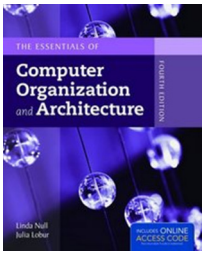


The Essentials of Computer Organization and Architecture
4th Edition

Linda Null and Julia Lobur
Jones and Bartlett Publishers © 2015

Chapter 2 Instructor's Manual

Chapter Objectives

Chapter 2, Data Representation, provides thorough coverage of the various means computers use to represent both numerical and character information. Addition, subtraction, multiplication, and division are covered once the reader has been exposed to number bases and the typical numeric representation techniques, including one's complement, two's complement, and BCD. In addition, EBCDIC, ASCII, and Unicode character representations are addressed. Fixed and floating point representations are also introduced. Codes for data recording and error detection and correction are covered briefly.

This chapter should be covered after Chapter 1, but before Chapters 4, 5, and 6.

Lectures should focus on the following points:

- **Number systems.** Most students have been exposed to positional number systems and different bases. However, these concepts are crucial to understanding the remainder of Chapter 2, so they should be covered in detail.
- **Decimal to binary conversions and binary to hexadecimal relationship.** Because the binary number system translates easily into electronic circuitry, it is important to become familiar with how computer represent values.
- **Signed versus unsigned numbers.** Representing unsigned numbers in binary form is much less complicated than dealing with signed numbers.
- **Signed integer representation.** There are basically three methods for representing signed numbers: signed magnitude, one's complement, and two's complement. Each of these methods should be covered, with the focus on signed magnitude and two's complement notations.
- **Binary arithmetic.** Although people do not often add binary values, performing binary addition and subtraction helps to reinforce the concepts necessary for understanding data

representation. In particular, these operations illustrate the dangers of overflow conditions.

- **Floating point representation.** Computers must be able to represent floating point numbers, and there are numerous possible formats for doing so. Potential errors that may result from the limitations of the representation are also important to discuss.
- **Character representation.** ASCII, EBCDIC, Unicode, and BCD are all important character codes. Lectures should emphasize the similarities and differences among these codes.
- **Error detection and correction.** Regardless of the coding method used, no communications channel or storage medium is error-free. Although simple parity bits can help to detect errors, more complicated codes, including cyclic redundancy checks and Hamming codes, and are often necessary for sophisticated error detection and correction.
- **Codes for data recording and transmission (in the Focus On section).** When binary data is written to some sort of medium or transmitted over long distances, the binary one's and zero's can become blurred. Some sort of encoding is necessary to ensure that characters are properly encoded in these situations.

Required Lecture Time

Chapter 2 can typically be covered in 6 lecture hours, depending on how detailed one wishes to go into recording and transmission codes and error detection and correction. We suggest that the focus be on integer, floating-point, and character representation, with emphasis given to complement notation. If time permits, data recording and transmission codes and error detection and correction codes can be covered.

Lecture Tips

Teachers should spend time explaining the ranges allowed by the different representation formats. For example, if we are using 4 bits to represent unsigned integers, we can represent values from 0 to 15. However, if we are using signed magnitude representation, the bits are interpreted differently, and the possible range is from −7 to +7. Make sure students understand why it is not possible to represent +9 or +10, as these will be seen as −1 and −2 respectively.

Converting unsigned whole numbers tends to be relatively straight-forward, but signed numbers and fractions tend to be more difficult. In particular, complement systems confuse students. **They often think that all numbers have to be negated to be represented.** For example, if a student is asked how a computer, using two's complement representation, would represent −6 (assuming 4-bit values), they answer: 1010 (take 0110, toggle each bit and add 1). This is, of course, the correct answer. However, if that student is asked how the computer using two's complement representation would represent +6, they will often do exactly the same thing, giving the same answer. Teachers need to be sure that students realize representing positive numbers does not require any "conversion." It is only when a number is

negative that two's complement, signed magnitude, and one's complement representations are necessary.

Instructors should spend time on overflow and look at several examples so that students can fully appreciate the consequences. Students find it difficult to understand why one time a carry results in overflow, but yet another carry may not result in overflow.

Floating point representation is fairly straight-forward, but students often have difficulty with the format and the bias. The simple model does not use an implied one as found in the IEEE standard.

Answers to Exercises

1. Perform the following base conversions using subtraction or division-remainder:

   - a) 458_{10} = _____ 3
   - b) 677_{10} = _____ 5
   - c) 1518_{10} = _____ 7
   - d) 4401_{10} = _____ 9

Ans.

a) 121222_3 b) 10202_5 c) 4266_7 d) 6030_9

---

2. Perform the following base conversions using subtraction or division-remainder:

   a) 588_{10} = _____ 3
   b) 2254_{10} = _____ 5
   c) 652_{10} = _____ 7
   d) 3104_{10} = _____ 9

Ans.

a) 210210_3 b) 33004_5 c) 1621_7 d) 4228_9

---

3. Perform the following base conversions using subtraction or division-remainder:

   a) 137_{10} = _____ 3
   b) 248_{10} = _____ 5
   c) 387_{10} = _____ 7
   d) 633_{10} = _____ 9

Ans.

a) 12002_3 b) 1443_5 c) 1062_7 d) 773_9

---

4. Perform the following base conversions.

a) 20101_3 = _____ 10
b) 2302_5 = _____ 10
c) 1605_7 = _____ 10
d) 687_9 = _____ 10

