3

Digital Arithmetic

Having discussed different methods of numeric and alphanumeric data representation in the first two chapters, the next obvious step is to study the rules of data manipulation. Two types of operation that are performed on binary data include arithmetic and logic operations. Basic arithmetic operations include addition, subtraction, multiplication and division. AND, OR and NOT are the basic logic functions. While the rules of arithmetic operations are covered in the present chapter, those related to logic operations will be discussed in the next chapter.

3.1 Basic Rules of Binary Addition and Subtraction

The basic principles of binary addition and subtraction are similar to what we all know so well in the case of the decimal number system. In the case of addition, adding '0' to a certain digit produces the same digit as the sum, and, when we add '1' to a certain digit or number in the decimal number system, the result is the next higher digit or number, as the case may be. For example, 6 + 1 in decimal equals '7' because '7' immediately follows '6' in the decimal number system. Also, 7 + 1 in octal equals '10' as, in the octal number system, the next adjacent higher number after '7' is '10'. Similarly, 9 + 1 in the hexadecimal number system is 'A'. With this background, we can write the basic rules of binary addition as follows:

1. $0 + 0 = 0$.
2. $0 + 1 = 1$.
3. $1 + 0 = 1$.
4. $1 + 1 = 0$ with a carry of '1' to the next more significant bit.
5. $1 + 1 + 1 = 1$ with a carry of '1' to the next more significant bit.

Table 3.1 summarizes the sum and carry outputs of all possible three-bit combinations. We have taken three-bit combinations as, in all practical situations involving the addition of two larger bit

Digital Electronics: Principles, Devices and Applications Anil K. Maini
 ISBN: 978-0-470-03214-5

Table 3.1 Binary addition of three bits.

A	B	Carry-in (C_{in})	Sum	Carry-out (C_o)	*A*	*B*	Carry-in (C_{in})	Sum	Carry-out (C_o)
0	0	0	0	0	1	0	0	1	0
0	0	1	1	0	1	0	1	0	1
0	1	0	1	0	1	1	0	0	1
0	1	1	0	1	1	1	1	1	1

numbers, we need to add three bits at a time. Two of the three bits are the bits that are part of the two binary numbers to be added, and the third bit is the carry-in from the next less significant bit column.

The basic principles of binary subtraction include the following:

1. $0 - 0 = 0$.
2. $1 - 0 = 1$.
3. $1 - 1 = 0$.
4. $0 - 1 = 1$ with a borrow of 1 from the next more significant bit.

The above-mentioned rules can also be explained by recalling rules for subtracting decimal numbers. Subtracting '0' from any digit or number leaves the digit or number unchanged. This explains the first two rules. Subtracting '1' from any digit or number in decimal produces the immediately preceding digit or number as the answer. In general, the subtraction operation of larger-bit binary numbers also involves three bits, including the two bits involved in the subtraction, called the minuend (the upper bit) and the subtrahend (the lower bit), and the borrow-in. The subtraction operation produces the difference output and borrow-out, if any. Table 3.2 summarizes the binary subtraction operation. The entries in Table 3.2 can be explained by recalling the basic rules of binary subtraction mentioned above, and that the subtraction operation involving three bits, that is, the minuend (A), the subtrahend (B) and the borrow-in (B_{in}), produces a difference output equal to ($A - B - B_{in}$). It may be mentioned here that, in the case of subtraction of larger-bit binary numbers, the least significant bit column always involves two bits to produce a difference output bit and the borrow-out

Table 3.2 Binary subtraction.

Inputs			Outputs	
Minuend (A)	Subtrahend (B)	Borrow-in (B_{in})	Difference (D)	Borrow-out (B_o)
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

bit. The borrow-out bit produced here becomes the borrow-in bit for the next more significant bit column, and the process continues until we reach the most significant bit column. The addition and subtraction of larger-bit binary numbers is illustrated with the help of examples in sections 3.2 and 3.3 respectively.

3.2 Addition of Larger-Bit Binary Numbers

The addition of larger binary integers, fractions or mixed binary numbers is performed columnwise in just the same way as in the case of decimal numbers. In the case of binary numbers, however, we follow the basic rules of addition of two or three binary digits, as outlined earlier. The process of adding two larger-bit binary numbers can be best illustrated with the help of an example.

