Bits and Bytes

Computers are used to store and process data. Processed data is called information. You are used to see or hear information processed with the help of a computer: a paper you've just typed using Microsoft World, a digital photograph, a MP3 song you've downloaded on the Internet, etc. But how is the data stored in the computer?

Whatever the medium in which it is stored, the most fundamental piece of data is always represented by a "two-value system": low or high voltage, positive or negative polarity of a magnetic medium, absence or presence of "bumps" on a CD or DVD… These two values are logically represented by the digits 0 and 1. This is called a binary representation. Usually, the digit 0 is interpreted as the low value or absence of some kind of signal (such as no bump) and the digit 1 is interpreted as the high value of presence of some kind of signal. In other word, we encode the physical signal with the logical binary representation.

This 0 or 1 digit is called a **bit** (short for **bi**nary digi**t**). Obviously, just one bit cannot give much information since there are only two possible values. Hence, to store more useful data we must combine bits. However, even combining several (even many) bits will only give a finite numbers of possible objects. For example combining four bits gives the sequences 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111. These sixteen sequences can be selected to represent any sixteen objects of our choice but *only* sixteen objects. So if we want to represent integers, we can decide they represent the integers 0 to 15; if we represent letters they could represent the letters a, b, c, d, e, f, g, h, i, j, k, l, m, and n (or any other combination of sixteen letters of our choice); if we represent colors they could represent 16 predefined colors of our choice. We say that the data stored on a computer, and represented logically by combinations of bits, is **discrete**: it is finite and made of separable units.

Analog versus digital

Many different kinds of data can be represented in a computer:

- numbers: whole numbers, positive and negative integers, real numbers;
- text: made of letters, characters, digits;
- audio: speech, music, sound tracks;
- images and graphics;
- video, etc.

Some of these data types, such as text or integers on a bounded range, are discrete and it will be rather straightforward to convert these data to a binary representation that can be stored in the computer. But most data is analog, which means that it is continuous in nature. Let's look at some examples.

Pictures → Continuous in space
→ Infinite number of colors

Sound → Continuous in time
→ Infinite number of frequencies

Real numbers ⟶ Continuous even if bounded: always a number between two given numbers

Because most data is analog, it must be broken in discrete pieces and those pieces must be represented by a binary representation to be stored in the computer. When this process is performed, we say that we **digitize** the data. Note that because digitizing data transforms continuous infinite data into finite discrete data, some data will be lost. For example we cannot really represent all the colors of the color spectrum, we cannot represent all the real numbers between 0 and 1.
In this unit of the course, we will study how some types of data are represented and discuss some of the issues that arise from the conversion from analog to digital data. We will also use some tools to manipulate data.

Binary Numbers

Because the representation of more complex data such as sound or color is based on the representation of quantities (i.e. numbers), we must start our study with the representation of whole numbers.
As seen above the sixteen first whole numbers (0 to 15) can be represented by all possible four-bit sequences. We will generalize this to longer bit sequences, but we will show how to derive a meaningful way to convert from a decimal number (the "regular" number) to a binary number and vice versa.
The number system we use is the number system in base 10.

Base 10 number system

We use 10 digits, 0 to 9. In a number, the value of each digit depends of its position in the number. Each position corresponds to a value that is a power of 10. Table 1 shows the decomposition of the number 2324 in the number system of base 10 (also called decimal number system).

2	3	2	4
$2 \cdot 10^3$	$3 \cdot 10^2$	$2 \cdot 10^1$	$4 \cdot 10^0$
thousands	hundreds	tens	units

Table 1

This can also be written as $2324 = 2 \cdot 10^3 + 3 \cdot 10^2 + 2 \cdot 10^1 + 4 \cdot 10^0$.

Base 2 number system

The binary number system is the number system in base 2. We use 2 digits, 0 and 1, and each position in a number corresponds to a power of 2. Table 2 shows the decomposition of the binary number 1110 in the number system of base 2.

1	1	1	0
$1 \cdot 2^3$	$1 \cdot 2^2$	$1 \cdot 2^1$	$0 \cdot 2^0$
eights	fours	twos	units

Table 2

Therefore the value of the binary number 1110 is $1 \cdot 2^3 + 1 \cdot 2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 8 + 4 + 2 = 14$.
When necessary, to avoid confusion we will append the subscript 2 to binary numbers. So the equality $1110_2 = 14$ means that the number with binary representation 1110 is the same as the number with decimal representation 14 (we obviously cannot write 1110 = 14!).

We can check that the 16 sequences given in page 1 do match the decimal numbers 0 to 15 according to our definition.

