Chapter 18

Numbering Systems

18.1 THE BINARY, OCTAL, AND HEXADECIMAL SYSTEMS

The most commonly used numbering system is the decimal system. It is named this way because it is based on the number 10, hence the prefix "deci." The number 10 itself is called the *basis* or *radix* of the system. The basis tells us how many different individual symbols may be used to write numbers in the system. In the decimal system these symbols are called *digits*; they are the familiar 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. Note that they range in value from 0 to 9. This is a particular case of a more general rule that can be stated as follows: *Given any positive basis or radix N, there are N different individual symbols that can be used to write numbers in the system. The value of these symbols range from 0 to N − 1.*

In addition to the decimal system, there are other numerical systems that are extensively used in the computer and telecommunication fields. They are the binary system (its basis is 2), the octal system (its basis is 8), and the hexadecimal system (its basis is 16).

EXAMPLE 18.1. How many different individual symbols do the binary, octal, and hexadecimal systems have? What is the numerical range of these symbols?

In the binary system the basis is $N=2$; therefore, there are 2 different individual symbols. The values of these symbols range from 0 to $(2-1)$, that is, from 0 to 1. These symbols are 0 and 1. In computer jargon, these symbols are called the 0 bit and 1 bit, respectively. The word *bit* is an abbreviated form of the words binary digit.

In the octal system the basis is $N=8$; therefore, there are 8 different individual symbols. The values of these symbols range from 0 to $(8-1)$, that is, from 0 to 7. These symbols are 0, 1, 2, 3, 4, 5, 6, and 7. We call all of these symbols by their decimal names. These are the familiar names one, two, three, and so on.

In the hexadecimal system the basis is $N=16$; therefore, there are 16 different individual symbols. The values of these symbols range from 0 to $(16-1)$, that is, from 0 to 15. These symbols are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. Notice that after 9, the symbols are the letters A through F. In this system, the A stands for 10, the B for 11, the C for 12, the D for 13, the E for 14, and the F for 15. The reason for choosing letters instead of symbols to represent symbols higher than 9 is to keep all symbols single characters. We call the symbols 0 through 9 by their decimal names. The letters are called by their common names.

To explicitly state the basis on which a number is written, we will use a subscript to the lower right side of the number. The format of the subscript is "(basis." For example, $1010_{(2}$ indicates that the number is binary. Likewise, $145_{(10}$ is a decimal number, $1A2_{(16}$ is a hexadecimal number, and $145_{(8}$ is an octal number.

Although the word digit generally refers to the individual symbols of the decimal system, it is also common to call the individual symbols of the other numerical systems by this name.

18.2 NUMERICAL VALUES IN POSITIONAL SYSTEMS AND THEIR DECIMAL EQUIVALENTS

All the numerical systems that we have considered so far, including the decimal system, are positional systems. That is, the value represented by a digit in the numerical representation of a number always depends on its position in the number. For example, in the decimal system, the 5 in 157 represents the value 50 units; the 5 in 548 represents 500 units.

In the decimal system, we call the rightmost position of a number the ones (10^0) place, the next position from the right the tens (10^1) place, the next position the hundreds (10^2) place, and so on. Notice that the powers increase from the right. The power of the rightmost digit is zero, the next digit has a power of one, the next a power of two, and so on. These powers are sometimes called the *weight* of the digit. To better visualize this we generally write superscripts beginning with 0 (the rightmost digit of the number) and increasing the powers by 1 as we move toward the left digits of the number.

In any positional system, *the decimal value of a digit in the representation of a number is the digit's own value, in decimal, multiplied by a power of the basis in which the number is represented. The sum of all these powers is the decimal equivalent of the number.*

The following example illustrates this.

EXAMPLE 18.2. What is the value of each digit in the number $1554_{(10}$?

Notice that the basis is 10 since we are working in the decimal system. Writing superscripts beginning with 0, on the rightmost digit, and increasing them as we move toward the right, the number would look like this: $1^3 5^2 5^1 4^0$. Using these superscripts as the power of the basis:

$$1554 = (1 \times 10^3) + (5 \times 10^2) + (5 \times 10^1) + (4 \times 10^0) \quad \text{Remember that } 10^0 \text{ is equal to 1.}$$
$$= (1 \times 1000) + (5 \times 100) + (5 \times 10) + (4 \times 1)$$
$$= 1000 + 500 + 50 + 4$$

Observe that reading the number from left to right, the 1 represents 1000 units, the first 5 represents 500 units, the next 5 represents 50 units and the 4 represents the single units.

EXAMPLE 18.3. What is the value of each digit in the number $1554_{(8}$? What is the decimal equivalent of the number?

Notice that the basis is 8 since we are working with the octal system. Using a similar procedure as in the previous example, we can use superscripts and write the number as $1^3 5^2 5^1 4^0$. Using these superscripts as the power of the basis

$$1554_{(8} = (1 \times 8^3) + (5 \times 8^2) + (5 \times 8^1) + (4 \times 8^0) \quad \text{Remember that } 8^0 \text{ is equal to 1.}$$
$$= (1 \times 512) + (5 \times 64) + (5 \times 8) + (4 \times 1)$$
$$= 512 + 320 + 40 + 4$$
$$= 876_{(10}$$

We can observe that reading the octal number from left to right, the 1 represents 512 units, the first 5 represents 320 units, the next 5 represents 40 units and the 4 represents the single units. The decimal equivalent is 876.

EXAMPLE 18.4. What is the value of each digit in the number $1554_{(16}$? What is the decimal equivalent of the number?

Notice that the basis is 16 since we are working with the hexadecimal system. Using superscripts as the power of the basis we can write the number as $1^3 5^2 5^1 4^0$. Therefore, the hexadecimal values of each digit can be calculated as follows:

$$1554_{(16} = (1 \times 16^3) + (5 \times 16^2) + (5 \times 16^1) + (4 \times 16^0) \quad \text{Remember that } 16^0 \text{ is equal to 1}$$
$$= (1 \times 4096) + (5 \times 256) + (5 \times 16) + (4 \times 1)$$
$$= 4096 + 1280 + 80 + 4$$
$$= 5460_{(10}$$

Observe that reading the hexadecimal number from left to right, the 1 represents 4096 units, the first 5 represents 1280 units, the next 5 represents 80 units and the 4 represents the single units. The decimal equivalent is 5460.

Examples 18-2 through 18-4 show that, depending on the basis, numbers with identical representation may have different decimal equivalent values.

EXAMPLE 18.5. What is the value of each digit in the number $11001_{(2}$? What is the decimal equivalent of the number?

The number can be written as $1^4 1^3 0^2 0^1 1^0$ using superscripts. The value of each digit can be calculated as follows:

$$\begin{aligned} 11001_{(2} &= (1\times 2^4)+(1\times 2^3)+(0\times 2^2)+(0\times 2^1)+(1\times 2^0) \quad \text{Remember that } 2^0 \text{ is equal to 1} \\ &= (1\times 16)+(1\times 8)+(0\times 4)+(0\times 2)+(1\times 1) \\ &= (16)+(8)+(0)+(0)+1 \\ &= 25_{(10} \end{aligned}$$

Notice that the basis is 2 since we are working with the binary system. Observe that reading the binary number from left to right, the first 1 represents 16 units, the second 1 represents 8 units, and the last 1 represents the single units. The decimal equivalent of the number is 25.

EXAMPLE 18.6. What is the value of each digit in the number $1AB2_{(16}$? What is the decimal equivalent of the number?

Using superscripts the number can be written as $1^3 A^2 B^1 2^0$. The value of each digit can be calculated as follows:

$$\begin{aligned} 1AB2_{(16} &= (1\times 16^3)+(10\times 16^2)+(11\times 16^1)+(2\times 16^0) \quad \text{Remember that } 16^0 \text{ is equal to 1} \\ &= (1\times 4096)+(10\times 256)+(11\times 16)+(2\times 1) \\ &= (4096)+(2560)+(176)+2 \\ &= 6834_{(10} \end{aligned}$$

The basis is 16 since we are working with the hexadecimal system. Notice that we have replaced A and B by their decimal equivalents 10 and 11, respectively. Observe also that reading the hexadecimal number from left to right, the first 1 represents 4096 units, the A represents 2560 units, the B represents 176 units and the 2 represents the single units. The decimal equivalent of the number is 6834.

18.3 CONVERSION OF DECIMAL NUMBERS TO OTHER BASES

So far, we have seen how to obtain the equivalent decimal value of numbers expressed in the binary, octal, or hexadecimal systems. In this section we will learn how to express a given decimal number in any of these bases. Before converting decimal numbers to the binary system, remember that when we divide a number (called the dividend) by another number (called the divisor), we will obtain a quotient and a remainder. If the remainder is zero, we say that the division is exact; otherwise we say that the division is not exact. If we call the dividend (D), the divisor (d), the quotient (q) and the remainder (r), the following formula always holds true:

$$D = d \times q + r \qquad \text{where} \qquad 0 \le r < q$$

With this in mind, we can now proceed to describe an algorithm or method to carry out the conversion of decimal numbers to binary. This process consists primarily of dividing a given number by the basis of the system to which we want to convert it until we get a quotient of zero. The remainders of the different divisions (in the opposite order to that in which they were obtained) will give the representation of the decimal number in the new basis. Algorithm 1 formalizes this process.

Algorithm 1 (Conversion from decimal to any nonnegative integer basis)

Step 1. Set the value of $i = 0$. Call N_i the given decimal number.

Step 2. Divide N_i by the new basis. Obtain the quotient q_i and the remainder r_i.

Step 3. Check the value of the quotient q_i. If the quotient q_i is zero continue with step 5 otherwise continue with step 4.

Step 4. Increase the value of i by 1. Set N_i equal to $q_{(i-1)}$ and go back to step 2.

