3e2.1

2.1
(a) $1101011_2 = 6B_{16}$ (b) $174003_8 = 1111100000000011_2$
(c) $10110111_2 = B7_{16}$ (d) $67.24_8 = 110111.0101_2$
(e) $10100.1101_2 = 14.D_{16}$ (f) $F3A5_{16} = 1111001110100101_2$
(g) $11011001_2 = 331_8$ (h) $AB3D_{16} = 1010101100111101_2$
(i) $101111.0111_2 = 57.34_8$ (j) $15C.38_{16} = 101011100.00111_2$

For exclusive use of adopters of the book *Digital Design Principles and Practices*, Fourth Edition, by John F. Wakerly, ISBN 0-13-186389-4.

4e2.2 2.2 (a) $1234_8 = 1010011100_2 = 29C_{16}$

(b) $174637_8 = 1111100110011111_2 = F99F_{16}$

(c) $365517_8 = 11110101101001111_2 = 1EB4F_{16}$

(d) $2535321_8 = 10101011101011010001_2 = ABAD1_{16}$

(e) $7436.11_8 = 111100011110.001001_2 = F1E.24_{16}$

(f) $45316.7414_8 = 100101011001110.111100001100_2 = 4ACE.F0C_{16}$

3e2.3

2.3 (a) $1023_{16} = 1000000100011_2 = 10043_8$

(b) $7E6A_{16} = 111111001101010_2 = 77152_8$

(c) $ABCD_{16} = 1010101111001101_2 = 125715_8$

(d) $C350_{16} = 1100001101010000_2 = 141520_8$

(e) $9E36.7A_{16} = 1001111000110110.0111101_2 = 117066.364_8$

(f) $DEAD.BEEF_{16} = 1101111010101101.1011111011101111_2 = 157255.575674_8$

4e2.4 2.4 $32107654321_8 = 11010001\ 00011111\ 01011000\ 11010001_2$
$= (011\ 010\ 001)\ (000\ 011\ 111)\ (001\ 011\ 000)\ (011\ 010\ 001)_2 = (321)\ (037)\ (130)\ (321)_8$

3e2.5 2.5 (a) $1101011_2 = 107_{10}$ (b) $174003_8 = 63491_{10}$
(c) $10110111_2 = 183_{10}$ (d) $67.24_8 = 55.3125_{10}$
(e) $10100.1101_2 = 20.8125_{10}$ (f) $F3A5_{16} = 62373_{10}$
(g) $12010_3 = 138_{10}$ (h) $AB3D_{16} = 43837_{10}$
(i) $7156_8 = 3694_{10}$ (j) $15C.38_{16} = 348.21875_{10}$

3e2.6

2.6 (a) $125_{10} = 1111101_2$ (b) $3489_{10} = 6641_8$
(c) $209_{10} = 11010001_2$ (d) $9714_{10} = 22762_8$
(e) $132_{10} = 1000100_2$ (f) $23851_{10} = 5D2B_{16}$
(g) $727_{10} = 10402_5$ (h) $57190_{10} = DF66_{16}$
(i) $1435_{10} = 2633_8$ (j) $65113_{10} = FE59_{16}$

4e2.7 2.7

(a)
$$\begin{array}{r} 100100 \\ 110011 \\ +\ 11010 \\ \hline 1001101 \end{array}$$

(b)
$$\begin{array}{r} 1011100 \\ 100111 \\ +\ 101010 \\ \hline 1010001 \end{array}$$

(c)
$$\begin{array}{r} 11111110 \\ 11100011 \\ +\ 1011101 \\ \hline 101000000 \end{array}$$

(d)
$$\begin{array}{r} 11000000 \\ 1100110 \\ +\ 1111001 \\ \hline 11011111 \end{array}$$

4e2.8 2.8

(a)	110000	(b)	110000	(c)	00111000	(d)	1110010
	110011		100111		11100011		1100110
	- 11010		- 101010		- 1011101		- 1111001
	011001		111101		10000110		1101101

4e2.9 2.9 (a)
$$\begin{array}{r} 1776 \\ +\ 1432 \\ \hline 3430 \end{array}$$
(b)
$$\begin{array}{r} 57734 \\ +\ 1066 \\ \hline 61022 \end{array}$$
(c)
$$\begin{array}{r} 252757 \\ +\ 465521 \\ \hline 740500 \end{array}$$
(d)
$$\begin{array}{r} 511042 \\ +\ 57647 \\ \hline 570711 \end{array}$$

