# Chapter One

DATA STORAGE

 **Chapter Summary**

This chapter presents the rudiments of data storage within digital computers. It introduces the basics of digital circuitry and how a simple flip-flop can be used to store a single bit. It then discusses addressable memory cells and mass storage systems (magnetic disk, compact disks, and flash memory). Having established this background, the chapter discusses how information (text, numeric values, images, and sound) are encoded as bit patterns and lays a foundation for the binary system. The optional sections in this chapter are not required for subsequent material in the text. However, these topics are relevant to Computer Science Principles and answer some common questions that students have about binary representation of information.

 **Comments**

1. This chapter could really be thought of as four separate chapters, covering the topics of Representation of Information as Bits, Storage of Bits in Memory with Logic Gates, Compression and Transmission of Bits, and Programming in Python. In a 15-week course, you may find that this material takes 2-3 full weeks, and in a high school course, this material may take 4-6 weeks. Although the Python component of the chapter is one section, students respond well to a regular, small amount of programming each week.

2. The material in section 1.1 often overwhelms students who have never seen it before. I prefer to start this chapter, with sections 1.4 (Representing Information as Bit Patterns) and 1.5 (The Binary System). I would then move through the remaining sections in order: 1.1, 1.2, 1.3, 1.6, 1.7, ….

3. Section 1.1 needs careful, and slow, explanations. It may help to give students extra examples of logic gates and combinations of logic gates before a Flip-Flop is introduced. One of the Student Activities for this chapter helps build this foundation. The textbook places the process for building adders using logic gates in Appendix B, but I have found it is worth the time to discuss adders in lecture.

4. While storing integers (1.6) and fractions (1.7) are complicated processes, they answer questions that students should have after reading section 1.5, namely, “How do computers represent negative numbers?” and “How do computers represent numbers in Scientific Notation?” Some students may find these sections difficult.

5. As mentioned above, the Python sections could be taught in little parts that occur regularly throughout the course. Decide ahead of time how you will support your students in setting up an IDE to use Python. Several free, cloud-based Python systems have been developed. I prefer these environments to those that must be installed on a machine, and I encourage my students to use them.

6. The sections on data compression and communication errors allow students a chance to act out some algorithms by hand. It can also reinforce the earlier ideas of the chapter. While these topics are not needed for future chapters, I find it gives students a taste of what computer scientists do. It’s important for students to understand that computers have practical limitations, but that computer scientists have developed ways to work around these limitations.

 **Answers to Chapter Review Problems**

1. With a 1 on the upper input and a 0 on the lower input, all circuits will produce an output 0. If instead a 0 is on the upper input and 1 is on the lower input, circuits b and c will produce an output 1, and circuit a will still produce a 0.

2. a. The entire circuit is equivalent to a single AND gate.

b. The entire circuit is equivalent to an Exclusive OR gate.

3. a. After the third pulse, this circuit will produce an output of 1 and 1. After the fourth pulse, both flip-flops are flipped back to a 0 state so the circuit will again produce an output of 0 and 0. It is interesting to note that this circuitry forms a binary counter that will repeatedly count from 00 to 11. Thus, this circuit forms an abstract tool that can be used as a building block in other circuits. Additional flip-flops can be added to count through a larger range of numbers.

b. A 1 will be sent on Output B on the 2nd, 6th, 10th … pulses of the clock. Likewise, a 1 will be sent to Output C on the 3rd, 7th, 11th … pulses of the clock. A 1 will not be sent to any output on the 4th, 8th, 12th … pulses of the clock. As we move forward into the next chapter, a circuit similar to this can be used to drive the machine cycle (composed of fetch, decode, and execute). Output A would be connected to the input that activates the fetch circuitry. Likewise Output B and Output C would be connected to the decode and execute circuits respectively.

4. This is a flip-flop that is triggered by 0s rather than 1s. That is, temporarily changing the upper input to 0 will establish an output of 1, whereas temporarily changing the lower input to 0 will establish an output of 0. To obtain an equivalent circuit using NAND gates, simply replace each AND-NOT gate pair with a NAND gate.

5. Address Contents

 0x00 0x02

 0x01 0x53

 0x02 0x01

 0x03 0x53

6. 256 using two hexadecimal digits (16 bits) , 65536 using four hexadecimal digits (32 bits).

7. a. 11001101 b. 01100111 c. 10011010 d. 11111111 e. 00010000

8. a. 1 b. 1 c. 0 d. 0

9. a. 0xA0A b. 0xC7B c. 0x0BE

10. The image consists of 1024 x 1024 = 1,048,576 pixels and therefore 3 x 1,048,576 = 3,145,728 bytes, or about 3MB. This means that about 86 images could be stored in the 256MB camera storage system. (By comparing this to actual camera storage capacities, students can gain an appreciation for the benefits of image compression techniques. Using them, a typical 256MB storage system can hold as many as 300 images.)

11. 786,432. (Each pixel would require one memory cell.)

12. Data retrieval from main memory is much faster than from disk storage. Also data in main memory can be referenced in byte-sized units rather than in large blocks. On the other hand, disk storage systems have a larger capacity than main memory and the data stored on disk is less volatile than that stored in main memory.

13. There are 70GB of material on the hard-disk drive. Each CD can hold no more than 700MB. Thus, it will require at least 100 CDs to store all the material. That does not seem practical. On the other hand, DVDs have capacities of about 4.7GB, meaning that only about 15 DVDs would be required. This may still be impractical, but its a big improvement over CDs. (The real point of this problem is to get students to think about storage capacities in a meaningful way.)

