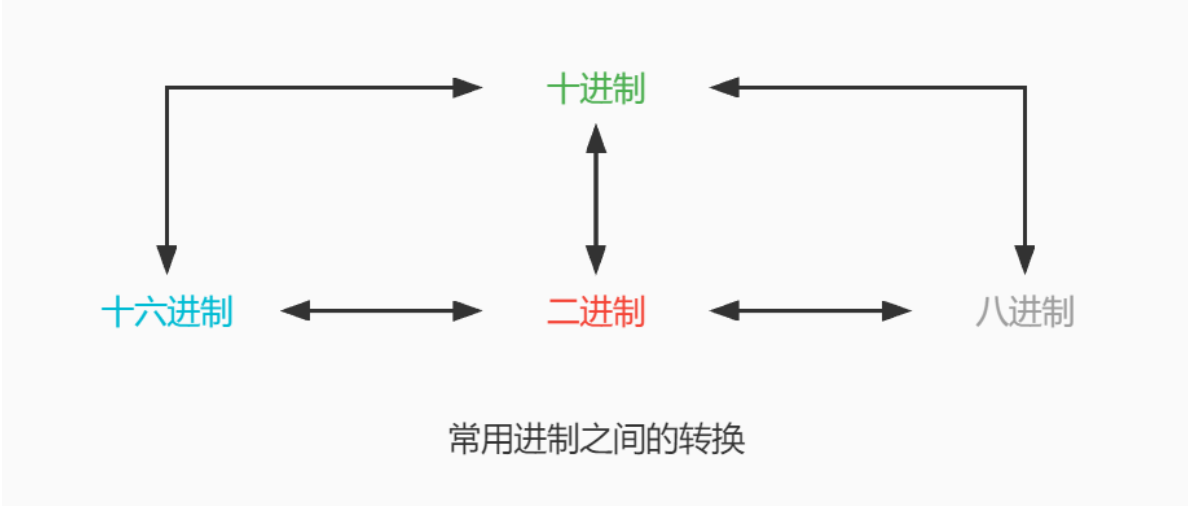
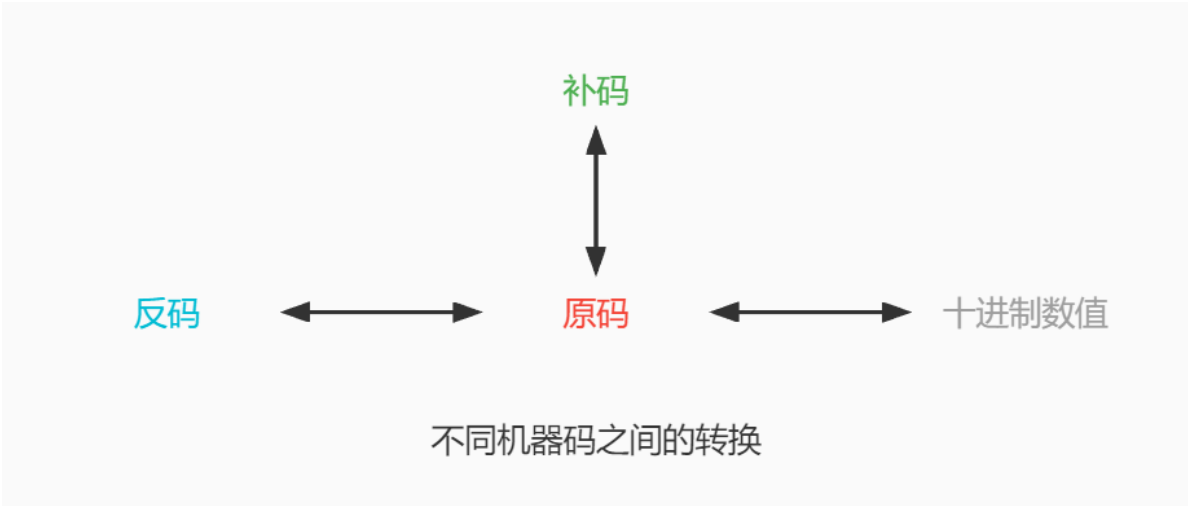
本文的核心内容是向同学们介绍**计算机的数制基础**。学习本文将有助于你：

