Number Representation October 2, 2015
Given a binary string, what are all the ways it can be interpreted or converted to a human understandable form?

Rules:
1) For n bits, max value represented as unsigned number is 2^n -1
2) 2's complement: represent positive and negative numbers
    Negative number with the largest magnitude: 2^(n-1)
    Positive number with largest magnitude: 2^(n-1) – 1
    Split *almost* evenly between positive and negative numbers
    Ex. 5 bits: -16 to 15

Source Representation:
    Decimal: base 10
    Binary: base 2
    Hexadecimal: base 16

You should be able to convert between all three.
Given any quantity, you should be able to determine the minimum number of bits required.
If more bits than necessary are given, you should be able to tell if these bits are 0 or 1.

Conversion from binary to unsigned decimal notation:
    1 1 0 1 0 0
bit pos:2^5 2^4 2^3 2^2 2^1 2^0 = 32+16+0+4+0+0 = 52
How do you know whether to interpret this as signed/unsigned? → Indicated in the question. You need to be told whether it's unsigned/two's complement/floating point, etc.

Questions that don't make sense:
Convert -73 to unsigned number → unsigned # can only be positive!

Another example:
0 0 1 1 0 0 → 12. Why? Try it!

How do you convert decimal notation to binary?
52. You are not told how many bits we need! So how??
1. Think of the minimum number of bits required. You could try thinking about log base 2 of the number. Ex. Log base 2(52) = 5.7. Round it down to 5. Raise 2^5 = 32. What is the maximum number you can raise with 5 bits?

1 1 1 1 1
2^4 2^3 2^2 2^1 2^0 → 31.

31 is clearly less than 52! So we try adding another bit.

1 1 1 1 1 1
2^5 2^4 2^3 2^2 2^1 2^0 → 61

So we need six bits!

Let's write down the powers starting from the most significant bit position.
32, 16, 8, 4, 2, 1

Power	#	>=?	Remainder	Bit value
32	52	yes	20	1
16	20	yes	4	1
8	4	no		0
4	4	yes	9	1
2	N/A	N/A		0
1	N/A	N/A		0

Power	#	>=?	Remainder	Bit value
32	23	No	23	0
16	23	Yes	7	1
8	7	No	7	0
4	7	Yes	3	1
2	3	Yes	1	1
1	1	Yes	0	1

Let's try something that's impossible

Power	#	>=?	Remainder	Bit value
32	101	Yes	69	1
16	69	Yes	53	1
8	53	Yes	45	1
4	45	Yes	41	1
2	41	Yes	39	1
1	39	Yes	38	1

But you have a nonzero remainder left! Therefore this is incorrect. That indicates that we have too few bits. Or we made a mistake somewhere.

Let's go back and look at representing +52 as a two's complement number. We know we need 7 bits to represent this.

If this is a **positive** number, you know that that the **most significant bit** is 0. You can then compute the remaining 6 bits.
0 110100

How about -52?
Compute the unsigned representation → 110100
Compute the 1's complement (invert all the bits) → 001011
Add them together

001011 + 1 = 001100
Then we append the final bit at the front. Since it's negative, it equals 1
→ 1001100

If we look at a signed number through an unsigned context, it would be a completely different number!

If you add two negative numbers with simple bitwise addition in 2's complement, you should get the correct answer!

You should be familiar and comfortable with this conversion.

Just like you can represent fractions in decimal, you can also do floating point notation with decimals. Displayed are examples of numbers in binary notation. You can consider all the negative exponents of 2 to represent fractions.

Floating point representation: .43 is represented as .4299999999999999999999 because that's how it's stored. (This is much more painful to do.) Guaranteed: on the exam we will not go below negative powers of 7. Too easy to get wrong...

This will be covered in more detail on Monday!