14. There would be about 5,000 characters on the page requiring two bytes for each two byte Unicode character. So the page would require about 10,000 bytes or 10 sectors of size 1024 bytes.

15. The novel would require about 1.4MB using ASCII and about 2.8MB if two byte Unicode characters were used.

16. The latency time of a disk spinning at 3600 revolutions per minute is only 0.00833 seconds.

17. About 11.4 milliseconds.

18. About 7 years!

19. What does it say?

20. hex notation

21. a. D o e s 1 0 0

01000010 01101111 01100101 01110011 00100000 00110001 00110000 00110000

 / 5 = 2

00100000 00101111 00100000 01101010 00100000 00111101 00100000 00110010

 0 ?

00110000 00111111

b.

 T h e t o t a

01010100 01101000 01100101 00100000 01110100 01101111 01110100 01100001

 l c o s t i

01101100 00100000 01100011 01101111 01110011 01110100 00100000 01101001

 s $ 7 . 2 5 .

01110011 00100000 00100100 00110111 00101110 00110010 00110101 00101110

22. a. 0x42 0x6F 0x65 0x73 0x20 0x31 0x30 0x30 0x20 0x2F 0x20 0x35 0x20 0x3D 0x20 0x32 0x30 0x3F

 b. 0x54 0x68 0x65 0x20 0x74 0x6F 0x74 0x61 0x6C 0x20 0x63 0x6F 0x74 0x20 0x69 0x73 0x20 0x24 0x37 0x2E 0x32 0x35 0x2E

23. 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111, 10000, 10001, 10010

24. a. 00110010 00110011 b. 10111

25. They are the powers of two. 1 10 100 1000 10000 100000

26. a. 15 b. 1 c. 21 d. 16 e. 19 f. 0 g. 5 h. 9 i. 17 j. 25 k. 26 l. 27

27. a. 111 b. 1011 c. 10000 d. 10001 e. 11111

28. a. 1 b. 5 c. -3 d. -1 e. 15

29. a. 100 b. 111 c. 010 d. 011 e. 110

30. a. 15 b. -12 c. 12 d. -16 e. -10

31. a. 0001101 b. 1110011 c. 1111111 d. 0000000 e. 0010000

32. a. 01101 b. 00000 c. 10000 (incorrect) d. 10001 e. 11110

 f. 10011 (incorrect) g. 11110 h. 01101 i. 10000 (incorrect) j. 11111

33. a.

 5 00101

 + 1 becomes + 00001

 00110 which represents 6

b.

 5 00101 00101

 - 1 becomes - 00001 which converts to + 11111

 00100 which represents 4

c.

 12 01100 01100

 - 5 becomes - 00101 which converts to + 11011

 00111 which represents 7

d.

 8 01000 01000

 - 7 becomes - 00111 which converts to + 11001

 00001 which represents 1

e.

 12 01100

 + 5 becomes + 00101

 10001 which represents -15 (overflow)

f.

 5 00101 00101

 - 11 becomes - 01011 which converts to + 10101

 11010 which represents -6

34. a. 3 3/4 b. 4 5/16 c. 13/16 d. 1 e. 2 1/4

35. a. 101.11 b. 1111.1111 c. 101.011 d. 1.01 e. 110.101

36. a. 1 1/8 b. -1/2 c. -3/16 d. 9/32

37. a. 11111111 b. 01001000 c. 11101111

 d. 00101110 e. 00011111 (truncation)

38. 00111100, 01000110, and 01010011

39. The best approximation of the square root of 2 is 1 3/8 represented as 01011011. The square of this value when represented in floating-point format is 01011111, which is the representation of 1 7/8.

40. The value one-eighth, which would be represented as 00101000.

41. Since the value one-tenth cannot be represented accurately, such recordings would suffer from truncation errors.

42. a. The value is either eleven or negative five.

 b. A value represented in two's complement notation can be changed to excess notation by changing the high-order bit, and vice versa.

43. The value is two; the patterns are excess, floating-point , and two's complement, respectively.

44. b would require too many significant digits. c would require too large of an exponent. d would require too many significant digits.

45. When using binary notation, the largest value that could be represented would change from 15 to 63. When using two's complement notation the largest value that could be represented would change from 7 to 31.

46. 0x4FFFFF

47. 1123221343435

48. yyxy xx yyxy xyx xx xyx

49. Starting with the first entries, they would be *x*, *y*, space, *xxy*, *yyx*, and *xxyy*.

50. Not a chance. MPEG requires transfer rates of 40 Mbps.

51. a.

 D o e s 1 0 0

001000010 001101111 001100101 101110011 100100000 100110001 000110000 000110000

 / 5 = 2

100100000 100101111 100100000 000110101 100100000 100111101 100100000 100110010

 0 ?

000110000 000111111

b.

 T h e t o t a

101010100 101101000 001100101 100100000 001110100 001101111 001110100 101100001

 l c o s t i

101101100 100100000 001100011 001101111 101110011 001110100 100100000 001101001

 s $ 7 . 2 5 .

101110011 100100000 100100100 100110111 000101110 100110010 000110101 000101110

52. The underlined strings definitely contain errors.

11001 11011 10110 00000 11111 10001 10101 00100 01110

53. The code would have a Hamming distance of 3. Thus, by using it, one could detect up to 2 errors per character and correct up to 1 error per character.