Binary Number	Decomposition in sum of powers of 2's	Decimal Value
0000	$0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = 0 + 0 + 0 + 0$	0
0001	$0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 0 + 0 + 0 + 1$	1
0010	$0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 0 + 0 + 2 + 0$	2
0011	$0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 0 + 0 + 2 + 1$	3
0100	$0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = 0 + 4 + 0 + 0$	4
0101	$0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 0 + 4 + 0 + 1$	5
0110	$0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 0 + 4 + 2 + 0$	6
0111	$0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 0 + 4 + 2 + 1$	7
1000	$1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = 8 + 0 + 0 + 0$	8
1001	$1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 8 + 0 + 0 + 1$	9
1010	$1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 8 + 0 + 2 + 0$	10
1011	$1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 8 + 0 + 2 + 1$	11
1100	$1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = 8 + 4 + 0 + 0$	12
1101	$1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 8 + 4 + 0 + 1$	13
1110	$1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 8 + 4 + 2 + 0$	14
1111	$1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 8 + 4 + 2 + 1$	15

Table 3

We can notice the pattern: $1_2 = 2^0 = 1$, $10_2 = 2^1 = 2$, $100_2 = 2^2 = 4$, $1000_2 = 2^3 = 8$. In general, $100 \cdots 0_2 = 2^n$, where n is the number of zero following the 1. This is, of course, consistent with the definition of the binary number: there are n + 1 digits; each digit in this binary number corresponds to a power of 2, from 2^n for the left-most digit to 2^0 for the right-most digit; all powers of 2's except the left most (2^n) are multiplied by 0.

Recreation: Magic cards[1]

A magician has four magic cards; each card holds eight numbers between 1 and 15.

1	3	5
7	9	11
13	15	

Blue

2	3	6
7	10	11
14	15	

Green

4	5	6
7	12	13
14	15	

Red

8	9	10
11	12	13
14	15	

Yellow

She instructs each member of the audience to choose a number between 1 and 15 and hands out the cards to one person at random. The selected person returns only the cards that contain the number she/he had chosen (and kept secret). The magician guesses the number chosen. Suppose for example the person hands out the blue, green and yellow cards. Then the magician says "Eleven"! And she is right!

How does this work?

The cards are based on the binary representation of the numbers 1 to 15. The blue card contains all the numbers that have a 1 as right-most bit (i.e., the bit that corresponds to the units). The green card

[1] Adapted from Aha! by Martin Gardner (1)

contains all the numbers that have a 1 as second digit from the right (i.e., the bit that corresponds to 2^1). The red card contains all the numbers that have a 1 as the second digit from the left (i.e., the bit that corresponds to $2^2 = 4$). And finally the yellow card contains all the numbers that have a 1 as left-most digit (i.e., the bit that corresponds to $2^3 = 8$). So, in our example, when the magician is given the blue, green and yellow cards, he knows that the number is 1011 in binary and therefore 8 + 2 + 1 = 11 in decimal. Note that the magician does not have to know binary numbers; she just needs a clever aide who made the cards for her! She just has to add the number that is at the top left corner of the cards that the member of the audience gave her.

Exercises: all about patterns…

1. Table 3 shows clearly that the larger the number the more bits are needed to express a number in its binary representation. For example, 6 can be expressed with only 3 bits: 110_2 but 12 cannot since 12 is 1100_2. It is permissible to add or remove leading zeros (i.e., zeros with no 1's to their left), but it is not permissible to remove any other digits from the binary representation without altering its value. This is similar as 01060 being the same as 1060 in the decimal number system, but certainly different from 0106 or 0160.
   With one bit we can express just two numbers 0 and 1.
   a) Observe table 3. How many numbers can be represented with 2 bits? with 3 bits? with 4 bits?
   b) You should get a pattern in question a). From it, can you deduce the number of numbers that can be represented with 5 bits? with 8 bits? with n bits?

2. Observe table 3. Explain how you can distinguish between even and odd numbers if you are given just the binary representation.

3. If you observe the Binary Number column of table 3 you can imagine that this column is split into 4 columns: the column of the first digit (corresponds to the eights), the columns of the second digit (column of the fours), etc. Explain the pattern of 0 and 1 in each of these four columns, starting from the right-most column to the left-most column.

4. To write all the binary numbers from 0 to 31, you need to add one more column(to the left) to the imaginary columns that make up the Binary Number column of table 3 (see question 3) and continue the pattern you described in question 3. Create the table, similar to table 3, that shows all binary numbers from 0 to 31.

