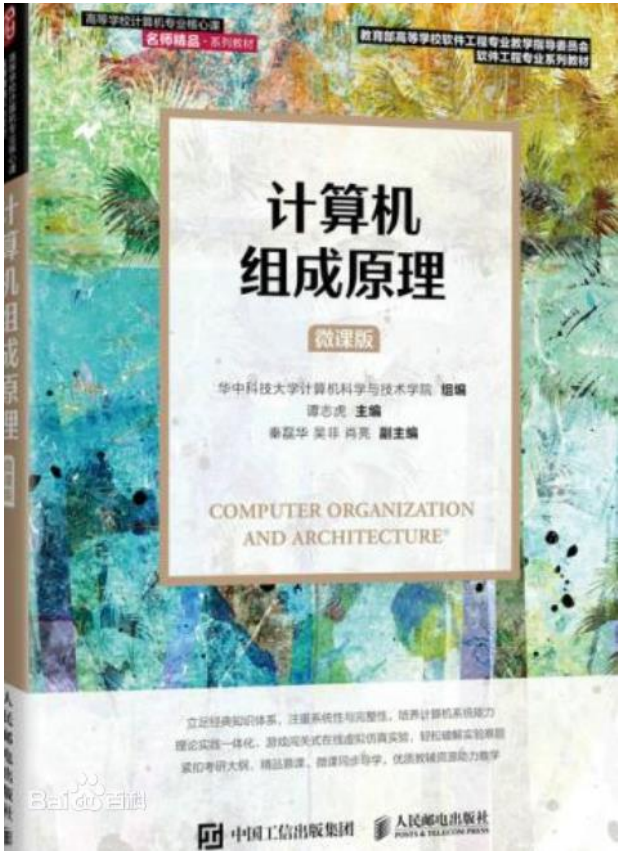
《计算机组成原理》

（第二讲）

厦门大学信息学院软件工程系 曾文华

2023年3月2日

目录



第2章　数据信息的表示

- 2.1　数据表示的作用
- 2.2　数值数据的表示
- 2.3　非数值数据的表示
- 2.4　数据信息的校验

2.1 数据表示的作用

- 在设计和选择计算机内的数据表示方式时，一般需要综合考虑以下几方面的因素：

① 数据的类型：数值数据（小数、整数、实数（浮点数）等）、非数值数据（ASCII码、汉字等）。

② 表示的范围和精度：通过选择适当的数据类型和字长来实现。

③ 存储和处理的代价：数据格式易于表示、存储和处理。

④ 软件的可移植性：数据格式符合相应的规范，方便软件在不同计算机之间的移植。

2.2 数值数据的表示

- **2.2.1 数的机器码表示**
  - 真值（二进制数）：
    - **+1011**（十进制：**+11**）
    - **-1100**（十进制：**-12**）
    - **+0.1011**（十进制：**+0.6875**）
    - **-0.1100**（十进制：**-0.75**）
  - 机器数（机器码）：
    ① 原码
    ② 反码
    ③ 补码
    ④ 移码

– 1、原码

- 正小数：**x=+0.1101，[x]$_{原}$=0.1101**
- 正整数：**x=+1101，[x]$_{原}$=0,1101**

- 负小数：**x=-0.1111，[x]$_{原}$=1.1111**
- 负整数：**x=-1111，[x]$_{原}$=1,1111**

- **0**的原码（小数）：**[+0]$_{原}$=0.0000，[-0]$_{原}$=1.0000**
- **0**的原码（整数）：**[+0]$_{原}$=0,0000，[-0]$_{原}$=1,0000**

- 正数的原码数值位为本身，符号位为**0**；负数的原码数值位为本身，符号位为**1**。

- **+0**的原码和**-0**的原码是不一样的。

– 2、反码

- 正小数：x=+0.1101，$[x]_{反}$=0.1101
- 正整数：x=+1101，$[x]_{反}$=0,1101

- 负小数：x=-0.1111，$[x]_{反}$=1.0000
- 负整数：x=-1111，$[x]_{反}$=1,0000

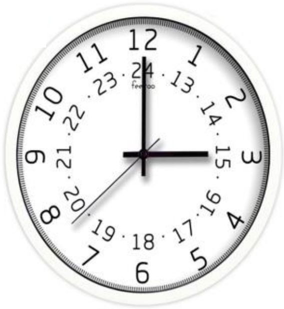
- 0的反码（小数）：$[+0]_{反}$=0.0000，$[-0]_{反}$=1.1111
- 0的反码（整数）：$[+0]_{反}$=0,0000，$[-0]_{反}$=1,1111

- 正数的反码数值位为本身，符号位为0；负数的反码数值位为本身取反，符号位为1。

- +0的反码和-0的反码是不一样的。

– 3、补码

- （1）模的概念



  – 时钟：模（模数）为12

  – 15点，即下午3点，因此有：15 ≡ 12+3 ≡ 3（mod 12）

  – 假设时钟的刻度为11点，为了将时钟调到3点，有二种方法：沿着顺时针拨4小时，或者沿着逆时针拨8小时。

  – 即：11 + 4 ≡ 3（mod 12），11 - 8 ≡ 3 （mod 12），或者：-8 ≡ 12-8 ≡ +4（mod 12）

  – -8与+4对模12是互补的，或者说以12为模时，-8的补码是+4

  – 以12为模时，-2的补码是+10；以12为模时，-5的补码是+7

- （2）补码的定义

  - **例2.1：求补码**
    - » 小数（正数）：**x=+0.0101，$[x]_{补}$=0.0101**
    - » 小数（负数）：**x=-0.0101，$[x]_{补}$=2+x=2-0.0101=1.1011**
    - » 小数（负数）：**x=-0.0000，$[x]_{补}$=2+x=2-0.0000=0.0000**
    - » 小数（负数）：**x=-1.0000，$[x]_{补}$=2+x=2-1.0000=1.0000**

  - **例2.2：设某计算机的字长为8位，分别求真值x=$(-10101)_2$，真值x=-128的补码。**
    - » 整数（负数）：**x=$(-10101)_2$，$[x]_{补}$=1 0000 0000+x=1 0000 0000-10101=1110 1011**
    - » 整数（负数）：**x=-128，$[x]_{补}$=256+x=256-128=128=1000 0000**

  - **例2.3：求补码**
    - » 整数（负数）：**x=-1000，$[x]_{补}$=1 0000+x=1 0000-1000=1,1000**
    - » 整数（负数）：**x=-0001，$[x]_{补}$=1 0000+x=1 0000-0001=1,1111**
    - » 小数（负数）：**x=-0.0001，$[x]_{补}$=2+x=2-0.0001=1.1111**

  - **例2.4：根据补码求真值**
    - » 整数（负数）：**$[x]_{补}$=1,1000，x=1 0000-$[x]_{补}$=1 0000-1 1000=-1000**
    - » 整数（负数）：**$[x]_{补}$=1,1111，x=1 0000-$[x]_{补}$=1 0000-1 1111=-0001**

  - **0的补码（小数）：$[+0]_{补}$=0.0000，$[-0]_{补}$=0.0000**
  - **0的补码（整数）：$[+0]_{补}$=0,0000，$[-0]_{补}$=0,0000**

  - 正数的补码数值位为本身，符号位为0；负数的补码数值位为本身取反加1，符号位为1。

  - **+0的反码和-0的补码是一样的。**

– 负数的真值求补码时，数值位为本身取反加1

例2.1

- x=-0.0101，数值位=0101，取反加1=1010+1=1011，$[x]_{补}$=1.1011
- x=-0.0000，数值位=0000，取反加1=1111+1=0000，$[x]_{补}$=0.0000（-0的补码与+0的补码是一样的）
- x=-1.0000，数值位=0000，取反加1=1111+1=0000，$[x]_{补}$=1.0000

例2.2

- x=$(-10101)_2$，数值位=10101，取反加1=01010+1=01011，$[x]_{补}$=1110 1011
- x=-128，数值位=128=1000 0000，取反加1=0111 1111 +1=1000 0000，$[x]_{补}$=1000 0000

例2.3

- x=-1000，数值位=1000，取反加1=0111+1=1000，$[x]_{补}$=1,1000
- x=-0001，数值位=0001，取反加1=1110+1=1111，$[x]_{补}$=1,1111
- x=-0.0001，数值位=0001，取反加1=1110+1=1111，$[x]_{补}$=1.1111

– 负数的补码求真值时，数值位也是本身取反加1

例2.4

- $[x]_{补}$=1,1000，数值位=1000，取反加1=0111+1=1000，x=-1000
- $[x]_{补}$=1,1111，数值位=1111，取反加1=0000+1=0001，x=-0001

- （3）变形补码（双符号补码，模4补码）
  - 例2.5：
    - 小数：x=-0.0101，$[x]_{补}$=4+x=4-0.0101=11.1011
    - （取反加1：0101取反=1010，加1=1011，$[x]_{补}$=11.1011）
  - 设某计算机的字长为8位，求真值x=-10101的变形补码：
    - 整数：x=-10101，$[x]_{补}$=1 0000 0000+x=1 0000 0000-10101=1110 1011
    - （取反加1：10101取反=01010，加1=01011， $[x]_{补}$=1110 1011）
  - 采用变形补码时，如果运算结果的符号位为00或11，表示运算结果没有溢出；如果运算结果的符号位为01或10时，表示有溢出，其中01表示正溢出，10表示负溢出。
  - 例如：
    - 9位二进制整数，双符号位补码的表示范围是：-128～+127，即11,000 0000 ～00,111 1111
    - 27+45=00,001 1011+00,010 1101=00,100 1000=72，没有溢出
    - 27-45=27+(-45)=00,001 1011+11,101 0011=11,110 1110=-18，没有溢出
    - 80+90=00,101 0000+00,101 1010=01,010 1010（80+90=170），正溢出
    - -80-90=-80+(-90)=11,011 0000+11,010 0110=10,101 0110（-80-90=-170），负溢出

– 4、移码

- 例2.6：
  - x=+1010110，$[x]_{补}$=0,1010110， $[x]_{移}$=1,1010110
  - x=-1010110，$[x]_{补}$=1,0101010， $[x]_{移}$=1,0101010

- 0的移码（整数）：$[+0]_{移}$=1,0000，$[-0]_{移}$=1,0000；+0的移码和-0的移码是一样的。

- 整数：移码为补码的符号位取反，移码的符号位为0时表示负数，移码的符号位为1时表示正数。小数：没有移码。

- 移码通常用于表示浮点数的阶码，浮点数的阶码用移码表示时具有以下的优点：

  ① 可以直接用无符号数规则比较两个浮点数阶码的大小。
  - » 例如，字长为8位时，-6与+5的补码分别是1111 1010和0000 0101，如果看成是无符号数，则1111 1010大于0000 0101，得到错误的结果。
  - » 如果采用移码， -6与+5的移码分别是0111 1010和1000 0101，显然无符号数1000 0101大于0111 1010，得到正确的结果。

  ② 有利于“浮点机器0”的判断，当移码的各位均为0，对应的阶码最小，此时，当尾数为全0，对应的就是“浮点机器0”。
  - » 字长为8位时，移码=0000 0000，对应的补码=1000 0000，真值=-128，为最小的阶码。

表2.3　真值的补码和移码表示对照表（见教材）

真值（十进制）	二进制表示	补码表示（8位）	移码表示（8位）
-128	-1000 0000	1,000 0000	0,000 0000
-127	-0111 1111	1,000 0001	0,000 0001
……	………………	………………	………………
-1	-0000 0001	1,111 1111	0,111 1111
0	0000 0000	0,000 0000	1,000 0000
1	0000 0001	0,000 0001	1,000 0001
……	………………	………………	………………
126	0111 1110	0,111 1110	1,111 1110
127	0111 1111	0,111 1111	1,111 1111

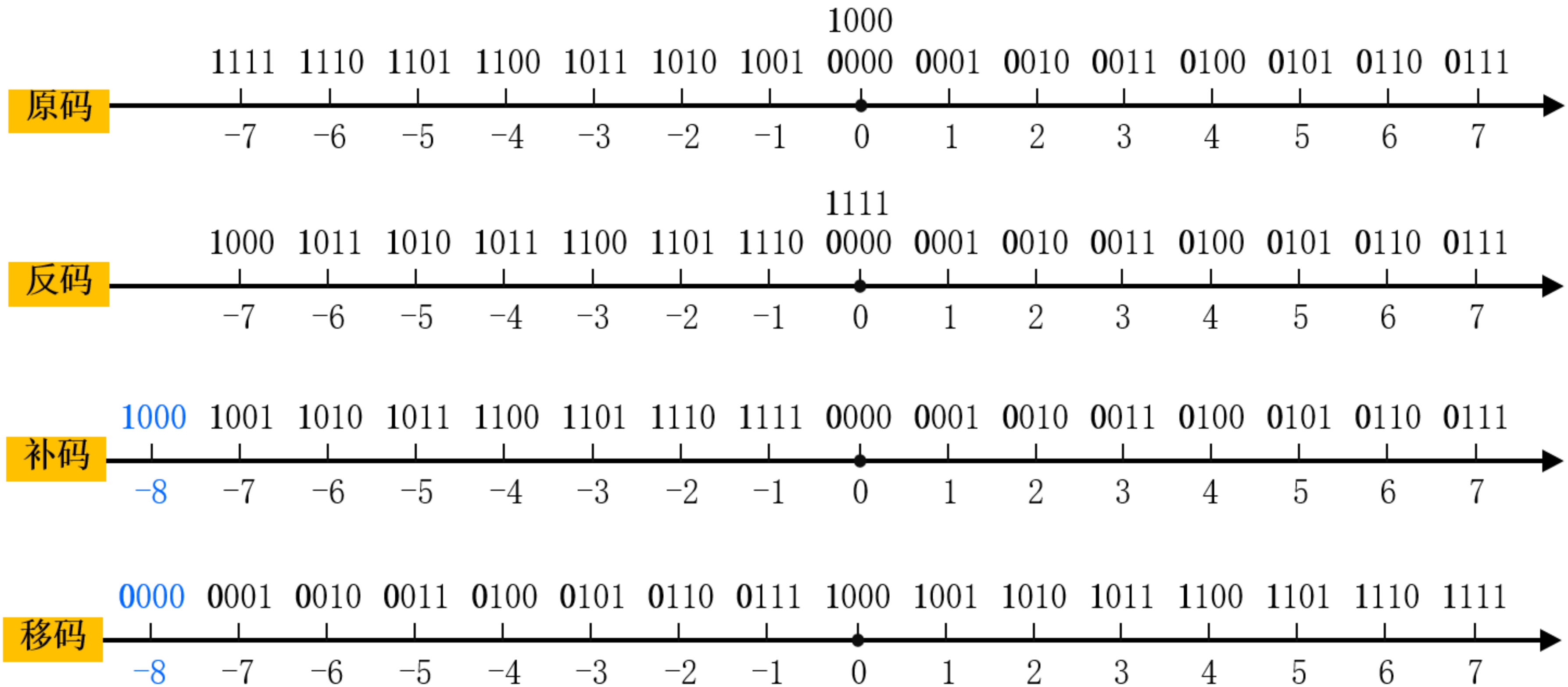


图2.2　4位不同机器码在数轴上的表示（见教材）

4位二进制数整数

原码表示范围：**-7～+7**；反码表示范围：**-7～+7**

补码表示范围：**-8～+7**；移码表示范围：**-8～+7**

• 2.2.2 定点数表示

- 1、定点小数（纯小数）
  - $x=x_0.x_1x_2…x_n$
  - 例如：**x=0.1101**（正数），**x=1.1001**（负数）
  - x_0表示数的符号位；x_1～x_n是数值的有效部分，也称为尾数；x_1为最高有效位。
- 2、定点整数（纯整数）
  - $x=x_0x_1x_2…x_n$
  - 例如：**x=0,1101**（正数），**x=1,1001**（负数）
  - x_0表示数的符号位；x_1～x_n是数值的有效部分。
  - **C**语言中的**char**、**short**、**int**、**long**都属于定点整数。

– 3、定点数表示范围

- 定点数的表示范围与机器字长以及机器码（机器数的表示方法：原码、反码、补码、移码）有关。
- 若计算机字长为n+1（含1位符号位），则它可以表示2^{n+1}个数据状态。采用不同机器码（原码、反码、补码、移码）的定点数表示范围为：

表2.4　定点数的表示范围（见教材）

	定点整数		定点小数	
原码、反码	$[1-2^n, 2^n-1]$	$(-2^n, 2^n)$	$[-2^{-n}-1, 1-2^{-n}]$	$(-1, 1)$
补码	$[-2^n, 2^n-1]$	$[-2^n, 2^n)$	$[-1, 1-2^{-n}]$	$[-1, 1)$
移码	$[-2^n, 2^n-1]$	$[-2^n, 2^n)$	小数无移码	

- 数轴上的定点数的表示范围：

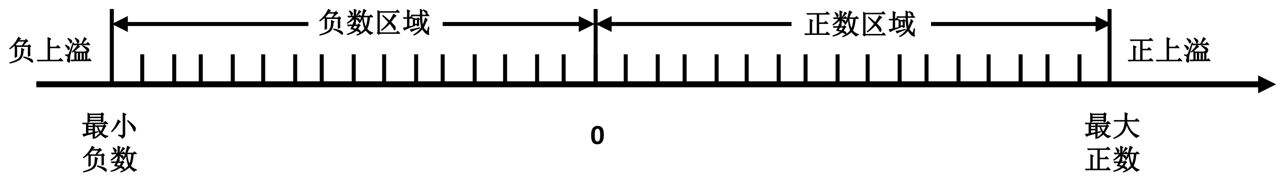


图2.5　定点数的表示范围（见教材）

- 对于定点整数，数轴上每个刻度间隔为1。

- 对于定点小数，数轴上每个刻度间隔为2^{-n}。

- 溢出：当数据超出计算机所能表示的数据范围时称为溢出，数据大于最大正数时，称为正上溢；数据小于最小负数时，称为负上溢。

  – 例如：假设n=7（包括符号位共8位），整数补码的表示范围为：-128～+127；如果数据=130，则发生正上溢；如果数据=-130，则发生负上溢。

  – 例如：假设n=7（包括符号位共8位），小数补码的表示范围为：-1～+(1-2^{-7})；如果数据=1，则发生正上溢；如果数据=-2，则发生负上溢。

- 精度溢出：对于小数还存在精度的问题，所有不在数轴上的小数都超出了定点小数所能表示的精度，无法表示，此时定点小数发生精度溢出，只能采用舍入的方法近视表示。

  - 例如：假设n=3（包括符号位共4位），小数补码的表示范围为：$-1\sim+(1-2^{-3})$；数轴上每个刻度间隔为2^{-3}。数轴上的数值为：-1，-7/8，-6/8，-5/8，-4/8，-3/8，-2/8，-1/8，0，1/8，2/8，3/8，4/8，5/8，6/8，7/8。

  - 对于x=5/16（x=2.5/8），在数轴上无法表示，发生精度溢出。因为x=5/16=0.0101，小数点后面有4位，超出了3位（n=3）。

  - 此时，可以采用舍入的方法近似表示；或者是x=0.010=2/8=4/16；或者是x=0.011=3/8=6/16。

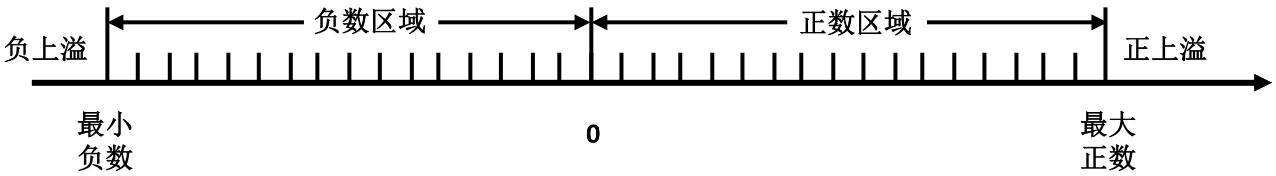


图2.5　定点数的表示范围（见教材）

• 2.2.3 浮点数表示

– 1、浮点数的表示形式

- 定点数的小数点是固定的（定点小数，小数点在符号位的右边；定点整数，小数点在最右边）；而浮点数的小数点位置是不固定的，小数点位置可以浮动，故称为浮点数。
  - 例如：$123.456 = 0.123456\text{x}10^3 = 1.23456\text{x}10^2 = 12.3456\text{x}10^1$（小数点位置可以浮动）
- 浮点数表示为：$N = 2^E\text{x}M = 2^{\pm e}\text{x}(\pm 0.m)$
  - E称为阶码（Exponent），为纯整数，e为阶码的数值部分；M称为尾数（Mantissa），为纯小数，m为尾数的数值部分。