Step 5. The equivalent number (in the new basis) to the given decimal number is obtained by concatenating (writing next to each other) the remainders r_i obtained in the opposite order in which they were obtained. That is, the equivalent number (in the new basis) is $r_i r_{i-1} r_{i-2} \cdots r_0$. Notice that the subscripts decrease from left to right.

EXAMPLE 18.7. What is the binary representation of $25_{(10}$?

The new basis is 2.

Step 1. Set $i = 0$ and call $N_0 = 25$.

Step 2. Divide $N_0 = 25$ by 2, the new basis. Obtain the quotient q_0 and the remainder r_0. That is, divide 25 by 2. Obtain $q_0 = 12$ and $r_0 = 1$.

Step 3. Check that q_0 is not zero. In this case, $q_0 = 12$, therefore it is not zero.

Step 4. Set $i = 1$ (i is incremented by 1).

$N_1 = q_{i-1} = q_0 = 12$ (N_i is set to $q_{(i-1)}$).

Go back to step 2 of the algorithm.

Step 2 (2nd time). Divide $N_1 = 12$ by 2. Obtain $q_1 = 6$ and $r_1 = 0$.

Step 3 (2nd time). $q_1 = 6$. Therefore, q_1 is not zero.

Step 4 (2nd time). Set $i = 2$ (i is incremented by 1).

Set $N_2 = q_{2-1} = q_1 = 6$.

Go back to step 2 of the algorithm.

Step 2 (3rd time). Divide $N_2 = 6$ by 2. Obtain $q_2 = 3$ and $r_2 = 0$.

Step 3 (3rd time). $q_2 = 3$. Therefore, q_2 is not zero.

Step 4 (3rd time). Set $i = 3$ (i is incremented by 1).

Set $N_3 = q_{3-1} = q_2 = 3$.

Go back to step 2 of the algorithm.

Step 2 (4th time). Divide $N_3 = 3$ by 2. Obtain $q_3 = 1$ and $r_3 = 1$.

Step 3 (4th time). $q_3 = 1$. Therefore, q_3 is not zero.

Step 4 (4th time). Set $i = 4$ (i is incremented by 1).

Set $N_4 = q_{4-1} = q_3 = 1$.

Go back to step 2 of the algorithm.

Step 2 (5th time). Divide $N_4 = 1$ by 2. Obtain $q_4 = 0$ and $r_4 = 1$.

Step 3 (5th time). q_4 is zero.

Step 5. The equivalent binary number is obtained by concatenating the remainders of the divisions in the opposite order to that in which they were obtained. The equivalent binary number is $N_0 = 25_{(10} = r_4 r_3 r_2 r_1 r_0 = 11001_{(2}$.

We can verify this result as follows:

$$\begin{aligned} 1^4 1^3 0^2 0^1 1^0_{(2} &= (1 \times 2^4) + (1 \times 2^3) + (0 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) \\ &= (1 \times 16) + (1 \times 8) + (0 \times 4) + (0 \times 2) + (1 \times 1) \\ &= 16 + 8 + 0 + 0 + 1 \\ &= 25 \end{aligned}$$

The steps of this algorithm can also be illustrated in an abbreviated form as follows:

Number	Quotient when dividing by 2	Remainder
25	12	1
12	6	0
6	3	0
3	1	1
1	0	1

Writing the remainders in opposite order, we have that $25_{(10} = 11001_{(2}$.

EXAMPLE 18.8. What is the octal representation of decimal 125?

The new basis is 8.

Step 1. Set $i = 0$ and $N_0 = 125$.

Step 2. $125 \div 8$. Obtain $q_0 = 15$ and $r_0 = 5$.

Step 3. q_0 is not zero.

Step 4. Set $i = 1$ (i is incremented by 1).

$N_1 = q_{i-1} = q_0 = 15$ (N_i is set to $q_{(i-1)}$).

Go back to step 2 of the algorithm.

Step 2 (2nd time). $15 \div 8$. Obtain $q_1 = 1$ and $r_1 = 7$.

Step 3 (2nd time). q_1 is not zero.

Step 4. Set $i = 2$.

Set $N_2 = q_{2-1} = q_1 = 1$.

Go back to step 2 of the algorithm.

Step 2 (3rd time). $1 \div 8$. Obtain $q_2 = 0$ $r_2 = 1$.

Step 3 (3rd time). q_2 is zero.

Step 5. The equivalent octal number is obtained by concatenating the remainders of the divisions in the opposite order to that in which they were obtained. Therefore $N_0 = 125_{(10} = r_2 r_1 r_0 = 175_{(8}$.

We can verify this result as follows:

$$\begin{aligned} 1^2 7^1 5^0{}_{(8} &= (1 \times 8^2) + (7 \times 8^1) + (5 \times 8^0) \\ &= (1 \times 64) + (7 \times 8) + (5 \times 1) \\ &= 64 + 56 + 5 \\ &= 125_{(10} \end{aligned}$$

The steps of the algorithm can be illustrated in an abbreviated form as follows:

Number	Quotient when dividing by 8	Remainder
125	15	5
15	1	7
1	0	1

Writing the remainders in opposite order, we have that $125_{(10} = 175_{(8}$.

EXAMPLE 18.9. What is the hexadecimal representation of $48_{(10}$?

The new basis is 16.

Step 1. Set $i = 0$ and $N_0 = 48$.

Step 2. $48 \div 16$. Obtain $q_0 = 3$ and $r_0 = 0$.

Step 3. q_0 is not zero.

Step 4. Set $i = 1$ (i is incremented by 1).

Set $N_i = q_{i-1} = q_0 = 3$ (N_i is set to $q_{(i-1)}$).

Go back to step 2 of the algorithm.

Step 2 (2nd time). $3 \div 16$. Obtain $q_1 = 0$ and $r_1 = 3$.

Step 3 (2nd time). q_1 is zero.

Step 5. The equivalent hexadecimal number is obtained by concatenating the remainders of the divisions in the opposite order to that in which they were obtained. The equivalent hexadecimal number is $N_0 = 48_{(10} = r_1 r_0 = 30_{(16}$.

We can verify this result as follows:

$$\begin{aligned} 3^1 0^0_{(16} &= (3 \times 16^1) + (0 \times 16^0) \\ &= (3 \times 16) + (0 \times 1) \\ &= 48 + 0 \\ &= 48_{(10} \end{aligned}$$

The steps of the algorithm can be illustrated in an abbreviated form as follows:

Number	Quotient when dividing by 16	Remainder
48	3	0
3	0	3 ↑

Writing the remainders in opposite order, we have that $48_{(10} = 30_{(8}$.

18.4 CONVERSION BETWEEN HEXADECIMAL AND BINARY NUMBERS

Conversion between these two systems can be accomplished as a two-step process. If the number is hexadecimal we first convert it to decimal and then to binary. If the number is binary we first convert it to decimal and then to hexadecimal. However, there is an even faster process for converting numbers between these two bases. As Table 18.1 shows, with four binary digits we can represent the numbers 0 through 15. Since each four-bit binary number corresponds to one and only one hexadecimal digit and vice versa, we can view the hexadecimal system as a shorthand notation of the binary system.

Algorithm 2 gives the process for converting binary numbers to their hexadecimal equivalents.

Algorithm 2 (Conversion from binary to hexadecimal)

Step 1. Form four-bit groups beginning from the rightmost bit of the number. If the last group (at the leftmost position) has fewer than four bits, add extra zeros to the left of the bits in this group to make it a four-bit group.

Step 2. Replace each four-bit group by its hexadecimal equivalent.

EXAMPLE 18.10. What is the hexadecimal equivalent of the binary number 101011110000101111010001111?

Table 18.1. Decimal, Binary, Octal and Hexadecimal Equivalents

Decimal	Binary	Octal	Hexadecimal
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Step 1. Forming four-bit groups beginning from the rightmost bit we obtain

10 1011 1100 0010 1111 0100 0111

Since the last group (the leftmost) has only two bits we add two extra zeros to the left of these bits to make it a four-bit group. The number now looks like this:

0010 1011 1100 0010 1111 0100 0111 ← Notice the extra zeros added to the leftmost group to make it a 4-bit group.

Step 2. Replacing each group by its hexadecimal equivalent (see Table 18.1) we obtain 2BC2F47.

The procedure to convert hexadecimals to binary is basically the reverse of the previous algorithm. However, there is no need to add extra zeros since each hexadecimal number will always convert to a group with four binary bits.

EXAMPLE 18.11. What is the binary equivalent to the hexadecimal number 3ACBD2?

Replacing each hexadecimal number by its binary equivalent (see Table 18.1) we will obtain

0011 1010 1100 1011 1101 0010

We have separated the binary number in groups of four bits to facilitate the reading of the resulting binary number.

18.5 RULES FOR FORMING NUMBERS IN ANY SYSTEM

Table 18.1 shows the decimal equivalent of some numbers in binary, octal, and hexadecimal. In any of these systems how do we form consecutive numbers higher than their largest individual symbol? Let us examine the decimal system. First, notice that with a single digit we can only represent the numbers 0 through 9. To represent numbers exceeding nine we use more than one digit as indicated below:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37

38 39 ... 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110

Beginning at the left of the top row, we have written all single digits. Since there is no single digit to represent numbers higher than 9, we form all the two-digit combinations of numbers beginning with 1. Then we form all two-digit combinations that begin with 2 and so on until we reach 99. After exhausting all two-digit combinations we proceed to form combination of three digits. Once we have exhausted all three-digit combinations we proceed with four digits. This process can be continued forever. This rule can be applied to any other numerical system. Table 18.1 shows that we have followed a similar procedure for representing numbers higher than 7 in the octal system.

