



# 《数字电路与微机系统》

王博、王志、辛博

南京大学

---



- 《数字电路与微机系统》共6学分（教学4，实验2）
  - 教学时间：周二上午3-4节、下午7-8节
  - 教学地点：仙1-317
  - 实验时间：周一下午5-8节（第三周开始）
  - 实验地点：基础实验楼乙区435
- 成绩要求
  - 作业：每次布置作业后，下一次上课时交
  - 实验：必须按辛博老师规定时间提交实验报告
  - 考试：第17-18周（学校统一安排）
  - 成绩：平时作业15% + 实验25% + 考试60%

# 《数字电路》部分

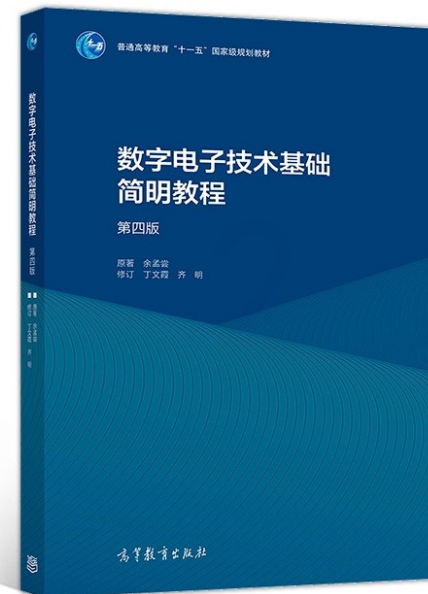
## ■ 教材

- 《数字电子技术基础简明教程（第4版）》，高等教育出版社，余孟尝 主编

- 课件下载网址：

<https://heyuanmingong.github.io/teaching-dc.html>

高等教育出版社  
HIGHER EDUCATION PRESS



## ■ 参考书

- 《数字逻辑与计算机组成》，机械工业出版社，袁春风 主编
- 《数字电路与逻辑设计（Verilog HDL&Vivado版）》，清华大学出版社，汤勇明 张圣清 陆佳华 编著

# 数字电路及其特点

## ■ 信息处理

- 在人类生存的自然环境中，存在这各种各样的信息，例如声音、温度、湿度、压力和流量等
- 这些信息可以通过相应的传感器（Sensor）转换为电信号，被输入到电子系统中进行处理与传输

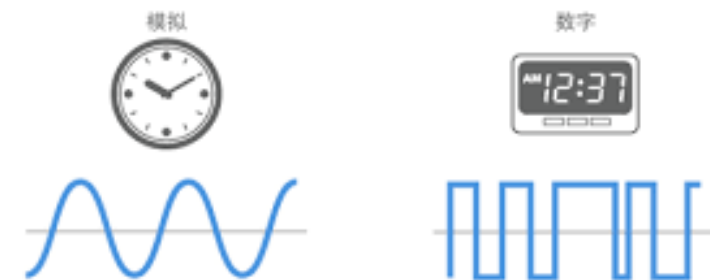
## ■ 电子电路中，电信号分为模拟信号和数字信号

- 模拟信号（Analog signal）：信号幅度随时间连续变化，即在时间和数值上均具连续性
- 处理模拟信号的电子电路称为模拟电路



# 数字电路及其特点

- 电子电路中，电信号分为**模拟信号**和**数字信号**
  - 数字信号（Digital signal）：在幅度和时间上都是离散的、突变的信号
  - 通常采用的数字信号具有双值性，低电平用0表示，高电平用1表示
- 数字电路就是用于传输和处理数字信号的电子电路
  - 主要研究输入与输出信号之间的因果关系，一般称为**逻辑关系**
  - 因此，数字电路又称为**数字逻辑电路**



# 数字电路及其特点

- 电子电路中，电信号分为模拟信号和数字信号
  - 模拟信号和数字信号可以通过模/数转换电路（ADC, analog to digital converter）或数/模转换电路（DAC, digital to analog converter）进行转换
- 与模拟电路相比，数字电路主要有如下特点
  - 电路结构简单，便于集成化
  - 在数字电路中晶体管均工作在饱和区或截止区，工作在开关状态，因而数字电路的抗干扰能力，可靠性高
  - 数字信息便于长期保存和加密
  - 数字集成电路产品系列全，通用性强，成本低
  - 数字电路不仅能完成数值运算，还能进行逻辑判断等

# 逻辑代数：数字电路的基础数学工具

## ■ 逻辑代数

- 由英国数学家 George Boole 在19世纪中叶创立，因此也称“布尔代数”
- 直到20世纪30年代，美国人 Claude E. Shannon 在开关电路中才找到它的用途，并且很快成为分析和综合开关电路的重要数学工具，因此也称“开关代数”

## ■ 逻辑代数与普通代数

- 逻辑代数也用英文字母表示变量，但情况要简单得多
- 在二值逻辑中，变量取值不是1就是0，没有第三种可能
- 0和1并不表示数值的大小，只代表两种不同的逻辑状态
- 例如，用1和0分别表示一件事的是与非、真与假，电压的高与低，电流的有与无，一个开关的通与断，一盏电灯的亮与灭

## 一. 数制

1. 进位计数制
2. 二进制数与其他计数制数之间的转换
3. 二进制数的算术运算

## 二. 编码

1. 数值型数据的编码表示
  - ① 定点数的编码：原码、补码、反码、移码
  - ② 整数的表示
  - ③ 浮点数的表示
  - ④ 十进制数的二进制编码表示
2. 非数值型数据的编码表示



- 对现实世界中的感觉媒体信息（声音、文字、静止图像、活动图像等）进行定时采样
- 将连续信息转换为计算机中的离散“样本”信息，然后对它们用“0”和“1”进行数字化编码



计算机的外部信息和内部数据

# 1.1 进位计数制

- 所谓“数制”，指进位计数制，即用进位的方法来计数
  - 数制指多位数码中每一位的构成方法以及从低位到高位进位的规则
- 数制包括计数符号（数码）和进位规则两个方面
- 常用数制有二进制、十进制、十二进制、十六进制、六十进制等

# 常用数制 - 十进制

- 计数符号: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- 进位规则: 逢十进一

- 十进制数按权展开式

$$(N)_{10} = \sum_{i=-m}^n a_i \times 10^i$$

系数  $\xrightarrow{\quad}$   $\uparrow$   $\nwarrow$  权

例:  $(1987.45)_{10} = 1 \times 10^3 + 9 \times 10^2 + 8 \times 10^1 + 7 \times 10^0 + 4 \times 10^{-1} + 5 \times 10^{-2}$

- 计数符号: 0, 1
- 进位规则: 逢二进一

- 二进制数按权展开式

$$(N)_2 = \sum_{i=-m}^n a_i \times 2^i$$

系数  $\xrightarrow{\quad}$   $\uparrow$   $\quad$   $\uparrow$  权

例:  $(1001.01)_2 = 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$   
 $+ 0 \times 2^{-1} + 1 \times 2^{-2}$

- 计数符号: 0, 1, 2, 3, 4, 5, 6, 7
- 进位规则: 逢八进一

- 八进制数按权展开式

$$(N)_8 = \sum_{i=-m}^n a_i \times 8^i$$

系数  $\xrightarrow{\quad}$   $\uparrow$   $\quad \uparrow$  权

例:  $(63.45)_8 = 6 \times 8^1 + 3 \times 8^0 + 4 \times 8^{-1} + 5 \times 8^{-2}$

# 常用数制 - 十六进制

- 计数符号: 0, 1, ..., 9, A, B, C, D, E, F

- 进位规则: 逢十六进一

- 十六进制数按权展开式

$$(N)_{16} = \sum_{i=-m}^n a_i \times 16^i$$

系数  $\xrightarrow{\quad}$   $\uparrow$   $\quad$   $\uparrow$  权

例:  $(6D.4B)_{16} = 6 \times 16^1 + D \times 16^0 + 4 \times 16^{-1} + B \times 16^{-2}$   
 $= 6 \times 16^1 + 13 \times 16^0 + 4 \times 16^{-1} + 11 \times 16^{-2}$

