Our final challenge is figuring out how to represent signed integers, for example, what should be our representation for the number -2000?

In decimal notation, the convention is to precede the number with a "+" or "-" to indicate whether it's positive or negative, usually omitting the "+" to simplify the notation for positive numbers. We could adopt a similar notation -- called "signed magnitude" -- in binary, by allocating a separate bit at the front of the binary string to indicate the sign, say "0" for positive numbers and "1" for negative numbers.

So the signed-magnitude representation for -2000 would be an initial "1" to indicate a negative number, followed by the representation for 2000 (as described on the previous two slides). However there are some complications in using a signed-magnitude representation. There are two possible binary representations for zero: "+0" and "-0". This makes the encoding slightly inefficient but, more importantly, the circuitry for doing addition of signed-magnitude numbers is different than the circuitry for doing subtraction. Of course, we're used to that – in elementary school we learned one technique for addition and another for subtraction.

To keep the circuitry simple, most modern digital systems use the two's complement binary representation for signed numbers. In this representation, the high-order bit of an N-bit two's complement number has a negative weight, as shown in the figure.

Thus all negative numbers have a 1 in the high-order bit and, in that sense, the high-order bit is serving as the "sign bit" – if it's 1, the represented number is negative.

The most negative N-bit number has a 1-bit in the high-order position, representing the value -2^(N-1). The most positive N-bit number has a 0 in the negative-weight high-order bit and 1's for all the positive-weight bits, representing the value 2^(N-1)-1. This gives us the range of possible values – for example, in an 8-bit two's complement representation, the most negative number is -2^7 = -128 and the most positive number is 2^7 – 1 = 127.

If all N bits are 1, think of that as the sum of the most negative number with the most positive number, i.e., -2^(N-1) + 2^(N-1)-1, which equals -1.

And, of course, if all N bits are 0, that's the unique representation of 0.

Let's see what happens when we add the N-bit values for -1 and 1, keeping an N-bit answer.

In the rightmost column, 1 plus 1 is 0, carry the 1.

In the second column, the carry of 1 plus 1 plus 0 is 0, carry the 1.

And so on – the result is all zero's, the representation for 0… perfect!

Notice that we just used ordinary binary addition, even when one or both of the operands are negative. Two's complement is perfect for N-bit arithmetic!

To compute B - A, we can just use addition and compute B + (-A).

So now we just need to figure out the two's complement representation for –A, given the two's complement representation for A. Well, we know that A + (-A) = 0 and using the example above, we can rewrite 0 as 1 + (-1).

Reorganizing terms, we see that –A equals 1 plus the quantity (-1) – A.

As we saw above, the two's complement representation for -1 is all 1-bits, so we can write that subtraction as all 1's minus the individual bits of A: A_0, A_1, … up to A_N-1.

If a particular bit A_i is 0, then 1-A_i = 1 and if A_i is 1, then 1-A_i = 0.

So in each column, the result is the bitwise complement of A_i, which we'll write using the C-language bitwise complement operator tilde.

So we see that –A equals the bitwise complement of A plus 1.

Ta-dah! To practice your skill with two's complement, try your hand at the following exercises. All you need to remember is how to do binary addition and two's complement negation (which is "bitwise complement and add 1").