5. Create magic cards with numbers from 0 to 31. Practice the magic trick with those cards.

6. Find the decimal values of the following binary numbers. (Note: to make them more readable we often write the digits of a binary numbers by groups of four.)
   a) `1000 0000`
   b) `1111 1111`
   c) `0010 1110`

d) `1011 0000`

From decimal to binary

We know how to convert a binary number into a decimal number; but how do we do the reverse?
From what we have already learned, some numbers are easy to convert:

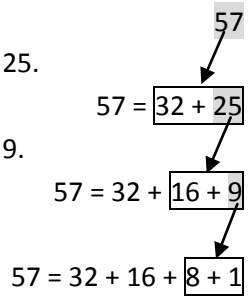
1 is of course 1_2
$2 = 2^1$ so 2 is 10_2
$4 = 2^2$ so 4 is 100_2
$16 = 2^4$ so 16 is $1\,0000_2$
In general, $2^n = 100\cdots0_2$, where 1 is followed by n zeros.

For other positive integers we will use the fact that if we can write them as a sum of powers of 2's then we can write their binary representations. For example, since 57 = 32 + 16 + 8 + 1 we can write $57 = 11\,1001_2$. But how do we find that 57 = 32 + 16 + 8 + 1?
First, let us review the powers of 2's: 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024 … (How do you go from one power of 2 to the next one?[2])
Now let's start with 57.

Find the largest power of 2 no larger than 57: it is 32. Subtract 32 from 57: 57 – 32 = 25.

Find the largest power of 2 no larger than 25: it is 16. Subtract 16 from 25: 25 – 16 = 9.

Find the largest power of 2 no larger than 9: it is 8. Subtract 8 from 9: 9 – 8 = 1.



When we reach 0 or 1, we stop.

Repeated subtraction algorithm

Input: A non-negative integer (for practical purpose no greater than 1023)
Output: The binary representation of the input
Step 1: Let A be the input. Find the largest power of 2 no greater than A, say 2^n.
Step 2: Draw a table with 2 rows and n + 1 columns. The cells in the first row should contains the powers of 2's from 2^n (found in step 1) to $2^0 = 1$, written from left to right. Place a 1 in the left most column in row 2.
Step 3: Subtract 2^n from A and make this value the new value of A.
Step 4: repeat steps 4.a and 4.b until A is equal to 0 or 1

Step 4.a: Find the largest power of 2 no greater than A. Place a 1 in row 2 in the column that contains this power of 2.

Step 4.b: Subtract the power of 2 found in the previous step from A and make this value the new value of A.

[2] Multiply by 2!

Step 5: Place one zero in each empty cell of row 2 in the table.
Step 6: Write row 2 of the table as one number. It is the binary number representation of the input.
Let see how it works on the number 108.
Step 1: A = 108. The largest power of 2 no greater than 108 is 64 = 2^6.
Step 2:

64	32	16	8	4	2	1
1						

Step 3: A becomes 108 – 64 = 44.
Step 4:

Step 4.a: 32 is the largest power of 2 no greater than 44.

64	32	16	8	4	2	1
1	1					

Step 4.b: A becomes 44 – 32 = 12

Step 4.a: 8 is the largest power of 2 no greater than 12.

64	32	16	8	4	2	1
1	1		1			

Step 4.b: A becomes 12 – 8 = 4

Step 4.a: 4 is the largest power of 2 no greater than 4.

64	32	16	8	4	2	1
1	1		1	1		

Step 4.b: A becomes 4 – 4 = 0. Since A is equal to 0 we exit step 4.

Step 5:

64	32	16	8	4	2	1
1	1	0	1	1	0	0

Step 6: 108 = 110 1100_2
Check: 64 + 32 + 8 + 4 = 108.

Exercises

7. Find the binary representation of the following numbers.
   a) 64
   b) 65
   c) 43
   d) 213
8. The standard unit of computer memory is 8 bits, called a **byte**. How many nonnegative integers can be represented with 8 bits and what is the largest integer that can be represented with 8 bits(i.e. a byte)?

Hexadecimal numbers

Binary numbers are easy to handle for the computer because they are made up with only two digits, but they have two big disadvantages for humans:

- They get quickly very long. For example 256, a 3-digit numbers in our regular decimal number system is 1 0000 0000_2, a 9-digit number, in the binary system.

- They are difficult to read which can result in errors.

In many applications we use the hexadecimal number system to express the numbers. It is the number system in base 16. It will share advantages of both the binary system and the decimal system:

- The numbers have fewer digits than decimal numbers and are easy to read.
- The process to convert between binary and hexadecimal numbers is very simple and efficient, so computers can handle hexadecimal numbers without problems (they are in fact just a "shorthand" notation for binary numbers).

Since hexadecimal numbers are numbers in base 16, it means we have 16 digits. Since in the Arabic notation we have only 10 distinct digits (0 to 9) we add letters to make up 16:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

The letter A has the value 10, the letter B the value 11, C is 12, D is 13, E is 14 and F is 15. As for the decimal and binary systems, the position of a digit in a number indicates its value, which now will be a power of 16, the base. For example the number $3B9_{16} = 3 \cdot 16^2 + 11 \cdot 16^1 + 9 \cdot 16^0 = 953$.