4e2.10 2.10

(a)
$$\begin{array}{r} 1776 \\ +\ 1432 \\ \hline 2BA8 \end{array}$$

(b)
$$\begin{array}{r} 4F1A5 \\ +\ B8D5 \\ \hline 5AA7A \end{array}$$

(c)
$$\begin{array}{r} F35B \\ +\ 27E6 \\ \hline 11B41 \end{array}$$

(d)
$$\begin{array}{r} 1B90F \\ +\ C44E \\ \hline 27D5D \end{array}$$

4e2.11 2.11

decimal	+ 25	+ 120	+82	–42	–6	–111
signed-magnitude	00011001	01111000	01010010	10101010	10000110	11101111
two’s-complement	00011001	01111000	01010010	11010110	11111010	10010001
one’s-complement	00011001	01111000	01010010	11010101	11111001	10010000

4e2.12 2.12

	(a)	(b)	(c)	(d)
	11010100	10111111	01011101	01100001
+	11101011	11011111	00110001	00011111
	10111111	10011110	10001110	10000000
	no	no	yes	yes

3e2.13 2.13 *d*

3e2.14 2.14 The number of parity bits is minimized by making the array as close as possible to square, so the number of parity bits is on the order of $2\sqrt{n}+1$. The exact answer depends on the value of n, and will be either $2\lceil\sqrt{n}\rceil$ (e.g., for $10 \le n \le 12$ or $2\lceil\sqrt{n}\rceil+1$ (e.g., for $13 \le n \le 16$).

4e2.15 2.15 Christmas and Halloween, since Dec 25 = Oct 31.

4e2.16 2.16 C0FFEE

4e2.17 2.17 11000000 is the only one (–64).

3e2.16 2.18 (a) any $b > 6$ (b) $b = 8$
(c) any $b > 3$ (d) $b = 5$
(e) $b = 4$ (f) $b = 6$

For exclusive use of adopters of the book *Digital Design Principles and Practices*, Fourth Edition, by John F. Wakerly, ISBN 0-13-186389-4.

3e2.17 2.19 Assuming that the solution $x = 8$ is correct in some base $b > 8$, we can write the following equation using base-10 arithmetic:

$$5 \cdot 8^2 + (5 \cdot b + 0) \cdot 8 + 1 \cdot b^2 + 2 \cdot b + 5 = 0$$

Solving the quadratic, we get $b = 13$ or $b = 25$. Next, we try the $x = 5$ solution for both possible bases, and find that it works only for $b = 13$; the Martians had 13 fingers.

3e2.18 2.20

$$h_j = \sum_{i=0} b_{4j+i} \cdot 2^j$$

Therefore,

$$B = \sum_{i-0}^{4n-1} b_i \cdot 2^i = \sum_{i=0}^{n-1} h_i \cdot 16^i$$

$$-B = 2^{4n} - \sum_{i=0}^{4n-1} b_i \cdot 2^i = 16^n - \sum_{i=0}^{n-1} h_i \cdot 16^i$$

Suppose a $3n$-bit number B is represented by an n-digit octal number Q. Then the two's-complement of B is represented by the 8's-complement of Q.

3e2.19 2.21 The result follows directly from Tables 2–1 and 2–3. For any 4-bit string *B* and corresponding hex digit *H*, the ones' complement of *B* is represented by the 15s' complement of *H*. Since the ones' complement is obtained by complementing individual bits, we can complement them in groups of four to arrive at the result.

3e2.20 2.22 Cases 1 and 2 assume no overflow.

Case 1 $(x, y \geq 0)$:

$$\begin{aligned} [x+y] &= x+y \\ &= [x]+[y] \end{aligned}$$

Case 2 $(x, y < 0)$:

$$\begin{aligned} [x+y] &= 2^n - (|x| + |y|) \\ &= 2^n + 2^n - (|x| + |y|) \text{ modulo } 2^n \\ &= (2^n - |x|) + (2^n - |y|) \\ &= [x] + [y] \end{aligned}$$

Case 3 $(x < 0, y \geq 0)$:

subcase 3a: $|x| = |y|$, so $x + y = 0$

$$\begin{aligned} [x+y] &= 0 \\ &= 2^n \text{ modulo } 2^n \\ &= (2^n - |x|) + |y| \\ &= [x] + [y] \end{aligned}$$

subcase 3b: $|x| > |y|$, so $x + y < 0$

$$\begin{aligned} [x+y] &= 2^n - (|x| - |y|) \text{ (using signed-magnitude rules)} \\ &= (2^n - |x|) + |y| \\ &= [x] + [y] \end{aligned}$$

Case 4 $(x \geq 0, y < 0)$: $|x| \geq |y|$, so $x + y \geq 0$

3e2.21 **2.23** If $x < 0$, then $[x] = 2^n - 1 - |x|$. We want to show that $[x+y] = [x] + [y]$ modulo $2^n - 1$.

