格式A		格式B	
位	值	位	值
011 0000	1	0111 000	1
101 1110			
010 1001			
110 1111			
000 0001			

2.4.5 浮点运算

IEEE 标准指定了一个简单的规则，来确定诸如加法和乘法这样的算术运算的结果。把浮点值 x 和 y 看成实数，而某个运算⊙定义在实数上，计算将产生 $Round(x \odot y)$，这是对实际运算的精确结果进行舍入后的结果。在实际中，浮点单元的设计者使用一些聪明的小技巧来避免执行这种精确的计算，因为计算只要精确到能够保证得到一个正确的舍入结果就可以了。当参数中有一个是特殊值(如 -0、$-\infty$ 或 NaN)时，IEEE 标准定义了一些使之更合理的规则。例如，定义 $1/-0$ 将产生 $-\infty$，而定义 $1/+0$ 会产生 $+\infty$。

IEEE 标准中指定浮点运算行为方法的一个优势在于，它可以独立于任何具体的硬件或者软件实现。因此，我们可以检查它的抽象数学属性，而不必考虑它实际上是如何实现的。

前面我们看到了整数(包括无符号和补码)加法形成了阿贝尔群。实数上的加法也形成了阿贝尔群，但是我们必须考虑舍入对这些属性的影响。我们将 $x +^f y$ 定义为 $Round(x+y)$。这个运算的定义针对 x 和 y 的所有取值，但是虽然 x 和 y 都是实数，由于溢出，该运算可能得到无穷值。对于所有 x 和 y 的值，这个运算是可交换的，也就是说 $x +^f y = y +^f x$。另一方面，这个运算是不可结合的。例如，使用单精度浮点，表达式 (3.14+1e10)-1e10 求值得到 0.0——因为舍入，值 3.14 会丢失。另一方面，表达式 3.14+(1e10-1e10)得出值 3.14。作为阿贝尔群，大多数值在浮点加法下都有逆元，也就是说 $x +^f -x = 0$。无穷(因为 $+\infty - \infty = NaN$)和 NaN 是例外情况，因为对于任何 x，都有 $NaN +^f x = NaN$。

浮点加法不具有结合性，这是缺少的最重要的群属性。对于科学计算程序员和编译器编写者来说，这具有重要的含义。例如，假设一个编译器给定了如下代码片段：

```
x = a + b + c;
y = b + c + d;
```

编译器可能试图通过产生下列代码来省去一个浮点加法：

```
t = b + c;
x = a + t;
y = t + d;
```

然而，对于 x 来说，这个计算可能会产生与原始值不同的值，因为它使用了加法运算的不同的结合方式。在大多数应用中，这种差异小得无关紧要。不幸的是，编译器无法知道在效率和忠实于原始程序的确切行为之间，使用者愿意做出什么样的选择。结果是，编译器倾向于保守，避免任何对功能产生影响的优化，即使是很轻微的影响。

另一方面，浮点加法满足了单调性属性：如果 $a \geqslant b$，那么对于任何 a、b 以及 x 的值，除了 NaN，都有 $x+a \geqslant x+b$。无符号或补码加法不具有这个实数(和整数)加法的属性。

浮点乘法也遵循通常乘法所具有的许多属性。我们定义 $x *^f y$ 为 $Round(x \times y)$。这个运算在乘法中是封闭的(虽然可能产生无穷大或 NaN)，它是可交换的，而且它的乘法单位元