Converting from Binary to Hexadecimal

Table 4 shows the 16 hexadecimal digits, their decimal values and their binary representations. Notice that the 16 numbers corresponds to exactly all the numbers that can be represented with 4 bits.

Hexadecimal Digit	Decimal Value	Binary Number
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Table 4

Take a binary number, say `10010011001100`. Count the number of digits (here 14) and add enough zero to the LEFT of the number to make the number of digits a multiple of 4 (add 2 digits here).
We rewrite the number as `0010 0100 1100 1100`, separating each group of 4 bits. Match each group of 4 bits with the corresponding hexadecimal digit from table 4: `0010 0100 1100 1100`

2 4 C C

The result is $10010011001100_2 = 24CC_{16}$.

To convince us that this work, we can convert each of the two numbers to its decimal equivalent.

$$10010011001100_2 = 2^{13} + 2^{10} + 2^7 + 2^6 + 2^3 + 2^2 = 9420$$

$$24CC_{16} = 2 \cdot 16^3 + 4 \cdot 16^2 + 12 \cdot 16^1 + + 12 \cdot 16^0 + 2^3 + 2^2 = 9420$$

Converting from Hexadecimal to Binary

To convert a hexadecimal number to binary, we complete the reverse process. We replace each hexadecimal digit in the number by its 4-bit binary representation. For example the number $A39_{16}$ will be $1010\ 0011\ 1001_2$. We usually leave the space between the 4-bit groups to make the number more legible.

Caution

You can apply the methods described above only between the binary and the hexadecimal number systems. To convert between the decimal number system and the binary numbers system you must apply the methods describe earlier.

The conversions between binary and hexadecimal numbers are easier because the base 16 is a power of base 2. Hence the 16 hexadecimal digits are all digits that can be represented with exactly 4 bits. Such a relationship does not exist between the base 2 and 10.

Exercises

9. Convert the following hexadecimal numbers to decimals.
   a) $3A_{16}$
   b) 612_{16}
   c) $FEB2_{16}$
10. Convert the following binary numbers to hexadecimals.
   a) 101100101
   b) 10001001111
11. Convert the hexadecimal numbers from question 9 . to binary.

Representing Graphics

Representing images and graphics on the computer is much more challenging than representing nonnegative integers because images and graphics are analog. Graphics can involve an infinite number of colors and a 2D-image is continuous rather than discrete. The most common way to digitize an image is to use raster graphics. In this process, the rectangle that includes the image is divided into a grid where each cell of the grid is called a pixel. Each pixel is assigned a color. If the grid has few rows and columns then the picture would be blurry as shown on Picture 1 below. The same picture (as seen in picture 2) with more rows and columns give a sharp picture. Increasing the number of pixel improves the **resolution**.

Picture 1

Picture 2

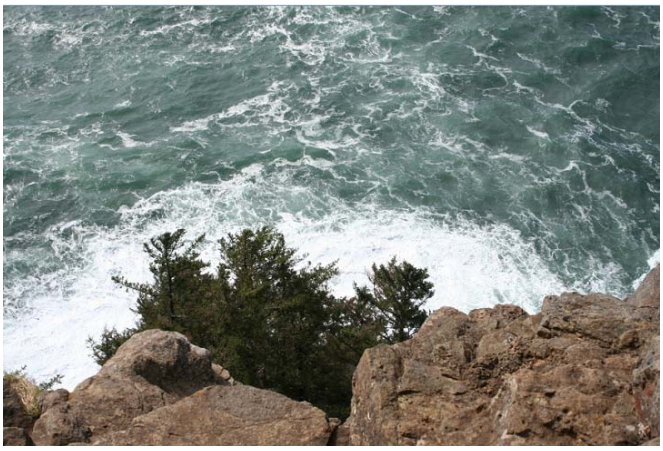
The grid deals with the problem of the continuous space; remains the problem of representing an infinite number of colors. Different models can be used to encode colors. The most used and the one we will study is the RGB model. This model is based on how we perceive color. Our retinas have receptors that respond to the light frequencies corresponding to red, green and blue. Mixing those three primary colors and varying the intensity of each of the primary color create all the other colors. Black is an absence of all three colors, white is a combination of all three colors with maximum intensity, and the varying shades of gray are mixtures of all three primary color with the same intensity.
In the RGB model, a color will be represented by three numbers: the first one indicates the amount of red, the second one the amount of green and the third one the amount of blue in the color. The amount for each color is measured on a scale from 0 to 255 where 0 indicates an absence of the color and 255 indicate the color present at full intensity.