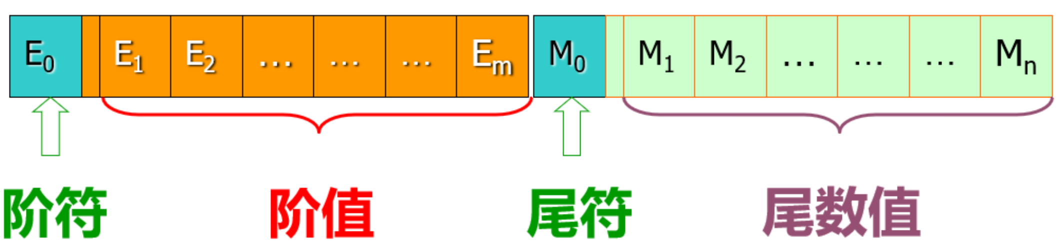


图2.6 浮点数的数据格式（见教材）

– 2、浮点数的表示范围

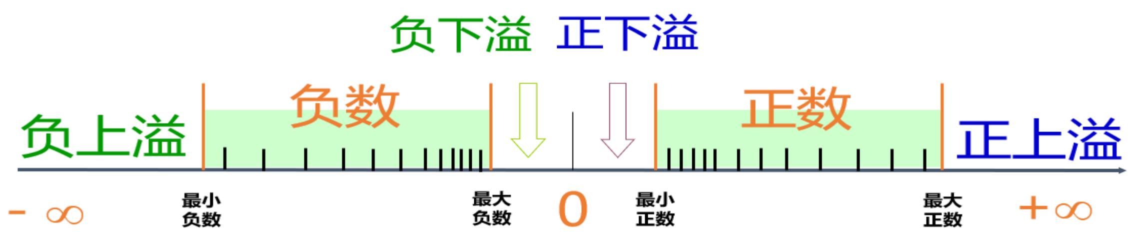


图2.7　浮点数的表示范围（见教材）

- 最大正数：阶码为最大数、尾数为最大正数
- 最小正数：阶码为最小数、尾数为最小正数
- 最大负数：阶码为最小数、尾数为最大负数
- 最小负数：阶码为最大数、尾数为最小负数

- 正上溢（+∞）：浮点数大于最大正数
- 负上溢（-∞）：浮点数小于最小负数
- 正下溢（作为机器0处理）：浮点数正数小于最小正数
- 负下溢（作为机器0处理）：浮点数负数大于最大负数

- 精度溢出：某个浮点数在表示区间，但不在数轴刻度上，此时只能用近似数表示。

- 浮点数的数轴刻度是不均匀的（定点数的数轴刻度是均匀的），越往数轴的左右两端，刻度越稀疏（为什么？请同学们自行分析！）。

- 例如16位浮点数，其中阶码为5位（含1位阶符，用移码表示）、尾数为11位（含1位数符，用补码表示），则：

  - 阶码的表示范围为：-16～15；0,0000～1,1111
  - 尾数的表示范围为：-1～$+(1-2^{-10})$；1.00 0000 0000～0.11 1111 1111

  - 阶码最大数：+15；阶码最小数：-16
  - 尾数最大正数：$+(1-2^{-10})$；尾数最小正数：$+2^{-10}$；尾数最大负数：-2^{-10}；尾数最小负数：-1

  - 浮点数最大正数：$2^{15}\times(1-2^{-10})$；1,1111　0.11 1111 1111
  - 浮点数最小正数：$2^{-16}\times2^{-10}$；　0,0000　0.00 0000 0001
  - 浮点数最大负数：$2^{-16}\times(-2^{-10})$；　0,0000　1.11 1111 1111
  - 浮点数最小负数：$2^{15}\times(-1)$；　1,1111　1.00 0000 0000

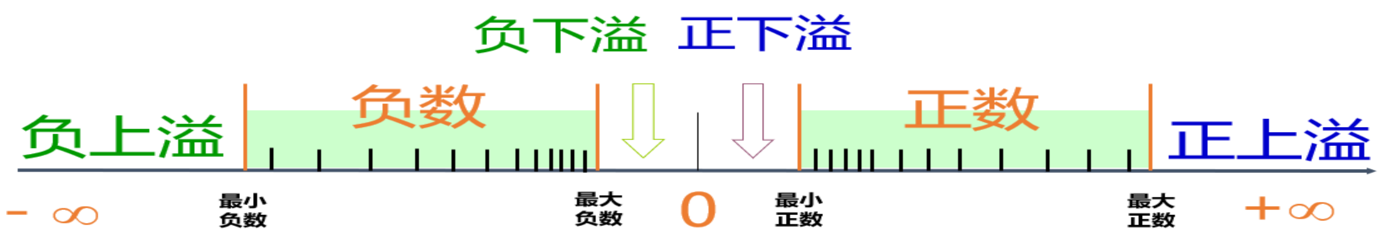


图2.7　浮点数的表示范围

- 3、浮点数的规格化

- 十进制浮点数的规格化：
  - 123.456 = 0.123456x10^3
  - 尾数为纯小数，且尾数的绝对值大于等于0.1、小于1，[0.1, 1)。
  - 非规格化的十进制浮点数：123.456 = 0.0123456x10^4
  - 非规格化的十进制浮点数：123.456 = 1.23456x10^2

- 二进制浮点数的规格化：
  - 1101.1001 = 0.11011001x2^{100}
  - 尾数为纯小数，且尾数的绝对值大于等于0.5、小于1，[0.5, 1)；即尾数的最高有效位为1（尾数用原码表示）。
  - 非规格化的二进制浮点数：1101.1001 = 0.011011001x2^{101}
  - 非规格化的二进制浮点数：1101.1001 = 1.1011001x2^{011}

- 非规格化的浮点数可以通过左规或右规的方法变为规格化的浮点数。

- 左规：如果尾数的绝对值小于0.5，则需要对尾数进行左移，每左移一次，尾数乘2，阶码减1，直到尾数的绝对值大于等于0.5。
  - 1101.1001 = $0.0011011001 \times 2^{110}$ （非规格化）
  - 左移1次，得到0.011011001，阶码为101；再左移1次，得到0.11011001，阶码为100；最后得到规格化的浮点数：$0.11011001 \times 2^{100}$

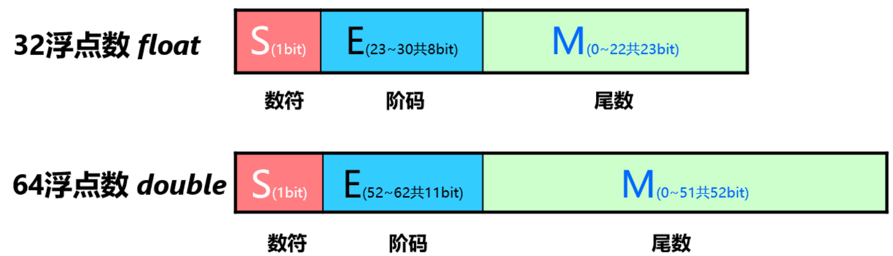
- 右规：如果尾数的绝对值大于等于1，则需要对尾数进行右移，每右移一次，尾数除2，阶码加1，直到尾数的绝对值小于1。
  - 1101.1001 = 11.011001×2^{10} （非规格化）
  - 右移1次，得到1.1011001，阶码为11；再右移1次，得到0.11011001，阶码为100；最后得到规格化的浮点数：$0.11011001 \times 2^{100}$

- 另一种规格化的浮点数：
  - $N = 2^E \times M = 2^{\pm e} \times (\pm 1.m)$ $(1 \le |M| < 2)$
  - 尾数的绝对值大于等于1、小于2，[1, 2)

- 隐藏位：当尾数采用原码表示时，规格化的尾数最高有效位一定是1，可以将最高有效位的1隐藏，从而节省1位存储空间，被隐藏的这一位称为隐藏位。

– 4、IEEE754浮点数标准

- 1985年，美国电气及电子工程师协会（IEEE）发布了浮点数标准IEEE754，其主要设计者威廉·卡亨（**William Kahan**）教授因此贡献获得1989年**图灵奖**。



- IEEE754浮点数标准：
  - 32位**单精度浮点数**：数符1位、阶码8位、尾数23位；对应C语言中的**float**型数据。
  - 64位**双精度浮点数**：数符1位、阶码11位、尾数52位；对应C语言中的**double**型数据。
  - 表2.5（见教材） **IEEE754数据格式规范**：包括半精度浮点数（16位）、单精度浮点数（32位）、扩展单精度浮点数（≥43位）、双精度浮点数（64位）、扩展双精度浮点数（≥79位）、四精度浮点数（128位）、八精度浮点数（256位）、Decimal32（32位）、Decimal64（64位）、Decimal128（128位）等。

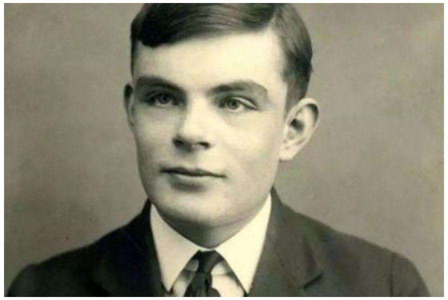
图灵奖

- 图灵奖（Turing Award），全称A.M.图灵奖（ACM A.M Turing Award），是由美国计算机协会（ACM）于1966年设立的计算机奖项，名称取自艾伦·麦席森·图灵（Alan M. Turing），旨在奖励对计算机事业作出重要贡献的个人。图灵奖对获奖条件要求极高，评奖程序极严，一般每年仅授予一名计算机科学家。图灵奖是计算机领域的国际最高奖项，被誉为“计算机界的诺贝尔奖”。

- 图灵奖一般在每年3月下旬颁发。从1966年至2020年，图灵奖共授予74名获奖者，以美国、欧洲科学家为主。2000年，中国科学家姚期智获图灵奖，这是中国人第一次也是唯一一次获得图灵奖。

- 截至2021年4月，世界各高校的图灵奖获奖人数依次为美国斯坦福大学（29位）、美国麻省理工学院（26位）、美国加利福尼亚大学伯克利分校（25位）、美国普林斯顿大学（16位）、美国哈佛大学（14位）。

- 艾伦·麦席森·图灵（Alan Mathison Turing，1912年6月23日—1954年6月7日），英国数学家、逻辑学家，被称为计算机之父、人工智能之父。1931年，图灵进入剑桥大学国王学院，毕业后到美国普林斯顿大学攻读博士学位。二战爆发后，回到剑桥大学，后曾协助军方破解德国的著名密码系统Enigma，帮助盟军取得了二战的胜利。图灵对于人工智能的发展有诸多贡献，提出了一种用于判定机器是否具有智能的试验方法，即图灵试验。每年都有试验的比赛。此外，图灵提出的著名的图灵机模型为现代计算机的逻辑工作方式奠定了基础。



– 5、IEEE754单精度浮点数

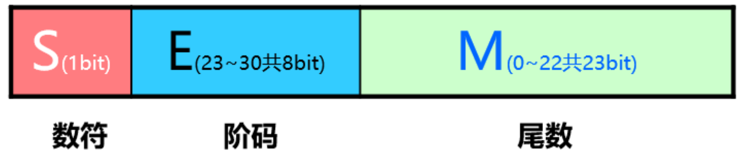


图2.8　IEEE754 32位单精度浮点数（见教材）

- 阶码E采用移码表示，其偏移量是127，而不是标准移码的128。
  - IEEE754阶码的真值=E-127；E=0～255，真值为：-127～+128
  - 标准移码的真值=E-128；E=0～255，真值为：-128～+127
- 尾数M为定点小数，其形式为1.M，1隐含了，只需保存M，节省了1位存储空间。
- 符号位S为0时表示正数，为1时表示负数。

- **IEEE754单精度浮点数规范：**
  - S=0/1，E=255，M≠0，浮点数真值=NaN（Not a Number，非数）；运算异常（例如0除0）
  - S=0/1， E=255，M=0，浮点数真值=+∞/-∞；正、负无穷
  - S=0/1， E=1～254，M，浮点数真值=$(-1)^S \times 2^{E-127} \times 1.M$；规格化数
  - S=0/1， E=0，M≠0，浮点数真值=$(-1)^S \times 2^{-126} \times 0.M$；非规格化数
  - S=0/1， E=0，M=0，浮点数真值=+0/-0；两个机器0

表2.6　IEEE754单精度浮点数规范（见教材）

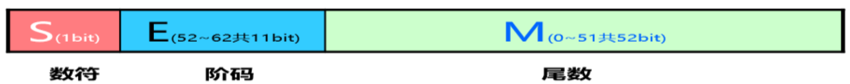
符号位 S	阶码 E	尾数 M	表示
0/1	255	M(非零)	NaN　Not a Number
0/1	255	0	+∞/-∞
0/1	1~254	M	$(-1)^S \times (1.M) \times 2^{(E-127)}$
0/1	0	M (非零)	$(-1)^S \times (0.M) \times 2^{(-126)}$
0/1	0	0	+0/-0

- IEEE754浮点数表示范围（单精度浮点数）：
  - 单精度规格化浮点数绝对值最小数：E=1，M=0，$f=2^{1-127} \times 1.0=2^{-126}$
  - 单精度规格化浮点数绝对值最大数：E=254，M=11…11（尾数=1.11…11），$f=2^{254-127} \times (2-2^{-23})=2^{128}-2^{-104} \approx +3.4 \times 10^{38}$
  - 单精度非规格化浮点数绝对值最小数：E=0，M=00…01（尾数=0.00…01），$f=2^{-126} \times 2^{-23}=2^{-149}$
  - 单精度非规格化浮点数绝对值最大数：E=0，M=11…11（尾数=0.11…11），$f=2^{-126} \times (1-2^{-23})=2^{-126}-2^{-149}$

表2.7　IEEE754浮点数表示范围（见教材）

格式	最小值	最大值
单精度规格化 $(-1)^s \times 1.m \times 2^{E-127}$	$E_{min}=1, M=0,$ $1.0 \times 2^{1-127}=2^{-126}$	$E_{max}=254, M=1.1111...1 \times 2^{254-127}$ $=2^{127} \times (2-2^{-23})$ 约 $+3.4 \times 10^{38}$
单精度非规格化 $(-1)^s \times 0.m \times 2^{-126}$	$E=0, M=2^{-23},$ $2^{-23} \times 2^{-126}=2^{-149}$	$E=0, M=0.1111...1 \times 2^{-126}$ $=2^{-126} \times (1-2^{-23})$
双精度规格化 $(-1)^s \times 1.m \times 2^{E-1023}$	$E_{min}=1, M=0,$ $1.0 \times 2^{1-1023}=2^{-1022}$	$E_{max}=2046,$ $M=1.1111...1 \times 2^{2046-1023}$ $=2^{1023} \times (2-2^{-52})$ 约 $+1.8 \times 10^{308}$
双精度非规格化 $(-1)^s \times 0.m \times 2^{-1022}$	$E=0, M=2^{-52},$ $2^{-52} \times 2^{-1022}=2^{-1079}$	$E=0, M=0.11111...1 \times 2^{-1022}$ $=2^{-1022} \times (1-2^{-52})$

- **IEEE754浮点数表示范围（双精度浮点数）：**
  - 双精度规格化浮点数绝对值最小数：E=1，M=0，$f=2^{1-1023}\times1.0=2^{-1022}$
  - 双精度规格化浮点数绝对值最大数：E=2046，M=11...11（尾数=1.11...11），$f=2^{2046-1023}\times(2-2^{-52})=2^{1024}-2^{-971}\approx+1.8\times10^{308}$
  - 双精度非规格化浮点数绝对值最小数：E=0，M=00...01（尾数=0.00...01），$f=2^{-1022}\times2^{-52}=2^{-1074}$
  - 双精度非规格化浮点数绝对值最大数：E=0，M=11...11（尾数=0.11...11），$f=2^{-1022}\times(1-2^{-52})=2^{-1022}-2^{-1074}$
  - 双精度浮点数阶码E采用移码表示时偏移量为1023，而不是标准移码的1024。



IEEE754双精度浮点数

表2.7　IEEE754浮点数表示范围（见教材）

格式	最小值	最大值
单精度规格化 $(-1)^s\times1.m\times2^{E-127}$	$E_{min}=1, M=0,$ $1.0\times2^{1-127}=2^{-126}$	$E_{max}=254, M=1.1111...1\times2^{254-127}$ $=2^{127}\times(2-2^{-23})$ 约 $+3.4\times10^{38}$
单精度非规格化 $(-1)^s\times0.m\times2^{-126}$	$E=0, M=2^{-23},$ $2^{-23}\times2^{-126}=2^{-149}$	$E=0, M=0.1111...1\times2^{-126}$ $=2^{-126}\times(1-2^{-23})$
双精度规格化 $(-1)^s\times1.m\times2^{E-1023}$	$E_{min}=1, M=0,$ $1.0\times2^{1-1023}=2^{-1022}$	$E_{max}=2046,$ $M=1.1111...1\times2^{2046-1023}$ $=2^{1023}\times(2-2^{-52})$ 约 $+1.8\times10^{308}$
双精度非规格化 $(-1)^s\times0.m\times2^{-1022}$	$E=0, M=2^{-52},$ $2^{-52}\times2^{-1022}=2^{-1079}$	$E=0, M=0.11111...1\times2^{-1022}$ $=2^{-1022}\times(1-2^{-52})$

– 6、单精度浮点数与真值之间的转换流程

- 将十进制数N转换为单精度浮点数（32位二进制数）
- 将单精度浮点数（32位二进制数）转换为十进制数N

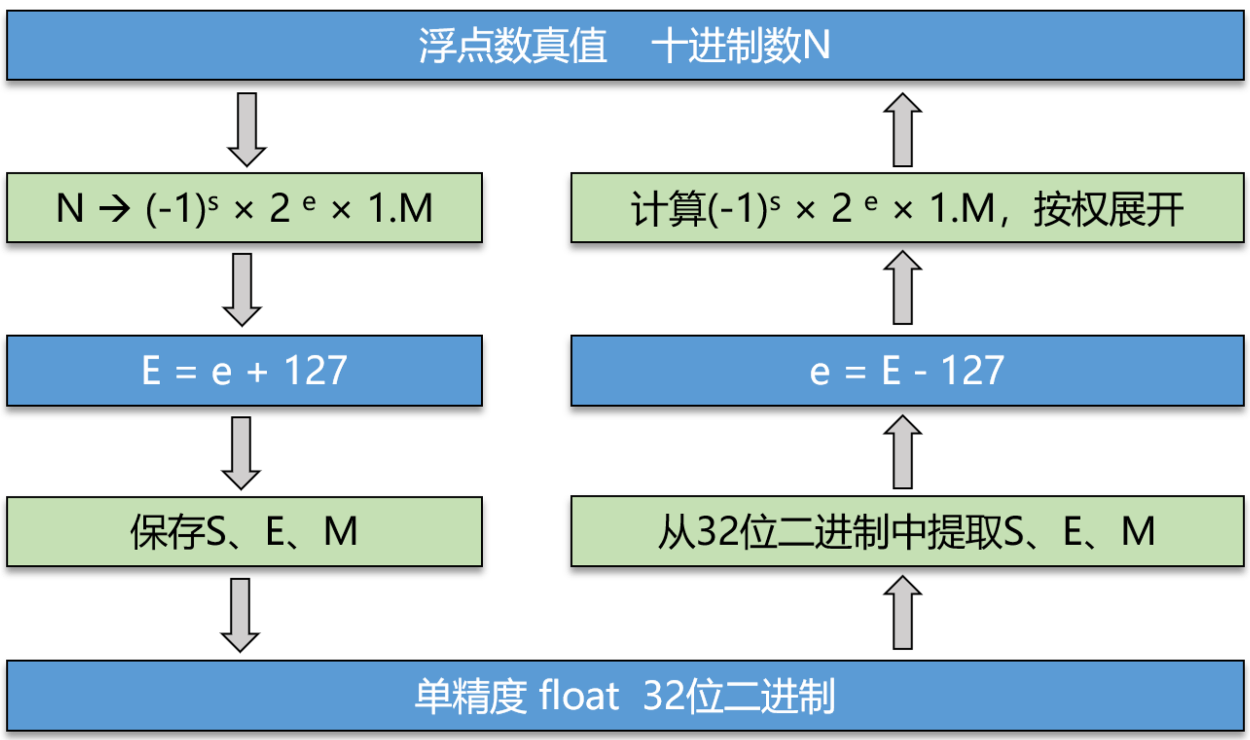


图2.10 单精度浮点数与真值之间的转换流程（见教材）

- 例2.7：将十进制数20.59375转换成IEEE754单精度浮点数的十六进制机器码。
- 解：
  - $(20.59375)_{10} = (10100.10011)_2$
  - $10100.10011 = 1.010010011 \times 2^4 = 1.M \times 2^e$
  - S=0（正数），e=4，E=e+127=131 = 10000011，M = 010010011
  - 单精度浮点数格式：0　1000 0011　010 0100 1100 0000 0000 0000
  - 最终的机器码=0100 0001 1010 0100 1100 0000 0000 0000=41A4C000H

