1.1 计算机的发展

什么是计算机系统

计算机系统=硬件+软件

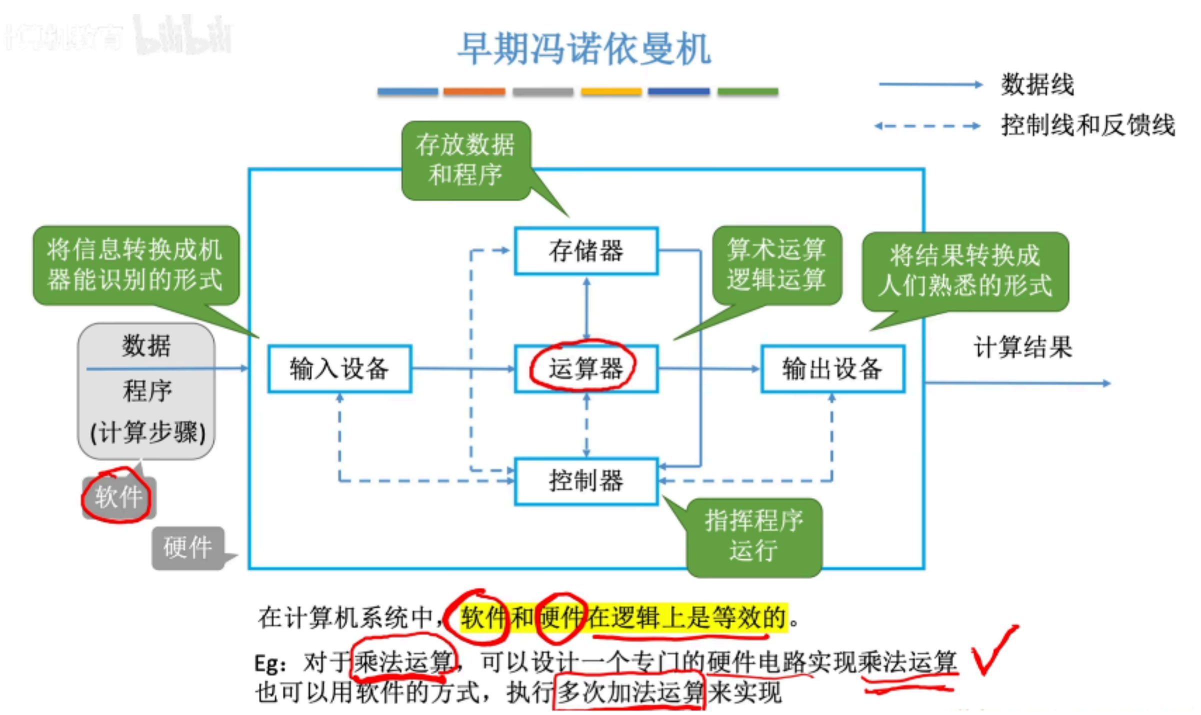
软件：

- 系统软件
- 应用软件

1.2 计算机硬件的基本组成

早期冯诺依曼机

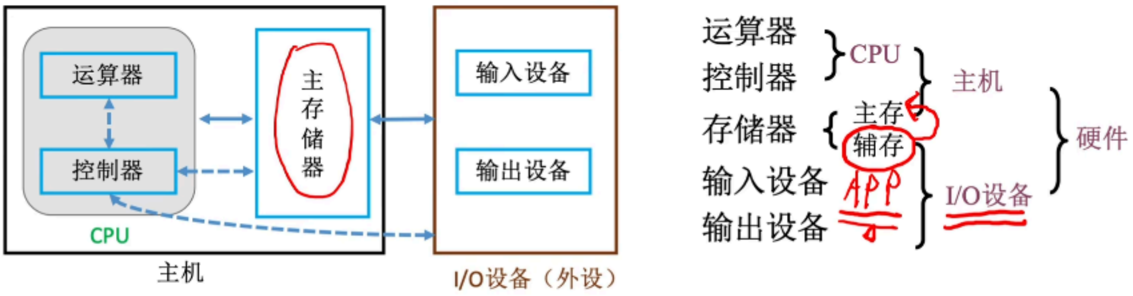
"存储程序 "的概念是指**将指令以二进制代码的形式事先输入计算机的主存储器** 然后按其在存储器中的首地址执行程序的第一条指令，以后就按该程序的规定顺序行其他指令，直至程序执行结束



