\begin{array}{r} 4 \\ -\ 1 \\ \hline 3 \end{array} \qquad \begin{array}{r} 100 \\ -\ 01 \\ \hline 11 \end{array}$$

先 减 1 倍 的被乘数
再 加 4 倍 的被乘数

(5) 原码两位乘运算规则

乘数判断位$y_{n-1}y_n$	标志位C_j	操 作 内 容
0 0	0	$z \rightarrow 2, y^* \rightarrow 2, C_j$ 保持“0”
0 1	0	$z+x^* \rightarrow 2, y^* \rightarrow 2, C_j$ 保持“0”
1 0	0	$z+2x^* \rightarrow 2, y^* \rightarrow 2, C_j$ 保持“0”
1 1	0	$z-x^* \rightarrow 2, y^* \rightarrow 2$, 置“1” C_j
0 0	1	$z+x^* \rightarrow 2, y^* \rightarrow 2$, 置“0” C_j
0 1	1	$z+2x^* \rightarrow 2, y^* \rightarrow 2$, 置“0” C_j
1 0	1	$z-x^* \rightarrow 2, y^* \rightarrow 2, C_j$ 保持“1”
1 1	1	$z \rightarrow 2, y^* \rightarrow 2, C_j$ 保持“1”

共有操作 $+x^*$ $+2x^*$ $-x^*$ $\longrightarrow 2$

实际操作 $+[x^*]_{补}$ $+[2x^*]_{补}$ $+[-x^*]_{补}$ $\longrightarrow 2$ 补码移

例6.22 已知 $x = 0.111111$ $y = -0.111001$ 求 $[x \cdot y]_{原}$

解：数值部分的运算

部分积	乘数	C_j	说明
000.000000	00.111001	0	初态 $z_0 = 0$
000.111111			$+x^*$，$C_j = 0$
000.111111			
000.001111	11 001110	0	→2
001.111110			$+2x^*$，$C_j = 0$
010.001101	11		
000.100011	0111 0011	0	→2
111.000001			$-x^*$，$C_j = 1$
111.100100	0111		
111.111001	000111 00	1	→2
000.111111			$+x^*$，$C_j = 0$
000.111000	000111		

补码右移

补码右移

例6.22 结果

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

② 数值部分的运算

$$x^* \cdot y^* = 0.\,1\,1\,1\,0\,0\,0\,0\,0\,0\,1\,1\,1$$

则 $[x \cdot y]_{原} = 1.\,1\,1\,1\,0\,0\,0\,0\,0\,0\,1\,1\,1$

特点　绝对值的补码运算

用移位的次数判断乘法是否结束

算术移位

(6) 原码两位乘和原码一位乘比较　　6.3

	原码一位乘	原码两位乘
符号位	$x_0 \oplus y_0$	$x_0 \oplus y_0$
操作数	绝对值	绝对值的补码
移位	逻辑右移	算术右移
移位次数	n	$\frac{n}{2}$（n为偶数）
最多加法次数	n	$\frac{n}{2}+1$（n为偶数）

思考　n 为奇数时，原码两位乘 移 ? 次 最多加 ? 次

5. 补码乘法

（以小数为例）

设 被乘数 $[x]_{补} = x_0.x_1x_2 \ \dots \ x_n$

$[2^{-1}x]_{补} = x_0.x_0x_1x_2 \ \dots \ x_n = 2^{-1}[x]_{补}$

$[-0.110]_{补}=1.010$

$[2^{-1}(-0.110)]_{补}=[-0.0110]_{补}=1.1010$

乘数 $[y]_{补} = y_0.y_1y_2 \ \cdots \ y_n$

$y = [y]_{补} - 2y_0$

$= y_0.y_1y_2 \ \cdots \ y_n - 2y_0$

$= 0.y_1y_2 \ \cdots y_n - y_0$

y为正时$y_0=0$

y为负时$y_0=1$

(1) 补码一位乘运算规则

① 校正法

和原码一位乘法相同，部分积右移时采用算术移位

$[x \cdot y]_{补}$

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

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

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

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

用校正法求$[x \cdot y]_{补}$是用$[x]_{补}$乘以 $[y]_{补}$的数值位，最后再根据$[y]_{补}$的符号y_0做一次校正

$\begin{cases} y\text{为负，加}[-x]_{补} \\ y\text{为正，不加} \end{cases}$

③ Booth 算法（被乘数、乘数符号任意） 6.3

设$[x]_补 = x_0.x_1x_2 \cdots x_n$　　$[y]_补 = y_0.y_1y_2 \cdots y_n$

$$[x \cdot y]_补$$

$$-[x]_补 = +[-x]_补$$

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

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

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

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

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

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

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

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

附加位 y_{n+1}

④ Booth 算法递推公式

$$[z_0]_补 = 0$$

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

$$\vdots$$

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

$$[x \cdot y]_补 = [z_n]_补 + (y_1 - y_0)[x]_补$$

最后一步不移位

如何实现 $y_{i+1} - y_i$ ？

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

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

解：

部分积	乘数	y_{n+1}	说明
00.0000	1.0101	0	
11.1101			$+[-x]_{补}$
11.1101			
11.1110	1 1010	1	$\rightarrow 1$
00.0011			$+[x]_{补}$
00.0001	1		
00.0000	11 101	0	$\rightarrow 1$
11.1101			$+[-x]_{补}$
11.1101	11		
11.1110	111 10	1	$\rightarrow 1$
00.0011			$+[x]_{补}$
00.0001	111		
00.0000	1111 1	0	$\rightarrow 1$
11.1101			$+[-x]_{补}$
11.1101	1111		最后一步不移位

算术移位

$[x]_{补} = 0.0011$

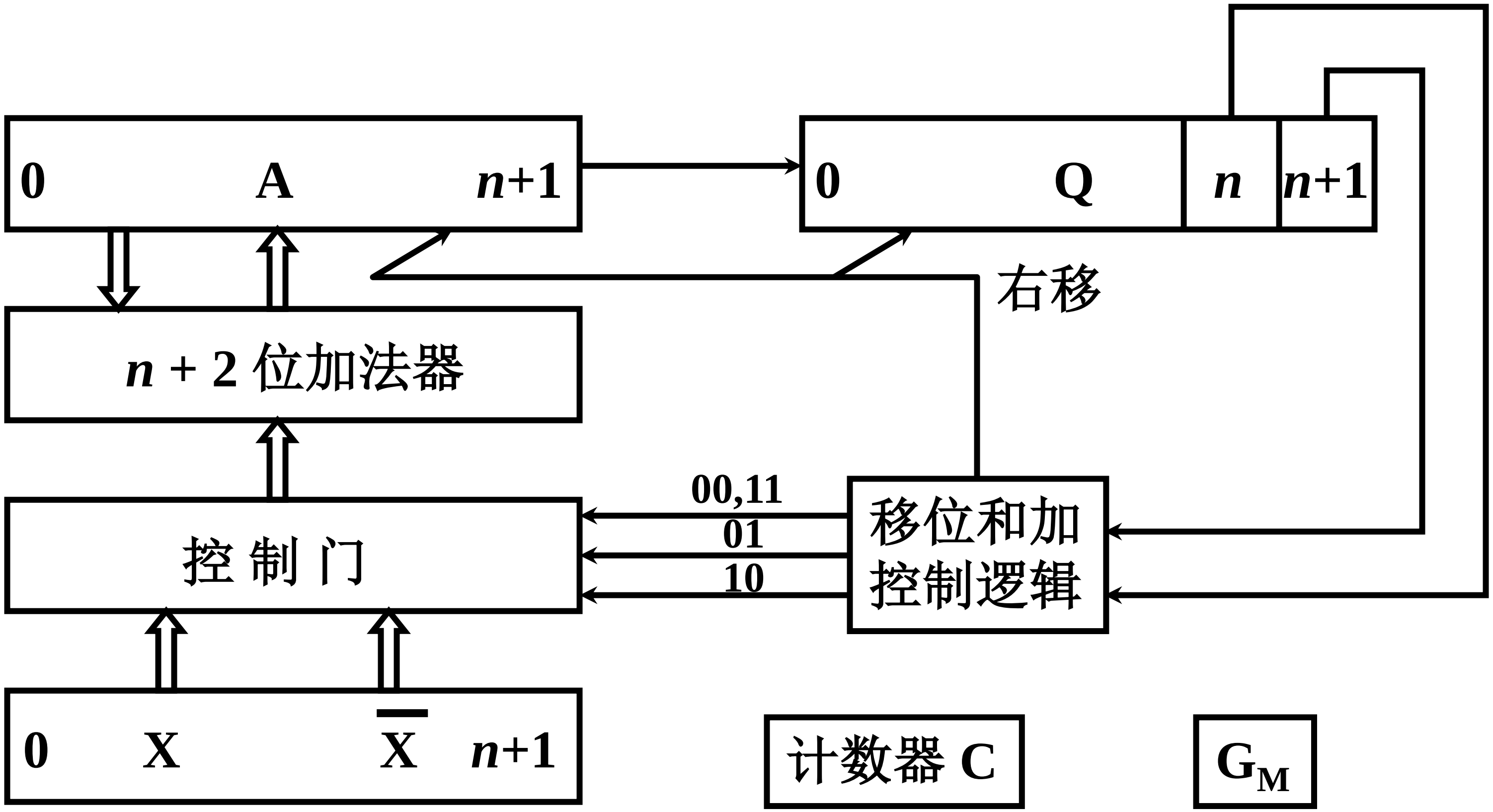
$[y]_{补} = 1.0101$

$[-x]_{补} = 1.1101$

$\therefore [x \cdot y]_{补}$

$= 1.11011111$

(2) Booth 算法的硬件配置 6.3



A、X、Q　均 $n+2$ 位

移位和加受末两位乘数控制

乘法小结

- 整数乘法与小数乘法完全相同
  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

四、除法运算

1. 分析笔算除法

$x = -0.1011 \quad y = 0.1101 \quad$ 求 $x \div y$

```
               0 . 1 1 0 1
          ________________
0 . 1 1 0 1 ) 0 . 1 0 1 1 0
              0 . 0 1 1 0 1
              ---------------
              0 . 0 1 0 0 1 0
              0 . 0 0 1 1 0 1
              ---------------
              0 . 0 0 0 1 0 1 0 0
              0 . 0 0 0 0 1 1 0 1
              ---------------
              0 . 0 0 0 0 0 1 1 1
```