Cases 1 and 2 assume no overflow.

Case 1 $(x, y \geq 0)$:

$$\begin{aligned} |x+y| &= x + y \\ &= |x| + |y| \end{aligned}$$

Case 2 $(x, y < 0)$:

$$\begin{aligned} [x+y] &= x + y \\ &= (2^n - 1 + 2^n - 1 - (|x| + |y|) \text{ mod ulo } 2^n - 1) \\ &= ((2^n - 1 - |x|) + (2^n - 1 - |y|)) \\ &= x + y \end{aligned}$$

Case 3 $(x < 0, y \geq 0)$:

subcase 3a: $|x| = |y|$, so $x + y = 0$

$$\begin{aligned} [x] + |y| &= 0 \\ &= 2^n - 1 \text{ modulo } 2^n - 1 \\ &= 2^n - 1 - |x| + |x| \\ &= (2^n - 1 - |x|) + |y| \\ &= [x] + [y] \end{aligned}$$

subcase 3b: $|x| = |y|$, so $x + y < 0$

$$\begin{aligned} [x+y] &= 2^n - 1 - (|x| - |y|) \text{ (using signed-magnitude rules)} \\ &= (2^n - 1 - |x|) + y \\ &= x + y \end{aligned}$$

Case 4 $(x \geq 0, y < 0)$: $|x| \geq |y|$, so $x + y \geq 0$

$$\begin{aligned} [x+y] &= |x| - |y| \text{ (using signed-magnitude rules)} \\ &= |x| + 2^n - 1 - |y| \text{ modulo } 2^n - 1 \\ &= [x] + [y] \end{aligned}$$

3e2.22 2.24 Starting with the arrow pointing at any number, adding a positive number causes overflow if the arrow is advanced through the +7 to –8 transition. Adding a negative number to any number causes overflow if the arrow is not advanced through the +7 to –8 transition.

3e2.23 2.25 Case 1 $(X \geq 0)$ trivial

Case 2 $X < 0$ Let x be the positive magnitude of X.

$$X(\text{m-bit}) = 2^m - x$$

$$X(\text{n-bit}) = 2^m - x + \sum_{i=m}^{n-1} 2^i \text{ (append 1s)}$$

$$= 2^n - x$$

3e2.24 2.26 Let the binary representation of X be $x_{n-1}x_{n-2}\ldots x_1x_0$. Then we can write the binary representation of Y as $x_m x_{m-1}\ldots x_{1x_0}$, where $m = n-d$. Note that x_{m-1} is the sign bit of Y. The value of Y is

$$Y = -2^{m-1}\cdot x_{m-1} + \sum_{i=0} x_i \cdot 2^i$$

The value of X is

$$X = -2^{n-1}\cdot x_{n-1} + \sum_{i=0}^{n-2} x_i \cdot 2^i$$

$$= -2^{n-1}\cdot x_{n-1} + Y + 2^{m-1}\cdot x_{m-1} + \sum_{i=m-1}^{n-2} x_i \cdot 2^i$$

$$= -2^{n-1}\cdot x_{n-1} + Y + 2\cdot 2^{m-1} + \sum^{n-2} x_i \cdot 2^i$$

Case 1 $(x_{m-1} = 0)$ In this case, $X = Y$ if and only if $-2^{n-1}\cdot x_{n-1} + \sum_{i=m}^{n-2} x_i \cdot 2^i = 0$, which is true if and only if all of the discarded bits $(x_m \ldots x_{n-1})$ are 0, the same as x_{m-1}.

Case 2 $(x_{m-1} = 1)$ In this case, $X = Y$ if and only if $-2^{n-1}\cdot x_{n-1} + 2\cdot 2^{m-1} + \sum_{i=m}^{n-2} x_i \cdot 2^i = 0$, which is true if and only if all of the discarded bits $(x_m \ldots x_{n-1})$ are 1, the same as x_{m-1}.

3e2.25 2.27 If the radix point is considered to be just to the right of the leftmost bit, then the largest number is $1.11\cdots1$ and the 2's complement of D is obtained by subtracting it from 2 (singular possessive). Regardless of the position of the radix point, the 1s' complement is obtained by subtracting D from the largest number, which has all 1s (plural).