54. a. HE b. FED c. DEAD d. CABBAGE e. CAFE

55. Answers will vary as the exchange rates change daily.

56. Answers will vary depending on the currency selected.

57. This is an interactive exercise, results will depend on the browser.

58. A change to the dollar amount would need to be made in each of the expressions.

59.

no\_B = 123234234

no\_KB = no\_B / 1024

no\_MB = no\_KB / 1024

no\_GB = no\_MB / 1024

no\_TB = no\_TB / 1024

print(str(no\_B) + ' B')

print(str(no\_KB) + ' KB')

print(str(no\_MB) + ' MB')

print(str(no\_GB) + ' GB')

print(str(no\_TB) + ' TB')

print(str(no\_KB) + ' KB')

print(str(no\_KB) + ' KB')

no\_TB = 1.123

no\_GB = no\_TB \* 1024

no\_MB = no\_GB \* 1024

no\_KB = no\_MB \* 1024

no\_B = no\_KB \* 1024

print(str(no\_B) + ' B')

print(str(no\_KB) + ' KB')

print(str(no\_MB) + ' MB')

print(str(no\_GB) + ' GB')

print(str(no\_TB) + ' TB')

print(str(no\_KB) + ' KB')

print(str(no\_KB) + ' KB')

60.

mins = 50

secs = 23

tot\_secs = mins \* 60 + secs

samples\_per\_sec = 44100

bytes\_per\_sample = 2

tot\_bytes = tot\_secs \* samples\_per\_sec \* bytes\_per\_sample

print(tot\_bytes)

61. ‘\*\*’ should be ‘\*’ and ‘PRINT’ should be ‘print’

Chapter 1 Python Labs

These labs assume that your instructor has given you directions to:

- Choose a Python 3 development environment.

- Understand how to create Python files, run them, and save them.

**Python Lab 1a: The print function, Unicode values**

**Before you do this lab, you should review Section 1.8 in the text.**

|  |  |
| --- | --- |
| 1. Print your name to the console.  | In Python, the print function is used to print text-based information to the console. A typical function call would look like this:**print ('Welcome to my program! ' )**Use this syntax to print out your name. Use Quotations marks.  |
| 2. Print out your city of birth. | We could print out all of our program's output on a single line, but that would not be very easy to read. It’s typical to have a separate print () function call for each thing we want to print out. After your first line of code (from above), print out your city of birth. |
| 3. Print the results of mathematical expressions. | When you want Python to print something that is not a literal sequence of characters, you do not use the quotation marks. **print (3 \* 4 \* 5)**Add another line of code to print out the number of seconds in the current month. Have Python do the multiplication, using the example as a guide. |
| 4. Print the results of integer division and remainder.  | Use the // operator for integer division and % for the integer remainder. **print (44 // 3)****print (44 % 3)**Add code to calculate how many full dozens are made by 403 eggs, and how many eggs you have remaining.  |
| 4. Place variables inside a print() function.  | You can assign a value to a variable. Here are two examples, one that is an integer and one that is a string. **age = 20****job\_title = 'cashier'****print (age)****print (job\_title)**Think of other variables. Give them meaningful names. Use the = operator to store values in the variables. Print out the value of the variables.  |
| 5. Print out different Unicode values and their names.  | Each character in Unicode can be identified by a hexadecimal digit. Do a search on the web for a Unicode table. **print ('\u03bb' + ' Greek small letter lambda')**Print out 6 different Unicode values that you have never seen before. Use the + operator to concatenate the value and the name of the character.  |

**Python Lab 1b: Binary, Decimal, Hex**

**Before you do this lab, you should review Section 1.8 in the text.**

|  |  |
| --- | --- |
| 1. Print using the comma to separate inputs. | The print function, like other functions in Python, takes inputs. We can give the print function multiple inputs which are separated by a comma. In the console, the output is separated by a space. **print (21, 4\*32, 'hello')** |
| 2. Print out the value of a variable in binary and decimal form.  | The bin() and hex() functions will return the binary and hexadecimal representations of a variable. **x = 31****print (x, bin(x), hex(x)**)  |
| 3. Identify the error when giving an incorrect input type.  | Change the value of x to **x = 1.825**What error did you get? Why is this? Change the value of x to **x = 18** |
| 4. Assigning a binary or hex value to a variable. | You can assign binary or hex values to a variable.**y = 0b1011****z = 0xA3****print (y, z)** |
| 5. You can add numbers in any representation. | You can add variables in any representation.**w = x + y + z****print ('the sum is ' , w)** |
| 7. If Time | The comma in a print function separates inputs. There are ways to make the comma separate in a different way. Investigate this and try out different ways to use the comma in a print statement.  |

**Python Lab 1c: Calculating Compression Rate**

**Before you do this lab, you should review Section 1.8 in the text.**

In Activity 1c, you learned about lossless compression. In this lab, you will write a program which calculates the percent of compression.

|  |  |
| --- | --- |
| 1. Choose meaningful variables.  | This program will calculate the percent of compression given the formula: Percent of compression = (Compressed Text Size + Dictionary size) divided by original size. Subtract this value from 100%. **original\_size = 240****dictionary\_size = 5** |
| 2. Assign values to your variables. Calculate the result.  | Imagine you have an original text size of 240 characters, a compressed text size of 148 characters, and a dictionary size of 5 characters. The percent of compression should be 0.2792, or 27.92%. |
| 3. Add comments. | At the top of your code, add comments for your name. Add a comment explaining what the program does.If your code has some complicated statements, add comments explaining what they do. |
| 4 Print out your result. | Print your results. Make your output similar to the following.**Compressed text size: 148 characters Dictionary size: 25 characters Total: 173 characters Original text size: 240 characters Compression: 27.92%**Use spaces before the words to make the semicolons line up. Include spaces around numbers. **print (' Dictionary size:', str(d\_size), 'characters')** |
| 5. Double-check your program.  | Test your program with different inputs to make sure that it works with many different possible inputs. Did you remember to print out the percent sign in the compression?  |

