Nathan Tung
CS33 Dis 1C
April 16, 2012

CS33 Homework #2

2.83) `return ((ux<<1)==0 && (uy<<1)==0) || (sx && !sy) || (sx && sy && ux >= uy) || (!(sx||sy) && ux <= uy);`

Explanation: We return true if any of the following are true:

- Both fraction parts are equal to 0
- Only x has a signed bit in front
- Both are negative and unsigned part of x is greater than or equal to that of y
- Neither are negative and unsigned part of x is less than or equal to that of y

In all other cases, we return false. We accomplish this by using the logical OR operator.

2.88) A. **No**, casting an int into a float makes it lose precision, while first casting it into a double increases precision. This applies when x is, say, TMax.

B. **No**, having two doubles in addition can yield different answers than if two ints were first operated on, then cast into a double. (Intermediate integer answer might truncate precision.) For example, take y to have a value of TMin, which overflows and causes loss of precision.

C. **Yes**, doubles can be added commutatively – especially since each of dx, dy, and dz are between the values of TMin and TMax.

D. **No**, multiplication in different order can cause overflow and loss of precision, especially for large values. For example, take dx = TMax, dy = TMax-3, and dz = TMax-1.

E. **No**, we just let one of the values (say x and dx) be 0, and the other (say z and dz) be 1.

2.89)
```
float fpwr2(int x)
{
    /* Result exponent and fraction */
    unsigned exp, frac;
    unsigned u;
    if (x < -149) {
        /* Too small. Return 0.0 */
        exp = 0;
        frac = 0;
    } else if (x < -126) {
        /* Denormalized result */
        exp = 0;
        frac = 1 << (x+149);
    } else if (x < -128) {
        /* Normalized result. */
        exp = x+127;
        frac = 0;
    } else {
        /* Too big. Return +oo */
        exp = 255;
        frac = 0;
    }
    /* Pack exp and frac into 32 bits */
```

```
    u = exp << 23 | frac;
    /* Return as float */
    return u2f(u);
}
```

3.56)

```
1   int loop(int x, int n)
2   {
3       int result = -1;
4       int mask;
5       for (mask = 1; mask !=0; mask = (mask<<n)) {
6           result ^= (mask&x);
7       }
8       return result;
9   }
```

A. Which registers hold program values x, n, result, and mask?
→Value of x is held at %esi, n at %ebx, result at %edi, and mask at %edx.
B. What are the initial values of result and mask?
→Result is initially -1; mask is initially 1.
C. What is the test condition for mask?
→The test condition checks if mask is not equal to 0.
D. How does mask get updated?
→Mask is left-shifted by n bits.
E. How does result get updated?
→Result is XOR-ed with itself and the AND-ed product of mask and x.
F. Fill in all the missing parts of the C code.
→See above for bolded parts.