EXAMPLE 18.12. In the hexadecimal system, the sequence of numbers that exceed F (or decimal 15) is

10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 40 41 and so on.

Solved Problems

18.1. If 0, 1, 2, 3, 4 are the single digits of a numbering system, what is the smallest basis to have such digits?

Notice that the range of numbers goes from 0 to 4. Since 4 is the largest, the minimum basis is 5. Remember that for a given basis N, there are $N-1$ different symbols with a range from 0 to $(N-1)$.

18.2. What is the decimal equivalent of $1631_{(8}$?

$$\begin{aligned} 1631_{(8} &= (1\times 8^3)+(6\times 8^2)+(3\times 8^1)+(1\times 8^0) \\ &= (1\times 512)+(6\times 64)+(3\times 8)+(1\times 1) \\ &= 512+384+24+1 \\ &= 921_{(10} \end{aligned}$$

18.3. What is the decimal equivalent of $333_{(5}$?

$$\begin{aligned} 333_{(5} &= (3\times 5^2)+(3\times 5^1)+(3\times 5^0) \\ &= (3\times 25)+(3\times 5)+(3\times 1) \\ &= 75+15+3 \\ &= 93_{(10} \end{aligned}$$

18.4. Is 13481 the representation of a valid octal number?

No! This number cannot be an octal number since the largest single digit in this base is a 7 and this number has an 8.

18.5. What is the decimal representation of $1100111_{(2}$?

$$\begin{aligned}1100111_{(2} &= (1\times 2^6)+(1\times 2^5)+(0\times 2^4)+(0\times 2^3)+(1\times 2^2)+(1\times 2^1)+(1\times 2^0)\\ &= (1\times 64)+(1\times 32)+(0\times 16)+(0\times 8)+(1\times 4)+(1\times 2)+(1\times 1)\\ &= 64+32+0+0+4+2+1\\ &= 103_{(10}\end{aligned}$$

18.6. What is the binary equivalent of $FC2B_{(16}$?

Replacing each decimal number by its binary equivalent (see Table 18.1) we obtain

F C 2 B

1111 1100 0010 1011

Therefore, $FC2B_{(16} = 1111110000101011_{(2}$.

18.7. What is the hexadecimal equivalent of $1011101001110101111_{(2}$?

Forming four-bit groups beginning with the rightmost bit and adding extra zeros to the leftmost group, if necessary, to make it a 4-bit group, we obtain

$1011101001110101111_{(2} = 0101\ 1101\ 0011\ 1010\ 1111$ ← An extra zero was added to the leftmost group.

Replacing each group by its hexadecimal equivalent (see Table 18.1) we obtain that

$$1011101001110101111_{(2} = 5D3AF_{(16}$$

18.8. What's the octal equivalent to $11011000101_{(2}$?

Since we are working with the octal basis, we can follow a similar algorithm to the one used in Example 18.9. However, instead of forming groups of four bits, we form three-bit groups beginning with the rightmost bit and add extra digits to the leftmost group if necessary. Proceeding this way, we can form groups of 3-bits as follows:

$11011000101_{(2} = 011\ 011\ 000\ 101$ ← A zero was added to the leftmost group.

Replacing each group by its octal equivalent (see Table 18.1), we obtain

$$\begin{aligned}11011000101_{(2} &= 011\ 011\ 000\ 101\\ &= 3305_{(8}\end{aligned}$$

18.9. What is the binary equivalent of $31_{(10}$?

Since we want to express the number in binary we need to apply Algorithm 1 with 2 as the basis.

Step 1. Set $i = 0$ and $N_0 = 31$.

Step 2. $31 \div 2$. Obtain $q_0 = 15$ and $r_0 = 1$.

Step 3. q_0 is not zero.

Step 4. Set $i = 1$.

Set $N_1 = q_{i-1} = q_0 = 15$.

Go back to step 2 of the algorithm.

Step 2 (2nd time). $15 \div 2$. Obtain $q_1 = 7$ and $r_1 = 1$.

Step 3 (2nd time). q_1 is not zero.

Set $i = 2$.

Step 4 (2nd time). $N_2 = q_{i-1} = q_1 = 7$.

Go back to step 2 of the algorithm.

Step 2 (3rd time). $7 \div 2$. Obtain $q_2 = 3$ and $r_2 = 1$.

Step 3 (3rd time). q_2 is not zero.

Step 4 (3rd time). Set $i = 3$.

$N_3 = q_{i-1} = q_2 = 3$.

Go back to step 2 of the algorithm.

Step 2 (4th time). $3 \div 2$. Obtain $q_3 = 1$ and $r_3 = 1$.

Step 3 (4th time). q_3 is not zero.

Step 4 (4th time). Set $i = 4$.

$N_4 = q_{i-1} = q_3 = 1$.

Go back to step 2 of the algorithm.

Step 2 (5th time). $1 \div 2$. Obtain $q_4 = 0$ and $r_4 = 1$.

Step 3 (5th time). q_4 is zero.

Step 5. The equivalent binary number is obtained by concatenating the remainders of the divisions in the opposite order to that in which they were obtained. The equivalent binary number is $N_0 = 31_{(10} = r_4 r_3 r_2 r_1 r_0 = 11111_{(2}$.

We can verify this result as follows:

$$\begin{aligned} 1^4 1^3 1^2 1^1 1^0_{(2} &= (1 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) \\ &= (1 \times 16) + (1 \times 8) + (1 \times 4) + (1 \times 2) + (1 \times 1) \\ &= 16 + 8 + 4 + 2 + 1 \\ &= 31_{(10} \end{aligned}$$

The conversion process using the abbreviated form of the algorithm is shown below.

Number	Quotient when dividing by 2	Remainder
31	15	1
15	7	1
7	3	1
3	1	1
1	0	1

Writing the remainders in opposite order, we have that $31_{(10} = 11111_{(2}$.

18.10. In any numerical system, what does a 10 represent?

This question seems trivial since we "think" in decimal. However, the answer depends upon the numbering system being considered. Looking at Table 18.1, observe that under each of the columns labeled binary and octal there is a 10. The binary 10 represents the decimal value 2. The octal 10 represents the decimal value 8. Notice that in all cases 10 represents the basis of the system in consideration. This is true in any other numbering system including the decimal and hexadecimal system.

18.11. How many different binary numbers can be represented with four bits? What are their decimal equivalents?

We can begin by forming all possible combinations using all four bits and then finding their decimal equivalent. Looking at Table 18.1, we can see that there are 16 different binary numbers. There is a rule that can give use the maximum number of different numbers that can be represented with N bits. We can state it as follows: *Given N bits, it is possible to form 2^N different binary numbers. The range of these values is from 0 to $2^N - 1$.*

18.12. Given a sequence of seven bits, what is the range of values that can be represented in the binary system? How many different numbers are there?

Since $N = 7$, it is possible to form 2^7 or 128 different binary numbers. The range of these values will vary from 0 to $2^7 - 1$. That is, from 0 to 127.

18.13. What is the binary equivalent of $113_{(10}$?

Using the abbreviated form of Algorithm 1, we have

Number	Quotient when dividing by 2	Remainder
113	56	1 ↑
56	28	0 |
28	14	0 |
14	7	0 |
7	3	1 |
3	1	1 |
1	0	1

Writing the remainders in the opposite order to that in which they were obtained, we have that $113_{(10} = 1110001_{(2}$.

This can be verified as follows:

$$\begin{aligned} 1110001_{(2} &= (1\times 2^6)+(1\times 2^5)+(1\times 2^4)+(0\times 2^3)+(0\times 2^2)+(0\times 2^1)+(1\times 2^0) \\ &= (1\times 64)+(1\times 32)+(1\times 16)+(0\times 8)+(0\times 4)+(0\times 2)+(1\times 1) \\ &= 64+32+16+0+0+0+1 \\ &= 113_{(10} \end{aligned}$$

18.14. What is octal representation of $111_{(10}$?

Using the abbreviated form of Algorithm 1, we have

Number	Quotient when dividing by 8	Remainder
111	13	7 ↑
13	1	5 |
1	0	1

Writing the remainders in the opposite order to that in which they were obtained, we have that $111_{(10} = 157_{(8}$.

We can verify this result as follows:

$$157_{(8} = (1 \times 8^2) + (5 \times 8^1) + (7 \times 8^0)$$
$$= (1 \times 64) + (5 \times 8) + (7 \times 1)$$
$$= 64 + 40 + 7$$
$$= 111_{(10}$$

18.15. What is the hexadecimal representation of $142_{(10?}$

Using the abbreviated form of Algorithm 1, we have

Number	Quotient when dividing by 16	Remainder
142	8	14
8	0	8

Writing the remainders in the opposite order to that in which they were obtained, and keeping in mind that E stands for 14 in the hexadecimal system, we have that $142_{(10} = 8E_{(16}$.

This result can be verified as follows:

$$8E_{(16} = (8 \times 16^1) + (14 \times 16^0)$$
$$= (8 \times 16) + (14 \times 1)$$
$$= 142_{(10}$$

Supplementary Problems

The answers to this set of problems are at the end of this chapter.

18.16. Given the decimal equivalent of the following binary numbers:

(*a*) 1100011101 (*b*) 11001100111 (*c*) 100010001 (*d*) 11110001111 (*e*) 1111111

18.17. Give the decimal equivalent of the following hexadecimal numbers:

(*a*) F1AB (*b*) ABCD (*c*) 5942 (*d*) 1A2B (*e*) 6DCE

18.18. Convert the following decimal numbers to binary:

(*a*) 1949 (*b*) 750 (*c*) 117 (*d*) 432 (*e*) 89

18.19. Convert the following hexadecimal numbers to binary:

(*a*) B52 (*b*) FAFA (*c*) 512 (*d*) CAFE

18.20. Convert the following hexadecimal numbers to octal:

(*a*) 35AB (*b*) F2CA (*c*) 853F (*d*) CAB (*e*) 7A8B

18.21. Convert the following octal number to decimal:

(*a*) 7714 (*b*) 3532 (*c*) 6320 (*d*) 1515 (*e*) 4050

18.22. Given a sequence of bits representing a number, how can you tell without making any conversion that the decimal equivalent is even or odd?

