Why Do Integers Wrap?

As we discussed in the lecture, this is a direct result of the fact that $2^b = n$, where b is 32 for the `int` data type.

To really understand why this happens, though, we need to understand how binary arithmetic works and how integers are represented in binary. Let's start with binary arithmetic for integers.

When we add two binary numbers together, the rules are straightforward:

```
0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 10
```

The first three rules probably seem reasonable to you, but the last one might seem strange. To understand it, let's look at how decimals work for a specific decimal number, 315:

$$\begin{array}{ccc} 10^2 & 10^1 & 10^0 \\ 3 & 1 & 5 \end{array}$$

The least significant digit (`5`) gives us `5 * 1` (10^0 is `1`), the next digit (`1`) gives us `1 * 10` (10^1 is `10`), and the most significant digit (`3`) gives us `3 * 100` (10^2 is `100`). When we add those all together, we get 315.

Binary works the same way, it's just that it's base 2 instead of base 10. Let's look at how binary works for a specific binary number, 110:

$$\begin{array}{ccc} 2^2 & 2^1 & 2^0 \\ 1 & 1 & 0 \end{array}$$

The least significant digit (`0`) gives us `0 * 1` (2^0 is `1`), the next digit (`1`) gives us `1 * 2` (2^1 is `2`), and the most significant digit (`1`) gives us `1 * 4` (2^2 is `4`). When we add those all together, we get 6 (in decimal).

Back to the fourth rule for binary addition, if we convert to decimal, we have 1 + 1 = 2 (just as we'd expect!). Now if we convert 2 back into binary, we get 10, which is exactly what the rule says should be the answer.

To help understand why integers wrap, think of the rule as saying "1 + 1 is 0 carry the 1", just like in decimal we have the rule "9 + 1 is 0 carry the 1".

The other piece of information we need is that integers are represented in binary using *two's complement*. In two's complement, the most significant bit tells us the sign – 0 for positive and 1 for negative. If a number is positive, the magnitude of the number is just the rest of the bits in base 2 as described above. If the number is negative, we flip all the rest of the bits (change 0s to

1s and 1s to 0s) and add 1, then interpret those bits as the magnitude of the negative number in base 2 as described above.

How the heck does that explain the wrapping? Let's say we have 3 (instead of 32) bits for our int. That means the maximum positive number we can represent is 011 (3 in decimal). Let's add 1 to it:

```
  0 1 1
+     1
--------
  1 0 0
```

For the least significant bit, we have 1 + 1 = 0 carry the 1, for the middle bit we add the carry of 1 to the 1 we already have to get 0 carry the 1, and for the most significant bit we add the carry of 1 to the 0 we already have to get 1.

Because 100 is a negative number in two's complement, we flip all the bits (yielding 011), add 1 (yielding 100), and interpret that as the magnitude (-4 in decimal).

So adding 1 to the maximum positive integer we can represent with 3 bits gives us the minimum negative integer we can represent with 3 bits. You should also realize that this is also consistent with $2^b = n$, because $2^3 = 8$ and we can represent 8 integers with 3 bits: -4, -3, -2, -1, 0, 1, 2, and 3.

This of course works exactly the same way when we have 32 bits for an `int`, so that's why integers wrap! Whew.