# 扩展到一般情况 - $R$ 进制

- 计数符号:  $0, 1, \dots, R - 1$

- 进位规则: 逢  $R$  进一

- 在书写时, 可使用后缀字母标识该数的进位计数制

- B (Binary) 表示二进制, 例如 10011B

- O (Octal) 表示八进制, 例如 637O

- D (Decimal) 表示十进制, 十进制数的后缀可以省略, 例如 56D 或者 56

- H (Hexadecimal) 表示十六进制, 也可以在十六进制数之前用 0x 作为前缀, 例如 308FH 或者 0x308F

- 十进制数按权展开式

$$(N)_2 = \sum_{i=-m}^n a_i \times R^i$$

系数  $\xrightarrow{\quad}$   $\uparrow$   $\quad \uparrow$  权

# 扩展到一般情况 - $R$ 进制

二进制数	八进制数	十进制数	十六进制数
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	10	A
1011	13	11	B
1100	14	12	C
1101	15	13	D
1110	16	14	E
1111	17	15	F



# 十进制与二进制

- 为什么人类普遍实行了十进制?
- 为什么计算机系统采用二进制?
  - 二进制只有两种基本状态
    - 使用有两个稳定状态的物理器件就可以表示二进制数的每一位
    - 制造有两个稳定状态的物理器件比制造有多个稳定状态的器件容易得多
    - 高、低两个电位，脉冲的有、无，脉冲的正、负极等，方便可靠地表示0和1
  - 二进制的编码、计数和运算规则简单
    - 可以用开关电路实现，简便易行
  - 1、0与逻辑命题中的真、假相对应
    - 为计算机中的逻辑运算实现和程序中的逻辑判断提供了便利的条件
    - 能通过逻辑门电路方便地实现算术运算

## 一. 数制

1. 进位计数制
2. 二进制数与其他计数制数之间的转换
3. 二进制数的算术运算

## 二. 编码

1. 数值型数据的编码表示
  - ① 定点数的编码：原码、补码、反码、移码
  - ② 整数的表示
  - ③ 浮点数的表示
  - ④ 十进制数的二进制编码表示
2. 非数值型数据的编码表示

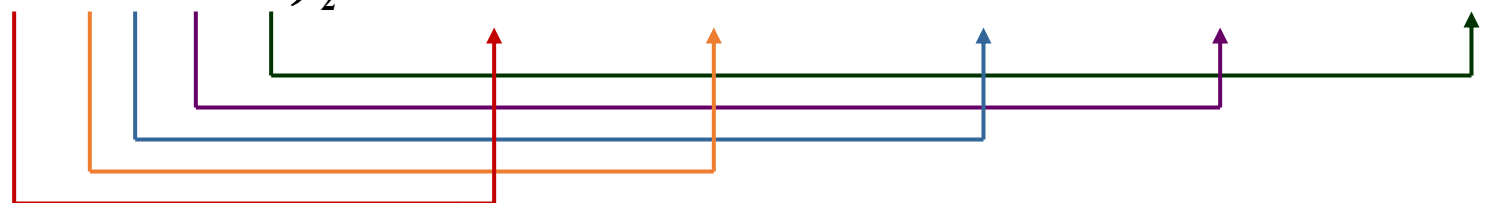
## 2.2 二进制数与其他计数制数之间的转换

- 计算及内部所有信息采用二进制编码表示
- 计算机外部，为了书写和阅读的方便，大多采用十进制或十六进制表示形式
- 因此，计算机在数据输入后或输入前都必须实现这些进制数和二进制数之间的转换

# $R$ 进制数转换为十进制数

- 任何一个 $R$  进制数转换成十进制数时，只要“按权展开法”即可

例：

$$(1011.101)_2 = 1 \times 2^3 + 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-3}$$

$$= 8 + 2 + 1 + 0.5 + 0.125$$

例：

$$(3A.C)_{16} = 3 \times 16^1 + 10 \times 16^0 + 12 \times 16^{-1} = 58.75$$

# 十进制数转换为 $R$ 进制数

- 任何一个十进制数转换成  $R$  进制数时，要将整数和小数部分分别进行转换

## 1. 整数部分转换：“除基取余，上低下高”

- 要转换的十进制整数除以基数  $R$ ，将得到的余数作为结果数据中各位的数字，直到上商为0为止
- 上面的余数（先得到的余数）作为右边低位上的数位，下面的余数作为左边高位上的数位

# 十进制数转换为 $R$ 进制数

## 1. 整数部分转换：“除基取余，上低下高”

- 要转换的十进制整数除以基数 $R$ ，将得到的余数作为结果数据中各位的数字，直到上商为0为止
- 上面的余数（先得到的余数）作为右边低位上的数位，下面的余数作为左边高位上的数位

**例 1.3** 将十进制整数 135 分别转换成八进制数和二进制数。

**解** 将 135 分别除以 8 和 2，将每次的余数按从低位到高位顺序排列如下：

	余数	低位
8   135	7	↑ 低位  高位
8   16	0	
8   2	2	
0		

	余数	低位
2   135	1	↑ 低位  高位
2   67	1	
2   33	1	
2   16	0	
2   8	0	
2   4	0	
2   2	0	
2   1	1	

所以， $135 = 207O = 1000\ 0111B$ 。

# 十进制数转换为 $R$ 进制数

## 1. 整数部分转换：“除基取余，上低下高”

- 要转换的十进制整数除以基数 $R$ ，将得到的余数作为结果数据中各位的数字，直到上商为0为止
- 上面的余数（先得到的余数）作为右边低位上的数位，下面的余数作为左边高位上的数位

课堂练习：将十进制整数1024分别转换成八进制数和二进制数。

# 十进制数转换为 $R$ 进制数

- 任何一个十进制数转换成  $R$  进制数时，要将整数和小数部分分别进行转换

## 2. 小数部分转换：“乘基取整，上高下低”

- 要转换的十进制小数乘以基数  $R$ ，将得到的乘积的整数部分作为结果中各位的数字，小数部分继续与基数  $R$  相乘；以此类推，直到某一步乘积的小数部分为0或已得到希望的位数为止
- 上面的整数部分（先得到的整数）作为左边高位上的数位，下面的整数部分作为右边低位上的数位



# 十进制数转换为 $R$ 进制数

## 2. 小数部分转换：“乘基取整，上高下低”

- 要转换的十进制小数乘以基数 $R$ ，将得到的乘积的整数部分作为结果中各位的数字，小数部分继续与基数 $R$ 相乘；以此类推，直到某一步乘积的小数部分为0或已得到希望的位数为止
- 上面的整数部分（先得到的整数）作为左边高位上的数位，下面的整数部分作为右边低位上的数位

**例 1.4** 将十进制小数 0.6875 分别转换成二进制数和八进制数。

**解**  $0.6875 \times 2 = 1.375$       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)

所以,  $0.6875 = 0.1011\text{B}$ 。

$0.6875 \times 8 = 5.5$       整数部分 = 5      (高位)

$0.5 \times 8 = 4.0$       整数部分 = 4      (低位)

所以,  $0.6875 = 0.54\text{O}$ 。

# 十进制数转换为 $R$ 进制数

## 2. 小数部分转换：“乘基取整，上高下低”

- 要转换的十进制小数乘以基数 $R$ ，将得到的乘积的整数部分作为结果中各位的数字，小数部分继续与基数 $R$ 相乘；以此类推，直到某一步乘积的小数部分为0或已得到希望的位数为止
- 上面的整数部分（先得到的整数）作为左边高位上的数位，下面的整数部分作为右边低位上的数位

