Table 2-3 Binary addition and subtraction table.

c_{in} or b_{in}	x	y	c_{out}	s	b_{out}	d
0	0	0	0	0	0	0
0	0	1	0	1	1	1
0	1	0	0	1	0	1
0	1	1	1	0	0	0
1	0	0	0	1	1	1
1	0	1	1	0	1	0
1	1	0	1	0	0	0
1	1	1	1	1	1	1

2.4 Addition and Subtraction of Nondecimal Numbers

Addition and subtraction of nondecimal numbers by hand uses the same technique that we learned in grammar school for decimal numbers; the only catch is that the addition and subtraction tables are different.

binary addition

Table 2-3 is the addition and subtraction table for binary digits. To add two binary numbers X and Y, we add together the least significant bits with an initial carry (c_{in}) of 0, producing carry (c_{out}) and sum (s) bits according to the table. We continue processing bits from right to left, adding the carry out of each column into the next column's sum.

Two examples of decimal additions and the corresponding binary additions are shown in Figure 2-1, using a colored arrow to indicate a carry of 1. The same examples are repeated below along with two more, with the carries shown as a bit string C:

$$\begin{array}{crr}
C & & 101111000 \\
X & 190 & 10111110 \\
Y & +141 & +\ \ 10001101 \\
\hline
X+Y & 331 & 101001011
\end{array}
\qquad
\begin{array}{crr}
C & & 001011000 \\
X & 173 & 10101101 \\
Y & +\ 44 & +\ 00101100 \\
\hline
X+Y & 217 & 11011001
\end{array}$$

$$\begin{array}{crr}
C & & 011111110 \\
X & 127 & 01111111 \\
Y & +\ 63 & +\ \ 00111111 \\
\hline
X+Y & 190 & 10111110
\end{array}
\qquad
\begin{array}{crr}
C & & 000000000 \\
X & 170 & 10101010 \\
Y & +\ 85 & +\ 01010101 \\
\hline
X+Y & 255 & 11111111
\end{array}$$

binary subtraction

minuend

subtrahend

difference

Binary subtraction is performed similarly, using borrows (b_{in} and b_{out}) instead of carries between steps, and producing a difference bit d. Two examples of decimal subtractions and the corresponding binary subtractions (*minuend* minus *subtrahend* yields *difference*) are shown in Figure 2-2. As in decimal subtraction, the binary minuend values in the columns are modified when borrows occur, as shown by the colored arrows and bits. The examples from the figure are

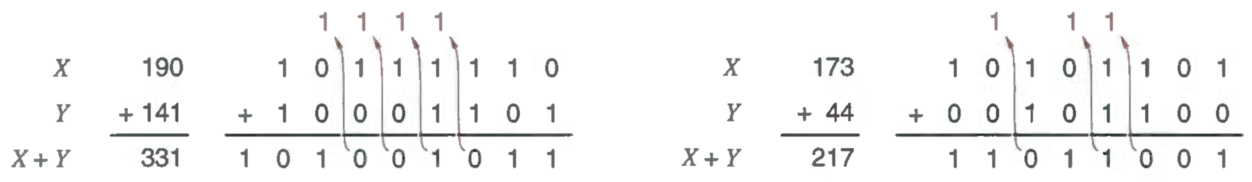

Figure 2-1 Examples of decimal and corresponding binary additions.

repeated below along with two more, this time showing the borrows as a bit string B:

B		001111100	B		011011010
X	229	11100101	X	210	11010010
Y	− 46	− 00101110	Y	−109	− 01101101
$X-Y$	183	10110111	$X-Y$	101	01100101
B		010101010	B		000000000
X	170	10101010	X	221	11011101
Y	− 85	− 01010101	Y	− 76	− 01001100
$X-Y$	85	01010101	$X-Y$	145	10010001

comparing numbers

A very common use of subtraction in computers is to compare two numbers. For example, if the operation $X - Y$ produces a borrow out of the most significant bit position, then X is less than Y; otherwise, X is greater than or equal to Y. The relationship between carries and borrows in adders and subtractors will be explored in Section 6.10.

Addition and subtraction tables can be developed for octal and hexadecimal digits, or any other desired radix. However, few computer engineers bother to memorize these tables. If you rarely need to manipulate nondecimal numbers,