特点：

- 计算机由五大部件组成
- 指令和数据以同等地位存于存储器，可按地址寻访
- 指令和数据用二进制表示
- 指令由操作码和地址码组成
- 存储程序
- **以运算器为中心**

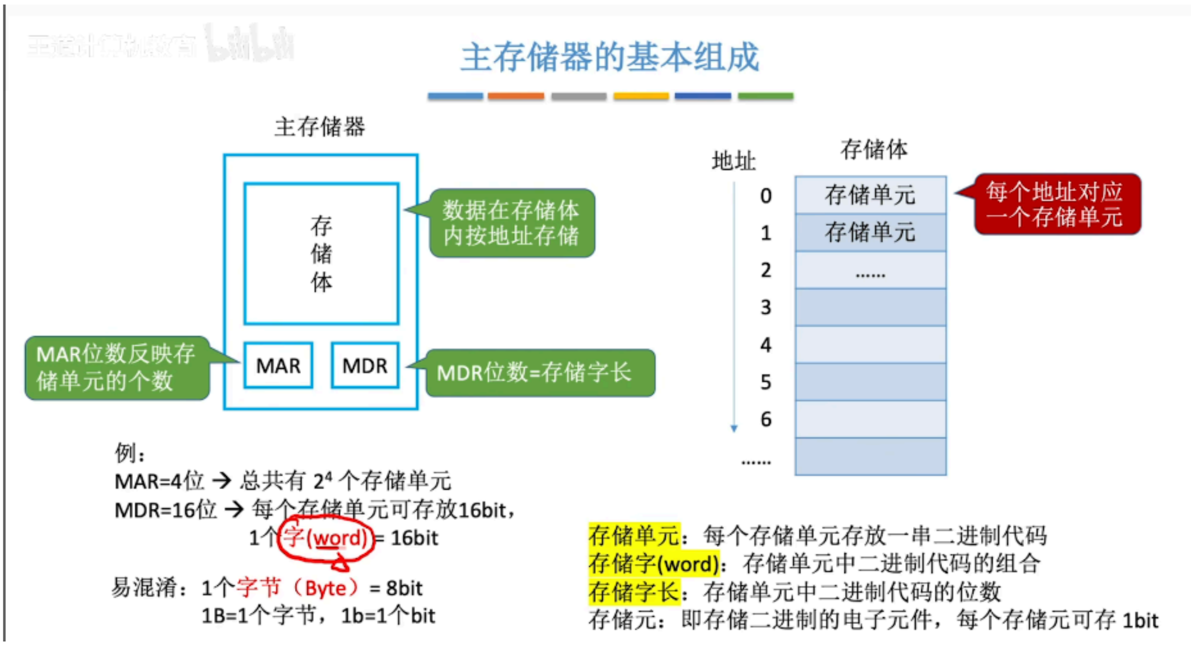
现代计算机结构



以存储器为核心

1.3 各个硬件的工作原理

主存储器的基本组成



MAR:储存数据

MDR：储存地址

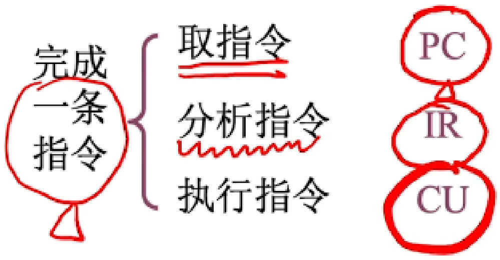
运算的基本组成

3个寄存器+1个核心部件ALU

ALU：算术逻辑单元，通过内部电路实现算术运算和逻辑运算

控制器的基本组成

核心部件：CU：控制单元，分析指令，给出控制信号



1.4 计算机性能指标

存储器的性能指标

MAR位数反应储存单元的个数

MDR位数=存储字长=每个存储单元的大小

总容量=存储单元个数`*`存储字长 bit=存储单元个数`*`存储字长 Byte

$$2^{10}:K$$

$$2^{20}:M$$

$$2^{30}:G$$

$$2^{40}:T$$

CPU的性能指标

CPU主频： CPU内数字脉冲信号振荡的频率

CPU时钟周期

CPU主频（时钟频率）=1/CPU时钟周期

CPI：每一条指令执行所需要的时钟周期

执行一条指令的耗时=CPI*CPU时钟周期

CPU执行时间（整个程序的耗时）=CPU时钟周期/主频=（指令条数*CPI）/主频

IPS：每秒执行多少条指令

问：若A、B两个CPU的平均CPI相同，那么A一定更快吗？
也不一定，还要看指令系统，如A不支持乘法指令，只能用多次加法实现乘法；而B支持乘法指令。

2.1 数据的表示和运算

真值和机器数

真值：符合人类习惯的数字

机器数：数字实际存到机器里的形式，正负号需要被“数字化”