✓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

$x \div y = -0.1101$ 商符心算求得

余数 -0.00000111

2. 笔算除法和机器除法的比较

笔算除法	机器除法
商符单独处理	符号位异或形成
心算上商	$\lvert x\rvert - \lvert y\rvert > 0$ 上商 1
	$\lvert x\rvert - \lvert y\rvert < 0$ 上商 0
余数 不动 低位补“0” 减右移一位 的除数	余数 左移一位 低位补“0” 减 除数
2 倍字长加法器	1 倍字长加法器
上商位置 不固定	在寄存器 最末位上商

3. 原码除法 6.3

以小数为例

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

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

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

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

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

- **除前预处理**

  ①若被除数=0且除数≠0，或定点整数除法|被除数|<|除数|，则商为0，不再继续

  ②若被除数≠0、除数=0，则发生“除数为0”异常

  ③若被除数和除数都为0，则有些机器产生一个不发信号的NaN，即“quiet NaN”

  当被除数和除数都≠ 0，且商≠ 0时，才进一步进行除法运算。

(1) 恢复余数法

6.3

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

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

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		$+[-y^*]_{补}$
1.1110	0	余数为负，上商 0
0.1101		恢复余数　$+[y^*]_{补}$
0.1011	0	恢复后的余数
逻辑左移　1.0110	0	←1
1.0011		$+[-y^*]_{补}$
0.1001	01	余数为正，上商 1
逻辑左移　1.0010	01	←1
1.0011		$+[-y^*]_{补}$

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

逻辑左移

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

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

上商 5 次

第一次上商判溢出

移 4 次

余数为正　上商 1

余数为负　上商 0，恢复余数

(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”， $R_{i+1} = 2R_i - y^*$

余数 $R_i<0$ 上商 “0”， $R_i + y^*$ 恢复余数

$$R_{i+1} = 2(R_i + y^*) - y^* = 2R_i + y^*$$

- 不恢复余数法运算规则

$R_i>0$ 上商 “1” $R_{i+1} = 2R_i - y^*$

$R_i<0$ 上商 “0” $R_{i+1} = 2R_i + y^*$

加减交替

例6.25 $x = -0.1011$ $y = -0.1101$ 求 $[\frac{x}{y}]_{原}$

解：

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

逻辑左移

$[x]_{原} = 1.1011$

$[y]_{原} = 1.1101$

$[y^*]_{补} = 0.1101$

$[-y^*]_{补} = 1.0011$

例6.25 结果

① $x_0 \oplus y_0 = 1 \oplus 1 = 0$

② $\dfrac{x^*}{y^*} = 0.1101$

$\therefore\ [\dfrac{x}{y}]_{原} = 0.1101$

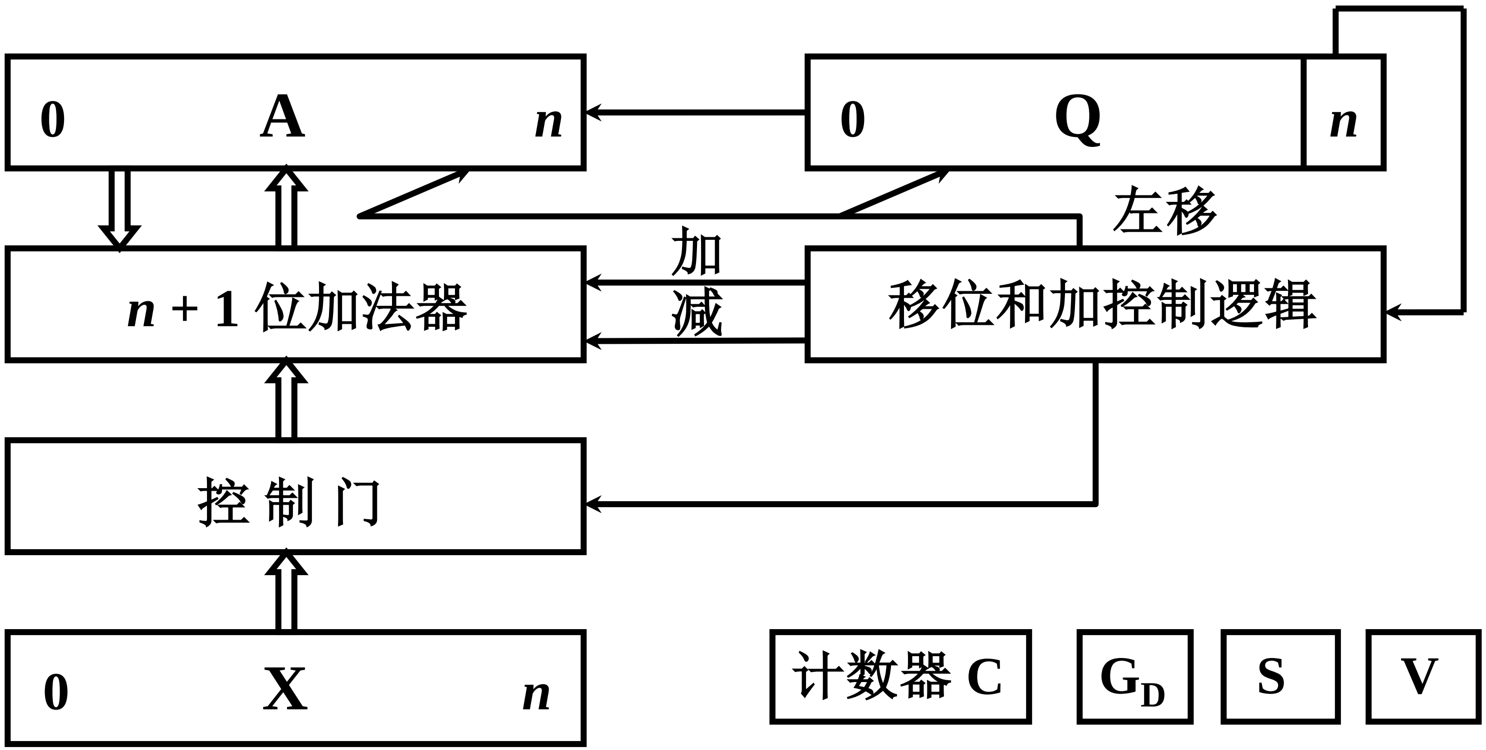
特点 上商 $n+1$ 次

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

4. 补码除法

(1) 商值的确定

① 比较被除数和除数绝对值的大小

➢ x 与 y 同号　用减法

$x = 0.1011$	$[x]_补 = 0.1011$	$[x]_补 = 0.1011$	$x^* > y^*$
$y = 0.0011$	$[y]_补 = \boxed{0.}0011$	$+[-y]_补 = 1.1101$	$[R_i]_补$与$[y]_补$同号
		$[R_i]_补 = \boxed{0.}1000$	“够减”

$x = -0.0011$	$[x]_补 = 1.1101$	$[x]_补 = 1.1101$	$x^* < y^*$
$y = -0.1011$	$[y]_补 = \boxed{1.}0101$	$+[-y]_补 = 0.1011$	$[R_i]_补$与$[y]_补$异号
		$[R_i]_补 = \boxed{0.}1000$	“不够减”

➢x 与 y 异号　用加法

$x = 0.1011$　　$[x]_{补} = 0.1011$

$y = -0.0011$　　$[y]_{补} = \boxed{1.}1101$

$$\begin{array}{rl} & [x]_{补} = 0.1011 \\ + & [y]_{补} = 1.1101 \\ \hline & [R_i]_{补} = \boxed{0.}1000 \end{array}$$

$x^* > y^*$

$[R_i]_{补}$与$[y]_{补}$异号

"够减"

$x = -0.0011$　　$[x]_{补} = 1.1101$

$y = 0.1011$　　$[y]_{补} = \boxed{0.}1011$

$$\begin{array}{rl} & [x]_{补} = 1.1101 \\ + & [y]_{补} = 0.1011 \\ \hline & [R_i]_{补} = \boxed{0.}1000 \end{array}$$

$x^* < y^*$

$[R_i]_{补}$与$[y]_{补}$同号

"不够减"

小结

$[x]_{补}$和$[y]_{补}$	求 $[R_i]_{补}$	$[R_i]_{补}$与 $[y]_{补}$
同号	$[x]_{补} - [y]_{补}$	同号，"够减"
异号	$[x]_{补} + [y]_{补}$	异号，"够减"

② 商值的确定　末位恒置“1”法

$$\times.\underbrace{\times\times\times\times}1$$

$[x]_{补}$与$[y]_{补}$同号　0.　原码　1　按原码上商　“够减”上“1”

正商　　“不够减”上“0”

$$\times.\underbrace{\times\times\times\times}1$$

$[x]_{补}$与$[y]_{补}$异号　1.　反码　1　按反码上商　“够减”上“0”

负商　　“不够减”上“1”

小 结

$[x]_{补}$与$[y]_{补}$	商	$[R_i]_{补}$与$[y]_{补}$	商 值	
同 号	正	够减　（同号）	1	原码上商
		不够减（异号）	0	
异 号	负	够减　（异号）	0	反码上商
		不够减（同号）	1	

简 化 为

$[R_i]_{补}$与$[y]_{补}$	商值
同 号	1
异 号	0

(2) 商符的形成 6.3

除法过程中自然形成

$[x]_补$和$[y]_补$同号 ⟶ $[x]_补-[y]_补$ ⟶ 比较$[R_i]_补$和$[y]_补$

同号(够) “1”

异号(不够) “0”

原码上商

小数除法 第一次“不够”上“0”

正商

$[x]_补$和$[y]_补$异号 ⟶ $[x]_补+[y]_补$ ⟶ 比较$[R_i]_补$和$[y]_补$

异号(够) “0”

同号(不够) “1”

反码上商

小数除法 第一次“不够”上“1”

负商

(3) 新余数的形成

加减交替

$[R_i]_补$和 $[y]_补$	商	新余数
同 号	1	$2[R_i]_补 + [-y]_补$
异 号	0	$2[R_i]_补 + [\ y\ ]_补$

补码加减交替法的算法规则

1. 符号位参加运算，除数与被除数均用双符号补码表示。
2. 被除数与除数同号，被除数减去除数。被除数与除数异号，被除数加上除数。商符号位取值见3。
3. 余数与除数同号，商上**1**，余数左移**1**位减去除数余数与除数异号，商上**0**，余数左移**1**位加上除数。（余数左移加上或减去除数就得到新余数）
4. 采用校正法包括符号位在内，应重复③**n+1**次.

