SECTION 3.6 Integers and Algorithms

2. To convert from decimal to binary, we successively divide by 2. We write down the remainders so obtained from right to left; that is the binary representation of the given number.

a) Since 321/2 is 160 with a remainder of 1, the rightmost digit is 1. Then since 160/2 is 80 with a remainder of 0, the second digit from the right is 0. We continue in this manner, obtaining successive quotients of 40, 20, 10, 5, 2, 1, and 0, and remainders of 0, 0, 0, 0, 1, 0, and 1. Putting all these remainders in order from right to left we obtain $(1\ 0100\ 0001)_2$ as the binary representation. We could, as a check, expand this binary numeral: $2^0 + 2^6 + 2^8 = 1 + 64 + 256 = 321$.

b) We could carry out the same process as in part **(a)**. Alternatively, we might notice that $1023 = 1024 - 1 = 2^{10} - 1$. Therefore the binary representation is 1 less than $(100\ 0000\ 0000)_2$, which is clearly $(11\ 1111\ 1111)_2$.

c) If we carry out the divisions by 2, the quotients are 50316, 25158, 12579, 6289, 3144, 1572, 786, 393, 196, 98, 49, 24, 12, 6, 3, 1, and 0, with remainders of 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, and 1. Putting the remainders in order from right to left we have $(1\ 1000\ 1001\ 0001\ 1000)_2$.

4. **a)** $1 + 2 + 8 + 16 = 27$ **b)** $1 + 4 + 16 + 32 + 128 + 512 = 693$

c) $2 + 4 + 8 + 16 + 32 + 128 + 256 + 512 = 958$

d) $1 + 2 + 4 + 8 + 16 + 1024 + 2048 + 4096 + 8192 + 16384 = 31775$

6. Following Example 6, we simply write the binary equivalents of each digit. Since $(A)_{16} = (1010)_2$, $(B)_{16} = (1011)_2$, $(C)_{16} = (1100)_2$, $(D)_{16} = (1101)_2$, $(E)_{16} = (1110)_2$, and $(F)_{16} = (1111)_2$, we have $(BADFACED)_{16} = (10111010110111111010110011101101)_2$. Following the convention shown in Exercise 3 of grouping binary digits by fours, we can write this in a more readable form as 1011 1010 1101 1111 1010 1100 1110 1101.

8. We follow the method stated in Example 6.

a) 1111 0111 becomes F7 **b)** 1010 1010 1010 becomes AAA **c)** 111 0111 0111 0111 becomes 7777

10. Following Example 6, we simply write the hexadecimal equivalents of each group of four binary digits. Note that we group from the right, so the left-most group, which is just 1, becomes 0001. Thus we have $(0001\ 1000\ 0110\ 0011)_2 = (1863)_{16}$.

12. Let $(\ldots h_1 h_1 h_0)_{16}$ be the hexadecimal expansion of a positive integer. The value of that integer is, therefore, $h_0 + h_1 \cdot 16 + h_2 \cdot 16^2 + \cdots = h_0 + h_1 \cdot 2^4 + h_2 \cdot 2^8 + \cdots$. If we replace each hexadecimal digit h_i by its binary expansion $(b_{i3}b_{i2}b_{i1}b_{i0})_2$, then $h_i = b_{i0} + 2b_{i1} + 4b_{i2} + 8b_{i3}$. Therefore the value of the entire number is $b_{00} + 2b_{01} + 4b_{02} + 8b_{03} + (b_{10} + 2b_{11} + 4b_{12} + 8b_{13}) \cdot 2^4 + (b_{20} + 2b_{21} + 4b_{22} + 8b_{23}) \cdot 2^8 + \cdots = b_{00} + 2b_{01} + 4b_{02} + 8b_{03} + 2^4 b_{10} + 2^5 b_{11} + 2^6 b_{12} + 2^7 b_{13} + 2^8 b_{20} + 2^9 b_{21} + 2^{10} b_{22} + 2^{11} b_{23} + \cdots$, which is the value of the binary expansion $(\ldots b_{23}b_{22}b_{21}b_{20}b_{13}b_{12}b_{11}b_{10}b_{03}b_{02}b_{01}b_{00})_2$.