- 理解数制的基本概念，掌握进制之间的转换方法和技巧
- 理解计算机二进制编码的三种解释方式：原码、反码、补码

阅读完本文后，你应该至少掌握下面两张图的内容。（如果你自认为已经掌握了，可以直接跳到本文末尾的拓展阅读）



常用进制之间的转换



不同机器码之间的转换

数制基础准备

> 本节的概念同学们只需要有一个感性的认识就可以，没有完全理解也不必纠结

数制也称为“计数制”，是用一组**固定的符号**和**统一的规则**来表示数值的方法。任何一个数制都包含两个基本要素：**基数**和**位权**。这句话听起来可能比较抽象，请看下述示例：

$$(810975)_{10} = 8 * 10^5 + 1 * 10^4 + 0 * 10^3 + 9 * 10^2 + 7 * 10^1 + 5 * 10^0 \tag{1}$$

这个例子使用了我们非常熟悉的十进制（810975右下角的角标10表明这个字符串使用的是十进制），我们借助这个例子来讲解上面的抽象概念

- **基数**：数制所使用**数码的个数**

  例如，十进制的基数是10，它的数码集合为$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

- **位权**：数制中某一位上的1所表示数值的大小（**所处位置的价值/权重**）

  例如，810975中的8所在位置的位权就是10^5，1所在位置的位权就是10^4，0所在位置的位权就是10^3，9所在位置的位权就是10^2，7所在位置的位权就是10^1，5所在位置的位权就是10^0

- **固定的符号**：数制所使用的数码

  例如，十进制数的数码集合为$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$，这意味着十进制数字的字符串只能由这些符号构成【实际上这里可以使用离散数学集合论中**字母表**和**语言**来形式化定义，感兴趣的同学可以在学习完离散数学II--集合论后再回过头来看这里的例子】

- **统一的规则**：对数制的解释，对于一个数制来说，一旦基数和位权被定义好，就意味着由“数码“构成的符号串会被唯一地解释成一个数值

从十进制数示例出发，我们推广到 n 位的 b 进制数 $(b_{n-1}b_{n-1}\dots b_1b_0)_b$，它的值可以定义为：

$$(b_{n-1}b_{n-1}\dots b_1b_0)_b = \sum_{i=0}^{n-1} b_i b^i \tag{2}$$

对于这样的b进制的数字（右下角的b代表这个字符串使用了b进制），我们不难看出：

- b进制数的基数为 b
- b进制数第 i 位的位权为 b^i

> **补充说明**
>
> 在前文的例子中，我们注意到一个字符串使用的进制信息是标注在字符串的右下角的。实际上我们也可以在字符串的末尾加上进制的首字母大写以表明使用的进制，例如：
>
> - 二进制 **B**(binary)
> $0101111011B$

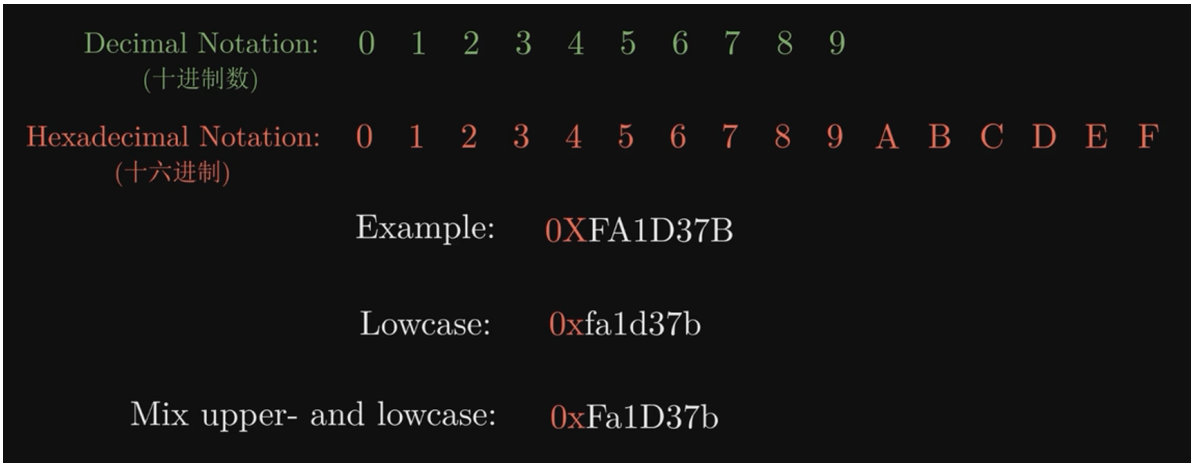
- 八进制 **O**(octal)
  $756547O$
- 十进制 **D**(decimal)
  $810975D$
- 十六进制 **H**(hexadecimal)
  $ADCB1FH$