- 例2.8：求IEEE754单精度浮点数C136000H对应的十进制值。
- 解：
  - C136000H=1100 0001 0011 0110 0000 0000 0000
  - 单精度浮点数=1　1000 0010　011 0110 0000 0000 0000
  - S=1（负数），E=1000 0010=130，M=011011
  - e=E-127=130-127=3
  - 尾数=1.M=1.011011
  - 浮点数对应的十进制值$=-2^3 \times 1.011011=-1011.011=(-11.375)_{10}$

- 十进制转换为二进制（20.59375对应的二进制是多少？）
  - 整数部分：
    - 20 = 16+4 = 10000 + 0100 = 10100
  - 小数部分：
    - 0.59375x2=1.1875
    - 0.1875x2=0.375
    - 0.375x2=0.75
    - 0.75x2=1.5
    - 0.5x2=1
    - 0.59375=0.10011
  - 20.59375=10100.10011

- 二进制转换为十进制（-1011.011_B对应的十进制是多少？）
  - 整数部分：$1011=(11)_{10}$
  - 小数部分：0.011=1x0.25+1x0.125=0.375
  - $-1011.011_B=(-11.375)_{10}$

- # 2.2.4 十进制数编码

  - ## 1、十进制整数

    ① BCD码（Binary Coded Decimal，二进制编码的十进制数）

    - 8421码：4位二进制的权值分别为8、4、2、1
      - » 0000(0),0001(1),0010(2),0011(3),0100(4),0101(5),0110(6),0111(7),1000(8),1001(9)
      - » 23表示为0010 0011（一目了然），如果采用二进制表示则为0001 0111
    - 2421码：4位二进制的权值分别为2、4、2、1
      - » 0000(0),0001(1),0010(2),0011(3),0100(4),1011(5),1100(6),1101(7),1110(8),1111(9)
      - » 2421码具有自补的特点，即各位取反后正好是该数对9的补码
      - » 如0000(0)取反得到1111(9)；0001(1)取反得到1110(8)；0010(2)取反得到1101(7)；0011(3)取反得到1100(6)；0100(4)取反得到1011(5)
    - 余3码：为8421码+3
      - » 0011(0),0100(1),0101(2),0110(3),0111(4),1000(5),1001(6),1010(7),1011(8),1100(9)
    - BCD码的编码效率=10/16（用4位二进制数表示1位十进制数，2^4=16）

② BID码（Binary Integer Decimal，十进制整数的二进制表示）

- 直接用二进制整数编码表示十进制整数
- 如十进制数20，其BID码=10100

③ DPD码（Densely Packed Decimal，紧凑十进制编码）

- 利用10位二进制数表示3位十进制数
- 表2.9（见教材） DPD码编码格式（8种情况）：
  - (0～7) (0～7) (0～7)：3个小数（例如375）
  - (0～7) (0～7) (8～9)：两小一大（例如378）
  - (0～7) (8～9) (0～7)：两小一大（例如385）
  - (8～9) (0～7) (0～7)：两小一大（例如875）
  - (8～9) (8～9) (0～7)：两大一小（例如895）
  - (8～9) (0～7) (8～9)：两大一小（例如879）
  - (0～7) (8～9) (8～9)：两大一小（例如389）
  - (8～9) (8～9) (8～9)：3个大数（例如899）
- 例如十进制数375，属于3个小数，D2=3=0abc=0011，D1=7=0def=0111，D0=5=0ghi=0101，DPD码=abcdef0ghi=0111110101
- DPD码的编码效率=1000/1024（10位二进制数：2^{10}=1024，3位十进制数：10^3=1000），高于BCD码的编码效率（10/16）

– 2、十进制浮点数

- 二进制浮点数不能精确表示十进制数（**精度溢出**）
- 如**0.7=0.1011001100110011001100……**；**1.05=1.0000110011001100……**；都不能用二进制数精确表示

- 财务结算中**0.7x1.05=0.735**元，四舍五入，得到**0.74**元
- 但是如果采用双精度浮点数进行计算：**0.7x1.05=0.73499999999999999**，四舍五入，得到**0.73**元，相差**1**分钱
- **Java**中的**BigDecimal**，是采用软件的方法实现十进制运算

BigDecimal

播报 编辑 讨论 上传视频

本词条由"科普中国"科学百科词条编写与应用工作项目 审核 。

Java在java.math包中提供的API类BigDecimal，用来对超过16位有效位的数进行精确的运算。双精度浮点型变量double可以处理16位有效数。在实际应用中，需要对更大或者更小的数进行运算和处理。float和double只能用来做科学计算或者是工程计算，在商业计算中要用java.math.BigDecimal。BigDecimal所创建的是对象，我们不能使用传统的+、-、*、/等算术运算符直接对其对象进行数学运算，而必须调用其相对应的方法。方法中的参数也必须是BigDecimal的对象。构造器是类的特殊方法，专门用来创建对象，特别是带有参数的对象。 [1]

外文名	BigDecimal	功　能	对超过16位有效位的数进行精确的运算
包　括	带有参数的对象	关　联	float、double
提　供	API类BigDecimal	应　用	商业计算

- **IEEE754-2008的十进制浮点数格式（教材图2.11）：**
  - 数符：s
  - 5位组合字段：comb，包含阶码和尾数的部分数据位：
    - » 格式①（尾数最高位=0～7）：comb的高2位为阶码的最高有效位（00～10），comb的低3位为尾数的最高有效位（000～111，0～7）
    - » 格式②（尾数最高位=8～9）：comb的高2位为11，comb的后续2位为阶码的最高有效位（00～10），comb的最低位为0则表示尾数的最高有效位为1000(8)、为1则表示尾数的最高有效位为1001(9)
    - » 格式③（无穷大或非数）：comb的高4位为1111，表示无穷大或非数（NaN）
  - 部分阶码：E（阶码的最高2位，存放在comb中）
  - 部分尾数：T（尾数的最高3位或1位，存放在comb中）

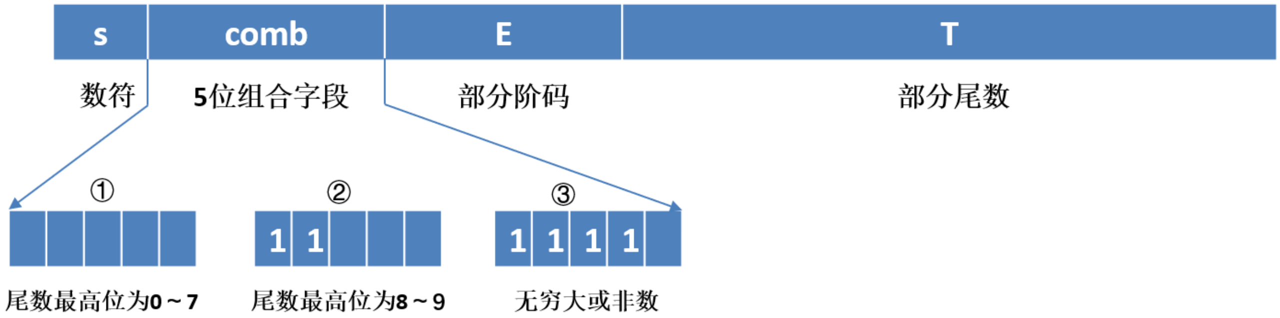


图2.11　IEEE754-2008的十进制浮点数格式（见教材）

- 十进制浮点数：$N = (-1)^s x10^{E-bias} xT$
  - 这里的基数为10，不是2；尾数T是定点整数，不是定点小数
- 表2.10（见教材） 不同位宽十进制浮点数格式的参数：
  - 32位（_Decimal32）：s（1位）、comb（5位）、E（6位）、T（20位）、bias=101（十进制），E的范围 = -101～90（E=00 000000～10 111111=0～191）
  - 64位（_Decimal64）：s（1位）、comb（5位）、E（8位）、T（50位）、bias=398（十进制），E的范围 = -398～369 （E=00 00000000～10 11111111=0～767）
  - 128位（_Decimal128）：s（1位）、comb（5位）、E（12位）、T（110位）、bias=6176（十进制），E的范围 = -6176～6111 （E=00 000000000000～10 111111111111=0～12287）
- 例如，十进制浮点数：$123.456 = 123456 \times 10^{-3}$；采用32位（_Decimal32）表示，则有：
  - 阶码=e+bias=-3+101=98=01 100010；尾数=123456=000 0001 1110 0010 0100 0000
  - 因为尾数最高位=0，故属于格式①
  - 数符s=0（正数）
  - 5位组合字段comb=01 000
  - 部分阶码E=100010（6位）
  - 部分尾数T= 0001 1110 0010 0100 0000（20位，采用BID码，即直接用二进制表示）

• 2.2.5 计算机中的数据类型

– 1、汇编语言中的数据类型

- 汇编语言中的操作数究竟是定点数还是浮点数（单精度浮点数、双精度浮点数）、是有符号数还是无符号数，完全取决于指令操作符。

- 表2.11（见教材）：汇编语言中不同指令集的数据运算类型

表2.11　汇编语言中不同指令集的数据运算类型（见教材）

ISA	无符号运算	有符号运算	浮点运算
X86	ADD/SUB 加减		FADD/FSUB 加减
	MUL/DIV 乘除	IMUL/IDIV 乘除	FMUL/FDIV 乘除
MIPS32	ADDU/SUBU 加减	ADD/SUB 加减	ADD.S ADD.D /SUB.S SUB.D 加减
	MULTU/DIVU 乘除	MULT/DIV 乘除	MUL.S MUL.D / DIV.S DIV.D 乘除
RISC-V32	ADD/SUB 加减		FADD.S FADD.D / FSUB.S FSUB. D 加减
	MULHU/DIVU 乘除	MULH/DIV 乘除	FMUL.S FMUL.D / FDIV. S FDIV. D 乘除

U：无符号数　　I：有符号数

F：浮点数　　S：单精度浮点数　　D：双精度浮点数

– 2、高级语言中的数据类型

- C语言的数据类型：
  - 整数： char（8位）、short（16位）、int（32位）、long（64位）
  - 浮点数：float（32位）、double（64位）
  - 默认为有符号数，在整数前加“unsigned”表示无符号数

- （1）C语言整型数据表示范围
  - 4位无符号数表示范围：0～15；如果两个无符号数相加大于15（7+9=16），或者两个无符号数相减小于0（7-9=-2），则会产生无符号溢出。见教材图2.12(a)。
  - 4位有符号数（补码）表示范围：-8～+7；如果两个有符号数相加（或相减）大于+7（3+5=8，3-(-5)=8），则会产生正上溢；如果两个有符号数相加（或相减）小于-8（-5+(-6)=-11，-5-6=-11），则会产生负上溢。见教材图2.12(b)。

- 整型变量的取值范围（有符号数采用补码表示）：

表2.12　整型变量的取值范围（见教材）

值		*char* (8 位)	*short* (16 位)	*int* (32 位)	*long* (64 位)
无符号最大值	机器码	0xFF	0xFFFF	0xFFFFFFFF	0xFFFFFFFFFFFFFFFF
	真值	255	65,535	4,294,967,295	18,446,744,073,709,551,615
有符号最小值	机器码	0x80	0x8000	0x80000000	0x8000000000000000
	真值	−128	−32,768	−2,147,483,648	−9,223,372,036,854,775,808
有符号最大值	机器码	0x7F	0x7FFF	0x7FFFFFFF	0x7FFFFFFFFFFFFFFF
	真值	127	32,767	2,147,483,647	9,223,372,036,854,775,807
-1	机器码	0xFF	0xFFFF	0xFFFFFFFF	0xFFFFFFFFFFFFFFFF
0	机器码	0x00	0x0000	0x00000000	0x0000000000000000

- （2）C语言数据表示实例

输出7个变量（32位整数、无符号整数、浮点数；16位短整数、无符号短整数；8位字符、无符号字符）的机器码和真值

```
#include "studio.h"

union
{
   int i;  unsigned int ui;  float f;              //32位整数、无符号整数、浮点数
   short s;  unsigned short us;                    //16位短整数、无符号短整数
   char c;  unsigned char uc;                      //8位字符、无符号字符
}t;

void hex_out(char a)                               //输出8位数据的十六进制值
{
   const char HEX[]="0123456789ABCDEF";
   printf("%c%c",HEX[(a&0xF0)>>4],HEX[a&0xF]);
}

void out_1byte(char *addr)                         //用十六进制输出地址中的8位数据机器码
{
   hex_out (*(addr +0));
}

void out_2byte(char *addr)                         //用十六进制输出地址中的16位数据机器码
{
   hex_out (*(addr +1));
   hex_out (*(addr +0));
}

void out_4byte(char *addr)                         //用十六进制输出地址中的32位数据机器码
{
   hex_out (*(addr +3));
   hex_out (*(addr +2));
   hex_out (*(addr +1));
   hex_out (*(addr +0));
}
```

输出7个变量（32位整数、无符号整数、浮点数；16位短整数、无符号短整数；8位字符、无符号字符）的机器码和真值

```
main()
{
  t.i=0xC77FFFFF;

  out_4byte(&t.i);                //输出i（整数）的机器码和真值
  printf(" = %d \n",t.i);         //C77FFFFF = -947912705
```

C77FFFFF=1100 0111 0111 1111 1111 1111 1111 1111（补码）对应的十进制数：1 0000 0000H-C77F FFFFH = -947912705

```
  out_4byte(&t.ui);               //输出ui（无符号整数）的机器码和真值
  printf(" = %u \n",t.ui);        //C77FFFFF = 3347054591
```

C77FFFFF=1100 0111 0111 1111 1111 1111 1111 1111（无符号数）对应的十进制数：C77F FFFFH = 3347054591

```
  out_4byte(&t.f);                //输出f（浮点数）的机器码和真值
  printf(" = %f \n",t.f);         //C77FFFFF = -65535.996094
```

C77FFFFF=1 100 0111 0111 1111 1111 1111 1111 1111（IEEE754浮点数）对应的十进制数：

E=10001110， M=111 1111 1111 1111 1111 1111

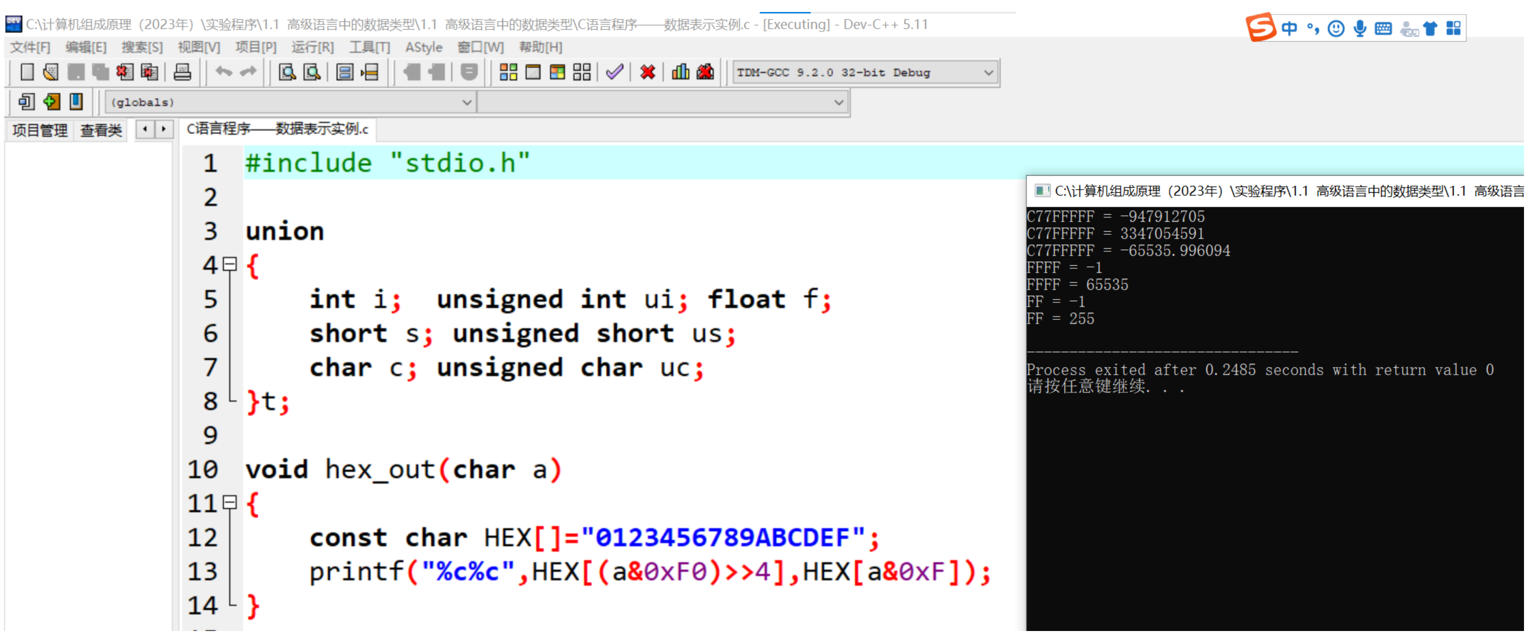
阶码=e=E-127=10001110-127=15，尾数=1.M=1.111 1111 1111 1111 1111 1111

对应的十进制数 = -2^{15}x1.111 1111 1111 1111 1111 = -2^{15}x$(2-2^{-23})$=$-(2^{16}-2^{-8})$ = -(65536-0.00390625) = -65535.99609375

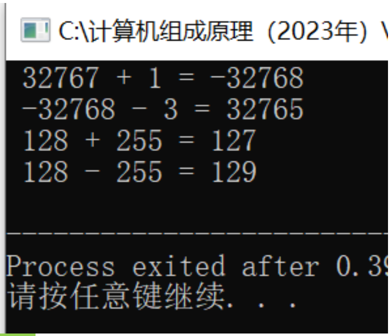
```
  out_2byte(&t.s);                //输出s（短整数）的机器码和真值
  printf(" = %d \n",t.s);         //FFFF = -1

  out_2byte(&t.us);               //输出us（无符号短整数）的机器码和真值
  printf(" = %u \n",t.us);        //FFFF = 65535

  out_1byte(&t.c);                //输出c（字符）的机器码和真值
  printf(" = %d \n",t.c);         //FF = -1

  out_1byte(&t.uc);               //输出uc（无符号字符）的机器码和真值
  printf(" = %d \n",t.uc);        //FF = 255
}
```

在电脑上运行书上的C语言程序

- 第一步：解压“devc++.zip”，运行解压后的“devcppPortable.exe”
- 第二步：在Dev C++ 5.11中打开源程序“C语言程序——数据表示实例.c”
- 第三步：点击“运行”将编译并运行程序



- （3）C语言运算溢出实例



4种运算溢出的例子（2种短整数运算、2种无符号数运算）

```
void main()
{
   short s1=32767,s2=-32768,s;               //短整数（16位）
   unsigned char uc1=128,uc2=255,uc;         //无符号字符（8位）

   s=s1+1;
   printf("%d + 1 = %d\n",s1,s);             //输出32767 + 1 = -32768    正正得负，出错，正上溢
```

32767 + 1 = 0111 1111 1111 1111 + 1 = 1000 0000 0000 0000 = -32768（补码）

```
   s=s2-3;
   printf("%d - 3 = %d\n",s2,s);             //输出-32768 - 3 = 32765    负负得正，出错，负上溢
```

-32768-3 = -32768 + (-3) = 1000 0000 0000 0000 + 1111 1111 1111 1101 = 0111 1111 1111 1101 = 32765（补码）

```
   uc=uc1+uc2;
   printf("%d + %d = %d\n",uc1,uc2,uc);      //输出128 + 255 = 127    越加越小，出错，无符号溢出
```

128 + 255 = 1000 0000 + 1111 1111 = 0111 1111 = 127（无符号数）

```
   uc=uc1-uc2;
   printf("%d - %d = %d\n",uc1,uc2,uc);      //输出128 - 255 = 129    越减越大，出错，无符号溢出
```

128 - 255 = 1000 0000 - 1111 1111 = 1000 0000 + (1111 1111 取反加1) = 1000 0000 + 0000 0001 = 1000 0001 = 129（无符号数）

```
}
```

