CSED211 Homework #1, Answer

1. Exercise 2.60 on page 164.

   Example code

```
unsigned replace_byte (unsigned x, int i, unsigned char b) {
        int itimes8 = i << 3;
        unsigned mask = 0xFF << itimes8;
        return (x & ~mask) | (b << itimes8);
}
```

2. Exercise 2.68 on page 168

   Example code

```
int lower_one_mask(int n) {
        return (2<<(n-1)) - 1;
}
```

   Does not work when n = 0 for the above code (meaningless).

   Think about why not using (1<<n)-1.

3. Exercise 2.73 on page 170

   Example code

```
int saturating_add(int x, int y) {
    int sum = x + y;
    int wm1 = (sizeof(int)<<3)-1;
    /* In the following, create "masks" consisting of all 1's when a condition is
    true, and all 0's when it is false */
    int xneg_mask = (x >> wm1);
    int yneg_mask = (y >> wm1);
    int sneg_mask = (sum >> wm1);
    int pos_over_mask = ~xneg_mask & ~yneg_mask & sneg_mask;
    int neg_over_mask = xneg_mask & yneg_mask & ~sneg_mask;
    int over_mask = pos_over_mask | neg_over_mask;
    /* Choose between sum, INT_MAX, and INT_MIN */
    int result =
        (~over_mask & sum) |
        (pos_over_mask & INT_MAX)|(neg_over_mask & INT_MIN);
    return result;
```

}

4. Exercise 2.83 on page 172

Example Answer

A. Letting V denote the value of the string, we can see that shifting the binary point k positions to the right gives a string y.yyyyyy---, which has numeric value Y + V , and also value V X 2^k. Equating these gives V = Y /(2^k - 1).
(Y + V = V X 2^k ➔ V X (2^k - 1) = Y ➔ V = Y /(2^k - 1))

B. (a) For y = 101, we have Y = 5, k = 3, V = 5/7
(b) For y = 0110, we have Y = 6, k = 4, V = 6/15 = 2/5
(c) For y = 010011, we have Y = 19, k = 6, V = 19/63.

5. Exercise 2.88 on page 174.

Format A		Format B		Comments
Bits	Value	Bits	Value	
1 01111 001	$\frac{-9}{8}$	1 0111 0010	$\frac{-9}{8}$	
0 10110 011	176	0 1110 0110	176	
1 00111 010	$\frac{-5}{1024}$	1 0000 0101	$\frac{-5}{1024}$	Norm $\rightarrow$ denorm
0 00000 111	$\frac{7}{131072}$	0 0000 0001	$\frac{1}{1024}$	Smallest positive denorm
1 11100 000	-8192	1 1110 1111	-248	Smallest number $> -\infty$
0 10111 100	384	0 1111 0000	$+\infty$	Round to ∞.

6. Exercise 2.94 on page 178

Example code

```
/* Compute 2*f. If f is NaN, then return f. */
float_bits float_twice(float_bits f) {
    unsigned sign = f>>31;
    unsigned exp = f>>23 & 0xFF;
    unsigned frac = f & 0x7FFFFF;
    if (exp == 0) {
        /* Denormalized. Must double fraction */
        frac = 2*frac;
        if (frac > 0x7FFFFF) {
            /* Result normalized */
            frac = frac & 0x7FFFFF; /* Chop off leading bit */
            exp = 1;
        }
```

```
    } else if (exp < 0xFF) {
        /* Normalized. Increase exponent */
        exp++;
        if (exp == 0xFF) {
            /* Infinity */
            frac = 0;
        }
    } else if (frac != 0) {
        /* NaN */
        return f;
    }
    /* Infinity does not require any changes */
    return (sign << 31) | (exp << 23) | frac;
}
```

7. For a single precision floating point number, it uses 32 bits. In fact, there are 2^32 different presentations possible using 32 bits. However, some presentations are not floating point number representation (ex. NaN). Find how many presentations are meaningful floating point number representation?

   All combination – infinity case - NaN = 2^32 – 2^24