计算机常用的进制转换

现代计算机是基于二进制的数制进行工作的，信息在计算机中以二进制字符串的形式进行存储。

然而使用**二进制**的表示法比较冗长，这种对机器友好的表示方法人是很难看明白的。而人们熟悉的**十进制**与二进制之间的转换又比较麻烦，因此我们引入了**十六进制**数。我们熟悉的十进制数是由数字0~9组成，而十六进制数字则是由数字0~9和字母A~F来表示十六个可能的数值。

十六进制数是以0x开头，这个x可以是小写也可以是大写。十六进制数字中的字母部分可以全部是大写，也可以全部是小写，甚至大小写混合也是正确的。下面这个例子很好的展示了十六进制数字的多种写法。



在历史上**八进制**的表示方式也曾广泛应用在12位、24位和36位的机器上（所以很多编程语言也支持八进制表示法），现在八进制的主要应用在处理UTF-8编码上。

我们发现，计算机领域经常使用的数制有二进制、八进制、十进制、十六进制。既然一个数值可以被表示为多种形式，那么紧接着出现的问题就是，这些不同的表示应该如何互相转换，也就是如何进行**进制转换**。

十进制⟷二进制、八进制、十六进制

二进制、八进制、十六进制⟶十进制转换

对于二、八、十六进制向十进制的转换，直接采用定义式即可，

$$(b_{n-1}b_{n-1}b_1b_0)_b = \sum_{i=0}^{n-1} b_i b^i \tag{3}$$

实例解析：

- 二进制数 ⟶ 十进制数

$$
\begin{aligned}
&(1100\ 0101\ 1111\ 1101\ 1111)B = \\
&1*2^{19}+1*2^{18}+0*2^{17}+0*2^{16}+ \\
&0*2^{15}+1*2^{14}+0*2^{13}+1*2^{12}+ \\
&1*2^{11}+1*2^{10}+1*2^{9}+1*2^{8}+ \\
&1*2^{7}+1*2^{6}+0*2^{5}+1*2^{4}+ \\
&1*2^{3}+1*2^{2}+1*2^{1}+1*2^{0} \\
&=810975D
\end{aligned} \tag{4}
$$

- 八进制数 $\longrightarrow$ 十进制数

$$
\begin{aligned}
&(305\ 7737)O = \\
&3*8^{6}+0*8^{5}+5*8^{4}+ \\
&7*8^{3}+7*8^{2}+3*8^{1}+7*8^{0} \\
&=810975D
\end{aligned} \tag{5}
$$

- 十六进制数 $\longrightarrow$ 十进制数

$$
\begin{aligned}
&(C5FDF)H = 0xC5FDF = \\
&C*16^{4}+5*16^{3}+F*16^{2}+D*16^{1}+F*16^{0} \\
&=810975D
\end{aligned} \tag{6}
$$

十进制$\longrightarrow$二进制、八进制、十六进制

十进制向其他进制转换的方法可以利用以下规律进行计算：

$$
\begin{aligned}
&\text{设}x\text{为一个非负整数，其 b 进制表达为}(b_{n-1}b_{n-2}\dots b_1b_0)_b\text{，存在以下等式：} \\
&x\ mod\ b = b_0 \\
&\lfloor x/b \rfloor = (b_{n-1}b_{n-2}\dots b_1)_b
\end{aligned} \tag{7}
$$

> 注：mod为取余操作，$\lfloor a \rfloor$ 为对 a 向下取整

相信聪明的你已经发现了，x/b 的余数就是十进制数 x 的 b 进制表达的最末一位，x/b 的商就是十进制数 x 的 b 进制表达除去最末一位的其余位。利用这个规律，我们可以递归地计算出十进制数的 b 进制表达。下面给出几个示例

- 十进制 $\longrightarrow$ 二进制

$$
\begin{aligned}
&step1:\ \lfloor 61/2 \rfloor = 30 \quad 61\ mod\ 2 = 1 \\
&step2:\ \lfloor 30/2 \rfloor = 15 \quad 30\ mod\ 2 = 0 \\
&step3:\ \lfloor 15/2 \rfloor = 7 \quad 15\ mod\ 2 = 1
\end{aligned}
$$

$$
\begin{aligned}
step4: \lfloor 7/2 \rfloor = 3 \quad 7\ mod\ 2 = 1 \\
step5: \lfloor 3/2 \rfloor = 1 \quad 3\ mod\ 2 = 1 \\
step6: \lfloor 1/2 \rfloor = 0 \quad 1\ mod\ 2 = 1 \\
\Rightarrow 61D = 111101B
\end{aligned} \tag{8}
$$