Answers to Supplementary problems

18.16. (*a*) 797 (*b*) 1639 (*c*) 273 (*d*) 1935 (*e*) 127

18.17. (*a*) 61867 (*b*) 43981 (*c*) 22850 (*d*) 6699 (*e*) 28110

18.18. (*a*) 11110011101 (*b*) 1011101110 (*c*) 1110101 (*d*) 110110000 (*e*) 1011001

18.19. (*a*) 101101010010 (*b*) 1111101011111010 (*c*) 10100010010 (*d*) 1100101011111110

18.20. (*a*) 32653 (*b*) 171312 (*c*) 102477 (*d*) 6253 (*e*) 75213

18.21. (*a*) 4044 (*b*) 1882 (*c*) 3280 (*d*) 845 (*e*) 2088

18.22. If the rightmost bit of the binary sequence is 1 its decimal equivalent is odd. Otherwise, its decimal equivalent is even.

Chapter 19

Arithmetic Operations in a Computer

19.1 REVIEW OF BASIC CONCEPTS OF ARITHMETIC

Before considering arithmetic operations in the binary, octal, or hexadecimal systems, let us review these operations in the decimal system. We will begin with a simple addition as shown below. Although the explanation may seem trivial, some generalization can be made to all other numerical systems to facilitate their understanding.

EXAMPLE 19.1. The result of adding the decimal numbers 194 and 271 is 465. How did we obtain this result?

We began by aligning the numbers by their rightmost digits. We then added the numbers beginning with the rightmost digits.

```
194 +
271
---
  5 ← 4 plus 1 is 5. Notice that there is a symbol for the number five.
```

```
1   ← carry of 1
194 +
271
---
 65 ← 9 plus 7 is 16. Notice that there is no single symbol for the number sixteen. Write 6 and carry 1 since we know
      that 16 = 10 + 6.
```

At this moment we may ask, Why do we write 6? Why do we carry 1? First remember that the single digits go from 0 to 9. Since we have exceeded 9 we need to use two digits. The rules for the formation of numbers of the previous chapter show that 6 "accompanies" the tens digit that we carry. Notice also that we can write 16 as $16 = 1 \times 10 + 6$. Therefore, we write 6 and carry 1. Finally,

```
1   ← carry of 1.
194 +
271
---
465 ← 1 + 1 is 2. Then 2 plus 2 is 4. There is a single symbol for the number four.
```

EXAMPLE 19.2. Subtract decimal 105 from decimal 4002.

We start by aligning the rightmost digits of the numbers. The following sequence explains the process.

```
   12
4002 −
 105
----
   7 ← Since 2 is less than 5 we "borrow" 1 unit from the digit to the left of the 2. The borrowed unit is equivalent to
       "borrowing 10." Therefore, write 7 since 12 − 5 = 7.
```

```
  9
4002 − ← The 0 to the left of the 2 became a 9 (the basis minus 1).
 105
----
  97 ← Write 9 since 9 − 0 = 9.
```

```
 9
4002 −
 105
-----
 897 ← Write 8 since 9 − 1 = 8.
```

```
 3
4002 − ← The 4 "paid" the 10 that was initially borrowed by the rightmost 2. The 4 decreases to 3.
 105
-----
3897 ← Write 3 since 3 − 0 = 3.
```

19.2 ADDITION AND SUBTRACTION OF BINARY NUMBERS

When adding or subtracting binary numbers, we can follow a process similar to the one used for adding or subtracting decimal numbers. Using the rules of addition shown in Table 19-1, let us illustrate this with an example.

Table 19-1. Addition of Binary Numbers

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 0 with a carry of 1

EXAMPLE 19.3. Add the binary numbers 01010 and 10111.

We begin by aligning the rightmost digits of the number; therefore,

```
01010 +
10111
-----
    1 ← 0 plus 1 is 1. There is a single symbol to represent one in binary.
```

```
01010 + ← "Thinking in decimal" we have that 1 plus 1 is 2. There is not a single symbol for the number two in binary.
10111       Two is written as 10 in binary. Therefore, we write 0 and carry 1 as indicated in Table 19.1. Notice that
-----       in "decimal" 2 = 1 × 2 + 0. Therefore, write 0 and carry 1.
   01 ← Write 0 and carry 1.
```

```
  1
01010 + ← 1 (the carry) plus 0 is 1. This 1 plus 1 is 2. Therefore, write 0 and carry 1.
10111
-----
  001 ← Write 0 and carry 1.
```

```
 1
01010 + ← 1 (the carry) plus 1 is 2. This 2 plus 0 is 2. Therefore, write 0 and carry 1.
10111
-----
 0001
```

```
 1
 01010 +  ← 1 (the carry) plus 0 is 1. This 1 plus 1 is 2. Therefore, write 0 and carry 1.
 10111
------
100001 ← Write 0 and the carry 1.
```

Subtraction of binary numbers is similar to addition, except that we may generate a "borrow" of "2" from the higher digits (those to the left of a particular digit). Example 19.4 illustrates this.

EXAMPLE 19.4. Subtract 111 from 10001.

After aligning the rightmost digits of the number as shown below, we have

```
10001 −
  111
-----
    0 ←Write 0 since 1 − 1 = 0.

  "10"
10001 −  ← Since we cannot subtract 1 from 0 we "borrow" 1 unit from the next higher unit. This is equivalent to
  111        borrowing "2." Notice that if we were working with the decimal system, in a similar situation, we say that
-----        we borrow "ten" from the next higher place.
   10 ← Write 1 since 2 − 1 = 1.

   1
10001 −  ← The next 0 becomes 1 (the basis minus 1).
  111
-----
  010 ← Write 0 since 1 − 1 = 0.

  1
10001 −  ← The next 0 also becomes 1 (the basis minus 1).
  111
-----
 1010 ← Write 1 since 1 − 0 = 1. Notice that the 0 in the subtrahend is not shown but assumed.

 0
10001 −  ← The 1 "paid" the unit that was borrowed by the rightmost 1. This 1 becomes 0.
  111
-----
01010 ← Write 0 since 0 − 0 = 0.
```

19.3 ADDITION AND SUBTRACTION OF HEXADECIMAL NUMBERS

Operating with a long string of 0s and 1s is an error-prone task. For this reason it is preferable to carry out arithmetic operations in the hexadecimal system. As we have seen, we can easily transform numbers between these two bases. Adding hexadecimal numbers is a process similar to that of adding decimal or binary numbers. However, it is necessary to remember that there is a value carried whenever the sum exceeds F hexadecimal or 15 decimal. In subtraction operations, whenever necessary, we may borrow "16" from the higher units "to the left." To simplify the process it is convenient to think in "decimal." Therefore, in our heads we may translate letters into their decimal equivalent, do the operations in decimal, and translate the results back to hexadecimal. The following examples illustrate this process.

EXAMPLE 19.5. What is the sum of the following hexadecimal numbers: 4C1A and 12B3?

After aligning the numbers on their rightmost digit we have

```
4C1A +
12B3
----
   D  ← A is equivalent to 10. 10 plus 3 is 13. In hexadecimal 13 is written D. Therefore, write D.

4C1A +
12B3
----
  CD  ← B is equivalent to 11. 1 plus 11 is 12. In hexadecimal 12 is written C. Therefore, write C.

4C1A +
12B3
----
 ECD  ← C is equivalent to 12. 12 plus 2 is 14. In hexadecimal 14 is written E. Therefore, write E.

4C1A +
12B3
----
5ECD  ← 4 plus 1 is 5. Therefore, write 5.
```

EXAMPLE 19.6. Add $EDF4_{(16}$ and $223_{(16}$.

After aligning the numbers on their rightmost digit we have that

```
EDF4 +
 223
----
   7  ← 4 plus 3 is 7.

EDF4 +
 223
----
  17  ← F is equivalent to 15. 15 plus 2 is 17. Since 17 = 1 × 16 + 1 write 1 and carry 1.

 1    ← carry
EDF4 +
 223
----
 017  ← D is equivalent to 13. 13 plus 1 (the carry) is 14. 14 plus 2 is 16. Since 16 = 1 × 16 + 0. Write 0 and carry 1.

1     ← carry
EDF4 +
 223
----
F017  ← E is equivalent to 14. 1 (the carry) plus 14 is 15. Write F since its decimal equivalent is 15.
```

EXAMPLE 19.7. What is the result of subtracting $51AF_{(16}$ from $F3002_{(16}$?

```
  "18"
F3002 −
 51AF
-----
    3  ← F is equivalent to 15. We cannot subtract 15 from 2. Borrow "16" from the higher order unit to the left
         of the 2. This 16 plus 2 is 18. Write 3 since 18 − 15 = 3.

   F
F3002 −
 51AF
-----
   53  ← The 0 next to the rightmost 2 became an F (the basis minus 1). A is equivalent to 10. Write 5 since 15 − 10 = 5.
```

```
  F
F3002 −
 51AF
 ----
 E53  ← The next 0 also became an F (the basis minus 1). Since 15 − 1 = 14, write E since its equivalent is decimal 14.

  2
F3002 −
 51AF
 ----
 DE53 ← The 3 "paid" the "16" that was "borrowed" by the rightmost 2. The 3 decreases to a 2. Since we cannot
        subtract 5 from 2, we need to "borrow" 16 from the higher unit to the left. This 16 plus 2 is 18. Write D since
        18 − 5 = 13. D in hexadecimal is 13.

  E
F3002 −
 51AF
 ----
EDE53 ← The next F "paid" the "16" that was "borrowed" by the 2 to its right. The F became an E. Since 14 − 0 = 14,
        write E since in hexadecimal E is 14.
```

19.4 REPRESENTING NONNEGATIVE INTEGERS IN A COMPUTER

Numerical data in a computer is written in basic "units" of storage made up of a fixed number of consecutive bits. The most commonly used units throughout the computer and communication industries are shown in Table 19-2. A number is represented in each of these units by setting the bits according to the binary representation of the number. By convention the bits in a *byte* are numbered right to left, beginning with zero. Thus, the rightmost bit is bit number 0 and the leftmost bit is number 7. The rightmost bit is called the least significant bit. Likewise, the leftmost bit is called the most significant bit. Higher units are numbered also from right to left. In general, the rightmost bit is labeled 0, the leftmost bit is labeled $(n - 1)$ where n is the number of bits available.

