**2.87 考虑下面两个基于 IEEE 浮点格式的 9 位浮点表示。

1. 格式 A
- 有一个符号位。
- 有 $k=5$ 个阶码位。阶码偏置量是 15。
- 有 $n=3$ 个小数位。

2. 格式 B
- 有一个符号位。
- 有 $k=4$ 个阶码位。阶码偏置量是 7。
- 有 $n=4$ 个小数位。

下面给出了一些格式 A 表示的位模式，你的任务是把它们转换成最接近的格式 B 表示的值。如果需要舍入，你要向 $+\infty$ 舍入。另外，给出用格式 A 和格式 B 表示的位模式对应的值。要么是整数（例如，17），要么是小数（例如，17/64 或 $17/2^6$）。

格式A		格式B	
位	值	位	值
1 01110 001	$\frac{-9}{16}$	1 0110 0010	$\frac{-9}{16}$
0 10110 101	208	0 1110 1010	208
1 00111 110	$\frac{-7}{2^{10}}$	1 0000 0111	$\frac{-7}{2^{10}}$
0 00000 101	$\frac{5}{2^{17}}$	0 0000 0001	$\frac{1}{2^{10}}$
1 11011 000	-4096	1 1111 0000	$-\infty$
0 11000 100	768	0 1111 0000	$+\infty$

*2.88 我们在一个 int 类型为 32 位补码表示的机器上运行程序。float 类型的值使用 32 位 IEEE 格式，而 double 类型的值使用 64 位 IEEE 格式。

我们产生随机整数 x、y 和 z，并且把它们转换成 double 类型的值：

```
/* Create some arbitrary values */
int x = random();
int y = random();
```

•2.88 我们在一个 int 类型为 32 位补码表示的机器上运行程序。float 类型的值使用 32 位 IEEE 格式，而 double 类型的值使用 64 位 IEEE 格式。

我们产生随机整数 x、y 和 z，并且把它们转换成 double 类型的值：

```
/* Create some arbitrary values */
int x = random();
int y = random();
int z = random();
/* Convert to double */
double   dx = (double) x;
double   dy = (double) y;
double   dz = (double) z;
```

对于下列的每个 C 表达式，你要指出表达式是否总是为 1。如果它总是为 1，描述其中的数学原理。否则，列举出使它为 0 的参数的例子。请注意，不能使用 IA32 机器运行 GCC 来测试你的答案，因为对于 float 和 double，它使用的都是 80 位的扩展精度表示。

A. (double)(float) x == dx

不总是为 1；float 型尾数只有 23 位，当阶码大于 23 时，float 类型就不能表示所有整数，但 double 型尾数有 52 位，当 $X = 2^{24}+1 = 16777217$ 时 X 转 float 会丢失精度变为 16777216.00000，再转 double 也还是 16777216.00000，而直接转 double 就不会丢失精度，依然还是 16777217.00000，所以 A 式不总为 1

B. dx + dy == (double) (x+y)

不总是为 1；

反例 $X = Y = 2^{31}-1$，dx+dy 不会溢出，但 x+y 会溢出得到负数，所以等式不会成立。

C. dx + dy + dz == dz + dy + dx

总是为 1；因为 double 可以表示 2^{53} 内所有整数，而 dx, dy, dz 分别由 x, y, z 转换而来，三者最大值为 $2^{31}-1$，即使三者相加也不会超过 2^{53}，不会丢失精度也不会溢出，所以服从交换律

D. dx * dy * dz == dz * dy * dx

不总是为 1；dx, dy, dz 相乘有可能超出 2^{53}，会造成舍入，运算顺序不同舍入的结果也有可能不同，

反例：X = 1804289383　Y = 846930886　Z = 1681692777

E. dx / dx == dy / dy

不总是为 1；反例：X 或 Y 等于 0

*2.71 你刚刚开始在一家公司工作，他们要实现一组过程来操作一个数据结构，要将4个有符号字节封装成一个32位unsigned。一个字中的字节从0（最低有效字节）编号到3（最高有效字节）。分配给你的任务是：为一个使用补码运算和算术右移的机器编写一个具有如下原型的函数：

```
/* Declaration of data type where 4 bytes are packed
   into an unsigned */
typedef unsigned packed_t;

/* Extract byte from word.  Return as signed integer */
int xbyte(packed_t word, int bytenum);
```

也就是说，函数会抽取出指定的字节，再把它符号扩展为一个32位int。

你的前任（因为水平不够高而被解雇了）编写了下面的代码：

```
/* Failed attempt at xbyte */
int xbyte(packed_t word, int bytenum)
{
    return (word >> (bytenum << 3)) & 0xFF;
}
```

FFFFFFFF
3 2 1 0

A. 这段代码错在哪里？

B. 给出函数的正确实现，只能使用左右移位和一个减法。

A: 没有进行符号位扩展

B:
```
int xbyte(packed_t word, int bytenum)
{
    return (word << ((3 - bytenum) << 3)) >> 24
```

**2.87 考虑下面两个基于IEEE浮点格式的9位浮点表示。

1. 格式A
- 有一个符号位。
- 有 $k=5$ 个阶码位。阶码偏置量是15。

**2.61 写一个C表达式，在下列描述的条件下产生1，而在其他情况下得到0。假设x是int类型。

A. x的任何位都等于1。

B. x的任何位都等于0。

C. x的最高有效字节中的位都等于1。

D. x的最低有效字节中的位都等于0。

代码应该遵循位级整数编码规则，另外还有一个限制，你不能使用相等（==）和不相等（!=）测试。

A：若x的任何位都为1，则按位取反后再做逻辑非运算会产生1

即 !(~x)

B：x的任何位都为0，则做逻辑非运算会产生1

即 !x

C：x的最高有效字节中的位都为1

取x最高有效字节：x>>24

检验是否为1：!(~(x>>24))

D：x的最低有效字节中的位都为0

取最低有效字节：x & 0xFF

检验是否为0：!(x & 0xFF)

综上，C表达式为：

(!(~x)) || (!x) || (!(~(x>>24))) || (!(x&0xFF))