- 十进制 $\longrightarrow$ 八进制

$$
\begin{aligned}
step1: \lfloor 61/8 \rfloor = 7 \quad 61\ mod\ 8 = 5 \\
step2: \lfloor 7/8 \rfloor = 0 \quad 7\ mod\ 8 = 7 \\
\Rightarrow 61D = 75O
\end{aligned} \tag{9}
$$

- 十进制 $\longrightarrow$ 十六进制

$$
\begin{aligned}
step1: \lfloor 61/16 \rfloor = 3 \quad 61\ mod\ 16 = 13 = D \\
step2: \lfloor 3/16 \rfloor = 0 \quad 3\ mod\ 16 = 3 \\
\Rightarrow 61D = 3DH = 0x3D
\end{aligned} \tag{10}
$$

十六进制$\longleftrightarrow$二进制$\longleftrightarrow$八进制

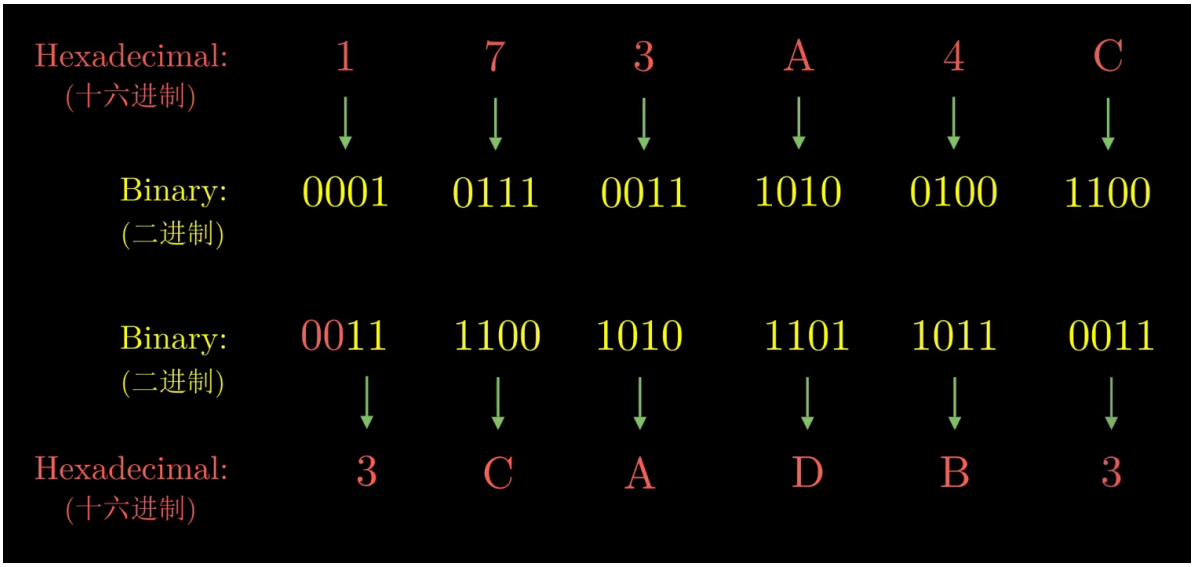
二进制 $\longleftrightarrow$ 十六进制

二进制与十六进制之间的转换比较简单直接，数字之间的转换可以参考下表。《深入理解计算机》中介绍了一个小技巧，记住十六进制数A\C\F所对应的十进制数值，那么B\D的数值可以由A\C加一得到，E的数值可以由F减一得到。

Hex digit	0	1	2	3	4	5	6	7
Decimal value	0	1	2	3	4	5	6	7
Binary value	0000	0001	0010	0011	0100	0101	0110	0111
Hex digit	8	9	A	B	C	D	E	F
Decimal value	8	9	10	11	12	13	14	15
Binary value	1000	1001	1010	1011	1100	1101	1110	1111

十六进制数向二进制数的转换比较简单，通过展开每个十六进制数字，将它转换为上表中对应的二进制格式即可。

反过来，给出一个二进制数，将它转换为十六进制。首先，我们从右向左，每四位为一组来转换为相应的十六进制数，不过要注意的是，如果总位数不是四的倍数，那么最左边的一组会出现小于四位的情况，这时将前面进行补零，然后将每四位为一组的二进制数进行一一转换，即可得到对应的十六进制数。