课堂练习：将十进制小数0.1024分别转换成八进制数和二进制数。

## 二、十六进制数的相互转换

### ■ 十六进制数转二进制数

- 把每一个十六进制数改写成等值的4位二进制数，且保持高低位的次序不变

$$(0)_{16} = 0000 \quad (1)_{16} = 0001 \quad (2)_{16} = 0010 \quad (3)_{16} = 0011$$

$$(4)_{16} = 0100 \quad (5)_{16} = 0101 \quad (6)_{16} = 0110 \quad (7)_{16} = 0111$$

$$(8)_{16} = 1000 \quad (9)_{16} = 1001 \quad (A)_{16} = 1010 \quad (B)_{16} = 1011$$

$$(C)_{16} = 1100 \quad (D)_{16} = 1101 \quad (E)_{16} = 1110 \quad (F)_{16} = 1111$$

例：将十六进制数 2B.5EH 转换成二进制数。

解：2B.5EH = 0010 1011 . 0101 1110 B = 101011.0101111B

## 二、十六进制数的相互转换

### ■ 二进制数转十六进制数

- 整数部分从低位向高位方向每四位用一个等值的十六进制数字来替换，最后不足四位时在高位补0凑满四位
- 小数部分从高位到低位方向每四位用一个等值的十六进制数字来替换，最后不足四位时在低位补0凑满四位

例：将二进制数 11001.11B 转换成十六进制数。

解：  $11001.11\text{B} = 0001\ 1001\ 1100\text{B} = 19.\text{CH}$

## 二、十六进制数的相互转换

- 二进制数与十六进制数之间有简单、直观的对应关系
  - 如果要将十进制数转换为十六进制数，可以先转为二进制数，再将二进制数转为十六进制数
- 计算机中只使用二进制一种计数制
  - 但二进制数太长，书写、阅读均不方便
- 十六进制像十进制数一样简练，易写易记
  - 为了在开发和调试程序、查看机器代码时便于书写和阅读，人们经常使用十六进制来等价地表示二进制

## 一. 数制

1. 进位计数制
2. 二进制数与其他计数制数之间的转换
3. 二进制数的算术运算

## 二. 编码

1. 数值型数据的编码表示
  - ① 定点数的编码：原码、补码、反码、移码
  - ② 整数的表示
  - ③ 浮点数的表示
  - ④ 十进制数的二进制编码表示
2. 非数值型数据的编码表示

## 1.3 二进制的算术运算

- 当两个二进制数表示两个数量大小时，它们之间可以进行数值运算，这种运算成为**算术运算**
  - 二进制算术运算和十进制算术运算的规则基本相同，唯一的区别在于“逢二进一”而不是“逢十进一”

### 加法运算

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

### 减法运算

$$\begin{array}{r} 1001 \\ - 0101 \\ \hline 0100 \end{array}$$

## 1.3 二进制的算术运算：加法

运算法则：

$$0+0=0$$

$$0+1=1$$

$$1+1=10 \text{ (产生了进位1)}$$

$$1+1+1=11 \text{ (产生了进位1)}$$

例1：1101和1011相加

1111	进位
1101	被加数
+ 1011	加数
<hr/>	
11000	和

结论：

两个二进制数相加时，每一位是被加数、加数和低位的进位三个数的相加



## 1.3 二进制的算术运算：加法

运算法则：

$$0+0=0$$

$$0+1=1$$

$$1+1=10 \text{ (产生了进位1)}$$

$$1+1+1=11 \text{ (产生了进位1)}$$

例2：10001111B和10110101B相加

10111111	进位
10110101	被加数
+ 10001111	加数
<hr/>	
101000100	和

结论：

两个二进制数相加时，每一位是被加数、加数和低位的进位三个数的相加

## 1.3 二进制的算术运算：减法

运算法则：

$$0-0=0$$

$$1-1=0$$

$$1-0=1$$

$$0-1=1 \text{ (产生了借位1)}$$

例1：11011B和1101B相减

	0	1	0	1	1	借位后的被减数
	1	1	0	1	1	被减数
—	0	1	1	0	1	减数
		1	1	1	0	差

结论：

两个二进制数相减时，每一位是带借位的被减数与减数的差

## 1.3 二进制的算术运算：减法

运算法则：

$$0-0=0$$

$$1-1=0$$

$$1-0=1$$

$$0-1=1 \text{ (产生了借位1)}$$

例2：11000100B和00100101B相减

	1	0	1	1	1	1	0	1	1	0
	1	1	0	0	0		1	0		0
—	0	0	1	0	0		1	0		1
	1	0	0	1	1	1	1			1

借位后的被减数

被减数

减数

差

结论：

两个二进制数相减时，每一位是带借位的被减数与减数的差

## 1.3 二进制的算术运算：乘法

运算法则：

$$0 \times 0 = 0$$

$$1 \times 1 = 1$$

$$1 \times 0 = 0$$

$$0 \times 1 = 0$$

乘法运算

$$\begin{array}{r} 1001 \\ \times 0101 \\ \hline 1000 \\ 0000 \\ 1001 \\ 0000 \\ \hline 0101101 \end{array}$$

## 1.3 二进制的算术运算：乘法

例1：1111B和11011B相乘

运算法则（边乘、边加的方法）：

$$0 \times 0 = 0$$

$$1 \times 1 = 1$$

$$1 \times 0 = 0$$

$$0 \times 1 = 0$$

1111	被乘数
× 1101	乘数
1111	第1次部分积
0000	
01111	第2次部分积
1111	
1001011	第3次部分积
1111	
11000011	第4次部分积

结论：

从乘数的低位开始，用乘数的每一位分别去乘被乘数，所得的各中间结果的最低有效位与相应的乘数位对齐，最后把这些中间结果同时相加即得到最后乘积。

## 1.3 二进制的算术运算：除法

例如：100011B除以101B

	0111	商
除数101)	100011	被除数
	101	
	011(1)	余数
	101	
	010(1)	余数
	101	
	0	余数

	0001.11...
010	1001
1	
	0101
	1000
	0101
	0110
	0101
	0010

结论：

应用减法规则可实现除法运算，从被除数最高位开始，找到足以减去除数的位数商1，再从被除数减去除数，依次除下去.....

## 一. 数制

1. 进位计数制
2. 二进制数与其他计数制数之间的转换
3. 二进制数的算术运算

## 二. 编码

1. 数值型数据的编码表示
  - ① 定点数的编码：原码、补码、反码、移码
  - ② 整数的表示
  - ③ 浮点数的表示
  - ④ 十进制数的二进制编码表示
2. 非数值型数据的编码表示

## 2.1 数值型数据的编码表示

- 表示一个数值型数据要确定三要素
  - 进位计数制、定点/浮点表示、编码规则
  - 任何给定的一个二进制0/1序列，在未确定它采用什么进位计数制、定点还是浮点以及编码方法之前，它所表示的值是无法确定的
- 日常生活中所使用的数有整数和实数之分
  - 整数的小数点固定在数的最右边，可以省略不写；而实数的小数点则不固定
  - 计算机内部数据中的每一位只能是0或1，不可能出现小数点
  - 必须要解决小数点的表示问题，计算机才能够处理日常使用的数值型数据



## 2.1 数值型数据的编码表示

- 在计算机中通常通过约定小数点的位置来实现
  - 小数点位置约定在固定位置的数称为定点数
  - 小数点位置约定为可浮动的数称为浮点数
  - 任意一个浮点数都可以用一个定点小数和一个顶点整数来表示，因此，只需要考虑定点数的编码表示
- 定点数的编码表示
  - 主要有四种：原码、补码、反码、移码

## 2.1.1 定点数的编码

- 计算机内部数据中的每一位只能是0或者1，所以正负号也用0和1来表示
  - 一般规定0表示正号，1表示负号
- 数字化了的符号能否和数值部分一起参加运算？
  - 为此，产生了把符号位和数值部分一起进行编码的各种方法
- 机器数和真值
  - 将数值型数据在计算机内部编码表示后的数称为**机器数**（一定是一个0/1序列）
  - 机器数真正的值（即现实世界中带正负号的数）称为机器数的**真值**

