1. Evaluate the following C expression on 32 bits (Please show work for partial credit):
   `(17 << 2) & (32 >> 3) | 5 ^ 7`

   ```
   17 = 00010001
   17 << 2 = 01000100 = 68

   32 = 00100000
   32 >> 3 = 00000100 = 4

   68 & 4 | 5 ^ 7

   68 & 4 = 01000100 & 00000100 = 00000100 = 4
   5 ^ 7 = 00000101 ^ 00000111 = 00000010 = 2

   4 | 2 = 00000100 | 00000010 = 00000110 = 6
   ```

2. Answer and explain the following:
   a. Determine the value of the following logical expressions:
      i. **`0U < -500`**
         `= 0 < 4294966796 = 1 = true`
      ii. **`2147483647 > (int)2147483648U`**
         `(int)2147483648U = Overflow`
         `This would result in the signed int = -2147483648`
         `2147483647 > -2147483648 = 1 = true`
   b. Determine the values of the following functions:
      i. $B2U_8$(1111 1101)
         = 253
      ii. $B2T_6$(011101)
         = 29
      iii. $B2T_{12}$(1111 1111 1101)
         = -3

3. Given the hexadecimal representation of a floating-point number as **0x8FF0C000**, find the equivalent decimal value of the number.

   0x8FF0C000 = 10001111 11110000 11000000 00000000

   1 00011111 11100001100000000000000

   Sign = 1 = -
   Exponent = 31 - 127 = -96
   Significant = 1.111000011 = 1.880859375

   = -(1.111000011 x 2^-96)
   Decimal = -2.37397828665e-29

4. Write code to implement the following function:

```
/*
* Generate mask indicating leftmost 1 in x. Assume w = 32.
* For example, 0xFF00 -> 0x8000, and 0x6600 -> 0x4000
* If x=0, then return 0.
*/
int leftmost_one(unsigned x) {
    x |= x >> 1;
    x |= x >> 2;
    x |= x >> 4;
    x |= x >> 8;
    x |= x >> 16;
    return x - (x >> 1);
}
```