```
    } else if (x < _____) {
        /* Normalized result. */
        exp = _____;
        frac = _____;
    } else {
        /* Too big.  Return +oo */
        exp = _____;
        frac = _____;
    }

    /* Pack exp and frac into 32 bits */
    u = exp << 23 | frac;
    /* Return as float */
    return u2f(u);
}
```

*2.91　大约公元前250年，希腊数学家阿基米德证明了$\frac{223}{71}<\pi<\frac{22}{7}$。如果当时有一台计算机和标准库<math.h>，他就能够确定π的单精度浮点近似值的十六进制表示为0x40490FDB。当然，所有的这些都只是近似值，因为π不是有理数。

A. 这个浮点值表示的二进制小数是多少？

B. $\frac{22}{7}$的二进制小数表示是什么？**提示**：参见家庭作业2.83。

C. 这两个π的近似值从哪一位(相对于二进制小数点)开始不同的？

位级浮点编码规则

在接下来的题目中，你所写的代码要实现浮点函数在浮点数的位级表示上直接运算。你的代码应该完全遵循IEEE浮点运算的规则，包括当需要舍入时，要使用向偶数舍入的方式。

为此，我们把数据类型float-bits等价于unsigned：

```
/* Access bit-level representation floating-point number */
typedef unsigned float_bits;
```

你的代码中不使用数据类型float，而要使用float_bits。你可以使用数据类型int和unsigned，包括无符号和整数常数和运算。你不可以使用任何联合、结构和数组。更重要的是，你不能使用任何浮点数据类型、运算或者常数。取而代之，你的代码应该执行实现这些指定的浮点运算的位操作。

下面的函数说明了对这些规则的使用。对于参数f，如果f是非规格化的，该函数返回± 0(保持f的符号)，否则，返回f。

```
/* If f is denorm, return 0.  Otherwise, return f */
float_bits float_denorm_zero(float_bits f) {
    /* Decompose bit representation into parts */
    unsigned sign = f>>31;
    unsigned exp =  f>>23 & 0xFF;
    unsigned frac = f     & 0x7FFFFF;
    if (exp == 0) {
        /* Denormalized.  Set fraction to 0 */
        frac = 0;
    }
    /* Reassemble bits */
    return (sign << 31) | (exp << 23) | frac;
}
```

**2.92　遵循位级浮点编码规则，实现具有如下原型的函数：

```
/* Compute -f.  If f is NaN, then return f. */
float_bits float_negate(float_bits f);
```

对于浮点数f，这个函数计算$-f$。如果f是NaN，你的函数应该简单地返回f。

测试你的函数，对参数f可以取的所有2^{32}个值求值，将结果与你使用机器的浮点运算得到的结果