Subtraction Using Complements

by

John Kennedy
Mathematics Department
Santa Monica College
1900 Pico Blvd.
Santa Monica, CA 90405

rkennedy@ix.netcom.com

Except for this comment explaining that it is blank for some deliberate reason, this page is intentionally blank!

Subtraction Using Complements

Consider the subtraction problem:
$$\begin{array}{r} 7,541 \\ -\,4,786 \\ \hline ?????? \end{array}$$

Any n-digit base-10 integer y has what is called a ***nines' complement*** that is defined to be $10^n - 1 - y$.

Now what is significant about $10^n - 1$ is that it is an integer consisting of n digits that are all 9's. The justification of the previous sentence is that 10^n is itself an $(n+1)$-digit integer that is the digit 1 followed by n 0's. So when we subtract 1 from this number we get an integer consisting of exactly n 9's.

It is virtually trivial to subtract any base-10 integer from a sequence of all 9's that has the same length as the given integer. Next, compare the two subtraction problems:

$$\begin{array}{r} 7,541 \\ -\,4,786 \\ \hline \end{array} \qquad\qquad \begin{array}{r} 9,999 \\ -\,4,786 \\ \hline \end{array}$$

The second problem is so much easier than the first because the second problem doesn't require borrowing, whereas the first is usually considered to require borrowing. We can even use the second problem's answer in an intermediate step that allows us to easily get the answer to the first subtraction by actually performing addition! Yes, the following will show how subtraction can be accomplished by adding instead. A very simple but also surprising subtraction rule or algorithm is the following.

Instead of trying to subtract y from x directly, we can more simply add the nines complement of y to x, and then add another 1 to that and finish by dropping the first digit which will always be a leading 1 carry-digit.

For example, the nines' complement of $4,786$ is quickly and easily found to be $5,213$ and when we add $5,213$ to $7,541$ we get:
$$\begin{array}{r} 7,541 \\ +\,5,213 \\ \hline 12,754 \end{array}$$

Now add another 1 to get $12,755$ and finally drop the leading 1 digit from $12,755$ to get $2,755$ as the result. Note that
$$7,541 - 4,786 = 2,755$$

Why does the above algorithm always work? Simple high-school algebra is all that is required!

$$\begin{aligned} x - y &= x - 10^n + 10^n - y \\ &= x + [(10^n - 1) - y] + 1 - 10^n \\ &= \{\{x + [(10^n - 1) - y]\} + 1\} - 10^n \end{aligned}$$

When properly read, the last expression written above precisely describes the surprising subtraction rule or algorithm because it shows that to subtract y from x we can begin by adding the nines' complement of y to x, and then add 1 to that sum, and then drop the leading digit. The last step of subtracting 10^n is the same as dropping the leading digit from the previous two additions. Note that the leading digit will always be a 1 carry-digit. The algorithm for subtraction can be remembered by the three simple steps:

1. Add the nines' complement of y to x.
2. Add 1 to the previous sum.
3. Drop the first digit from the second sum.

A couple of more examples will help explain more about what we are calling nines' complements. If x is any positive integer written using base-10 positional notation, then x plus the nines' complement of x is the same as the integer consisting of all $9'$s that has the same length as x. For example, the nines' complement of $125,403$ is the integer $874,596$ since $125,403 + 874,596 = 999,999$. In fact, the two positive integers $125,403$ and $874,596$ are both nines' complements of each other since they have the same length in terms of the number of digits (six in this case) and since their sum is the six-digit integer $999,999 = 10^6 - 1$.

Another subtraction example is to compute $2,341 - 16$ using the above three steps. First note that we want to write $16 = 0,016$ to give 16 the same 4-digit length that $2,341$ has. Then trivially, the nines' complement of $0,016$ is the integer $9,983$ and $2,341 + 9,983 = 12,324$ and now if we add 1, we get $12,325$ and then we drop the leading 1 digit and we get $2,325$ as the answer to $2,341 - 16$.

Subtraction Using Any Base b

The real importance of the above algorithm is that it works even when the numbers are written in any base b other than 10. When $b \neq 10$, we don't talk about the nines' complement of the integer but we can talk about the $(b-1)$'s complement. In particular, when $b = 2$ then we can talk about the complement of a binary integer that is written using only the digits 0 and 1. This is the form in which virtually all modern microprocessors and computers are considered to store and perform computations with integers.

When $b \geq 2$, any n-digit integer written in the base b will only consist of the digits $0, 1, 2, \ldots, (b-1)$. In fact, any such n-digit integer can be written exactly as an $(n-1)$st degree polynomial value, where all the polynomial coefficients are integers in the range between 0 and $(b-1)$. Note that any nonnegative integer in the range from 0 to $b^n - 1$ can be written in the form of a polynomial evaluated at $x = b$. If the n-digit integer $y = a_{n-1}a_{n-2}\cdots a_3a_2a_1a_0$ in the base b, then we can write:

$$y = \left\{ \sum_{k=0}^{n-1} a_k x^k \right\} \Bigg|_{x=b}$$

where the a_k coefficients are precisely the base b digits of y. For each $k,\ 0 \leq\ a_k \leq b-1$ and each a_k is a nonnegative integer.

