Subtraction

We can subtract one binary number from another by using the standard techniques adapted for decimal numbers (subtraction of each bit pair, right to left, "borrowing" as needed from bits to the left). However, if we can leverage the already familiar (and easier) technique of binary addition to subtract, that would be better. As we just learned, we can represent negative binary numbers by using the "two's complement" method and a negative place-weight bit. Here, we'll use those negative binary numbers to subtract through addition. Here's a sample problem:

Subtraction: $7_{10} - 5_{10}$ Addition equivalent: $7_{10} + (-5_{10})$

If all we need to do is represent seven and negative five in binary (two's complemented) form, all we need is three bits plus the negative-weight bit:

positive seven = 0111_2
negative five = 1011_2

Now, let's add them together:

```
.                    1111  <--- Carry bits
.                     0111
.                   + 1011
.                   ------
.                    10010
.                    |
.             Discard extra bit
.
```

. Answer = 0010_2

Since we've already defined our number bit field as three bits plus the negative-weight bit, the fifth bit in the answer (1) will be discarded to give us a result of 0010_2, or positive two, which is the correct answer.

Another way to understand why we discard that extra bit is to remember that the leftmost bit of the lower number possesses a negative weight, in this case equal to negative eight. When we add these two binary numbers together, what we're actually doing with the MSBs is subtracting the lower number's MSB from the upper number's MSB. In subtraction, one never "carries" a digit or bit on to the next left place-weight.

Let's try another example, this time with larger numbers. If we want to add -25_{10} to 18_{10}, we must first decide how large our binary bit field must be. To represent the largest (absolute value) number in our problem, which is twenty-five, we need at least five bits, plus a sixth bit for the negative-weight bit. Let's start by representing positive twenty-five, then finding the two's complement and putting it all together into one numeration:

$+25_{10} = 011001_2$ (showing all six bits)
One's complement of $11001_2 = 100110_2$
One's complement + 1 = two's complement = 100111_2
$-25_{10} = 100111_2$

Essentially, we're representing negative twenty-five by using the negative-weight (sixth) bit with a value of negative thirty-two, plus positive seven (binary 111_2).

Now, let's represent positive eighteen in binary form, showing all six bits:

. $18_{10} = 010010_2$
.
. Now, let's add them together and see what we get:
.

```
.                      11   <--- Carry bits
.                    100111
.                  + 010010
.                  --------
.                    111001
```

Since there were no "extra" bits on the left, there are no bits to discard. The leftmost bit on the answer is a 1, which means that the answer is negative, in two's complement form, as it should be. Converting the answer to decimal form by summing all the bits times their respective weight values, we get:

$$(1 \times -32_{10}) + (1 \times 16_{10}) + (1 \times 8_{10}) + (1 \times 1_{10}) = -7_{10}$$

Indeed -7_{10} is the proper sum of -25_{10} and 18_{10}.

Source: http://www.allaboutcircuits.com/vol_4/chpt_2/4.html