商的校正原则

- 精度要求不高时，将商的最低位恒置1，最大误差2^{-n}

例6.26 设 $x = -0.1011$　$y = 0.1101$ 求$[\frac{x}{y}]_{补}$ 并还原成真值

解：　$[x]_{补} = 1.0101$　　$[y]_{补} = 0.1101$　　$[-y]_{补} = 1.0011$

1.0101	0.0000	
0.1101		异号做加法
0.0010	1	同号上“1”
0.0100	1	←1
1.0011		$+[-y]_{补}$
1.0111	10	异号上“0”
0.1110	10	←1
0.1101		$+[y]_{补}$
1.1011	100	异号上“0”
1.0110	100	←1
0.1101		$+[y]_{补}$
0.0011	1001	同号上“1”
0.0110	10011	←1 末位恒置“1”

逻辑左移

$\therefore [\frac{x}{y}]_{补} = 1.0011$

则 $\frac{x}{y} = -0.1101$

(4) 小结

- 补码除法共上商 $n+1$ 次（末位恒置 1）第一次为商符
- 加 n 次　移 n 次
- 第一次商可判溢出
- 精度误差最大为 2^{-n}

十进制数的加减运算

- **有的机器有十进制加减法指令，用于对BCD码进行加减运算。所以这些机器中必须要有相应的十进制加减运算逻辑。**
- **以NBCD码（8421码）为例，讨论十进制整数的加减运算。**
- **一般规定数符在最高位 1100：正，1101：负**

或 0：正， 1：负

例如：+2039 1100 0010 0000 0011 1001

或 0 0010 0000 0011 1001

-1265 1101 0001 0010 0110 0101

1 0001 0010 0110 0101

符号和数值部分分开处理！

十进制加法运算举例

例1 25+31=56

```
   0010 0101
 + 0011 0001
 -----------
   0101 0110
```

例2 25+39=64

```
   0010 0101
 + 0011 1001
 -----------
   0101 1110
        0110
 -----------
   0110 0100
```

$(1110)_2>9$ 需“+6”校正

例3 27+39=66

```
   0010 0111
 + 0011 1001
 -----------
   0101 0000
      1 0110
 -----------
   0110 0110
```

低位有进位，则进到高位，同时该低位“+6”校正

问题：本位和在什么范围内需“+6”校正？

- 结果<=9时，不需校正；大于9或有进位时，需“+6”校正
- 最高位有进位时，发生溢出

大于9：1010,1011,……,1111

有进位：10000,10001,10010和10011

最大为19: 2x9+1=19, 范围为10~19

6.4 浮点四则运算

一、浮点加减运算

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

1. 对阶

(1) 求阶差

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

(2) 对阶原则

小阶向大阶看齐　　为什么？

6.4 浮点四则运算

一、浮点加减运算

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

1. 对阶

2. 尾数相加

尾数相加与定点数加减法相同

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

求 $x+y$

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

1. 对阶

① 求阶差 $[\Delta j]_{补} = [j_x]_{补} - [j_y]_{补}$

$$
\begin{array}{rr}
 & =00,01 \\
+ & 11,01 \\
\hline
 & 11,10
\end{array}
$$

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

② 对阶 $[x]_{补'} = 00, 11; 00.0011$

2. 尾数求和

$$
\begin{array}{rlr}
 & [S_x]_{补'} & = 00.0011 \\
+ & [S_y]_{补} & = 11.0110 \\
\hline
 & & 11.1001
\end{array}
$$

对阶后的 $[S_x]_{补'}$

$\therefore [x+y]_{补} = 00, 11; 11.1001$

3. 规格化

(1) 规格化数的定义

$$r = 2 \qquad \frac{1}{2} \le |S| < 1$$

(2) 规格化数的判断

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

原码　不论正数、负数，第一数位为1

补码　符号位和第 1 数位不同

特例

$$S = -\frac{1}{2} = -0.100\cdots0$$

$$[S]_{原} = 1.100\cdots0$$

$$[S]_{补} = \boxed{1.1}00\cdots0$$

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

机器判别方便

$$S = -1$$

$$[S]_{补} = \boxed{1.0}00\cdots0$$

$\therefore$ $[-1]_{补}$ 是规格化的数

(3) 左规

尾数←1，阶码减 1，直到数符和第一数位不同为止

上例 $[x+y]_{补} = 00, 11; 11.1001$

左规后 $[x+y]_{补} = 00, 10; 11.0010$

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

(4) 右规

当 尾数溢出（ >1）时，需 右规

即尾数出现 01. ×× ⋯×或 10. ××⋯×时

尾数→1，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：　$[x]_{补} = 00, 010; 00.110100$

$[y]_{补} = 00, 001; 00.101100$

① 对阶

$$[\Delta j]_{补} = [j_x]_{补} - [j_y]_{补} = \begin{array}{r} 00,010 \\ +\ 11,111 \\ \hline \boxed{1}00,001 \end{array}$$

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

$\therefore [y]_{补'} = 00, 010; 00.010110$

② 尾数求和

$$\begin{array}{rl} [S_x]_{补} = & 00.110100 \\ +\ [S_y]_{补'} = & 00.010110 \\ \hline & 01.001010 \end{array}$$

对阶后的 $[S_y]_{补'}$

尾数溢出需右规

$[x+y]_{补} = 00, 010; 01. 001010$

右规后

$[x+y]_{补} = 00, 011; 00. 100101$

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

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： 移掉的最高位为1，尾数末尾加1；为0，舍掉。

(2) 恒置 “1” 法右移时，丢掉低位值，就将结果最低位置1。

例 6.28 $x = (-\frac{5}{8}) \times 2^{-5}$ $y = (\frac{7}{8}) \times 2^{-4}$ 6.4

求 $x-y$（除阶符、数符外，阶码取 3 位，尾数取 6 位）

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

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

① 对阶

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

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

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

② 尾数求和

$$
\begin{array}{r}
[S_x]_{补'} = 11.\ 101100 \\
+\ \ [-S_y]_{补} = 11.\ 001000 \\ \hline
\boxed{1}10.\ 110100
\end{array}
$$

③ 右规

$$[x+y]_{补} = 11,\ 100;\ 10.\ 110100$$

右规后

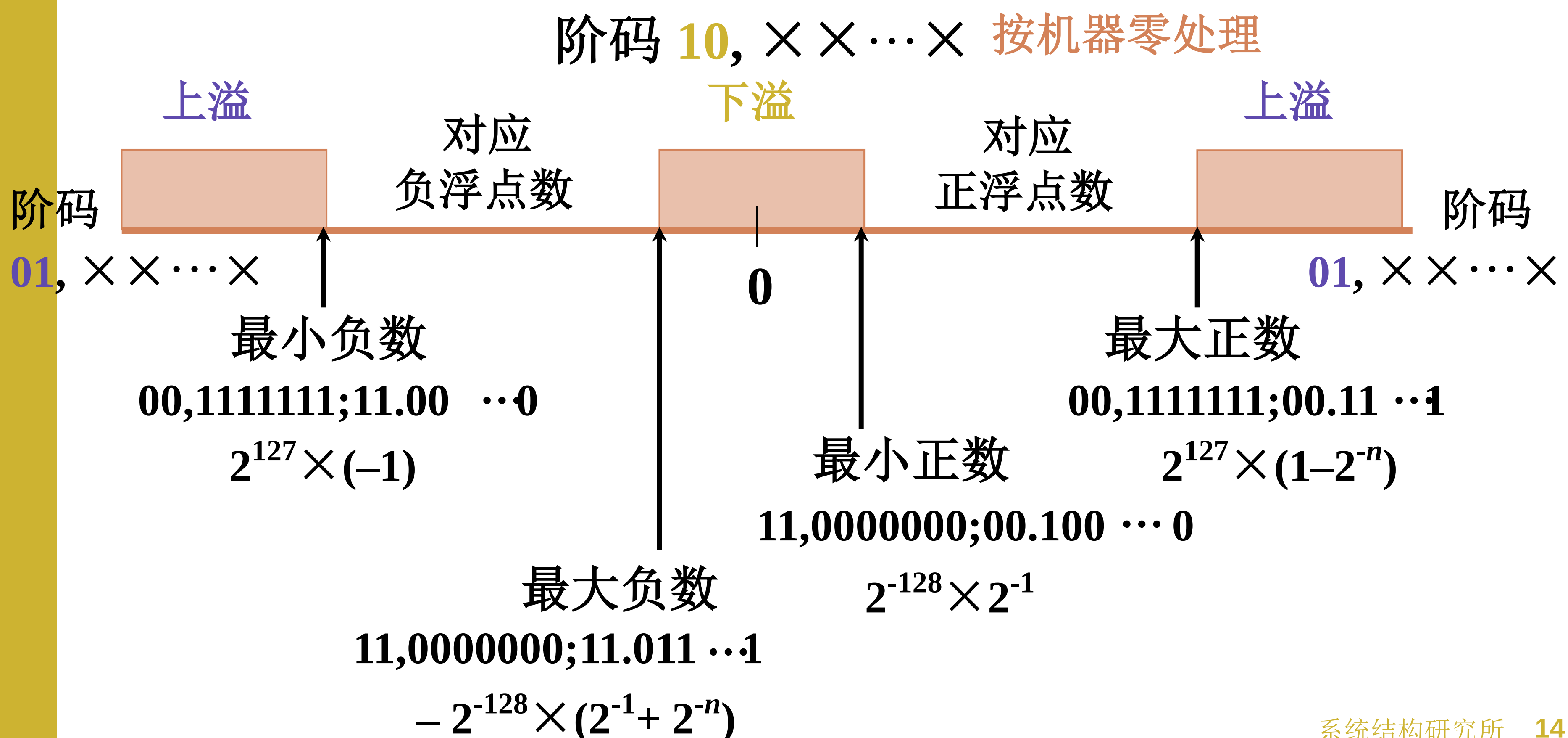
$$[x+y]_{补} = 11,\ 101;\ 11.\ 011010$$

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

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

5. 溢出判断

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



移码加/减运算

- 用于浮点数阶码运算
- 符号位和数值部分可以一起处理
- 运算公式（假定在一个n位ALU中进行加法运算）

$$[E1]_{移}+[E2]_{移}=2^{n-1}+E1+2^{n-1}+E2=2^{n}+E1+E2=[E1+E2]_{补}\ (\bmod\ 2^{n})$$

$$\begin{aligned}[E1]_{移}-[E2]_{移}&=[E1]_{移}+[-[E2]_{移}]_{补}=2^{n-1}+E1+2^{n}-[E2]_{移}\\&=2^{n-1}+E1+2^{n}-2^{n-1}-E2\\&=2^{n}+E1-E2=[E1-E2]_{补}\ (\bmod\ 2^{n})\end{aligned}$$