## 2.1.1 定点数的编码

- 机器数和真值
  - 将数值型数据在计算机内部编码表示后的数称为**机器数**（一定是一个0/1序列）
  - 机器数真正的值（即现实世界中带正负号的数）称为机器数的**真值**
- 例：机器数  $X$  有  $n$  位，表示为  $X = X_{n-1}X_{n-2} \cdots X_1X_0$ 
  - $X_i = 0$  或者  $1$ ，第一位  $X_{n-1}$  是数的符号，后  $n-1$  位是数值部分
  - 真值  $X_T = \pm X_{n-2} \cdots X_1X_0$
- 数值型数据在计算机内部的编码问题，实际上就是机器数  $X$  的各位  $X_i$  取值与真值  $X_T$  的关系问题
  - 常用编码表示法：**原码**表示法、**补码**表示法、**反码**表示法、**移码**表示法

## 2.1.1 定点数的编码：原码表示法

- 一个数的原码由符号位后直接跟数值位构成
  - 因此也称“符号-数制”（sign and magnitude）表示法
  - 正数和负数仅符号位不同，数值部分完全相同
- 原码编码规则
  - 机器数  $X$  有  $n$  位，表示为  $X = X_{n-1}X_{n-2} \cdots X_1X_0$
  - 当  $X_T$  为正数时， $X_{n-1} = 0$ ；当  $X_T$  为负数时， $X_{n-1} = 1$

## 2.1.1 定点数的编码：原码表示法

- 例：数-10（-1010B），用8位原码表示
  - 真值 $X_T = -0001010$ ，机器数 $X = 10001010B$ （8AH或0x8A）
- 例：数10（1010B），用8位原码表示
  - 真值 $X_T = +0001010$ ，机器数 $X = 00001010B$ （0AH或0x0A）
- 原码0有两种表示形式
  - $[+0]_{\text{原}} = 000 \cdots 0$
  - $[-0]_{\text{原}} = 100 \cdots 0$

## 2.1.1 定点数的编码：原码表示法

- 优点
  - 与真值的对应关系直观、方便
- 缺点
  - 0的表示不唯一，给使用带来不便
  - 原码运算中符号和数值部分必须分开处理

## 2.1.1 定点数的编码：补码表示法

- 补码表示可以实现加减运算的统一，即用加法来实现减法运算
  - 在计算机中，补码用来表示带符号整数
  - 也称“2-补码”（two's complement）表示法，由符号位后跟真值的模 $2^n$ 补码构成
- 模运算
  - 若 $A, B, M$ 满足 $A = B + K \times M$ （ $K$ 为整数），则记为 $A \equiv B \pmod{M}$
  - $A, B$ 各除以 $M$ 后的余数相同，故称 $B$ 和 $A$ 为模 $M$ 同余

## 2.1.1 定点数的编码：补码表示法

### ■ 模运算

- 若 $A, B, M$ 满足 $A = B + K \times M$  ( $K$ 为整数), 则记为 $A \equiv B \pmod{M}$
- $A, B$ 各除以 $M$ 后的余数相同, 故称 $B$ 和 $A$ 为 **模 $M$ 同余**

### ■ 例：钟表是一个典型的模运算系统

- 假定现在钟表时针指向10点, 要将它拨向6点, 有两种拨法
- 拨法1. 逆时针拨4格:  $10 - 4 = 6$
- 拨法2. 顺时针拨8格:  $10 + 8 = 18 \equiv 6 \pmod{12}$
- 在模12系统中,  $10 - 4 \equiv 10 + (12 - 4) \equiv 10 + 8 \pmod{12}$ , 即  $-4 \equiv 8 \pmod{12}$
- 我们称8是-4对模12的**补码**, 同理有  $-3 \equiv 9 \pmod{12}$ 和即  $-5 \equiv 7 \pmod{12}$ 等
- 对于某一确定的模, 某数 $A$ 减去小于模的另一数 $B$ , 可以用 $A$ 加上 $-B$ 的补码来代替, 这就是补码可以借助加法运算来实现减法运算的原因



## 2.1.1 定点数的编码：补码表示法

- 补码的表示
  - 正数的补码符号位0，数值部分是它本身
  - 负数的补码等于模与该负数的绝对值之差
- 公式表示
  - 当 $X_T$  为正数时， $[X_T]_{\text{补}} = X_T = M + X_T \pmod{M}$
  - 当 $X_T$  为负数时， $[X_T]_{\text{补}} = M - |X_T| = M + X_T \pmod{M}$
  - 统一：对于任意数 $X_T$ ，它的补码形式为 $[X_T]_{\text{补}} = M + X_T \pmod{M}$
- 计算机内部，具有一位符号位和 $n - 1$ 位数值位的二进制整数补码
  - $[X_T]_{\text{补}} = 2^n + X_T \pmod{2^n} \quad (-2^{n-1} \leq X_T < 2^{n-1})$

## 2.1.1 定点数的编码：特殊数据的补码表示

- 分别求出补码位数为 $n$ 和 $n + 1$ 时 $-2^{n-1}$ 的补码表示

- 当补码位数为 $n$ 时，其模为 $2^n$ ，因此：

$$[-2^{n-1}]_{\text{补}} = 2^n - 2^{n-1} = 2^{n-1} \pmod{2^n} = 10 \cdots 0 \text{ (} n - 1 \text{个} 0 \text{)}$$

- 当补码位数为 $n + 1$ 时，其模为 $2^{n+1}$ ，因此：

$$[-2^{n-1}]_{\text{补}} = 2^{n+1} - 2^{n-1} = 2^n + 2^{n-1} \pmod{2^{n+1}} = 110 \cdots 0 \text{ (} n - 1 \text{个} 0 \text{)}$$

- 同一个真值在不同维数的补码表示中，其对应的机器数不同

- 在给定编码表示时，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

## 2.1.1 定点数的编码：特殊数据的补码表示

- 补码的位数为 $n$ ，求 $-1$ 的补码表示
  - $[-1]_{\text{补}} = 2^n - 1 = 11 \cdots 1$  ( $n$ 个1)
- 补码的位数为 $n$ ，求 $2^{n-1}$ 的补码表示
  - $[2^{n-1}]_{\text{补}} = 2^n + 2^{n-1} \pmod{2^n} = 2^{n-1} = 10 \cdots 0$  ( $n-1$ 个0)
  - 最高位为1，说明对应的真值是负数，而这与实际情况不符
  - $n$ 位补码无法表示 $2^{n-1}$
  - $[X_T]_{\text{补}} = 2^n + X_T$  ( $-2^{n-1} \leq X_T < 2^{n-1}$ ,  $\pmod{2^n}$ )

## 2.1.1 定点数的编码：特殊数据的补码表示

- 补码的位数为 $n$ ，求0的补码表示
  - $[+0]_{\text{补}} = [-0]_{\text{补}} = 2^n \pm 0 \pmod{2^n} = 00 \cdots 0$  ( $n$ 个0)
  - 补码0的表示是唯一的
- “补码0的表示唯一”的优势
  - 减少了+0和-0之间的转换
  - 少占了一个编码表示，使补码比原码能多表示一个最小负数 $-2^{n-1}$

机器数	原码	补码
00 ... 0 :	+0	+0, -0
10 ... 0 :	-0	$-2^{n-1}$

## 2.1.1 定点数的编码：补码与真值之间的转换方法

- 原码与真值之间的对应关系简单
  - 只要转换符号，数值部分不变
- 对于补码，正数和负数的转换不同
  - 求一个正数的补码，将“+”号转换为0，数值部分不需要改变
  - 求一个负数的补码，需要做减法运算，因而不方便和直观

