CHAPTER 1

Notational Systems

Objective 1.1: Compare and contrast notational systems

Computers are designed to store and manipulate data in binary form, but that format isn't often convenient or appropriate for humans or software applications. *Notational systems* provide us with ways to use binary data storage technologies to represent numbers, text, and other data formats.

In this chapter, you'll learn everything you need to know about ITF+ objective 1.1, including the following topics:

- **Binary**
- **Hexadecimal**
- **Decimal**
- **Data representation**

STORING DATA

As we dive into the world of information technology, it's important to understand how computers store and work with data. Let's begin that discussion by talking about the basic units of storage in a computer system.

Binary Data

You've probably heard that computers work with *binary* data, or data that is stored as simply 0s and 1s. Everything that happens inside a computer system uses

combinations of 0s and 1s. From the operating system and software that we run to our Microsoft Word documents or even a video file, everything is encoded in binary format. The reason for this is that computers can easily use this binary format to store data on disk, keep it in memory, or send it over a network.

The basic unit of binary storage in any computer system is the *bit*. A bit is a single binary digit that can be either 1 or 0. Those are the only two possible values for a bit. You can't put the number 2 or the letter Z in a bit. It can only be a 1 or a 0, as shown in Figure 1.1.

FIGURE 1.1 A single bit can only hold two values: 0 and 1.

When we store data on a magnetic hard drive, the computer divides the drive up into billions of tiny little spaces, each designed to store a single bit. If the value of the bit is 1, the computer places a magnetic charge in the location used by that bit. If the value of the bit is 0, the computer leaves no magnetic charge in that location.

When data is stored on a *solid-state drive* (*SSD*) or in memory, the process works the same way but using electricity instead of magnetism. If the value of a bit in memory is 1, a small electrical charge changes the value in that memory location to the "on" position. If the value of the bit is 0, the value in that position is set to the "off" position.

Now, computers may think in 0s and 1s, but that's not the way that we human beings think. We'd much prefer to think of our data in terms of letters and numbers! Computers store the data that we're more familiar with by combining multiple bits together.

If we have 2 bits of data, we can use them together to represent four different values, as shown in Figure 1.2.

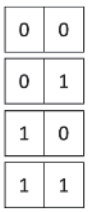

FIGURE 1.2 Two bits can hold four possible values.

We could use the 2-bit values in Figure 1.2 to store whole numbers between 0 and 3. We simply assign each of the 2-bit binary possibilities a whole-number equivalent. Table 1.1 shows the standard conversion for these 2-bit values.

TABLE 1.1 Decimal conversion of 2-bit values

Binary Value	Decimal Value
00	0
01	1
10	2
11	3

If we have 3 bits of data, we can use them to store eight possible values: 000, 001, 010, 011, 100, 101, 110, and 111. These convert to decimal values ranging from 0 to 7, as shown in Table 1.2.

TABLE 1.2 Decimal conversion of 3-bit values

Binary Value	Decimal Value
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Similarly, if we have 4 bits of data, we can store 16 possible values, and as we increase the number of bits, we increase the number of values. Once we get up to 8 bits, we find ourselves able to store 256 possible values, ranging from 0 to 255. That's an important number because we can store all of the possible characters and digits used by a computer system in this range.

This combination of 8 bits is known as a *byte*, and the byte is the second important unit of binary data storage. When you're thinking about text data, you can think of each character as being a single byte, made up of 8 bits.

Decimal Data

Whether or not you know it, you're already familiar with *decimal notation*. This is the numbering system that we use in our everyday lives, and it's based off multiples of the number 10.

In a decimal number, each digit can take on 10 possible values, ranging from 0 to 9, as shown in Figure 1.3.

FIGURE 1.3 One decimal digit can hold 10 possible values.

When we grow to two decimal digits, we can represent one hundred values, ranging from 0 to 99. Adding a third digit allows us to store numbers between 0 and 999. Every time that we add another digit, we increase the number of values we can store by 10.

Hexadecimal Data

Unless you've worked with computer memory before, you're probably not familiar with *hexadecimal notation*. In this notation, each value can store 16 possible values, ranging from 0 to 15. Now you're probably wondering how we can put a two-digit number like 10 or 15 into a single location. That's a good question!

We do this by using the values 0 through 9 to represent the numbers 0 through 9 but then using the values A through F to represent 10 through 15. Figure 1.4 shows the 16 possible values that may be stored in a single hexadecimal digit.

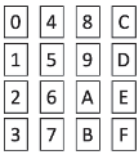

FIGURE 1.4 A single hexadecimal digit can hold 16 possible values.

Table 1.3 provides some examples of the same numbers expressed in decimal, binary, and hexadecimal forms.