二进制 $\longleftrightarrow$ 八进制

二进制与八进制之间的转换和前文类似，数字之间的转换可以参考下表

Deciaml 十进制	Octal 八进制	Binary 二进制
0	0	000
1	1	001
2	2	010
3	3	011
4	4	100
5	5	101
6	6	110
7	7	111

八进制数向二进制数的转换：通过展开每个八进制数字，将它转换为上表中对应的二进制格式即可，举例来说，$1507O = 001\ 101\ 000\ 111B = 001101000111B = 1101000111B$ 。

二进制数向八进制数的转换：从右向左，每三位为一组来转换为相应的八进制数，不过要注意的是，如果总位数不是三的倍数，那么最左边的一组会出现小于三位的情况，这时将前面进行补零，然后将每三位为一组的二进制数进行一一转换，即可得到对应的八进制数。举例来说：
$1101000111B = 1\ 101\ 000\ 111B = 001\ 101\ 000\ 111B = 1507O$

八进制 $\longleftrightarrow$ 十六进制

八进制和十六进制的互相转换，可以将八进制转换成二进制，再将二进制转换成十六进制即可。

进制转换工具

除了手工计算以外，你也应该熟悉一些进制转换的工具

- window系统的计算器专门提供了程序员模式，可以快速地在十六进制、十进制、八进制、二进制之间进行数制转换。
- 网上有许多在线的进制转换网站，大家可以自行查找，这里不做赘述
- 你已经是一个成熟的软件工程系学生了，该学会自己造轮子了（x）

原码、反码、补码的运算规则

首先我想把整套关于原码、反码、补码的定义和转换规则准确清晰地写一遍，方便同学们快速开始完成相应的作业和实验。后面会附上拓展阅读部分，探讨一些原理性的内容，也希望大家首先能记住这套规定，再开始进一步的探讨。

> 本节中对概念的解释基本按照【定义及解释】【形式化书写】【示例】的顺序行文。对于这三个部分，大家可以按需阅读（比如不喜欢形式化语言可以跳过）。“不管黑猫白猫，捉到老鼠就是好猫”，只要你能掌握它们的定义和转换规则就达到了要求。

原码

原码$\longrightarrow$十进制数值

在数学上表示有符号整数时，我们使用前导的负号 $-$ 表示负数；那么用二进制表示整数的最直观想法，就是使用一位来表示正负号，这种表示方法被称作原码。我们人为规定原码的最高位为符号位，正数为0，负数为1，其余n-1位用于表示数值。从原码的定义我们可以发现，原码对二进制数的解释方式使用了符号位的概念，借助这个定义，我们自然地给出从**原码求十进制数值**的形式化表达：设 n 位二进制数 $x_{n-1}x_{n-1}\dots x_1x_0$ 使用原码表示一个数 X，则数值 X 为：

$$\begin{aligned} X &= (-1)^{x_{n-1}} \cdot \left(x_{n-2} \cdot 2^{n-2} + x_{n-2} \cdot 2^{n-2} + \cdots + x_1 \cdot 2^1 + x_0 \cdot 2^0\right) \\ &= (-1)^{x_{n-1}} \sum_{i=0}^{n-2} x_i \cdot 2^i \end{aligned} \tag{11}$$

举例而言，一个8位的机器码，则 $0000\ 0101B$ 和 $1000\ 0101B$ 的原码分别可以通过如下步骤计算，
$(0000\ 0101) = (-1)^0(1 * 2^2 + 1 * 2^0) = 5$
$(1000\ 0101) = (-1)^1(1 * 2^2 + 1 * 2^0) = -5$

原码$\longleftarrow$十进制数值

反过来，**已知十进制数值，求对应的原码**，我们设数值为 X，则其原码 $[X]_{原}$为

$$[X]_{原} = \begin{cases} D2B(X)\ ,\ 0 \leqslant X \leqslant 2^{n-1} - 1 \\ D2B(2^{n-1} - X),\ -2^{n-1} + 1 \leqslant X \leqslant 0 \end{cases} \tag{12}$$

公式中$D2B$的含义是将十进制数转换为二进制数，是$Deciaml\ to\ Binary$的缩写

举例而言，一个8位的机器码，则 $[5]_{原}=D2B(5)=(0000\ 0101)$
$[-5]_{原}=D2B(2^7-(-5))=D2B(2^7+5)=(1000\ 0101)$

对于n位原码，最高位为0，其余位全为1时，原码表示的数值取得最大值 $2^{n-1}-1$；最高位为1，其余位全为1时，原码表示的数值取得最小值 $-2^{n-1}+1$。因此 n 位原码所表示的数值范围为 $[-2^{n-1}+1,2^{n-1}-1]$。