Ans.

a) 172_{10} b) 327_{10} c) 642_{10} d) 565_{10}

---

5. Perform the following base conversions.

a) 20012_3 = _____ 10
b) 4103_5 = _____ 10
c) 3236_7 = _____ 10
d) 1378_9 = _____ 10

Ans.

a) 140_{10} b) 528_{10} c) 1154_{10} d) 1043_{10}

---

6. Perform the following base conversions.

a) 21200_3 = _____ 10
b) 3244_5 = _____ 10
c) 3402_7 = _____ 10
d) 7657_9 = _____ 10

Ans.

a) 207_{10} b) 449_{10} c) 1227_{10} d) 5641_{10}

♦ 7. Convert the following decimal fractions to binary with a maximum of six places to the right of the binary point:

♦a) 26.78125 ♦b) 194.03125 ♦c) 298.796875 ♦d) 16.1240234375

Ans.

a) 11010.11001 b) 11000010.00001
c) 100101010.110011 d) 10000.000111

---

8. Convert the following decimal fractions to binary with a maximum of six places to the right of the binary point:

a) 25.84375 b) 57.55 c) 80.90625 d) 84.874023

Ans.

a) 11001.11011 b) 111001.100011
c) 1010000.11101 d) 1010100.110111

---

9. Convert the following decimal fractions to binary with a maximum of six places to the right of the binary point:

a) 27.59375 b) 105.59375 c) 241.53125 d) 327.78125

Ans.

a) 11011.10011 b) 1101001.10011
c) 11110001.10001 d) 101000111.11001

---

10. Convert the following binary fractions to decimal:

a) 10111.1101 b) 100011.10011 c) 1010011.10001 d) 11000010.111

Ans.

a) 23.8125 b) 35.59375
c) 83.53125 d) 194.875

---

11. Convert the following binary fractions to decimal:

a) 100001.111 b) 111111.10011 c) 1001100.1011 d) 10001001.0111

Ans.

a) 33.875 b) 63.59375
c) 76.6875 d) 137.4375

---

12. Convert the following binary fractions to decimal:

a) 110001.10101 b) 111001.001011 c) 1001001.10101 d) 11101001.110001

Ans.

a) 49.65625 b) 57.171875

c) 73.65625 d) 233.765625

---

13. Convert the hexadecimal number $AC12_{16}$ to binary.

Ans.

$AC12_{16} = 1010\ 1100\ 0001\ 0010_2$

---

14. Convert the hexadecimal number $7A01_{16}$ to binary.

Ans.

$7A01_{16} = 0111\ 1010\ 0000\ 0001_2$

---

15. Convert the hexadecimal number DEAD $BEEF_{16}$ to binary.

Ans.

DEAD $BEEF_{16} = 1101\ 1110\ 1010\ 1101\ 1011\ 1110\ 1110\ 1111_2$

---

16. Represent the following decimal numbers in binary using 8-bit signed magnitude, one's complement, two's complement, and excess-127 representation:

♦a) 77 ♦b) −42 c) 119 d) −107

Ans.

a) Signed magnitude: 01001101
One's complement: 01001101
Two's complement: 01001101
Excess-127: 11001100

b) Signed magnitude: 10101010
One's complement: 11010101
Two's complement: 11010110
Excess-127: 01010101

c) Signed magnitude: 01110111
One's complement: 01110111
Two's complement: 01110111
Excess-127: 11110110

d) Signed magnitude: 11101011
One's complement: 10010100
Two's complement: 10010101
Excess-127: 00010100

---

17. Represent the following decimal numbers in binary using 8-bit signed magnitude, one's complement, and two's complement representations:
a) 60 b) -60 c) 20 d) -20

Ans.

a) Signed magnitude: 00111100
One's complement: 00111100
Two's complement: 00111100
Excess-127: 10111011

b) Signed magnitude: 10111100
One's complement: 11000011
Two's complement: 11000100
Excess-127: 01000011

c) Signed magnitude: 00010100
One's complement: 00010100
Two's complement: 00010100
Excess-127: 10010011

d) Signed magnitude: 10010100
One's complement: 11101011
Two's complement: 11101100
Excess-127: 01101011

---

18. Represent the following decimal numbers in binary using 8-bit signed magnitude, one's complement, two's complement, and excess-127 representations:
a) 97 b) -97 c) 44 d) -44

Ans.

a) Signed magnitude: 01100001
One's complement: 01100001
Two's complement: 01100001
Excess-127: 11100000

b) Signed magnitude: 11100001
One's complement: 10011110
Two's complement: 10011111
Excess-127: 00011110

c) Signed magnitude: 00101100
One's complement: 00101100
Two's complement: 00101100
Excess-127: 10101011

d) Signed magnitude: 10101100
One's complement: 11010011
Two's complement: 11010100
Excess-127: 01010011

---

19. Represent the following decimal numbers in binary using 8-bit signed magnitude, one's complement, two's complement, and excess-127 representations:
a) 89 b) -89 c) 66 d) -66

Ans.

a) Signed magnitude: 01011001
One's complement: 01011001
Two's complement: 01011001
Excess-127: 11011000

b) Signed magnitude: 11011001
One's complement: 10100110
Two's complement: 10100111
Excess-127: 00100110

c) Signed magnitude: 01000010
One's complement: 01000010
Two's complement: 01000010
Excess-127: 11000001

d) Signed magnitude: 11000010
One's complement: 10111101
Two's complement: 10111110
Excess-127: 00111101

---

20. What decimal value does the 8-bit binary number 10011110 have if:
a) it is interpreted as an unsigned number?
b) it is on a computer using signed-magnitude representation?
c) it is on a computer using one's complement representation?
d) it is on a computer using two's complement representation?
e) it is on a computer using excess-127 representation?