结论：移码的和、差等于和、差的补码！

- 运算规则　　补码和移码的关系：符号位相反、数值位相同！

① 加法：直接将$[E1]_{移}$和$[E2]_{移}$进行模2^n加，然后对结果的符号取反。

② 减法：先将减数$[E2]_{移}$求补（各位取反，末位加1），然后再与被减数 $[E1]_{移}$进行模2^n相加，最后对结果的符号取反。

③ 溢出判断：进行模2^n相加时，如果两个加数的符号相同，并且与和数的符号也相同，则发生溢出。

移码加/减运算

例1： 用四位移码计算“−7+（− 6）” 和 “−3 + 6”的值。

解：$[-7]_{移}$ = 0001　　$[-6]_{移}$ = 0010　　$[-3]_{移}$ = 0101　　$[6]_{移}$ = 1110

$[-7]_{移} + [-6]_{移}$ = 0001 + 0010 = 0011 （两个加数与结果符号都为0，溢出）

$[-3]_{移} + [6]_{移}$ = 0101 + 1110 = 0011， 符号取反后为 1011，其真值为+3

问题：$[-7+(-6)]_{移}$=?　　　　$[-3+(6)]_{移}$ =？

例2： 用四位移码计算“−7 −（− 6）” 和 “−3 − 5”的值。

解：$[-7]_{移}$ = 0001　　$[-6]_{移}$ = 0010　　$[-3]_{移}$ = 0101　　$[5]_{移}$ = 1101

$[-7]_{移} - [-6]_{移}$ = 0001 + 1110 = 1111， 符号取反后为 0111，其真值为−1。

$[-3]_{移} - [5]_{移}$ = 0101 + 0011 = 1000，符号取反后为 0000，其真值为− 8。

二、浮点乘除运算 6.4

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

1. 乘法

$$x \cdot y = (S_x \cdot S_y) \times 2^{j_x + j_y}$$

2. 除法

$$\frac{x}{y} = \frac{S_x}{S_y} \times 2^{j_x - j_y}$$

3. 步骤

(1) 阶码运算

(2) 尾数乘除运算

(3) 规格化

4. 浮点运算部件

阶码运算部件，尾数运算部件

(1) 阶码运算规则

①补码运算规则

②移码运算规则 $[x]_{移} = 2^n + j_x \quad (2^n > j_x \geq -2^n)$

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

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

又： $[j_y]_{补} = 2^{n+1} + j_y$

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

$$= [j_x + j_y]_{移} \quad (\mathrm{mod}\ 2^{n+1})$$

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

移码和补码的数值位相同、符号位相反

为防止溢出，采用双符号位的阶码加法器

规定：移码的最高符号位

恒用 “ 0 ” 参加运算

结果符号：
- 00 } 正常 负
- 01 } 正常 正
- ?
- 10 上溢
- 11 下溢

例： $x=011$ $y=110$ 求 $[x\pm y]_{移}$

$[x]_{移}=01,011$ $[y]_{补}=00,110$ $[-y]_{补}=11,010$

$[x+y]_{移}=[x]_{移}+[y]_{补}=01,011+00,110$

$=10,001$ （上溢，9）

$[x-y]_{移}=[x]_{移}+[-y]_{补}=01,011+11,010$

$=00101$ $(\mathrm{mod}\ 2^5, -3)$

例：已知$x=0.1011\times 2^{0110}$ $y=-0.0101\times 2^{0011}$，试用浮点数运算方法计算x*y。要求浮点数的格式为：阶码6位（2位阶符），补码表示，尾数5位（1位数符），补码表示，并要求为规格化浮点数。

解 ① 对x及y按所要求的浮点数格式编码

x已是规格化数，可直接编码

$$[x]_{浮}=00,0110;0.1011$$

Y不是规格化数，先将y化为规格化数再编码

$$y=-0.0101\times 2^{0011}=-0.1010\times 2^{0010}$$

$$[y]_{浮}=00,0010;1.0110$$

② 求x*y

- 阶码相加　　E=Ex+Ey=00,0110+00,0010=00,1000
- 尾数相乘　　$[M]_{补}=[Mx*My]_{补}$=11.10010010
- 　规格化　左规
  - 积的尾数M左移一位　　$[M]_{补}$=11.0010010
  - 积的阶码E减1　　E=00,1000+11,1111=00,0111
- 舍入　按“0”舍“1”入法，$[M]_{补}$=1.0010
- 检查是否溢出　　阶符为00,无溢出

