chapter3 计算机的数据表示

> 一切指令和数据都用 01 来表示,阴阳交汇构筑了变化万千的世界

请大家务必注意题目中数据的进制,别把十六进制看成十进制

推荐一个做题工具 [锤子在线]

整数

进制转换

> 虽然人类钟爱十进制，但是冷酷的逻辑是是非分明的

怎么还考进制转换的

二进制	八进制	十进制	十六进制
bin	oct	dec	hex
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	10	a
1011	13	11	b
1100	14	12	c
1101	15	13	d
1110	16	14	e
1111	17	15	f

原码、反码与补码

> 抛开不知所谓的转化方法，这本质上是为了表示负数和减法

这里先给出原码反码补码长什么样：

digital	sign magnitude	1's complement	2's complement
-4	???	???	100
-3	111	100	101
-2	110	101	110
-1	101	110	111
-0	100	111	000
+0	000	000	000
+1	001	001	001
+2	010	010	010
+3	011	011	011

我们要表示正数和负数，最简单的想法是：
原码（Sign magnitude）：用 0 开头表示正数，用 1 开头表示负数

计算机是很擅长算加法的，大家想想行波加法器就知道了，但是如果用原码储存数据，我们是舒服了，计算机计算正数负数加法不就炸缸了？

那么很巧妙的，我们给所有 n 位数加上 2^n，上面的例子中，0 其实是 1000，那么 -3 是不是就是 1000-011 = 0101 呢？对于所有数字都加一个很好且足够大的数字，那么我们就能构造出 2's complement 了。

这样有什么好处呢？这个表示很自然的可以表示正数和负数的加法，并用加一个负数的方法表示减法。我们可以把它们拆成 2^n+x 或者 2^n-y 来看出这种方法的便利与正确性。

计算方法

反码为原码符号位不变后面全都取反，补码为反码加一。对于有 n 位数字位的数除了 2^n 只有补码之外，不需要考虑这个加一会不会溢出。

对于补码，计算相反数的方法为：原数取反加一。

范围：

原码表示法的范围是 $[-2^{n-1}+1, 2^{n-1}-1]$，其中 n 为数字位数（包含符号位）。

反码表示法的范围是 $[-2^{n-1}+1, 2^{n-1}-1]$，其中 n 为数字位数（包含符号位）。

补码表示法的范围是 $[-2^{n-1}, 2^{n-1}-1]$，其中 n 为数字位数（包含符号位）。

概念补充

biased notation（偏移表示法/移码）

就是把补码的符号位取反，目的是为了方便比大小

digital	biased notation	2's complement
+1011	11011	01011
-1011	00101	10101

sign extension（符号扩展）

为了保证补码表示数字不变

digital	sign extension
0101	000101
1101	111101

溢出

元件

加法器

加法器基本上数逻的时候就学过了，计组的时候补充了很多奇怪的优化方法感觉也不会考。

无符号数乘法

本质上都是根据乘数的每一位，做被乘数左移后与结果的加法

乘法器版本	被乘数位数	乘数需储存位数	计算单元位数	结果位数	优化方法
V1	128	64	128	128	/
V2	64	64	64	128	只要64位参与加法即可
V3	64	0	64	128	乘数最后一位可以丢弃，储存于结果中即可

1. 乘数保留在结果寄存器末位
2. 循环执行以下步骤：
    1. 取结果寄存器末位
    2. 若为 1 则在最高几位执行结果寄存器与被乘数的加法
    3. 无论如何都要右移一位

有符号数乘法

booth 算法

1. 乘数保留在结果寄存器末位，乘数被乘数均以补码形式

2. 记录刚刚丢出的一位，初始值为 0

3. 循环执行以下步骤：

    1. 取结果寄存器末位，结合丢出的一位
    2. 若为 10 则在最高几位执行结果寄存器与被乘数的减法，若为 10 则在最高几位执行加法
    3. 无论如何都要右移一位

13 * (-11) = - 143

01101 * 10101 = 11011 10001-->00100 01111

	step	Multiplicand	product
0	Initial Values	01101	00000 10101 0
1	1.c:10→Prod=Prod-Mcand	01101	10011 10101 0
	2: shift right Product	01101	11001 11010 1
2	1.b:01→Prod=Prod+Mcand	01101	00110 11010 1
	2: shift right Product	01101	00011 01101 0
3	1.c:10→Prod=Prod-Mcand	01101	10110 01101 0
	2: shift right Product	01101	11011 00110 1
4	1.d:01→Prod=Prod+Mcand	01101	01000 00110 1
	2: shift right Product	01101	00100 00011 0
	1.e:10→Prod=Prod-Mcand	01101	10111 00011 0
	2: shift right Product	01101	**11011 10001** 1