Ans.

a) 158 b) -30 c) -97 d) -98 e) 31

---

21. What decimal value does the 8-bit binary number 00010001 have if:
    a) it is interpreted as an unsigned number?
    b) it is on a computer using signed-magnitude representation?
    c) it is on a computer using one's complement representation?
    d) it is on a computer using two's complement representation?
    e) it is on a computer using excess-127 representation?

Ans.

a) 17 b) 17 c) 17 d) 17 e) -110

---

22. What decimal value does the 8-bit binary number 10110100 have if:
    a) it is interpreted as an unsigned number?
    b) it is on a computer using signed-magnitude representation?
    c) it is on a computer using one's complement representation?
    d) it is on a computer using two's complement representation?
    e) it is on a computer using excess-127 representation?

Ans.

a) 180 b) -52 c) -75 d) -76 e) 53

---

23. Given the following two binary numbers: 11111100 and 01110000.
    a) Which of these two numbers is the larger unsigned binary number?
    b) Which of these two is the larger when it is being interpreted on a computer using signed-two's complement representation?
    c) Which of these two is the smaller when it is being interpreted on a computer using signed-magnitude representation?

Ans.

a) 11111100 b) 01110000 c) 11111100

---

24. Using a "word" of 3 bits, list all of the possible signed binary numbers and their decimal equivalents that are representable in:

    a) Signed magnitude b) One's complement c) Two's complement

Ans.

a) 011 to 111, or +3 to −3 b) 011 to 100, or +3 to −3 c) 011 to 100, or +3 to −4

---

25. Using a "word" of 4 bits, list all of the possible signed binary numbers and their decimal equivalents that are representable in:

a) Signed magnitude b) One's complement c) Two's complement

Ans.

a) 0111 to 1111, or +7 to −7 b) 0111 to 1000, or +7 to −7 c) 0111 to 1000, or +7 to −8

---

26. From the results of the previous two questions, generalize the range of values (in decimal) that can be represented in any given *x* number of bits using:

a) Signed magnitude b) One's complement c) Two's complement

Ans.

a) $-(2^{x-1}-1)$ to $+(2^{x-1}-1)$
b) $-(2^{x-1}-1)$ to $+(2^{x-1}-1)$
c) $-(2^{x-1})$ to $+(2^{x-1}-1)$

---

27. Fill in the following table to indicate what each binary pattern represents using the various formats.

Unsigned Integer	4-bit Binary Value	Signed Magnitude	1's Complement	2's Complement	Excess-7
0	0000				
1	0001				
2	0010				
3	0011				
4	0100				
5	0101				
6	0110				
7	0111				
8	1000				
9	1001				
10	1010				
11	1011				
12	1100				
13	1101				
14	1110				
15	1111				

Ans.

Unsigned Integer	4-bit Binary Value	Signed Magnitude	1's Complement	2's Complement	Excess-7
0	0000	0	0	0	-7
1	0001	1	1	1	-6
2	0010	2	2	2	-5
3	0011	3	3	3	-4
4	0100	4	4	4	-3
5	0101	5	5	5	-2
6	0110	6	6	6	-1
7	0111	7	7	7	0
8	1000	-0	-7	-8	1
9	1001	-1	-6	-7	2
10	1010	-2	-5	-6	3
11	1011	-3	-4	-5	4
12	1100	-4	-3	-4	5
13	1101	-5	-2	-3	6
14	1110	-6	-1	-2	7
15	1111	-7	-0	-1	8

---

28. Given a (very) tiny computer that has a word size of 6 bits, what are the smallest negative numbers and the largest positive numbers that this computer can represent in each of the following representations?

♦ a) One's complement b) Two's complement

Ans.

a) Largest Positive: 011111_2 (31) Smallest Negative: 100000_2 (−31)

b) Largest Positive: 011111_2 (31) Smallest Negative: 100000_2 (−32)

---

29. To add 2 two's complement numbers together, what must be true?

Ans.

They must be the same number of bits.

---

30. What is the most common representation used in most computers to store signed integer values and why?

Ans.

2s complement. The algorithm for adding and subtracting numbers is easier, it has the best representation for zero, it has a larger range of negative numbers, and it easily extends to larger numbers.

---

31. You have stumbled on an unknown civilization while sailing around the world. The people, who call themselves Zebronians, do math using 40 separate characters (probably because there are 40 stripes on a zebra). They would very much like to use computers, but would need a computer to do Zebronian math, which would mean a computer that could represent all 40 characters. You are a computer designer and decide to help them. You decide the best thing is to use BCZ, Binary Coded Zebronian (which is like BCD except it codes Zebronian, not Decimal). How many bits will you need to represent each character if you want to use the minimum number of bits?

Ans.

40 characters need to be represented by binary coded Zebronian (BCZ), so you will need 6 bits. 5 bits would only give you 32 (2^5) unique characters. Note that 6 bits would allow you to represent 64 characters.