∴ [x*y]浮=00,0111;1,0010　即x*y=-0.1110×2^{7}

浮点数运算总结

- 加减运算流程

  对阶 → 尾数相加减 → 规格化

  → 舍入操作 → 判断是否溢出

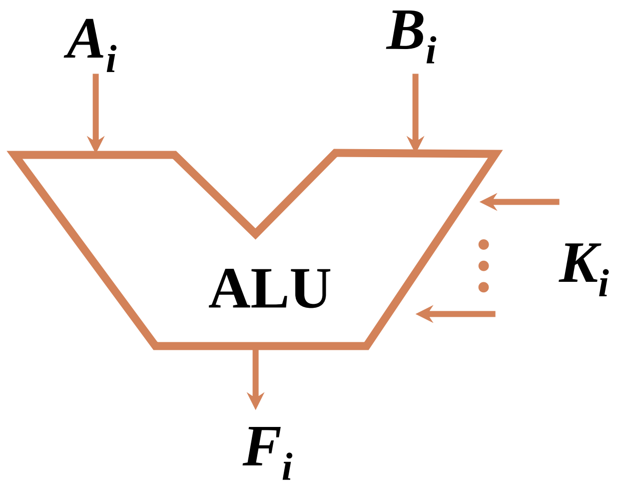
- 乘除运算流程

  阶码相加(减) →尾数相乘(除) →规格化

  →舍入操作 → 判断是否溢出

6.5 算术逻辑单元

一、ALU 电路



组合逻辑电路

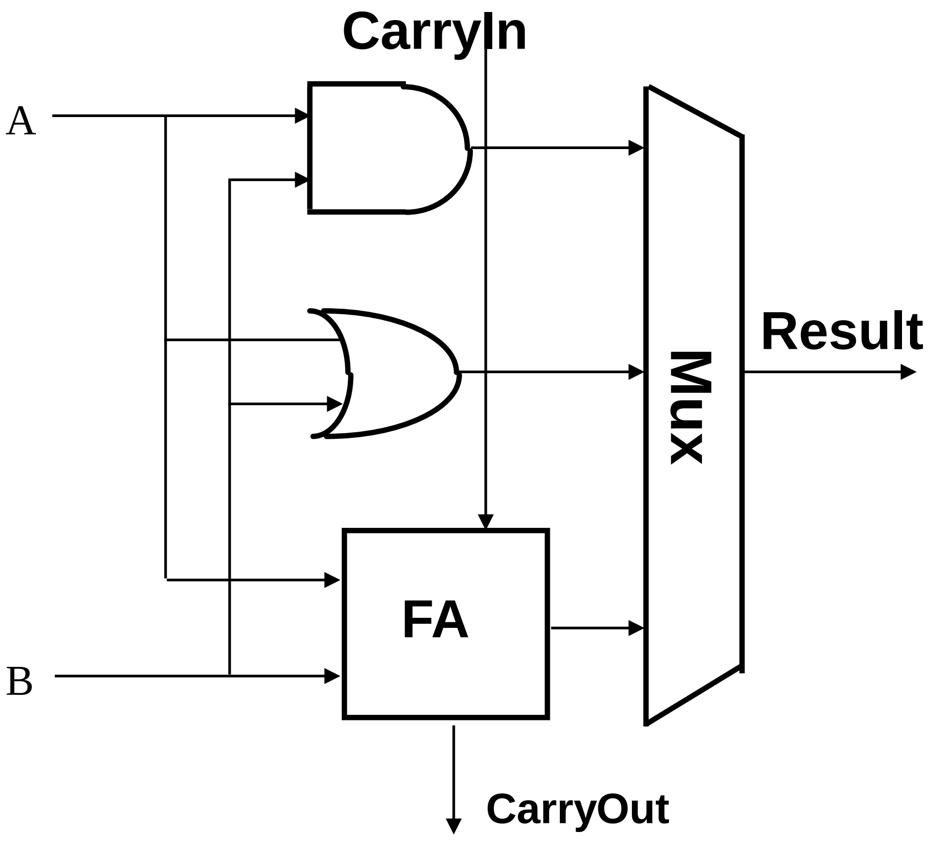
K_i 不同取值

F_i 不同

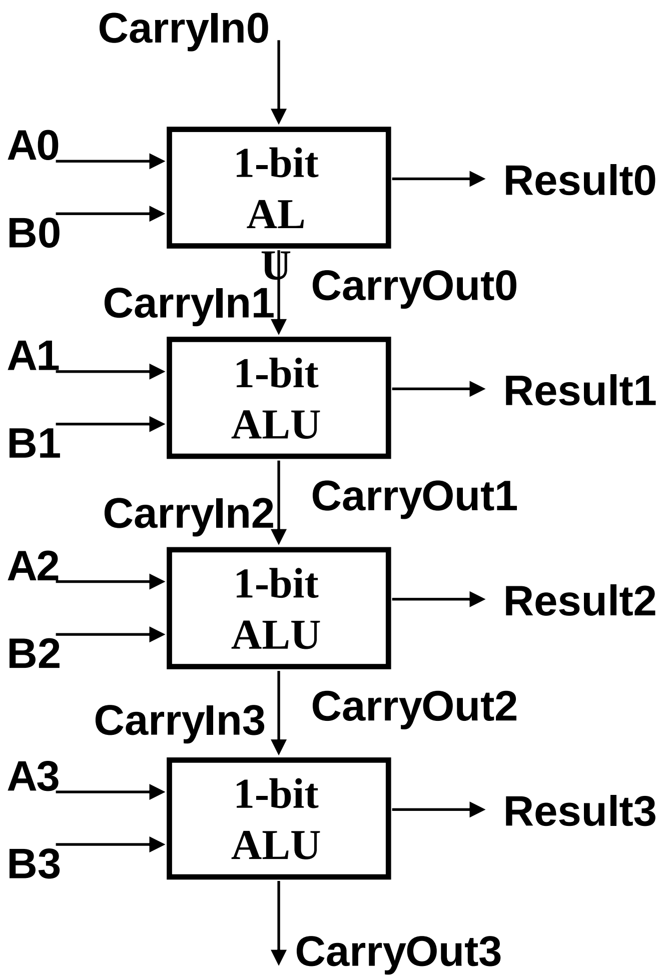
四位 ALU　74181

A 4-bit ALU

1-bit ALU



4-bit串行 ALU



MUX是什么？（数字电路课学过）

关键路径延迟长，速度慢！

先行进位ALU

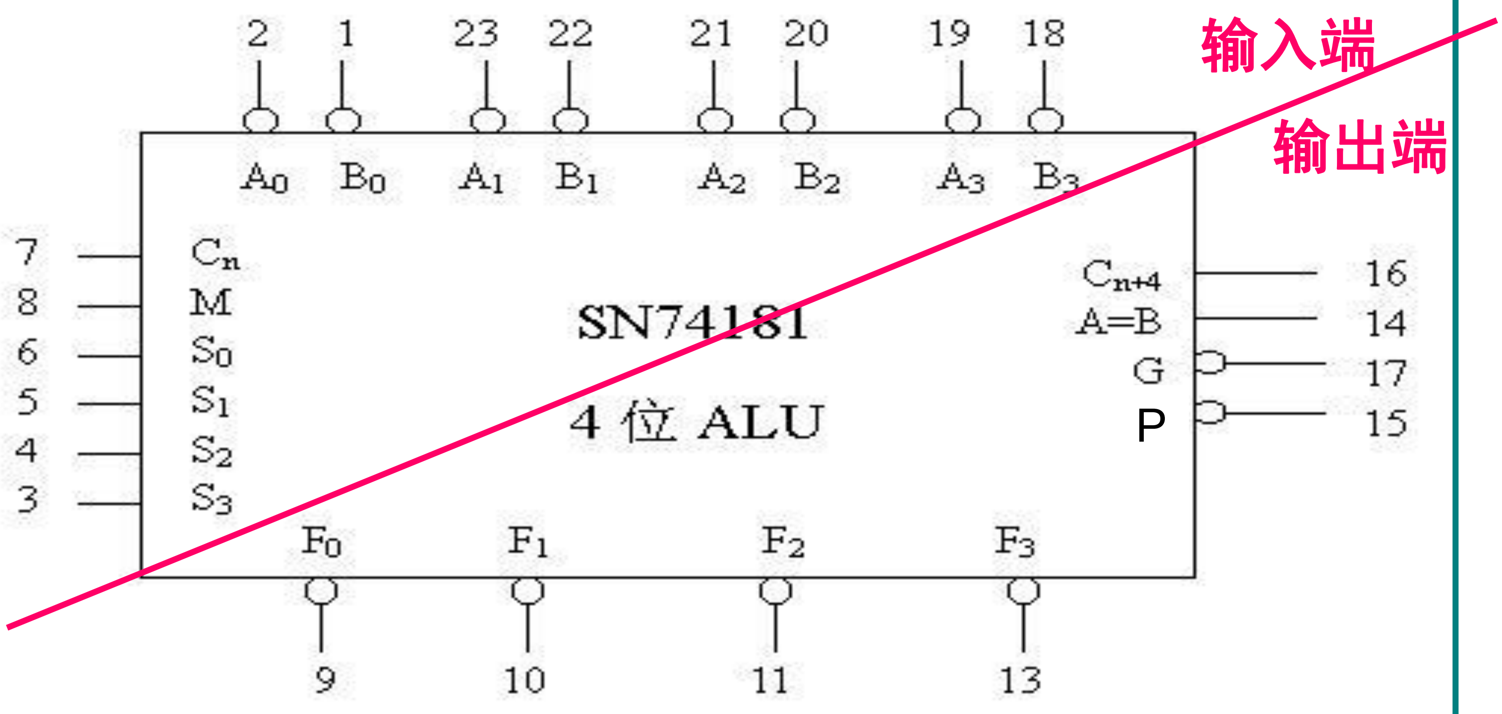
- **先行进位ALU 芯片（SN74181）**
  - 四位ALU芯片，中规模集成电路。在先行进位加法器基础上附加部分线路，具有基本的算术运算和逻辑运算功能。
  - SN74181的逻辑图和功能表
  - SN74182是4位BCLA (成组先行进位)芯片。
- **多芯片级联构成先行进位ALU（用于专用场合，如教学机等）**
  - 1个SN74181芯片直接构成一个4位全先行进位ALU
  - 4个SN74181芯片串行构成一个16位单级先行进位ALU
  - 4个SN74181芯片与1个SN74182芯片可构成16位两级先行进位ALU
  - 16个SN74181芯片与5个SN74182芯片可构成64位先行进位ALU
- **现代主流计算机中ALU是否通过芯片级联而成?**

无需芯片级联！一个CPU芯片中有多个处理器核，一个核中有多个ALU!

ALU的“加”运算电路相当于n档二进制加法算盘。所有其他运算都以ALU 中“加”运算为基础！

SKIP

回顾：SN74181的引脚



SN74181ALU 芯片负逻辑操作数

输入端： **Ai和Bi分别为第1和2操作数，Cn为低位进位，M为功能选择线，Si为操作选择线，共4位，故最多有16种运算。**

输出端： **Fi为运算结果，C_{n+4}、P和G为进位，“A=B”为相等标志**

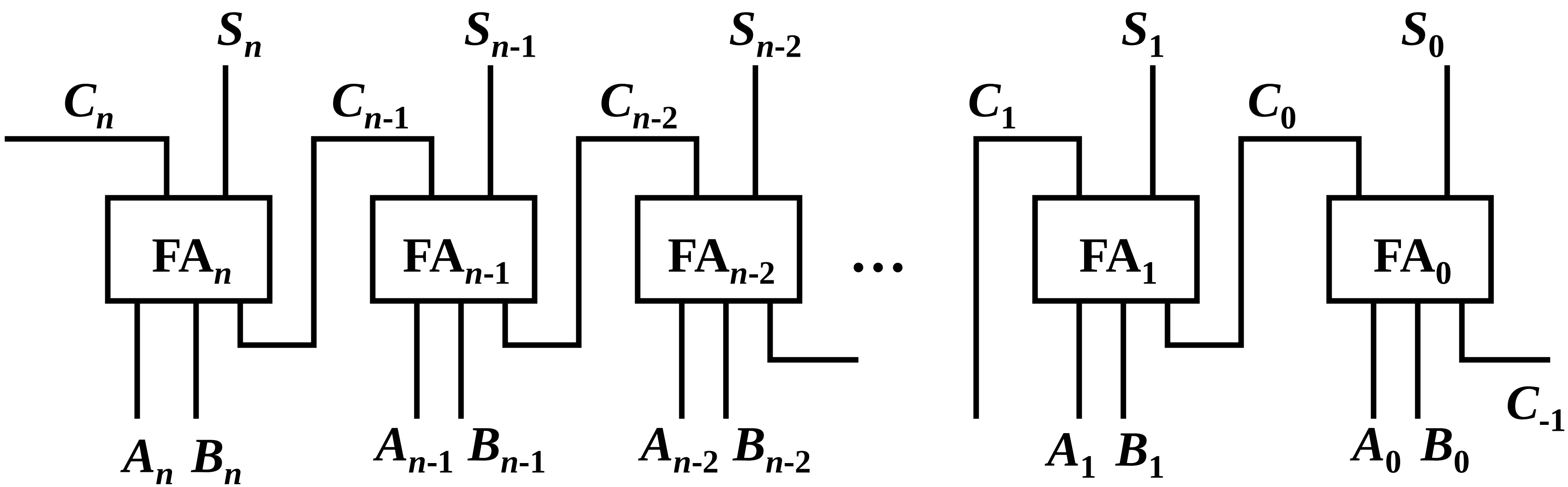
$M = 0$　　算术运算　　$M = 1$　　逻辑运算

74181的算术/逻辑运算功能（正逻辑方式）

工作方式选择	逻辑运算	算术运算 M=L	
S_3 S_2 S_1 S_0	M=H	C_n=1	C_n=0
0 0 0 0	$\overline{A}$	A	A+1
0 0 0 1	$\overline{A+B}$	A+B	(A+B)加 1
0 0 1 0	$\overline{A}\cdot B$	$A+\overline{B}$	($A+\overline{B}$)加 1
0 0 1 1	“0”	减 1	“0”
0 1 0 0	$\overline{A\cdot B}$	A 加 ($A\cdot\overline{B}$)	A 加 ($A\cdot\overline{B}$)加 1
0 1 0 1	$\overline{B}$	($A\cdot\overline{B}$) 加 (A+B)	($A\cdot\overline{B}$) 加 (A+B) 加 1
0 1 1 0	$A\oplus B$	A 减 B 减 1	A 减 B
0 1 1 1	$A\cdot\overline{B}$	($A\cdot\overline{B}$) 减 1	$A\cdot\overline{B}$
1 0 0 0	$\overline{A}+B$	A 加 ($A\cdot B$)	A 加 ($A\cdot B$) 加 1
1 0 0 1	$\overline{A\oplus B}$	A 加 B	A 加 B 加 1
1 0 1 0	B	($A\cdot B$) 加 ($A+\overline{B}$)	($A\cdot B$) 加 ($A+\overline{B}$) 加 1
1 0 1 1	$A\cdot B$	($A\cdot B$) 减 1	$A\cdot B$
1 1 0 0	“1”	A 加 A	A 加 A 加 1
1 1 0 1	A+B	A 加 (A+B)	A 加 (A+B) 加 1
1 1 1 0	A+B	A 加 ($A+\overline{B}$)	A 加 ($A+\overline{B}$) 加 1
1 1 1 1	A	A 减 1	A