**Python Lab 1d: Run Length Encoding a picture**

**Before you do this lab, you should review Section 1.8 in the text**

|  |  |
| --- | --- |
| 1. Concatenate and Replicate. | Strings in Python can be concatenated with the + operator.**print ('@' + '.' + '')**and replicated with the \* operator**print ('@@@' \* 3 + '...)**Write a single print statement that prints out **&&&&..&&&&..&&&&..&&&&..&&&&..+++++**Using replication and concatenation.  |
| 2. Make a grid picture.Fill certain squares to make a picture.  | In the 10x10 grid below, color in squares to make a picture that represents your two initials, or an emoji, or a picture.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

 |
| 3. Print out a textual representation of the picture using concatenate and replicate operators.  | Using 10 print() commands, make the picture in the console. Use one character (such as '@') for a filled in square, and another character (such as '.') for empty squares. Use the concatenate and replicate operators.  |

# Chapter One

DATA STORAGE

 **Chapter Summary**

This chapter presents the rudiments of data storage within digital computers. It introduces the basics of digital circuitry and how a simple flip-flop can be used to store a single bit. It then discusses addressable memory cells and mass storage systems (magnetic disk, compact disks, and flash memory). Having established this background, the chapter discusses how information (text, numeric values, images, and sound) are encoded as bit patterns. There are two optional sections, first in a brief introduction to Python highlighting the means by which a programming languages hides the details of memory represention. The second delves more deeply into data storage topics by presenting the problems of overflow errors, truncation errors, error detection and correction techniques, and data compression.

 **Comments**

1. Perhaps the most important comment I can make about this chapter (and the next one as well) is to explain its role in the chapters that follow. This involves the distinction between exposing students to a subject and requiring them to master the material—a distinction that is at the heart of the spirit in which the entire text was written. The intention of this chapter is to provide a realistic exposure to a very important area of computer science. It is not necessary for the students to master the material. All that is needed from this chapter in the remaining part of the book are the remnants that remain from a brief exposure to the issues of data storage. Even if the course you teach requires a mastery of these details or the development of manipulation skills, I encourage you to avoid emphasizing bit manipulations and representation conversions. In particular, I urge you to avoid becoming bogged down in the details of converting between base ten and binary notation. I can’t think of anything that would be more boring for the students. (I apologize for stating my opinion.)

2. The “required” sections in this chapter cover the composition of main memory (as a background for machine architecture in chapter 2 and data structures in chapter 8), the physical issues of external data storage systems (in preparation for the subjects of file and database systems in chapter 9), and the rudiments of data encoding (that serves as a background for the subject of data types and high-level language declaration statements in chapter 6). The optional sections explore the issues of error handling, including transmission error detection and correction as well as the problem of truncation and overflow errors resulting from numeric coding systems.

3. As mentioned in the preface of the text, there are several themes that run throughout the text, one of which is the role of abstraction. I like to include this theme in my lecture in which I introduce flip-flops. I end up with both flip-flop diagrams from the text on the board, and I emphasize that they represent two ways of accomplishing the same task. I then draw a rectangle around each diagram and erase the circuits within the rectangles leaving only the inputs, outputs, and rectangles showing. At this point the two look identical. I think that this creates a strong visual image that drives home the distinction between an abstract tool’s interface with the outside world and the internal details of the tool.

This is a specific example of teaching several topics at the same time—in this case, the concepts of abstraction and encapsulation are taught in the context of teaching digital circuits.

4. Don’t forget about the circuits in Appendix B. I used to have students who continued to record an extra bit in the answer to a two’s complement addition problem when a carry occurred—even though I had explained that all values in a two’s complement system were represented with the same number of bits. Once I started presenting the addition circuit in Appendix B, this problem disappeared. It gave the students a concrete understanding that the carry is thrown away. (Of course, in a later course computing students will learn that it really isn’t thrown away but saved as the "carry bit" for potential use in the future, but for now I ignore this.) I have also found that a good exercise is to ask students to extend the circuit in Figure B.3 so that it produces an additional output that indicates whether an overflow has occurred. For example, the output could be 1 in the case of an overflow and 0 otherwise.

5. For most students, seeing the reality of the things they are told is a meaningful experience. For this reason I often find it advantageous to demonstrate the distinction between numeric and character data using a spreadsheet. I like to show them how the manipulation of large numbers can lead to errors.

7. I have found that students respond well to hearing about CD and DVD storage systems, how sound is encoded, and image representation systems such as GIF and JPEG. I have often used these topics as a way of getting non-majors interested in technical issues.

8. For students not majoring in computer science, topics such as two's complement and floating-point notation can get a bit dry. The main point for them to understand is that when information is encoded, some information usually gets lost. This point can be made just as well using audio and video, which are contexts that seem to be more interesting to the non-majors.

 **Answers to Chapter Review Problems**

1. With a 1 on the upper input and a 0 on the lower input, all circuits will produce an output 0. If instead a 0 is on the upper input and 1 is on the lower input, circuits b and c will produce an output 1, and circuit a will still produce a 0.

