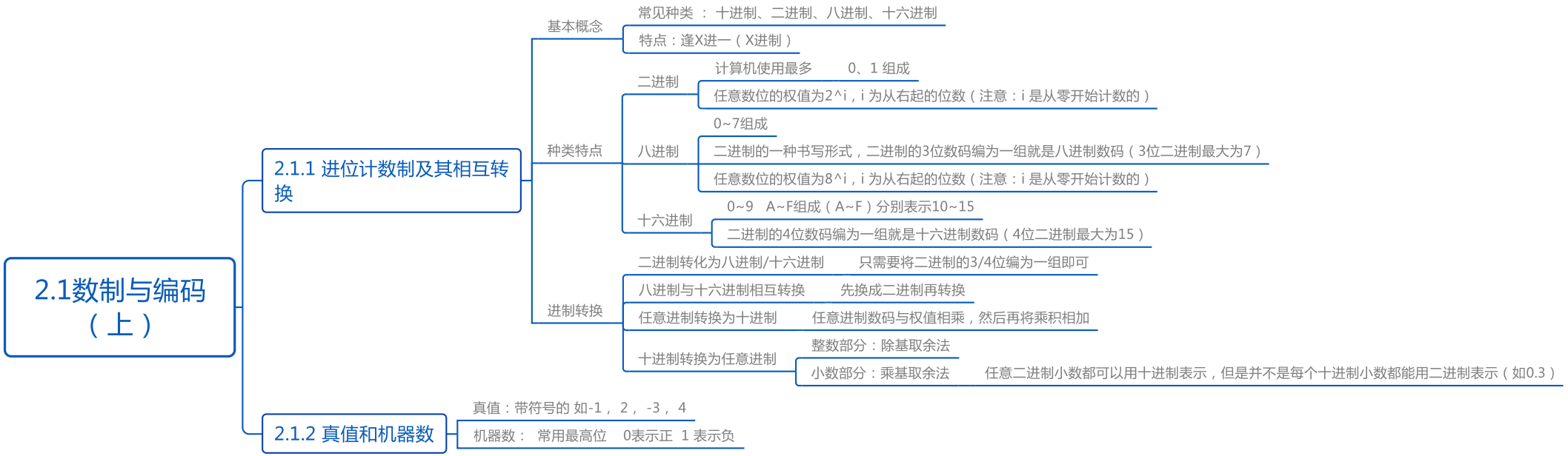

2.1数制与编码（上）
2.1.1 进位计数制及其相互转换
基本概念
常见种类 ： 十进制、二进制、八进制、十六进制
特点：逢X进一（X进制）
种类特点
二进制
计算机使用最多 0、1 组成
任意数位的权值为2^i，i 为从右起的位数（注意：i 是从零开始计数的）
八进制
0~7组成
二进制的一种书写形式，二进制的3位数码编为一组就是八进制数码（3位二进制最大为7）
任意数位的权值为8^i，i 为从右起的位数（注意：i 是从零开始计数的）
十六进制
0~9 A~F组成（A~F）分别表示10~15
二进制的4位数码编为一组就是十六进制数码（4位二进制最大为15）
进制转换
二进制转化为八进制/十六进制 只需要将二进制的3/4位编为一组即可
八进制与十六进制相互转换 先换成二进制再转换
任意进制转换为十进制 任意进制数码与权值相乘，然后再将乘积相加
十进制转换为任意进制
整数部分：除基取余法
小数部分：乘基取余法 任意二进制小数都可以用十进制表示，但是并不是每个十进制小数都能用二进制表示（如0.3）
2.1.2 真值和机器数
真值：带符号的 如-1， 2， -3， 4
机器数： 常用最高位 0表示正 1 表示负

2.1数制与编码（下）

2.1.3 BCD码

- 8421码（最常用）　注意：如果两个8421码相加后的和小于9（10进制），需要加6（10进制）进行修正
- 余3码（无权码）　在8421码的基础上加（0011），因为每个数都多3，所以称为余3码　8--->1011
- 2421码（有权码）
  - 权值由高到低为 2，4,2,1
  - >=5的4位二进制数中最高位为1 ，<5的最高位为0　5--->1011