BCD码

快速转换——对应

eg：985->1001 1000 0101

8421码，2421码，余三码

余三码：8421码+（0011）D

无符号整数的表示和运算

无符号整数的表示

无符号整数的表示有范围 0~2的n次方-1

无符号整数的运算

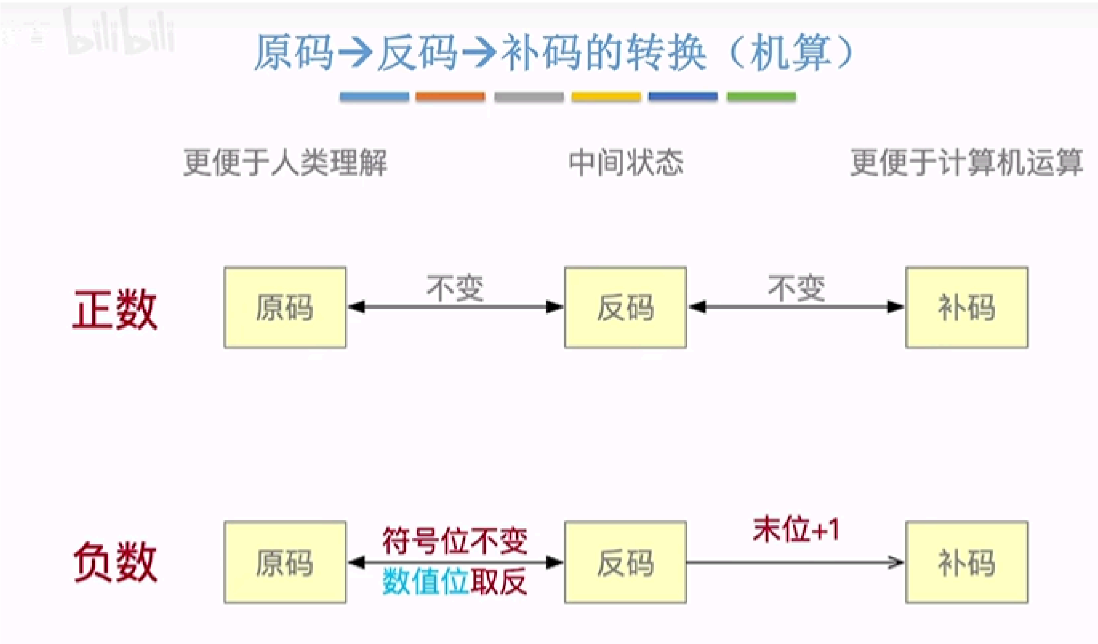
- 被减数不变，减数全部按位取反，末位+1，减法变加法
- 按位相加

带符号整数的表示和运算

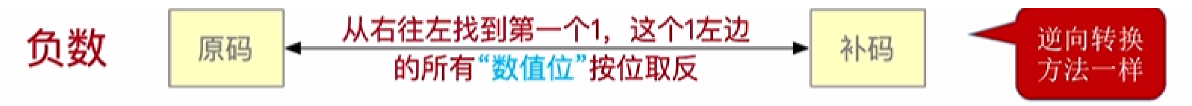
原码： ① 符号位“0/1”对应“正/负”，剩余的数值位表示真值的绝对值

② 若机器字长n+1位，带符号整数的原码表示范围：$-(2^n-1) \leqslant x \leqslant 2^n-1$

③ 真值0有两种形式：+0 和 −0，$[+0]_{原}=0,0000000$；$[-0]_{原}=1,0000000$



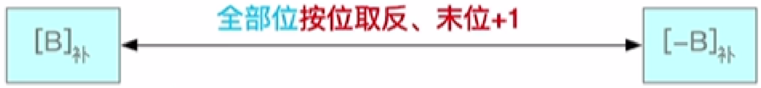
计算补码快速方法



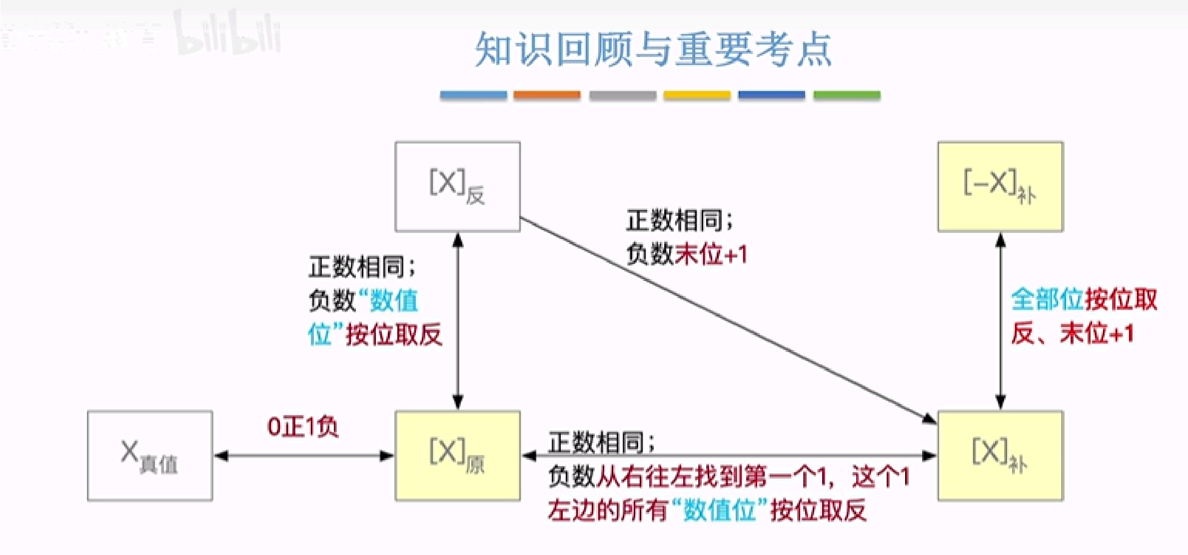
补码的减法

A-B=A+(-B)　　　　$[A]_{补}-[B]_{补}=[A]_{补}+[-B]_{补}$

接下来要解决的问题：已知“减数”的补码，如何求其负值的补码表示？



总结：



n+1 bit	合法表示范围	最大的数	最小的数	真值0的表示