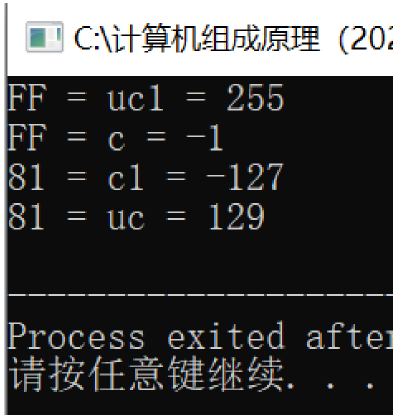
不能采用短整数（16位），而应该采用整数（32位）

不能采用无符号字符（8位），而应该采用整数（32位）或短整数（16位）

- （4）C语言整型数据类型转换

① 相同字长的整型数据转换（8位转8位）

```
main()
{
   unsigned char uc1=255,uc;                   //无符号字符（8位）
   char c1=-127,c;                             //字符（8位）

   c=(char)uc1;                                //无符号字符（8位）转换为字符（8位）

   out_1byte(&uc1);
   printf(" = uc1 = %u \n",uc1);               //输出原数据的机器码和真值 FF = uc1 =255

                                               uc1 = 255 = FF（无符号数）

   out_1byte(&c);
   printf(" = c = %d \n",c);                   //输出转换后的机器码和真值 FF = c = -1

                                               c = uc1 = FF = 1111 1111 = -1（补码）

   uc=c1;                                      //字符（8位）转换为无符号字符（8位）

   out_1byte(&c1);
   printf(" = c1 = %d \n",c1);                 //输出原数据的机器码和真值 81 = c1 = -127

                                               c1 = -127 = 1000 0001 = 81H（补码）

   out_1byte(&uc);
   printf(" = uc = %u \n",uc);                 //输出转换后的机器码和真值 81 = c1 = 129

                                               uc = -127 = 1000 0001 = 81H = 129（无符号数）

}
```



FF为无符号字符时，
真值=255

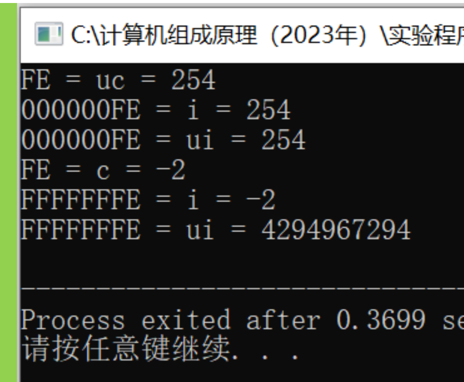
FF为字符时，真值=-1

-127为字符时，机器码
=81H

81H为无符号字符时，
真值=129

② 小字长转大字长（8位转32位）

```
main()
{
  unsigned char uc=254; char c=uc;           //无符号字符（8位）、字符（8位）
  int i;  unsigned ui;                       //整数（32位）、无符号整数（32位）

  i=uc; ui=uc;

  out_1byte(&uc);
  printf(" = uc = %d \n",uc);        //输出原数据的机器码和真值  FE = uc =254
                                     uc = 254 = FE（无符号数）

  out_4byte(&i);
  printf(" = i = %d \n",i);          //输出转换后的机器码和真值  000000FE = i = 254
                                     i = 254 = 0000 00FE（32位整数）

  out_4byte(&ui);
  printf(" = ui = %u \n",ui);        //输出转换后的机器码和真值  000000FE = ui = 254
                                     ui = 254 = 0000 00FE（32位无符号整数）

  i=c; ui=c;

  out_1byte(&c);
  printf(" = c = %d \n",c);          //输出原数据的机器码和真值  FF = c = -2
                                       c = 254 = FE = -2（补码）

  out_4byte(&i);
  printf(" = i = %d \n",i);          //输出转换后的机器码和真值  FFFFFFFE = i = -2
                                       i = FE带符号扩展到32位后的值 = FFFF FFFE = -2（补码）

  out_4byte(&ui);
  printf(" = ui = %u \n",ui);        //输出转换后的机器码和真值  FFFFFFFE = ui = 4294967294
                                       ui = FFFF FFFE = 4294967294（无符号数）

}
```



③ 大字长转小字长（32位转16位）

```
main()
{
   int i=0xFFFF1001;                 //整数（32位）
   short s; unsigned short us;       //短整数（16位）、无符号短整数（16位）

   s=i;
   us=i;

   out_4byte(&i);
   printf(" = i = %d \n",i);        //输出原数据的机器码和真值  FFFF1001 = i = -61439
```

i = 0xFFFF 1001 = -61439（补码）=1 0000 0000H – FFFF 1001H

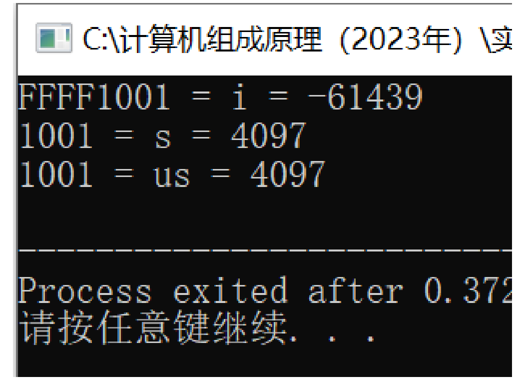
```
   out_2byte(&s);
   printf(" = s = %d \n",s);        //输出转换后的机器码和真值  1001 = s = 4097
```

s = 0x1001 = 4097（补码）

```
   out_2byte(&us);
   printf(" = us = %u \n",us);      //输出转换后的机器码和真值  1001 = us = 4097
```

us = 0x1001 = 4097（无符号数）

```
}
```



- （5）C语言中的浮点数据类型
  - 单精度浮点数：float（32位），双精度浮点数：double（64位）
  - 半精度浮点数：_Float16（16位），四精度浮点数：long double（128位）
  - 十进制浮点数：_Decimal32（32位）、_Decimal64（64位）、_Decimal128（128位）

  - int、float、double之间的转换：

    ① float（32位）转换为double（64位）：没有问题

    ② double（64位）转换为float（32位）：可以会发生溢出，或者精度丢失（舍入）

    ③ float（32位）/double（64位）转换为int（32位）：可以会发生溢出，或者精度丢失（舍入）

    ④ int（32位）转换为float（32位）：可能会发生精度溢出（int对应的数轴是均匀的，而float对应的数轴是不均匀的，两个数轴不完全对应）

    ⑤ int（32位）转换为double（64位）：没有问题

- 数据类型转换实例：

表2.13　数据类型转换实例（见教材）

C语言表达式	是否恒成立	原因
i==(int)(float)i	否	int转float，精度溢出
i==(int)double(i)	是	int转double，没有问题
f==(float)(int)f	否	float转int，溢出或精度丢失
f==(float)(double)f	是	float转double，没有问题
d==(float)d	否	double转float，溢出或精度丢失
f==-(-f)	是	浮点数采用原码表示，单目运算的功能只是符号取反
(d+f)-d==f	否	浮点数不满足结合律，如果d是一个大数，f是一个小数，式子左边的结果是0

i为整型量（int），f为单精度浮点数（float），d为双精度浮点数（double）

– 例2.9：已知 $f(n) = \sum_{i=0}^{n} 2^i = 2^{n+1}-1 = 11…1B$ (n+1个1)，计算f(n)的C语言函数f1如下：

» 例如：n=0，f(0)=1；n=1，f(1)=11；n=7，f(7)=1111 1111。

```
int f1(unsigned n)
{
   int sum=1,power=1;
   for(unsigned i=0; i<=n-1; i++)
   {
       power *= 2;
       sum += power;
   }
   return sum;
}
```

```
float f2(unsigned n)
{
   float sum=1,power=1;
   for(unsigned i=0; i<=n-1; i++)
   {
       power *= 2;
       sum += power;
   }
   return sum;
}
```

– 将f1中的int型数据都改为float型数据，可得到计算f(n)的另一个函数f2。假设unsigned和int型数据都占32位，float型数据采用IEEE754单精度标准。请回答以下问题：

– （1）当n=0时，f1会出现死循环，为什么？若将f1中的变量i和n都定义为int型，则f1是否还会出现死循环？为什么？

– （2）f1(23)和f2(23)的返回值是否相等？机器数各是什么（用十六进制表示）？

– （3） f1(24)和f2(24)的返回值分别是33554431和33554432.0，为什么不相等？

– （4）$f(31)=2^{32}-1$，而f1(31)的返回值却为-1，为什么？若要使f1(n)的返回值与f(n)相等，则最大的n是多少？

– （5）f2(127)的机器数为7F80 0000H，对应的值是什么？若要使f2(n)的结果不溢出，则最大的n是多少？

– 例2.9：已知 $f(n) = \sum_{i=0}^{n} 2^i = 2^{n+1}-1 = 11…1B$ (n+1个1)，计算f(n)的C语言函数f1如下：

» 例如：n=0，f(0)=1；n=1，f(1)=11；n=7，f(7)=1111 1111。

```
f3(0)=1
f1(23)=16777215      f2(23)=16777215.000000
f1(24)=33554431      f2(24)=33554432.000000
f(31)=4294967295     f1(31)=-1
f(30)=2147483647     f1(30)=2147483647
f2(127)=1.#INF00
f2(126)=170141183460469230000000000000000000000.000000
```

```
int f1(unsigned n)
{
   int sum=1,power=1;
   for(unsigned i=0; i<=n-1; i++)
   {
      power *= 2;
      sum += power;
   }
   return sum;
}
```

```
float f2(unsigned n)
{
   float sum=1,power=1;
   for(unsigned i=0; i<=n-1; i++)
   {
      power *= 2;
      sum += power;
   }
   return sum;
}
```

– 将f1中的int型数据都改为float型数据，可得到计算f(n)的另一个函数f2。假设unsigned和int型数据都占32位，float型数据采用IEEE754单精度标准。请回答以下问题：

– （1）当n=0时，f1会出现死循环，为什么？若将f1中的变量i和n都定义为int型，则f1是否还会出现死循环？为什么？

– 答：

» 由于i和n都是unsigned型（32位），n=0时，n-1=0-1=11…11（32个1）=2^{32}-1，“i<=n-1”永远成立，因此出现死循环。

» 如果将i和n都定义为int型，n=0时，n-1=0-1=-1，“i<=n-1”条件不成立，直接退出循环，不会出现死循环。

– 例2.9：已知 $f(n)=\sum_{i=0}^{n}2^{i}=2^{n+1}-1=11\ldots1B$ (n+1个1)，计算f(n)的C语言函数f1如下：

» 例如：n=0，f(0)=1；n=1，f(1)=11；n=7，f(7)=1111 1111。

```
f3(0)=1
f1(23)=16777215      f2(23)=16777215.000000
f1(24)=33554431      f2(24)=33554432.000000
f(31)=4294967295     f1(31)=-1
f(30)=2147483647     f1(30)=2147483647
f2(127)=1.#INF00
f2(126)=170141183460469230000000000000000000000.000000
```

```c
int f1(unsigned n)
{
   int sum=1,power=1;
   for(unsigned i=0; i<=n-1; i++)
   {
      power *= 2;
      sum += power;
   }
   return sum;
}
```

```c
float f2(unsigned n)
{
   float sum=1,power=1;
   for(unsigned i=0; i<=n-1; i++)
   {
      power *= 2;
      sum += power;
   }
   return sum;
}
```

– 将f1中的int型数据都改为float型数据，可得到计算f(n)的另一个函数f2。假设unsigned和int型数据都占32位，float型数据采用IEEE754单精度标准。请回答以下问题：

– （2）f1(23)和f2(23)的返回值是否相等？机器数各是什么（用十六进制表示）？

– 答：

» f(23)=1111 1111 1111 1111 1111 1111B（24个1）=FF FFFFH，该数可用int型数据表示，所以f1(23)=00FF FFFFH= 16,777,215

» 该数也可以转换为float型，表示为：1.111 1111 1111 1111 1111 1111x2^{23}，采用IEEE754标准单精度浮点数表示为：

- 阶码：E=23+127=150=1001 0110
- 尾数：M=111 1111 1111 1111 1111 1111
- 符号位：S=0

» f2(23)=0 1001 0110 111 1111 1111 1111 1111 1111=4B7F FFFFH=2^{E-127}x1.M=2^{23}x($2-2^{-23}$)=$2^{24}-2^{0}$= 16,777,215

» f1(23)和f2(23)的返回值相等。f1(23)的机器数=00FF FFFFH；f2(23)的机器数=4B7F FFFFH。

– 例2.9：已知 $f(n)=\sum_{i=0}^{n} 2^i = 2^{n+1}-1 = 11\ldots1B$ (n+1个1)，计算f(n)的C语言函数f1如下：

» 例如：n=0，f(0)=1；n=1，f(1)=11；n=7，f(7)=1111 1111。

```
f3(0)=1
f1(23)=16777215      f2(23)=16777215.000000
f1(24)=33554431      f2(24)=33554432.000000
f(31)=4294967295     f1(31)=-1
f(30)=2147483647     f1(30)=2147483647
f2(127)=1.#INF00
f2(126)=170141183460469230000000000000000000000.000000
```

```c
int f1(unsigned n)
{
   int sum=1,power=1;
   for(unsigned i=0; i<=n-1; i++)
   {
      power *= 2;
      sum += power;
   }
   return sum;
}
```

```c
float f2(unsigned n)
{
   float sum=1,power=1;
   for(unsigned i=0; i<=n-1; i++)
   {
      power *= 2;
      sum += power;
   }
   return sum;
}
```

– 将f1中的int型数据都改为float型数据，可得到计算f(n)的另一个函数f2。假设unsigned和int型数据都占32位，float型数据采用IEEE754单精度标准。请回答以下问题：

– （3） f1(24)和f2(24)的返回值分别是33 554 431和33 554 432.0，为什么不相等？

– 答：

» f(24)=1 1111 1111 1111 1111 1111 1111B（25个1）=1FF FFFFH，该数可用int型数据表示，所以f1(24)=01FF FFFFH=33 554 431

» 该数也可以转换为float型，表示为：1.1111 1111 1111 1111 1111 1111x2^{24}，采用IEEE754标准单精度浮点数表示为：

- 阶码：E=24+127=151=1001 0111
- 尾数：M=000 0000 0000 0000 0000 0000（舍入处理，末位加1）
- 符号位：S=0

» f2(24)=0 1001 0111 000 0000 0000 0000 0000 0000=4B80 0000H

» f2(24)=1.Mx2^{E-127}=1.000 0000 0000 0000 0000 0000x2^{24}=2x2^{24}=2^{25}= 33 554 432

» 故f1(24)和f2(24)的返回值不相等。原因是浮点数尾数只有23位，f(24)的尾数有24位，舍入处理，末位加1。

– 例2.9：已知 $f(n)=\sum_{i=0}^{n}2^i$ = 2^{n+1}-1 = 11…1B (n+1个1)计算f(n)的C语言函数f1如下：

» 例如：n=0，f(0)=1；n=1，f(1)=11；n=7，f(7)=1111 1111。

```
f3(0)=1
f1(23)=16777215      f2(23)=16777215.000000
f1(24)=33554431      f2(24)=33554432.000000
f(31)=4294967295     f1(31)=-1
f(30)=2147483647     f1(30)=2147483647
f2(127)=1.#INF00
f2(126)=170141183460469230000000000000000000000.000000
```

```
int f1(unsigned n)
{
   int sum=1,power=1;
   for(unsigned i=0; i<=n-1; i++)
   {
      power *= 2;
      sum += power;
   }
   return sum;
}
```

```
float f2(unsigned n)
{
   float sum=1,power=1;
   for(unsigned i=0; i<=n-1; i++)
   {
      power *= 2;
      sum += power;
   }
   return sum;
}
```

– 将f1中的int型数据都改为float型数据，可得到计算f(n)的另一个函数f2。假设unsigned和int型数据都占32位，float型数据采用IEEE754单精度标准。请回答以下问题：

– （4）f(31)=2^{32}-1，而f1(31)的返回值却为-1，为什么？若要使f1(n)的返回值与f(n)相等，则最大的n是多少？

– 答：

» f(31)=11…11（32个1）=FFFF FFFFH= 4,294,967,295= 2^{32}-1

» f1(31)=FFFF FFFFH=-1（补码）

» 因为f1的返回值sum是整数（有符号数），有符号数FFFF FFFFH为-1。

» 若要使f1(n)的返回值与f(n)相等，则n的最大值为30。

» n=30时，f(30)=011…11（31个1）=7FFF FFFFH= 2,147,483,647；f1(30)=7FFF FFFFH= 2,147,483,647（补码）。

- 例2.9：已知 $f(n)=\sum_{i=0}^{n} 2^i=2^{n+1}-1$=11…1B(n+1个1)，计算f(n)的C语言函数**f1**如下：
  - » 例如：n=0，f(0)=1；n=1，f(1)=11；n=7，f(7)=1111 1111。

```
f3(0)=1
f1(23)=16777215      f2(23)=16777215.000000
f1(24)=33554431      f2(24)=33554432.000000
f(31)=4294967295     f1(31)=-1
f(30)=2147483647     f1(30)=2147483647
f2(127)=1.#INF00
f2(126)=170141183460469230000000000000000000000.000000
```

```c
int f1(unsigned n)
{
   int sum=1,power=1;
   for(unsigned i=0; i<=n-1; i++)
   {
      power *= 2;
      sum += power;
   }
   return sum;
}
```

```c
float f2(unsigned n)
{
   float sum=1,power=1;
   for(unsigned i=0; i<=n-1; i++)
   {
      power *= 2;
      sum += power;
   }
   return sum;
}
```

- 将f1中的int型数据都改为float型数据，可得到计算f(n)的另一个函数**f2**。假设unsigned和int型数据都占32位，float型数据采用IEEE754单精度标准。请回答以下问题：

- （5）f2(127)的机器数为7F80 0000H，对应的值是什么？若要使f2(n)的结果不溢出，则最大的n是多少？

- 答：
  - » 浮点数=7F80 0000H=0111 1111 1000 0000 0000 0000 0000 0000，E=1111 1111=255，M=0，对应值为+∞（见教材表2.6）。
  - » 当n=126时，f(126)=1…1(127个1)=$2^{127}-1=(2-2^{-126})\times 2^{126}$=1.1…1(126个1)x$2^{126}$，对应的阶码为E=127+126=253，尾数部分舍入后阶码加1，最终阶码E=**254**，是单精度浮点数的最大阶码。故要使f2(n)的结果不溢出，最大的n是**126**。

2.3 非数值数据的表示

2.3.1 字符表示
2.3.2 汉字编码

- **2.3.1 字符表示**
  - ASCII码：American Standard Code for Information Interchange，美国信息交换标准代码，用7位二进制数表示128个字符。见教材表2.14。
  - 扩展ASCII码：在标准ASCII码基础上，增加128个字符，共256个字符，用8位二进制数表示；其中前128个字符与标准ASCII码相同，后128个字符为扩展的ASCII吗。
  - MSB：Most Significant Bit，最高有效位，如85H=1000 0101B的最高有效位为1。
  - LSB：Least Significant Bit，最低有效位，如44H=0100 0100B的最低有效位为0。

表2.14　标准ASCII码（见教材）

十进制	二进制	符号	十进制	二进制	符号	十进制	二进制	符号	十进制	二进制	符号
0	0000 0000	NUL	32	0010 0000	[空格]	64	0100 0000	@	96	0110 0000	`
1	0000 0001	SOH	33	0010 0001	!	65	0100 0001	A	97	0110 0001	a
2	0000 0010	STX	34	0010 0010	"	66	0100 0010	B	98	0110 0010	b
3	0000 0011	ETX	35	0010 0011	#	67	0100 0011	C	99	0110 0011	c
4	0000 0100	EOT	36	0010 0100	$	68	0100 0100	D	100	0110 0100	d
5	0000 0101	ENQ	37	0010 0101	%	69	0100 0101	E	101	0110 0101	e
6	0000 0110	ACK	38	0010 0110	&	70	0100 0110	F	102	0110 0110	f
7	0000 0111	BEL	39	0010 0111	'	71	0100 0111	G	103	0110 0111	g
8	0000 1000	BS	40	0010 1000	(	72	0100 1000	H	104	0110 1000	h
9	0000 1001	HT	41	0010 1001	)	73	0100 1001	I	105	0110 1001	i
10	0000 1010	LF	42	0010 1010	*	74	0100 1010	J	106	0110 1010	j
11	0000 1011	VT	43	0010 1011	+	75	0100 1011	K	107	0110 1011	k
12	0000 1100	FF	44	0010 1100	,	76	0100 1100	L	108	0110 1100	l
13	0000 1101	CR	45	0010 1101	-	77	0100 1101	M	109	0110 1101	m
14	0000 1110	SO	46	0010 1110	.	78	0100 1110	N	110	0110 1110	n
15	0000 1111	SI	47	0010 1111	/	79	0100 1111	O	111	0110 1111	o
16	0001 0000	DLE	48	0011 0000	0	80	0101 0000	P	112	0111 0000	p
17	0001 0001	DC1	49	0011 0001	1	81	0101 0001	Q	113	0111 0001	q
18	0001 0010	DC2	50	0011 0010	2	82	0101 0010	R	114	0111 0010	r
19	0001 0011	DC3	51	0011 0011	3	83	0101 0011	S	115	0111 0011	s
20	0001 0100	DC4	52	0011 0100	4	84	0101 0100	T	116	0111 0100	t
21	0001 0101	NAK	53	0011 0101	5	85	0101 0101	U	117	0111 0101	u
22	0001 0110	SYN	54	0011 0110	6	86	0101 0110	V	118	0111 0110	v
23	0001 0111	ETB	55	0011 0111	7	87	0101 0111	W	119	0111 0111	w
24	0001 1000	CAN	56	0011 1000	8	88	0101 1000	X	120	0111 1000	x
25	0001 1001	EM	57	0011 1001	9	89	0101 1001	Y	121	0111 1001	y
26	0001 1010	SUB	58	0011 1010	:	90	0101 1010	Z	122	0111 1010	z
27	0001 1011	ESC	59	0011 1011	;	91	0101 1011	[	123	0111 1011	{
28	0001 1100	FS	60	0011 1100	<	92	0101 1100	\	124	0111 1100	|
29	0001 1101	GS	61	0011 1101	=	93	0101 1101	]	125	0111 1101	}
30	0001 1110	RS	62	0011 1110	>	94	0101 1110	^	126	0111 1110	~
31	0001 1111	US	63	0011 1111	?	95	0101 1111	_	127	0111 1111	DEL