3e2.26 2.28 For the first part, note that the bit-by-bit complement of Y may be written $\overline{Y} = 2^n - 1 - Y$. Therefore, $Y = 2^n - 1 - \overline{Y}$ and

$$\begin{aligned} X - Y &= X - (2^n - 1 - \overline{Y}) \\ &= X + (-2^n + 1 + \overline{Y}) \\ &= (X + \overline{Y} + 1 - 2^n) \end{aligned}$$

Next, consider the behavior of an *n*-bit adder for the operation X +\overline Y+1 - 2n. Suppose that no carry is produced:

$$\begin{aligned} \text{no carry} &\Rightarrow X + \overline{Y} + 1 < 2^n \\ &\Rightarrow X < 2^n - 1 - \overline{Y} \\ &\Rightarrow X < \overline{Y} \\ &\Rightarrow X - \overline{Y} < 0 \\ &\Rightarrow \text{subtraction produces a borrow} \end{aligned}$$

Likewise, we can show

$$\begin{aligned} \text{carry} &\Rightarrow X + \overline{Y} + 1 \geq 2^n \\ &\Rightarrow X \geq 2^n - 1 - \overline{Y} \\ &\Rightarrow X \geq Y \\ &\Rightarrow X - Y \geq 0 \\ &\Rightarrow \text{subtraction does not produce a borrow} \end{aligned}$$

3e2.27 2.29 All 2n bits are needed to represent the product of the most negative number (-2^{n-1}) times itself: $-2^{n-1} \cdot -2^{n-1} = 2^{2n-1} = 01000\ldots000$.

3e2.28 2.30

$$B = -b_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} b_i \cdot 2^i$$

$$2B = -b_{n-1} \cdot 2^n + \sum_{i=0}^{n-2} b_i \cdot 2^{i+1}$$

Case 1 $(b_{n-1} = 0)$ First term is 0, summation terms have shifted coefficients as specified. Overflow if $b_{n-2} = 1$.

Case 2 $(b_{n-1} = 1)$ Split first term into two halves; one half is cancelled by summation term $b_{n-2} \cdot 2^{n-1}$ if $b_{n-2} = 1$. Remaining half and remaining summation terms have shifted coefficients as specified. Overflow if $b_{n-2} = 0$.

3e2.29 2.31 To multiply a ones'-complement number by two, shift it one bit position to the left, with end-around carry from the most significant bit position. The operation overflows if b_{n-1} and b_{n-2} have opposite values at the start of the operation.

For exclusive use of adopters of the book *Digital Design Principles and Practices*, Fourth Edition, by John F. Wakerly, ISBN 0-13-186389-4.

4e2.32 2.32 BCD Subtraction Rule: Subtract the 4-bit binary numbers using binary arithmetic. If there was a borrow, subtract 6 from the result and record a BCD borrow. Examples:

8	1000	4	0100	5	0101	2	0010
–3	–0011	–8	–1000	–9	–1001	–7	–0111
5	0101	12	1100, B	12	1100, B	11	1011, B
		– 6	–0110	– 6	–0110	–6	–0110
		6, B	0110, B	6, B	0110, B	5, B	0101, B

For exclusive use of adopters of the book *Digital Design Principles and Practices*, Fourth Edition, by John F. Wakerly, ISBN 0-13-186389-4.

4e2.33 2.33 With five states and eight possible 3-bit encodings, there are $\binom{8}{5}$ ways to choose the encodings, and $5!$ ways to assign encodings to states for each choice. Thus, the total is

$$\binom{8}{5} \cdot 5! = \frac{8!}{5! \cdot 3!} \cdot 5! = 8!/6 = 6720$$

For seven states and eight possible 3-bit encodings, there are $\binom{8}{7}$ ways to choose the encodings, and $7!$ ways to assign encodings to states for each choice. Thus, the total is

$$\binom{8}{7} \cdot 7! = \frac{8!}{7! \cdot 1!} \cdot 7! = 8!/1 = 40320$$

For eight states and eight possible 3-bit encodings, there is just one way to choose the encodings. Thus, the total is 8! or 40320. This is the same answer as with seven states, because with each 7-state encoding, there is exactly one leftover state which would be the eighth state in a corresponding 8-state encoding.