二、 快速进位链

1. 并行加法器



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

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

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

$d_i = A_i\,B_i$ 本地进位 $t_i = A_i + B_i$ 传送条件

则 $C_i = d_i + t_i C_{i-1}$

2. 串行进位链

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

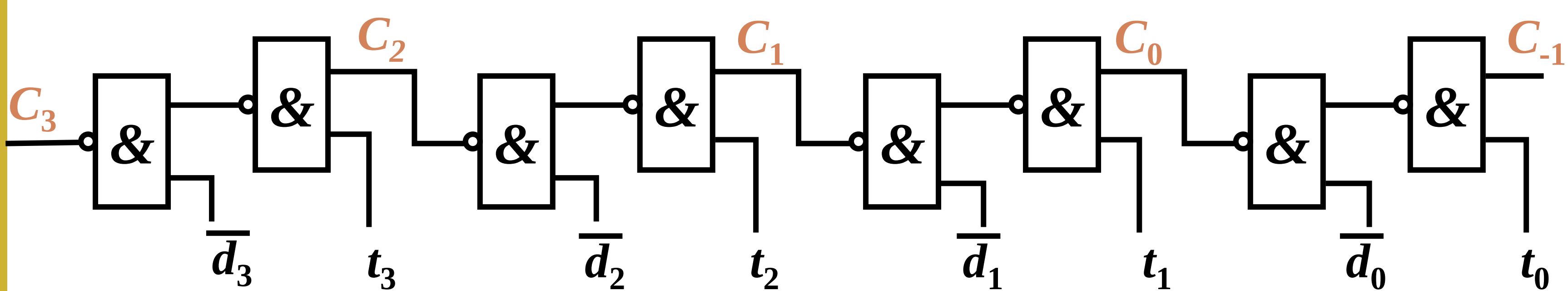
$$C_0 = d_0 + t_0C_{-1} = \overline{\overline{d_0} \cdot \overline{t_0C_{-1}}}$$

$$C_1 = d_1 + t_1C_0$$

$$C_2 = d_2 + t_2C_1$$

$$C_3 = d_3 + t_3C_2$$

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）

n 位加法器的进位同时产生　　以 4 位加法器为例

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

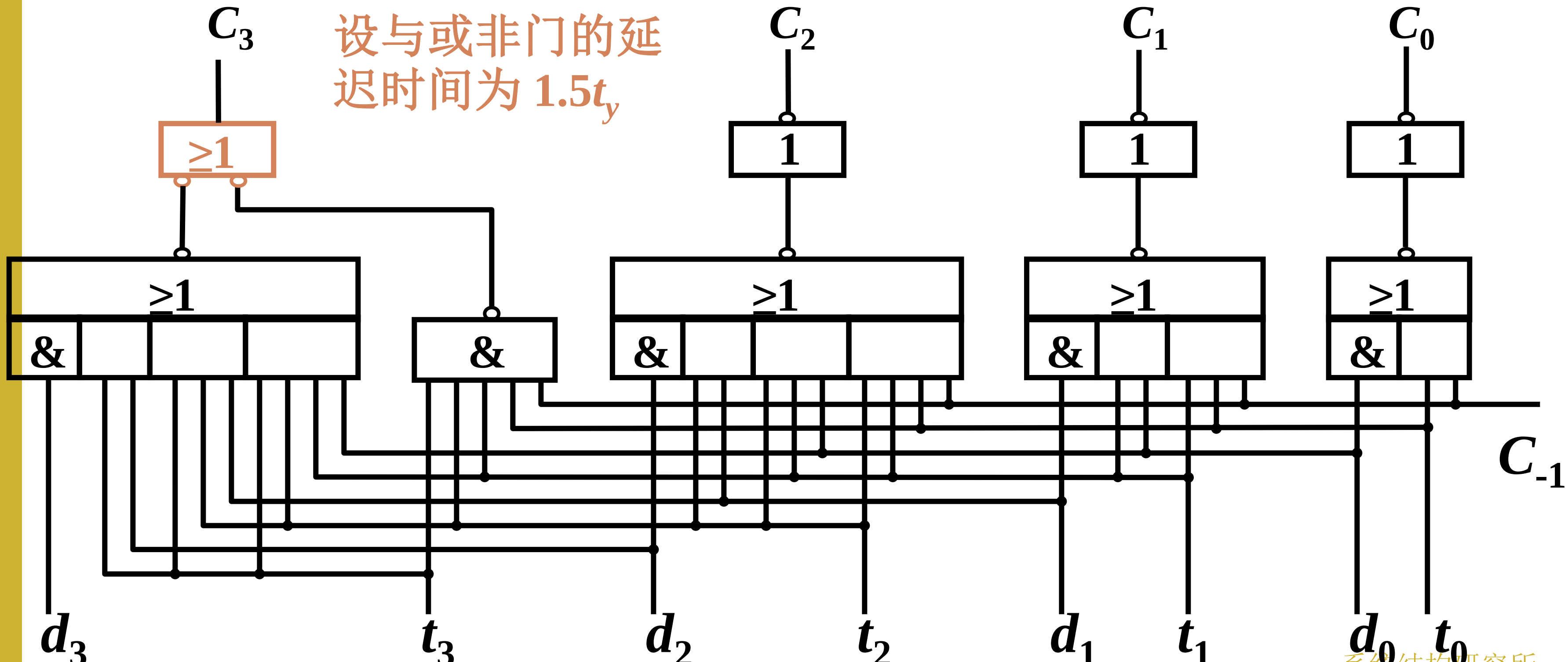
$$C_1 = d_1 + t_1 C_0 = d_1 + t_1 d_0 + t_1 t_0 C_{-1}$$