反码

反码⟵原码

反码，英语里叫ones' complement（对1求补），这里的1，本质上是一个有限位计数系统里所能表示出的最大值，在8位二进制里就是11111111，在1位十进制里就是9，在3位十六进制里就是FFF。**求反又被称为对一求补**，用最大数减去一个数就能得到它的反。

既然反码的本质是对1求补，那为什么我们叫它反码呢？这是因为在二进制里11111111减去任何数结果都是把这个数按位取反，0变1，1变零。因此用原码求反码的方法是，正数不变，负数保留符号位1不变，剩下位按位取反。

形式化地说，设n 位二进制数 $x_{n-1}x_{n-1}\dots x_1x_0$ 使用原码表示一个数 X，则数值 X 的反码为：

$$[X]_{反}=\begin{cases}x_{n-1}x_{n-1}\cdots x_1x_0, & x_{n-1}=0\\ x_{n-1}\left(1-x_{n-1}\right)\cdots\left(1-x_1\right)\left(1-x_0\right), & x_{n-1}=1\end{cases}\tag{13}$$

举例而言，一个8位的机器码，给出原码 $(0101\ 0011)$ 和 $(1000\ 0001)$ 可以通过上述公式求出其反码：
$[0101\ 0011]原=[0101\ 0011]反$【最高位为0】
$[1000\ 0001]原=[1111\ 1110]反$【最高位为1】

反码⟶原码

反过来，通过**反码求出其原码**，只需要对前面的操作进行逆运算即可。正数不变，负数保留符号位不变，其余位取反。

举例而言，一个8位的机器码，给出反码 $(0001\ 1011)$ 和 $(1001\ 1011)$ 可以通过上述公式求出其原码：
【最高位为0】 $[0001\ 1011]反=[0001\ 1011]原$
【最高位为1】 $[1001\ 1011]反=[1110\ 0100]原$

对于n位的反码，最高位为0，其余位全为1时，原码表示的数值取得最大值 $2^{n-1}-1$；最高位为1，其余位全为0时，反码表示的数值取得最小值 $-2^{n-1}+1$。因此 n 位反码所表示的数值范围为 $[-2^{n-1}+1,2^{n-1}-1]$。

补码

补码⟵原码

补码，英语里叫做two's complement（对2求补），这个2指的是计数系统的容量（模），就是计数系统所能表示的状态数，所以补码是在模意义上定义的。对1位二进制数来说只有0和1两种状态，所以模是$10B$。对7位二进制数来说就是$1000\ 0000B$，用模减去一个数（无符号部分）就能得到这个数的。比如$1000\ 0000B-101\ 0010B=010\ 1110B$，事实上因为$1000\ 0000B=111\ 1111B+1B$，稍加改变就成了$(1111111B-1010010B)+1B$，括号里面的式子和前文提到的反码取反操作相同，所以**求补又可以表述为先求反再加1。**因此：**已知原码求补码的方法**就是**正数依旧不变，负数保留符号位不变，其余部分先求反码再加上1。**

形式化地说，设 n 位二进制数 $x_{n-1}x_{n-1}\dots x_1x_0$使用原码表示一个数 X，则数值 X 的补码为：

$$[X]_{补}=\begin{cases}x_{n-1}x_{n-1}\cdots x_1x_0, & x_{n-1}=0\\ x_{n-1}\left(1-x_{n-1}\right)\cdots\left(1-x_1\right)\left(1-x_0\right)+1, & x_{n-1}=1\end{cases}\tag{14}$$

举例而言，一个8位的机器码，给出原码 $(0001\ 1011)$ 和 $(1001\ 1011)$ 可以通过上述公式求出其补码：
$[0001\ 1011]_{原}=[0001\ 1011]_{补})$【最高位为0】
$[1001\ 1011]_{原}=([1110\ 0100]+1)_{补}=[1110\ 0101]_{补}$【最高位为1】

补码⟶原码

反过来，已知一个数值X的补码想求对应原码，只需要对前面的操作进行逆运算即可。正数不变，负数的符号位不变，其余位减一再取反。

举例而言， 一个8位的机器码，给出反码 $(0001\ 1011)$ 和 $(1001\ 1011)$ 可以通过上述公式求出其原码：
【最高位为0】 $[0001\ 1011]_{补}=[0001\ 1011]_{原}$
【最高位为1】 $[1001\ 1011]_{补}=[1001\ 1011]_{反}-1_{反}=[1001\ 1010]_{反}=[1110\ 0101]_{原码}$

对于n位的补码，最高位为0，其余位全为1时，补码表示的数值取得最大值 $2^{n-1}-1$；最高位为1，其余位全为0时，补码表示的数值取得最小值 -2^{n-1}。因此 n 位补码所表示的数值范围为 $[-2^{n-1},2^{n-1}-1]$。

拓展阅读

补码是怎么来的?

