王道计算机组成原理笔记

王艺霖

2022 年 9 月 13 日

1 检验码

1.1 循环冗余检验码（CRC 码）

CRC 码的基本思想

1. 数据发送、接收方约定一个“除数”
2. K 个信息位 +R 个检验位作为“被除数”，添加检验位后需保证除法的余数为 0
3. 收到数据后，进行除法检查余数是否为 0, 若余数非 0 说明出错，则进行重传或者纠错

如何构造

模 2 除法介绍：

1. 被除数首位为 1 时，商为 1；被除数首位为 0 时，商为 0；
2. 每一步得到的余数都要抛弃首位；
3. 若新的被除数首位 (即已抛弃首位的余数) 为 0，除数为 0；

如何检查纠错

CRC 是一种检错方法，而 FCS 是添加在数据后面的冗余码，在检错方法上可以选用 CRC，但也可以不选用 CRC。

在接受端把接收到的数据以帧为单位进行 CRC 检验：把收到的每一个帧都除以同样的除数 P（模 2 运算），然后检查得到的余数 R。

补码的真值 0，只有一种表示形式

备注

K 个信息位, R 个检验位, 若生成多项式选择得当, 且 $2^R >= K+R+1$, 可纠正 1 位错误。

原因为 R 位可以表示出 2^R 种状态，其中有 $2^R - 1$ 种错误的状态，而有 $K + R$ 位

理论上可以证明循环冗余码的检错能力有以下特点:

1. 可检测出所有奇数个错误
2. 可检测出所有双比特的错误
3. 可检测出所有小于等于检验位长度的连续错误

2 定点数

2.1 定点数 vs 浮点数

定点数小数: 位数固定 3.14

浮点数小数: 位数不固定 $3.14 * 10^2$

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，相当于数的绝对值

表示范围：

8 位二进制数：2^8 种不同的状态

00000000 11111111 = 100000000 − 1

n 位的无符号整数表示范围为：0 到 $2^n - 1$

有符号数的定点表示

定点整数: 符号位，数值部分，小数点位置

定点小数: 符号位，小数点位置，数值部分

原码

原码：用尾数表示真值的绝对值，符号位“0/1”对应“正/负”

若机器字长为 $n + 1$ 位，那么表示的数的范围为：$-2^n + 1$ $2^n - 1$

反码

若符号位为 0，则反码与原码相同

若符号位为 1，则**数值位**全部取反

真值有 **+0** 和**-0** 两种

补码

1. 正数的补码 = 原码

2. 负数的补码 = 反码末尾 +1
3. 0 的补码只有一种真值形式 00000000
4. 定点整数：[X] 补 = 1,0000000 表示 X=2^7
5. 若机器字长 $n+1$ 位，补码整数的表示范围：$-2^n <= x <= 2^n - 1$
6. 定点小数补码 [X] 补 =1.0000000 表示 X=-1
7. 若机器字长为 n+1 位，补码小数的表示范围：$-1 <= x <= 1 - 2^{-n}$（比原码多一个-1）

移码

移码：补码的基础上将符号位取反。**注意：移码只能用于表示整数**

2.2 各种码的作用

模运算的性质

带余除法——$x, m \in Z, m > 0$ 则存在唯一决定的整数 q 和 r 使得：

$x = q * m + r$

加法或者减法运算：

$a + b$ 为 a+b 的补码

移位运算

原码的算术移位——符号位保持不变，仅对数值为进行移位

右移：高位补 0，低位舍弃。若舍弃的位 =0，则相当于/2 若舍弃的位不等于 0，则会舍弃精度。

左移：地位补 0，高位舍弃。若舍弃的位 =0，则相当于 *2，否则会出现严重误差。

反码的算术移位——正数的反码与原码相同，因此对正数反码的移位运算也和原码相同。

右移：高位补 0，低位舍弃

左移：低位补 0，高位舍弃

负数的移位：

右移：高位补 1，地位舍弃

左移：低位补 1，高位舍弃

补码的算术移位：

补码的算术移位——正数的补码与原码相同：因此对正数反码的移位运算也和原码相同

右移：高位补 0，低位舍弃左移：低位补 0，高位舍弃

负数补码 = 反码末尾 +1 导致反码最右边几个连续的 1 都因为进位而变为 0，直到进位碰到第一个 0 为止

规律：——负数补码中，最右边的 1 及其右边同原码。最右边的 1 的左边同反码

算术移位的应用举例: 快速幂

逻辑移位:

逻辑右移：高位补 0，低位舍弃逻辑左移：低位补 0，高位舍弃

逻辑移位的应用举例:RGB

循环移位: CF 位来存储进位：在左移或者右移时补上。

循环移位的应用举例：高低字节的转换

2.3 加减运算和溢出判断

加法器直接对原码进行加法运算，可能出错

在这里阐述一下基本的原理是什么，首先对于一个负数取补码相当于把它变为了自己的补数

假设我们求解 15 - 10 , 此时 15 还是 15, 而-10 则会变为 $256-10=246$, 原来的往后移动 10 位，等价于 15 + 246 以后移动的位置，思想和钟表差

不多，这个时候我们变回原码就可以了。

对于补码来说，无论加法还是减法，最后都会转变位加法，由加法器实现运算，符号位也参与运算。

溢出判断

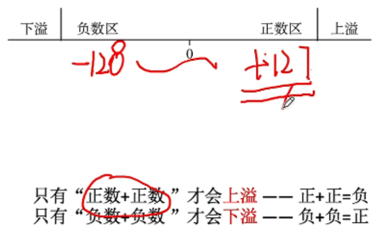

图 1: 溢出判断

具体判断后面再去看

2.4 原码的乘法运算

手动乘法的本质在于 r 进制，具体内容不再赘述。

同理得到手动二进制乘法

设机器字长位 n+1=5 位，含有 1 位符号位，$[x]$ 的原码为 1.1101, y 的原码为 0.1011，采用原码一位乘法求 $x * y$。

原码乘法

2.5 补码的一位乘法运算

设机器字长为 5 位 (含有 1 位符号位, n=4), x = -0.1101, y = +0.1011, 采用 Boooth 算法求 x*y，x 的补码为 1.0011,-x 的补码为 0.1101,y 的补码为 0.1011