In particular, we can show that

$$b^n - 1 = \left\{ \sum_{k=0}^{n-1} (b-1) \cdot x^k \right\} \Bigg|_{x=b}$$

where each a_k polynomial coefficient is the same constant value that is always $(b-1)$.

When thought of in this form, $b^n - 1$ is really the sum of a geometric series in which the first term is $(b-1)$.

$$\left\{ \sum_{k=0}^{n-1} (b-1) \cdot x^k \right\} \Bigg|_{x=b} = (b-1) \cdot \sum_{k=0}^{n-1} b^k$$

$$= (b-1)\{1 + b + b^2 + b^3 + \cdots + b^{n-1}\} = (b-1)\left\{ \frac{b^n - 1}{b-1} \right\} = b^n - 1$$

and this identity helps explain why the largest n-digit integer that can be represented in the base b using only the digits 0 to $(b-1)$ is $b^n - 1$.

The $(b-1)$s$'$ complement of any positive n-digit integer y written in base-b positional notation is by definition $b^n - 1 - y$.

In particular, the subtraction $x - y$ using the base b representation of x and y is simply:

$$x - y = (\{x + [(b^n - 1) - y]\} + 1) - b^n$$

The rule for subtracting $x - y$ using integers x and y written in base-b positional notation is as follows:

1. Add the $(b-1)$s' complement of y to x.
2. Add 1 to the previous sum.
3. Drop the first digit from the second sum.

What is even more significant is that when $b = 2$ then $b^n - 1$ is an integer that consists of all 1 digits. So when it comes to computing $2^n - 1 - y$ we need only take the complementary bit of each bit in y because binary arithmetic means $1 - 1 = 0$ while $1 - 0 = 1$. In a computer, taking complementary bits is extremely fast and easy because reversing a bit requires the simplest electronic circuit with just one negation logic gate. In fact, computers don't necessarily need a separate circuit for subtraction because they can use the circuit for addition as long as they complement every bit in y, then add x, then add 1, and finally drop the leading 1 carry-bit.

It is unusual if not ironic that the same algorithm is advantageous for both humans and computers. For humans, the advantage is that they don't have to perform borrowing. For computers, it is the simplicity of the logic circuits in terms of the number of logic gates that are needed to implement an algorithm. In any event, it is useful to know there is a simple and robust algorithm for performing integer subtraction in any base-b positional system using the equation: $x - y = (\{x + [(b^n - 1) - y]\} + 1) - b^n$

Implementing the above subtraction algorithm in a computer chip is more involved than what we might at first think because when performing exact integer arithmetic in any computer we should worry about both overflow and underflow and we need to worry about how to represent negative numbers. However, unless you are a computer engineer you probably wouldn't be concerned with these three potential problems. In some computer systems, there are actually two zeros, one is considered a positive zero and the other is considered a negative zero. Because of the two zero problem, special representations have been invented so that there is only one zero and yet negative integers can also be represented in a binary format. For more information about the history of positional number systems and their relationship to computers we recommend reading Donald Knuth's ***The Art of Computer Programming, Volume 2, Semi-Numerical Algorithms***, Addison-Wesley Co., 1969, pp. 161-217.

Mathematicians who are numerical analysts are concerned with proving that neither underflow nor overflow will occur under the right conditions, and they also are concerned with how to handle computer arithmetic when those conditions are not satisfied. In general there is a real difference between algorithms like the one above that perform exact integer arithmetic and algorithms that use floating point arithmetic with numbers that are not necessarily exact because they cover a large range that involves scientific notation. When computing with numbers in scientific notation another concern that arises is the precision of the answer.

How numbers, even negative integers, can be represented in a binary form, let alone how they are to be added or subtracted leads to other questions that are best answered after gaining more experience with computer arithmetic. Historically it is interesting that the first electronic computer wasn't designed very well for doing arithmetic, precisely because of lack of experience in thinking of how to best represent numbers and how to best perform computations.

Just like jet airplanes designed after 50 years of experience are radically different from the first designs, so are the computers of today more efficient and faster than their first cousins. The first fully electronic computer was called the ENIAC and it became operational in the spring of 1945 and it stored and used numbers in a base-10 format. It didn't take too long before the ENIAC designers and all computer designers after them decided that base 2 representations would be more efficient in terms of the number of logic gates and in terms of raw processing speeds. Thus virtually all microprocessors built today use binary (base 2) not base 10 to internally represent integers.