14. This is exactly the same as what we can do with hexadecimal expansion, replacing groups of four with groups of three. Specifically, convert each octal digit into its 3-digit binary equivalent. For example, $(306)_8 = (011\ 000\ 110)_2$.

16. Since we have procedures for converting both octal and hexadecimal to and from binary (Example 6 and Exercises 13–15), to convert from hexadecimal to octal, we first convert from hexadecimal to binary and then convert from binary to octal.

18. We work through binary in each case; see Exercises 16 and 17. Thus

$$(12345670)_8 = (001\ 010\ 011\ 100\ 101\ 110\ 111\ 000)_2 = (0010\ 1001\ 1100\ 1011\ 1011\ 1000)_2 = (29CBB8)_{16}$$

and

$$\begin{aligned}(ABB093BABBA)_{16} &= (1010\ 1011\ 1011\ 0000\ 1001\ 0011\ 1011\ 1010\ 1011\ 1011\ 1010)_2 \\ &= (010\ 101\ 011\ 101\ 100\ 001\ 001\ 001\ 110\ 111\ 010\ 101\ 110\ 111\ 010)_2 \\ &= (25354111672567 2)_8.\end{aligned}$$

20. In effect, this algorithm computes $11 \bmod 645$, $11^2 \bmod 645$, $11^4 \bmod 645$, $11^8 \bmod 645$, $11^{16} \bmod 645$, ..., and then multiplies (modulo 645) the required values. Since $644 = (1010000100)_2$, we need to multiply together $11^4 \bmod 645$, $11^{128} \bmod 645$, and $11^{512} \bmod 645$, reducing modulo 645 at each step. We compute by repeatedly squaring: $11^2 \bmod 645 = 121$, $11^4 \bmod 645 = 121^2 \bmod 645 = 14641 \bmod 645 = 451$, $11^8 \bmod 645 = 451^2 \bmod 645 = 203401 \bmod 645 = 226$, $11^{16} \bmod 645 = 226^2 \bmod 645 = 51076 \bmod 645 = 121$. At this point we notice that 121 appeared earlier in our calculation, so we have $11^{32} \bmod 645 = 121^2 \bmod 645 = 451$, $11^{64} \bmod 645 = 451^2 \bmod 645 = 226$, $11^{128} \bmod 645 = 226^2 \bmod 645 = 121$, $11^{256} \bmod 645 = 451$, and $11^{512} \bmod 645 = 226$. Thus our final answer will be the product of 451, 121, and 226, reduced modulo 645. We compute these one at a time: $451 \cdot 121 \bmod 645 = 54571 \bmod 645 = 391$, and $391 \cdot 226 \bmod 645 = 88366 \bmod 645 = 1$. So $11^{644} \bmod 645 = 1$. A computer algebra system will verify this; use the command "`1 &^ 644 mod 645;`" in *Maple*, for example. The ampersand here tells *Maple* to use modular exponentiation, rather than first computing the integer 11^{644}, which has over 600 digits, although it could certainly handle this if asked. The point is that modular exponentiation is much faster and avoids having to deal with such large numbers.

22. In effect this algorithm computes powers $123 \bmod 101$, $123^2 \bmod 101$, $123^4 \bmod 101$, $123^8 \bmod 101$, $123^{16} \bmod 101$, ..., and then multiplies (modulo 101) the required values. Since $1001 = (1111101001)_2$, we need to multiply together $123 \bmod 101$, $123^8 \bmod 101$, $123^{32} \bmod 101$, $123^{64} \bmod 101$, $123^{128} \bmod 101$, $123^{256} \bmod 101$, and $123^{512} \bmod 101$, reducing modulo 101 at each step. We compute by repeatedly squaring: $123 \bmod 101 = 22$, $123^2 \bmod 101 = 22^2 \bmod 101 = 484 \bmod 101 = 80$, $123^4 \bmod 101 = 80^2 \bmod 101 = 6400 \bmod 101 = 37$, $123^8 \bmod 101 = 37^2 \bmod 101 = 1369 \bmod 101 = 56$, $123^{16} \bmod 101 = 56^2 \bmod 101 = 3136 \bmod 101 = 5$, $123^{32} \bmod 101 = 5^2 \bmod 101 = 25$, $123^{64} \bmod 101 = 25^2 \bmod 101 = 625 \bmod 101 = 19$, $123^{128} \bmod 101 = 19^2 \bmod 101 = 361 \bmod 101 = 58$, $123^{256} \bmod 101 = 58^2 \bmod 101 = 3364 \bmod 101 = 31$, and $123^{512} \bmod 101 = 31^2 \bmod 101 = 961 \bmod 101 = 52$. Thus our final answer will be the product of 22, 56, 25, 19, 58, 31, and 52. We compute these one at a time modulo 101: $22 \cdot 56$ is 20, $20 \cdot 25$ is 96, $96 \cdot 19$ is 6, $6 \cdot 58$ is 45, $45 \cdot 31$ is 82, and finally $82 \cdot 52$ is 22. So $123^{1001} \bmod 101 = 22$.