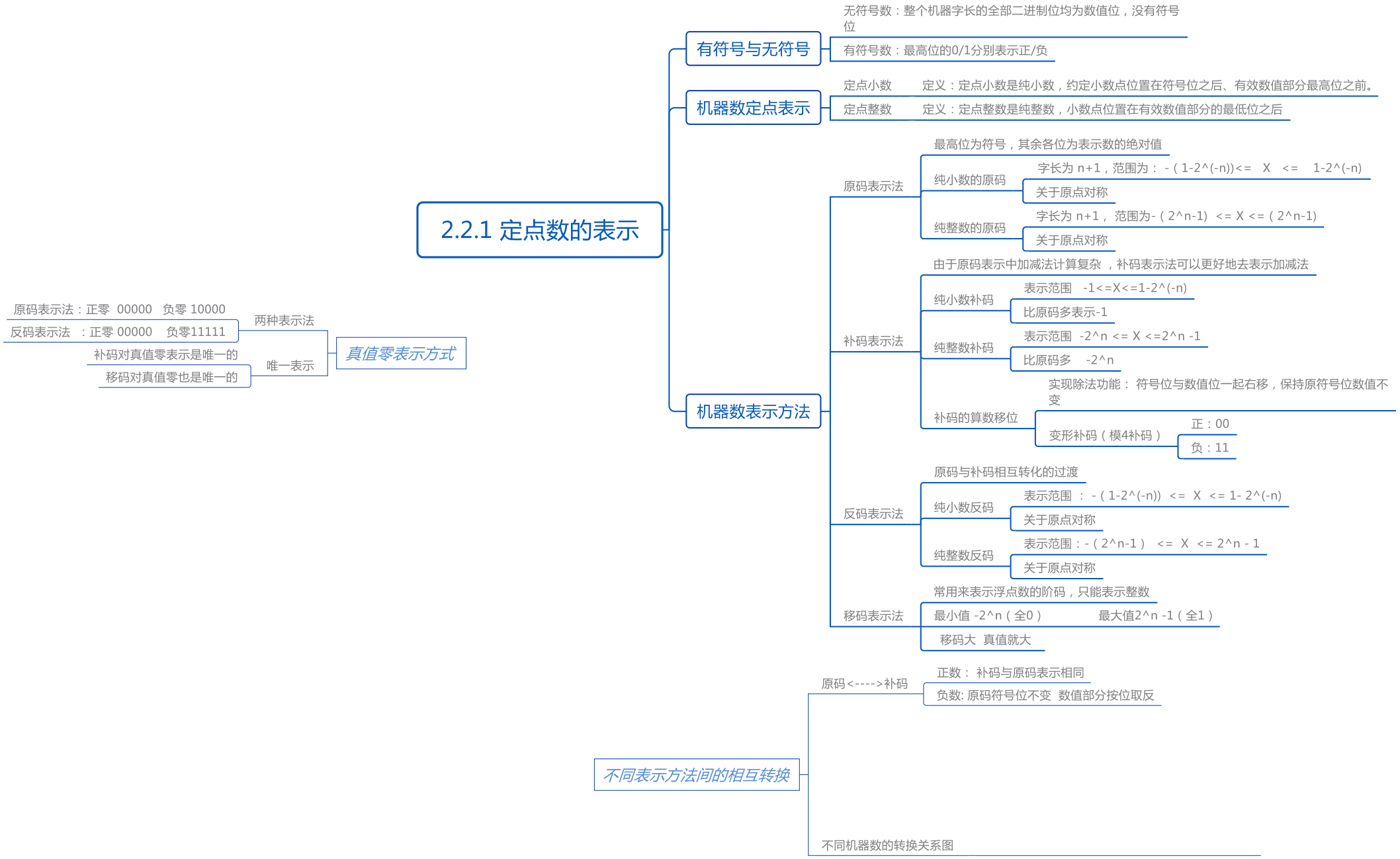
2.1.4 字符与字符串

- 字符串编码ASCII码　7位二进制编码
- 汉字的表示和编码
  - 每个编码用两个字节表示
  - 种类
    - 输入编码　计算机输入
    - 汉字内码　计算机内部处理
    - 汉字字形码　计算机输出
- 字符串存放
  - 小端模式　将数据的最高有效字节存放在高地址单元中
  - 大端模式　将数据的最高有效字节存放在低地址单元中
- 从低地址到高地址逐个字符存储，常采用'\0'作为结尾标志