带符号整数:原码	$-(2^n-1) \le x \le 2^n-1$	**0**,111...111 = 2^n-1	**1**,111...111 = $-(2^n-1)$	$[+0]_{原}$ =**0**,000...000 $[-0]_{原}$ =**1**,000...000
带符号整数:反码	$-(2^n-1) \le x \le 2^n-1$	**0**,111...111 = 2^n-1	**1**,000...000 = $-(2^n-1)$	$[+0]_{反}$ =**0**,000...000 $[-0]_{反}$ =**1**,111...111
带符号整数:补码	$-2^n \le x \le 2^n-1$	**0**,111...111 = 2^n-1	**1**,000...000 = -2^n	$[0]_{补}$ =**0**,000...000 真值0只有一种补码
无符号整数	$0 \le x \le 2^{n+1}-1$	1111...111 = $2^{n+1}-1$	0000...000 = 0	0000...000

原码和反码的合法表示范围完全相同，都有两种方法表示真值0
补码的合法表示范围比原码多一个负数，只有一种方法表示真值0

注意补码的范围，反码补码最小数的表示方法

反码和原码的真值0有两种表示方式

带符号整数移码表示

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

真值0:10000000　、、。。“”“”“”“”“”；。、0o

若机器字长n+1位，移码整数的表示范围：
$-2^n \le x \le 2^n-1$（与补码相同）