补码的一位乘法：进行 n 轮加法，移位，最后再多来一次加法。

符号位参与运算

辅助位 - MQ 中最低位 =1 时，（ACC）+ x 的补码

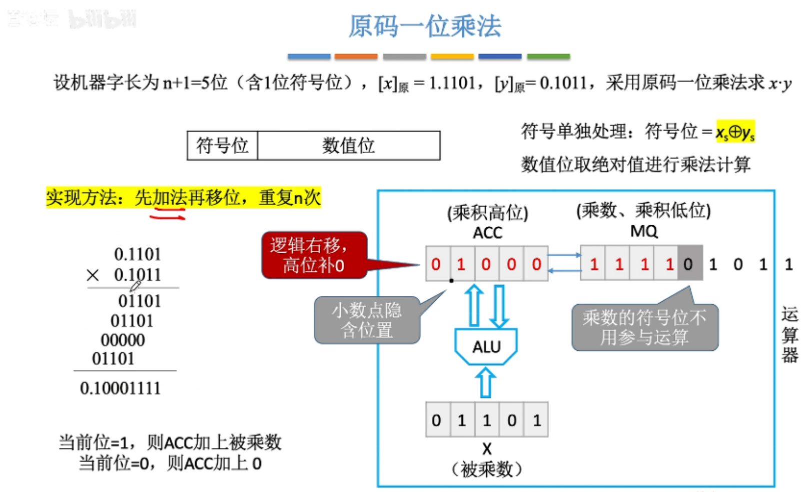

图 2: 原码乘法

辅助位 - MQ 中最低位 =0 时，（ACC）+0

辅助位 - MQ 中最低位 =-1 时，（ACC）+ -x 的补码

补码乘法 手工模拟计算

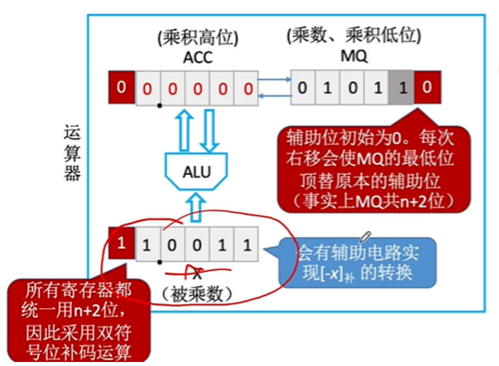

图 3: 补码乘法

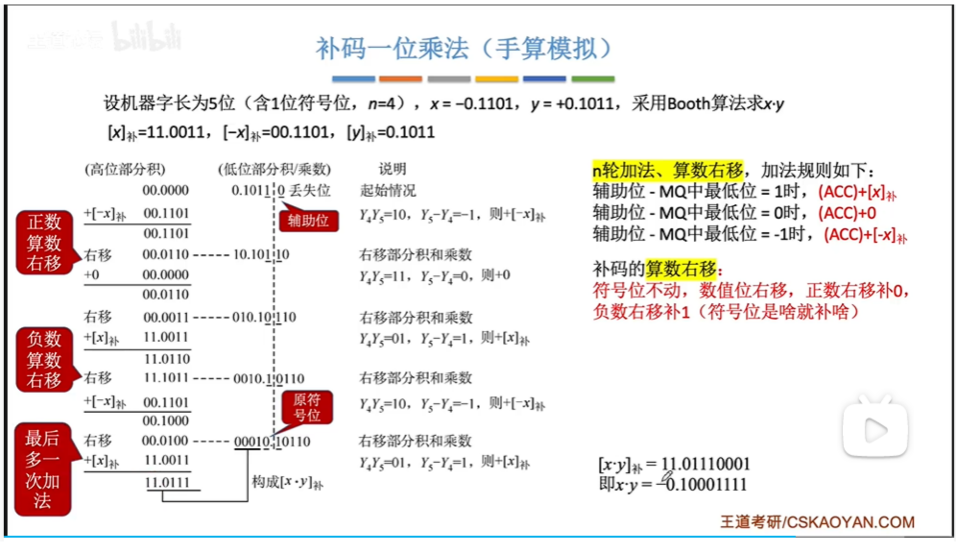

图 4: 模拟计算乘法

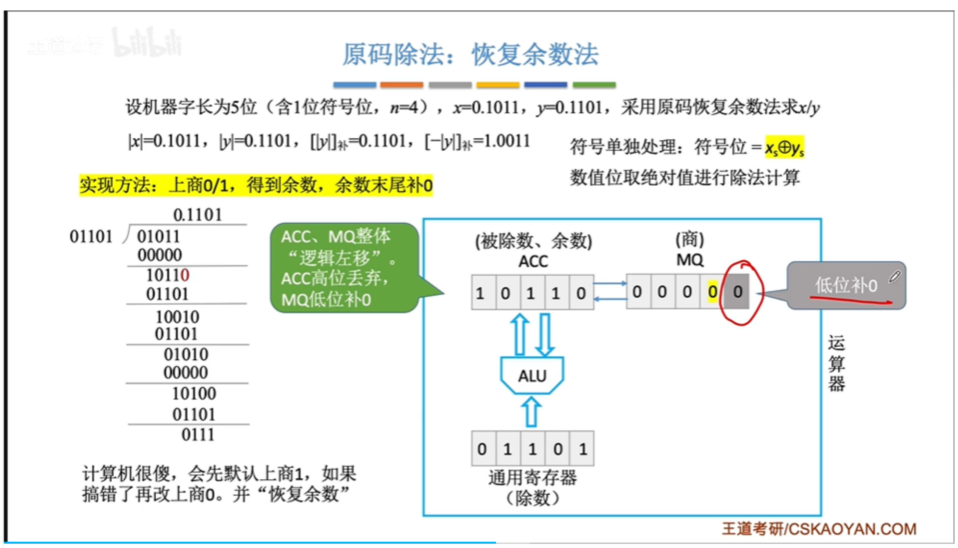

图 5: 原码除法

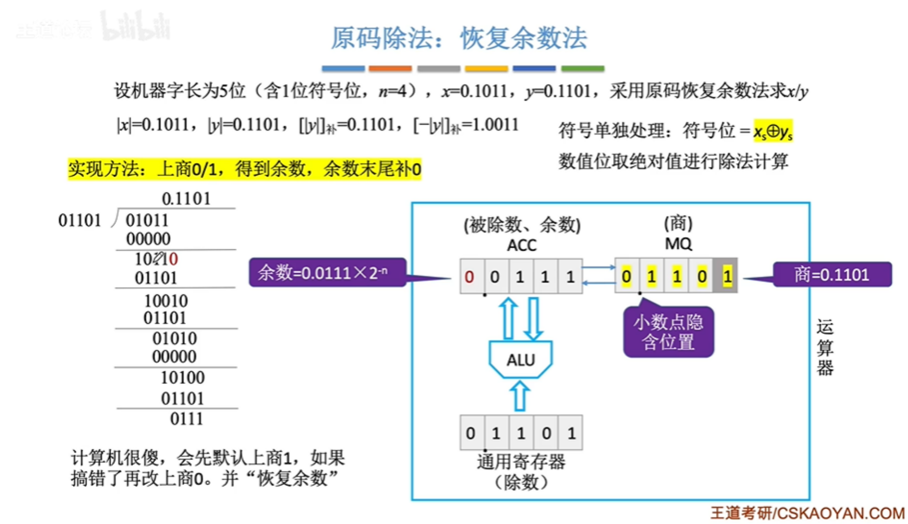

图 6: 原码除法 1

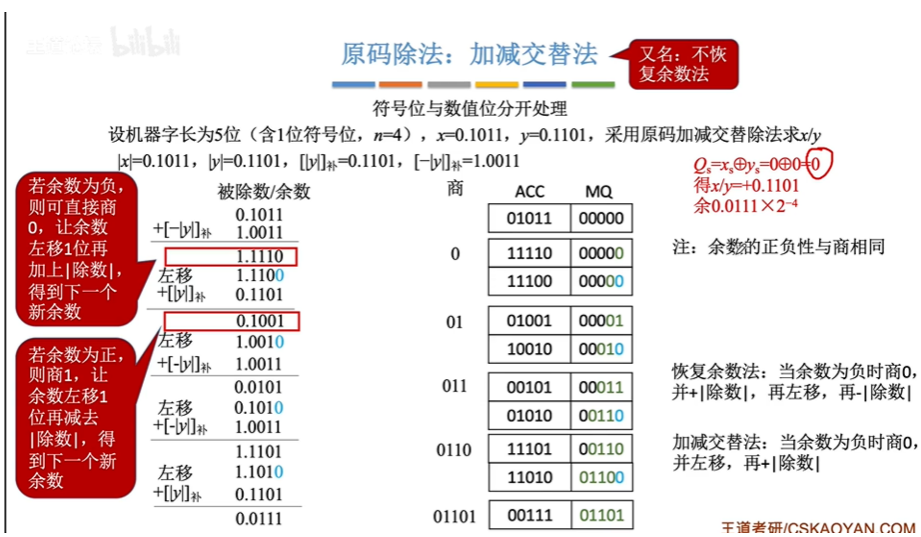

图 7: 加减交替法

2.6 原码的除法如上图

2.7 数据存储与排列

多字节数据在内存里面一定是占连续的几个字节。

分为大端方式和小端方式：