## 2.1.1 定点数的编码：补码与真值之间的转换方法

例：设补码的位数为8，求 110 1100 和 -110 1100 的补码表示

解：补码的位数为8，说明补码数值部分有 7 位，根据补码的定义可知

$$[110\ 1100]_{\text{补}} = 2^8 + 110\ 1100 \pmod{2^8} = 0110\ 1100$$

$$\begin{aligned} [-110\ 1100]_{\text{补}} &= 2^8 - 110\ 1100 = 1\ 0000\ 0000 - 110\ 1100 \\ &= 1000\ 0000 + 1000\ 0000 - 110\ 1100 \\ &= 1000\ 0000 + (111\ 1111 - 110\ 1100) + 1 \\ &= 1000\ 0000 + 001\ 0011 + 1 \pmod{2^8} \\ &= 1001\ 0100 \end{aligned}$$

## 2.1.1 定点数的编码：补码与真值之间的转换方法

$$[110\ 1100]_{\text{补}} = 2^8 + 110\ 1100 \pmod{2^8} = 0110\ 1100$$

$$\begin{aligned} [-110\ 1100]_{\text{补}} &= 2^8 - 110\ 1100 = 1\ 0000\ 0000 - 110\ 1100 \\ &= 1000\ 0000 + 1000\ 0000 - 110\ 1100 \\ &= 1000\ 0000 + (111\ 1111 - 110\ 1100) + 1 \\ &= 1000\ 0000 + 001\ 0011 + 1 \pmod{2^8} \\ &= 1001\ 0100 \end{aligned}$$

- 负数补码计算：符号位为1，数值部分“各位取反，末位加1”
  - 负数的补码计算过程中第一个 1000 0000 用于产生最后的符号 1
  - 第二个 1000 0000 拆分为 111 1111+1，而 (111 1111-110 1100) 实际是将数值部分 110 1100 各位取反
- 正数补码计算：符号位取0，其余同真值中相应各位

## 2.1.1 定点数的编码：补码与真值之间的转换方法

- 负数，由补码求真值

- 由真值求负数补码：数值部分“各位取反，末位加1”
- 逆推，由补码求真值：补码数值部分先减1然后再取反
- $11 \dots 1 - (X - 1)$ ，等价于  $(11 \dots 1 - X) + 1$ ，即：“各位取反，末位加1”

- 由补码求真值的简便方法

- 若符号位为0，则真值的符号为正，其数值部分不变
- 若符号位为1，则真值的符号位负，其数值部分的各位由补码“各位取反，末位加1”



## 2.1.1 定点数的编码：补码与真值之间的转换方法

- 由补码求真值的简便方法
  - 若符号位为0，则真值的符号为正，其数值部分不变
  - 若符号位为1，则真值的符号位负，其数值部分的各位由补码“各位取反，末位加1”

例：已知 $[X_T]_{\text{补}} = 1011\ 0100$ ，求真值 $X_T$

解： $X_T = -(100\ 1011 + 1) = -100\ 1100 = -76$

## 2.1.1 定点数的编码：反码表示法

- 负数的补码可采用“各位取反，末位加1”的方法得到
  - 仅各位取反而末位不加1，那么就得到负数的反码表示
  - 反码的定义就是在相应的补码表示中再末位减1
- 反码特点
  - 0的表示不唯一
  - 表数范围比补码少一个最小负数
  - 运算时必须考虑循环进位
  - 因此，反码在计算机中很少被使用，有时作为数码变换的中间表示形式或者用于数据校验

## 2.1.1 定点数的编码：移码表示法

- 浮点数实际上是用两个定点数来表示的
  - 用定点小数表示浮点数的尾数，用定点整数表示浮点数的阶（即指数）
  - 一般情况下，浮点数的阶用一种称为“移码”的编码方式表示
  - 通常，将阶的编码称为阶码
- 为什么用移码表示阶？
  - 阶可以是正数，也可以是负数，当进行浮点数的加减运算时，必须先“对阶”（即比较两个数的阶的大小并使之相等）

## 2.1.1 定点数的编码：移码表示法

### ■ 对阶的简化操作

- 为简化比较操作，使操作过程不涉及阶的符号，可以对每个阶都加上一个正的常数，称为**偏置常数**（bias），使所有阶都转换为正整数
- 这样，在对浮点数的阶进行比较时，就是对两个正整数进行比较，因此可以直观地将两个数按位从左到右进行比对，从而**简化对阶操作**

### ■ 移码表示

- 假设用来表示阶  $E$  的移码的位数为  $n$ ，则  $[E]_{\text{移}} = \text{偏置常数} + E$
- 通常，偏置常数取  $2^{n-1}$  或  $2^{n-1} - 1$

## 2.1.2 整数的表示

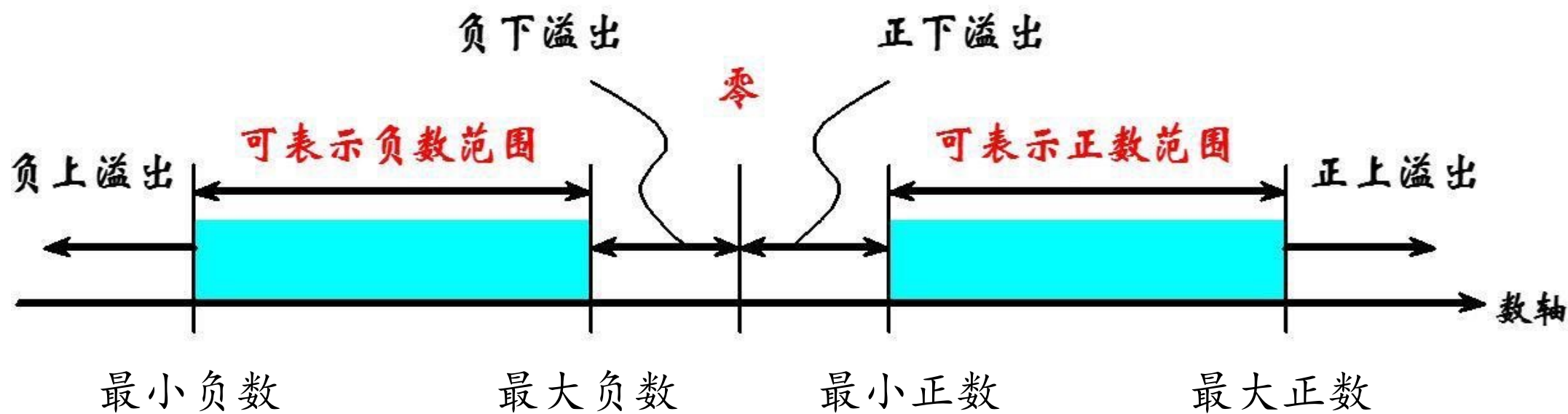
- 整数的小数点隐含在数的最右边，无须表示小数点，也称“定点数”
  - 计算机中的整数分为无符号整数（unsigned integer）和带符号整数（signed integer）
- 无符号整数，简称无符号数
  - 一个编码的所有二进位都用来表示数值而没有符号位，非负整数
  - 用在不出现负值的场合，如地址运算，或用来表示指针、下标等
  - 省略了一位符号位， $n$  位能表示的数的范围为  $0 \sim 2^n - 1$
- 带符号整数
  - 必须用一个二进位表示符号
  - 原码、补码、反码、移码都可以用来表示
  - 补码表示有其突出的优点，现代计算机中带符号正数都用补码表示

## 2.1.3 浮点数的表示

- 计算机中专门用浮点数来表示实数
  - 用定点数表示数值型数据时，其表示的范围很小，运算结果很容易溢出
  - 用定点数也无法表示带有小数点的实数
- 浮点数的表示范围
  - 任一浮点数可用两个定点数表示，一个定点小数表示浮点数的尾数，一个定点整数表示浮点数的阶
  - 阶的编码称为阶码，为便于对阶，阶码通常采用移码形式