移码真值是递增的，方便比较大小

带符号整数:移码	$-2^n \le x \le 2^n-1$	1111...111 $= 2^n-1$	0000...000 $= -2^n$	$[0]_{移}$ =1000...000 真值0只有一种移码

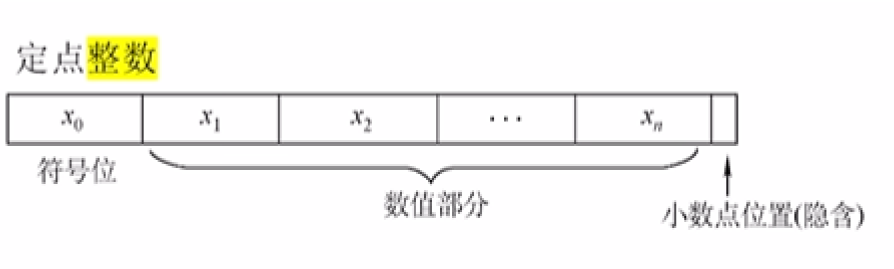
移码可以多表示一个负数

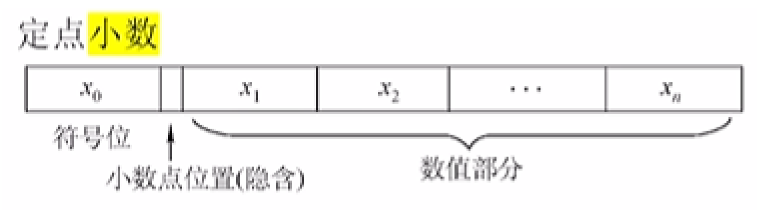
定点小数

定点：小数点固定

定点整数表示：原码、反码、补码、移码

定点小数表示：原码、反码、补码





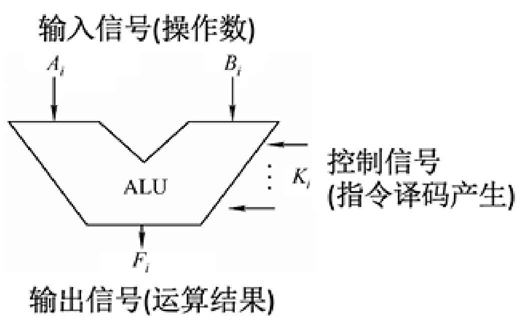
定点小数:原码	$-(1-2^{-n}) \le x \le 1-2^{-n}$	0,111...111 $= 1-2^{-n}$	1,111...111 $= -(1-2^{-n})$	$[+0]_{原}$ =0,000...000 $[-0]_{原}$ =1,000...000
定点小数:反码	$-(1-2^{-n}) \le x \le 1-2^{-n}$	0,111...111 $= 1-2^{-n}$	1,000...000 $= -(1-2^{-n})$	$[+0]_{反}$ =0,000...000 $[-0]_{反}$ =1,111...111
定点小数:补码	$-1 \le x \le 1-2^{-n}$	0,111...111 $= 1-2^{-n}$	1,000...000 $= -1$	$[0]_{补}$ =0,000...000 真值0只有一种补码