♦ 32. Add the following unsigned binary numbers as shown.

a) 01110101 + 00111011

b) 00010101 + 00011011

c) 01101111 + 00010001

Ans. a) 10110000 b) 00110000 c) 10000000

---

33. Add the following unsigned binary numbers as shown.

a) 01000100 + 10111011

b) 01011011 + 00011111

c) 10101100 + 00100100

Ans. a) 11111111 b) 01111010 c) 11010000

♦ 34. Subtract the following signed binary numbers as shown using 2's complement arithmetic.

a) 01110101 – 00111011

b) 00110101 – 00001011

c) 01101111 – 00010001

Ans. a) 001110010 b) 00101010 c) 01011110

---

35. Subtract the following signed binary numbers as shown using 2's complement arithmetic.

a) 11000100 − 00111011

b) 01011011 − 00011111

c) 10101100 − 00100100

Ans. a) 10001001 b) 00111100 c) 10001000

---

36. Perform the following binary multiplications, assuming unsigned integers:

♦ a) 1100 × 101 b) 10101 × 111 c) 11010 × 1100

Ans.

a) 111100 b) 10010011 c) 100111000

---

37. Perform the following binary multiplications, assuming unsigned integers:

a) 1011 × 101 b) 10011 × 1011 c) 11010 × 1011

Ans.

a) 1000010 b) 1111001 c) 100011110

---

38. Perform the following binary divisions, assuming unsigned integers:

♦ a) 101101 ÷ 101 b) 10000001 ÷ 101 c) 1001010010 ÷ 1011

Ans.

a) 1001 b) 1101 c) 110110

---

39. Perform the following binary divisions, assuming unsigned integers:

a) 11111101 ÷ 1011 b) 110010101 ÷ 1001 c) 1001111100 ÷ 1100

Ans.

a) 10111 b) 101101 c) 110101

♦ 40. Use the double-dabble method to convert 10212_3 directly to decimal. (Hint: you have to change the multiplier.)

Ans.

10212_3 converted to decimal is 104, as shown below:

```
 1     0    2    1    2
       3    9   33  102
      +0   +2   +1   +2
       3   11   34  104
x3    x3   x3   x3
 3     9   33  102
```

---

41. Using signed-magnitude representation, complete the following operations:

+ 0 + (−0) =
(−0) + 0 =
0 + 0 =
(−0) + (−0) =

Ans. (assuming 4 bit representation)

+0 + (−0) = 0000 + 1000 = 1000 (−0)
(−0) + 0 = 1000 + 0000 = 1000 (−0)
0 + 0 = 0000 + 0000 = 0000 (+0)
(−0) + (−0) = 1000 + 1000 = 0000 (+0)

♦ 42. Suppose a computer uses 4-bit one's complement representation. Ignoring overflows, what value will be stored in the variable *j* after the following pseudocode routine terminates?

```
 0 → j   // Store 0 in j
−3 → k   // Store −3 in k
while k ≠ 0
 j = j + 1
 k = k − 1
end while
```

Ans.

J	(Binary)	K	(Binary)
0	0000	−3	1100
1	0001	−4	1011 (1100 + 1110) (where last carry is added to sum doing 1's complement addition)
2	0010	−5	1010 (1011 + 1110)
3	0011	−6	1001 (1010 + 1110)
4	0100	−7	1000 (1001 + 1110)
5	0101	7	0111 (1000 + 1110) (This is overflow -- but you can ignore)
6	0110	6	0110
7	0111	5	0101
−7	1000	4	0100
−6	1001	3	0011
−5	1010	2	0010

−4 1011 1 0001
−3 1100 0 0000

---

43. Perform the following binary multiplications using Booth's algorithm, assuming signed two's complement integers:

a) 1011 × 0101 b) 0011 × 1011 c) 1011 × 1100

Ans.

a) 11100111 b) 11110001 c) 00010100

---

44. Using arithmetic shifting, perform the following:
    a) double the value 00010101_2
    b) quadruple the value 01110111_2
    c) divide the value 11001010_2 in half

Ans.

a) 00101010 b) error (sign bit changes) c) 01100101

---

45. If the floating-point number representation on a certain system has a sign bit, a 3-bit exponent and a 4-bit significand:
    a) What is the largest positive and the smallest positive number that can be stored on this system if the storage is normalized? (Assume no bits are implied, there is no biasing, exponents use two's complement notation, and exponents of all zeros and all ones are allowed.)
    b) What bias should be used in the exponent if we prefer all exponents to be non-negative? Why would you choose this bias?

Ans.

a) Largest Positive: $0.1111_2 \times 2^3 = 111.1_2 = 7.5$
   Smallest Positive: $0.1_2 \times 2^{-4} = .00001_2 = 1/32 = 0.03125$

b) For all non-negative exponents, we would need a bias of 3 ($2^{3-1} - 1$).

♦ 46. Using the model in the previous question, including your chosen bias, add the following floating-point numbers and express your answer using the same notation as the addend and augend:

0	1	1	1	1	0	0	0
0	1	0	1	1	0	0	1

Calculate the relative error, if any, in your answer to the previous question.

Ans.