## 2.1.3 浮点数的表示

- 浮点数的表示范围
  - 在可表示范围内的数值并不是连续的
  - 根据浮点数的表示格式，只要尾数为0，不管阶码是什么，其值都是0，这样的数被称为机器零；因此，机器零的表示不唯一
  - 通常用阶码和尾数同时为0来唯一表示机器零；即当结果出现尾数为0时，不管阶码是什么，都将阶码取为0，机器零有+0和-0之分



## 2.1.3 浮点数的表示

- 浮点数的规格化
  - 在浮点数运算中，尽可能多地保留有效数字的位数，使有效数字尽量占满尾数数位
  - 规格化操作可以使浮点数的表示具有唯一性
  - 规格化数的标志是 真值的尾数部分中最高位具有非零数字，“左规”和“右规”
- 右规
  - 当尾数的有效数位进到最高位前面时，需要进行右规
  - 右规时，尾数每右规一位，阶码加一，直到尾数变为规格化形式为止
  - 右规时阶码会增加，因此阶码有可能上溢
- 左规
  - 当尾数出现形如  $\pm 0.0 \dots 0bb \dots b$  的运算结果时，需要进行左规
  - 左规时，尾数每左移一位，阶码减1，直到尾数变为规格化形式为止
  - 左规时阶码会减小，因此阶码有可能下溢



## 2.1.3 浮点数的表示

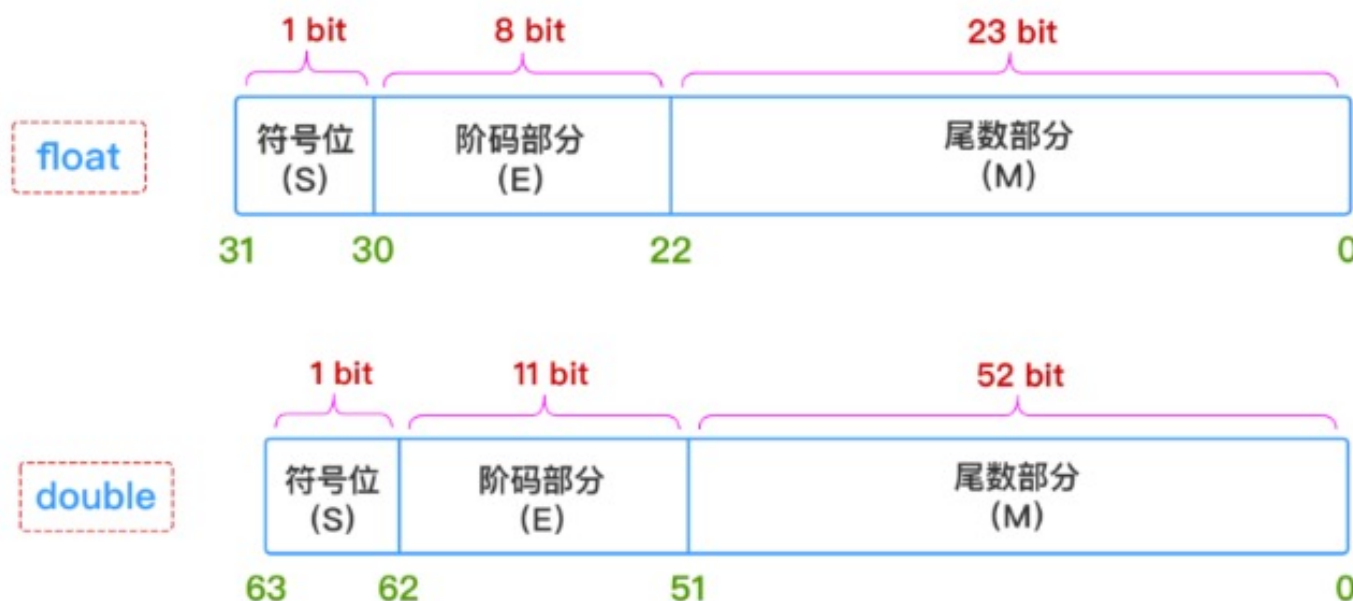
- IEEE 754 浮点数标准（目前几乎所有计算机都采用的标准）
  - 提供了两种基本格式：32位单精度和64位双精度格式



## 2.1.3 浮点数的表示

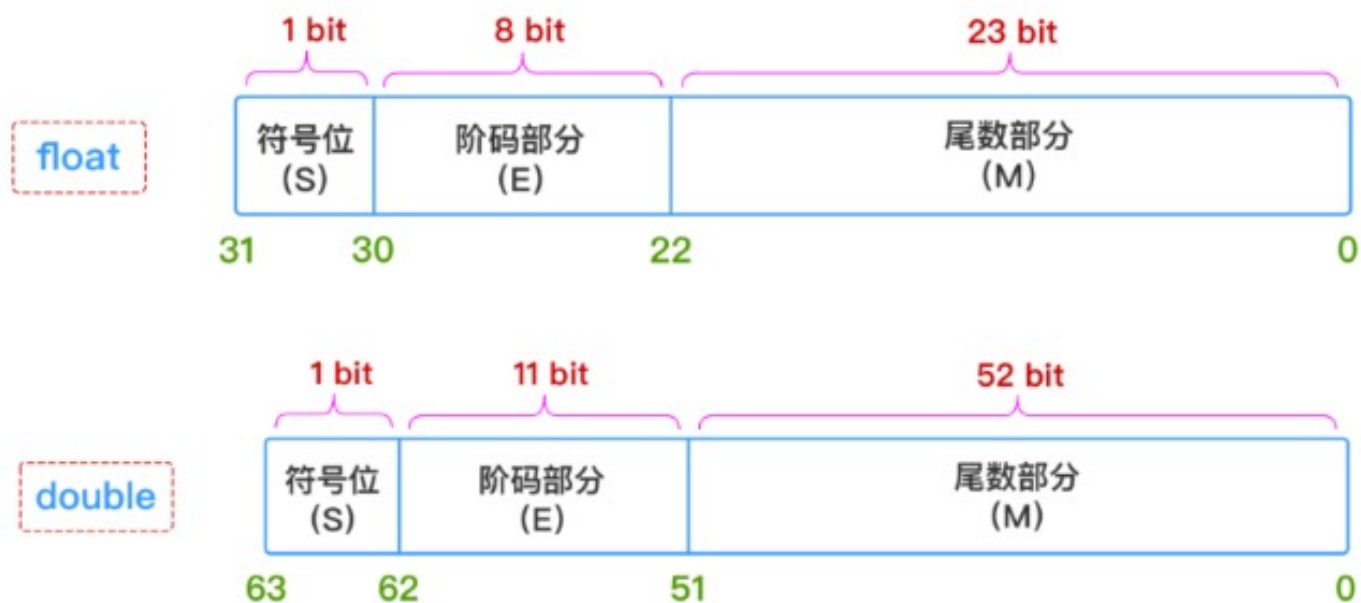
### ■ 尾数部分

- 尾数用原码表示，第一位总为1，因而可在尾数中缺省第一位的1，成为隐藏位
- 这使得单精度格式的23位尾数实际上表示了24位有效数字，双精度格式的尾数实际上表示了53位有效数字
- IEEE 754 规定，小数点前面的“1”是隐藏位