在上一节中我们介绍了原码、反码、补码的定义和转换规则。不难发现，原码的表示方式最容易被人理解，那为什么还会出现反码和补码呢？这个问题实际上和计算机硬件的设计有关。计算机实现加法运算是很容易的，但直接作减法则比较复杂，需要处理借位等情况，内部逻辑组件会增多，效率也就会降低。

所以我们需要一种编码方式，它支持**将减法转换成加法**，即$A-B=A+(-B)$

原码

先来看看原码，原码的优点是编码格式对人很友好，类似十进制中的正负号，原码用最高位0和1分别代码正负数，很直观的表示了正负数。但是原码也有一个很大的缺点，就是无法将减法转换成加法运算，如：

例： $4-2$【使用原码】

$$4D - 2D = 4D + (-2D) = 0100B + 1010B = 1110B = -6H \tag{15}$$

上面例子计算$4-2$，将$4-2$转换成$4+(-2)$并用原码计算，得出的结果错误，原码虽然很直观转换了十进制数，但是并不支持将减法转换为加法。

反码

于是人们考虑寻找一种支持将减法转换为加法二进制编码方式，最初找到的解决方案是反码。反码的负数编码格式不像原码那样直观，但是却可以将减法转换成加法了，反码减法规则为：A - B = A + （-B），如果最高位发生了溢位，则需要在最低位加上1，如下面两个例子：

例： $4-2$【使用反码】

$$4D - 2D = 4D + (-2D) = 0100B + 1101B = 0010B = 2D \tag{16}$$

例： $2-2$【使用反码】

$$2D - 2D = 2D + (-2D) = 0010B + 1101B = 1111B = 0D \tag{17}$$

运用反码减法规则，得到的上面两个例子的减法结果是正确的，所以计算机是可以使用反码存储和计算的，早期的计算机如CDC 6000、LINC、PDP-1等都是使用反码的，但是反码也有两个缺点：1）0有两种编码，+0（0000）和 -0 （1111），在判断0时，需要分别判断0000和1111；2）反码减法的算法规则比较复杂，需要增加计算机内部逻辑组件额外判断溢位，会影响计算效率。

补码

于是人们又对反码进行了改进，最后发现在模系统下定义的补码可以解决了上面反码的两个缺点。补码的减法规则比较简单，按照最简单的转换公式$A-B=A+(-B)$，当减去一个数时直接转换成加上被减数的负数即可，不用像反码那样额外处理溢位，如下面两个例子：

例： $4-2$【使用补码】

$$4D - 2D = 4D + (-2D) = 0100B + 1110B = 0010B = 2D \tag{18}$$

例： $2-2$【使用补码】

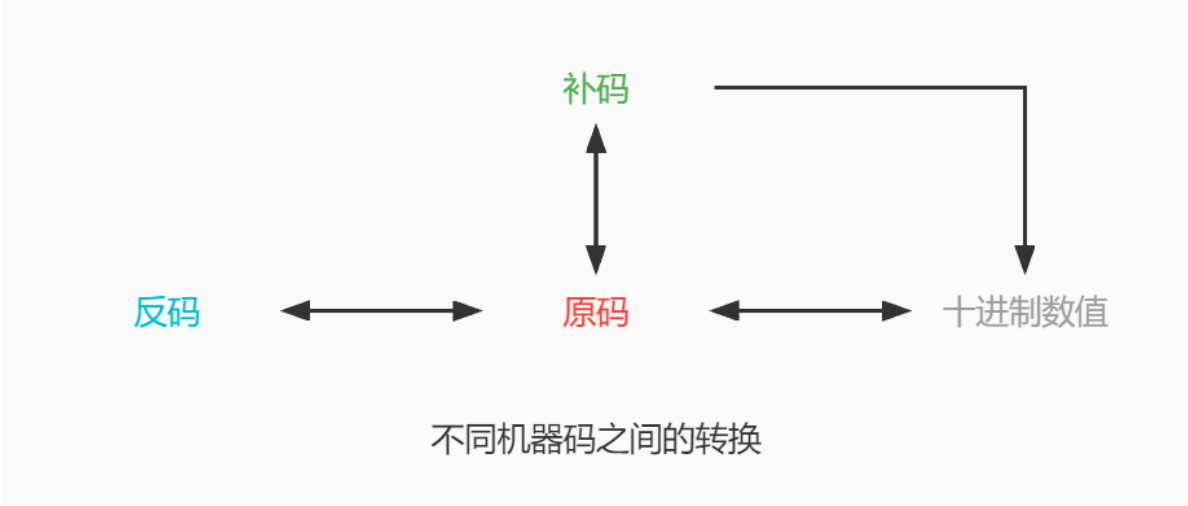
$$2D - 2D = 2D + (-2D) = 0010B + 1110B = 0000B = 0D \tag{19}$$