奇偶校验码

- **奇校验码**：整个校验码（有效信息位和校验位）中"1"的个数为奇数。
- **偶校验码**：整个校验码（有效信息位和校验位）中"1"的个数为偶数。

偶校验的硬件实现：各信息进行异或运算，得到的即为偶校验位

算数逻辑单元（ALU）



溢出判断

- 方法一：采用一位符号位

设A的符号为A_S，B的符号为B_S，运算结果的符号为S_S，则溢出逻辑表达式为

$$V = A_S B_S \overline{S_S} + \overline{A_S}\,\overline{B_S} S_S$$

若V=0，表示无溢出；
若V=1，表示有溢出。

- 方法二：最高位和符号位同时进位则未溢出
- 方法三：采用双符号位

正数符号为00，负数为11

记两个符号位为$S_{S1}S_{S2}$，则$V=S_{S1}\oplus S_{S2}$
若V=0，表示无溢出；若V=1，表示有溢出。

双符号位又称模4补码，单符号位又称模2补码

符号拓展

- 定点整数的符号扩展：
- 在原符号位和数值位**中间**添加新位，正数都添0；负数原码添0，负数反、补码添1
- 定点小数的符号扩展：
- 在原符号位和数值位**后面**添加新位，正数都添0；负数原、补码添0，负数反码添1

标志位的生成

OF：溢出标志（仅在有符号的加减运算中有含义）

CF：进位/借位标志，进位/借位时置1，否则置0（只对无符号加减法有意义）

$$\mathrm{CF} = \text{最高位产生的进位} \oplus \mathrm{sub} - \begin{cases} \mathrm{sub}=1\text{表示减法} \\ \mathrm{sub}=0\text{表示加法} \end{cases}$$

SF：符号标志。结果为负置1，否则置0

ZF：零标志，运算结果为0时ZF置1，否则为0

移位计算

原码的算数移位--符号位不变，仅对数值位进行移位

算数移位：

左移相当于`*`基数，右移相当于`/`基数

右移：高位补0，低位舍弃，若舍弃的位！=0，则会丢失精度

左移：低位补0

反码的算数移位

右移：高位补1，低位舍弃

左移：低位补1，高位舍弃

逻辑移位

右移高位补0低位舍弃

循环移位

溢出的位会补到另一侧

原码的乘法运算

符号单独处理：符号位 = $x_s \oplus y_s$

数值位取绝对值进行乘法计算

核心方法：先加法再移位（逻辑右移），重复n次

具体方法参考下图

	(高位部分积)	(低位部分积/乘数)	说明
	00.0000	1011 丢失位	起始情况
$+\lvert x\rvert$	00.1101		$C_4=1$, 则$+\lvert x\rvert$
	00.1101		
右移	00.0110	1101 1	右移部分积和乘数
$+\lvert x\rvert$	00.1101		$C_4=1$, 则$+\lvert x\rvert$
	01.0011		
右移	00.1001	1110 11	右移部分积和乘数
+0	00.0000		$C_4=0$, 则+0
	00.1001		
右移	00.0100	1111 011	右移部分积和乘数
$+\lvert x\rvert$	00.1101		$C_4=1$, 则$+\lvert x\rvert$
	01.0001		
右移	00.1000	1111 1011	右移部分积和乘数 乘数全部移出