24. To apply the Euclidean algorithm, we divide the larger number by the smaller, replace the larger by the smaller and the smaller by the remainder of this division, and repeat this process until the remainder is 0. At that point, the smaller number is the greatest common divisor.

a) $\gcd(1,5) = \gcd(1,0) = 1$ **b)** $\gcd(100,101) = \gcd(100,1) = \gcd(1,0) = 1$

c) $\gcd(123,277) = \gcd(123,31) = \gcd(31,30) = \gcd(30,1) = \gcd(1,0) = 1$

d) $\gcd(1529,14039) = \gcd(1529,278) = \gcd(278,139) = \gcd(139,0) = 139$

e) $\gcd(1529,14038) = \gcd(1529,277) = \gcd(277,144) = \gcd(144,133) = \gcd(133,11) = \gcd(11,1) = \gcd(1,0) = 1$

f) $\gcd(11111,111111) = \gcd(11111,1) = \gcd(1,0) = 1$

26. We need to divide successively by 55, 34, 21, 13, 8, 5, 3, 2, and 1, so 9 divisions are required.

28. a) $5 = 9 - 3 - 1$ b) $13 = 9 + 3 + 1$ c) $37 = 27 + 9 + 1$ d) $79 = 81 - 3 + 1$

30. The key fact here is that $10 \equiv -1 \pmod{11}$, and so $10^k \equiv (-1)^k \pmod{11}$. Thus 10^k is congruent to 1 if k is even and to -1 if k is odd. Let the decimal expansion of the integer a be given by $(a_{n-1}a_{n-2}\ldots a_3a_2a_1a_0)_{10}$. Thus $a = 10^{n-1}a_{n-1} + 10^{n-2}a_{n-2} + \cdots + 10a_1 + a_0$. Since $10^k \equiv (-1)^k \pmod{11}$, we have $a \equiv \pm a_{n-1} \mp a_{n-2} + \cdots - a_3 + a_2 - a_1 + a_0 \pmod{11}$, where signs alternate and depend on the parity of n. Therefore $a \equiv 0 \pmod{11}$ if and only if $(a_0 + a_2 + a_4 + \cdots) - (a_1 + a_3 + a_5 + \cdots)$, which we obtain by collecting the odd and even indexed terms, is congruent to 0 (mod 11). Since being divisible by 11 is the same as being congruent to 0 (mod 11), we have proved that a positive integer is divisible by 11 if and only if the sum of its decimal digits in even-numbered positions minus the sum of its decimal digits in odd-numbered positions is divisible by 11.

32. a) Since the binary representation of 22 is 10110, the six bit one's complement representation is 010110.

b) Since the binary representation of 31 is 11111, the six bit one's complement representation is 011111.

c) Since the binary representation of 7 is 111, we complement 000111 to obtain 111000 as the one's complement representation of -7.

d) Since the binary representation of 19 is 10011, we complement 010011 to obtain 101100 as the one's complement representation of -19.

34. Every 1 is changed to a 0, and every 0 is changed to a 1.

36. We just combine the two ideas in Exercises 34 and 35: to form $a - b$, we compute $a + (-b)$, using Exercise 34 to find $-b$ and Exercise 35 to find the sum.