4e2.34 2.34 Basically, your boss has forbidden the 111 encoding. Since there are 6 states and now only seven allowed 3-bit encodings, there are $\binom{7}{6}$ ways to choose the encodings, and $6!$ ways to assign encodings to states for each choice. Thus, the total is

$$\binom{7}{6} \cdot 6! \;=\; \frac{7!}{6! \cdot 1!} \cdot 6! \;=\; 7!/1 \;=\; 5040$$

4e2.33 2.35 001–010, 011–100, 101–110, 111–000.

3e2.33 2.36 When the LSB changes from 0 to 1, there's no problem. When the LSB changes from 1 to 0, so does the next bit (and possibly others), guaranteeing a problem. So, half the boundaries are bad, a total of 2^{n-1} for an *n*-bit encoding disc.

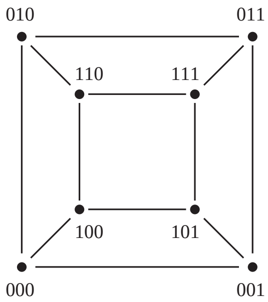
4e2.37 2.37 The manufacturer's code fails every 4th time, for a total of 2^{n-2} for an *n*-bit encoding disc. A standard binary code fails when the LSB changes from 1 to 0, which changes the next bit (and possibly others), guaranteeing a problem. So, half the boundaries in a standard binary code are bad, a total of 2^{n-1} for an *n*-bit encoding disc. The manufacturer's code is only half as bad as a standard binary code.

3e2.34 2.38 Perhaps the designers were worried about what would happen if the aircraft changed altitude in the middle of a transmission. With the Gray code, the codings of "adjacent" alitudes (at 50-foot increments) differ in only one bit. An altitude change during transmission affects only one bit, and whether the changed bit or the original is transmitted, the resulting code represents an altitude within one step (50 feet) of the original. With a binary code, larger altitude errors could result, say if a plane changed from 12,800 feet (000100000000_2) to 12,750 feet (000011111111_2) in the middle of a transmission, possibly yielding a result of 25,500 feet (000111111111_2).

3e2.35 2.39 If a pair of bits (H,L) represents the states of the high and low filaments in the bulb, a normal switch cycles through the sequence (0,0)–(0,1)–(1,0)–(1,1)–(0,0), yielding the lighting sequence OFF–DIM–MEDIUM–BRIGHT–OFF. According to the binary sequence, the low filament is stressed twice during each complete cycle. Less stress would occur if the switch used a Gray-code sequence, (0,0)–(0,1)–(1,1)–(1,0)–(0,0), yielding the lighting sequence OFF–DIM–BRIGHT–MEDIUM–OFF.

3e2.36 2.40 In the string representation, each position may have a 0, a 1, or an x, a total of three possibilities per position, and 3^n combinations in all.

3e2.37 2.41



3e2.38 2.42 It can’t be done, but it probably takes a topologist to prove it.

3e2.39 2.43 Think of it in terms of the string notation for subcubes. There are $\binom{n}{m}$ ways to pick the m positions that contain 0s and 1s, and 2^m ways to assign 0s and 1s to a particular set of m positions. So the total is $\binom{n}{m} \cdot 2^m$.

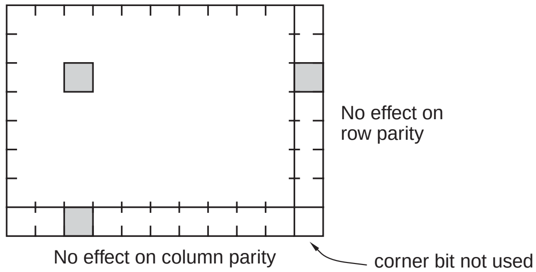
3e2.40 2.44 A possible Hamming matrix is .