Since each bit may have one of two values, 0 or 1, there are 2^8 configurations in a byte. That is, it is possible to represent 256 different binary numbers. The range of these nonnegative numbers varies from 0 to 255. In general, given n bits there are 2^n different numbers that can be represented. The range of these nonnegative integers varies from 0 to $2^n - 1$.

Table 19-2.

Units	Number of bits
A byte	8 consecutive bits
A word	16 consecutive bits
A double word	32 consecutive bits

EXAMPLE 19.8. How many different binary numbers can be represented in a word or a double word? What is the range of these nonnegative values?

Since a word has 16 bits, $n = 16$. Therefore, there are 2^{16} or 65,536 different representations of binary numbers. The range of nonnegative numbers is from 0 to $2^{16} - 1$. That is, from 0 to 65,535. Since a double word has 32 bits, $n = 32$. Thus, it is possible to represent 2^{32} or 4,294,967,296 different binary numbers. The range of these values is from 0 to 4,294,967,295.

19.5 COMPUTER ADDITION

Addition of binary numbers in a computer is carried out with a fixed amount of storage. Therefore, when adding two n-bit numbers the result may have $n + 1$ bits. Since this number is too big to fit in n bits we say that an *overflow* condition has occurred. In the next sections we will consider a more general condition for detecting overflows.

EXAMPLE 19.9. Add the binary numbers 10010101 and 11001011 assuming that the numbers are represented in a byte. Does this addition result in an overflow condition?

```
 10010101 +
 11001011
----------
101100000 ← This number requires 9 bits. An overflow condition has occurred.
```

The result of this addition has more than 8 bits. This number is too big to fit in a byte. Therefore, there is an overflow. We can verify this result by considering the decimal equivalent of the binary result. In this case $101100000_{(2} = 352_{(10}$. This value falls outside the range that can be represented with a byte.

19.6 REPRESENTING NEGATIVE NUMBERS IN A COMPUTER

So far we have considered nonnegative values. However, the operation of subtraction raises the possibility of having negative results. When representing negative numbers one of the bits is chosen as the *sign bit*. By convention, the leftmost bit (or most significant bit) is considered the sign bit (Fig. 19-1). This convention is illustrated below for a byte. A value of *0* in the sign bit indicates a positive number, whereas a value of *1* indicates a negative number. A similar convention is also followed for higher storage units including words and double words.

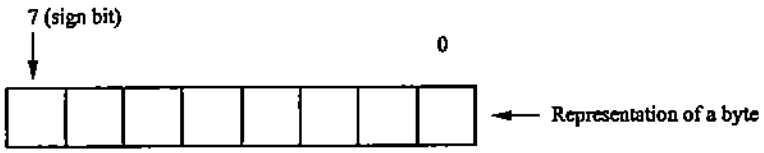

Fig. 19-1

Conventions

19.7 THE SIGN-MAGNITUDE

In this convention the leftmost bit is used exclusively to hold the sign bit. To represent negative numbers using n bits, the sign bit is set to 1 and the remaining $n - 1$ bits are used to represent the absolute value of the number (its magnitude). Positive numbers are represented in a similar way except that the sign bit is set to 0. The range of integer values that can be represented with this convention using n bits varies from $-2^{n-1} + 1$ to $2^{n-1} - 1$.

EXAMPLE 19.10. What is the sign-magnitude representation of decimal 75 and -75 if these values are represented in a byte?

Let us consider the representation of $+75$. The sign bit will be set to 0 since the number is positive. The remaining 7 bits will be used to represent the magnitude. The binary representation of $75_{(10} = 1001011_{(2}$. Therefore, the sign-magnitude representation of $+75$ is $01001011_{(2}$.

Since the magnitude of -75 is the same as that of $+75$, by changing the sign bit from 0 to 1, we will have that the representation of -75 is $11001011_{(2}$. The sign bit is set to 1 since the number is negative.

EXAMPLE 19.11. What is the sign magnitude representation of decimal 0? Assume you are working with bytes. The magnitude of 0 using 7 bits, in binary, is $0000000_{(2}$. However, notice that we can set the sign bit to either 0 or 1, therefore, representing 0 in two different forms: one as "positive" 0 and the other as "negative 0." This double representation of 0 has been one of the biggest drawbacks of this convention.

In sign-magnitude, arithmetic operations are carried out in binary following the usual convention. When adding numbers that have the same sign, we add their magnitudes and copy the common sign into the sign bit. If the signs are different, we first determine the number that has the largest magnitude and then subtract the other number from it. The sign of the result is the sign of the number that has the largest magnitude.

EXAMPLE 19.12. Using the sign-magnitude convention find the result of adding the decimal numbers 50 and 12 if these values are represented in a byte. The sign-magnitude representation of $50_{(10}$ is $00110010_{(2}$; the sign-magnitude representation of $12_{(10}$ is $00001100_{(2}$. Since both numbers are positive the result is positive. Adding their magnitudes we have that

$$\begin{array}{l} 0110010 + \\ 0001100 \\ \hline 0111110 \end{array}$$

The decimal equivalent of $0111110_{(2}$ is $62_{(10}$. The sign-magnitude representation of the result is $00111110_{(2}$.

EXAMPLE 19.13. What is the sign-magnitude representation of the addition of the decimal numbers -55 and 27 if both values are represented in a byte?

First, let us find the sign-magnitude of both numbers. The representation of $-55_{(10}$ is $10110111_{(2}$. The representation of $27_{(10}$ is $00011011_{(2}$. Since the number with the largest magnitude is -55 the result is negative. That is, the sign-bit of the result is 1. The magnitude is calculated by subtracting the binary representation of 27 from that of 55. That is,

$$\begin{array}{l} 0110111 - \\ 0011011 \\ \hline 0011100 \end{array}$$

Therefore, the sign-magnitude representation of the result of this addition is $10011100_{(2} = -28_{(10}$.

In the sign-magnitude convention an overflow occurs when the results fall outside the range from $-2^{n-1} + 1$ to $2^{n-1} - 1$.

19.8 ONE'S COMPLEMENT

This convention has the advantage of making the addition of two numbers with different signs the same as for two numbers with the same sign. Positive numbers are represented in the usual way. To represent a negative number in this convention, find the binary representation of the absolute value of the number and then complement all the bits including the sign bit. *To complement* the bits of a binary number, change all 0s to 1s and all 1s to 0s. Changing 0s to 1s and vice versa is equivalent to subtracting from 1 each of the individual bits of the number.

EXAMPLE 19.14. What is the 1's complement representation of decimal 15 and -15 if these values are represented in a word?

The binary representation of $15_{(10}$ is $0000000000001111_{(2}$. To represent -15, change 0s to 1s and 1s to 0s. The resulting number is $1111111111110000_{(2}$. Notice that the sign bit changed from 0 (positive) to 1 (negative).

EXAMPLE 19.15. What is the decimal equivalent of the 1's complement number $1111111111000011_{(2}$?

First notice that the number is negative since its sign bit is 1. The absolute value of this number is found by complementing all its bits. This results in $0000000000111100_{(2} = 60_{(10}$. Therefore, the original sequence of bits is the 1's complement representation of decimal -60.

Arithmetic operations in 1's complement are carried out in the usual way, without regard to the sign bits, which are treated as any other bit. However, when performing an addition, if there is a carry out of the leftmost position, this carry is added to the rightmost bit. This is known as an *end-around carry.*

An overflow condition will occur if the result of an operation falls outside the range $-2^{n-1} + 1$ to $+2^{n-1} - 1$, where n is the number of bits available to represent the number. This condition will be detected when operands of the same sign produce a result with opposite sign. An overflow will occur if either of the following conditions is satisfied: (1) There is a carry into the sign bit and no carry out of the sign bit. (2) There is no carry into the sign bit and there is a carry out of the sign bit.

EXAMPLE 19.16. Add the following 1's complement numbers 00110010 and 11100001? Is there an overflow?

Adding the binary numbers in the usual manner and remembering to add the end-around carry to the rightmost bit, if there is one, we have that:

```
 11          ← carries
 00110010  +
 11100001
 ---------
1 00010011  +
  ↘
    1       ← end-around carry
 ---------
 00010100   ← final result
```

There is no overflow since there was a carry into the sign bit and a carry out of the sign bit. To verify this result, let us find first the decimal equivalent of the given numbers. The decimal equivalent of 00110010 is $50_{(10}$, and the decimal equivalent of 11100001 is $-30_{(10}$. The decimal equivalent of the result 00010100 is $+20_{(10}$ as it should be.

One's complement also suffers from the "negative and positive" zero malady. However, because of the end-around carry, the "negative zero" behaves as the "positive" zero. When performing computations with a computer, the hardware needs to check for both representations of zero. This extra check for zero and the required sum whenever there is an end-around carry has made the one's complement convention an unpopular choice for hardware manufacturers.