Consider two generalized four-bit binary numbers $(A_3\,A_2\,A_1\,A_0)$ and $(B_3\,B_2\,B_1\,B_0)$, with A_0 and B_0 representing the LSB and A_3 and B_3 representing the MSB of the two numbers. The addition of these two numbers is performed as follows. We begin with the LSB position. We add the LSB bits and record the sum S_0 below these bits in the same column and take the carry C_0, if any, to the next column of bits. For instance, if $A_0 = 1$ and $B_0 = 0$, then $S_0 = 1$ and $C_0 = 0$. Next we add the bits A_1 and B_1 and the carry C_0 from the previous addition. The process continues until we reach the MSB bits. The four steps are shown ahead. C_0, C_1, C_2 and C_3 are carrys, if any, produced as a result of adding first, second, third and fourth column bits respectively, starting from LSB and proceeding towards MSB. A similar procedure is followed when the given numbers have both integer as well as fractional parts:

1.

			(C_0)	
A_3	A_2	A_1	A_0	
B_3	B_2	B_1	B_0	
			S_0	

2.

		(C_1)	(C_0)	
A_3	A_2	A_1	A_0	
B_3	B_2	B_1	B_0	
		S_1	S_0	

3.

(C_2)	(C_1)	(C_0)		
A_3	A_2	A_1	A_0	
B_3	B_2	B_1	B_0	
	S_2	S_1	S_0	

4.

	(C_2)	(C_1)	(C_0)	
A_3	A_2	A_1	A_0	
B_3	B_2	B_1	B_0	
C_3	S_3	S_2	S_1	S_0

3.2.1 Addition Using the 2's Complement Method

The 2's complement is the most commonly used code for processing positive and negative binary numbers. It forms the basis of arithmetic circuits in modern computers. When the decimal numbers to be added are expressed in 2's complement form, the addition of these numbers, following the basic laws of binary addition, gives correct results. Final carry obtained, if any, while adding MSBs should be disregarded. To illustrate this, we will consider the following four different cases:

1. Both the numbers are positive.
2. Larger of the two numbers is positive.
3. The larger of the two numbers is negative.
4. Both the numbers are negative.

Case 1

- Consider the decimal numbers +37 and +18.
- The 2's complement of +37 in eight-bit representation = 00100101.
- The 2's complement of +18 in eight-bit representation = 00010010.
- The addition of the two numbers, that is, +37 and +18, is performed as follows

$$\begin{array}{r} 00100101 \\ +\ 00010010 \\ \hline 00110111 \end{array}$$

- The decimal equivalent of $(00110111)_2$ is (+55), which is the correct answer.

Case 2

- Consider the two decimal numbers +37 and -18.
- The 2's complement representation of +37 in eight-bit representation = 00100101.
- The 2's complement representation of −18 in eight-bit representation = 11101110.
- The addition of the two numbers, that is, +37 and −18, is performed as follows:

$$\begin{array}{r} 00100101 \\ +\ 11101110 \\ \hline 00010011 \end{array}$$

- The final carry has been disregarded.
- The decimal equivalent of $(00010011)_2$ is +19, which is the correct answer.

Case 3

- Consider the two decimal numbers +18 and −37.
- −37 in 2's complement form in eight−bit representation = 11011011.
- +18 in 2's complement form in eight−bit representation = 00010010.
- The addition of the two numbers, that is, −37 and +18, is performed as follows:

$$\begin{array}{r} 11011011 \\ +\ 00010010 \\ \hline 11101101 \end{array}$$

- The decimal equivalent of $(11101101)_2$, which is in 2's complement form, is −19, which is the correct answer. 2's complement representation was discussed in detail in Chapter 1 on number systems.

Case 4

- Consider the two decimal numbers −18 and −37.
- −18 in 2's complement form is 11101110.
- −37 in 2's complement form is 11011011.
- The addition of the two numbers, that is, −37 and −18, is performed as follows:

$$\begin{array}{r} 11011011 \\ +\ 11101110 \\ \hline 11001001 \end{array}$$

- The final carry in the ninth bit position is disregarded.
- The decimal equivalent of $(11001001)_2$, which is in 2's complement form, is -55, which is the correct answer.

