1.

Assume:

```
int x = rand();
int y = rand();
unsigned ux = (unsigned) x;
```

Are the following statements always true?

a.

`ux >> 3 == ux/8`

True

- For unsigned integers, right shifting always rounds towards 0, as all unsigned integers are non-negative and extra 1's on the right are discarded while right shifting.
- Thus, shifting to the right by 3 is equivalent to integer division by 2^3, which also rounds towards 0.

b.

given x > 0,

`((x << 5) >> 6) > 0`

False

- In the case where (x << 5) has a 1 for its most significant bit, right shifting by 6 will produce a negative number.

c.

`~x + x >= ux`

True

- ~x + x would be UMAX.

d.

given x & 15 == 11,

`( ~((x >> 3) & (x >> 2)) << 31) >= 0`

False

- The final comparison against 0 effectively checks if the most significant bit of the left hand sign is 0 or not.
- By the given statement, we know that the 4 least significant bits (lsb) of x are 1011. Thus (x >> 3) has a lsb of 1, while (x >> 2) has a lsb of 0.
- AND-ing the two together has a lsb of 0, which when negated is 1.
- Left-shifting by 31 thus results in a number with a most significant bit of 1, and the remaining bits being 0
- This is a negative number

e.

given ((x < 0) && (x + x < 0))
x + ux < 0
False

- In an addition of an unsigned integer with a signed integer, the signed integer is implicitly cast to unsigned.
- Thus, the addition of two unsigned integers will always be non-negative
  - This is regardless of the given

f.
given ((x < 0) && (y < 0) && (x + y > 0))
((x | y) >> 30) == -1
False

- Per the given, we know that the two most significant bits of x and y can be either 10 and 10, 11 and 10, or 10 and 11.
- In the case where x and y are 10 and 10, (x | y) would have most significant bits of 10
- In that case, Right shifting (x | y) by 30 would the result in -2

2. Write a function that, given a number n, returns another number where the k^{th} bit from the right is set to to 0.
Examples:
`killKthBit(37, 3) = 33` because 37_{10} = 100**1**01$_2$ ~> 100**0**01$_2$ = 33_{10}
`killKthBit(37, 4) = 37` because the 4^{th} bit is already 0.

```
int killKthBit(int n, int k) {

    return n & ~(1 << (k - 1));

}
```

3.
Given: x has a 4 byte value of 255
What is the value of the byte with the lowest address in a
255 is represented as 0x000000FF
a.
big endian system?
0x00
b.
little endian system?
0xFF

4. **Endianness**

a. Suppose we declared the following 4 byte int:

```
int x = 254;
```

and we stored this in memory location 0x100 on a little-endian system. What values would be stored in the following memory locations?

0x100	0x101	0x102	0x103
0xfe	0x00	0x00	0x00

b. Suppose we declared an array of ints:

```
int arr[] = {1, 2};
```

and we stored this in memory location 0x100 on a little endian system. What values would be stored in the following memory locations?

0x100	0x101	0x102	0x103	0x104	0x105	0x106	0x107
0x01	0x00	0x00	0x00	0x02	0x00	0x00	0x00

c. Suppose we declared a string:

```
char * s = "hello";
```

and we stored this in memory location 0x100 on a little endian system. What values would be stored in the following memory locations?

note: it's a good idea to get familiar with hex encodings of alphabetical characters, but for convenience, the hexadecimal encodings of the characters are: h (0x68), e (0x65), l (0x6c), and o (0x6f)

0x100	0x101	0x102	0x103	0x104	0x105
0x68	0x65	0x6c	0x6c	0x6f	0x00