2. a. The entire circuit is equivalent to a single AND gate.

b. The entire circuit is equivalent to an Exclusive OR gate.

3. a. After the third pulse, this circuit will produce an output of 1 and 1. After the fourth pulse, both flip-flops are flipped back to a 0 state so the circuit will again produce an output of 0 and 0. It is interesting to note that this circuitry forms a binary counter that will repeatedly count from 00 to 11. Thus, this circuit forms an abstract tool that can be used as a building block in other circuits. Additional flip-flops can be added to count through a larger range of numbers.

b. A 1 will be sent on Output B on the 2nd, 6th, 10th … pulses of the clock. Likewise, a 1 will be sent to Output C on the 3rd, 7th, 11th … pulses of the clock. A 1 will not be sent to any output on the 4th, 8th, 12th … pulses of the clock. As we move forward into the next chapter, a circuit similar to this can be used to drive the machine cycle (composed of fetch, decode, and execute). Output A would be connected to the input that activates the fetch circuitry. Likewise Output B and Output C would be connected to the decode and execute circuits respectively.

4. This is a flip-flop that is triggered by 0s rather than 1s. That is, temporarily changing the upper input to 0 will establish an output of 1, whereas temporarily changing the lower input to 0 will establish an output of 0. To obtain an equivalent circuit using NAND gates, simply replace each AND-NOT gate pair with a NAND gate.

5. Address Contents

 00 02

 01 53

 02 01

 03 53

6. 256 using two hexadecimal digits (16 bits) , 65536 using four hexadecimal digits (32 bits).

7. a. 11001101 b. 01100111 c. 10011010 d. 11111111 e. 00010000

8. a. 1 b. 1 c. 0 d. 0

9. a. A0A b. C7B c. 0BE

10. The image consists of 1024 x 1024 = 1,048,576 pixels and therefore 3 x 1,048,576 = 3,145,728 bytes, or about 3MB. This means that about 86 images could be stored in the 256MB camera storage system. (By comparing this to actual camera storage capacities, students can gain an appreciation for the benefits of image compression techniques. Using them, a typical 256MB storage system can hold as many as 300 images.)

11. 786,432. (Each pixel would require one memory cell.)

12. Data retrieval from main memory is much faster than from disk storage. Also data in main memory can be referenced in byte-sized units rather than in large blocks. On the other hand, disk storage systems have a larger capacity than main memory and the data stored on disk is less volatile than that stored in main memory.

13. There are 70GB of material on the hard-disk drive. Each CD can hold no more than 700MB. Thus, it will require at least 100 CDs to store all the material. That does not seem practical to me. On the other hand, DVDs have capacities of about 4.7GB, meaning that only about 15 DVDs would be required. This may still be impractical, but its a big improvement over CDs. (The real point of this problem is to get students to think about storage capacities in a meaningful way.)

14. There would be about 5,000 characters on the page requiring two bytes for each two byte Unicode character. So the page would require about 10,000 bytes or 10 sectors of size 1024 bytes.

15. The novel would require about 1.4MB using ASCII and about 2.8MB if two byte Unicode characters were used.

16. The latency time of a disk spinning at 3600 revolutions per minute is only 0.00833 seconds.

17. About 11.4 milliseconds.

18. About 7 years!

19. What does it say?

20. hex notation

21. a. D o e s </inst>1 0 0

01000010 01101111 01100101 01110011 00100000 00110001 00110000 00110000

 / 5 = 2

00100000 00101111 00100000 01101010 00100000 00111101 00100000 00110010

 0 ?

00110000 00111111

b.

 T h e t o t a

01010100 01101000 01100101 00100000 01110100 01101111 01110100 01100001

 l c o s t i

01101100 00100000 01100011 01101111 01110011 01110100 00100000 01101001

 s $ 7 . 2 5 .

01110011 00100000 00100100 00110111 00101110 00110010 00110101 00101110

22. a. 42 6F 65 73 20 31 30 30 20 2F 20 35 20 3D 20 32 30 3F

 b. 54 68 65 20 74 6F 74 61 6C 20 63 6F 74 20 69 73 20 24 37 2E 32 35 2E

23. 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111, 10000, 10001, 10010

24. a. 00110010 00110011 b. 10111

25. They are the powers of two. 1 10 100 1000 10000 100000

26. a. 15 b. 1 c. 21 d. 16 e. 19 f. 0 g. 5 h. 9 i. 17 j. 25 k. 26 l. 27

27. a. 111 b. 1011 c. 10000 d. 10001 e. 11111

28. a. 1 b. 5 c. -3 d. -1 e. 15

29. a. 100 b. 111 c. 010 d. 011 e. 110

30. a. 15 b. -12 c. 12 d. -16 e. -10

31. a. 0001101 b. 1110011 c. 1111111 d. 0000000 e. 0010000

32. a. 01101 b. 00000 c. 10000 (incorrect) d. 10001 e. 11110

 f. 10011 (incorrect) g. 11110 h. 01101 i. 10000 (incorrect) j. 11111

33. a.

 5 00101

 + 1 becomes + 00001

 00110 which represents 6

b.

 5 00101 00101

 - 1 becomes - 00001 which converts to + 11111

 00100 which represents 4

c.

 12 01100 01100

 - 5 becomes - 00101 which converts to + 11011

 00111 which represents 7

d.

 8 01000 01000

 - 7 becomes - 00111 which converts to + 11001

 00001 which represents 1

e.