19.9 TWO'S COMPLEMENT

The most popular convention for representing negative numbers is the *2's complement.* This variation of the 1's complement method corrects the problems of the negative zero and the end-around carry. The representation of positive numbers in this new convention is similar to that of the 1's complement. Negative numbers are represented using a two-step process: First, complement all bits of the representation of the absolute value of the number. Second, add 1 to the binary configuration of the number obtained in the previous step.

EXAMPLE 19.17. What is the 2's complement representation of $-45_{(10}$ if the value is represented in a word?

The binary representation of decimal 45 is $0000000000101101_{(2}$.

Complementing all bits of this number we obtain: 1111111111010010.

Adding 1 to this configuration of bits we have

$$\begin{array}{r} 1111111111010010 + \\ \underline{\qquad\qquad\quad 1} \\ 1111111111010011 \end{array}$$

Therefore, the 2's complement of decimal -45 is $1111111111010011_{(2}$.

There is an alternate algorithm to obtain the 2's complement representation of a integer value. Scan the binary representation of the number from right to left until the first 1-bit is encountered. Leave this bit and all the 0-bits to its right, if any, unchanged. Complement all bits to the left of this first 1-bit.

EXAMPLE 19.18. Find the 2's complement of 00010100 using the alternate algorithm.

Scanning the number from right to left we find that the first 1-bit occupies the bit position number 2. Leaving this 1-bit and all the 0-bits to its right unchanged we have

00010100

↑ First 1-bit. This 1-bit and all 0-bits to its right are left unchanged.

Complementing all bits to the left of the first 1-bit, we have that

11101100 ← Complement all bits to the left of the first 1-bit.

Therefore, the 2's complement representation of 00010100 is 11101100.

The range of numbers that can be represented in two's complement with n bits is -2^{n-1} to $2^{n-1} - 1$. Notice that the range of negative numbers is one unit larger than the range of positive values. Overflows are detected using the similar convention to that of the 1's complement notation.

To add 2's complement integers, follow the usual rules for adding binary numbers and ignore the carry out of the leftmost bit if there is one.

EXAMPLE 19.19. What is the result of adding 01100010 and 00110001 as 2's complement numbers if these values are represented in a byte? Does the result overflow?

Adding the two binary numbers in the usual way we obtain

$$\begin{array}{ll} 11 & \leftarrow \text{carries} \\ 01100010 & + \\ 00110001 & \\ \hline 10010011 & \leftarrow \text{The result is negative since its sign is 1.} \end{array}$$

Notice that an overflow has occurred because the operands are both positive but the result is negative. Another way to check that an overflow has occurred is by noticing that there is a carry into the leftmost bit but there is no carry out of it.

EXAMPLE 19.20. What is the result of adding 00100001 and 11101011 as 2's complement numbers if these values are represented in a byte? Does an overflow occur?

Adding the two binary numbers in the usual way we have that

```
 11    11  ← carries
 00100001 +
 11101011
----------
100001100 ← The end-around carry is ignored.
```

In this case there is an end-around carry which is ignored. There is no overflow since there is a carry into the sign bit and a carry out of the sign bit.

We can verify the result by noticing that $00100001_{(2} = 33_{(10}$ and that $11101011_{(2} = -21_{(10}$. The sum is $00001100_{(2} = 12_{(10}$ as it should be.

Subtraction of 2's complement numbers can be reduced to a simpler integer addition as illustrated in the following example.

EXAMPLE 19.21. What is the result of subtracting 00010110 from 00111100 if both numbers are represented in 2's complement notation?

Subtracting 00010110 from 00111100 is equivalent to adding the negative of 00010110 to 00111100. Since the 2's complement of 00010110 is 11101010 we have that

```
 00111100 +
 11101010
----------
100100110 ← Remember to ignore the carry out of the sign bit.
```

An alternative way to subtract 2's complement numbers is the following: Subtract one number from the other as if both were unsigned binary numbers. If a borrow is needed in the leftmost place, borrow as if there were another bit to the left. This is illustrated in the following example.

EXAMPLE 19.22. Subtract 00111001 from 00100011. Assume that both numbers are in 2's complement representation.

```
 00100011 −
 00111001
----------
        0 ← Write 0 since 1 minus 1 is 0.

 00100011 −
 00111001
----------
       10 ← Write 1 since 1 minus 0 is 1.

 00100011 −
 00111001
----------
      010 ← Write 0 since 0 minus 0 is 0.

    10
 00100011 −
 00111001
----------
     1010 ← We cannot subtract 1 from 0. Borrow "2" from higher unit to left. Write 1 since 2 − 1 = 1.

   1
 00100011 −
 00111001
----------
    01010 ← The 0 became 1 (the basis minus 1). Write 0 since 1 minus 1 is 0.
```

```
  10
  0
00100011 −
00111001
--------
  101010 ← 1 paid the "2" that was borrowed. The 1 became 0. We cannot subtract 1 from 0. Borrow
           "2" from higher unit to the left. Write 1 since 2 minus 1 is 1.

 1
00100011 −
00111001
--------
 1101010 ← The 0 to the left became 1 (the basis minus 1). Write 0 since 1 minus 1 is 0.

   1
I  00100011 −
   00111001
   --------
   11101010 ← The 0 to the left became 1 (the basis minus 1). Write 0 since 1 minus 1 is 0.
```

At this moment we can ask, "Who pays the last '2' that was borrowed?" The answer is simple; the 1 that is assumed to be at the leftmost position (shown here in bold italics).

The user can verify the answer using the method shown in Example 19-21. In fact, the 2's complements of 00111001 is 11000111. If we add this value to 00100011 we have that

```
    111  ← carries
00100011 +
11000111
--------
11101010
```

The result of this addition is identical to the one we obtained previously.

19.10 MULTIPLICATION AND DIVISION OF BINARY NUMBERS

Multiplication and division of binary numbers are carried out in the same way that both operations are carried out in decimal. A multiplication, like $A \times B$, is equivalent to the addition of A to itself B times. Therefore, we can multiply two numbers using successive additions. Likewise, a division such as A/B can be carried out by successive subtraction. Since the computers have special hardware to do additions and subtractions very rapidly, any multiplication or division is carried out as an addition or subtraction, respectively.

Solved Problems

19.1. What is the result of the following binary addition? Assume that no overflow can occur.

```
101011 +
011010
------
     1 ← 0+1=1.
```

```
101011 +
011010
------
    01  ← 1 + 1 = 0 and carry 1. Thinking in "decimal" notice that 2 = 1 × 2 + 0. Therefore, write 0
          and carry 1.

   1    ← carry
101011 +
011010
------
   101  ← 1 (the carry) plus 0 is 1. This 1 plus 0 is 1. Therefore, write 1.

101011 +
011010
------
  0101  ← 1 + 1 = 0 and carry 1. Thinking in "decimal" notice that 2 = 1 × 2 + 0. Therefore, write 0
          and carry 1.

 1
101011 +
011010
------
 00101  ← 1 (the carry) plus 0 is 1. This 1 plus 1 is 2. Therefore, write 0 and carry 1.

1
101011 +
011010
------
1000101 ← 1 (the carry) plus 1 is 2. This 2 plus 0 is 2. Therefore, write 0 and carry 1.
```

19.2. Subtract 100101 from 10010110 in binary.

```
     "10"
10010110 −
  100101
--------
       1  ← We cannot subtract 1 from 0. Borrow "2" from the higher unit to the left. Since 2−1=1, write 1.

      0
10010110 −
  100101
--------
      01  ← 1 "paid" the "2" that the 0 to its left borrowed. 1 decreased to 0. Since 0 − 0 = 0, write 0.

10010110 −
  100101
--------
     001  ← Since 1 − 1 = 0, write 0.

10010110 −
  100101
--------
    0001  ← Since 0 − 0 = 0, write 0.

10010110 −
  100101
--------
   10001  ← Since 1 − 0 = 1, write 1.
```

```
  "10"
 10010110 −
   100101
 ---------
   110001 ← We cannot subtract 1 from 0. Borrow "2" from the higher unit to the left. Since
            2 − 1 = 1, write 1.

  1
 10010110 −
   100101
 ---------
  1110001 ← The 0 became 1 (the basis minus 1). Write 1 since 1 − 0 = 0.

 0
 10010110 −
   100101
 ---------
 01110001 ← 1 "paid" the "2" that the 0 had borrowed. 1 decreased to 0. Since 0 − 0 = 0, write 0.
```

19.3. What is the result of subtracting 1 from 10000000 in binary?

```
      "10" ← carry
 10000000 −
        1
 ---------
        1 ← We cannot subtract 1 from 0. Borrow "2" from the higher unit to the left. Since
            2 − 1 = 1, write 1.

      1  ← carry
 10000000 −
        1
 ---------
       11 ← The next zero became 1 (the basis minus 1). Write 1 since 1 − 0 = 1.

     1    ← carry
 10000000 −
        1
 ---------
      111 ← The next zero also became 1 (the basis minus 1). Write 1 since 1 − 0 = 1.

    1     ← carry
 10000000 −
     1
 ---------
     1111 ← The next zero also became 1 (the basis minus 1). Write 1 since 1 − 0 = 1.

   1       ← carry
 10000000 −
        1
 ---------
    11111 ← The next zero also became 1 (the basis minus 1). Write 1 since 1 − 0 = 1.

  1         ← carry
 10000000 −
        1
 ---------
   111111 ← The next zero also became 1 (the basis minus 1). Write 1 since 1 − 0 = 1.
```

```
 1        ← carry
10000000 −
       1
---------
1111111  ← The next zero also became 1 (the basis minus 1). Write 1 since 1 − 0 = 1.

0        ← carry
10000000 −
       1
---------
01111111 ← 1 "paid" the "2" that the 0 had borrowed. 1 decreased to 0. Since 0 − 0 = 0, write 0.
```