0	1	1	1	1	0	1	0

Error = 2.4%

---

47. Assume we are using the simple model for floating-point representation as given in this book (the representation uses a 14-bit format, 5 bits for the exponent with a bias of 15, a normalized mantissa of 8 bits, and a single sign bit for the number):
    a) Show how the computer would represent the numbers 100.0 and 0.25 using this floating-point format.
    b) Show how the computer would add the two floating-point numbers in part a by changing one of the numbers so they are both expressed using the same power of 2.
    c) Show how the computer would represent the sum in part b using the given floating-point representation. What decimal value for the sum is the computer actually storing? Explain.

Ans.

a) 100.0 =

0	1	0	1	1	0	1	1	0	0	1	0	0	0

0.25 =

0	0	1	1	1	0	1	0	0	0	0	0	0	0

b) Adding in part(a) we get:

$$\begin{aligned} .11001 \times 2^7 &= .11001 \times 2^7 \\ + \; .1 \times 2^{-1} &= \underline{.000000001 \times 2^7} \\ &= .110010001 \times 2^7 \end{aligned}$$

c) The sum from part b in the given floating point notation is:

0	1	0	1	1	0	1	1	0	0	1	0	0	0

which is $.11001 \times 2^7 = 1100100 = 100$.

---

48. What causes divide underflow and what can be done about it?

Ans.

Divide underflow is happens when the divisor is much smaller than the dividend. The computer may not be able to reconcile the two numbers of greatly different magnitudes, and thus the smaller gets represented by zero. The result of the division then becomes the equivalent of a division by zero error. Compare this to underflow, which is a condition that can occur when the result of a floating point operation would be smaller in magnitude (closer to zero, either positive or negative) than the smallest representable quantity. Division underflow could be avoided by using repeated subtraction.

---

49. Why do we usually store floating-point numbers in normalized form? What is the advantage of using a bias as opposed to adding a sign bit to the exponent?

Ans.

There are two reasons to use normalized form. First, normalized form creates a standard so each floating point value has a unique binary representation. Second, with normalized numbers, we can "imply" the high-order 1 in the significand, giving us an extra bit of accuracy for "free". Using a bias allows an extra bit position to be used for the magnitude of the exponent, as no sign bit is required.

---

50. Let $a = 1.0 \times 2^{9}$, $b = -1.0 \times 2^{9}$ and $c = 1.0 \times 2^{1}$. Using the floating-point model described in the text (the representation uses a 14-bit format, 5 bits for the exponent with a bias of 15, a normalized mantissa of 8 bits, and a single sign bit for the number), perform the following calculations, paying close attention to the order of operations. What can you say about the algebraic properties of floating-point arithmetic in our finite model? Do you think this algebraic anomaly holds under multiplication as well as addition?

b + (a + c) =
(b + a) + c =

Ans.

$$
\begin{aligned}
b + (a + c) &= b + [1.0 \times 2^{9} \\
&\quad + 1.0 \times 2^{1}] \\
&= b + [1.0 \times 2^{9} \\
&\quad + .00000001 \times 2^{9}) \\
&= b + 1.00000001 \times 2^{9} \\
&= b + 0.100000001 \times 2^{10} \quad \text{(but we can only have 8 bits in the mantissa)} \\
&= b + 0.10000000 \times 2^{10} \\
&= b + 1.0 \times 2^{9} \\
&= (-1.0 \times 2^{9}) + (1.0 \times 2^{9}) = 0
\end{aligned}
$$

$$
\begin{aligned}
(b + a) + c &= [(-1.0 \times 2^{9}) + (1.0 \times 2^{9})] + c \\
&= 0 + c \\
&= c \\
&= 1.0 \times 2^{1} = 10
\end{aligned}
$$

When one number is significantly larger than the other, round off error occurs due to the limited number of bits in the mantissa. Multiplication does not require the values to be expressed with the same powers of 2, and does not suffer from this problem.

---

51. Show how each of the following floating point values would be stored using IEEE-754 single precision (be sure to indicate the sign bit, the exponent, and the significand fields):

a) 12.5 b) −1.5 c) 0.75 d) 26.625

Ans.

a) $12.5 = 1.1001 \times 2^3$ 0 10000010 1001000...0
3+127 = 130 = 10000010

b) $-1.5 = -1.1 \times 2^0$ 1 01111111 1000000...0
0 + 127 = 127 = 01111111

c) $0.75 = 1.1 \times 2^{-1}$ 0 01111110 1000000...0
−1 + 127 = 126 = 01111110

d) $26.625 = 1.1010101 \times 2^4$ 0 10000011 1010101...0
4 + 127 = 131 = 10000011

---

52. Show how each of the following floating point values would be stored using IEEE-754 double precision (be sure to indicate the sign bit, the exponent, and the significand fields):

a) 12.5 b) −1.5 c) 0.75 d) 26.625

Ans.

a) $12.5 = 1.1001 \times 2^3$ 0 10000000010 1001000...0
3+1023 = 1026 = 10000000010

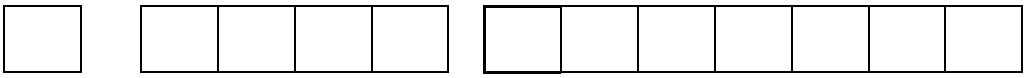
b) $-1.5 = -1.1 \times 2^0$ 1 01111111111 1000000...0
0 + 1023 = 1023 = 01111111111

c) $0.75 = 1.1 \times 2^{-1}$ 0 01111111110 1000000...0
−1 + 1023 = 1022 = 01111111110

d) $26.625 = 1.1010101 \times 2^4$ 0 10000000011 1010101...0
4 + 1023 = 1027 = 10000000011

---

53. Suppose we have just found yet another representation for floating point numbers. Using this representation, a 12-bit floating point number has 1 bit for the sign of the number, 4 bits for the exponent and 7 bits for the mantissa, which is normalized as in the Simple Model so that the first digit to the right of the radix points must be a 1. Numbers in the exponent are in signed 2's complement representation. No bias is used and there are no implied bits. Show the representation for the smallest positive number this machine can represent using the following format (simply fill in the squares provided). What decimal number does this equate to?