 12 01100

 + 5 becomes + 00101

 10001 which represents -15 (overflow)

f.

 5 00101 00101

 - 11 becomes - 01011 which converts to + 10101

 11010 which represents -6

34. a. 3 3/4 b. 4 5/16 c. 13/16 d. 1 e. 2 1/4

35. a. 101.11 b. 1111.1111 c. 101.011 d. 1.01 e. 110.101

36. a. 1 1/8 b. -1/2 c. -3/16 d. 9/32

37. a. 11111111 b. 01001000 c. 11101111

 d. 00101110 e. 00011111 (truncation)

38. 00111100, 01000110, and 01010011

39. The best approximation of the square root of 2 is 1 3/8 represented as 01011011. The square of this value when represented in floating-point format is 01011111, which is the representation of 1 7/8.

40. The value one-eighth, which would be represented as 00101000.

41. Since the value one-tenth cannot be represented accurately, such recordings would suffer from truncation errors.

42. a. The value is either eleven or negative five.

 b. A value represented in two's complement notation can be changed to excess notation by changing the high-order bit, and vice versa.

43. The value is two; the patterns are excess, floating-point , and two's complement, respectively.

44. b would require too many significant digits. c would require too large of an exponent. d would require too many significant digits.

45. When using binary notation, the largest value that could be represented would change from 15 to 63. When using two's complement notation the largest value that could be represented would change from 7 to 31.

46. 4FFFFF

47. 1123221343435

48. yyxy xx yyxy xyx xx xyx

49. Starting with the first entries, they would be *x*, *y*, space, *xxy*, *yyx*, and *xxyy*.

50. Not a chance. MPEG requires transfer rates of 40 Mbps.

51. a.

 D o e s </inst>1 0 0

001000010 001101111 001100101 101110011 100100000 100110001 000110000 000110000

 / 5 = 2

100100000 100101111 100100000 000110101 100100000 100111101 100100000 100110010

 0 ?

000110000 000111111

b.

 T h e t o t a

101010100 101101000 001100101 100100000 001110100 001101111 001110100 101100001

 l c o s t i

101101100 100100000 001100011 001101111 101110011 001110100 100100000 001101001

 s $ 7 . 2 5 .

101110011 100100000 100100100 100110111 000101110 100110010 000110101 000101110

52. The underlined strings definitely contain errors.

11001 11011 10110 00000 11111 10001 10101 00100 01110

53. The code would have a Hamming distance of 3. Thus, by using it, one could detect up to 2 errors per character and correct up to 1 error per character.

54. a. HE b. FED c. DEAD d. CABBAGE e. CAFE

55. Answers will vary as the exchange rates change daily.

56. Answers will vary depending on the currency selected.

57. This is an interactive exercise, results will depend on the browser.

58. A change to the dollar amount would need to be made in each of the expressions.

59.

no\_B = 123234234

no\_KB = no\_B / 1024

no\_MB = no\_KB / 1024

no\_GB = no\_MB / 1024

no\_TB = no\_TB / 1024

print(str(no\_B) + ' B')

print(str(no\_KB) + ' KB')

print(str(no\_MB) + ' MB')

print(str(no\_GB) + ' GB')

print(str(no\_TB) + ' TB')

print(str(no\_KB) + ' KB')

print(str(no\_KB) + ' KB')

no\_TB = 1.123

no\_GB = no\_TB \* 1024

no\_MB = no\_GB \* 1024

no\_KB = no\_MB \* 1024

no\_B = no\_KB \* 1024

print(str(no\_B) + ' B')

print(str(no\_KB) + ' KB')

print(str(no\_MB) + ' MB')

print(str(no\_GB) + ' GB')

print(str(no\_TB) + ' TB')

print(str(no\_KB) + ' KB')

print(str(no\_KB) + ' KB')

60.

mins = 50

secs = 23

tot\_secs = mins \* 60 + secs

samples\_per\_sec = 44100

bytes\_per\_sample = 2

tot\_bytes = tot\_secs \* samples\_per\_sec \* bytes\_per\_sample

print(tot\_bytes)

61. ‘\*\*’ should be ‘\*’ and ‘PRINT’ should be ‘print’

# Chapter One

DATA STORAGE

 **Chapter Summary**

This chapter presents the rudiments of data storage within digital computers. It introduces the basics of digital circuitry and how a simple flip-flop can be used to store a single bit. It then discusses addressable memory cells and mass storage systems (magnetic disk, compact disks, and flash memory). Having established this background, the chapter discusses how information (text, numeric values, images, and sound) are encoded as bit patterns and lays a foundation for the binary system. The optional sections in this chapter are not required for subsequent material in the text. However, these topics are relevant to Computer Science Principles and answer some common questions that students have about binary representation of information.

 **Comments**

1. This chapter could really be thought of as four separate chapters, covering the topics of Representation of Information as Bits, Storage of Bits in Memory with Logic Gates, Compression and Transmission of Bits, and Programming in Python.  In a 15-week course, you may find that this material takes 2-3 full weeks, and in a high school course, this material may take 4-6 weeks. Although the Python component of the chapter is one section, students respond well to a regular, small amount of programming each week.

 2.  The material in section 1.1 often overwhelms students who have never seen it before.  I prefer to start this chapter, with sections 1.4 (Representing Information as Bit Patterns) and 1.5 (The Binary System).  I would then move through the remaining sections in order: 1.1, 1.2, 1.3, 1.6, 1.7, ….