It may also be mentioned here that, in general, 2's complement notation can be used to perform addition when the expected result of addition lies in the range from -2^{n-1} to $+(2^{n-1}-1)$, n being the number of bits used to represent the numbers. As an example, eight-bit 2's complement arithmetic cannot be used to perform addition if the result of addition lies outside the range from -128 to $+127$. Different steps to be followed to do addition in 2's complement arithmetic are summarized as follows:

1. Represent the two numbers to be added in 2's complement form.
2. Do the addition using basic rules of binary addition.
3. Disregard the final carry, if any.
4. The result of addition is in 2's complement form.

Example 3.1

Perform the following addition operations:

1. $(275.75)_{10} + (37.875)_{10}$.
2. $(AF1.B3)_{16} + (FFF.E)_{16}$.

Solution

1. As a first step, the two given decimal numbers will be converted into their equivalent binary numbers (decimal-to-binary conversion has been covered at length in Chapter 1, and therefore the decimal-to-binary conversion details will not be given here):

$$(275.75)_{10} = (100010011.11)_2 \text{ and } (37.875)_{10} = (100101.111)_2$$

The two binary numbers can be rewritten as $(100010011.110)_2$ and $(000100101.111)_2$ to have the same number of bits in their integer and fractional parts. The addition of two numbers is performed as follows:

$$\begin{array}{r} 100010011.110 \\ 000100101.111 \\ \hline 100111001.101 \end{array}$$

The decimal equivalent of $(100111001.101)_2$ is $(313.625)_{10}$.

2. $(AF1.B3)_{16} = (1010111100001.10110011)_2$ and $(FFF.E)_{16} = (1111111111111.1110)_2$. $(1111111111.1110)_2$ can also be written as $(1111111111111.11100000)_2$ to have the same number of bits in the integer and fractional parts. The two numbers can now be added as follows:

$$\begin{array}{r} 0101011110001.10110011 \\ 0111111111111.11100000 \\ \hline 1101011110001.10010011 \end{array}$$

The hexadecimal equivalent of $(1101011110001.10010011)_2$ is $(1AF1.93)_{16}$, which is equal to the hex addition of $(AF1.B3)_{16}$ and $(FFF.E)_{16}$.

Example 3.2

Find out whether 16-bit 2's complement arithmetic can be used to add 14 276 and 18 490.

Solution

The addition of decimal numbers 14 276 and 18 490 would yield 32 766. 16-bit 2's complement arithmetic has a range of -2^{15} to $+(2^{15} - 1)$, i.e. $-32\,768$ to $+32\,767$. The expected result is inside the allowable range. Therefore, 16-bit arithmetic can be used to add the given numbers.

Example 3.3

Add -118 and -32 firstly using eight-bit 2's complement arithmetic and then using 16-bit 2's complement arithmetic. Comment on the results.

Solution

- -118 in eight-bit 2's complement representation = 10001010.
- -32 in eight-bit 2's complement representation = 11100000.
- The addition of the two numbers, after disregarding the final carry in the ninth bit position, is 01101010. Now, the decimal equivalent of $(01101010)_2$, which is in 2's complement form, is $+106$. The reason for the wrong result is that the expected result, i.e. -150, lies outside the range of eight-bit 2's complement arithmetic. Eight-bit 2's complement arithmetic can be used when the expected result lies in the range from -2^7 to $+ (2^7 - 1)$, i.e. -128 to $+127$. -118 in 16-bit 2's complement representation = 1111111110001010.
- -32 in 16-bit 2's complement representation = 1111111111100000.
- The addition of the two numbers, after disregarding the final carry in the 17th position, produces 1111111101101010. The decimal equivalent of $(1111111101101010)_2$, which is in 2's complement form, is -150, which is the correct answer. 16-bit 2's complement arithmetic has produced the correct result, as the expected result lies within the range of 16-bit 2's complement notation.

3.3 Subtraction of Larger-Bit Binary Numbers

Subtraction is also done columnwise in the same way as in the case of the decimal number system. In the first step, we subtract the LSBs and subsequently proceed towards the MSB. Wherever the subtrahend (the bit to be subtracted) is larger than the minuend, we borrow from the next adjacent

higher bit position having a '1'. As an example, let us go through different steps of subtracting $(1001)_2$ from $(1100)_2$.