## 2.1.3 浮点数的表示

- 阶码部分
  - 阶码用移码形式，偏置常数是  $2^{n-1} - 1$
  - 单精度和双精度浮点数的偏置常数分别为127和1023





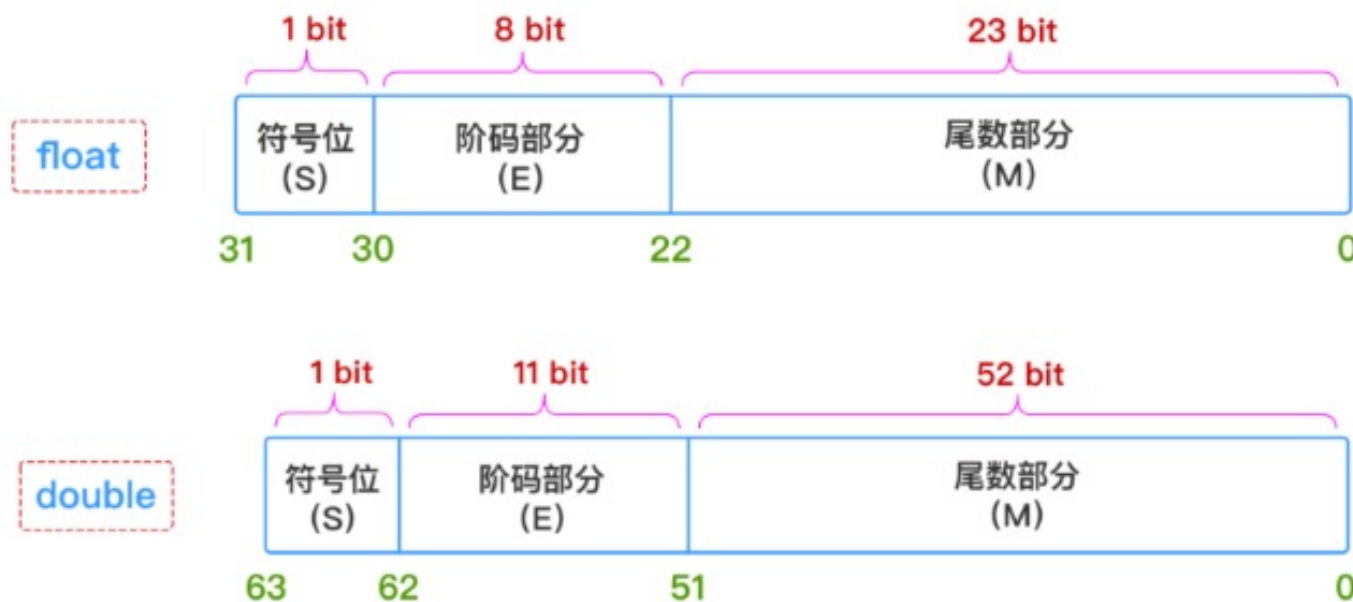
## 2.1.3 浮点数的表示

表 1.2 IEEE 754 浮点数的解释

值的类型	单精度 (32 位)				双精度 (64 位)			
	符号	阶码	尾数	值	符号	阶码	尾数	值
正零	0	0	0	0	0	0	0	0
负零	1	0	0	-0	1	0	0	-0
正无穷大	0	255 (全 1)	0	$\infty$	0	2047 (全 1)	0	$\infty$
负无穷大	1	255 (全 1)	0	$-\infty$	1	2047 (全 1)	0	$-\infty$
无定义数 (非数)	0 或 1	255 (全 1)	$\neq 0$	NaN	0 或 1	2047 (全 1)	$\neq 0$	NaN
规格化非零正数	0	$0 < e < 255$	$f$	$2^{e-127} (1.f)$	0	$0 < e < 2047$	$f$	$2^{e-1023} (1.f)$
规格化非零负数	1	$0 < e < 255$	$f$	$-2^{e-127} (1.f)$	1	$0 < e < 2047$	$f$	$-2^{e-1023} (1.f)$
非规格化正数	0	0	$f \neq 0$	$2^{-126} (0.f)$	0	0	$f \neq 0$	$2^{-1022} (0.f)$
非规格化负数	1	0	$f \neq 0$	$-2^{-126} (0.f)$	1	0	$f \neq 0$	$-2^{-1022} (0.f)$

## 2.1.3 浮点数的表示

- 正常的规格化非0数：阶码非全0且非全1
  - 阶码范围：1~254（单精度）和 1~2046（双精度）
  - 阶的范围：-126~+127（单精度）和 -1022~+1023（双精度）
  - 计算公式： $(-1)^s \times 1.m \times 2^{e-127}$  和  $(-1)^s \times 1.m \times 2^{e-1023}$



## 2.1.3 浮点数的表示

- 正常的规格化非0数：阶码非全0且非全1
  - 计算公式： $(-1)^s \times 1.m \times 2^{e-127}$  和  $(-1)^s \times 1.m \times 2^{e-1023}$

例：将十进制数  $-0.75$  转换为 IEEE 754 的单精度浮点数格式表示

解： $(-0.75)_{10} = (-0.11)_2 = (-1.1)_2 \times 2^{-1} = (-1)^s \times 1.m \times 2^{e-127}$

所以  $s = 1, m = 0.100 \dots 0, e = (127 - 1)_{10} = (126)_{10} = (0111 \ 1110)_2$

表示为单精度格式的浮点数为 1 0111 1110 1000 0000...0000 000

用十六进制表示为 BF40 0000H

## 2.1.3 浮点数的表示

- 正常的规格化非0数：阶码非全0且非全1
  - 计算公式： $(-1)^s \times 1.m \times 2^{e-127}$  和  $(-1)^s \times 1.m \times 2^{e-1023}$

例：IEEE 754 单精度表示 C0A0 0000H 的真值是多少？

解：C0A0 0000H = 1 1000 0001 010 0000 ... 0000

所以  $s = 1, m = (0.01)_2 = (0.25)_{10}, e = (1000\ 0001)_2 = (129)_{10}$

真值为  $(-1)^s \times 1.m \times 2^{e-127} = (-1)^1 \times 1.25 \times 2^{129-127} = -1.25 \times 2^2 = -5$



## 2.1.4 十进制数的二进制编码表示

- 用四位二进制代码来表示一位十进制数码，这样的代码称为“二 - 十进制码” (Binary Coded Decimal codes, 又称BCD码)
  - 人们日常使用和熟悉的是十进制数，计算机外部看到的数据基本上是十进制形式
  - 用ASCII字符串方式表示十进制数，可方便地进行十进制的输入/输出；但是，因为这种表示形式中含有非数值信息（高4位编码），所以对十进制的运算很不方便
  - 计算机内部处理二进制信息，须将十进制转换为二进制或用BCD码表示十进制数
- 存在多种BCD码方案
  - 四位二进制有16种状态，从中选取10中状态来表示0~9的方法很多



## 2.1.4 十进制数的二进制编码表示

- 有权BCD码：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
  - 最常用的有权码是8421码，每位的权从左到右分别是8, 4, 2, 1
  - 它选取4位二进制数按计数顺序的前10个代码与十进制数字相对应
  - 也称为 自然BCD码，NBCD码
- 其他有权BCD码
  - 5421码，1011代表 $5+0+2+1=8$
  - 2421码，1011代表 $2+0+2+1=5$
  - 5421码和2421码不唯一，例如2421码中1100和0110都可以表示6

## 2.1.4 十进制数的二进制编码表示

### ■ 5421BCD码

- 前5个码和8421码相同，后5个码在前5个码的基础上加1000构成
- 这样的码，前5个码和后5个码一一对应，仅高位不同

### ■ 2421BCD码

- 前5个码和8421码相同，后5个码以中心对称取反
  - 这样的码称为自反代码
- |   |   |      |   |   |      |
|---|---|------|---|---|------|
| 4 | → | 0100 | 5 | → | 1011 |
| 0 | → | 0000 | 9 | → | 1111 |

## 2.1.4 十进制数的二进制编码表示

- 无权BCD码：十进制数位的4个二进制位（称为基2码）没有确定的权
  - 常用的有应余3码和格雷码
- 余3码
  - 由8421码加上0011形成的一种无权码，例如6的余3码为： $0110+0011=1001$
  - 特点1：当两个十进制数的和是10时，相应的二进制编码正好是16
  - 特点2：0和9，1和8，...，5和4的余3码互为反码
- 格雷码
  - 任意两个相邻的编码只有一位二进制位不同，这种特点叫做“循环码”
  - 格雷码有多种编码形式

常用BCD码

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

## 2.1.4 十进制数的二进制编码表示

十进制	8421BCD码	十进制	8421BCD码
0	0000	8	1000
1	0001	9	1001
2	0010	10	0001 0000
3	0011	11	0001 0001
4	0100	12	0001 0010
5	0101	13	0001 0011
6	0110	14	0001 0100
7	0111	15	0001 0101