Number of colors represented by the RGB model

You should immediately notice that the RGB model does not represent an infinite number of colors because the number 255 is finite! But if we use all 256 possible values for each of the colors we can represent 256^3 = 16,777,216 colors…
Now where does the number 255 come from? As we have seen in exercise 8 page 6, 256 nonnegative integers can be represented with a byte: the integers 0 to 255. So reserving one byte to store each color value allows the values 0 to 255 for each color. This 24-bits representation of a color is called TrueColor. The number of bits used to express the color is called the color depth. The larger the color depth the more color can be represented. HiColor is a color depth of only 15 or 16 bits. There are now color depths greater than True Color such as 30 or 32-bit colors.

An issue of size

Have you ever downloaded a website and waited, waited, waited… because a picture took forever to load. Do not be too quick to blame your computer or your internet connection. Let us do some computations.
Let's look at picture 3 below.



Picture 3

The size of this picture is 3456 × 2304, which means the grid has 3456 columns and 2304 rows. Hence there are 7,962,624 pixels (i.e., almost 8 millions pixels). If the color for each pixel is stored as a 3-byte data, the amount of data stored is 7,962,624 × 3 = 23,887,872 bytes or almost 24 Mega Bytes (MB). (A megabyte is a million bytes.) This is very large: you could not send a file of that size as an attachment with Hotmail or Gmail for example. Hotmail has a 10 MB limit on email message (attachment included) and Gmail has a 20 MB limit. However, the file size for the picture would be about 24 MB only if the 24-bit TrueColor information of all the pixels were actually saved. This is done in a bitmap file (extension bmp). Most pictures are saved in the JPEG format (file extension jpg and more rarely jpeg). This format applies an algorithm that compresses the data resulting in a smaller file size. The compression is said to be lossy because some data is lost in the process and cannot be recovered. Different amount of compression can be applied resulting in images of varying quality: the higher the compression, the lower the quality. The compression technique used to create a JPEG file is based on averaging colors of neighboring pixels. It works well with image with complex colors and/or subtle color variations. It does not work as well with graphics with very few colors and sharp color contrast such as in line drawings.

You will learn more about graphic and pictures in the hands-on lab activities.

Back to numbers…

Binary arithmetic

We will only talk about adding binary numbers. Good news! There are very few addition facts to learn. Here they are.

+	**0**	**1**
0	0	1
1	1	10

Let us apply the addition facts to the addition of multiple digits numbers. First without carry:

```
 110010          110011
+  1001         +1000100
 111011          1110111
```

Now with some carries:

```
1 1 ←  Carry →   11
 101              1101
+101             +1110
1010             11011     Note that 1 + 1 + 1 = 11
```

Exercises

12. Perform the following additions.
    a) `1001 + 110`
    b) `110010 + 1100`
13. Perform the following additions.
    a) `11001 + 1011`
    b) `10110 + 1100`
    c) `1111 + 101011`

Representing Integers

We know how to represent non-negative integers: we use their binary representation. The only limitation is that we always will limit the number of bits available for representing an integer, which in turn limit the number of positive integers that can be represented. We have seen that with n bits we can represent 2^n integers. To represent negative as well as positive integers, we must be able to represent the sign (+ or -).

Assume that we consider 8-bit integers. We can let the first bit represent the sign and the remaining 7 bits represent the number magnitude. The convention is usually to assign + to 0 and – to 1. With this pattern `0001 1001` would represent the number 16 + 8 + 1 = 25, while `1001 1001` would represent -25. But what would `0000 0000` and `1000 0000` represent? For both these numbers the magnitude would be 0, therefore both represent 0. Because having two representations for 0 in a computer causes complications, an alternative way of representing signed integers is used.

Two's complements

Signed integers are represented with a fixed numbers of bits, 8 (rarely), 16 or 32 most often. For simplicity we will describe the technique first with only 4 bits. As described above the first bit will

indicate the sign of the integer, 0 for a positive integer and 1 for a negative integer. In the case of a positive integer, then the 3 remaining digits will give the magnitude of the integer. The table below show how integers 0 to 7 are represented with this scheme.

0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7

We now have 8 possible values for negative numbers from 1000 to 1111, so we will represent numbers -1 to -8. The "trick" here is to restart the count at -8 for 1000 and count forward.

1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1

Why the name two's complement?
Add the 2's complement representations of 1 and -1: `0001 + 1111 = 1 0000`
Add the 2's complement representations of 2 and -2: `0010 + 1110 = 1 0000`
Add the 2's complement representations of 3 and -3: `0011 + 1101 = 1 0000`
I guess you get the pattern! If n is positive between 1 and 7, the 2's complement representation of –n is the binary number that need to be added to the binary representation of n in order to "complete it to `1 0000`."
This also gives us a direct way to compute directly the 2's complement of negative integer. Let illustrate it with -5.
Step 1: find the **4-bit** binary representation of the absolute value of the number.
Here find the binary representation 5: it is `0101`. Note: it is extremely important to include the leading zeros.
Step 2: Switch all 0's to 1's and all 1's to 0's in the binary number found in step 1.
Here: `0101` → `1010`
Step 3: Add one to the binary number found in step 2.
Here: `1010 + 1 = 1011`. Note that this is the number that the table gave us.