3.  Section 1.1 needs careful, and slow, explanations.  It may help to give students extra examples of logic gates and combinations of logic gates before a Flip-Flop is introduced.  One of the Student Activities for this chapter helps build this foundation. The textbook places the process for building adders using logic gates in Appendix B, but I have found it is worth the time to discuss adders in lecture.

4.  While storing integers (1.6)  and fractions (1.7) are complicated processes, they answer questions that students should have after reading section 1.5, namely,  “How do computers represent negative numbers?” and “How do computers represent numbers in Scientific Notation?” Some students may find these sections difficult.

5.  As mentioned above, the Python sections could be taught in little parts that  occur regularly throughout the course. Decide ahead of time how you will support your students in setting up an IDE to use Python. Several free, cloud-based Python systems have been developed.  I prefer these environments to those that must be installed on a machine, and I encourage my students to use them.

6.  The sections on data compression and communication errors allow students a chance to act out some algorithms by hand. It can also reinforce the earlier ideas of the chapter.  While these topics are not needed for future chapters, I find it gives students a taste of what computer scientists do. It’s important for students to understand that computers have practical limitations, but that computer scientists have developed ways to work around these limitations.

 **Answers to Chapter Review Problems**

1. With a 1 on the upper input and a 0 on the lower input, all circuits will produce an output 0. If instead a 0 is on the upper input and 1 is on the lower input, circuits b and c will produce an output 1, and circuit a will still produce a 0.

2. a. The entire circuit is equivalent to a single AND gate.

b. The entire circuit is equivalent to an Exclusive OR gate.

3. a. After the third pulse, this circuit will produce an output of 1 and 1. After the fourth pulse, both flip-flops are flipped back to a 0 state so the circuit will again produce an output of 0 and 0. It is interesting to note that this circuitry forms a binary counter that will repeatedly count from 00 to 11. Thus, this circuit forms an abstract tool that can be used as a building block in other circuits. Additional flip-flops can be added to count through a larger range of numbers.

b. A 1 will be sent on Output B on the 2nd, 6th, 10th … pulses of the clock. Likewise, a 1 will be sent to Output C on the 3rd, 7th, 11th … pulses of the clock. A 1 will not be sent to any output on the 4th, 8th, 12th … pulses of the clock. As we move forward into the next chapter, a circuit similar to this can be used to drive the machine cycle (composed of fetch, decode, and execute). Output A would be connected to the input that activates the fetch circuitry. Likewise Output B and Output C would be connected to the decode and execute circuits respectively.

4. This is a flip-flop that is triggered by 0s rather than 1s. That is, temporarily changing the upper input to 0 will establish an output of 1, whereas temporarily changing the lower input to 0 will establish an output of 0. To obtain an equivalent circuit using NAND gates, simply replace each AND-NOT gate pair with a NAND gate.

5. Address Contents

 00 02

 01 53

 02 01

 03 53

6. 256 using two hexadecimal digits (16 bits) , 65536 using four hexadecimal digits (32 bits).

7. a. 11001101 b. 01100111 c. 10011010 d. 11111111 e. 00010000

8. a. 1 b. 1 c. 0 d. 0

9. a. A0A b. C7B c. 0BE

10. The image consists of 1024 x 1024 = 1,048,576 pixels and therefore 3 x 1,048,576 = 3,145,728 bytes, or about 3MB. This means that about 86 images could be stored in the 256MB camera storage system. (By comparing this to actual camera storage capacities, students can gain an appreciation for the benefits of image compression techniques. Using them, a typical 256MB storage system can hold as many as 300 images.)

11. 786,432. (Each pixel would require one memory cell.)

12. Data retrieval from main memory is much faster than from disk storage. Also data in main memory can be referenced in byte-sized units rather than in large blocks. On the other hand, disk storage systems have a larger capacity than main memory and the data stored on disk is less volatile than that stored in main memory.

13. There are 70GB of material on the hard-disk drive. Each CD can hold no more than 700MB. Thus, it will require at least 100 CDs to store all the material. That does not seem practical to me. On the other hand, DVDs have capacities of about 4.7GB, meaning that only about 15 DVDs would be required. This may still be impractical, but its a big improvement over CDs. (The real point of this problem is to get students to think about storage capacities in a meaningful way.)

14. There would be about 5,000 characters on the page requiring two bytes for each two byte Unicode character. So the page would require about 10,000 bytes or 10 sectors of size 1024 bytes.

15. The novel would require about 1.4MB using ASCII and about 2.8MB if two byte Unicode characters were used.

16. The latency time of a disk spinning at 3600 revolutions per minute is only 0.00833 seconds.

17. About 11.4 milliseconds.

18. About 7 years!

19. What does it say?

20. hex notation

21. a. D o e s </inst>1 0 0

01000010 01101111 01100101 01110011 00100000 00110001 00110000 00110000

 / 5 = 2

00100000 00101111 00100000 01101010 00100000 00111101 00100000 00110010

 0 ?

00110000 00111111

b.

 T h e t o t a

01010100 01101000 01100101 00100000 01110100 01101111 01110100 01100001

 l c o s t i

01101100 00100000 01100011 01101111 01110011 01110100 00100000 01101001

 s $ 7 . 2 5 .

