1.A 8000007F
800000000000007F

Rio Bacon
HW 1

B FFFFFF80
FFFFFFFFFFFFFF80

C FFFFFF81
FFFFFFFFFFFFFF81

2.A E: E = e – (2^(k-1) – 1)   e = 2^(k-1) - 2^2
= 2^(k-1) – 2^2 – (2^(k-1) – 1)
= 3
M: M = 2^-(k-1) + 2^-(k-2) + 1
f: f = 2^-(k-1) + 2^-(k-2)
V: V = 2^(2^(k-1) – 2^2 – (2^(k-1) – 1))*(2^-(k-1) + 2^-(k-2) + 1)
= (2^3)*(2^-(k-1) + 2^-(k-2) + 1)

In binary numbers, the left most bit is a 0, so it is positive. For the next k bits, the left most bit is 1, the second to right most bit is 1, and the rest 0s. For the final bits, the left 2 bits are 1, and the rest are 0s.

B E: E = e – (2^(k-1) – 1)
= n
M: M = (2^-k) * (2^k – 1) + 1
f: f = (2^-k) * (2^k – 1)
V: V = (2^n)* ((2^-k) * (2^k – 1) + 1)

In binary numbers, the left most bit is a 0, so it is positive. For the next k bits, the k bits will represent e such that e – (2^(k-1) – 1) will equal n. This is because 2^n will create the largest number that can be multiplied by M and become odd, as M will have a denominator of 2^n, and a numerator of 2^n – 1.To make a fraction of (2^n – 1)/(2^n), the final n bits must be all 1s.

2.88:

A: Always yields 1, because dx being a double, will be an exact representation of x. Because x and dx represent the exact same value, x and dx will get rounded to the same value as they become a float.

B: Always yields 1, because dx and dy being doubles, they precisely reflect x and y. The arithmatic subtraction performed do not cause rounding errors, because all four variables are precise values. Because of this, (double) (x – y) and dx – dy will always yield the same results.

C: Always yields 1, because addition or subtraction do not require rounding, precision is not lost at any point and therefore is associative.

D: Always yields 1, because due to its original nature of int, it will not lose precision unless an overflow occurs. But overflow will occur regardless of association, because none of the double variables can hold a number between 0 and 1.

E: Always yields 1, because dx and dz come from x and z, which are ints and have exact values. Because they have exact value, dx / dx and dz / dz will both always evaluate to 1.

3.54:

```
int decode2(int x, int y, int z) {
        y = y – z;
        int w = y;
        w = w << 31;
        w = w >> 31;
        y = x * y;
        w = w ^ y;
        return w;
}
```

3.68:

E1(n) = 2n + 1
E2(n) = 3n