- BCD码既有二进制码的形式（四维二进制码），又有十进制数的特点（每四位二进制数就是一位十进制数）
  - 二进制与BCD码之间的转换需经过十进制

# 十进制数与8421码的相互转换

- 十进制数256，BCD码为
  - $256\text{ D} = 0010\ 0101\ 0110\text{ BCD}$
- 十进制数0.764，BCD码为
  - $0.764\text{ D} = 0.0111\ 0110\ 0100\text{ BCD}$
- BCD码转为十进制数
  - $0110\ 0010\ 1000 . 1001\ 0101\ 0100\text{ BCD} = 628.954\text{ D}$

# 二进制数转为8421码

- 二进制数1011.01，BCD码为

$$\begin{aligned} 1011.01B &= (1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2}) D \\ &= 11.25 D \\ &= 0001\ 0001 . 0010\ 0101\ BCD \end{aligned}$$

# 计算机内部的BCD码

- 十进制数中每个数字对应4个二进制位，两个数字占一个字节
- 符号可用1位二进制数表示（1表示负数，0表示正数），或用4位二进制数表示并放在字符串最后
- Pentium处理器中的十进制数占80位，第一个字节中的最高位为符号位，后面的9个字节表示18位十进制数



## 一. 数制

1. 进位计数制
2. 二进制数与其他计数制数之间的转换
3. 二进制数的算术运算

## 二. 编码

1. 数值型数据的编码表示
  - ① 定点数的编码：原码、补码、反码、移码
  - ② 整数的表示
  - ③ 浮点数的表示
  - ④ 十进制数的二进制编码表示
2. 非数值型数据的编码表示

## 2.2 非数值型数据的编码表示

- 逻辑值、字符等数据是非数值型数据，在机器内部用一个二进制位串表示
- 逻辑值的表示
  - $n$ 位二进制可表示 $n$ 个逻辑值
  - 逻辑数据只能参加逻辑运算，并且是按位进行的（按位“与”、按位“或”、逻辑左移、逻辑右移等）
  - 逻辑数据和数值型数据都是一串0/1序列，在形式上无任何差异，需要通过指令的操作码类型来识别它们
  - 逻辑运算指令处理的是逻辑数据，算术运算指令处理的是数值型数据

# 西文字符的表示

- 计算机内部处理的西文字符有拉丁字母、数字、标点符号及一些特殊符号
  - 所有字符的集合构成字符集
  - 字符集中每一个字符都有一个代码（二进制编码的0/1序列），构成了该字符集的代码表，简称码表
  - 码表中的代码具有唯一性
  - 字符主要用于外部设备和计算机之间交换信息，一旦确定了所使用的的字符集和编码方法后，计算机内部表示的字符代码和外部设备输入、打印和显示的字符之间就有唯一的对应关系

# 西文字符的表示

- 目前计算机中使用最广泛的西文字符集及其编码是**ASCII码**
  - 美国标准信息交换码 (American National Standard Code for Information Interchange)

**7位ASCII码表示 $2^7 = 128$ 种不同的字符，包括：**

可显示字符 (94个)

阿拉伯数字 (10个): 0~9

英文大小写字母 (52个): A~Z, a~z

西文符号 (32个): !, <, { 等

控制符 (34个)

NUL (空白), CR (控制)等

**扩展版ASCII码，8位，256个不同字符**

ASCII表

高四位  低四位		ASCII 码控制字符												ASCII 码打印字符												
		0000						0001						0010		0011		01/0		0101		0110		0111		
		0						1						2		3		4		5		6		7		
十进制	字符	Ctrl	代码	转义 字符	字符解释	十进制	字符	Ctrl	代码	转义 字符	字符解释	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	十进制	字符	Ctrl
0000	0	0		^@	NUL	\0	空字符	16	►	^P	DLE	数据链路转义	32		48	0	64	@	80	P	96	`	112	p		
0001	1	1	☉	^A	SOH		标题开始	17	◄	^Q	DC1	设备控制1	33	!	49	1	65	A	81	Q	97	a	113	q		
0010	2	2	⦿	^B	STX		正文开始	18	↕	^R	DC2	设备控制2	34	"	50	2	66	B	82	R	98	b	114	r		
0011	3	3	♥	^C	ETX		正文结束	19	!!	^S	DC3	设备控制3	35	#	51	3	67	C	83	S	99	c	115	s		
0100	4	4	♦	^D	EOT		传输结束	20	¶	^T	DC4	设备控制4	36	\$	52	4	68	D	84	T	100	d	116	t		
0101	5	5	♣	^E	ENQ		查询	21	§	^U	NAK	否定应答	37	%	53	5	69	E	85	U	101	e	117	u		
0110	6	6	♠	^F	ACK		肯定应答	22	—	^V	SYN	同步空闲	38	&	54	6	70	F	86	V	102	f	118	v		
0111	7	7	●	^G	BEL	\a	响铃	23	↑↓	^W	ETB	传输块结束	39		55	7	71	G	87	W	103	g	119	w		
1000	8	8	▣	^H	BS	\b	退格	24	↑	^X	CAN	取消	40	(	56	8	72	H	88	X	104	h	120	x		
1001	9	9	○	^I	HT	\t	横向指标	25	↓	^Y	EM	介质结束	41	)	57	9	73	I	89	Y	105	i	121	y		
1010	A	10	▣	^J	LF	\n	换行	26	→	^Z	SUB	替代	42	*	58	:	74	J	90	Z	106	j	122	z		
1011	B	11	⦿	^K	VT	\v	纵向制表	27	←	^[	BSC	\c	溢出	43	+	59	;	75	K	91	[	107	k	123	{	
1100	C	12	♀	^L	FF	\f	换页	28	L	^\	FS	文件分隔符	44	,	60	<	76	L	92	\	108	l	124			
1101	D	13	♪	^M	CR	\r	回车	29	↔	^]	GS	组分分隔符	45	-	61	=	77	M	93	]	109	m	125	}		
1110	E	14	♫	^N	SOH		移出	30	▲	^^	RS	记录分隔符	46	.	62	>	78	N	94	^	110	n	126	~		
1111	F	15	☼	^O	SI		移入	31	▼	^~	US	单元分隔符	47	/	63	?	79	O	95	_	111	o	127	△	^Backspace 代码:DEL	

注：表中的ASCII字符可以用“Alt + 小键盘上的数字键”方法输入

- 字符0~9这10个数字字符的高三位编码为011，低4位分别为0000~1001，正好是0~9这10个数字的二进制表示
- 英文字母的编码值也满足正常的字母排序关系，且大小写字母编码之间的差别仅在第5位上，使得大、小写之间的转换非常方便

## 1. 数制：计数方法或计数体制（由基数和位权组成）

种 类	基 数	位 权	应 用	备 注
十进制	0 ~ 9	$10^i$	日常	
二进制	0 , 1	$2^i$	数字电路	$2 = 2^1$
八进制	0 ~ 7	$8^i$	计算机程序	$8 = 2^3$
十六进制	0 ~ 9, A ~ F	$16^i$	计算机程序	$16 = 2^4$

各种数制之间的相互转换，特别是十进制→二进制的转换，要求熟练掌握。

2. 码制：常用的 BCD 码有 8421 码、2421 码、5421 码、余 3 码等，其中以 8421 码使用最广泛。

# 第一章 小结

- 了解数字电路的基本概念
- 掌握数制概念和计算机中二进制数特点
  - 二进制数与其他计数制数之间的转换规则
- 掌握编码概念和计算机中对数据的编码表示
  - 定点数的补码编码方式
  - 浮点数的编码表示
  - 十进制数的二进制编码表示

教材：《数字电子技术基础简明教程（第四版）》

- 习题：1.1, 1.2, 1.3



THE END