Binary and Digital Communication

Analog Signals

Binary is the language of our digital age and few of us, even those with computer science experience, rarely see it in our daily interaction with technology. Before the age of the computer in the early 20th century, radio and other analog forms of communication were developing at a rapid pace. Telephones and radios transmit audio by correlating the intensity of a sound wave to a voltage. Simply put, these changes in voltage form an **analog signal**, which can be picked up by a receiver and replicated as audio. An analog signal is any continuous signal with a time varying quantity (frequency, amplitude, etc).

Analog signals have many advantages and are still core to many technologies that involve wireless communication. They also have some serious downsides. If you have ever heard the crackling sound of your car radio or the fuzziness of your television, you will have already experienced the issues that arise with analog signal interference, electric noise, and distortion. These signal errors are acceptable when listening to a talk show, but become problematic if calculating the telemetry of the rocket midflight and the numbers become garbled. Houston, we've got a problem!

Binary

Digital signals answer this problem quite elegantly. A digital signal is a series of quantized or discrete values. In more human language, digital signals can only have **two states**, on or off. These states are represented by the numbers `1` (on) and `0` (off). **Binary** is a way in which these digital signals can be transformed into intelligible data. Binary is a base-2 numerical system, meaning that each digit represents a multiple of two. The numbers that we normally use (base-10) have a one's place, a ten's place and so on. Now imagine if each of those placeholders could only take a `1` or a `0`, and each digit represents an ascending multiple of two.

Take the number 27, for example. In our traditional number system, we think of 27 as having three "ones" and two "tens":

```
2 x 10 + 7 x 1 = 20 + 7 = 27
```

Now, let us try concocting this number in binary. In binary, we think of 27 being composed of one "sixteen", one "eight", one "two", and one "one". Written out in binary, that is 00011011. Getting there takes a couple steps.

```
00011011 = 1 x 16 + 1 x 8 + 1 x 2 + 1 x 1 = 16 + 8 + 2 +
 1 = 27
```

Now hopefully most of that makes sense, but what the heck are those zeroes doing there? Well, those zeroes are placeholders. Just like in our base-10 number system, binary number representation use zeroes to hold the place of multiples of its base (in the case of binary, this is 2). The digit values of binary start from 1 and progress onwards by multiples of 2.

```
128 64 32 16 8 4 2 1       # first eight digit values
 0  0  0  1  1 0 1 1       # binary representation of 27
```

Using the above table as a guide, multiply the binary value (`1` or `0`) by the base-2 digit value above it and sum the total to yield the binary number in base-10 (normal numbers). Try some of these examples to practice converting to binary, and back again.

Practice Set 1

Number (base-10)	Binary (base-2)
8	?
21	?
55	?
256	?
?	00001101
?	01011001
?	10001101
?	01110111

Bits and Bytes

All of you know or have heard about the concept of digital storage, whether that is the number of photos you can fit on your phone or the size of your computer's hard drive. You have also probably heard about a couple ways to describe storage, such as Gigabyte (GB) and Megabyte (MB). These terms describe the size of digital data and for storage devices, indicate the amount of data they can hold. Digital data, as you saw previously, is encoded in the `1` s and `0` s of binary code. In terms of this kind of data, we refer to each `1` or `0` as a **bit**. Eight sequential bits make up what we call a **Byte**. Now let's do some simple math to deteremine how many unique values are in one byte. First, we know that each bit represents two values, `0` and `1`. So if we have eight of these bits in a row, there are many permutations that arise:

```
00000001
00000010
00000011
00000100
00000110
00000111
    etc.
```

So to calculate how many different values there are, we multiply the number of possible values of each digit. In this case there are 2 possible values for each of 8 digits, so it follows that the number of unique values of one Byte is

```
2 x 2 x 2 x 2 x 2 x 2 x 2 x 2 = 2^8 = 256
```

Computers use these 256 different possible values to represent letters,

Computers use these 256 different possible values to represent letters, numbers and other symbols. Start combining Bytes of different values and you start making new and more complex types of data! Your favorite song is made up of many thousands of bytes of data. At that size, it doesn't make much sense to describe the size of the data in Bytes and is usually expressed in kilobytes (kB), megabytes (MB) or gigabytes (GB). Once again, remembering that binary is base-2, note that each of these multipliers is of *magnitude* 10 greater than the previous and not ten *times* greater. For example, a kilobyte is 2^10 times greater than one Byte. Therefore one kilobyte is 1024 Bytes, one megabyte is 1024 kilobytes, and so on. Notice that, by convention, Byte is always written in uppercase and bit is always lowercase. Thus, if one is talking about a gigabyte of storage, the abbreviation is written as GB, whereas the same for gigabit ethernet connection would be written as Gb.

Hexadecimal

Hexadecimal encoding is a technique that allows an 8-digit binary to be written with two different **"hex"** digits. Each "hex" digit represents four bits (called a **nibble**). Each nibble (four bits) has 16 unique permutations which are represented by the numbers 0 through 9 and the letters A through F. Therefore a pair of nibbles represents one Byte. Take the binary number `10011110` for example. To convert this number to hex, first split the number into two, four-digit nibbles.

```
1100 and 1110
```

Now find the base-10 value of each nibble

```
1001 = 1 x 8 + 0 x 4 + 0 x 2 + 1 x 1 = 9

1110 = 1 x 8 + 1 x 4 + 1 x 2 + 0 x 1 = 14
```

The first nibble is straightforward, as numbers 0 through 9 are represented in hex. The second nibble, base-10 value 14, is represented in hex with the letter E. If you are confused at this point, just think of the letters A through F as the numbers 10 through 15. Note that the number 16 is not included, as the number 0 is part of the set and yields 16 total values. So the value of `10011110` in hexadecimal is 9E.

Practice Set 2

Number (base-10)	Binary (base-2)	Hexadecimal (base-16)
145	10010001	?
89	01011001	?
?	?	6F
?	?	D8
?	01001101	?
?	01011011	?

Resources

External Links

- binary practice (easy)
- binary practice (medium)
- binary practice (hard)

Answer Keys

Practice Set #1

Number (base-10)	Binary (base-2)
8	00001000
21	00010101
55	00110111
256	11111111
13	00001101
89	01011001
141	10001101
119	01110111

Practice Set #2

Number (base-10)	Binary (base-2)	Hexadecimal (base-16)

145	10010001	91
89	01011001	?
191	01101111	6F
216	11011000	D8
77	01001101	4D
91	01011011	5B