除法器

设计思路与乘法器类似

1. 被除数保留在结果寄存器末位

2. 循环执行以下步骤：

    1. 判断此时高位能不能做减法
    2. 若有剩余记录 1 并做减法否则不做
    3. 无论如何都要左移一位

3. 最后高位储存余数低位储存结果

Example 7/2 for Modified Division

- Well known numbers: 0000 0111/0010

iteration	step	Divisor	Remainder
0	Initial Values	0010	0000 0111
	Shift Rem left 1	0010	**0000 1110**
1	1.Rem=Rem-Div	0010	1110 1110
	2b: Rem<0 →+Div,sll R,R_0=0	0010	**0001 1100**
2	1.Rem=Rem-Div	0010	1111 1100
	2b: Rem<0 →+Div,sll R,R_0=0	0010	**0011 1000**
3	1.Rem=Rem-Div	0010	0001 1000
	2a: Rem>0 →sll R,R_0=1	0010	**0011 0001**
4	1.Rem=Rem-Div	0010	0001 0001
	2a: Rem>0 →sll R,R_0=1	0010	**0010 0011**
	Shift left half of Rem right 1		**0001 0011**

浮点数

浮点数的表示

规范化数

位数	S	exp	frac	bias	大小	指数范围	真数范围	范围
32	1	8	23	127	$(-1)^S \times (1+frac) \times 2^{exp-bias}$	[-126,127]	[0,1)	$(-2^{128}, -2^{-126}] \cup [2^{-126}, 2^{128})$
64	1	11	52	1023	$(-1)^S \times (1+frac) \times 2^{exp-bias}$	[-1022,1023]	[0,1)	$(-2^{1024}, -2^{-1022}] \cup [2^{-1022}, 2^{1024})$

非规范化数（Denormal numbers）

单精度	单精度	双精度	双精度	含义
exponent	fraction	exponent	fraction	
0	0	0	0	0
0	非0	0	非0	$(-1)^S \times (0+frac) \times 2^{1-bias}$ 用于表示很小的数
255	0	2047	0	∞，$x+\infty=\infty, \frac{x}{\infty}=0$
255	非0	2047	非0	NAN，表示除 0 等异常

nan(非数)

1. 任何数值与nan作运算，其运算结果均为nan
2. 对负数开平方之类的运算

3. infinity-infinity、infinity/infinity
4. infinity * 0、infinity/0、0/0
5. nan != nan、nan^0=1

infinity(无限大)

1. 任意正数N除以零，即N/0。
2. 任意正数N乘以无穷大，即infinity * N
3. infinity+infinity、infinity * infinity

原文链接：https://blog.csdn.net/UmbrellaCorporation/article/details/139937600

浮点数的计算

指数莫忘bias，真数莫忘1

加法

1. 十进制转为浮点表示
2. 指数小数字真数右移，指数增加，直至和大数字一样
3. 真数加法
4. 规范化，即真数转换为 $[1, 2)$
5. 检查溢出
6. 舍入
7. 转换为需要结果

乘法

1. 指数相加减去 bias
2. 乘真数
3. 规范化
4. 溢出
5. 舍入
6. 符号位

舍入

数字	向0舍入	向正无穷舍入	向负无穷舍入	四舍五入（向最近偶数舍入）
23.6	23	24	23	24
23.5	23	24	23	24
23.4	23	24	23	23
23.0	23	23	23	23
-23.3	-23	-23	-24	-23