结果的绝对值部分

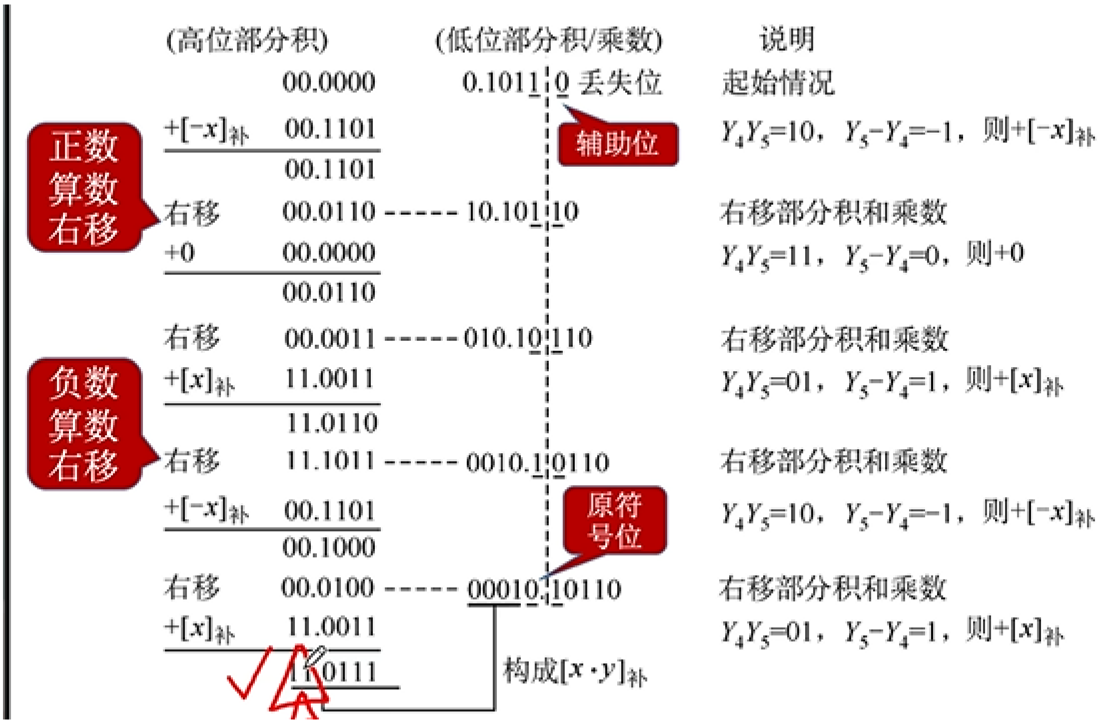
补码的乘法运算

辅助位 - MQ中最低位 = 1时，$(ACC)+[x]_{补}$
辅助位 - MQ中最低位 = 0时，$(ACC)+0$
辅助位 - MQ中最低位 = -1时，$(ACC)+[-x]_{补}$