2.1.5 校验码

- 概念：能够发现或者自动纠错的数据编码
- 原理：通过添加一些冗余码，实现检验或者纠错编码
- 分类
  - 奇偶校验码
    - 奇校验码：有效信息位和校验位中 1 的个数为奇数
    - 偶校验码：有效信息位和校验位中 1 的个数为偶数
    - 码距：2

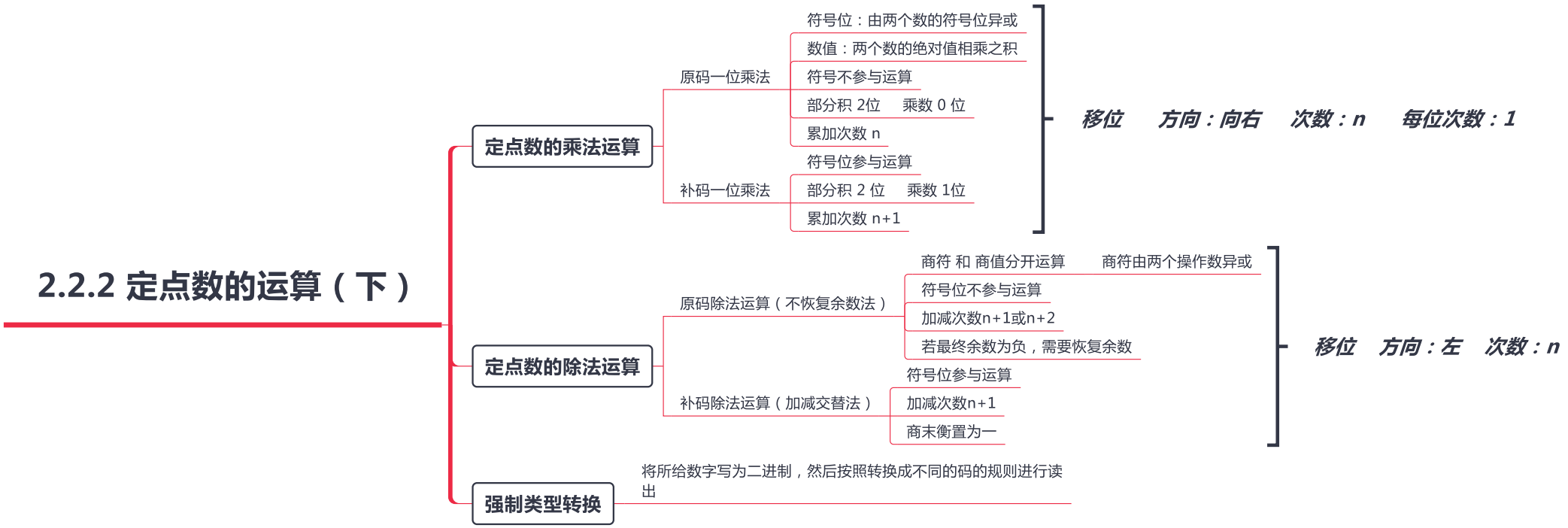
    但是如果编码中出现偶数位错误，无法检测
  - 海明码
    - 在有效信息位中添加几个校验码形成海明码
    - 不仅可以发现错位，还可以对错位进行纠错
    - 编码最小码距L越大，检测位数越多，纠错能力越强（纠错能力恒小于等于检测能力）　先检错，才能纠错
    - 补充
      - 海明码有1位纠错，2位检错能力
      - 为了区分1位错和2位错，还需添加"全校验位"对整体进行偶校验
      - 注意：有的题目位置编号可能是从小到大的，但处理方法雷同
  - CRC（循环冗余码）
    - 常用于大量的数据传送时的校验
    - 接收到循环冗余码后，对生成多项式做模2除法，余数为0则无错误
    - 余数不为0，对相应位置取反
    - 检错、纠错能力
      - 可检测出所有奇数个错误
      - 可检测出所有双比特的错误
      - 可检测出所有小于等于校验位长度的连续错误
      - 若选择合适的生成多项式，且2^R≥K+R+1，则可纠正单比特错