SIGN **EXPONENT** **MANTISSA**

Ans.

0 1000 1000000 = $.1 \times 2^{-7}$ = .00000001 = 1/256 = 0.00390625

---

54. Find three floating point values to illustrate that floating point addition is not associative. (You will need to run a program on specific hardware with a specific compiler.)

Ans.

No answer.

---

55. a) Given that the ASCII code for A is 1000001, what is the ASCII code for J?
b. Given that the EBCDIC code for A is 1100 0001, what is the EBCDIC code for J?

Ans)

a. If A = 1000001, then J = 1001010
b) If A = 1100 0001, then J = 1101 0001

---

56. a) The ASCII code for the letter A is 1000001, and the ASCII code for the letter a is 1100001. Given that the ASCII code for the letter G is 1000111, without looking at Table 2.7, what is the ASCII code for the letter g?

b) The EBCDIC code for the letter A is 1100 0001, and the EBCDIC code for the letter a is 1000 0001. Given that the EBCDIC code for the letter G is 1100 0111, without looking at Table 2.7, what is the ASCII code for the letter g?

c) The ASCII code for the letter A is 1000001, and the ASCII code for the letter a is 1100001. Given that the ASCII code for the letter Q is 1010001, without looking at Table 2.7, what is the ASCII code for the letter q?

d) The EBCDIC code for the letter J is 1101 0001, and the EBCDIC code for the letter j is 1001 0001. Given that the EBCDIC code for the letter Q is 1101 1000, without looking at Table 2.6, what is the ASCII code for the letter q?

e) In general, if you were going to write a program to convert uppercase ASCII characters to lowercase, how would you do it? Looking at Table 2.6, could you use the same algorithm to convert uppercase EBCDIC letters to lowercase?

f) If you were tasked with interfacing an ECIDIC-based computer with an ASCII or Unicode computer, what would be the best way to convert the EBCIDIC characters to ASCII characters?

Ans.

a) If G = 1000111, then g = 1100111

b) If G = 1100 0111, then g = 1000 0111
c) If Q = 1010001, then q = 1110001
d) If Q = 1101 1000, then q = 1001 1000
e) As you can see by Table 2.6, EBCDIC is arranged in a pattern of blocks rather than a numerical sequence like ASCII. A lowercase/uppercase conversion in ASCII is a matter of merely adding or subtracting 32. This algorithm does not work with EBCDIC.
f) The best approach for EBCDIC-to-ASCII conversions is through a lookup table. The EBCIDIC (or ASCII) character value would be the index and the character stored at the index location would be the ASCII (or EBCDIC) character.

♦ 57. Assume a 24-bit word on a computer. In these 24 bits, we wish to represent the value 295.
♦ a) If our computer uses even parity, how would the computer represent the string 295?
♦ b) If our computer uses 8-bit ASCII and even parity, how would the computer represent the string 295?
♦ c) If our computer uses packed BCD with zero padding, how would the computer represent the number +295?

Ans.

Binary Value	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 1	0 0 1 0 0 1 1 1
ASCII	1 0 1 1 0 0 1 0	0 0 1 1 1 0 0 1	0 0 1 1 0 1 0 1
Packed BCD	0 0 0 0 0 0 0 0	0 0 1 0 1 0 0 1	0 1 0 1 1 1 0 0

---

58. Decode the following ASCII message, assuming 7-bit ASCII characters and no parity:
1001010 1001111 1001000 1001110 0100000 1000100 1001111 1000101

Ans.

100 1010 = J
100 1111 = O
100 1000 = H
100 1110 = N
010 0000 = space
100 0100 = D
100 1111 = O
100 0101 = E

---

59. Why would a system designer wish to make Unicode the default character set for their new system? What reason(s) could you give for not using Unicode as a default? (Hint: Think about language compatibility versus storage space.)

Ans.

Although Unicode would make systems compatible (with each other and with other ASCII systems), it requires 16 bytes for storage compared to the 7 or 8 required by ASCII or EBCDIC.

---

60. Assume we wish to create a code using 3 information bits, 1 parity bit (appended to the end of the information), and odd parity. List all legal code words in this code. What is the Hamming distance of your code?

Ans.

The legal code words are:

0001	1000
0010	1011
0100	1101
0111	1110

and the Hamming distance of these code words is 2.

---

61. Suppose we are given the following subset of codewords, created for a 7-bit memory word with one parity bit: 11100110, 00001000, 10101011, and 11111110.
Does this code use even or odd parity? Explain.

Ans.

Each codeword has an odd numbers of bits so odd parity is used.

---

62. Are the error-correcting Hamming codes systematic? Explain.

Ans.

Error-correcting Hamming codes interleave additional error-checking (parity) bits into the actual information bits, but do not append these bits. In a systematic code, the first k bits of the codeword must be the same as the corresponding message bits (the parity bits must be appended), so by definition, Hamming codes are not systematic.