01110011 00100000 00100100 00110111 00101110 00110010 00110101 00101110

22. a. 42 6F 65 73 20 31 30 30 20 2F 20 35 20 3D 20 32 30 3F

 b. 54 68 65 20 74 6F 74 61 6C 20 63 6F 74 20 69 73 20 24 37 2E 32 35 2E

23. 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111, 10000, 10001, 10010

24. a. 00110010 00110011 b. 10111

25. They are the powers of two. 1 10 100 1000 10000 100000

26. a. 15 b. 1 c. 21 d. 16 e. 19 f. 0 g. 5 h. 9 i. 17 j. 25 k. 26 l. 27

27. a. 111 b. 1011 c. 10000 d. 10001 e. 11111

28. a. 1 b. 5 c. -3 d. -1 e. 15

29. a. 100 b. 111 c. 010 d. 011 e. 110

30. a. 15 b. -12 c. 12 d. -16 e. -10

31. a. 0001101 b. 1110011 c. 1111111 d. 0000000 e. 0010000

32. a. 01101 b. 00000 c. 10000 (incorrect) d. 10001 e. 11110

 f. 10011 (incorrect) g. 11110 h. 01101 i. 10000 (incorrect) j. 11111

33. a.

 5 00101

 + 1 becomes + 00001

 00110 which represents 6

b.

 5 00101 00101

 - 1 becomes - 00001 which converts to + 11111

 00100 which represents 4

c.

 12 01100 01100

 - 5 becomes - 00101 which converts to + 11011

 00111 which represents 7

d.

 8 01000 01000

 - 7 becomes - 00111 which converts to + 11001

 00001 which represents 1

e.

 12 01100

 + 5 becomes + 00101

 10001 which represents -15 (overflow)

f.

 5 00101 00101

 - 11 becomes - 01011 which converts to + 10101

 11010 which represents -6

34. a. 3 3/4 b. 4 5/16 c. 13/16 d. 1 e. 2 1/4

35. a. 101.11 b. 1111.1111 c. 101.011 d. 1.01 e. 110.101

36. a. 1 1/8 b. -1/2 c. -3/16 d. 9/32

37. a. 11111111 b. 01001000 c. 11101111

 d. 00101110 e. 00011111 (truncation)

38. 00111100, 01000110, and 01010011

39. The best approximation of the square root of 2 is 1 3/8 represented as 01011011. The square of this value when represented in floating-point format is 01011111, which is the representation of 1 7/8.

40. The value one-eighth, which would be represented as 00101000.

41. Since the value one-tenth cannot be represented accurately, such recordings would suffer from truncation errors.

42. a. The value is either eleven or negative five.

 b. A value represented in two's complement notation can be changed to excess notation by changing the high-order bit, and vice versa.

43. The value is two; the patterns are excess, floating-point , and two's complement, respectively.

44. b would require too many significant digits. c would require too large of an exponent. d would require too many significant digits.

45. When using binary notation, the largest value that could be represented would change from 15 to 63. When using two's complement notation the largest value that could be represented would change from 7 to 31.

46. 4FFFFF

47. 1123221343435

48. yyxy xx yyxy xyx xx xyx

49. Starting with the first entries, they would be *x*, *y*, space, *xxy*, *yyx*, and *xxyy*.

50. Not a chance. MPEG requires transfer rates of 40 Mbps.

51. a.

 D o e s </inst>1 0 0

001000010 001101111 001100101 101110011 100100000 100110001 000110000 000110000

 / 5 = 2

100100000 100101111 100100000 000110101 100100000 100111101 100100000 100110010

 0 ?

000110000 000111111

b.

 T h e t o t a

101010100 101101000 001100101 100100000 001110100 001101111 001110100 101100001

 l c o s t i

101101100 100100000 001100011 001101111 101110011 001110100 100100000 001101001

 s $ 7 . 2 5 .

101110011 100100000 100100100 100110111 000101110 100110010 000110101 000101110

52. The underlined strings definitely contain errors.

11001 11011 10110 00000 11111 10001 10101 00100 01110

53. The code would have a Hamming distance of 3. Thus, by using it, one could detect up to 2 errors per character and correct up to 1 error per character.

54. a. HE b. FED c. DEAD d. CABBAGE e. CAFE

55. Answers will vary as the exchange rates change daily.

56. Answers will vary depending on the currency selected.

57. This is an interactive exercise, results will depend on the browser.

58. A change to the dollar amount would need to be made in each of the expressions.

59.

no\_B = 123234234

no\_KB = no\_B / 1024

no\_MB = no\_KB / 1024

no\_GB = no\_MB / 1024

no\_TB = no\_TB / 1024

print(str(no\_B) + ' B')

print(str(no\_KB) + ' KB')

print(str(no\_MB) + ' MB')

print(str(no\_GB) + ' GB')

print(str(no\_TB) + ' TB')

print(str(no\_KB) + ' KB')

print(str(no\_KB) + ' KB')

no\_TB = 1.123

no\_GB = no\_TB \* 1024

no\_MB = no\_GB \* 1024

no\_KB = no\_MB \* 1024

no\_B = no\_KB \* 1024

print(str(no\_B) + ' B')

print(str(no\_KB) + ' KB')

print(str(no\_MB) + ' MB')

print(str(no\_GB) + ' GB')

print(str(no\_TB) + ' TB')

print(str(no\_KB) + ' KB')

print(str(no\_KB) + ' KB')

60.

mins = 50

secs = 23

tot\_secs = mins \* 60 + secs

samples\_per\_sec = 44100

bytes\_per\_sample = 2

tot\_bytes = tot\_secs \* samples\_per\_sec \* bytes\_per\_sample

print(tot\_bytes)

61. ‘\*\*’ should be ‘\*’ and ‘PRINT’ should be ‘print’