扩展ASCII码（扩展部分）

ASCII扩展表

(American Standard Code for Information Interchange 美国标准信息交换代码)

高四位		1000		1001		1010		1011		1100		1101		1110		1111	
		8		9		A		B		C		D		E		F	
低四位		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	十进制	字符
0000	0	128	Ç	144	É	160	á	176	░	192	└	208	╨	224	α	240	≡
0001	1	129	ü	145	æ	161	í	177	▒	193	┴	209	╤	225	ß	241	±
0010	2	130	é	146	Æ	162	ó	178	▓	194	┬	210	╥	226	Γ	242	≥
0011	3	131	â	147	ô	163	ú	179	│	195	├	211	╙	227	π	243	≤
0100	4	132	ä	148	ö	164	ñ	180	┤	196	─	212	╘	228	Σ	244	⌠
0101	5	133	à	149	ò	165	Ñ	181	╡	197	┼	213	╒	229	σ	245	⌡
0110	6	134	å	150	û	166	ª	182	╢	198	╞	214	╓	230	µ	246	÷
0111	7	135	ç	151	ù	167	º	183	╖	199	╟	215	╫	231	τ	247	≈
1000	8	136	ê	152	ÿ	168	¿	184	╕	200	╚	216	╪	232	Φ	248	°
1001	9	137	ë	153	Ö	169	⌐	185	╣	201	╔	217	┘	233	Θ	249	•
1010	A	138	è	154	Ü	170	¬	186	║	202	╩	218	┌	234	Ω	250	·
1011	B	139	ï	155	¢	171	½	187	╗	203	╦	219	█	235	δ	251	√
1100	C	140	î	156	£	172	¼	188	╝	204	╠	220	▄	236	∞	252	^{n}
1101	D	141	ì	157	¥	173	¡	189	╜	205	═	221	▌	237	φ	253	^{2}
1110	E	142	Ä	158	Pts	174	«	190	╛	206	╬	222	▐	238	∈	254	■
1111	F	143	Å	159	ƒ	175	»	191	┐	207	╧	223	▀	239	∩	255	ÿ

注：表中的ASCII字符可以用“Alt + 小键盘上的数字键 ”方法输入。 制作:MHL QQ:1208980380 2013/08/08

• 2.3.2 汉字编码

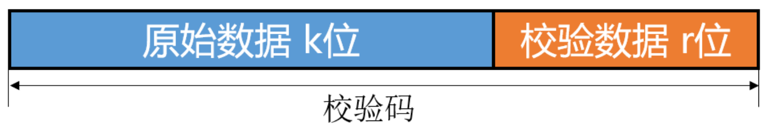
- 国标码（GB2312编码）：
  - 用2个字节（16位）表示1个汉字，每个字节的MSB均为1，实际上1个汉字用14位二进制数表示。
  - 国标码包含7445个字符：6763为常用汉字，682为全角非汉字字符。
- 区位码（国标码的另一种表示形式）：
  - 94行（区，区号）、94列（位，位号）
  - 区位码 + A0A0H = 国标码（GB2312）
- GBK标准（扩展之后的国标码，GB表示国标、K表示扩展）：
  - 不要求每个字节的MSB必须为1，增加了20,000个新的汉字和符号。
- GB18030标准：
  - 4字节的汉字编码，支持少数民族文字。
- UTF编码（Unicode Transformation Format）：
  - UTF-8、UTF-16、UTF-32
- Unicode编码：统一码
- Big5：又称为大五码或五大码
  - 是使用繁体中文（正体中文）社区中最常用的电脑汉字字符集标准，共收录13,060个汉字。

- 1、汉字处理流程
  - 汉字输入码：也称为外码，就是使用英文键盘输入汉字时的编码。
  - 汉字机内码：是计算机内部存储、处理加工和传输汉字时所用的统一编码（如国标码、区位码、Unicode编码等）。
- 2、汉字输入码
  - 流水码：如国标码、区位码
  - 音码：如拼音码（全拼、简拼、双拼等）
  - 形码：如五笔字型码
  - 音形码：如自然码、钱码
- 3、汉字字形码
  - 字形码是汉字的输出码，也称字型码。
  - 汉字字形点阵：16x16、24x24、32x32、48x48；1个32x32点阵的汉字字形码需要4Bx32=128B（128字节）的存储空间（见教材图2.15）。
  - 汉字库：存放汉字字形码的字库。

2.4 数据信息的校验

- 计算机在对数据进行处理、传输和存储过程中难免出现错误。校验码是具有发现错误或纠正错误能力的数据编码。

- 校验码（k+r位）= 原始数据（k位）+ 校验数据（r位）



- 其中校验数据是按照某种规则通过编码电路进行编码的。当校验码在传输或存储过程中出现错误时，会破坏预定的规则，通过解码电路可以发现或纠正错误。

• 2.4.1 码距与校验

– 码距（又称海明距离）：

- 两个编码对应二进制位不同的个数称为码距。
- 如10101和00110，第1、4、5位不同，则码距为3。
- 一个有效编码集中，任意两个码字的最小码距称为该编码集的码距。
- 校验码的目的就是扩大码距，从而通过编码规则来识别错误代码。
- 码距越大，抗干扰能力、纠错能力越强；但是，数据冗余越大，编码效率越低。

– 例2.10：现有两种编码体系，分别分析它们各自的码距：

- （1）设用4位二进制数表示16种状态：0000～1111
- （2）4位二进制数可表示8种状态：0000、0011、0101、0110、1001、1010、1100、1111

– 解：

- 第（1）种编码的码距为1（例如0000和0001，只有1位不同），任何一个合法编码发生一位错误时，就会变成另外一位合法编码，因此这种编码不具备检测错误的能力。
- 第（2）种编码的最小码距为2（例如0000和00011，有2位不同），任何一个合法编码如果发生一位错误时（如0000变成1000），就会变成无效编码，因此这种编码可以识别一位错误。但是发生二位错误时（如0000变成0011），又会变成另一个合法的编码，因此该编码对两位错误无法检测。

– 码距（d）与校验码的检错（e）和纠错（t）能力的关系（见教材表2.16）：

序号	码距	检错、纠错能力
1	d≥e+1	可检测e个错误
2	d≥2t+1	可纠正t个错误
3	d≥e+t+1 && e≥t	可检测e个错误并纠正t个错误

– 不同码距的检错、纠错能力（见教材表2.17）：

码距(d)	检错(e)	纠错(t)	检错(e)且纠错(t)	说明
1	0	0	0, 0	d=1,e=0,t=0 d≥e+1 (1≥0+1) d≥2t+1 (1≥2*0+1) d=1,e=0,t=0 d≥e+t+1 && e≥t (1≥0+0+1 && 0≥0)
2	1	0	1, 0	d=2,e=1,t=0 d≥e+1 (2≥1+1) d≥2t+1 (2≥2*0+1) d=2,e=1,t=0 d≥e+t+1 && e≥t (2≥1+0+1 && 1≥0)
3	2	1	1, 1	d=3,e=2,t=1 d≥e+1 (3≥2+1) d≥2t+1 (3≥2*1+1) d=3,e=1,t=1 d≥e+t+1 && e≥t (3≥1+1+1 && 1≥1)
4	3	1	2, 1	d=4,e=3,t=1 d≥e+1 (4≥3+1) d≥2t+1 (4≥2*1+1) d=4,e=2,t=1 d≥e+t+1 && e≥t (4≥2+1+1 && 2≥1)
5	4	2	2, 1	d=5,e=4,t=2 d≥e+1 (5≥4+1) d≥2t+1 (5≥2*2+1) d=5,e=2,t=1 d≥e+t+1 && e≥t (5≥2+1+1 && 2≥1)
6	5	2	3, 2	d=6,e=5,t=2 d≥e+1 (6≥5+1) d≥2t+1 (6≥2*2+1) d=6,e=3,t=2 d≥e+t+1 && e≥t (6≥3+2+1 && 3≥2)
7	6	3	3, 3	d=7,e=6,t=3 d≥e+1 (7≥6+1) d≥2t+1 (7≥2*3+1) d=7,e=3,t=3 d≥e+t+1 && e≥t (7≥3+3+1 && 3≥3)

- **2.4.2 奇偶校验**
  - 1、简单奇偶校验
    - 奇校验：校验码 = 原始数据 + 1位校验数据，校验码中1的个数为奇数个。
      - 原始数据=000 0000，校验数据=1，校验码=0000 0001
      - 原始数据=111 1111，校验数据=0，校验码=1111 1110
      - 原始数据=101 1001，校验数据=1，校验码=1011 0011
    - 偶校验：校验码 = 原始数据 + 1位校验数据，校验码中1的个数为偶数个。
      - 原始数据=000 0000，校验数据=0，校验码=0000 0000
      - 原始数据=111 1111，校验数据=1，校验码=1111 1111
      - 原始数据=101 1001，校验数据=0，校验码=1011 0010

异或：$Y=A\oplus B$

A=0,B=0	Y=0
A=0,B=1	Y=1
A=1,B=0	Y=1
A=1,B=1	Y=0

- 设原始数据为：$D = D_1D_2\ldots D_n$
- 奇校验位P = 非$(D_1\oplus D_2\oplus\ldots\oplus D_n)$
- 偶校验位$P = D_1\oplus D_2\oplus\ldots\oplus D_n$

- 设生成的校验码为：$D_1D_2\ldots D_nP$，接收到的校验码为：$D'_1D'_2\ldots D'_nP'$
- 奇校验检错位G = 非$(D'_1\oplus D'_2\oplus\ldots\oplus D'_n\oplus P')$；
- 偶校验检错位$G = D'_1\oplus D'_2\oplus\ldots\oplus D'_n\oplus P'$

- 若G=0，表示没有奇数位错（无错，或者是2位错，或者是4位错……）；若G=1，表示有奇数位错（1位错，或者是3位错，或者是5位错……）。
  - 例如，奇校验：
    - 原始数据D=000 0000，奇校验位P=1，校验码=0000 0001
    - 接收到的校验码=0000 0001，无错，奇校验检错位G=0
    - 接收到的校验码=1000 0001，1位错，奇校验检错位G=1
    - 接收到的校验码=1100 0001，2位错，奇校验检错位G=0
    - 接收到的校验码=1110 0001，3位错，奇校验检错位G=1
  - 例如，偶校验：
    - 原始数据D=000 0000，偶校验位P=0，校验码=0000 0000
    - 接收到的校验码=0000 0000，无错，偶校验检错位G=0
    - 接收到的校验码=1000 0000，1位错，偶校验检错位G=1
    - 接收到的校验码=1100 0000，2位错，偶校验检错位G=0
    - 接收到的校验码=1110 0000，3位错，偶校验检错位G=1

- 简单奇偶校验只能发现1位错，但不能纠正错误。

– 2、交叉奇偶校验

- 简单奇偶校验因为只有1位校验位，因此只能发现1位出错，并且不能纠错（即不知道是哪一位出错）。

- 多重奇偶校验：将原始数据分成若干个校验组，每个数据位至少位于两个或两个以上的校验组，当某个数据位出错时，能在多个检错位中被指出（能够改变多个检错位），从而可以知道是哪一位数据位出错。交叉奇偶校验是多重奇偶校验最典型的例子。

- 假设有4个原始数据R_3、R_2、R_1、R_0，每个都是7位，R_3=1010110、R_2=1110110、R_1=0010001、R_0=1100100。对这4个数据进行交叉偶校验，具体见教材表2.20：

- 其中，R_3、R_2、R_1、R_0的每一行产生一个偶校验位P_r（共4个），每一列产生一个偶校验位P_c（共7个）；此外，还有一个公共偶校验位（右下角的那一位）；共12个校验位。

表2.20　交叉偶校验（见教材）

	C_6	C_5	C_4	C_3	C_2	C_1	C_0	P_r
R_3	1	0	1	0	1	1	0	0
R_2	1	1	1	0	1	1	0	1
R_1	0	0	1	0	0	0	1	0
R_0	1	1	0	0	1	0	0	1
P_c	1	0	1	0	1	0	1	0

- 假设该4个原始数据（R_3、R_2、R_1、R_0）和12个校验位，经过传输或存储后，出现1位错，例如R_3的最高位出错，R_3=0010110。接收到的交叉偶校验码为：

接收到的交叉偶校验（1位出错）

	C_6	C_5	C_4	C_3	C_2	C_1	C_0	P_r	G_r
R_3	0	0	1	0	1	1	0	0	1
R_2	1	1	1	0	1	1	0	1	0
R_1	0	0	1	0	0	0	1	0	0
R_0	1	1	0	0	1	0	0	1	0
P_c	1	0	1	0	1	0	1	0	0
G_c	1	0	0	0	0	0	0	0	1

- 计算行检错码G_r（偶校验）和列检错码G_c（偶校验），根据G_r和G_c可以知道是第1行（R_3）、第1列（C_6）出错了，只需要将该位取反，即可得到正确的数据。

- 因此，交叉奇偶校验可以发现并纠正1位错。

- 假设该4个原始数据（R_3、R_2、R_1、R_0）和12个校验位，经过传输或存储后，出现2位错，例如R_3的最高位和次高位出错，R_3=0110110。接收到的交叉偶校验码为：

接收到的交叉偶校验（2位出错）

	C_6	C_5	C_4	C_3	C_2	C_1	C_0	P_r	G_r
R_3	0	1	1	0	1	1	0	0	0
R_2	1	1	1	0	1	1	0	1	0
R_1	0	0	1	0	0	0	1	0	0
R_0	1	1	0	0	1	0	0	1	0
P_c	1	0	1	0	1	0	1	0	0
G_c	1	1	0	0	0	0	0	0	0

- 计算行检错码G_r（偶校验）和列检错码G_c（偶校验），根据G_c可以知道是第1列（C_6）和第2列（C_5）出错了，即是2位出错；但是因为G_r中没有1，因此不能知道是哪一行（哪个数据）的第1列和第2列出错。

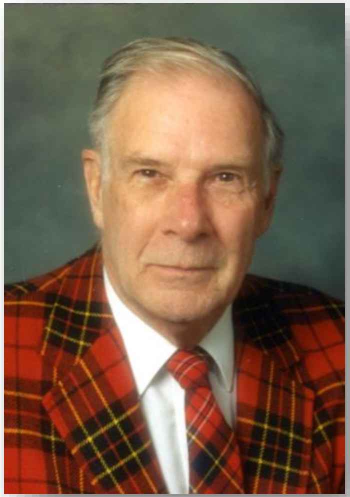
- 因此，交叉奇偶校验可以发现2位错，但不能纠正2位错。

• 2.4.3 海明校验

– 1950年，理查德·海明（Richard Hamming）提出了海明码，海明码本质上是一种多重奇偶校验码，它是一种既能检错也能纠错的校验码（Error-Correcting Codes，ECC码）。

– 能纠正一位错误的海明码也称为SEC码（Single-bit Error Correction），其最小码距是3。

– 1、校验位的位数

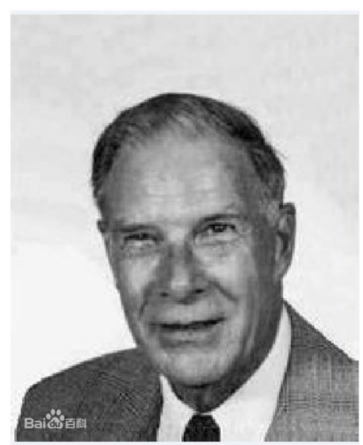


- 假设原始数据为：$D_k…D_2D_1$，共k位
- 校验码为：$P_r…P_2P_1$，共r位
- 则海明码为：$H_n…H_2H_1$，共n位，n=k+r，也称(n,k)码
- n、k、r应满足如下关系：$n = k+r \le 2^r-1$
  - k=1，r=2（$3 \le 2^2-1$）
  - k=2～4，r=3（$5 \le 2^3-1$；$6 \le 2^3-1$；$7 \le 2^3-1$）
  - k=5～11，r=4（$9 \le 2^4-1$；$10 \le 2^4-1$；$11 \le 2^4-1$；$12 \le 2^4-1$；$13 \le 2^4-1$；$14 \le 2^4-1$；$15 \le 2^4-1$）
  - …………
- 编码效率 = k/(k+r)；例如：k=8、r=4，k/(k+r)=8/12=66.7%

理查德·海明

- 理查德·卫斯里·海明（**Richard Wesley Hamming**），**1915**年**2**月**11**日至**1998**年**1**月**7**日，美国数学家，主要贡献在计算机科学和电讯。

- **1937**年芝加哥大学学士学位毕业，**1939**年内布拉斯加大学硕士学位毕业，**1942**年伊利诺伊大学香槟分校博士学位毕业，博士论文为《一些线性微分方程边界值理论上的问题》（**Some Problems in the Boundary Value Theory of Linear Differential Equations**）。二战期间在路易斯维尔大学当教授，**1945**年参加曼哈顿计划，负责编写电脑程式，计算物理学家所提供方程的解。该程式是判断引爆核弹会否燃烧大气层，结果是不会，于是核弹便开始试验。**1946**至**76**年在贝尔实验室工作。他曾和约翰·怀尔德·杜奇、克劳德·艾尔伍德·香农合作。**1956**年他参与了**IBM 650**的编程语言发展工作。**1976**年**7**月**23**日起在海军研究院当兼任教授，**1997**年成为名誉教授。他是美国电脑协会（**ACM**）的创立人之一，曾任该组织的主席。



– 2、编码分组规则

- 问题1：r位校验码$P_r…P_2P_1$位于海明码$H_n…H_2H_1$的什么位置？
  - 假设：原始数据=$D_4D_3D_2D_1$，校验码=$P_3P_2P_1$；即：k=4，r=3，n=k+r=7
  - 此时海明码为：$H_n…H_2H_1 = D_4D_3D_2P_3D_1P_2P_1$；也称(7,4)码
  - 校验码$P_r…P_2P_1$位于海明码的第1、2、4、8、16、……位（从右往左看）

- 问题2：如何得到校验码$P_r…P_2P_1$的值？
  - 对于(7,4)码有（见教材表2.22和图2.18）：
  - $P_1 = D_1 \oplus D_2 \oplus D_4$
  - $P_2 = D_1 \oplus D_3 \oplus D_4$
  - $P_3 = D_2 \oplus D_3 \oplus D_4$

- 问题3：如何得到检错码$G_r…G_2G_1$的值？
  - 对于(7,4)码有（见教材表2.22和图2.18）：
  - 假设海明码经过传输或存储后变成：$D'_4D'_3D'_2P'_3D'_1P'_2P'_1$；则检错码为：
  - $G_1 = P'_1 \oplus D'_1 \oplus D'_2 \oplus D'_4$
  - $G_2 = P'_2 \oplus D'_1 \oplus D'_3 \oplus D'_4$
  - $G_3 = P'_3 \oplus D'_2 \oplus D'_3 \oplus D'_4$

- 3、检错与纠错

  - 当检错码=$G_r \ldots G_2G_1=0$时，表示海明码正确。
  - 当检错码=$G_r \ldots G_2G_1 \neq 0$时，表示海明码出错，根据检错码的值可以确定海明码是哪一位出错。
  - 对于(7,4)码有：
    - $G_3G_2G_1=000$，无错
    - $G_3G_2G_1=001$，H_1出错
    - $G_3G_2G_1=010$，H_2出错
    - $G_3G_2G_1=011$，H_3出错
    - $G_3G_2G_1=100$，H_4出错
    - $G_3G_2G_1=101$，H_5出错
    - $G_3G_2G_1=110$，H_6出错
    - $G_3G_2G_1=111$，H_7出错