---

63. Compute the Hamming distance of the following code:

0011010010111100
0000011110001111
0010010110101101
0001011010011110

Ans.

4

---

64. Compute the Hamming distance of the following code:

0000000101111111
0000001010111111
0000010011011111
0000100011101111
0001000011110111
0010000011111011
0100000011111101
1000000011111110

Ans.

4

---

65. In defining the Hamming distance for a code, we choose to use the minimum (Hamming) distance between any two encodings. Explain why it would not be better to use the maximum or average distance.

Ans.

We must ensure than one code word cannot be transformed into another code word. Since we base the error detecting capabilities of a code using its Hamming distance, we must therefore use the smallest possible distance.

---

66. Suppose we want an error-correcting code that will allow all single-bit errors to be corrected for memory words of length 10.
    a) How many parity bits are necessary?
    b) Assuming we are using the Hamming algorithm presented in this chapter to design our error-correcting code, find the code word to represent the 10-bit information word: 1001100110.

Ans.

a) $m + r + 1 \leq 2^r$
   $10 + r + 1 \leq 2^r$
   $11 + r \leq 2^r$

   which implies that r should be 4

b) The code word for 1001100110 is found as follows:

1	0	0	1	1	0	1	0	1	1	1	0	0	0
14	13	12	11	10	9	8	7	6	5	4	3	2	1

Parity bit 1 checks 1,3,5,7,9,11,13, so Bit 1 must be 0 (assuming even parity)
Parity bit 2 checks 2,3,6,7,10,11,14, so Bit 2 must be 0
Parity bit 4 checks 4,5,6,7,12,13,14, so Bit 4 must be 1
Parity bit 8 checks 8,9,10,11,12,13,14, so Bit 8 must be 1

---

67. Suppose we want an error-correcting code that will allow all single-bit errors to be corrected for memory words of length 12.
    a) How many parity bits are necessary?
    b) Assuming we are using the Hamming algorithm presented in this chapter to design our error-correcting code, find the code word to represent the 12-bit information word: 100100011010.

Ans.

a) $m + r + 1 \leq 2^r$
$12 + r + 1 \leq 2^r$
$13 + r \leq 2^r$
which implies that r should be 5

b) The code word for 1001100110 is found as follows:

1	1	0	0	1	0	0	0	1	1	1	0	1	1	0	0	1
17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

Parity bit 1 checks 1,3,5,7,9,11,13, so Bit 1 must be 1 (assuming even parity)
Parity bit 2 checks 2,3,6,7,10,11,14, so Bit 2 must be 0
Parity bit 4 checks 4,5,6,7,12,13,14, so Bit 4 must be 1
Parity bit 8 checks 8,9,10,11,12,13,14,15 so Bit 8 must be 1
Parity bit 16 checks 16 and 1, so Bit 16 must be 1

♦ 68. Suppose we are working with an error-correcting code that will allow all single-bit errors to be corrected for memory words of length 7. We have already calculated that we need 4 check bits, and the length of all code words will be 11. Code words are created according to the Hamming Algorithm presented in the text. We now receive the following code word:

1 0 1 0 1 0 1 1 1 1 0

Assuming even parity, is this a legal code word? If not, according to our error-correcting code, where is the error?

Ans.

The error is in bit 5.

---

69. Repeat exercise 68 using the following code word:

0 1 1 1 1 0 1 0 1 0 1

Ans.

0	1	1	1	1	0	1	0	1	0	1
11	10	9	8	7	6	5	4	3	2	1

Bit 1 checks 1,3,5,7,9 and 11, but this is an odd number of 1's ⇒ error
Bit 2 checks 2,3,6,7,10, and 11, which is an odd number of 1's ⇒ error

Bit 4 checks 4,5,6, and 7, which is an even number of 1's ⇒ ok
Bit 8 checks 8,9,10, and 11, which is an odd number of 1's ⇒ error
Since errors occur in bit positions 1, 2, and 8, the error is in bit number 1+2+8=11

---

70. Suppose we are working with an error-correcting code that will allow all single-bit errors to be corrected for memory words of length 12. We have already calculated that we need 5 check bits, and the length of all code words will be 17. Code words are created according to the Hamming Algorithm presented in the text. We now receive the following code word:

0 1 1 0 0 1 0 1 0 0 1 0 0 1 0 0 1

Assuming even parity, is this a legal code word? If not, according to our error-correcting code, where is the error?

Ans.

The error is in bit 12.

The code word for 1001100110 is found as follows:

0	1	1	0	0	1	0	1	0	0	1	0	0	1	0	0	1
17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

Bit 1 checks 1,3,5,7,9,11,13, which is an odd number of 1's ⇒ ok
Bit 2 checks 2,3,6,7,10,11,14, which is an even number of 1's ⇒ ok
Bit 4 checks 4,5,6,7,12,13,14, which is an odd number of 1's ⇒ ok
Bit 8 checks 8,9,10,11,12,13,14,15, but this is an even number of 1's ⇒ error
Bit 16 checks 16 and 1, but this is an odd number of 1's ⇒ error

---

71. Name two ways in which Reed-Solomon coding differs from Hamming coding.

Ans.