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

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

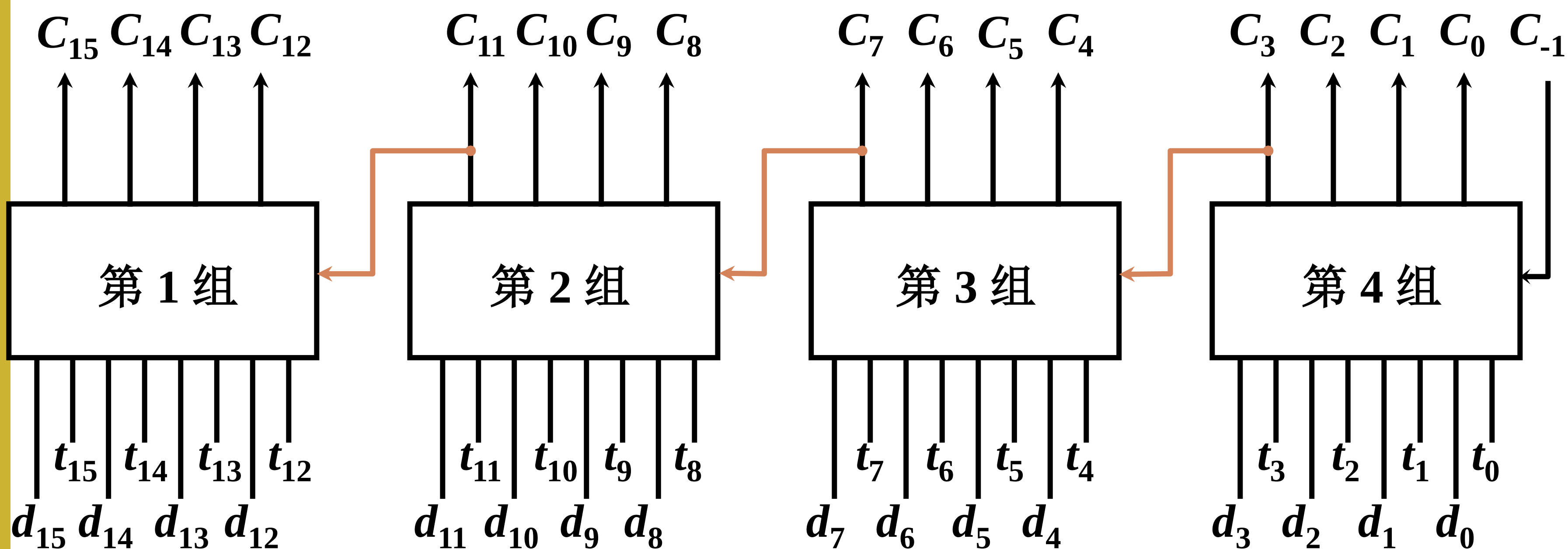
当 $d_i\, t_i$ 形成后，只需 $2.5t_y$ 产生全部进位

设与或非门的延迟时间为 $1.5t_y$



(1) 单重分组跳跃进位链

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 $C_7 \sim C_4$

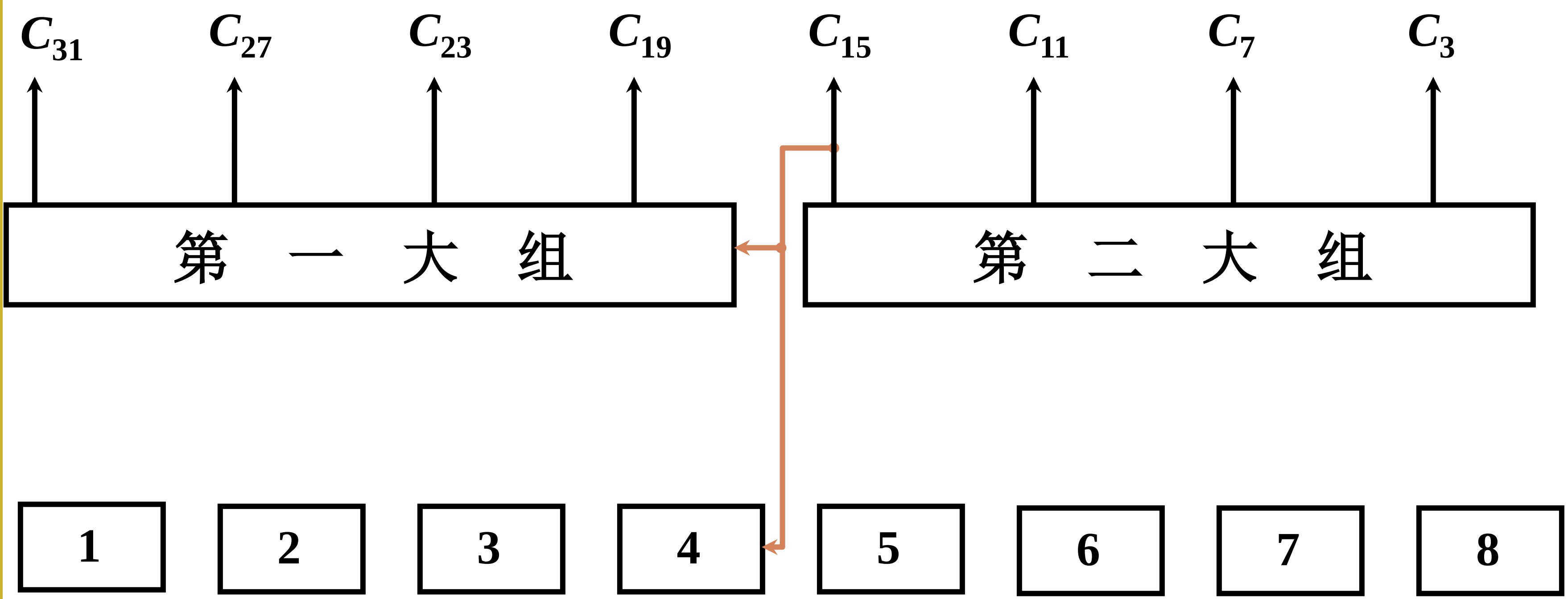
$7.5\ t_y$　产生 $C_{11} \sim C_8$

$10\ t_y$　产生 $C_{15} \sim C_{12}$

(2) 双重分组跳跃进位链

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

以第 8 小组为例

$$C_3 = d_3 + t_3C_2 = \underbrace{d_3 + t_3d_2 + t_3t_2d_1 + t_3t_2t_1d_0}_{D_8} + \underbrace{t_3t_2t_1t_0}_{T_8}C_{-1}$$

$$= D_8 + T_8C_{-1}$$

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 $C_7 = D_7 + T_7C_3$

第 6 小组 $C_{11} = D_6 + T_6C_7$

第 5 小组 $C_{15} = D_5 + T_5C_{11}$

进一步展开得

$$C_3 = D_8 + T_8C_{-1}$$

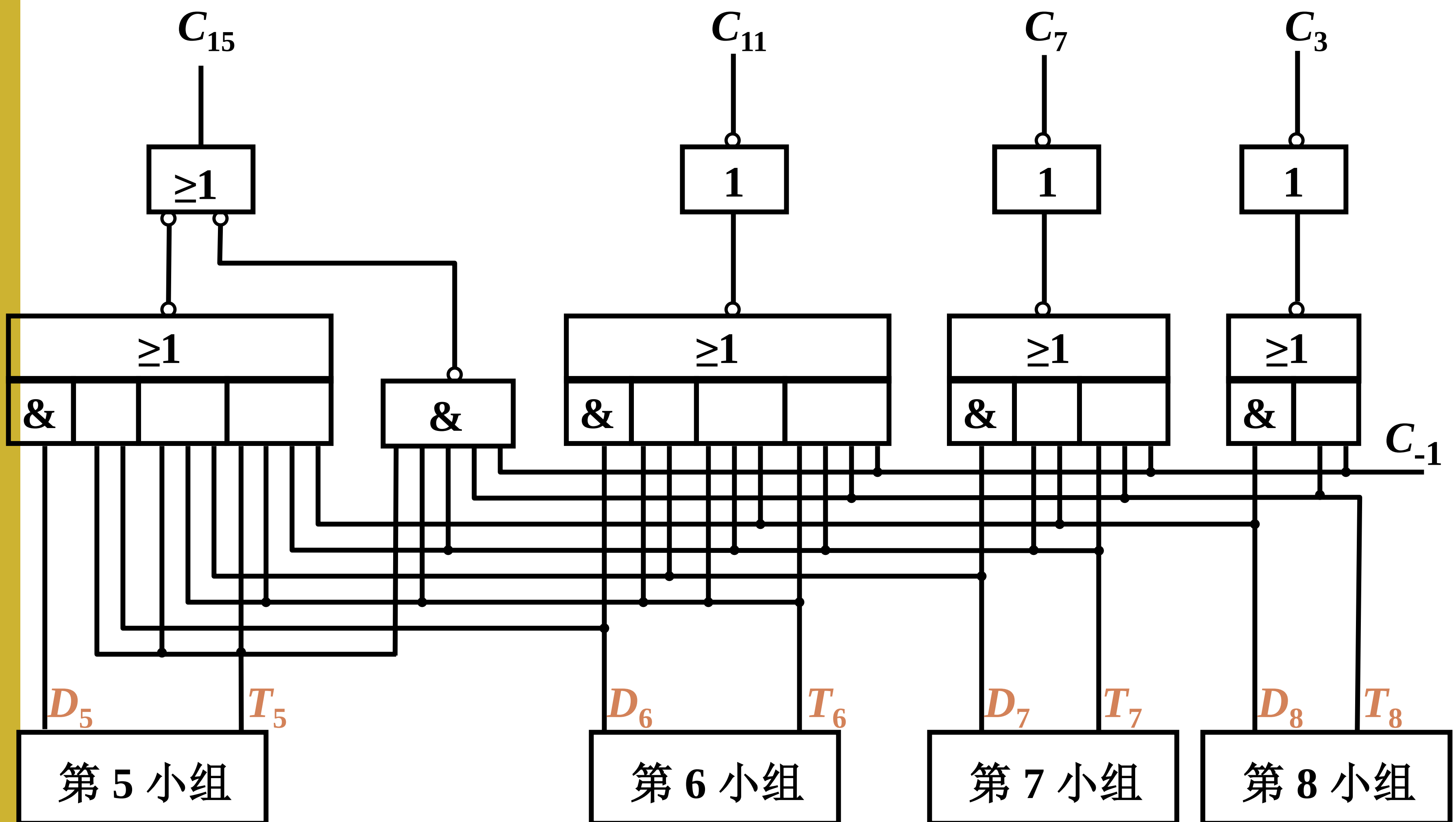
$$C_7 = D_7 + T_7C_3 = D_7 + T_7D_8 + T_7T_8C_{-1}$$