19.4. What is the result of the following addition in binary? Assume that no overflow can occur.

```
10101111 +
11111010
---------
       1 ← Write 1 since 1 + 0 = 1.

10101111 +
11111010
---------
      01 ← Write 0 since 1 + 1 = 0 and carry 1. In decimal, 2 = 1 × 2 + 0. Thus, write 0 and carry 1.

    1    ← carry
10101111 +
11111010
---------
     001 ← 1 (the carry) plus 1 is 2. This 2 plus 0 is 2. Therefore, write 0 and carry 1.

   1     ← carry
10101111 +
11111010
---------
    1001 ← In "decimal," 1 (the carry) plus 1 is 2. This 2 plus 1 is 3. Since 3 = 1 × 2 + 1, write 1 and
           carry 1. In "binary," 1 (the carry) plus 1 is 10. Write 0 and carry 1. The 0 plus 1 is 1. Write
           1 and remember the previous carry.

  1      ← carry
10101111 +
11111010
---------
   01001 ← 1 (the carry) plus 0 is 1. This 1 plus 1 is 2. Therefore, write 0 and carry 1.

 1        ← carry
10101111 +
11111010
---------
  101001 ← In "decimal," 1 (the carry) plus 1 is 2. This 2 plus 1 is 3. Since 3 = 1 × 2 + 1, write 1 and
           carry 1. In "binary," 1 (the carry) plus 1 is 10. Write 0 and carry 1. The 0 plus 1 is 1. Write 1
           and remember the previous carry.

1        ← carry
10101111 +
11111010
---------
0101001  ← 1 (the carry) plus 0 is 1. This 1 plus 1 is 2. Therefore, write 0 and carry 1 since 2 = 1 × 2 + 0.
```

```
 1          ← carry
 10101111 +
 111111010
-----------
110101001 ← In "decimal," 1 (the carry) plus 1 is 2. This 2 plus 1 is 3. Since 3 = 1 × 2 + 1, write 1 and
            carry 1. In "binary," 1 (the carry) plus 1 is 10. Write 0 and carry 1. This 0 plus 1 is 1.
            Write 1 and remember the previous carry.
```

19.5. What is the result of adding the following hexadecimal numbers? Assume that no overflow can occur.

```
A1BF3 +
ECD1B
-----
    E ← B is decimal 11. 3 plus 11 is 14. Write E since E is decimal 14.

A1BF3 +
ECD1B
-----
   0E ← F is decimal 15. 15 plus 1 is 16. Since 16 = 1 × 16 + 0, write 0 and carry 1.

  1   ← carry
A1BF3 +
ECD1B
-----
  90E ← 1 (the carry) plus B (=11) is 12. This 12 plus D (=13) is 25. Since 25 = 1 × 16 + 9, write 9 and
        carry 1.

 1    ← carry
A1BF3 +
ECD1B
-----
 E90E ← 1 (the carry) plus 1 is 2. This 2 plus C (=12) is 14. Write E since E is decimal 14.

A1BF3 +
ECD1B
-----
18E90E ← A (=10) plus E (=14) is 24. Since 24 = 1 × 16 + 8, write 8 and carry 1.
```

19.6. What is the result of the following hexadecimal subtraction?

```
F3005 −
 1AD
-----
    8 ← We cannot subtract D (=13) from 5. Borrow "16," from higher unit to the left. 16 plus 5 is 21. This
        21 minus D (=13) is 8. Write 8.

  F
F3005 −
 1AD
-----
  58 ← The 0 became an F (the basis minus 1). F (=15) minus A (=10) is 5. Therefore, write 5.

 F
F3005 −
 1AD
-----
 E58 ← The next 0 became also an F (the basis minus 1). F (=15) minus 1 is 14.
       Write E since E is decimal 14.
```

```
 2
F300 5 −
  1AD
-------
2E5 8 ← The 3 "paid" the "16" that was borrowed by the rightmost 5. 3 decreases to 2. Write 2 since
        2 − 0 = 2.

F300 5 −
  1AD
-------
F2E5 8 ← Write F since F (=15) minus 0 is 15. Hexadecimal F is decimal 15.
```

19.7. What is the result of the following hexadecimal addition? Assume that no overflow occur.

```
ABC +
DAB
---
  7 ← C (=12) plus B (=11) is 23. Since 23 = 1 × 16 + 7, write 7 and carry 1.

 1  ← carry
ABC +
DAB
---
 67 ← 1 (the carry) plus B (=11) is 12. This 12 plus A (=10) is 22. Since 22 = 1 × 16 + 6, write 6 and
      carry 1.

 1   ← carry
ABC +
DAB
---
18 6 7 ← 1 (the carry) plus A (=10) is 11. This 11 plus D (=13) is 24. Since 24 = 1 × 16 + 8, write 8 and
         carry 1.
```

19.8. What is the result of the following hexadecimal subtraction?

```
FFEA −
EEFF
----
   B ← Since F (=15) cannot be subtracted from A (=10), borrow "16" from next higher unit. A (=10) plus
       16 is 26. Since 26 minus F (=15) is 11, write B.

  D
FFEA −
EEFF
----
  EB ← The E "paid" the "16" borrowed by the A. The E (=14) decreases to D (=13). Since F (=15)
       cannot be subtracted from D borrow "16" from the next higher unit. D (=13) plus 16 = 29. Since
       29 minus F (=15) is 14, write E.

 E
FFEA −
EEFF
----
 0EB ← The F "paid" the "16" borrowed by the D. The F (=15) decreases to E (=14). Since E (=14) minus
       E (=14) is 0, write 0.
```

```
FFEA −
EEFF
----
10EB ← F (=15) minus E (=14) is 1. Therefore, write 1.
```

19.9. What is the sign-magnitude representation of decimal 77? Use a byte to represent the number.

The sign bit is 0 since the number is positive. The binary representation of 77 is $1001101_{(2}$. Therefore, the sign magnitude is 01001101.

19.10. What is the decimal equivalent of the sign-magnitude binary sequence 10011101?

The number is negative since its sign bit is 1. The magnitude is given by the decimal equivalent of the remaining bits. In this case, $0011101_{(2} = 29_{(10}$. Therefore, the decimal equivalent of the original binary sequence is decimal -29.

19.11. What is the sign magnitude representation of -45? Use a byte to represent the number.

Since the number is negative, the sign bit is 1. The magnitude is given by the binary representation of 45. Since $45_{(10} = 101101_{(2}$, the sign-magnitude of -45 is $1101101_{(2}$.

19.12. What is the decimal equivalent of the sign-magnitude representation given by the binary sequence 01011101?

The number is positive since the sign bit is 0. The magnitude is given by the decimal equivalent of the remaining bits. In this case, $1011101_{(2} = 93_{(10}$. Therefore, the decimal equivalent of the original binary sequence is $+93$.

19.13. What is the result of adding the sign-magnitude binary numbers 00101011 and 00010001?

Since both numbers are positive, the result is positive. Therefore, the sign bit is 0. To find the magnitude of the result, add the magnitude of the given numbers. That is, add

```
0101011 +
0010001
-------
      0 ← Write 0 and carry 1.

     1  ← carry
0101011 +
0010001
-------
     00 ← Write 0 and carry 1.

    1   ← carry
0101011 +
0010001
-------
    100 ← Write 1 since 1 + 0 = 1.
```

```
0101011 +
0010001
-------
   1100 ← Write 1 since 0 plus 1 is 1.

0101011 +
0010001
-------
  11100 ← Write 1 since 0 plus 1 is 1.

0101011 +
0010001
-------
 111100 ← Write 1 since 0 plus 1 is 1.

0101011 +
0010001
-------
0111100 ← Write 0 since 0 plus 0 is 0.
```

We can verify the correctness of the result by noting that the decimal equivalents of the operands are $00101011_{(2} = +43_{(10}$ and $00010001_{(2} = +17_{(10}$. Therefore, the result is $00111100_{(2} = +60_{(10}$ as it should be.

19.14. Add the sign-magnitude binary numbers 11001000 and 00110000.

Since the numbers have opposite signs, the sign of the addition is the sign of the number with the largest magnitude. Therefore, it is necessary to determine which of the two numbers has the largest magnitude.

The number 11001000 has a negative sign. Its magnitude is $1001000_{(2} = 72_{(10}$.

The number 00110000 has a positive sign. Its magnitude is $0110000_{(2} = 48_{(10}$.

Since the number with the largest magnitude is -72, the sign of the addition will be negative. The magnitude is calculated by subtracting the absolute value of the smaller number from the absolute value of the largest. That is,

```
1001000 −
0110000
-------
      0 ← Write 0 since 0 minus 0 is 0.

1001000 −
0110000
-------
     00 ← Write 0 since 0 minus 0 is 0.

1001000 −
0110000
-------
    000 ← Write 0 since 0 minus 0 is 0.

1001000 −
0110000
-------
   1000 ← Write 1 since 1 minus 0 is 1.

 "10"
1001000 −
0110000
-------
  11000 ← 1 cannot be subtracted from 0. Borrow "2" from higher unit to the left. Write 1 since 2 − 1 = 1.
```

```
 1
1001000 −
0110000
-------
 011000 ← The next 0 became a 1 (the basis minus 1). Write 1 since 1 minus 0 is 1.

 0
1001000 −
0110000
-------
0011000 ← 1 "paid" the 2 that was borrowed before. It decreases to 0. Write 0 since 0 minus 0 is 0.
```

We can verify the correctness of this result by noticing that $0011000_{(2} = 24_{(10}$. Thus, the result of this addition, expressed in sign-magnitude, is $10011000_{(2}$.