Reed-Solomon codes are formed by polynomial division operating on whole characters (symbols), and are thus block-level codes. Hamming codes are bit-parity operations over certain defined bits of a symbol (so these are bit-level codes). Reed-Solomon codes are good at detecting and correcting burst errors, whereas Hamming codes are not.

---

72. When would you choose a CRC code over a Hamming code? A Hamming code over a CRC?

Ans.

CRCs are useful for checking data sent over telecommunication lines. If a CRC error occurs, a retransmission is requested. You would choose a CRC when you can ask for retransmission and do not want to sustain the (space and time) overhead of Hamming codes. Hamming codes are good at forward error correction: they can correct errors when retransmission is not possible, such as when data is stored on a disk. CRC is better than Hamming in terms of

speed, but Hamming is better than CRC in terms of complexity (Hamming codes do not require complex circuits).

---

73. Find the quotients and remainders for the following division problems modulo 2.
    - a) $1010111_2 \div 1101_2$
    - b) $1011111_2 \div 11101_2$
    - c) $1011001101_2 \div 10101_2$
    - d) $111010111_2 \div 10111_2$

Ans.

a) 1101 Remainder 110
b) 111 Remainder 1100
c) 100111 Remainder 110
d) 11001 Remainder 1000

---

74. Find the quotients and remainders for the following division problems modulo 2.
    a) $1111010_2 \div 1011_2$
    b) $1010101_2 \div 1100_2$
    c) $1101101011_2 \div 10101_2$
    d) $1111101011_2 \div 101101_2$

Ans.

a) 1101 Remainder 101
b) 1100 Remainder 101
c) 111011 Remainder 1100
d) 11010 Remainder 1001

---

75. Find the quotients and remainders for the following division problems modulo 2.
    a) $11001001_2 \div 1101_2$
    b) $1011000_2 \div 10011_2$
    c) $11101011_2 \div 10111_2$
    d) $111110001_2 \div 1001_2$

Ans.

a) 10010 Remainder 11
b) 101 Remainder 111
c) 1100 Remainder 1111
d) 11101 Remainder 0

---

76. Find the quotients and remainders for the following division problems modulo 2.
    a) $1001111_2 \div 1101_2$
    b) $1011110_2 \div 1100_2$
    c) $1001101110_2 \div 11001_2$
    d) $111101010_2 \div 10011_2$

Ans.

a) 1111 Remainder 100 b) 1101 Remainder 10
c) 111001 Remainder1111 d) 11100 Remainder 1110

♦77. Using the CRC polynomial 1011, compute the CRC code word for the information word, 1011001. Check the division performed at the receiver.

Ans.

Code word: 1011001011

---

78. Using the CRC polynomial 1101, compute the CRC code word for the information word, 01001101. Check the division performed at the receiver.

Ans.

The code word is 01001101100. Dividing this by 1101 modulo 2 should yield a zero remainder.

Append three 0s to the end of the information word and divide:

```
1101 | 01001101000
       1101
        1001
        1101
          1000
          1101
           1011
           1101
            1100
            1101
               100 → remainder
```

The information word (with appended zeros) + remainder = codeword
so we have: 01001101000 + 100 = 01001101100

To check the division:

```
1101 | 01001101100
       1101
        1001
        1101
        1000
        1101
         1011
         1101
          1101
          1101
```

0000 --> remainder

---

79. Using the CRC polynomial 1101, compute the CRC code word for the information word, 1100011. Check the division performed at the receiver.

Ans.

The code word is 1100011010. Dividing this by 1101 modulo 2 will yield a zero remainder.

---

80. Using the CRC polynomial 1101, compute the CRC code word for the information word, 01011101. Check the division performed at the receiver.

Ans.

The code word is 01011101101. Dividing this by 1101 modulo 2 will yield a zero remainder. Note: Ignore the leading zero in the information word when setting up the dividend, as illustrated in Problem 76.

---

81. Pick an architecture (such as 80486, Pentium, Pentium IV, SPARC, Alpha, or MIPS). Do research to find out how your architecture approaches the concepts introduced in this chapter. For example, what representation does it use for negative values? What character codes does it support?

Ans.

No answer.

---

82. We have seen that floating point arithmetic is neither associative nor distributive. Why do you think this is the case?

Ans.

Because some floating point numbers are not represented accurately, these properties do not hold.

Chapter 2A: Focus On Codes for Data Recording and Transmission

1. Why is non-return-to-zero coding avoided as a method for writing data to a magnetic media?

Ans.

NRZ coding is seldom used for recording data on magnetic media because it has unacceptably high error rates and not contain sufficient transitions to keep read/write heads synchronized.

---

2. Why is Manchester coding not a good choice for writing data to a magnetic disk?

Ans.

Manchester coding is inefficient for use in data storage. It would require twice the bit density of NRZ.

3. Explain how run-length-limited encoding works.

Ans.

The length of time between consecutive transition 0 to 1 or 1 to 0 is known as the run length. For example, the run lengths in the word 01111100111 are of length 1, 4, 2, and 3. RLL encoding limits the length of these sequences. There are two parameters, d and k, which stipulates a minimum of d and a maximum of k run lengths.

---

4. Write the 7-bit ASCII code for the character 4 using the following encoding:
   a) Non-return-to-zero b) Non-return-to-zero-invert c) Manchester Code
   d) Frequency modulation e) Modified frequency modulation f) Run length limited
   (Assume 1 is "high," and 0 is "low.")

Ans.