– （补充）例1：设原始数据=1010，求该数据的海明码。假设在传输或存储过程中，该海明码出现1位错误，请验证海明码能够发现并纠正1位错。

– 解：

- 原始数据=$D_4D_3D_2D_1$=1010，k=4，则r=3

- 校验码$P_3P_2P_1$如下：
  – $P_1 = D_1 \oplus D_2 \oplus D_4 = 0 \oplus 1 \oplus 1 = 0$
  – $P_2 = D_1 \oplus D_3 \oplus D_4 = 0 \oplus 0 \oplus 1 = 1$
  – $P_3 = D_2 \oplus D_3 \oplus D_4 = 1 \oplus 0 \oplus 1 = 0$

- 海明码$H_7 \ldots H_2H_1 = D_4D_3D_2P_3D_1P_2P_1$=1010010

- 假设在传输或存储过程中，海明码的第7位（H_7）出错，传输或存储后的海明码=$H'_7 \ldots H'_2H'_1 = D'_4D'_3D'_2P'_3D'_1P'_2P'_1$=0010010

- 检错码为：
  – $G_1 = P'_1 \oplus D'_1 \oplus D'_2 \oplus D'_4 = 0 \oplus 0 \oplus 1 \oplus 0 = 1$
  – $G_2 = P'_2 \oplus D'_1 \oplus D'_3 \oplus D'_4 = 1 \oplus 0 \oplus 0 \oplus 0 = 1$
  – $G_3 = P'_3 \oplus D'_2 \oplus D'_3 \oplus D'_4 = 0 \oplus 1 \oplus 0 \oplus 0 = 1$

同学们可以验证其它位出错时（如第1位H_1出错），海明码发现错误的能力！

- $G_3G_2G_1$=111，表示H_7出错，只需要将该位取反，即可得到正确的海明码，说明海明码能够发现并纠正1位错。

- （补充）例2：假设接收到的海明码为1010110，且假设接收到的海明码最多出现1位错误。请问该海明码对应的原始数据是多少？

- 解：
  - 接收到的海明码$H'_7…H'_2H'_1=D'_4D'_3D'_2P'_3D'_1P'_2P'_1=1010110$
  - 检错码为：
    - $G_1 = P'_1 \oplus D'_1 \oplus D'_2 \oplus D'_4 = 0 \oplus 1 \oplus 1 \oplus 1=1$
    - $G_2 = P'_2 \oplus D'_1 \oplus D'_3 \oplus D'_4 = 1 \oplus 1 \oplus 0 \oplus 1=1$
    - $G_3 = P'_3 \oplus D'_2 \oplus D'_3 \oplus D'_4 = 0 \oplus 1 \oplus 0 \oplus 1=0$
  - $G_3G_2G_1=011$，表示H'_3出错，只需要将该位取反，即可得到正确的海明码
  - 正确的海明码$=H_7…H_2H_1 = D_4D_3D_2P_3D_1P_2P_1=1010010$
  - 原始数据$=D_4D_3D_2D_1=1010$

– 4、扩展海明码

- 扩展海明码：**SECDED码**（Single Error Correction Double Error Detection），其最小码距为4，可以同时检测两位错，并能纠正一位错。

- 扩展海明码是在普通的海明码基础上增加一个**总偶校验码P_{all}**，用于区分一位错和两位错：$P_{all} = (D_1 \oplus D_2 \oplus \ldots\ldots \oplus D_k) \oplus (P_1 \oplus P_2 \oplus \ldots\ldots \oplus P_r)$

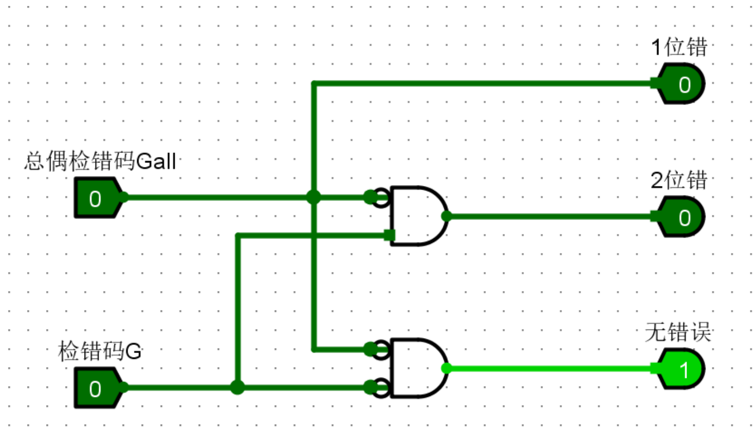
- 即扩展的海明码为：$D_k \ldots D_2 D_1$、$P_r \ldots P_2 P_1$和 P_{all}，共k+r+1=**n+1位**

- 假设扩展的海明码经过传输或存储后，变成$D'_k \ldots D'_2 D'_1$、$P'_r \ldots P'_2 P'_1$和 P'_{all}

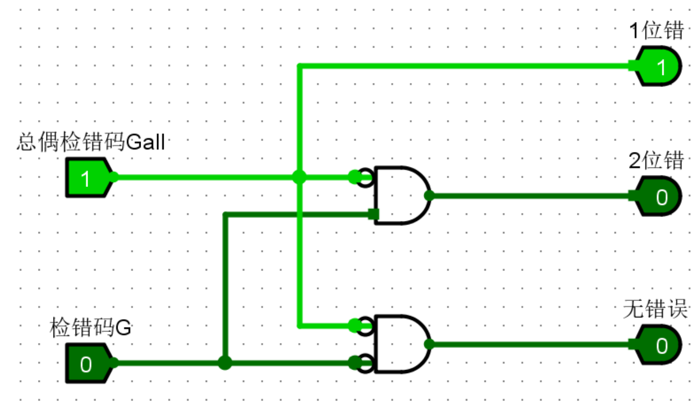
- 则**总偶检错码G_{all}**为：$G_{all} = P'_{all} \oplus (D'_1 \oplus D'_2 \oplus \ldots\ldots \oplus D'_k) \oplus (P'_1 \oplus P'_2 \oplus \ldots\ldots \oplus P'_r)$

- 如果$G_{all}=0$，$G=G_r \ldots G_2 G_1=0$，表示没有错误
- 如果$G_{all}=1$，$G=G_r \ldots G_2 G_1=0$，表示出现1位错，为P_{all}发生错，不需要纠错
- 如果$G_{all}=1$，$G=G_r \ldots G_2 G_1 \neq 0$，表示出现1位错，可以根据检错码G的值纠错
- 如果$G_{all}=0$，$G=G_r \ldots G_2 G_1 \neq 0$，表示出现2位错，无法纠错

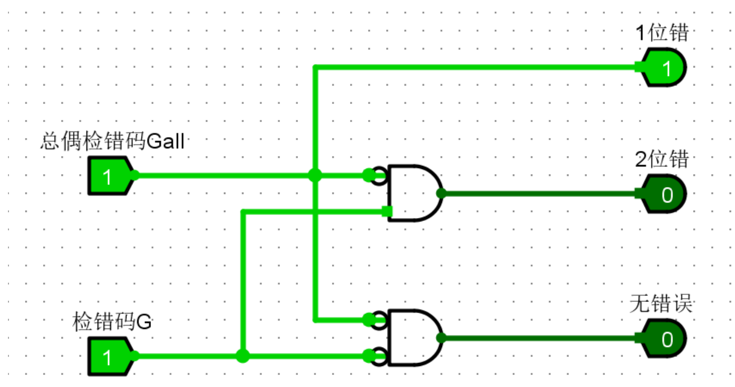
- 服务器中常用的ECC校验内存就采用了SECDED码，它可以检测内存条的两位错并纠正一位错。例如数据宽度为64位的内存，引入7位海明校验位（k=64，r=7；$n = k+r \le 2^r-1$，$71 \le 2^7-1$）以及1位总校验位，实际的位数为64+8=72位，因此16GB的ECC内存实际容量为18GB（16GBx72/64=18GB）。



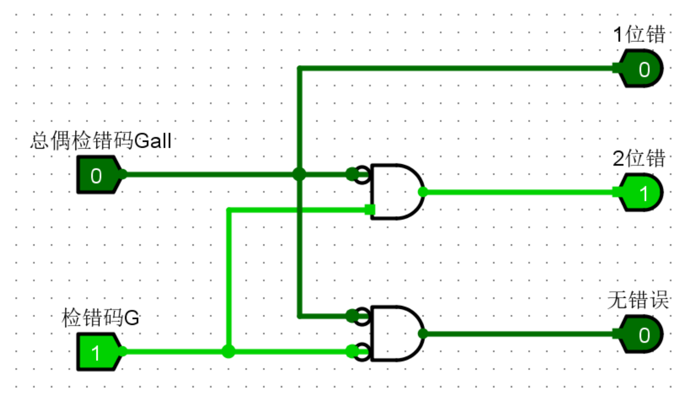
无错误



1位错（不需要纠错）

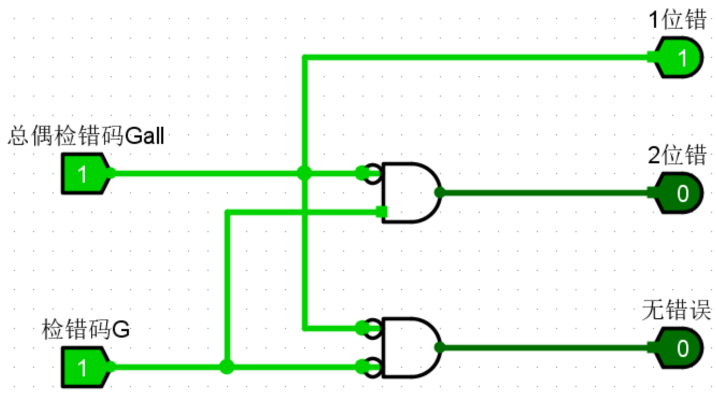


1位错（可以纠错）

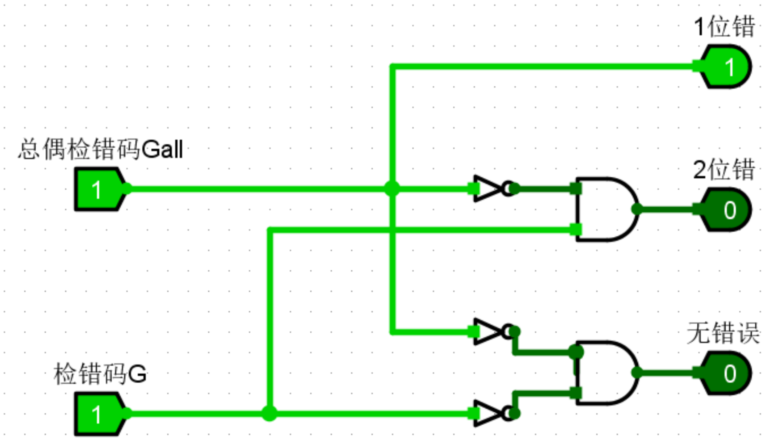


2位错（无法纠错）

图2.19　扩展海明码检错附加电路（见教材）



扩展海明码检错附加电路（教材图2.19）



扩展海明码检错附加电路（与上面等价的电路）

- （补充）例3：设原始数据=1010，求该数据的扩展海明码。假设在传输或存储过程中，该扩展海明码分别出现：无错、1位错误、2位错误，请验证扩展海明码能够发现2位错误并纠正1位错。
- 解：
  - 原始数据=$D_4D_3D_2D_1$=1010，k=4，则r=3
  - 校验码$P_3P_2P_1$如下：
    - $P_1 = D_1 \oplus D_2 \oplus D_4 = 0 \oplus 1 \oplus 1 = 0$
    - $P_2 = D_1 \oplus D_3 \oplus D_4 = 0 \oplus 0 \oplus 1 = 1$
    - $P_3 = D_2 \oplus D_3 \oplus D_4 = 1 \oplus 0 \oplus 1 = 0$
  - 海明码$H_7 \ldots H_2H_1 = D_4D_3D_2P_3D_1P_2P_1$=1010010
  - 总偶校验码$P_{all} = (D_1 \oplus D_2 \oplus D_3 \oplus D_4) \oplus (P_1 \oplus P_2 \oplus P_3)$=1
  - 扩展海明码= $H_7 \ldots H_2H_1P_{all}$ =10100101

  - （1）假设在传输或存储过程中，扩展海明码没有出现错误，传输或存储后的扩展海明码 = $H'_7 \ldots H'_2H'_1P'_{all} = D'_4D'_3D'_2P'_3D'_1P'_2P'_1P'_{all}$=10100101

  - 检错码为：
    - $G_1 = P'_1 \oplus D'_1 \oplus D'_2 \oplus D'_4 = 0 \oplus 0 \oplus 1 \oplus 1 = 0$
    - $G_2 = P'_2 \oplus D'_1 \oplus D'_3 \oplus D'_4 = 1 \oplus 0 \oplus 0 \oplus 1 = 0$
    - $G_3 = P'_3 \oplus D'_2 \oplus D'_3 \oplus D'_4 = 0 \oplus 1 \oplus 0 \oplus 1 = 0$

  - 总偶检错码$G_{all} = P'_{all} \oplus (D'_1 \oplus D'_2 \oplus D'_3 \oplus D'_4) \oplus (P'_1 \oplus P'_2 \oplus P'_3) = 1 \oplus 0 \oplus 1 \oplus 0 \oplus 1 \oplus 0 \oplus 1 \oplus 0 = 0$

  - $G_{all}=0$，$G=G_3G_2G_1=000$，表示没有出错。

- （补充）例3：设原始数据=1010，求该数据的扩展海明码。假设在传输或存储过程中，该扩展海明码分别出现：无错、1位错误、2位错误，请验证扩展海明码能够发现2位错误并纠正1位错。
- 解（续）：
  - 原始数据=$D_4D_3D_2D_1$=1010，k=4，则r=3
  - 校验码$P_3P_2P_1$如下：
    - $P_1 = D_1 \oplus D_2 \oplus D_4 = 0 \oplus 1 \oplus 1 = 0$
    - $P_2 = D_1 \oplus D_3 \oplus D_4 = 0 \oplus 0 \oplus 1 = 1$
    - $P_3 = D_2 \oplus D_3 \oplus D_4 = 1 \oplus 0 \oplus 1 = 0$
  - 海明码$H_7 \ldots H_2H_1 = D_4D_3D_2P_3D_1P_2P_1$=1010010
  - 总偶校验码$P_{all} = (D_1 \oplus D_2 \oplus D_3 \oplus D_4) \oplus (P_1 \oplus P_2 \oplus P_3)$=1
  - 扩展海明码= $H_7 \ldots H_2H_1P_{all}$ =10100101
  - （2-1）假设在传输或存储过程中，扩展海明码的第8位（H_7）出错，传输或存储后的扩展海明码= $H'_7 \ldots H'_2H'_1P'_{all} = D'_4D'_3D'_2P'_3D'_1P'_2P'_1P'_{all}$=00100101
  - 检错码为：
    - $G_1 = P'_1 \oplus D'_1 \oplus D'_2 \oplus D'_4 = 0 \oplus 0 \oplus 1 \oplus 0 = 1$
    - $G_2 = P'_2 \oplus D'_1 \oplus D'_3 \oplus D'_4 = 1 \oplus 0 \oplus 0 \oplus 0 = 1$
    - $G_3 = P'_3 \oplus D'_2 \oplus D'_3 \oplus D'_4 = 0 \oplus 1 \oplus 0 \oplus 0 = 1$
  - 总偶检错码$G_{all} = P'_{all} \oplus (D'_1 \oplus D'_2 \oplus D'_3 \oplus D'_4) \oplus (P'_1 \oplus P'_2 \oplus P'_3) = 1 \oplus 0 \oplus 1 \oplus 0 \oplus 0 \oplus 0 \oplus 1 \oplus 0 = 1$
  - G_{all}=1，$G=G_3G_2G_1=111 \neq 0$，表示出现1位错，可以根据检错码G的值纠错；$G=G_3G_2G_1$=111，表示H_7出错，只需要将该位取反，即可得到正确的海明码，说明扩展海明码能够发现并纠正1位错。

- （补充）例3：设原始数据=1010，求该数据的扩展海明码。假设在传输或存储过程中，该扩展海明码分别出现：无错、1位错误、2位错误，请验证扩展海明码能够发现2位错误并纠正1位错。
- 解（续）：
  - 原始数据=$D_4D_3D_2D_1$=1010，k=4，则r=3
  - 校验码$P_3P_2P_1$如下：
    - $P_1 = D_1 \oplus D_2 \oplus D_4 = 0 \oplus 1 \oplus 1 = 0$
    - $P_2 = D_1 \oplus D_3 \oplus D_4 = 0 \oplus 0 \oplus 1 = 1$
    - $P_3 = D_2 \oplus D_3 \oplus D_4 = 1 \oplus 0 \oplus 1 = 0$
  - 海明码$H_7 \ldots H_2H_1 = D_4D_3D_2P_3D_1P_2P_1$=1010010
  - 总偶校验码$P_{all} = (D_1 \oplus D_2 \oplus D_3 \oplus D_4) \oplus (P_1 \oplus P_2 \oplus P_3)$=1
  - 扩展海明码= $H_7 \ldots H_2H_1P_{all}$ =10100101

  - （2-2）假设在传输或存储过程中，扩展海明码的第1位（P_{all}）出错，传输或存储后的扩展海明码= $H'_7 \ldots H'_2H'_1P'_{all}=D'_4D'_3D'_2P'_3D'_1P'_2P'_1P'_{all}$=10100100

  - 检错码为：
    - $G_1 = P'_1 \oplus D'_1 \oplus D'_2 \oplus D'_4 = 0 \oplus 0 \oplus 1 \oplus 1 = 0$
    - $G_2 = P'_2 \oplus D'_1 \oplus D'_3 \oplus D'_4 = 1 \oplus 0 \oplus 0 \oplus 1 = 0$
    - $G_3 = P'_3 \oplus D'_2 \oplus D'_3 \oplus D'_4 = 0 \oplus 1 \oplus 0 \oplus 1 = 0$

  - 总偶检错码$G_{all} = P'_{all} \oplus (D'_1 \oplus D'_2 \oplus D'_3 \oplus D'_4) \oplus (P'_1 \oplus P'_2 \oplus P'_3) = 0 \oplus 0 \oplus 1 \oplus 0 \oplus 1 \oplus 0 \oplus 1 \oplus 0 = 1$

  - G_{all}=1，$G=G_3G_2G_1$=000，表示出现1位错，并且是P_{all}出错，不需要纠错。

- （补充）例3：设原始数据=1010，求该数据的扩展海明码。假设在传输或存储过程中，该扩展海明码分别出现：无错、1位错误、2位错误，请验证扩展海明码能够发现2位错误并纠正1位错。
- 解（续）：
  - 原始数据=$D_4D_3D_2D_1$=1010，k=4，则r=3
  - 校验码$P_3P_2P_1$如下：
    - $P_1 = D_1 \oplus D_2 \oplus D_4 = 0 \oplus 1 \oplus 1 = 0$
    - $P_2 = D_1 \oplus D_3 \oplus D_4 = 0 \oplus 0 \oplus 1 = 1$
    - $P_3 = D_2 \oplus D_3 \oplus D_4 = 1 \oplus 0 \oplus 1 = 0$
  - 海明码$H_7 \ldots H_2H_1 = D_4D_3D_2P_3D_1P_2P_1$=1010010
  - 总偶校验码$P_{all} = (D_1 \oplus D_2 \oplus D_3 \oplus D_4) \oplus (P_1 \oplus P_2 \oplus P_3)$=1
  - 扩展海明码= $H_7 \ldots H_2H_1P_{all}$ =10100101

  - （3）假设在传输或存储过程中，扩展海明码的第8位（H_7）和第6位（H_5）出错，传输或存储后的扩展海明码= $H'_7 \ldots H'_2H'_1P'_{all} = D'_4D'_3D'_2P'_3D'_1P'_2P'_1P'_{all}$=00000101

  - 检错码为：
    - $G_1 = P'_1 \oplus D'_1 \oplus D'_2 \oplus D'_4 = 0 \oplus 0 \oplus 0 \oplus 0 = 0$
    - $G_2 = P'_2 \oplus D'_1 \oplus D'_3 \oplus D'_4 = 1 \oplus 0 \oplus 0 \oplus 0 = 1$
    - $G_3 = P'_3 \oplus D'_2 \oplus D'_3 \oplus D'_4 = 0 \oplus 1 \oplus 0 \oplus 0 = 1$

  - 总偶检错码$G_{all} = P'_{all} \oplus (D'_1 \oplus D'_2 \oplus \ldots\ldots \oplus D'_k) \oplus (P'_1 \oplus P'_2 \oplus \ldots\ldots \oplus P'_r) = 1 \oplus 0 \oplus 0 \oplus 0 \oplus 0 \oplus 0 \oplus 1 \oplus 0 = 0$