核心方法：先加法再移位，重复n次，**最后多来一次加法**

辅助位初始为0。每次右移会使MQ的最低位顶替原本的辅助位

Booth算法（双符号位）

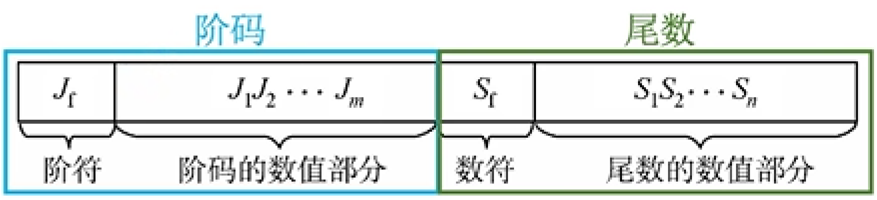


补码的右移为**算数右移**：符号位不动，数值位右移，符号位是什么就补什么

最后多进行一次加法（符号位参与运算），只有加法没有移位

浮点数的表示和运算

浮点数的表示（类比科学计数法）



- 阶码：常用补码或移码表示的定点整数
- 尾数：常用原码或补码表示的定点小数

$$\text{浮点数的真值：}\quad N = r^E \times M$$

r是阶码的底，通常为2，M是尾数，E是阶码

浮点数的规格化

规格化浮点数：规定尾数的最高数值位必须是一个有效值

左规：尾数左移一位，阶码减一

右规：尾数算数右移一位，阶码加一

表示范围

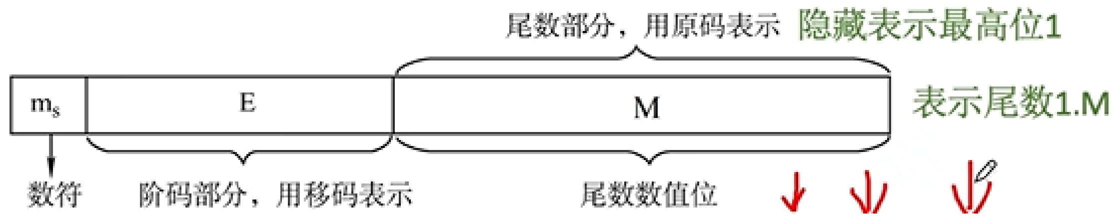
原码表示的尾数规格化：尾数的最高数值位必须是1

补码表示的尾数规格化：尾数的最高数值位必须和尾数符号位相反

1. 用原码表示的尾数进行规格化：
正数为0.1××…×的形式，其最大值表示为0.11…1；最小值表示为0.10…0。
尾数的表示范围为$1/2 \le M \le (1-2^{-n})$。
负数为1.1××…×的形式，其最大值表示为1.10…0；最小值表示为1.11…1。
尾数的表示范围为$-(1-2^{-n}) \le M \le -1/2$。

2. 用补码表示的尾数进行规格化：
正数为0.1××…×的形式，其最大值表示为0.11…1；最小值表示为0.10…0。
尾数的表示范围为$1/2 \le M \le (1-2^{-n})$。
负数为1.0××…×的形式，其最大值表示为1.01…1；最小值表示为1.00…0。
尾数的表示范围为$-1 \le M \le -(1/2+2^{-n})$。

IEEE 754



注意隐含位1

	类型	数符	阶码	尾数数值	总位数	偏置值	
						十六进制	十进制
float（单精度浮点型）	短浮点数	1	8	23	32	7FH	127
double（双精度浮点型）	长浮点数	1	11	52	64	3FFH	1023
long double	临时浮点数	1	15	64	80	3FFFH	16383

注意偏置值为127

阶码真值范围为-126~127（全1和全0用作特殊用途）

规格化的短浮点数的真值为：$(-1)^s \times 1.M \times 2^{E-127}$

阶码真值=移码-偏移量

当做无符号数进行加减

最小绝对值：尾数全为0，阶码真值最小-126，对应移码机器数 0000 0001
此时整体的真值为 $(1.0)_2 \times 2^{-126}$

当阶码E全为0，尾数M不全为0时，表示非规格化小数 $\pm(0.xx...x)_2 \times 2^{-126}$

阶码真值固定视为-126

当阶码E全为0，尾数M全为0时，表示真值 ±0

当阶码E全为1，尾数M全为0时，表示无穷大 ±∞

浮点数的加减运算

1. 对阶

   小阶向大阶靠齐

   ① 求阶差：$[\Delta E]_{补}$=11011+00100=11111，知ΔE=−1
   ② 对阶：X：11011,11.011000000 → 11100,11. 101100000　　$X = -0.0101 \times 2^{-100}$

2. 尾数加减
3. 规格化
4. 舍入
5. 判溢出

舍入

舍入方法存在不同，需根据题目灵活判断

“0”舍“1”入法：类似于十进制数运算中的“四舍五入”法，即在尾数右移时，被移去的最高数值位为0，则舍去；被移去的最高数值位为1，则在尾数的末位加1。这样做可能会使尾数又溢出，此时需再做一次右规。

恒置“1”法：尾数右移时，不论丢掉的最高数值位是“1”还是“0”，都使右移后的尾数末位恒置“1”。这种方法同样有使尾数变大和变小的两种可能。