

Representing Negative Numbers: Two's Complement

In class we have discussed binary representations of the non-negative integers $\{0, 1, 2, \dots\}$. However, one must often deal with negative numbers; how can they be represented as well? One obvious solution would be to use a single bit to represent the *sign* of the number (+ or -) and the remaining bits to represent the *magnitude* of the number (how positive or negative it is). In such a *signed magnitude* representation, a most-significant bit of 0 represents “+” while a most-significant bit of 1 represents “-”; the remaining bits give the magnitude of the number. For example, an 8-bit signed magnitude representation of 13 is 00001101 while -13 is 10001101. Note that using 8-bit signed magnitude, one can represent integers in the range -127 (11111111) to 127 (01111111). Signed magnitude has one peculiarity, however. The integer 0 can be represented in two ways: 00000000 = +0 and 10000000 = -0.

By far, the most common representation of positive and negative integers is *two's complement*. In two's complement, positive integers are represented in standard binary, as in signed magnitude. However, the representation of a negative number is determined as follows: (1) compute a binary representation of the magnitude of the number, (2) flip all the bits, and (3) add 1. For example, the 8-bit two's complement representation of 13 is 00001101 (as before) while -13 is represented as follows (using the steps given above):

$$-13 \xrightarrow{(1)} 00001101 \xrightarrow{(2)} 11110010 \xrightarrow{(3)} 11110011$$

Note that a most-significant bit of 1 again signifies a negative number, but the remaining bits do not encode the magnitude in the usual way. Here are a few more examples of converting integers to 8-bit two's complement form (remember that non-negative integers are represented in standard binary):

$$\begin{aligned} 15 &\implies 00001111 \\ -15 &\xrightarrow{(1)} 00001111 \xrightarrow{(2)} 11110000 \xrightarrow{(3)} 11110001 \\ 28 &\implies 00011100 \\ -28 &\xrightarrow{(1)} 00011100 \xrightarrow{(2)} 11100011 \xrightarrow{(3)} 11100100 \end{aligned}$$

To convert a negative two's complement number back to decimal, follow these steps: (1) flip all the bits, (2) add 1, and (3) interpret the result as a binary representation of the magnitude and add a negative sign. For example,

$$11110011 \xrightarrow{(1)} 00001100 \xrightarrow{(2)} 00001101 \xrightarrow{(3)} -13$$

Here are a few more examples of converting 8-bit two's complement back to decimal (remember that if the number begins with a 0, it's a non-negative integer represented in standard binary):

$$\begin{array}{rcl}
00010110 & \implies & 22 \\
10010110 & \xRightarrow{(1)} & 01101001 \xRightarrow{(2)} 01101010 \xRightarrow{(3)} -106 \\
01001001 & \implies & 73 \\
11001001 & \xRightarrow{(1)} & 00110110 \xRightarrow{(2)} 00110111 \xRightarrow{(3)} -55
\end{array}$$

Using 8-bit two's complement, one can represent integers in the range -128 (10000000) to 127 (01111111), and 0 is represented in only one way (00000000). Finally, the real utility and power of two's complement is that one can add pairs of two's complement numbers (whether positive *or* negative) in the usual way, and the result will be the correct answer, in two's complement! In the following examples, superscripts in the binary addition represent carries.

$$\begin{array}{rcl}
\begin{array}{r} 13 \\ + 15 \\ \hline 28 \end{array} & \iff & \begin{array}{r} 0 \ 0 \ 0 \ 0^1 \ 1^1 \ 1^1 \ 0^1 \ 1 \\ + 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \\ \hline 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0 \end{array}
\end{array}$$

Note that this is just standard binary addition. Now, however, let's consider subtracting 15 from 28; this is equivalent to adding -15 to 28.

$$\begin{array}{rcl}
\begin{array}{r} 28 \\ - 15 \\ \hline 13 \end{array} & \iff & \begin{array}{r} 28 \\ + -15 \\ \hline 13 \end{array} & \iff & \begin{array}{r} 0^1 \ 0^1 \ 0^1 \ 1 \ 1 \ 1 \ 0 \ 0 \\ + 1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 1 \\ \hline 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \end{array}
\end{array}$$

Note that we ignore the carry out of the last column.¹ Now consider subtracting 28 from 15.

$$\begin{array}{rcl}
\begin{array}{r} 15 \\ - 28 \\ \hline -13 \end{array} & \iff & \begin{array}{r} 15 \\ + -28 \\ \hline -13 \end{array} & \iff & \begin{array}{r} 0 \ 0 \ 0 \ 0^1 \ 1^1 \ 1 \ 1 \ 1 \\ + 1 \ 1 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \\ \hline 1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 1 \ 1 \end{array}
\end{array}$$

Note that the answer obtained is the proper two's complement representation of -13 . Finally, consider adding -13 and -15 .

$$\begin{array}{rcl}
\begin{array}{r} -13 \\ + -15 \\ \hline -28 \end{array} & \iff & \begin{array}{r} 1^1 \ 1^1 \ 1^1 \ 1 \ 0 \ 0^1 \ 1^1 \ 1 \\ + 1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 1 \\ \hline 1 \ 1 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \end{array}
\end{array}$$

Note that we again ignore the carry out of the last column, and we obtain the proper two's complement representation of -28 .

More examples of two's complement may be found at:

http://en.wikipedia.org/wiki/Two's_complement

¹In general, carries into or out of the most significant bit must be carefully considered as this may result from an overflow condition, i.e., obtaining a result which is too large (> 127) or too small (< -128) to be represented in 8-bit two's complement. In this brief note, however, we assume that all results can be properly represented in 8-bit two's complement.