  - $G_{all}=0$，$G=G_3G_2G_1=110 \neq 0$，表示出现2位错，但是无法知道是哪2位错误，即无法纠错。

- 例2.11：设7位ASCII码= $D_7……D_2D_1$=1101010，请给出能纠正1位错的海明码方案。在假设没有3位错的前提下，尝试分析该编码能否区分1位错和2位错。

- 解：
  - 原始数据=$D_7……D_2D_1$=1101010，k=7，则r=4
  - 校验码$P_4P_3P_2P_1$如下（参见教材表2.22）：
    - $P_1 = D_1 \oplus D_2 \oplus D_4 \oplus D_5 \oplus D_7 = 0\oplus1\oplus1\oplus0\oplus1=1$
    - $P_2 = D_1 \oplus D_3 \oplus D_4 \oplus D_6 \oplus D_7 = 0\oplus0\oplus1\oplus1\oplus1=1$
    - $P_3 = D_2 \oplus D_3 \oplus D_4 = 1\oplus0\oplus1=0$
    - $P_4 = D_5 \oplus D_6 \oplus D_7 = 0\oplus1\oplus1=0$
  - 海明码$H_{11}…H_2H_1 = D_7D_6D_5P_4D_4D_3D_2P_3D_1P_2P_1$=11001010011
  - 该海明码只能发现并纠正1位错误。假设在传输或存储过程中，该海明码的第6位（H_6）出错，传输或存储后的海明码= $H'_{11}…H'_2H'_1$=11001110011
  - 检错码为：
    - $G_1 = P'_1 \oplus D'_1 \oplus D'_2 \oplus D'_4 \oplus D'_5 \oplus D'_7 = 1\oplus0\oplus1\oplus1\oplus0\oplus1 =0$
    - $G_2 = P'_2 \oplus D'_1 \oplus D'_3 \oplus D'_4 \oplus D'_6 \oplus D'_7 = 1\oplus1\oplus0\oplus1\oplus1\oplus1 =1$
    - $G_3 = P'_3 \oplus D'_2 \oplus D'_3 \oplus D'_4 = 0\oplus1\oplus1\oplus1=1$
    - $G_4 = P'_4 \oplus D'_5 \oplus D'_6 \oplus D'_7 = 0\oplus0\oplus1\oplus1=0$
  - 检错码$G=G_4G_3G_2G_1$=0110=6，表示出现1位错，且是第6位出错，只需将第6位取反即可得到正确的数据。因此，该海明码可以发现并纠正1位错。

- 例2.11：设7位ASCII码= $D_7......D_2D_1$=1101010，请给出能纠正1位错的海明码方案。在假设没有3位错的前提下，尝试分析该编码能否区分1位错和2位错。

- 解（续）：

  - 但是该海明码不能发现2位出错。

  - 假设在传输或存储过程中，该海明码的第4位（H_4）和第6位（H_6）出错，传输或存储后的扩展海明码= $H'_{11}...H'_2H'_1$=11001111011

  - 检错码为：
    - $G_1 = P'_1 \oplus D'_1 \oplus D'_2 \oplus D'_4 \oplus D'_5 \oplus D'_7 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 \oplus 1=0$
    - $G_2 = P'_2 \oplus D'_1 \oplus D'_3 \oplus D'_4 \oplus D'_6 \oplus D'_7 = 1 \oplus 1 \oplus 0 \oplus 1 \oplus 1 \oplus 1=1$
    - $G_3 = P'_3 \oplus D'_2 \oplus D'_3 \oplus D'_4 = 1 \oplus 1 \oplus 1 \oplus 1=0$
    - $G_4 = P'_4 \oplus D'_5 \oplus D'_6 \oplus D'_7 = 0 \oplus 0 \oplus 1 \oplus 1=0$

  - 检错码$G=G_4G_3G_2G_1$=0010=2，显然G的值反映不了海明码是不是2位出错。

  - 如果要能发现2位错，就需要使用扩展的海明码。

对于该例子，请同学们自行分析扩展的海明码如何能够发现2位错！

• 2.4.4 循环冗余校验

– 循环冗余校验（Cyclic Redundancy Check，CRC）是一种基于模2运算的校验码，在磁盘存储和计算机通信方面应用广泛。

– 1、模2运算

- 加减：0 ± 0 = 0； 0 ± 1 = 1； 1 ± 0 = 1； 1 ± 1 = 0

有点像异或运算

- 乘法：根据模2加法运算求部分积，不考虑进位
  – 例2.12：1101 x 101 = 111001

```
   1101
  x 101
  -----
   1101
  0000
+1101
------
 111001
```

- 除法：根据模2减法运算求部分余数，部分余数首位为1，商1；部分余数首位为0，商0
  – 例2.13：10010 ÷ 101 = 商为101、余数为11

```
         101
     _______
101 | 10010
      -101
      ----
        011
       -000
       ----
         110
        -101
        ----
          11
```

– 2、编码规则

- 原始数据= $C_{k-1}…C_1C_0$，共k位
- 校验码=$P_{r-1}…P_1P_0$，共r位
- 则CRC码= $C_{k-1}…C_1C_0P_{r-1}…P_1P_0$，共n位，n=k+r，称为(n,k)码
- n、k、r应满足如下关系：$n = k+r \le 2^r-1$（与海明码相同）

- 原始数据用多项式M(x)表示：$M(x) = C_{k-1}x^{k-1}+……+C_1x^1+ C_0x^0$
- 将M(x)左移r位，可表示成：$M(x)\cdot 2^r$，右侧空出的r位用来放置校验码
- 选择一个r+1位的生成多项式G(x)
- 用$M(x)\cdot 2^r$按模2的运算规则除以生成多项式G(x)，得到商为Q(x)，余数R(x)作为校验码
- CRC码=$M(x)\cdot 2^r+R(x)$

```
             101
11101 | 1100000
        -11101
         01010
        -00000
          10100
         -11101
           1001
```

- 例如，原始数据=110，k=3，r=4，n=7；$M(x)\cdot 2^r$=110 0000
- 选择生成多项式G(x)=11101
- $M(x)\cdot 2^r/G(x)$ = 110 0000 / 11101 = 商为101、余数为1001
- 则CRC码：110 1001

模2运算，一定有：R(x)+R(x)=0

- 因为：$M(x)\cdot 2^r=Q(x)G(x)+R(x)$
- CRC码可以表示为：$M(x)\cdot 2^r+R(x)=[Q(x)G(x)+R(x)]+R(x) = Q(x)G(x)+[R(x)+R(x)] = Q(x)G(x)$
- 因此：CRC码/G(x)=Q(x)，即CRC码一定能被生成多项式整除
- 例如，110 1001 / 11101 = 商为101、余数为0

```
             101
11101 | 1101001
        -11101
         01110
        -00000
          11101
         -11101
              0
```

– 3、CRC编码特性（教材上放在第5部分）

- CRC码可以发现1位错和2位错，并可以纠正1位错（相当于扩展海明码的功能）。

- 表2.24（见教材）：CRC(7,3)码的出错模式（可以发现1位错，并且可以纠正1位错，因为每个出错位对应一个不同的余数）。
  - 余数=0000=0，没有错误
  - 余数=0001=1，第1位出错
  - 余数=0010=2，第2位出错
  - 余数=0100=4，第3位出错
  - 余数=1000=8，第4位出错
  - 余数=1101=13，第5位出错
  - 余数=0111=7，第6位出错
  - 余数=1110=14，第7位出错

- 表2.25（见教材）：CRC(7,3)码的出错模式（可以发现2位错，但是无法纠正2位错，因为每个余数对应3个不同的2位错）。
  - 余数=0011=3，第1、2位出错，或者第3、6位出错，或者第5、7位出错
  - 余数=0101=5，第1、3位出错，或者第2、6位出错，或者第4、5位出错
  - 余数=0110=6，第1、6位出错，或者第2、3位出错，或者第4、7位出错
  - 余数=1001=9，第1、4位出错，或者第3、5位出错，或者第6、7位出错
  - 余数=1010=10，第2、4位出错，或者第3、7位出错，或者第5、6位出错
  - 余数=1100=12，第1、5位出错，或者第2、7位出错，或者第3、4位出错
  - 余数=1111=15，第1、7位出错，或者第2、5位出错，或者第4、6位出错

- 假设前面的CRC(7,3)码= 110 1001，经过传输或存储后，有1位出错（如第7位出错），变成010 1001，则CRC码除以生成多项式的余数为：010 1001 / 11101 = 商为011，余数为1110。

- 余数=1110，表示第7位出错，只需要将该位取反，即得到正确的数据。

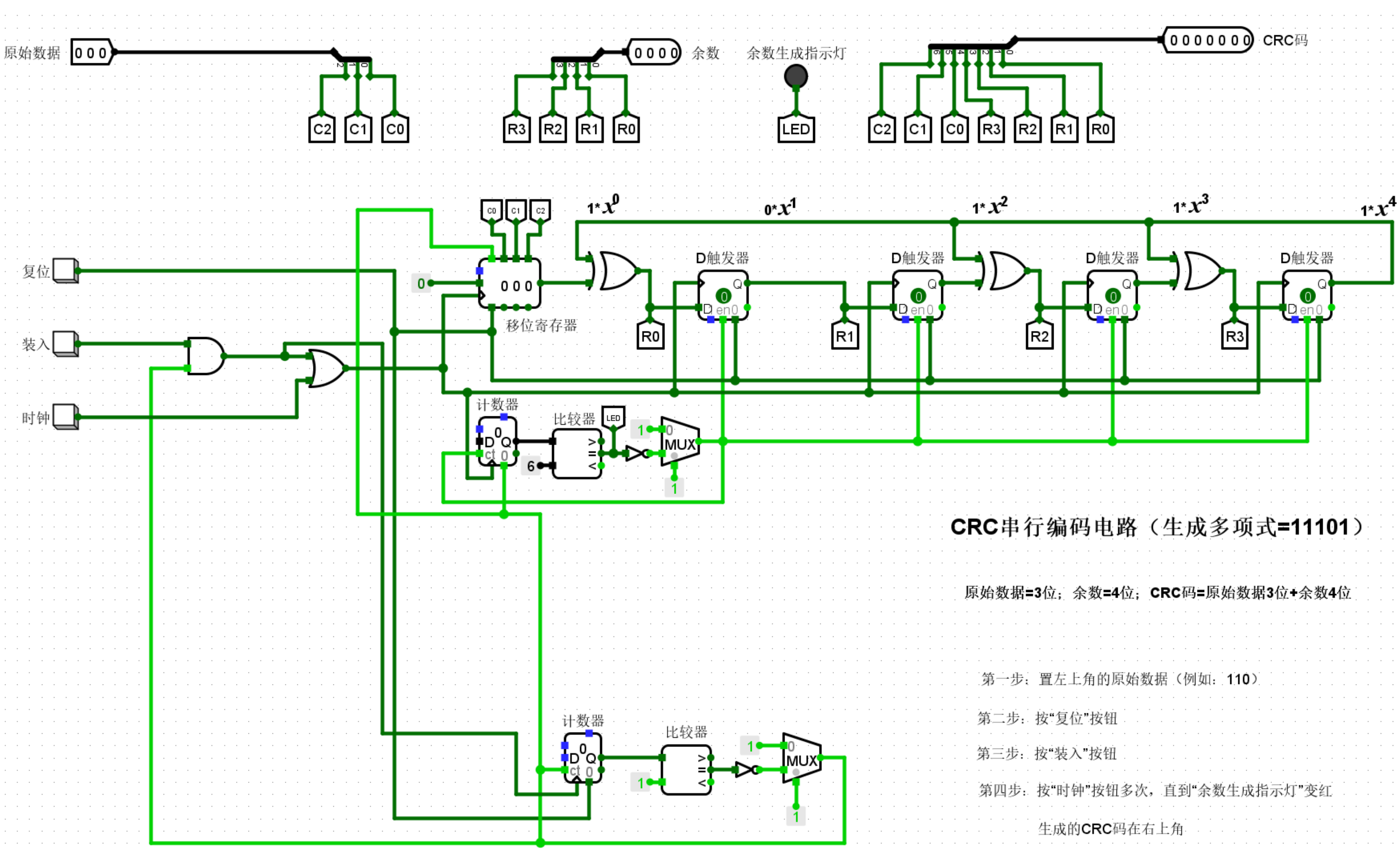
```
          011
11101 | 0101001
       -00000
        10100
       -11101
         10011
        -11101
          1110
```

- 4、CRC编、解码电路

  - 图2.20（见教材）：生成多项式为G(x)=11101时的CRC串行编、解码电路。
  - 编码：原始数据从Din端串行输入，经过n-1个时钟周期后，可以计算得到最终的余数$R_3R_2R_1R_0$；原始数据和余数拼接在一起就是CRC码。
  - 解码：接收到的CRC码从Din端串行输入，经过n-1个时钟周期后，可以计算得到最终的余数$R_3R_2R_1R_0$；根据余数的值，可以知道是哪一位出错（假设只有1位出错），或者无错。
  - 不同的生成多项式G(x)对应不同的CRC编、解码电路。

- 5、CRC编、解码流程

  - 图2.21（见教材）：CRC编、解码流程。
  - 原始数据=$a_k…a_2a_1$，左移r位后，送入CRC编码电路；即将左移后的原始数据除以生成多项式$g_r…g_1g_0$；将得到的r位余数$b_r…b_2b_1$与原始数据$a_k…a_2a_1$拼接成CRC码$a_k…a_2a_1b_r…b_2b_1$
  - CRC码经过传输或存储后，变成可能出错的CRC码$c_k…c_2c_1d_r…d_2d_1$
  - 将接收到的CRC码送CRC解码电路，即将接收到的CRC码除以生成多项式$g_r…g_1g_0$，将得到的r位余数$s_r…s_2s_1$送决策逻辑；若余数为0，表示没有错误；若余数不为0，则根据余数的值，可以确定是哪一位错（假设只有1位错），只需要将该位取反即可得到正确的数据。



CRC串行编码电路（生成多项式=11101）

原始数据=3位；余数=4位；CRC码=原始数据3位+余数4位

第一步：置左上角的原始数据（例如：110）

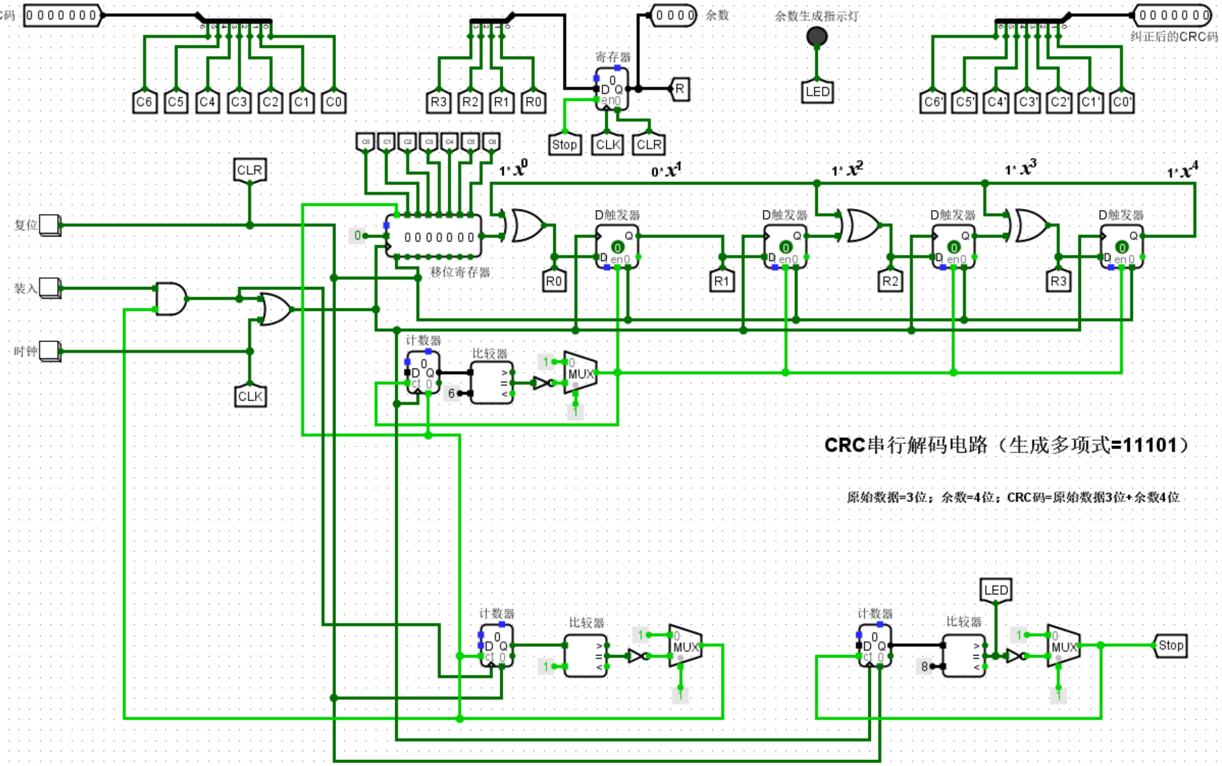
第二步：按“复位”按钮

第三步：按“装入”按钮

第四步：按“时钟”按钮多次，直到“余数生成指示灯”变红

生成的**CRC**码在右上角

利用这个电路可以验证表2.24和表2.25的结果



CRC串行解码电路（生成多项式=11101）

原始数据=3位；余数=4位；CRC码=原始数据3位+余数4位

第一步：置左上角的CRC码（如：1101001）

第二步：按"复位"按钮

第三步：按"装入"按钮

第四步：按"时钟"按钮多次，直到"余数生成指示灯"变红

此时下面的"无错"指示灯变红，表示无错

第五步：置有一位错的CRC码（如：0101001）

重复第二、三、四步

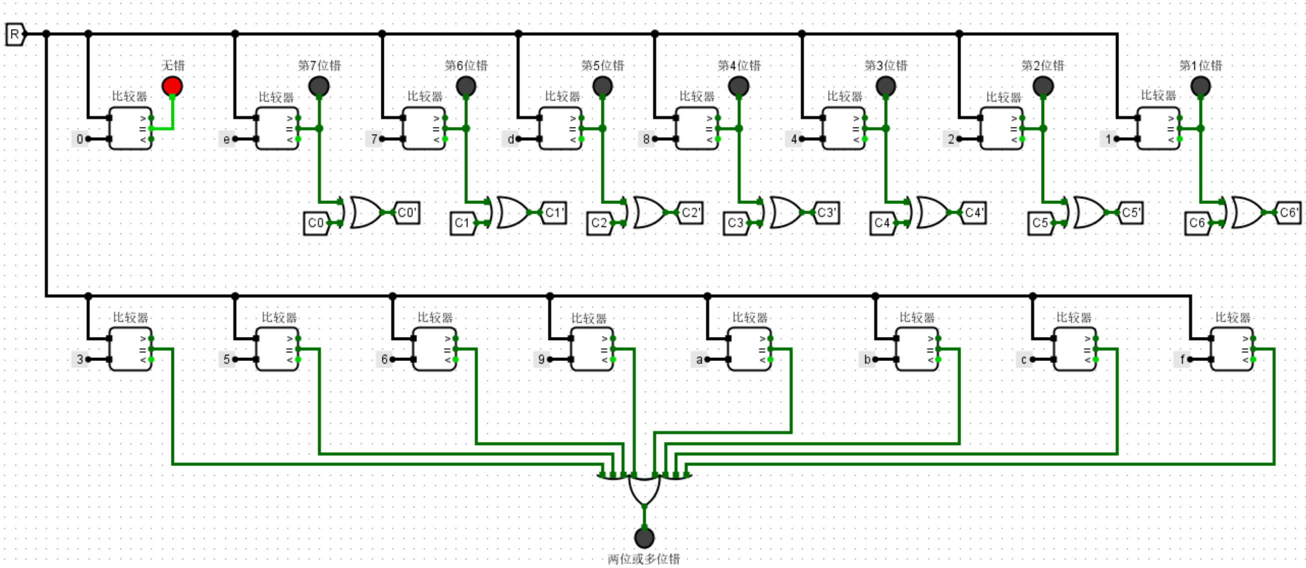
此时，下面的"第7位错"指示灯变红

纠正后的CRC码见右上角

第六步：置有两位错的CRC码（如：0001001）

重复第二、三、四步

此时，下面的"两位或多位错"指示灯变红



– 6、生成多项式

- CRC编码电路根据生成多项式得到的余数拼接成CRC码；CRC解码电路根据生成多项式得到的余数确定有没有错，或者是哪一位出错。

- 不是任何一个多项式都可以作为生成多项式，生成多项式有如下特殊要求：
  ① 生成多项式的最高位和最低位必须为1；
  ② 当CRC码任何1位发生错误时，被生成多项式模2除后，余数应不为0；
  ③ 不同位发生的错误，余数应不同；
  ④ 对余数继续做模2除法，应使余数循环（CRC：循环冗余校验）。