$$C_{11} = D_6 + T_6C_7 = D_6 + T_6D_7 + T_6T_7D_8 + T_6T_7T_8C_{-1}$$

$$C_{15} = D_5 + T_5C_{11} = D_5 + T_5D_6 + T_5T_6D_7 + T_5T_6T_7D_8 + T_5T_6T_7T_8C_{-1}$$

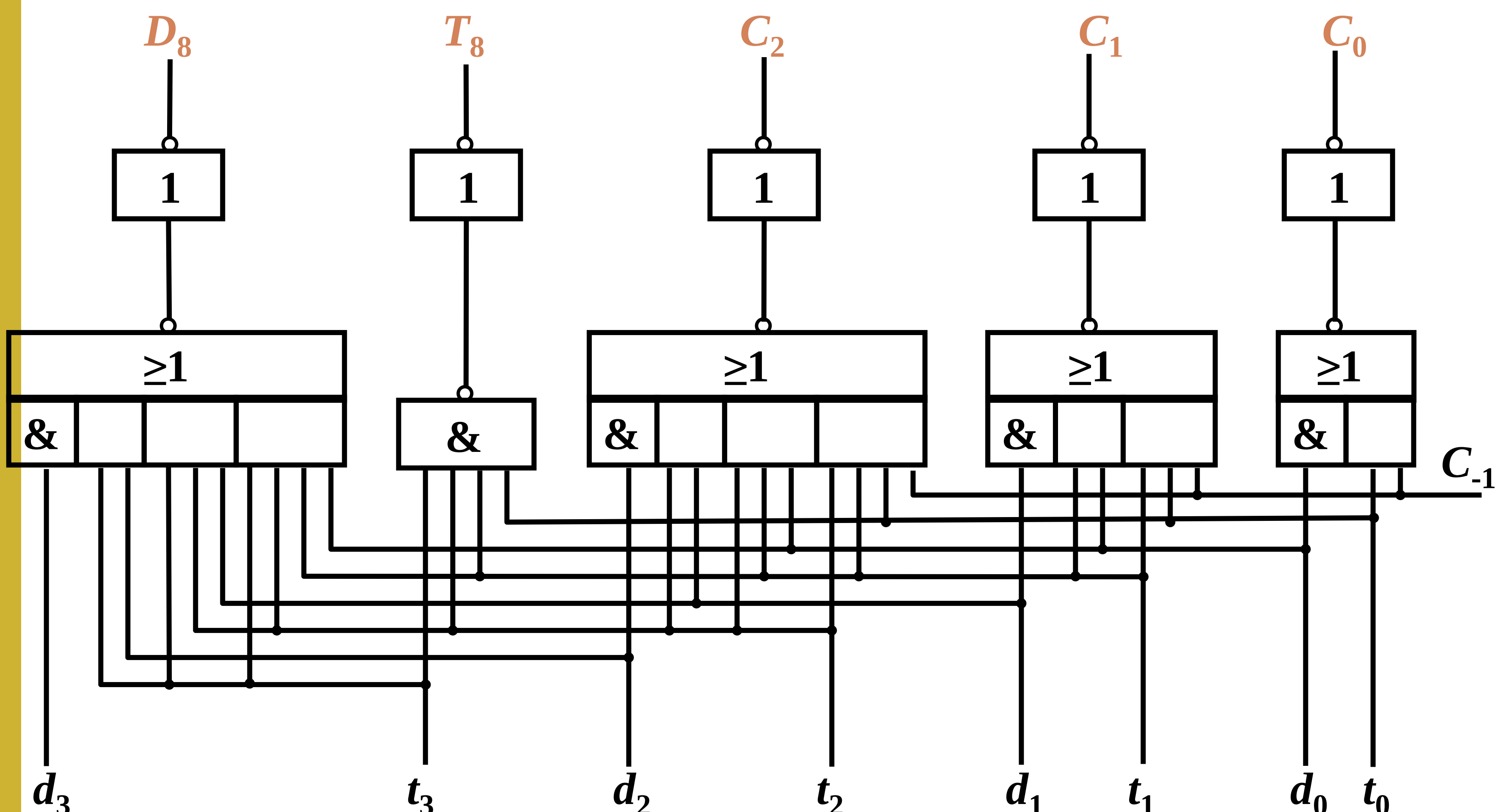
(4) 双重分组跳跃进位链的 大组 进位线路

以第 2 大组为例

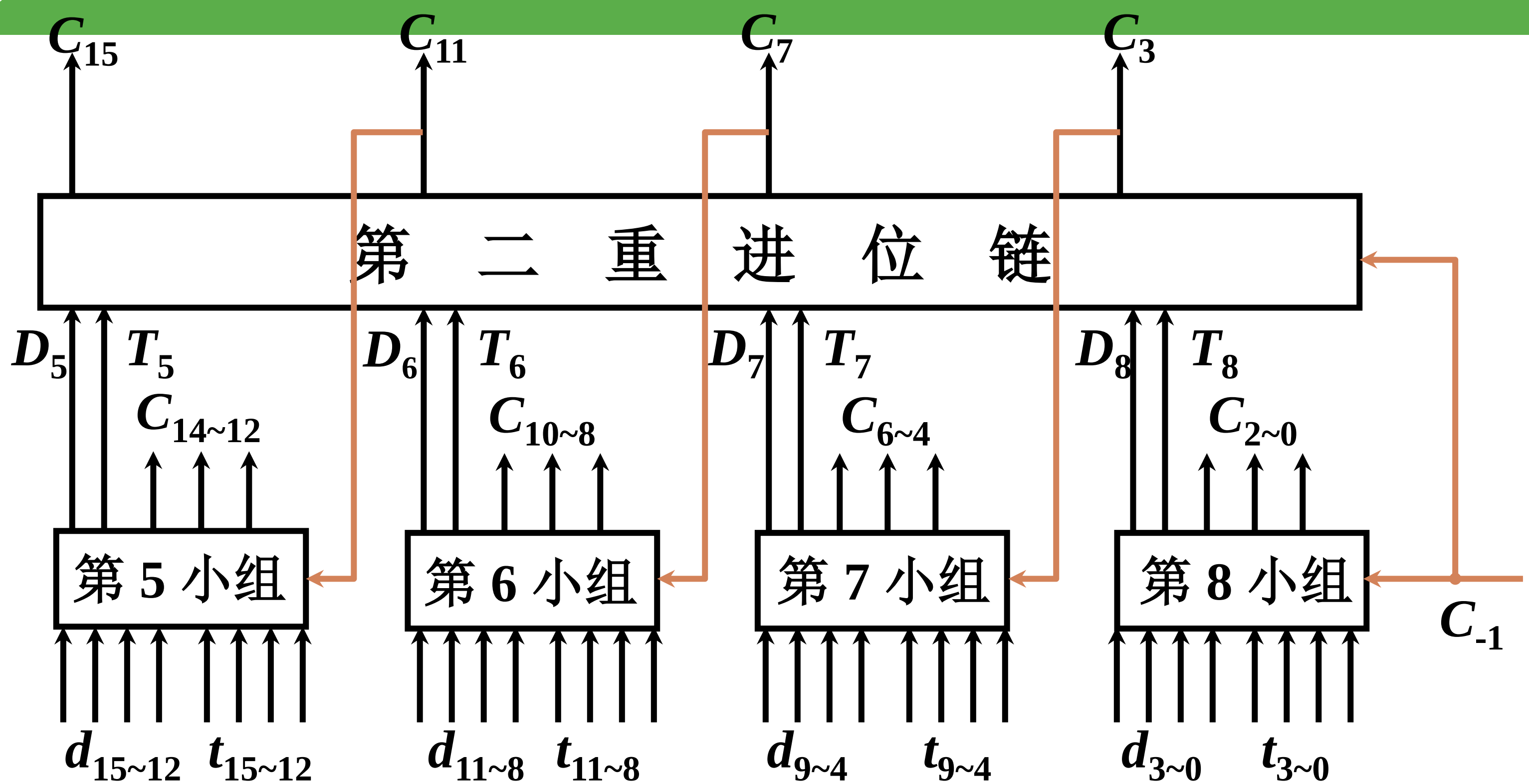


(5) 双重分组跳跃进位链的 小组 进位线路

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、$D_5 \sim D_8$、$T_5 \sim T_8$

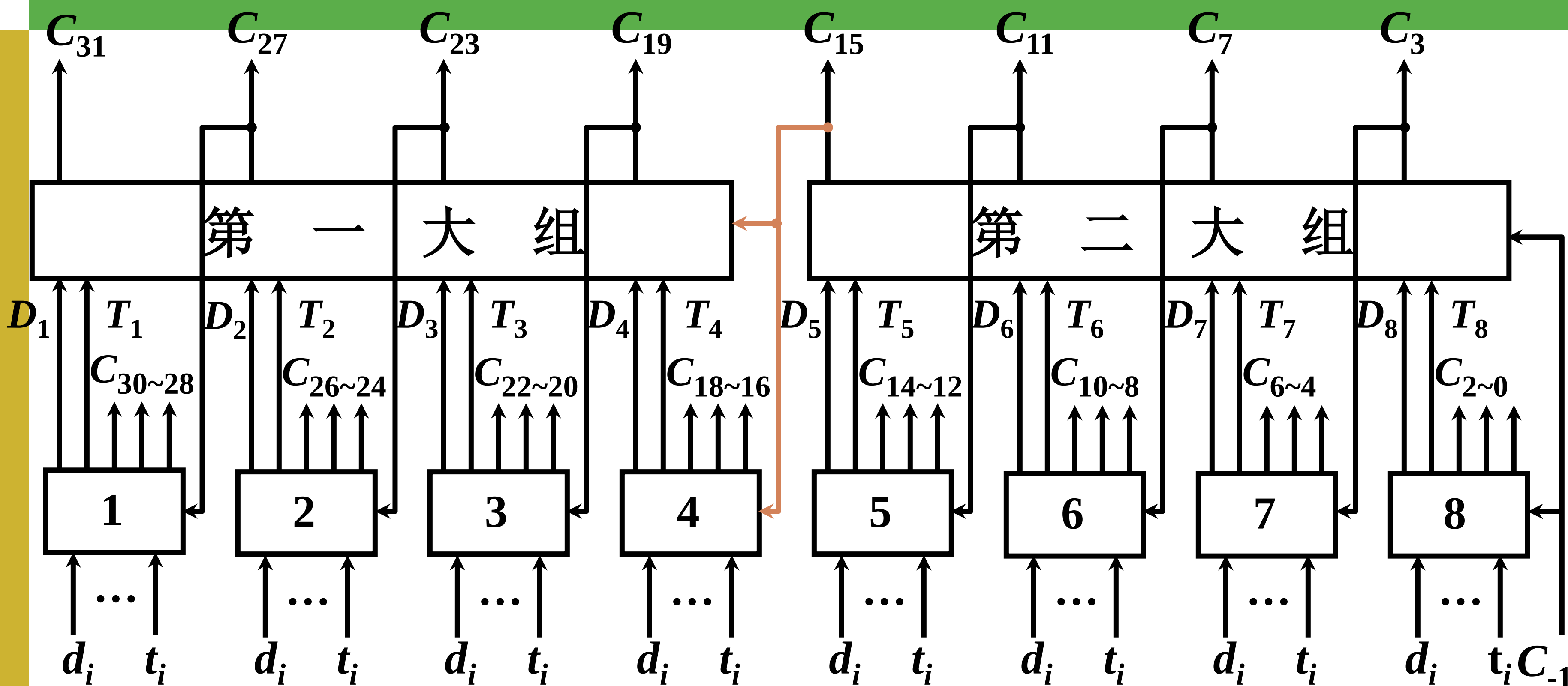
经　5 t_y　产生 C_{15}、C_{11}、C_7、C_3

经 7.5 t_y　产生 C_{14}~C_{12}、C_{10}~C_8、C_6~C_4

串行进位链　经 3 2 t_y　产生　全部进位

单重分组跳跃进位链　经 1 0 t_y　产生　全部进位

(7) n =32 双重分组跳跃进位链



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 $C_{30}\sim C_{28}$、$C_{26}\sim C_{24}$、$C_{22}\sim C_{20}$

计算机组成

Thank You !

例6.23 已知 $x = -0.1011$ $y = -0.1101$ 求$[x \cdot y]_补$ 6.3

解：

00.0000	1.0011	0	
00.1011			$+[-x]_补$
101			
11.1110	1 1010	1	$\rightarrow 1$
00.0011			$+[x]_补$
00.0001	1		
00.0000	11 101	0	$\rightarrow 1$
11.1101			$+[-x]_补$
11.1101	11		
11.1110	111 10	1	$\rightarrow 1$
00.0011			$+[x]_补$
00.0001	111		
00.0000	1111 1	0	$\rightarrow 1$
11.1101			$+[-x]_补$
11.1101	1111		最后一步不移位

$[x]_补 = 1.0101$

$[y]_补 = 1.0011$

$[-x]_补 = 0.1011$

$\therefore [x \cdot y]_补$

$= 1.11011111$