Two's complement of length four bits were used because it is easy to list them all, but they do not have much practical application. In the computer signed integers are usually represented with 16, 32, or 64

bits two's complements. With 16 bits (or 2 bytes) we represent the integers -32,768 to 32,767 and with 32 bits the integers -2,147,483,648 to 2,147,483,647.

Representing real numbers

We have seen how to represent nonnegative integers using their binary representation, but how do we store numbers such as 145.23 or 3.14 or $\sqrt{2}$ in the computer? Before exploring this topic let us review a topic you studied in math: scientific notation.

Scientific notation

In the course of computations on a scientific calculator you may encounter the following results: 3.023434134 and 3.023434134E+14. If you do not pay attention you may quickly round both number to 3.023 if you are working to the closest thousandth. However you would be making a big mistake! The second number is 3.023434134E+14 = $3.023434134 \cdot 10^{14}$ = 302,343,413,400,000. When the calculator encounters numbers that are very large or very small it writes them in *scientific notation*.

A number written in scientific notation has three parts: the sign (+ or -), the mantissa which is a decimal number with only one digit between 1 and 9 (inclusive) to the left of the decimal point, and a exponent. In a calculator and in many computers output, the power of 10 is replaced by E followed by the exponent.

The table below shows a few numbers written in scientific notation.

Number	Scientific Notation	Scientific Notation in Calculator
45	$4.5 \cdot 10^{1}$	4.5E+1
45098	$4.5098 \cdot 10^{4}$	4.5098E+4
0.0002398	$2.398 \cdot 10^{-4}$	2.398E-4
- 892.8765	$- 8.928765 \cdot 10^{2}$	- 8.928765E+2

How to multiply by 10^n
If n > 0 move decimal point n places to the right.
If n < 0 move decimal point |n| places to the left.

The scientific notation is an example of a *floating-point* representation of real numbers. The exponent allows the decimal point to "float." Take the mantissa 4.5098 for example. If we follow it by 10^3 then we have the number 4509.8; if we follow it by 10^6, we have the number 4509800; and if we follow it by 10^{-2}, we get the number 0.045098.

Exercises

14. Write the following numbers in scientific notation.
    a) 231
    b) 34.212
    c) – 3409992
    d) 0.01023

15. Write the following numbers in their decimal form.
    a) $- 9.86 \cdot 10^{2}$
    b) $1.24 \cdot 10^{3}$
    c) $6.9087 \cdot 10^{-5}$
    d) – 8.349E+2

Floating-point representation of real numbers

The computer representation of real number, as the scientific notation, is a floating-point representation, but the mantissa is a whole number. A floating-point representation will have a fixed length with a fixed number of digits reserved for the mantissa and fixed number of digits reserved for the exponents. The left most digit of the mantissa must be at least 1. Let see how it works.

Suppose we use a 10-digit floating point representation. The first digit is reserved for the sign; we will assume that we have 6 digits for the mantissa and 3 digits for the exponents (including its sign). Then the number 4673 would be represented as

Sign	**Mantissa**						**Exponent**		
+	4	6	7	3	0	0	–	0	2

and the number 46,730,000,000 would be represented as

Sign	**Mantissa**						**Exponent**		
+	4	6	7	3	0	0	+	0	5

Note that, in the computer, the floating-point representation involves only binary digits. The mantissa and the exponents are binary numbers and the exponent has to be interpreted as a power of 2. The sign are expressed with a bit where 0 represents a positive sign (+) and 1 represents the negative sign (–).

Suppose that we have a 16-bit floating-point number with one bit for the sign, 11 bits for the mantissa and 4 bits for the exponents. Then the floating point number `1101 0011 0010 0101` has sign 1 which means negative sign (–), the mantissa is `101 0011 0010` and the exponent is `0101`. We convert the mantissa to $2^{10} + 2^8 + 2^5 + 2^4 + 2^1 = 1330$, then have to multiply the mantissa by 2^5 = 32. Hence the number is – 1330 · 32 = – 42,560.

If the exponent had been `1011` instead of `0010`, it would have been – 5 in decimal (see 2's complements). We would have multiplied 1330 by 2^{-5} and found – 1330 · 2^{-5} = – 41.5625.

Exercises

16. Give the 10-digit floating- point representations with a 6-digit mantissa and 3-digit exponent of the following numbers:
    a) 3892
    b) .93452
    c) 4,231,123
    d) -0.0023012
17. In this problem we consider the same10-digit floating- point representation as in question 16. Consider the number 0.234211.
    a) What is its floating-point representation?
    b) What is the *smallest* number *larger* than 0.234211 that has a floating-point representation different from 0.234211 without loss of precision?
    (Hint: increase the mantissa you found in a) by 1 and convert the number back to decimal.)

c) Give a number that would have same floating-point representation as 0.234211 and a number that would have same floating-point representation as the number you found in b). Choose the 2 numbers so that they are between 0.234211 and the number found in b).

