# 第二章 数制与编码（自学重点）-- 第三周

这一章是计算机学科的基础，虽然有些枯燥，但建议好好过一下，对于理解编程时出现的莫名其妙的 bug 有帮助，理解数据溢出、类型转换、零扩展&符号位扩展、浮点数表示等都有很大帮助。

一、数制

1、掌握进制的两种表示方式：

1）并列表示法：如（4562）$_{10}$、（01101111）$_{2}$、（A7BF）$_{16}$，也常用 B 代表二进制，H 代表 16 进制（或前缀加 0x），D 代表十进制（没标注，默认是 10 进制）。

2）多项式表示法：

如（N）$_{M}=d_{n-1}(M)^{n-1}+d_{n-2}(M)^{n-2}+\cdots+d_{1}(M)^{1}+d_{0}(M)^{0}+d_{-1}(M)^{-1}+d_{-2}(M)^{-2}+\cdots+d_{-m}(M)^{-m}=\sum_{i=-m}^{n-1} d_i\,(M)^{i}$（任意进制数按它自己的基数加权展开后，进行的十进制运算得到的结果就是它对应的十进制数制表示）。---**任意进制数转十进制的方法**

2、进制转换

1）掌握**基数乘除法（记住口诀：小数点在头顶；除（乘）数是被转换的进制数；除（乘）到商（余）为零为止）**和**直接替换法**（只用于 2,8,16 进制间）。

2）小数部分转换经常是约等于，存在**转换精度**问题（如：十进制 0.32 转换为二进制）。

二、编码

1、有符号数的计算机表示方法

1）原码、反码、**补码**（重点：在后续学习中体会补码引入的三个好处：0 的编码问题、不需要减法器、符号位扩展的好处）、**移码**（浮点表示中阶数常使用的编码）。

计算机使用补码表示有符号数有几个重要的优点：

1. **简化硬件设计**：
   - 补码允许加法器电路同时处理正数和负数的加法操作，无需区分加法和减法操作。这意味着可以使用相同的硬件来执行加法和减法运算，简化了算术逻辑单元（ALU）的设计。
2. **统一的溢出检测**：
   - 使用补码表示法，无论是正数还是负数，都可以用相同的方式来检测运算结果是否溢出。这使得溢出检测更加简单和一致。
3. **零的唯一表示**：
   - 在补码表示中，零只有一个表示形式（即全0），这与原码或反码不同。原码和反码中零有两种形式（+0 和 -0），这可能导致不必要的复杂性。
4. **负数的简单表示**：
   - 对于一个给定的位数，补码表示下的负数可以通过将对应的正数取反后加一得到，这使得负数的生成非常直观且容易实现。
5. **最大值和最小值对称**：
   - 在补码系统中，对于一个特定的位数，可表示的最大正整数和最小负整数是对称的（除了额外的一个最小负数）。比如8位二进制数，最大正数是 `01111111`（十进制127），最小负数是 `10000000`（十进制-128）。
6. **便于算术移位**：
   - 在进行算术右移时，补码表示可以正确地保留最高有效位（即符号位），从而保持数值的符号不变。这对于保持数值的正确性非常重要。
7. **便于比较操作**：
   - 在补码表示中，比较两个数的大小只需要按照无符号数的大小顺序来进行比较即可，因为补码表示的序是自然的。

总的来说，使用补码表示有符号数可以使计算机中的算术运算更加高效和可靠，并且简化了硬件设计。这也是为什么现代计算机系统普遍采用补码作为有符号整数的标准表示方法。

2）关于**零扩展**（无符号数的数据对齐）和**符号位扩展**（有符号数的数据对齐），可通过表 7.3 中数据传输类指令实例体会什么时候用零扩展，什么时候用符号位扩展，数据通

路设计里也有涉及。

2、定点表示与浮点表示

两种表示的区别（书上有定义，也不难理解），他们的应用场合，定点数如日常生活中的金钱计数，浮点数在计算机中应用更多些（了解 IEEE754 标准，理解关于阶码和位数部分位数分配的取舍问题），了解浮点运算中的“大数吃小数”问题，拓展如何消除浮点运算的累计误差问题。

3、BCD 码

重点掌握常用的两种: 8421BCD 和余三码（自补码的一种）；

4、字符编码

了解 ASCII 码，拓展 Unicode 码，在本课程中用的不多，但编程中用的多。

ASCII 码中几个常用字符的编码可记住：

’a’ = 61H =97 ; ‘A’ = 41H = 65; ‘0’ = 30H = 48

大小写字母见差 20H，例如‘a’+20H=‘A’或 ‘a’ & 0xDF = ‘A’(bit5 清零,其他位不变)

5、可靠性编码

了解格雷码（编码本身不带校验位）、奇偶校验码（带校验位，只能发现错误，不能纠错）。拓展海明校验（带校验位，能发现错误，同时能纠错）、CRC 校验码（用的较多）。

**说明：**拓展部分不会考。