TABLE 1.3 Binary, decimal, and hexadecimal equivalent values

Decimal Value	Binary Value	Hexadecimal Value
0	0	0
5	101	5
10	1010	A
123	1111011	7B

The math here gets a little complicated, but the good thing is that you won't be asked to convert these values on the exam. What you want to understand is that if you see a value consisting of 0s and 1s, that's binary. If you see values made up of the digits 0 through 9, that's decimal. And if you see the letters A through F mixed in with those digits, that's hexadecimal.

EXAM TIP

Expect to see exam questions that ask you to identify the best notational system or data representation for a given situation. If the question mentions anything about non-English characters, you'll probably want to use a Unicode data representation. If the question asks about values that can be encoded as 0 or 1, true or false, off or on, yes or no, or similar two-value options, that's a key indicator that binary data storage is appropriate.

CHARACTER ENCODING

We've discussed three notation systems: binary, decimal, and hexadecimal. Those are the different ways that we can represent numbers. But we often want to store and process text values instead of numbers when we're working with data. The way that we do this is to encode text characters as numbers.

You may remember when we first discussed binary data, I mentioned that computers typically work in units of bytes and that each byte consists of 8 bits. Bytes can store decimal numbers from 0 to 255, and we use 1 byte to store one character of text. We do this by using standard codes to describe how we encode each character as a number.

When we're using English or another language that uses our alphabet, we use a code called the American Standard Code for Information Interchange, or *ASCII*. This code describes what numeric value to use for each of the uppercase and lowercase letters, digits, punctuation, and other symbols commonly used in the English language. ASCII was originally designed as a 7-bit code, but modern computers use an extended version of ASCII that uses 8-bit bytes.

If you're working in languages other than English, you need to have more characters available to you. This requires the use of a different code. *Unicode* is a large character set capable of representing thousands of different characters using 8 or 16 bits of data.

CERTMIKE EXAM ESSENTIALS

- Binary data is the native format used by computer systems and is used to store values that can be represented as either 0 or 1. Decimal values are the base-10 numbers that we use in our everyday lives that use digits between 0 and 9. Hexadecimal values extend the number of possible values in a single digit to 16 by adding the values A–F as possibilities.
- ASCII data representations are used to store the characters of standard English text in binary form. Unicode data representations can store English characters as well as characters used in other languages.

Practice Question 1

A developer is working on a new software program that will store data in memory about many different characteristics of customers of a bank. Each of these characteristics is best represented as a "yes" or "no" value.

What notational system would *best* store this type of data?

A. Hexadecimal
B. Decimal
C. ASCII
D. Binary

Practice Question 2

Your company recently entered into a partnership with an organization based in Egypt and you are helping an executive receive documents that must be translated from Arabic into English. The documents contain Arabic characters, but those characters are not rendering properly on the screen.

What representational system is *best* used for this type of document?

A. ASCII
B. Binary
C. Unicode
D. Hexadecimal

Practice Question 1 Explanation

This question is asking us to identify the notational system that would *best* meet the described need. This is a very common format for CompTIA exam questions, and you should be prepared to evaluate all of the possible answer choices and find the one that is better than all the others.

Let's evaluate these choices one at a time:

1. First, we have the possibility of using hexadecimal values. It would indeed be possible to store "yes" and "no" as hexadecimal values by using the hexadecimal value of 0 to represent "no" and 1 to represent "yes." However, this is wasteful because a single hexadecimal digit can be used to store up to 15 possible values and we only need the ability to store two possible values. So this isn't a great option.
2. Next, we are presented with the choice of storing the values in decimal form. As with hexadecimal, we could encode "no" as the decimal value 0 and "yes" as the decimal value 1, but that would be wasteful, so it is not the best option.
3. We could use ASCII to store the text strings "yes" and "no" as well, but this requires us to use 3 bytes of storage to store a three-letter word. Again, this is a possibility, but it is not the best option.
4. The last option, binary, is the best one here. We can encode "yes" as the binary value 1 and "no" as the binary value 0 and use our data storage efficiently.

Correct Answer: D. Binary

Practice Question 2 Explanation

In this question, you're being asked to identify the best representational system to use when storing text data in a file. That allows us to narrow down our answer choices quickly. ASCII and Unicode are representational systems for text, whereas binary and hexadecimal are notational systems for storing data. We can, therefore, quickly eliminate binary and hexadecimal as answer choices.

When you take the ITF+ exam, watch for opportunities like this where you can immediately eliminate answer choices that are obviously wrong. This can help you focus your attention quickly and increase the odds that you will pick the correct answer.

Next, we must decide among the two remaining answers. ASCII only allows the storage of English characters, so it won't work very well in this scenario. We can, however, store the Arabic characters using a Unicode data representation, making Unicode our best option.

Remember, the use of non-English characters is a key indicator that Unicode is an appropriate choice!

Correct Answer: C. Unicode