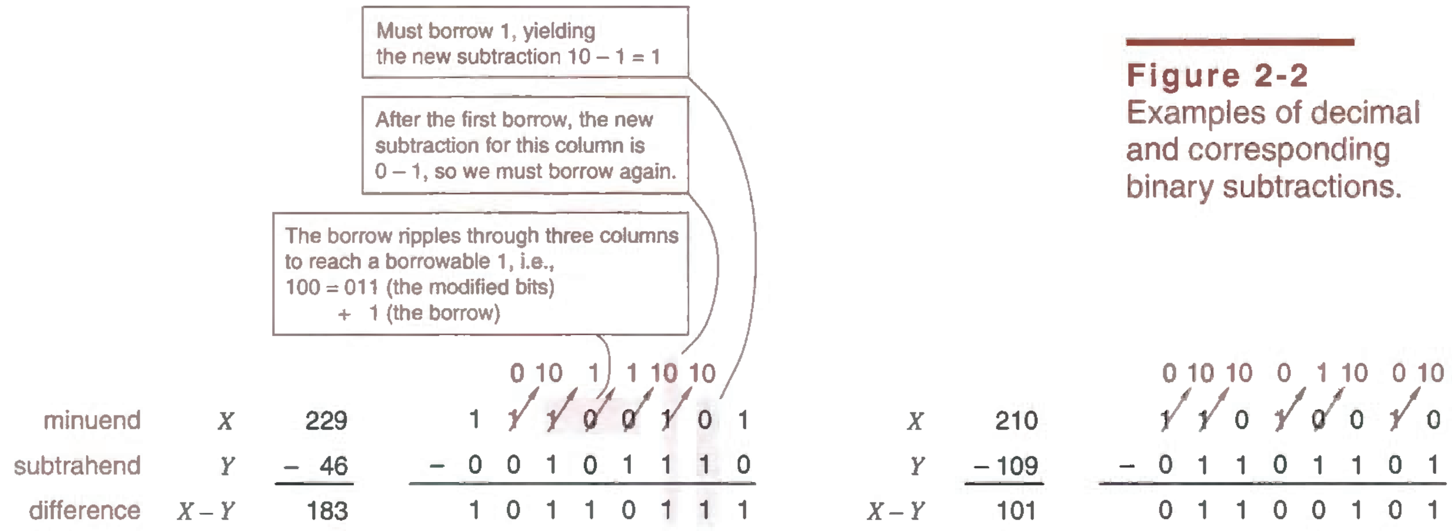
Must borrow 1, yielding the new subtraction 10 − 1 = 1
After the first borrow, the new subtraction for this column is 0 − 1, so we must borrow again.
The borrow ripples through three columns to reach a borrowable 1, i.e., 100 = 011 (the modified bits) + 1 (the borrow)
minuend
subtrahend
difference
X 229 1 1 1 0 0 1 0 1
Y − 46 − 0 0 1 0 1 1 1 0
X−Y 183 1 0 1 1 0 1 1 1
0 10 1 1 10 10
X 210 1 1 0 1 0 0 1 0
Y −109 − 0 1 1 0 1 1 0 1
X−Y 101 0 1 1 0 0 1 0 1
0 10 10 0 1 10 0 10

Figure 2-2 Examples of decimal and corresponding binary subtractions.

then it's easy enough on those occasions to convert them to decimal, calculate results, and convert back. On the other hand, if you must perform calculations in binary, octal, or hexadecimal frequently, then you should ask Santa for a programmer's "hex calculator" from Texas Instruments or Casio.

If the calculator's battery wears out, some mental shortcuts can be used to facilitate nondecimal arithmetic. In general, each column addition (or subtraction) can be done by converting the column digits to decimal, adding in decimal, and converting the result to corresponding sum and carry digits in the nondecimal radix. (A carry is produced whenever the column sum equals or exceeds the radix.) Since the addition is done in decimal, we rely on our knowledge of the decimal addition table; the only new thing that we need to learn is the conversion from decimal to nondecimal digits and vice versa. The sequence of steps for mentally adding two hexadecimal numbers is shown below:

hexadecimal addition

C	1 1 0 0	1	1	0	0
X	$1\ 9\ B\ 9_{16}$	1	9	11	9
Y	$+\ C\ 7\ E\ 6_{16}$	+12	7	14	6
$X+Y$	$E\ 1\ 9\ F_{16}$	14	17	25	15
		14	16+1	16+9	15
		E	1	9	F

2.5 Representation of Negative Numbers

So far, we have dealt only with positive numbers, but there are many ways to represent negative numbers. In everyday business we use the signed-magnitude system, discussed next. However, most computers use one of the complement number systems that we introduce later.

2.5.1 Signed-Magnitude Representation

signed-magnitude system

In the *signed-magnitude system*, a number consists of a magnitude and a symbol indicating whether the number is positive or negative. Thus, we interpret decimal numbers +98, –57, +123.5, and –13 in the usual way, and we also assume that the sign is "+" if no sign symbol is written. There are two possible representations of zero, "+0" and "–0", but both have the same value.

sign bit

The signed-magnitude system is applied to binary numbers by using an extra bit position to represent the sign (the *sign bit*). Traditionally, the most significant bit (MSB) of a bit string is used as the sign bit (0 = plus, 1 = minus), and the lower-order bits contain the magnitude. Thus, we can write several 8-bit signed-magnitude integers and their decimal equivalents:

$$\begin{array}{llll}
01010101_2 = +85_{10} & & 11010101_2 = -85_{10} \\
01111111_2 = +127_{10} & & 11111111_2 = -127_{10} \\
00000000_2 = +0_{10} & & 10000000_2 = -0_{10}
\end{array}$$