溢出

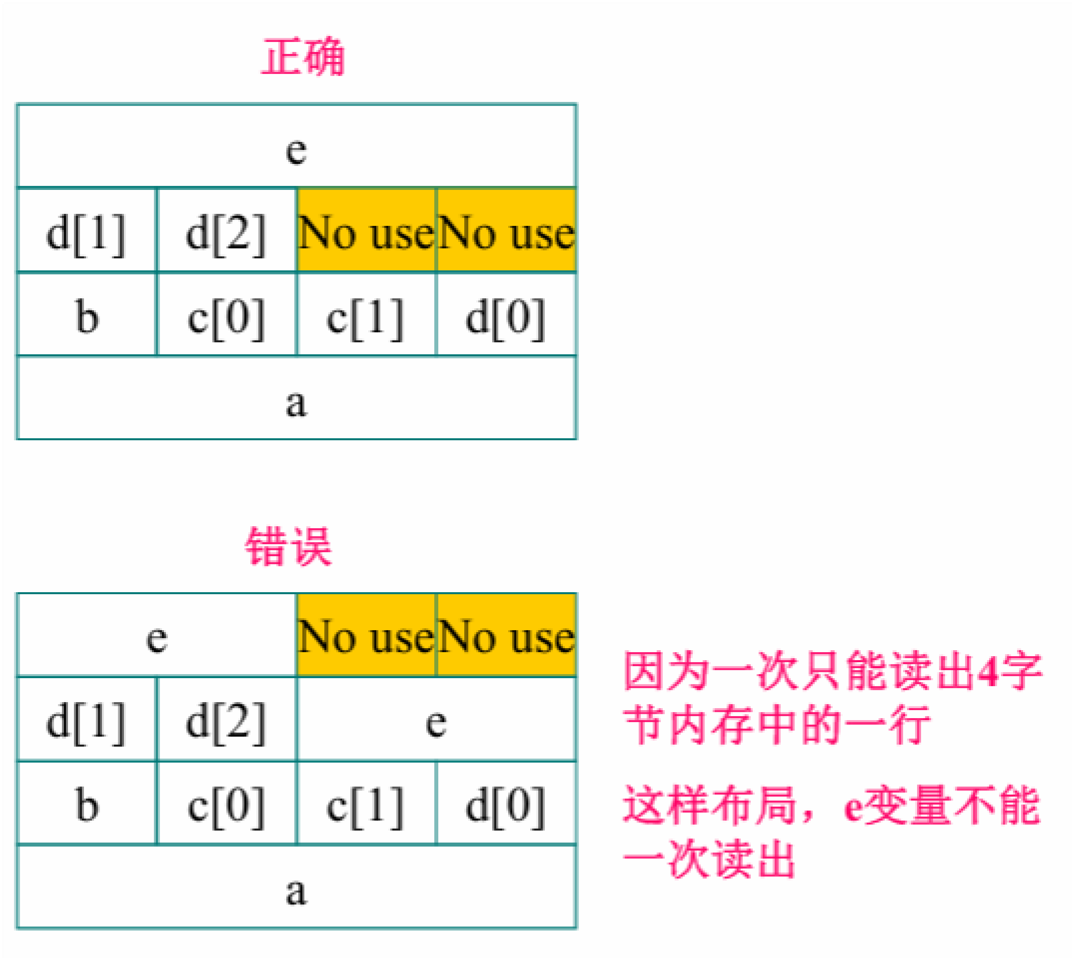
- Overflow: The number is too big to be represented
- Underflow: The number is too small to be represented

储存

对齐

例如，部分要求四个字节对齐

```
struct node{
 int a;
 char b;
 char c[2];
 char d[3]
 float e;
}
```



大小端

例如存储 12345678

存储方法	0x03 字节	0x02 字节	0x01 字节	0x00 字节
big endian	78	56	34	12
little endian	12	34	56	78

杂题选讲

1. 同一数据用原码反码补码各种浮点数表示大小比较

部分题目需要使用没有讲过的汇编语言，以下列出：

功能	汇编指令	解释
得到低位乘积	mul x5,x6,x7	x5=x6*x7取低一半位
得到高位乘积	mulh x5,x6,x7	x5=x6*x7取高一半位
有符号整除	div x5,x6,x7	x5=x6/x7，仅储存商
无符号整除	divu x5,x6,x7	x5=x6/x7，仅储存商
有符号取余	rem x5,x6,x7	x5=x6%x7，仅储存余数
无符号取余	remu x5,x6,x7	x5=x6%x7，仅储存余数
32位浮点加法	fadd.s f1,f2,f3	f1=f2+f3
64位浮点加法	fadd.d f1,f2,f3	f1=f2+f3

其余浮点数运算指令自己以此类推

$-7 \div 2 = -3 \cdots -1$
$-7 \div -2 = 3 \cdots -1$
$7 \div -2 = -3 \cdots 1$