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

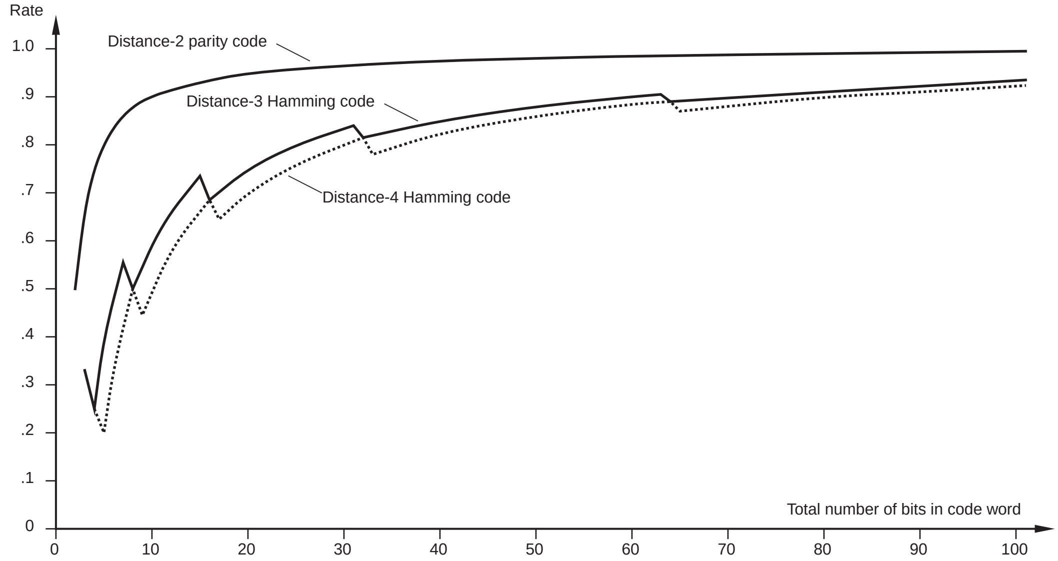
The first eleven columns correspond to information bits, and the last four to parity bits.

3e2.41 2.45 000, 111

3e2.42 2.46


No effect on
row parity
No effect on column parity
corner bit not used

3e2.43 2.47


Rate
1.0
.9
.8
.7
.6
.5
.4
.3
.2
.1
0
Distance-2 parity code
Distance-3 Hamming code
Distance-4 Hamming code
Total number of bits in code word
0
10
20
30
40
50
60
70
80
90
100

3e2.44 2.48

Minimum-distance-4 Hamming codes			
Information bits	Parity bits	Total bits	Rate
1	3	3	.25
≤ 4	4	≤ 8	.2–.5
≤ 11	5	≤ 16	.4444–.6875
≤ 26	6	≤ 32	.6471–.8125
≤ 57	7	≤ 64	.7879–.8906
≤ 120	8	≤ 128	.8769–.9375

Minimum-distance-4 two-dimensional codes				
Information matrix	Information bits	Parity bits	Total Bits	Rate
1×1	1	3	4	.25
1×2	2	4	6	.3333
2×2	3–4	5	8–9	.3750–.4444
2×3	5–6	6	11–12	.4545–.5000
3×3	7–9	7	14–16	.5000–.5625
3×4	10–12	8	18–20	.5556–.6000
4×4	13–16	9	22–25	.5909–.6400
4×5	17–20	10	27–30	.6296–.6667
5×5	21–25	11	32–36	.6563–.6944
5×6	26–30	12	38–42	.6842–.7143
6×6	31–36	13	44–49	.7045–.7347
6×7	37–42	14	51–56	.7255–.7500
7×7	43–49	15	58–64	.7414–.7656
7×8	50–56	16	66–72	.7576–.7778
8×8	57–64	17	74–81	.7703–.7901
8×9	65–72	18	83–90	.7831–.8000
9×9	73–81	19	92–100	.7935–.8100
9×10	82–90	20	102–110	.8039–.8182
10×10	91–100	21	112–121	.8125–.8264

3e2.45 2.49 To get a minimum distance of 6, we make a two-dimensional code using a distance-3 Hamming code for the rows and a distance-2 even-parity check for the columns. Three possible two-dimensional matrices with a total of four information bits are summarized below.

Row data bits	Row check bits	Column data bits	Column check bits	Matix size (rows $\times$ columns)	Total bits
4	3	1	1	2×7	14
2	3	2	1	3×5	15
1	2	4	1	5×3	15

To obtain the minimum number of bits in a code word, we select the first case. The row code is a 7-bit Hamming code with four information bits and three check bits. Such a code is given in the first two columns of Table 1–14 in the text. The column code is an even-parity check with one information bit and one check bit, which is equivalent to duplicating the information bit. Thus, the two rows of each 14-bit two-dimensional code word are simply duplicates of a 7-bit code word from Table 1–14.

If the third case is selected, we get a code that is even easier to construct, though it has one more bit, a total of 15. The first column contains four information bits and an even-parity check bit. The second and third columns are copies of the first column.

3e2.46 2.50 Read the old data in block *b* on drive *d*, and in block *b* on drive *n*+1 (the check drive). XOR the two blocks just read with the new data and store in block *b* on drive *n*+1. Store the new data in block *b*, drive *d*.

3e2.47 2.51