38. Following the definition, we find the two's complement expansion of a positive number simply by representing it in binary, using six bits; and we find the two's complement expansion of a negative number $-x$ by representing $2^5 - x$ in binary using five bits and preceding it with a 1.

a) Since 22 is positive, and its binary expansion is 10110, the answer is 010110.

b) Since 31 is positive, and its binary expansion is 11111, the answer is 011111.

c) Since -7 is negative, we first find the 5-bit binary expansion of $2^5 - 7 = 25$, namely 11001, and precede it by a 1, obtaining 111001.

d) Since -19 is negative, we first find the 5-bit binary expansion of $2^5 - 19 = 13$, namely 01101, and precede it by a 1, obtaining 101101.

40. We can experiment a bit to find a convenient algorithm. We saw in Exercise 38 that the expansion of -7 is 111001, while of course the expansion of 7 is 000111. Apparently to find the expansion of $-m$ from that of m we complement each bit and then add 1, working in base 2. Similarly, the expansion of -8 is 111000, whereas the expansion of 8 is 001000; again $110111 + 1 = 111000$. At the extremes (using six bits) we have 1 represented by 000001, so -1 is represented by $111110 + 1 = 111111$; and 31 is represented by 011111, so -31 is represented by $100000 + 1 = 100001$.

42. We just combine the two ideas in Exercises 40 and 41: to form $a - b$, we compute $a + (-b)$, using Exercise 40 to find $-b$ and Exercise 41 to find the sum.

44. If the number is positive (i.e., the left-most bit is 0), then the expansions are the same. If the number is negative (i.e., the left-most bit is 1), then we take the one's complement representation and add 1, working in base 2. For example, the one's complement representation of -19 using six bits is, from Exercise 32, 101100. Adding 1 we obtain 101101, which is the two's complement representation of -19 using six bits, from Exercise 38.

46. We obtain these expansions from the top down. For example in part **(e)** we compute that $7! > 1000$ but $6! \leq 1000$, so the highest factorial appearing is $6! = 720$. We use the division algorithm to find the quotient and remainder when 1000 is divided by 720, namely 1 and 280, respectively. Therefore the expansion begins $1 \cdot 6!$ and continues with the expansion of 280, which we find in the same manner.

a) $2 = 2!$ **b)** $7 = 3! + 1!$ **c)** $19 = 3 \cdot 3! + 1!$ **d)** $87 = 3 \cdot 4! + 2 \cdot 3! + 2! + 1!$

e) $1000 = 6! + 2 \cdot 5! + 4! + 2 \cdot 3! + 2 \cdot 2!$ **f)** $1000000 = 2 \cdot 9! + 6 \cdot 8! + 6 \cdot 7! + 2 \cdot 6! + 5 \cdot 5! + 4! + 2 \cdot 3! + 2 \cdot 2!$

48. The algorithm is essentially the same as the usual grade-school algorithm for adding. We add from right to left, one column at a time, carrying to the next column if necessary. A carry out of the column representing $i!$ is needed whenever the sum obtained for that column is greater than i, in which case we subtract $i + 1$ from that digit and carry 1 into the next column (since $(i + 1)! = (i + 1) \cdot i!$).

50. The partial products are 11100 and 1110000, namely 1110 shifted one place and three places to the left. We add these two numbers, obtaining 10001100.

52. Subtraction is really just like addition, so the number of bit operations should be comparable, namely $O(n)$. More specifically, if we analyze the algorithm for Exercise 51, we see that the loop is executed n times, and only a few operations are performed during each pass.

54. In the worst case, each bit of a has to be compared to each bit of b, so $O(n)$ comparisons are needed. An exact analysis of the procedure given in the solution to Exercise 53 shows that $n + 1$ comparisons of bits are needed in the worst case, assuming that the logical "and" condition in the **while** loop is evaluated efficiently from left to right (so that a_0 is not compared to b_0 there).

56. A multiplication modulo m consists of multiplying two integers, each at most $\log m$ bits long (since they are less than m), followed by a division by m, which is also $\log m$ bits long. Thus this takes $(\log m)^2$ bit operations by Example 10 and the analysis of Algorithm 4 mentioned in the text. This is what goes on inside the loop of Algorithm 5. The loop is iterated $\log n$ times. Therefore the total number of bit operations is $O((\log m)^2 \log n)$.