In this case, '1' is borrowed from the second MSB position, leaving a '0' in that position. The borrow is first brought to the third MSB position to make it '10'. Out of '10' in this position, '1' is taken to the LSB position to make '10' there, leaving a '1' in the third MSB position. $10-1$ in the LSB column gives '1', $1-0$ in the third MSB column gives '1', $0-0$ in the second MSB column gives '0' and $1-1$ in the MSB also gives '0' to complete subtraction. Subtraction of mixed numbers is also done in the same manner. The above-mentioned steps are summarized as follows:

1.	**1**	**1**	**0**	**0**	**2.**	**1**	**1**	**0**	**0**
	1	**0**	**0**	**1**		**1**	**0**	**0**	**1**
				1				**1**	**1**
3.	**1**	**1**	**0**	**0**	**4.**	**1**	**1**	**0**	**0**
	1	**0**	**0**	**1**		**1**	**0**	**0**	**1**
		0	**1**	**1**		**0**	**0**	**1**	**1**

3.3.1 Subtraction Using 2's Complement Arithmetic

Subtraction is similar to addition. Adding 2's complement of the subtrahend to the minuend and disregarding the carry, if any, achieves subtraction. The process is illustrated by considering six different cases:

1. Both minuend and subtrahend are positive. The subtrahend is the smaller of the two.
2. Both minuend and subtrahend are positive. The subtrahend is the larger of the two.
3. The minuend is positive. The subtrahend is negative and smaller in magnitude.
4. The minuend is positive. The subtrahend is negative and greater in magnitude.
5. Both minuend and subtrahend are negative. The minuend is the smaller of the two.
6. Both minuend and subtrahend are negative. The minuend is the larger of the two.

Case 1

- Let us subtract $+14$ from $+24$.
- The 2's complement representation of $+24 = 00011000$.
- The 2's complement representation of $+14 = 00001110$.
- Now, the 2's complement of the subtrahend (i.e. $+14$) is 11110010.
- Therefore, $+24 - (+14)$ is given by

$$\begin{array}{r} 00011000 \\ +\ 11110010 \\ \hline \underline{00001010} \end{array}$$

  with the final carry disregarded.
- The decimal equivalent of $(00001010)_2$ is $+10$, which is the correct answer.

Case 2

- Let us subtract +24 from +14.
- The 2's complement representation of +14 = 00001110.
- The 2's complement representation of +24 = 00011000.
- The 2's complement of the subtrahend (i.e. +24) = 11101000.
- Therefore, +14 − (+24) is given by

$$\begin{array}{r} 00001110 \\ +\ 11101000 \\ \hline 11110110 \end{array}$$

- The decimal equivalent of $(11110110)_2$, which is of course in 2's complement form, is −10 which is the correct answer.

Case 3

- Let us subtract −14 from +24.
- The 2's complement representation of +24 = 00011000 = minuend.
- The 2's complement representation of −14 = 11110010 = subtrahend.
- The 2's complement of the subtrahend (i.e. −14) = 00001110.
- Therefore, +24 − (−14) is performed as follows:

$$\begin{array}{r} 00011000 \\ +\ 00001110 \\ \hline 00100110 \end{array}$$

- The decimal equivalent of $(00100110)_2$ is +38, which is the correct answer.

Case 4

- Let us subtract −24 from +14.
- The 2's complement representation of +14 = 00001110 = minuend.
- The 2's complement representation of −24 = 11101000 = subtrahend.
- The 2's complement of the subtrahend (i.e. −24) = 00011000.
- Therefore, +14 − (−24) is performed as follows:

$$\begin{array}{r} 00001110 \\ +\ 00011000 \\ \hline 00100110 \end{array}$$

- The decimal equivalent of $(00100110)_2$ is +38, which is the correct answer.

Case 5

- Let us subtract −14 from −24.
- The 2's complement representation of −24 = 11101000 = minuend.