大端方式–便于人类阅读从高到低

小端方式–便于机器处理从低到高

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。通常也支持安字，按半字，按字节寻址。

假设存储字长为 32 位，则一个字 =32bit，半字 =16bit。每次访存只能读/写 1 个字。

现代计算机分为边界对齐和边界不对齐的方式。

边界对齐只需要一次访存（空间换时间）

边界不对齐效率相对较低。

2.8 浮点数的表示和运算

定点数的局限性：

比如我的财富位-8540，可以用 2B 定点整数 short 即可表示。

马云的财富：10000000000000

4B 定点整数 int.... 都表示不了

定点数可表示的数字范围有限，但是我们不能无限制地增加数据。

从科学计数法理解浮点数：

普通计数法：302657264536

科学计数法：

$+3.026 * 10^{11}$ 可以表示为 $+11 + 3.026$

浮点数的真值：$N = r^E * M$

阶码：常用补码或者移码表示的定点整数

尾数：常用原码或者补码表示的定点小数

例：阶码，尾数均用补码表示，求 a，b 的真值 $a = 0, -1; 1.1001$ $b = 0, 10; 0.001001$ a: 阶码 0,01 对应真值 +1，尾数 1.1001 对应真值-0.0111 $= -(2^{-2} + 2^{-3} + 2^{-4})$ a 的真值 =$2^1 * (-0.0111)$ = -0.111

规格化浮点数：规定尾数的最高位数值位必须是一个有效值

左规：当浮点数运算的结果位非规格化时要进行规格化处理。将尾数算数左移 1 位，阶码减 1

右规：当浮点数运算的结果尾数出现溢出（双符号位为 01 或者 10）时，将尾数算术右移一位，阶码 +1

规格化浮点数的特点：1. 用原码表示的尾数进行规格化正数用 $0.1****$ 的形式，其最大值的表示为 0.111111，最小值表示为 0.10...0。尾数的表示范围为 $1/2 <= M <= (1 - 2^{-n})$.

负数的表示范围为 $1.1*****$ 的形式，其最大值表示为 1.100...0, 最小值尾数的表示范围为 $-(1 - 2^{-n}) <= M <= -1/2$。

用**补码**的不再赘述。

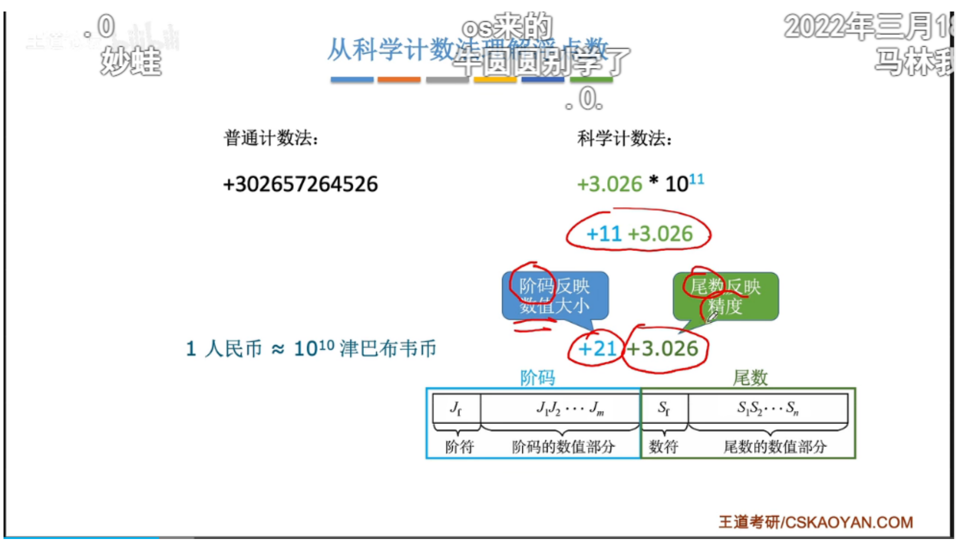

图 8: 从科学计数法理解浮点数