- 常用的生成多项式：
  ① $G(x)=x+1$
  ② $G(x)=x^3+x+1$
  ③ $G(x)=x^4+x+1$
  ④ $G(x)=x^5+x^4+x^2+1$
  ⑤ $G(x)=x^5+x^3+1$
  ⑥ …………

– 7、CRC检错性能

- 采用CRC码产生r位的校验码（$P_{r-1}…P_1P_0$），具有如下的检错能力：
  ① 所有突发长度小于等于r的突发错误（突发长度 ≤ r）；
  ② $(1-2^{-(r-1)})$比例的突发长度为r+1的突发错误（突发长度 = r+1）；
  ③ $(1-2^{-r})$比例的突发长度大于r+1的突发错误（突发长度 > r+1）；
  ④ 小于最小码距的任意位数的错误；
  ⑤ 如果生成多项式中1的个数为偶数，可以检测出所有奇数位错误。

- 如果r=16，就可以：
  ① 检测出所有突发长度小于等于16的突发错误（≤16）
  ② 检测出$(1-2^{-(r-1)})= (1-2^{-15})$=32767/32768≈99.997%突发长度等于17的突发错误（=17）
  ③ 检测出$(1-2^{-r})= (1-2^{-16})$=65535/65536≈99.998%突发长度大于17的突发错误（>17）

- 可见CRC码的检错能力很强，CRC码检错能力强、开销小、易于用编码器及检测电路实现。

- 在数据存储和通信领域，CRC码无处不在：
  ① 通信协议X.25的FCS（检错序列）采用CRC-CCITT
  ② WinRAR、ARJ、LHA等压缩工具采用CRC32
  ③ 磁盘驱动器的读写采用CRC16
  ④ 通用的图像存储格式GIF、TIFF也采用CRC作为检错手段

本章小结

- 真值（二进制数）：-0.1010，+1010
- 机器数（机器码）：原码、反码、补码、移码：
  ① 正数的原码、反码、补码是一样的，符号位都是0

  ② 负数的原码、反码、补码符号位都是1，原码的数值位与真值的数值位相同，反码的数值位为真值的数值位取反，补码的数值位为真值的数值位取反加1

  ③ 只有整数才有移码，小数没有移码，移码为补码的符号位取反，数值部分相同（正数的移码符号位为1，负数的移码符号位为0）
- 定点小数：纯小数；定点整数：纯整数
- 定点数的表示范围：
  - 4位二进制整数：
    - 原码：-7～+7（1,111～0,111）
    - 反码：-7～+7（1,000～0,111）
    - 补码：-8～+7（1,000～0,111）
    - 移码：-8～+7（0,000～1,111）
  - 4位二进制小数：
    - 原码：-7/8～+7/8（1.111～0.111）
    - 反码：-7/8～+7/8（1.000～0.111）
    - 补码：-1～+7/8（1.000～0.111）

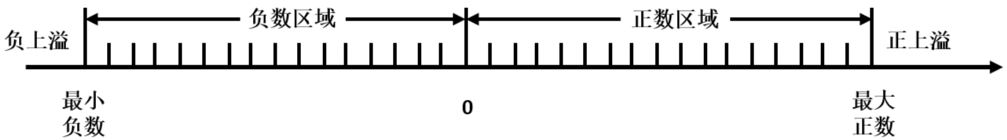
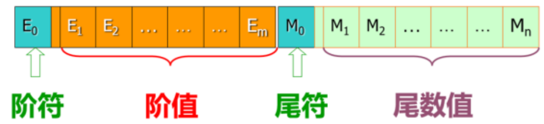


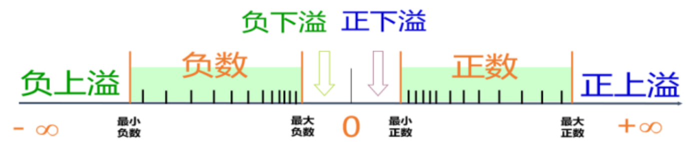
图2.5　定点数的表示范围

- 浮点数的表示形式：
  - $N = 2^{E} x M = 2^{\pm e} x(\pm 0.m)$



浮点数的数据格式

- 浮点数的表示范围：
  - 最小正数、最大正数
  - 最小负数、最大负数
  - 正上溢、负上溢
  - 正下溢、负下溢

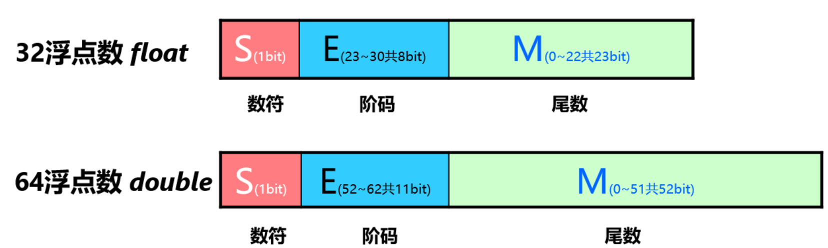


浮点数的表示范围

- 浮点数**规格化**：尾数的绝对值=[0.5, 1)；尾数用原码表示时，最高有效位为1

- 非规格化的浮点数可以通过**左规**或**右规**进行规格化：
  - 左规：尾数的绝对值太小，每左移尾数一次，尾数乘2，阶码减1
  - 右规：尾数的绝对值太大，每右移尾数一次，尾数除2，阶码加1

- **IEEE754**浮点数标准：
  - 阶码E用移码表示，偏移量为127（单精度浮点数）、1023（双精度浮点数），阶码的真值e=E-127（e=E-1023）
  - 尾数M为定点小数，尾数的真值=1.M
  - 单精度浮点数：$N = (-1)^{S} x 2^{E-127} x 1.M$
  - 双精度浮点数：$N = (-1)^{S} x 2^{E-1023} x 1.M$



- 十进制整数编码：
  - BCD码（8421码）
  - 采用BCD码时，25 = $0010\ 0101_{BCD}$
  - 采用二进制时，25 = $0001\ 1001_{B}$

- 十进制浮点数编码： $N = (-1)^{s} x 10^{E-bias} x T$
  - 这里的基数为10，尾数T不是定点小数，而是定点整数

- 汇编语言中的数据类型：
  - 由指令的操作码确定，包括无符号数运算、有符号数运算、浮点数运算

- C语言中的数据类型：
  - 整型数据类型有char（8位）、short（16位）、int（32位）、long（64位）
  - 浮点数据类型有float（32位）、double（64位）
  - 整型数据默认的为有符号数，在整型数据前加“unsigned”表示无符号数

  - C语言运算溢出
  - C语言整型数据类型转换
  - C语言浮点数据类型

- ASCII码（7位，128个字符）、扩展的ASCII码（8位，128+128个字符）

- 汉字编码：汉字机内码（如国标码），汉字输入码（也称外码，如拼音、五笔字型），汉字字形码（也称字型码，点阵码，汉字字库）

- 码距d：也称海明距离
- 码距d与检错e（能检测e个错误）、纠错t（能纠正t个错误）能力的关系
- 奇偶校验码：
  - n位原始数据+1位校验位
  - 奇校验：原始数据=101 1001，校验位=1，奇校验码=101 1001 1
  - 偶校验：原始数据=101 1001，校验位=0，偶校验码=101 1001 0
  - 奇偶校验可以发现1位错误，但不能纠正错误
  - 奇校验码=101 1001 1，经过传输或存储变为001 1001 1（最高位出错），经检测接收到的奇校验码1的个数为偶数个，表示出错
  - 偶校验码=101 1001 0，经过传输或存储变为101 1001 1（最低位出错），经检测接收到的偶校验码1的个数为奇数个，表示出错
- 交叉奇偶校验：可以发现并纠正1位错

- 海明码：是一种ECC码（Error-Correcting Codes，既能检错也能纠错的校验码）

- 能纠正1位错误的海明码也称为SEC码（Single-bit Error Correction）

- 海明码的编码：
  - 原始数据：$D_k \ldots D_2D_1$，共k位
  - 校验码：$P_r \ldots P_2P_1$，共r位
  - 海明码：$H_n \ldots H_2H_1$，共n位（n=k+r）； $n = k+r \le 2^r-1$
  - 校验码位于海明码的1、2、4、8、16、…、2^{r-1}位（从右往左数）；k=4，r=3时，海明码=$D_4D_3D_2P_3D_1P_2P_1$
  - 校验码的计算（k=4、r=3时）：
    - $P_1 = D_1 \oplus D_2 \oplus D_4$
    - $P_2 = D_1 \oplus D_3 \oplus D_4$
    - $P_3 = D_2 \oplus D_3 \oplus D_4$

- 海明码的检错和纠错：海明码能够发现并纠正1位错误
  - 检错码的计算（k=4、r=3时）：接收到的海明码为$D'_4D'_3D'_2P'_3D'_1P'_2P'_1$
    - $G_1 = P'_1 \oplus D'_1 \oplus D'_2 \oplus D'_4$
    - $G_2 = P'_2 \oplus D'_1 \oplus D'_3 \oplus D'_4$
    - $G_3 = P'_3 \oplus D'_2 \oplus D'_3 \oplus D'_4$
  - 如果$G_3G_2G_1=0$，表示没有错误；如果$G_3G_2G_1 \neq 0$，表示发生错误，根据$G_3G_2G_1$的值就可以确定是哪一位错，只需要将错误的位取反，即可以得到正确的数据

- 扩展的海明码SECDED：在普通的海明码基础上增加1位总偶校验码，扩展的海明码可以发现2位错误并纠正1位错误

- 循环冗余校验码：CRC码

- 模2运算：0 ± 0 = 0； 0 ± 1 = 1； 1 ± 0 = 1； 1 ± 1 = 0

- CRC码的编码：
  - 原始数据：k位，例如：原始数据=110，k=3
  - 生成多项式G(x)：r+1位，例如：G(x)=11101，r=4
  - 将原始数据左移r位（即原始数据右边添加r个0，r=4）：110 0000
  - 除以生成多项式（模2运算）：110 0000 / 11101；得到余数为：1001
  - CRC码为原始数据与余数的拼接：110 1001

- CRC码的解码：
  - CRC码经过传输或存储后，可能会出现错误
  - 将接收到的CRC码除以生成多项式（模2运算），得到余数，如果余数为0，则没有错误；如果余数不为0，则表示发生错误，根据余数的值可以知道是哪一位发生错误，只需要将该位取反即可得到正确的数据
  - 例如：CRC码110 1001经过传输或存储后，第7位（最高位）出错，变成010 1001
  - 010 1001 / 11101，得到余数1110（不为0），表示第7位（最高位）出错

习题（P53-56）

- 2.2
- 2.4
- 2.5
- 2.6
- 2.7
- 2.9
- 2.10
- 2.13
- 2.16
- 2.17
- 2.18

习题 2

2.1　解释下列名词。

真值　机器码　原码　反码　补码　移码　模　定点数　浮点数　溢出　精度溢出　浮点数规格化　隐藏位　BCD 码　有权码　无权码　BID 码　DPD 码　二进制浮点数　十进制浮点数　ASCII 码　机内码　字形码　字库　码距　校验码　多重奇偶校验　ECC 码　海明码　CRC 码

2.2　选择题（考研真题）。

（1）[2015] 由 3 个 “1” 和 5 个 “0” 组成的 8 位二进制补码，能表示的最小整数是________。

A. −126　　B. −125　　C. −32　　D. −3

（2）[2019] 考虑以下 C 语言代码：

```
unsigned short usi=65535;
short si=usi;
```

执行上述程序段后，si 的值是 ________。

A. −1　　B. −32767　　C. −32768　　D. −65535

（3）[2012] 假定编译器规定 int 和 short 类型长度分别为 32 位和 16 位，执行下列 C 语言语句：unsigned short x=65530；unsigned int y=x；得到 y 的机器数为 _______。

A. 0000 7FFAH　　B. 0000 FFFAH

C. FFFF 7FFAH　　D. FFFF FFFAH

（4）[2016] 有如下 C 语言程序段：short si=−32767；unsigned short usi=si；执行上述两条语句后，usi 的值为 _______。

A. −32767　　B. 32767　　C. 32768　　D. 32769

（5）[2011]float 型数据通常用 IEEE754 单精度浮点数格式表示。若编译器将 float 型变量 x 分配在一个 32 位浮点寄存器 FR1 中，且 x=−8.25，则 FR1 的内容是 _______。

A. C104 0000H　　B. C242 0000H

C. C184 0000H　　D. C1C2 0000H

（6）[2013] 某数采用 IEEE754 单精度浮点数格式表示为 C640 0000H，则该数的值是 _______。

A. -1.5×2^{13}　　B. -1.5×2^{12}　　C. -0.5×2^{13}　　D. -0.5×2^{12}

（7）[2012]float 型（即 IEEE754 单精度浮点数格式）能表示的最大正整数是 _______。

A. $2^{126}-2^{103}$　　B. $2^{127}-2^{104}$　　C. $2^{127}-2^{103}$　　D. $2^{128}-2^{104}$

（8）[2018]IEEE754 单精度浮点格式表示的数中，最小规格化正数是 _______。

A. 1.0×2^{-126}　　B. 1.0×2^{-127}　　C. 1.0×2^{-128}　　D. 1.0×2^{-149}

（9）[2014]float 型数据通常用 IEEE754 单精度浮点格式表示。假定两个 float 型变量 x 和 y 分别存放在 32 位寄存器 f1 和 f2 中，若 (f1)=CC90 0000H，(f2)=B0C0 0000H，则 x 和 y 之间的关系为 _______。

A. $x<y$ 且符号相同　　B. $x<y$ 且符号不同

C. $x>y$ 且符号相同　　D. $x>y$ 且符号不同

（10）[2010] 假定变量 i、f、d 的数据类型分别为 int、float、double（int 用补码表示，float 和 double 用 IEEE754 标准中的单精度和双精度浮点数据格式表示），已知 i=785，f=1.5678e3，d=1.5e100，若在 32 位计算机中执行下列关系表达式，则结果为真的是 ______。

Ⅰ. i==(int)(float)i　　Ⅱ. f==(float)(int)f　　Ⅲ. f==(float)(double)f　　Ⅳ. (d+f)−d==f

A. 仅Ⅰ、Ⅱ　　B. 仅Ⅰ、Ⅲ　　C. 仅Ⅱ、Ⅲ　　D. 仅Ⅲ、Ⅳ

（11）[2013] 用海明码对长度为 8 位的数据进行检错和纠错时，若能纠正一位错，则校验位数至少为 ______。

A. 2　　B. 3　　C. 4　　D. 5

2.3　回答下列问题。

（1）为什么计算机中采用二进制进行数据表示和运算？

（2）相对于奇偶校验，交叉奇偶校验的检错与纠错能力的提高需要付出哪些方面的代价？

（3）为什么计算机中采用补码表示带符号的整数？

（4）浮点数的表示范围和精度分别由什么决定？

（5）汉字输入码、机内码和字形码在汉字处理过程中各有何作用？

（6）在机内码中如何区分 ASCII 字符和汉字字符？

（7）为什么现代处理器中又开始支持十进制浮点数运算？

（8）如何识别浮点数的正负？浮点数能表示的数值范围和数值的精度取决于什么？

（9）浮点数有两个 0 会带来什么问题？

（10）简述 CRC 校验码的检错原理，CRC 能纠错吗？

2.4　写出下列各数的原码、反码和补码。

0，−0，0.10101，−0.10101，0.11111，−0.11111，−0.10000，0.10000

2.5　已知数的补码表示形式，求数的真值。

$[x]_{补}$=0.10010，　$[x]_{补}$=1.10010，　$[x]_{补}$=1.11111，

$[x]_{补}$=1.00000，　$[x]_{补}$=0.10001，　$[x]_{补}$=1.00001。

2.6　C 语言中允许无符号数和有符号整数之间的转换，下面是一段 C 语言代码。

```
int x =-1;
unsigned u=2147483648;
printf ("x=%u=%d\n", x, x);
printf ("u=%u=%d\n", u, u);
```

给出在 32 位计算机中上述程序段的输出结果并分析原因。

2.7　分析下列几种情况下所能表示的数据范围分别是多少。

（1）16 位无符号数；

（2）16 位原码定点小数；

（3）16 位补码定点小数；

（4）16 位补码定点整数。

2.8　用补码表示二进制整数，机器码为 $x_0x_1x_2x_3x_4x_5x_6x_7$，x_0 为符号位，补码的模为多少?

2.9　用 IEEE754 32 位单精度浮点数标准表示下列十进制数。

（1）$-6\dfrac{5}{8}$；　（2）3.1415927；　（3）64000。

2.10　求与单精度浮点数 43940000H 对应的十进制数。

2.11　求单精度浮点数能表示的最大数和最小数。

2.12　设有两个正浮点数：$N_1=2^m\times M_1$，$N_2=2^n\times M_2$。

（1）若 $m<n$，是否有 $N_1>N_2$？

（2）若 M_1 和 M_2 是规格化的数，上述结论是否正确?

2.13　设二进制浮点数的阶码为 3 位，尾数为 7 位。用模 2 补码写出它们所能表示的最大正数、最小正数、最大负数和最小负数，并将它们转换成十进制数。

2.14　将下列十进制数表示成浮点规格化数，阶码为 4 位，尾数为 10 位，各含 1 位符号，阶码和尾数均用补码表示。

（1）57/128；　（2）−69/128。

2.15　设有效信息为 01011011，分别写出其奇校验码和偶校验码。如果接收方收到的有效信息为 01011010，说明如何发现错误。

2.16　由 6 个字符的 7 位 ASCII 字符排列，再加上水平和垂直偶校验位构成表 2.27 所示的行列结构（最后一列 HP 为水平奇偶校验位，最后一行 VP 为垂直奇偶校验位）。

表 2.27　ASCII 交叉校验

字符	7 位 ASCII 字符							HP
3	0	X_1	X_2	0	0	1	1	0
Y_1	1	0	0	1	0	0	X_3	1
+	X_4	1	0	1	0	1	1	0
Y_2	0	1	X_5	X_6	1	1	1	1
D	1	0	0	X_7	1	0	X_8	0
=	0	X_9	1	1	1	X_{10}	1	1
VP	0	0	1	1	1	X_{11}	1	X_{12}

则 X_1、X_2、X_3、X_4 处的比特分别为 ____；X_5、X_6、X_7、X_8 处的比特分别为 ____；X_9、X_{10}、X_{11}、X_{12} 处的比特分别为 ____；Y_1 和 Y_2 处的字符分别为 ____ 和 ____。

2.17　设 8 位有效信息为 01101110，试写出它的海明校验码。给出过程，说明分组检测方式，并给出指错字及其逻辑表达式。如果接收方收到的有效信息变成 01101111，说明如何定位错误并纠正错误。

2.18　设要采用 CRC 码传送数据信息 x=1001，当生成多项式为 $G(x)$=1101 时，请写出它的循环冗余校验码。若接收方收到的数据信息为 x'=1101，说明如何定位错误并纠正错误。

实践训练

（1）在 Logisim 中设计包含 16 位数据位的海明码编解码电路，要求能够在假设没有 3 位错的前提下检测出两位错并纠正一位错。

（2）利用组合逻辑电路在 Logisim 中设计一个包含 16 位数据位的并行 CRC 编、解码电路，要求能够在假设没有 3 位错的前提下检测出两位错并纠正一位错。

关于作业提交

- 1周内必须提交（上传到学院的FTP服务器上），否则认为是迟交作业；如果期末仍然没有提交，则认为是未提交作业
  - 作业完成情况成绩=第1次作业提交情况*第1次作业评分+第2次作业提交情况*第2次作业评分+……+第N次作业提交情况*第N次作业评分
  - 作业评分：A（好）、B（中）、C（差）三挡
  - 作业提交情况：按时提交（1.0）、迟交（0.5）、未提交（0.0）
- 请采用电子版的格式（PPT文档）上传到FTP服务器上，文件名取“学号+姓名+第X次作业.pptx”
  - 例如：11920212203695+向泽旭+第2次作业.pptx
- 1周后上课时（2023年3月13日）会随机抽取1位同学到讲台上汇报作业。
- 第1次作业提交的截止日期为：2023年3月12日晚上24点。

实践训练（实验1）

- 在Logisim中设计包含16位数据位的海明码编解码电路，要求能够在假设没有3位错的前提下检测出两位错并纠正一位错。

- 利用组合逻辑电路在Logisim中设计一个包含16位数据位的并行CRC编解码电路，要求能够在假设没有3位错的前提下检测出两位错并纠正一位错。

Thanks