Question 3.3–1: (Solution, p 3)

a. Give an example of an eight-bit number which, when arithmetically right-shifted one place, is *different from* the same number logically right-shifted one place.

b. Give an example of an eight-bit number which, when arithmetically right-shifted one place, is the *same as* the same number logically right-shifted one place.

Question 3.3–2: (Solution, p 3) Consider the following C program.

```
#include <stdio.h>

int mystery(int n, int i) {
    return (n >> i) & ~(-1 << i);
}

int main() {
    printf("%d %d %d %d\n", mystery(0xFF, 2), mystery(0xFF, 5),
        mystery(0x77, 3), mystery(0x02040608, 8));
    return 0;
}
```

What would this program print when run?

Question 3.3–3: (Solution, p 3) Consider the following C function.

```
int f(int x, int n) {
    return x | (1 << (n - 1));
}
```

a. What does `f(0, 2)` return?
b. What about `f(8, 2)`?
c. What about `f(f(0, 1), 2)`?

Question 3.3–4: (Solution, p 3) Without using a loop, write a C function that retrieves the `which`th bit from a number `num`. The `which` parameter should be between 0 and 31, where 0 represents the one's bit of the bit pattern, 1 represents the two's bit, and so forth. For example, `getBit(12, 2)` and `getBit(12, 3)` should return 1, while `getBit(12, 1)` and `getBit(12, 4)` would return 0.

```
int getBit(int num, int which) {



}
```

Question 3.3–5: (Solution, p 3) Without using loops or conditional statements, complete the following C function so that it returns the largest power of 2 that divides into its parameter value `n` exactly. Thus, `divisor_pow2(52)` would return 4, while `divisor_pow2(56)` would return 8.

```
int divisor_pow2(int n) {


}
```

Hint: You can find the largest power of 2 dividing into a number exactly by finding the rightmost bit of the number. For example, $52_{(10)} = 110100_{(2)}$ has its rightmost bit in the 4's place; $56_{(10)} = 111000_{(2)}$ has the rightmost bit in the 8's place.

Question 3.4–1: (Solution, p 3) Consider a 6-bit floating-point representation with a 3 bits for the excess-3 exponent and 2 bits for the mantissa.

a. How would $0.75_{(10)}$ be represented in this 6-bit representation?
b. What decimal value does 011010 represent?
c. What decimal value does 000010 represent?
d. How would infinity (∞) be represented in this representation?

Question 3.4–2: (Solution, p 3) Consider a 7-bit floating-point representation with a 3 bits for the excess-3 exponent and 3 bits for the mantissa.

a. What values do 1010100 and 00000100 represent? Express each answer as a decimal number or a base-10 fraction.

b. What is the bit pattern of the smallest positive normalized number supported by this representation? Convert this to a decimal fraction or number.

c. What is the bit pattern of the largest denormalized number supported by this representation? Convert this to a decimal fraction or number.

d. Suppose we add 0101010 and 1111000 as 7-bit floating-point numbers. What is the bit pattern of the result?

Question 3.4–3: (Solution, p 3) Give an example of three floating-point numbers x, y, and z, such that the distributive property $x(y + z) = xy + xz$ does not hold. (Feel free to describe the values rather than give numerical values: For example, you might say "the largest denormalized number" rather than give a particular value.) **Note:** Your answer should include the values of $x(y + z)$ and $xy + xz$ for your values of x, y, and z.

Question 3.4–4: (Solution, p 3) Give an example of three floating-point numbers x, y, and z such that the associative property of addition $x + (y + z) = (x + y) + z$ does not hold. (Feel free to describe the values rather than give numerical values: For example, you might say "the largest denormalized number" rather than give a particular value.) **Note:** Your answer should include the values of $x + (y + z)$ and $(x + y) + z$ for your values of x, y, and z.

Solution 3.3–1: (Question, p 1)

a. 11111111 (or any other sequence beginning with 1).

b. 00000000 (or any other sequence beginning with 0).

Solution 3.3–2: (Question, p 1) `3 7 6 6`

Solution 3.3–3: (Question, p 1)
a. 2
b. 10
c. 3

Solution 3.3–4: (Question, p 1)

```
int getBit(int num, int which) {
    return (num >> which) & 1;
}
```

Solution 3.3–5: (Question, p 1)

```
int divisor_pow2(int n) {
    return n & -n;
}
```

Solution 3.4–1: (Question, p 2)
a. 001010
b. $12.0_{(10)}$
c. $0.125_{(10)}$
d. 011100

Solution 3.4–2: (Question, p 2)
a. $-0.75_{(10)}$, $0.125_{(10)}$
b. 0001000, which converts to $1/4$ or 0.25
c. 0000111, which converts to $7/32$ or 0.2187
d. 1111000 (since anything added to $-\infty$ is $-\infty$)

Solution 3.4–3: (Question, p 2) One possibility is x = 0.5, y = largest possible number, and z = 1. In this case, $x(y+z)$ is infinity, while $xy+xz$ is a finite number.
Another possibility is $x = \infty$, $y = -1$, and $z = 1$. In this case, $x(y+z)$ is infinity (since $\infty \cdot 0 = \infty$), while $xy + xz$ is NaN (since $-\infty + \infty = \text{NaN}$).
While these answers are fine, they are somewhat dissatisfying because of their reliance on overflow. Another possibility, which does not resort to nonnumeric values, has x = 0.5, y = smallest possible number, and z = smallest possible number. In this case, $x(y+z)$ is the smallest possible number, while $xy+xz$ results in adding two numbers that are too small to represent, so we get 0.

Solution 3.4–4: (Question, p 2) Suppose $x = -2^{100}$, $y = 2^{100}$, and $z = 1$. Then

$$x + (y + z) = -2^{100} + (2^{100} + 1) = -2^{100} + 2^{100} = 0$$

($2^{100} + 1 = 2^{100}$ since the 1 can't be represented within the number's precision) and

$$(x + y) + z = (-2^{100} + 2^{100}) + 1 = 0 + 1 = 1$$