2.2.1 定点数的表示
有符号与无符号
无符号数：整个机器字长的全部二进制位均为数值位，没有符号位
有符号数：最高位的0/1分别表示正/负
机器数定点表示
定点小数
定义：定点小数是纯小数，约定小数点位置在符号位之后、有效数值部分最高位之前。
定点整数
定义：定点整数是纯整数，小数点位置在有效数值部分的最低位之后
机器数表示方法
原码表示法
最高位为符号，其余各位为表示数的绝对值
纯小数的原码
字长为 n+1，范围为：-(1-2^(-n))<= X <= 1-2^(-n)
关于原点对称
纯整数的原码
字长为 n+1， 范围为-(2^n-1) <= X <= (2^n-1)
关于原点对称
补码表示法
由于原码表示中加减法计算复杂 ，补码表示法可以更好地去表示加减法
纯小数补码
表示范围 -1<=X<=1-2^(-n)
比原码多表示-1
纯整数补码
表示范围 -2^n <= X <=2^n -1
比原码多 -2^n
补码的算数移位
实现除法功能： 符号位与数值位一起右移，保持原符号位数值不变
变形补码（模4补码）
正：00
负：11
反码表示法
原码与补码相互转化的过渡
纯小数反码
表示范围 ：-(1-2^(-n)) <= X <= 1-2^(-n)
关于原点对称
纯整数反码
表示范围：-(2^n-1) <= X <= 2^n - 1
关于原点对称
移码表示法
常用来表示浮点数的阶码，只能表示整数
最小值 -2^n（全0） 最大值2^n -1（全1）
移码大 真值就大
不同表示方法间的相互转换
原码<---->补码
正数： 补码与原码表示相同
负数: 原码符号位不变 数值部分按位取反
不同机器数的转换关系图
真值零表示方式
两种表示法
原码表示法：正零 00000 负零 10000
反码表示法 ：正零 00000 负零11111
唯一表示
补码对真值零表示是唯一的
移码对真值零也是唯一的

2.2.2 定点数的运算（上）


定点数移位运算
算数移位
正数 移位后添零
负数
原码 添零
补码 左0右1
反码 全部填1
符号位不参与运算
逻辑移位
将操作数看做无符号数
左移或者右移 都要添零
循环移位
带进位标志位的循环移位
不带进位标志位的移位的循环移位
原码定点数的加减法运算
加法准则
符号相同：绝对值相加，符号不变
符号不同：绝对值大的减去绝对值小的 符号去绝对值大的数
减法准则 1.减数的符号取反 2.将其与被减数做原码加法运算
补码定点数加减法运算
1、参与的操作数均为补码
2、按照二进制规则运算 逢二进一
3、 符号位与数值位同时参与运算，符号位产生的进位丢掉，结果的符号由运算得出
4、补码运算的结果仍然是补码
符号扩展
正数 在原有的基础上，添零凑位数即可
负数
原码 将原有形式的符号移动到新形式的符号位上，新形式的附加位进行添零处理
补码 加1处理
反码 加1处理
溢出概念和判别方法
上溢：大于最大可以表示正数
下溢：小于最小可以表示的负数
补码判断溢出方法
一位符号位：参加运算的两个数的符号相同，但是结果符号出现变化，则结果溢出
双符号位：（模4补码）
00 正数 无溢出
01 正溢出
10 负溢出
11 负数 无溢出
一位符号位根据数据位进位判断 符号位进位与最高位进位相同，则无溢出

2.2.2 定点数的运算（下）


定点数的乘法运算
原码一位乘法
符号位：由两个数的符号位异或
数值：两个数的绝对值相乘之积
符号不参与运算
部分积 2位　乘数 0 位
累加次数 n
补码一位乘法
符号位参与运算
部分积 2 位　乘数 1位
累加次数 n+1
移位　方向：向右　次数：n　每位次数：1
定点数的除法运算
原码除法运算（不恢复余数法）
商符 和 商值分开运算　商符由两个操作数异或
符号位不参与运算
加减次数n+1或n+2
若最终余数为负，需要恢复余数
补码除法运算（加减交替法）
符号位参与运算
加减次数n+1
商末衡置为一
移位　方向：左　次数：n
强制类型转换
将所给数字写为二进制，然后按照转换成不同的码的规则进行读出