Representing Sound

To understand how sound is represented on the computer we first have to understand what sound is. Sounds are waves of air pressure. These waves vibrate our ear drums and the signal is transmitted to our brain. A wave can be represented by a two-dimensional curve. The simplest wave is a sinusoidal wave. Three such waves are shown in figure 2 to 4. The pressure is on the vertical axes and the time on the horizontal axes.

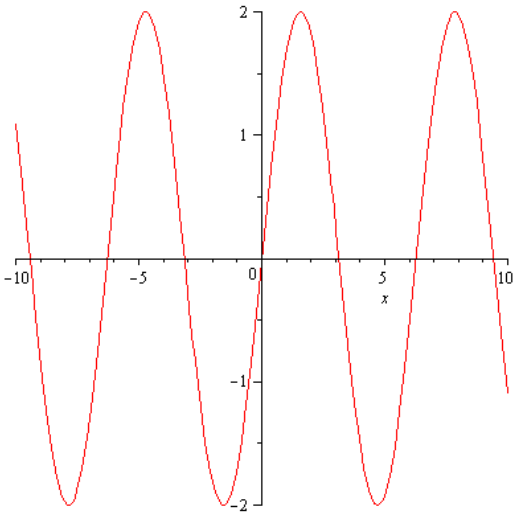


Figure 2- Wave 1

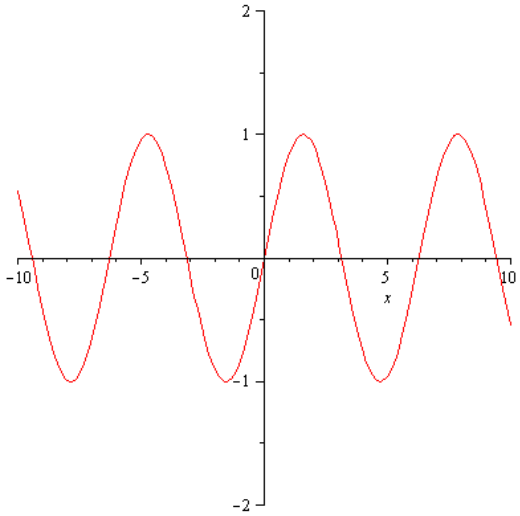


Figure 3- Wave 2

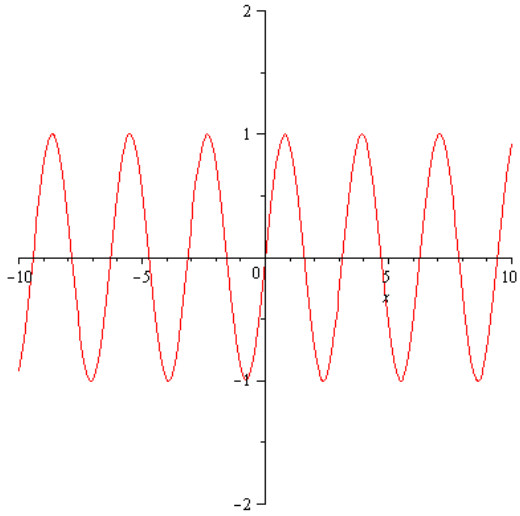


Figure 4- Wave 3

Two major characteristics of a wave curve are the amplitude and the frequency. The amplitude is half the distance between highest and lowest point in the curve. You notice that the wave repeats itself continuously. The frequency is the number of cycles per a given unit of time, usually per second. We can see wave 1 has same frequency as wave 2 but has amplitude twice as large as wave 2. On the other hand, wave 3 has same amplitude as wave 2 but a frequency twice as large (it repeats twice as fast).

Sound waves are not as simple as the ones above. They are combination of many different simple waves as above and can look very different. They are still characterized by amplitude and frequency but these amplitude and frequency change constantly. Figure 5 shows part of the sound wave generated by a brief whistle.

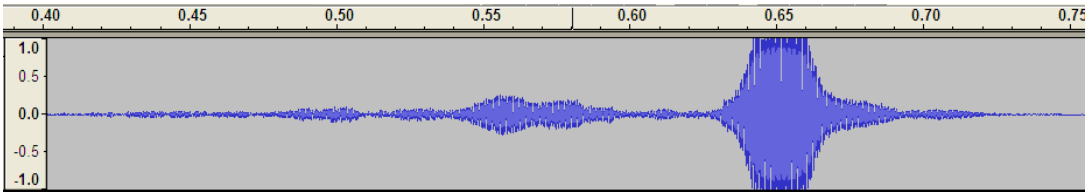


Figure 5

The picture seems to show more an area rather than a curve because the frequency is very high. If we zoom in (notice the change in the horizontal scale), you can see clearly the wave curve.