19.15. What is the 1's complement representation of decimal −27? Use a byte to represent the number.

Since the number is negative, we need to find the binary representation of +27. In this case we obtain that $27_{(10} = 00011011_{(2}$. To find the 1's complement of this number change all 0s to 1s and vice versa. The result is $11100100_{(2}$.

19.16. What is the decimal equivalent of the 1's complement sequence 10101010?

The number is negative since its sign bit is 1. Complementing all of its bits we obtain 01010101. The decimal equivalent of this binary sequence is the absolute value of the number. In this case, $01010101_{(2} = 85_{(10}$. Therefore, the original 1's complement binary sequence is equivalent to $-85_{(10}$.

19.17. Add the binary 1's complement numbers shown below. Is there an overflow?

```
11111100 +
00000110
--------
       0 ← Write 0 since 0+0=0.

11111100 +
00000110
--------
      10 ← Write 1 since 1+0=1.

11111100 +
00000110
--------
     010 ← In "decimal" we have that 2=1×2+0. Thus, write 0 and carry 1.

   1     ← carry
11111100 +
00000110
--------
    0010 ← Working in "decimal," 1 (the carry) plus 1 is 2. This 2 plus 0 is 2. Since 2=1×2+0, write
           0 and carry 1.
```

```
   1      ← carry
11111100 +
00000110
--------
   00010 ← Working in "decimal" 1 (the carry) plus 1 is 2. This 2 plus 0 is 2. Since 2 = 1 × 2 + 0, write 0
           and carry 1.

  1       ← carry
11111100 +
00000110
--------
  000010 ← Working in "decimal" 1 (the carry) plus 1 is 2. This 2 plus 0 is 2. Since 2 = 1 × 2 + 0, write 0
           and carry 1.

 1        ← carry
11111100 +
00000110
--------
 0000010 ← Working in "decimal" 1 (the carry) plus 1 is 2. This 2 plus 0 is 2. Since 2 = 1 × 2 + 0, write 0
           and carry 1.

 1         ← carry
 11111100 +
 00000110
--------
1 00000100 + ← Write 0 and add the end-around carry to the rightmost bit.
         1 ← end-around carry.
--------
 00000011 ← Final result.
```

We can verify the correctness of this operation by noticing that $11111100_{(2} = -3_{(10}$ and $00000110_{(2} = +6_{(10}$. The result is $00000011_{(2} = +3_{(10}$ as it should be.

There is no overflow since there is a carry into the sign bit and a carry out of the sign bit.

19.18. What is the decimal equivalent of the 2's complement binary representation 10100101?

Since the sign bit of the given number is 1, the number is negative. Following the 2's complement convention to determine the absolute value of the number, all we need to do is complement all its bits and then add 1 to the resulting number. Complementing all bits produces 01011010. Adding 1 to this number we have

```
01011010 +
       1
--------
01011011
```

Since $01011011_{(2} = 91_{(10}$. The decimal equivalent of the given number is -91.

19.19. What is the 2's complement representation of decimal 87 and -87? Assume that the value is represented in a byte.

The binary equivalent of $87_{(10}$ is $01010111_{(2}$. Since the number is positive, this representation is also its 2's complement representation.

The 2's complement representation of $-87_{(10}$ is found by complementing all bits of the binary equivalent of $87_{(10}$ and then adding 1 to it. This two-step process is shown below.

Complementing all bits of $01010111_{(2}$ produces $10101000_{(2}$.

Adding 1 to this number we obtain

```
10101000 +
       1
--------
10101001
```

Therefore the 2's complement representation of -87 is 10101001.

This result can be verified by noticing that $87_{(10}$ plus $-87_{(10} = 0$. A similar result is obtained when we add their binary equivalents as indicated below.

```
-87(2    10101001 +
+87(2    01010111
         --------
       1 00000000 ← Remember that in 2's complement the carry out of the sign bit is ignored.
       ↑  ignore this carry.
```

19.20. What is the result of adding the following 2's complement binary numbers? Does the result overflow?

```
00100111 +
11011011
--------
       0

     1
00100111 +
11011011
--------
      10

    1
00100111 +
11011011
--------
     010

   1
00100111 +
11011011
--------
    0010

  1
00100111 +
11011011
--------
   00010

 1
00100111 +
11011011
--------
  000010

1
00100111 +
11011011
--------
 0000010
```

```
  1
  00100111 +
  11011011
 ----------
 100000010 ← Final result.
 ↑ Ignore this carry.
```

There is no overflow since there is a carry into the sign bit and a carry out of it.

That this result is correct can be verified by adding the decimal equivalent of the numbers and comparing the sum against the final result that we just obtained.

The decimal equivalent of $00100111_{(2} = 39_{(10}$.

The other operand, 11011011, is negative according to its sign bit. Therefore, to calculate its decimal equivalent we need to find its 2's complement. Changing all 1s to 0s and vice versa, we obtain 00100100. Adding 1 to this number we obtain

```
  00100100 +
         1
 ----------
  00100101
```

In consequence, the decimal equivalent of $11011011_{(2}$ is $-37_{(10}$. Since $39_{(10} - 37_{(10} = 2_{(10}$ we confirm that our final result is correct because $00000010_{(2} = 2_{(10}$.

Supplementary Problems

The answers to this set of problems are at the end of this chapter.

19.21. Perform the following calculations in the binary number system. Assume that no overflow can occur.

(*a*) 10110111 + 11011001

(*b*) 11001100 + 10010111

(*c*) 10110101 + 10110101

(*d*) 01110011 − 1010010

(*e*) 01110001 − 01000101

(*f*) 01111110 − 00110000

19.22. Perform the following calculations in the given number system. Assume that no overflow can occur.

(*a*) A12B + 3C54

(*b*) 48A3 + C759

(*c*) ED21 + 25DF

(*d*) F4CD − 21DE

(*e*) BE2A− 9CFE

(*f*) FE25− 3ACD

19.23. Represent the following decimal numbers in sign-magnitude. Use a byte to represent the numbers.

(*a*) 93 (*b*) −26 (*c*) 127 (*d*) −4

19.24. Represent the following decimal numbers in 1's complement representation. Use a word to represent the numbers.

(*a*) −133 (*b*) 157 (*c*) −168 (*d*) −555

19.25. Represent the following decimal numbers in 2's complement representation. Use a byte to represent the numbers.

(*a*) −106 (*b*) −95 (*c*) −87 (*d*) −68 (*e*) −145

19.26. Perform the following calculations. Assume that the numbers are represented in sign-magnitude. If an overflow occurs explain why. Use a byte to represent the numbers.

(*a*) 10110101 + 01111011

(*b*) 01011010 + 01010101

(*c*) 11010101 + 00110101

(*d*) 10110011 + 11011100

(*e*) 00111100+ 01010101

19.27. Perform the following calculations. Assume that the numbers are represented in 1's complement. If an overflow occurs explain why. Use a byte to represent the numbers.

(*a*) 01100100 + 00011100

(*b*) 10011011 + 11100011

(*c*) 00100001 + 11101010

(*d*) 11011110 + 11101010

(*e*) 10110101 + 01111011

19.28. Perform the following calculations. Assume that the numbers are represented in 2's complement. If an overflow occurs explain why. Use a byte to represent the numbers.

(*a*) 01011100 + 11010110

(*b*) 10011010 + 11000110

(*c*) 01010101 + 00110111

(*d*) 01111111 + 11001110

(*e*) 00101000 + 01001101

Answers to Supplementary Problems

19.21. (*a*) 110010000, (*b*) 101100011, (*c*) 101101010, (*d*) 00100001, (*e*) 00101100, (*f*) 01001110

19.22. (*a*) DD7F, (*b*) 10FFFC, (*c*) 11300, (*d*) D2EF, (*e*) 212C, (*f*) C358

19.23. (*a*) 01011101, (*b*) 10011010, (*c*) 01111111, (*d*) 10000100

19.24. (*a*) 1111111101111010, (*b*) 0000000010011101, (*c*) 1111111101010111, (*d*) 1111110111010100

19.25. (*a*) 10010110, (*b*) 10100001, (*c*) 10101001, (*d*) 10111100, (*e*) It cannot be represented since this number is outside the range of values that can be represented with a byte.

19.26. (*a*) 01000110. There is no overflow.

(*b*) Overflow! The sum is 0101111 (using 7 bits). There is a carry out of the 7th bit. The number is too big to fit in 7 digits.

(*c*) 10100000. There is no overflow.

(*d*) Overflow! The sum is 0001111 (using 7 bits). There is a carry out of the 7th bit. The number is too big to fit in 7 digits.

(*e*) Overflow! The sum is 0010001 (using 7 bits). There is a carry out of the 7th bit. The number is too big to fit in 7 digits.

19.27. (*a*) Overflow! The sum, 10000000, is negative whereas both operands are positive. Notice that there is a carry into the sign bit but no carry out of the sign bit.

(*b*) Overflow! The sum, 01111111, is positive whereas both operands are negative. Notice that there is no carry into the sign bit but there is a carry out of the sign bit.

(*c*) The sum is 00001100. No overflow occurs.

(*d*) The sum is 11001001. No overflow occurs.

(*e*) The sum is 00110001. No overflow occurs.

19.28. (*a*) The result is 00110010. No overflow occurs.

(*b*) Overflow! The result, 01100000 (using 8 bits), is positive whereas both operands are negative. Notice that there is carry out of the sign bit but no carry into it.

(*c*) Overflow! The result is 10001100 (using 8 bits), is negative whereas both operands are positive. Notice that there is a carry into the sign bit but no carry out of it.

(*d*) The result is 01001101. No overflow occurs.

(*e*) The result is 01110101. No overflow occurs.