2.3.1浮点数的表示

- **表示格式**
  - 阶码是整数，阶符与阶码的位数共同反映浮点数的表示范围与小数点的实际位置
  - 数符表示浮点数的符号
  - 尾数的位数表示浮点数的精度
- **规格化浮点数**
  - 左规：尾数算数左移一位，阶码减一
  - 右规：结果出现溢出（双符号位01或10），将尾数算数右移一位，阶码加1

  左规可以多次，右规只能一次
- **原码规格化**
  - 正数：0.1XXX
  - 负数： 1.1XXX

  基数：4　原码规格化形式的尾数最高2位不全为0
  基数：8　原码规格化形式尾数最高3位不全为0
- **补码规格化**
  - 正数：0.1XXX
  - 负数：1.0XXXX
- **IEEE754**
  - 整数部分隐含1
    - 短浮点数（组成位数）
      - 数符：1
      - 阶码 8
      - 尾数数值：23
      - 总位数：32

      偏置值：7fh（16进制）　127（十进制）
    - 长浮点数（组成位数）
      - 数符：1
      - 阶码 11
      - 尾数数值：52
      - 总位数：64

      偏置值：3ffh（16进制）　1023（十进制）
  - 临时浮点数（组成位数）
    - 数符：1
    - 阶码 15
    - 尾数数值：64
    - 总位数：80

    偏置值：3fffh（16进制）　16383（十进制）
  - 浮点数范围
    - 单精度浮点数：2^（-126）~2^(127)*(2-2^(-23))
    - 双精度浮点数：2^(-1022)~2^(1023)*(2-2^(-52))
- **定点数与浮点数的区别**
  - 相同字长，浮点数范围更大
  - 浮点数精度降低
  - 浮点数运算复杂
  - 浮点数只有规格化后阶码超出表示范围，才发生溢出

2.3.2 浮点数的加减运算

- **运算步骤**
  - 对阶：小阶看齐大阶，阶码小的尾数右移一位，阶加一，直到阶码相等
  - 尾数求和　尾数按照定点数加减规则运算
  - 规格化
    - 最高数值位与符号位不同即为规格化形式
    - 左规：尾数左移1位，和的阶码减1　直到00.1XX或者11.0XXX
    - 右规，尾数求和结果溢出（10.XXX或者01.XXX） 尾数右移一位，和的阶码加1
  - 舍入
    - 0舍1入法
      - 尾数右移时，移去的最高数值位为0，则舍去
      - 尾数左移时，移去的最高数值位为1，则尾数末位加1

      } ***可能会导致溢出，此时需要再一次右规***
    - 恒置1法　尾数右移，不论最高数值位丢掉的是1还是0，都将尾数末位恒置为1
  - 溢出判断
    - 只有右规后，仍然溢出， 此时才是真正溢出
    - 上溢出：进入中断处理
    - 下溢出：按机器零处理
  - 强制类型转换
    - char-->int　在前面补0
    - int <----->unsigned　彼此都可能因为溢出丢失数据
    - int<------>float
      - float转换为int可能会出现精度损失和溢出
      - int转换为float 可能会出现数据舍入
- **边界对齐**
  - 现代计算机通常是按字节编址，即每个字节对应1个地址
  - 通常也支持按字、按半字、按字节寻址

2.4算术逻辑单元（ALU）

- **运算器组成：算术逻辑单元 累加器 状态寄存器 通用寄存器组**
- **2.4.1串行加法器和并行加法器**
  - 一位全加器
    - 两个加数输入，以及低位进入输入
    - 本位结果和进位输出
  - 串行加法器
    - 只有一个全加器，数据逐位的送入加法器中运算，逐位送回寄存器
    - 操作数n位 则进行n次
    - 成本低，但是速度慢
  - 并行加法器
    - 多个加法器共同组成，每个全加器都有一个低位送来的进位输入，向高位的进位输出
    - 进位方式
      - 串行进位 将全加器串接在一起，每级进位依赖于前一级进位
      - 并行进位 同时进位，各级进位信号同时形成

提高并行加法器速度的关键在于加快进位产生和传递速度

- **2.4.2 算术逻辑单元的功能和结构**
  - ALU：算数运算与逻辑运算