**Signed Binary/Decimal Conversion Using the Two's Complement Representation**

Students in 121 often find that Epp's coverage of this topic is insufficient for the course. So, we offer this supplement that explains the two meanings of "two's complement" and describes the algorithm for converting signed binary numbers that use the two's complement representation to and from decimal numbers.

Note: This supplement assumes you already know how to convert base 2 numbers to base 10. Look for base 2 to base 10 conversion in Epp Section 2.5.

**Two Meanings of "Two's Complement"**

There are two *very* different ways that we use the words "two's complement".

1. We use them as an action when we say to "take the two's complement of" some binary number or bit pattern. In that case, we mean to flip the bits and add 1 to them, ignoring any carry that's too big to store.

For example, the two's complement of 1011 is: flip the bits to get 0100 and add 1 to get the answer 0101. It doesn't matter at all whether 1011 was positive, negative, or even whether it represented a number.

"Ignoring any carry that's too big to store" refers to taking the two's complement of a number that's all 0s like 0000. When we flip the bits, we get 1111. When we add 1, we get 0000, *not* 10000. That's because the extra 1 doesn't fit in the available four bits.

1. We might say about a binary value that it is "a signed binary number that uses the two's complement representation". This means that the bits represent a number. If the leftmost bit is a 0, the number represented is simply the bits' value as a base 2 number. If the leftmost bit is a 1, the number represented is negative. To find its magnitude, we *take the two's complement* of the bits (flip them and add 1) and interpret the result as a base 2 number.

To summarize, when we say to "take the two's complement" of a value, we *always* mean to flip the bits and add 1. When we say a binary number "uses the two's complement representation", it means you may or may not need to "take the two's complement" in order to find the number's value, depending on whether the leftmost bit is a 1.

**Signed Binary/Decimal Conversion**

We now discuss the algorithm for converting signed binary numbers that use the two's complement representation to and from their decimal value. We assume in the following that all signed binary ints use two's complement representation, as is standard in CPSC 121.

**Signed binary int to decimal conversion**

To convert a signed binary int to a decimal number, use the following algorithm. We illustrate the algorithm on two four-bit examples: 1110 and 0101.

1. If the leftmost bit is a 1, the number is negative. Do the following... (**1**110 starts with a 1. It's negative.)
   1. Flip all the bits in the number. (1110 becomes 0001.)
   2. Add 1 to the number. (0001 + 1 = 0010.)
   3. Convert the result to base 10 and report its negation. (00102 is 210. The answer is -2.)
2. If the leftmost bit is a 0, the number is positive or zero. Convert the number from base 2 to base 10 and report the result. (**0**101 starts with a 0. 01012 is 510. The answer is 5.)

Note that 1000 is not a special case, although it does seem odd. 1000 starts with 1; so, it's negative. We flip the bits (0111) and add 1 (1000). 10002 is 810 so, the answer is -8. (The odd part is that when we flip the bits and add 1, we get the same number back again!)

**Decimal to Signed Binary Conversion**

This works much like the reverse of the algorithm above.

The only strange thing is that we again flip the bits and add 1, rather than subtracting 1 and then flipping the bits (the reverse of what we did above). Why is that? Because flipping and adding 1 is the same as subtracting 1 and flipping. If you doubt that, play around with some numbers to see!

We illustrate the algorithm on two examples, converting them to four-bit signed numbers: -4 and 1.

1. Convert the absolute value of the number to base 2. (The absolute value of -410 is 410. That's 1002.) (The absolute value of 110 is 110. That's 12.)
2. Put enough 0s on the left side to give the base 2 number as many bits as you have available (e.g., 4 bits in our examples). (100 needs one extra 0: 0100.) (1 needs three extra 0s: 0001.)
3. If the original number was positive stop and report the result. (1 is positive; so, the answer is 0001.)
4. Otherwise, flip the bits in the base 2 number and add 1 (ignoring any carry that goes beyond the number of bits available), and report the result. (-4 is negative; so, we flip the bits in 0100 to get 1011 and add 1. The answer is 1100.)

Last updated sometime on **Sat, 12 Sep 2013 by patrice**