使用了补码的加法，上面两个例子得出的结果都是正确的，相对于反码，补码加法更简单，直接丢弃溢位，不需要针对溢位单独处理，所以用补码做运算效率高。因此补码是现代计算机使用的编码格式。

> 那么为什么补码能解决另外两种编码的问题呢？感兴趣的同学可以阅读参考文章 [3] 中的3.2节 模N加减法

Tricks

下面介绍一些原码、补码、反码相关的小技巧，了解这些技巧后，不同机器码之间的转换图可以更新为如下方式：



不同机器码之间的转换

- 反码和补码之间的转换

  对于非负数，反码和补码是一致的，无需转换

  对于负数，前文已经提到过，补码=原码保留符号位，其余取反，再加一，又有反码=原码保留符号位，其余取反。那么我们发现，反码和补码之间其实只相差了1。也就是说：补码等于反码加一。

- 补码的补码是原码

  让我们回顾补码的定义——two's complement（对2求补），这个2是计数系统的模。假设一个计数系统的模为N，在这个计数系统下，A的补码为(N-A)。那么A补码的补码就是N-(N-A)，可以发现，A补码的补码就是A本身。所以对于负数，从补码求原码不仅可以通过逆运算**减一取反**，还可以利用模运算系统的性质，再次**取反加一**。

- 补码$\longrightarrow$十进制

  不知道你有没有注意到，本文一直在强调一个概念——**解释**。事实上，对于原码、反码、补码而言，从二进制翻译到十进制的过程，其实都可以看作是不同的解释方式，它们对二进制字符串赋予了不同的位权，进而产生了不同的编码效果。

  在CSAPP一书中给出了补码的位权解释方式，设补码为n 位二进制数 $x_{n-1}x_{n-1}\ldots x_1x_0$ 的二进制数，形式化地说，其对应的十进制数值 X 为

  $$X = x_{n-1}(-2^{n-1}) + x_{n-2}2^{n-2} + x_{n-3}2^{n-3} + \cdots + x_12^1 + x_02^0 \qquad (20)$$

  可以看到，补码的最高位的位权是-2^{n-1} 而不是 2^{n-1}（这样的位权解释方式恰好符合前文提到的对2求补模运算，同学们可以自行验证一下）。使用这个公式，我们可以快速地将补码转换为十进制数。

- 补码的符号扩展

  思考这样一个问题，现在有一个16位的补码$1000\ 0001\ 0000\ 1011$，要想得到同样数值的32位补码应该怎么办？你或许想把这个16位的补码先转换为十进制数值再将这个数值转换为32位的补码。但事实上，这样的转换有一个专门的名称——符号扩展。

这里我只简单介绍结论：补码扩展过程中，补码的符号正数前面补0，负数前面补1，也就是只需要补上符号位的数即可。

例如四位补码扩展成八位，正数0011，负数1101，经过符号扩展的结果是，正数0000 0011，负数1111 1101

> 具体的直觉观察过程和数学推导步骤有兴趣的同学可以自行查看参考内容[1] CSAPP第二章 信息的表示和处理P54-55页。

数制部分的内容就介绍到这里，祝各位同学在计算机硬件基础里玩的开心 :-)

Reference:

[1]《深入理解计算机系统》（CSAPP）——第二章 信息的表示和处理

[2] CSAPP-深入理解计算机系统】2-1.信息的存储(上) -B站 https://www.bilibili.com/video/BV1tV411U7N3?share_source=copy_web&vd_source=64521a646fa1bf3b43544380867c980b

[3] 原码、反码、补码的产生、应用以及优缺点有哪些？ - 祥先生的回答 - 知乎 https://www.zhihu.com/question/20159860/answer/713291288