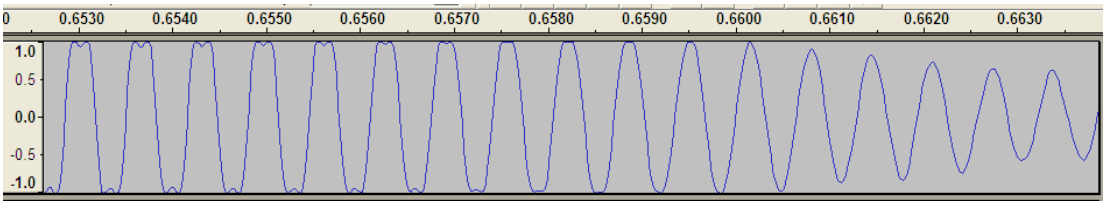


Figure 6

The amplitude of the sound wave corresponds to the volume of the sound: the larger the amplitude the louder the sound. The frequency of the sound wave corresponds to the pitch: the higher the frequency the higher the pitch. The frequency is measure in Hertz (abbreviation: Hz). One hertz is equal to 1 cycle per second. The range of human hearing is from 2 Hz to 20,000 Hz.

Now that we have some understanding of sound, let see how it is represented in the computer. As for an image it is clear that sound is analog in nature; hence it needs to be digitized to be stored in the computer. This will be done via a process called **sampling**. At regular interval, the sound is measured and recorded. This can is illustrated on the diagram of figure 7.

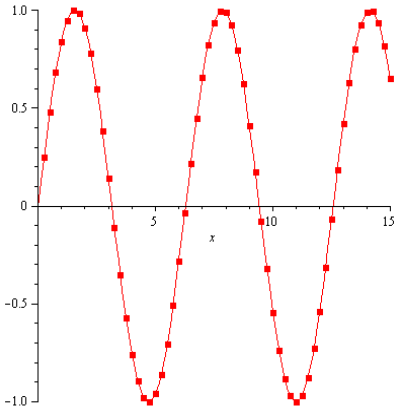


Figure 8

The dots on the curve at left represent the sample points. The data recorded is represented by the data points shown in figure 8.

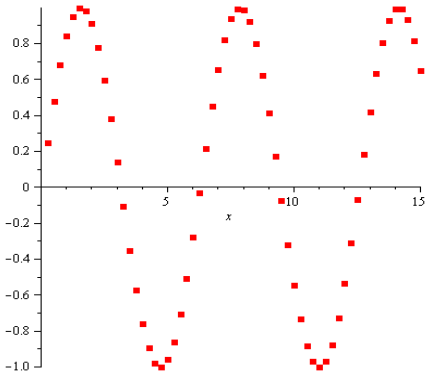


Figure 7

The sampling rate is the number of samples recorded per second. In general it is 44, 100 samples per second. It has been shown that with these many samples our ears cannot perceive the lost information. Each sample is then stored as either a 16-bit integer, sometime 24 –bit, or a 32-bit floating point number. The larger size sample allow for a greater range in amplitude recorded which means being able to record very soft or very loud sounds.
The device that captures the samples is called the analog-to-digital converter (ADC), which is part of the hardware of your computer. To playback the sounds, that is to recreate the sound waves from the samples the reverse

Sound files - MP3 files

The data collected for sound presents the same problem than images, a problem of size. This is of concern to anybody who likes to download music!
Let's compute the number of bits necessary to record a 3-minute song.
Assume that the sampling rate is the standard 44,100 samples per second and each sample is stored as a 16-bit integer. Assume that the song is recorded in stereo (we have to double the data because we record from two sources).

Computation: $44{,}100 \times 16 \times 180 \times 2 = 254{,}016{,}000$ bits needed

Even with a fast 1.5 megabits per second connection, it will take close to 3 minutes to download just this one song.
As for images, instead of storing the raw data, data collected during a recording (as described above) is usually processed and stored in a given format. Different sound file formats exist (e.g., WAV, AU, MP3). The different formats store and retrieve the data in distinct ways and, for most, part of the processing includes compression. One of the most popular sound file formats is the MP3 format. It uses a **compression algorithm** that utilizes the characteristics of human hearing to select the sounds that it eliminates, with a minimal impact on sound quality. In addition, by selecting the number of bits per second encoded in the MP3 files (the bit rate), we can create MP3 files of different file sizes and sound quality from the same recorded sound data.
For more information on the MP3 format, visit the website
http://computer.howstuffworks.com/mp3.htm .