- The 2's complement representation of -14=11110010 = subtrahend.
- The 2's complement of the subtrahend = 00001110.
- Therefore, $-24 - (-14)$ is given as follows:

$$\begin{array}{r} 11101000 \\ +\,00001110 \\ \hline 11110110 \end{array}$$

- The decimal equivalent of $(11110110)_2$, which is in 2's complement form, is -10, which is the correct answer.

Case 6

- Let us subtract -24 from -14.
- The 2's complement representation of -14 = 11110010 = minuend.
- The 2's complement representation of -24=11101000 = subtrahend.
- The 2's complement of the subtrahend = 00011000.
- Therefore, $-14 - (-24)$ is given as follows:

$$\begin{array}{r} 11110010 \\ +\,00011000 \\ \hline 00001010 \end{array}$$

with the final carry disregarded.

- The decimal equivalent of $(00001010)_2$, which is in 2's complement form, is $+10$, which is the correct answer.

It may be mentioned that, in 2's complement arithmetic, the answer is also in 2's complement notation, only with the MSB indicating the sign and the remaining bits indicating the magnitude. In 2's complement notation, positive magnitudes are represented in the same way as the straight binary numbers, while the negative magnitudes are represented as the 2's complement of their straight binary counterparts. A '0' in the MSB position indicates a positive sign, while a '1' in the MSB position indicates a negative sign.

The different steps to be followed to do subtraction in 2's complement arithmetic are summarized as follows:

1. Represent the minuend and subtrahend in 2's complement form.
2. Find the 2's complement of the subtrahend.
3. Add the 2's complement of the subtrahend to the minuend.
4. Disregard the final carry, if any.
5. The result is in 2's complement form.
6. 2's complement notation can be used to perform subtraction when the expected result of subtraction lies in the range from -2^{n-1} to $+(2^{n-1} - 1)$, n being the number of bits used to represent the numbers.

Example 3.4

Subtract $(1110.011)_2$ from $(11011.11)_2$ using basic rules of binary subtraction and verify the result by showing equivalent decimal subtraction.

Solution

The minuend and subtrahend are first modified to have the same number of bits in the integer and fractional parts. The modified minuend and subtrahend are $(11011.110)_2$ and $(01110.011)_2$ respectively:

$$\begin{array}{r} 11011.110 \\ -\ 01110.011 \\ \hline 01101.011 \end{array}$$

The decimal equivalents of $(11011.110)_2$ and $(01110.011)_2$ are 27.75 and 14.375 respectively. Their difference is 13.375, which is the decimal equivalent of $(01101.011)_2$.

Example 3.5

Subtract (a) $(-64)_{10}$ from $(+32)_{10}$ and (b) $(29.A)_{16}$ from $(4F.B)_{16}$. Use 2's complement arithmetic.

Solution:

(a) $(+32)_{10}$ in 2's complement notation $= (00100000)_2$.
$(-64)_{10}$ in 2's complement notation $= (11000000)_2$.
The 2's complement of $(-64)_{10} = (01000000)_2$.
$(+32)_{10} - (-64)_{10}$ is determined by adding the 2's complement of $(-64)_{10}$ to $(+32)_{10}$.
Therefore, the addition of $(00100000)_2$ to $(01000000)_2$ should give the result. The operation is shown as follows:

$$\begin{array}{r} 00100000 \\ +\ 01000000 \\ \hline 01100000 \end{array}$$

The decimal equivalent of $(01100000)_2$ is $+96$, which is the correct answer as $+32 - (-64) = +96$.

(b) The minuend $= (4F.B)_{16} = (01001111.1011)_2$.
The minuend in 2's complement notation $= (01001111.1011)_2$.
The subtrahend $= (29.A)_{16} = (00101001.1010)_2$.
The subtrahend in 2's complement notation $= (00101001.1010)_2$.
The 2's complement of the subtrahend $= (11010110.0110)_2$.
$(4F.B)_{16} - (29.A)_{16}$ is given by the addition of the 2's complement of the subtrahend to the minuend.

$$\begin{array}{r} 01001111.1011 \\ +\ 11010110.0110 \\ \hline 00100110.0001 \end{array}$$

with the final carry disregarded. The result is also in 2's complement form. Since the result is a positive number, 2's complement notation is the same as it would be in the case of the straight binary code.
The hex equivalent of the resulting binary number $= (26.1)_{16}$, which is the correct answer.