1 Number Systems

Almost all modern computers are ***digital computers***, which means that they can recognize only two[1] distinct electronic states of electrical charge. For simplicity, these states are identified as **0** and **1**, or equivalently, **false** and **true**, or **off** and **on**. Since 0 and 1 are the most compact means of representing two states, data is represented as sequences of 0's and 1's. Sequences of 0's and 1's are binary numerals, or in common jargon, binary numbers.

To understand binary numbers, or for that matter, any other number systems, you should first revisit the one you have been used to your whole life, namely, the decimal number system. If you truly understand decimal numbers, then the rest is a piece of cake.

1.1 The Decimal Number System

Throughout the world, the main system of mathematical notation today is the ***decimal number system***, which is also called the ***base-10*** system. The term "***base***" refers to the number of distinct symbols that can be found in the numerals. The word "***decimal***" comes from the Latin word for "ten". In the decimal system, there are ten symbols, called ***digits***, which are universally written 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. ***Numerals*** are sequences of one or more digits. Numerals represent quantities, called ***numbers***.

You probably learned when you were young that the numeral "5" denotes a quantity consisting of five things, and that the numeral "9" represents the concept of nine things. I am sure your primary school teachers drilled these things into your brain until you could no longer separate the idea of nine things from a picture of the digit 9. This was useful then, but it is a hindrance to you now. If you speak more than one language, you are one step ahead, because you know that the quantity 9 is verbally representable in language in many ways.

At some point you learned how to find the ***values*** of larger numerals, such as 7,154 or 83,762. The two digit number "10" means "ten" things. We use the word "value" to mean the actual quantity that a numeral represents. The value of "9" is nine things, and the value of "23" is twenty-three things. The concept of placeholders is used to evaluate numerals.

Pick an arbitrary five-digit number such as 83,762. You know that the value of this number is

$$(8 \times 10{,}000) + (3 \times 1{,}000) + (7 \times 100) + (6 \times 10) + (2 \times 1)$$

In other words, in the base-10 number system, the digits occupy specific positions in the number. Most people were also taught that the zero, "0", is an indispensable part of this system because without it the system could not represent all possible numbers. This is a false statement though; we do not need that zero. However, it is very convenient because it makes this place-value system easier. The fact that the zero exists makes it easy to represent the quantity six hundred four as the numeral 604:

$$(6 \times 100) + (0 \times 10) + (4 \times 1)$$

[1] This is not entirely accurate. The term "digital" does not imply that there are only two distinguishable states. It implies that there is a finite number of such states, but since almost all modern computers are based on two-state logic, the term "digital" has come to take on this meaning in usage.

because we can use the "0" to "hold" the tens place so that the "6" stays put in the hundreds place.

At some point you learned about ***powers***, or ***exponents***, such as the "3" in 10^3, which you learned is the number of times by which 10 is multiplied by itself, i.e., $10^3 = 10 \times 10 \times 10$. You also were taught that 1 is really 10^0, and 10 is 10^1, and 100 is 10^2, and so on. Thus, 83,762 could also be written as

$$(8 \times 10^4) + (3 \times 10^3) + (7 \times 10^2) + (6 \times 10^1) + (2 \times 10^0)$$

These exponents make the representation shorter, and they also make the concepts more concise. Each position in a decimal numeral can now be thought of as representing the amount by which to multiply a specific power of ten to form the amount contributed by that placeholder in the numeral's ***value***.

To generalize this idea, think of a five-digit numeral more abstractly by writing it as $d_4d_3d_2d_1d_0$, in which d_4 stands for the leftmost digit (e.g., 8 in this case), d_3 is the one after that (e.g., 3 in this case), d_2 is the one after that (e.g., 7 in this case), and so on. *Notice that there are five digits, but that their subscripts start at 0* and stop at 4: d_0 is the ones' place, d_1 is the tens' place, d_2, the hundreds' place, d_3, thousands', and finally d_4 ten-thousands' . Since $1 = 10^0$ and $10 = 10^1$ and so on, the value of the numeral $d_4d_3d_2d_1d_0$ is

$$(d_4 \times 10^4) + (d_3 \times 10^3) + (d_2 \times 10^2) + (d_1 \times 10^1) + (d_0 \times 10^0)$$

In other words, the subscript matches the power of ten by which the digit is multiplied. The usual way to visualize this is by thinking of the positions in a number with N digits is as follows:

10^{N-1}	...	10^4	10^3	10^2	10^1	10^0
d_{N-1}	...	d_4	d_3	d_2	d_1	d_0

1.2 The Binary Number System

We carry these same ideas into the a number system with just two digits. Everything works in exactly the same way. Suppose we only have two digits, 0 and 1. The digits 0 and 1 have the same value as in base 10, namely 0 represents zero things and 1, one thing. How can we write all possible numbers with just two digits? We can use the same principle as we used with base 10, except we will replace base 10 by base 2. In other words, a binary numeral is a sequence of ***binary digits***, called **bits**. ***Each bit position is multiplied by a power of 2***. Because these are called bits and not digits, I will use the letters b_1, b_2, and so on to stand for bits. The binary numeral 101, for example, represents the quantity that we write as 5 in decimal, because 101 stands for

$$(1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) = 2^2 + 0 + 2^0 = 4 + 0 + 1 = 5$$

More generally, to evaluate a binary numeral, you need to write out the powers of two the same was we wrote powers of ten before. The N-bit binary numeral b_{N-1} ... $b_4b_3b_2b_1b_0$ should be visualized by the following table:

2^{N-1}	...	2^4	2^3	2^2	2^1	2^0
b_{N-1}	...	b_4	b_3	b_2	b_1	b_0

If you do not know powers of two by heart (why should you?), it is easier to write out the table with an extra row in which the powers are expanded. For example, to find the value of the binary numeral 110101, make the table:

2^5	2^4	2^3	2^2	2^1	2^0
32	16	8	4	2	1
1	1	0	1	0	1

and then calculate its value:

$(1 \times 2^5) + (1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0)$

= 32 + 16 + 0 ++ 4 + 0 + 1

= 53

In general, you would write out the table with as many columns as the length of the number:

...	1024	512	256	128	64	32	16	8	4	2	1
...	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
...	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0

Since most people can double a number fairly easily, it is easier to determine the value represented by a sequence of bits if you start from the right hand side of the numeral and work towards the left, doubling as you go. Thus, 1000100101 is:

512	256	128	64	32	16	8	4	2	1
1	0	0	0	1	0	0	1	0	1

so that

$1000100101 = (1 \times 1) + (1 \times 4) + (1 \times 32) + (1 \times 512) = 1 + 4 + 32 + 512 = 549$.

The first few binary numbers are listed in the table below with their corresponding values in base 10.

Binary	Decimal
0	0
1	1
10	2
11	3
100	4
101	5
110	6
111	7
1000	8
1001	9
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15
10000	16

1.2.1 Binary Arithmetic

Notice in the table how the successive number is obtained from the one before it. Adding 1 in binary is just like adding 1 in decimal; the same methods are used, except that carrying and borrowing are in base 2.

Examples

```
                 1
    11        1 1       1001
+    1   =>  _-1       +   1
------       ---       -----
     0       100        1010


 10011      111
 + 101     +111
 -----     ----
 11000     1110
```

It is not a coincidence that 111 + 111 = 1110, i.e., the same numeral but with a zero on the right. After all, 111 + 111 is the same as doubling 111, which is the same as 2 × 111, and since 2 is written as 10 in binary, this is the same as

```
   111
×   10
  ----
   000
  111
 -----
  1110
```

So you see that multiplying by 2 in binary is as easy as multiplying by ten in decimal -- just put 0 on the right side of the numeral! If you know this fact and you know how to add, you can do multiplication in binary. You can do more complicated examples as long as you remember your binary arithmetic tables, namely

+	0	1
0	0	1
1	1	10

×	0	1
0	0	0
1	0	1

1.2.2 Decimal To Binary Conversion

Converting decimal numbers to binary is a little harder. There are two methods.

1.2.2.1 Remainder Method

In this method you use the kind of division you learned about in elementary school before you knew about the existence of fractions. It actually has a name. It is called ***integer division***. When you divide the ***dividend*** by the ***divisor*** you get a ***quotient*** and a ***remainder***.

For example, 11 divided by 4 is 2 remainder 3, and 17 divided by 3 is 5 remainder 2. In integer division, dividing by 2 has only two possible results: no remainder (a remainder of 0) or a remainder of 1: 9/2 = 4 remainder 1 and 8/2 = 4 remainder 0.

Most importantly:

1/2 = 0 remainder 1
0/2 = 0 remainder 0.

Let x be the decimal number you wish to write in binary. Use integer division in the following algorithm.

1. Divide x by 2.
2. Write the remainder to the immediate left of the previous remainder. If this is the first remainder, right it down leaving room to its left for more bits.
3. If the quotient is not 0, make the quotient the new value of x and go back to step 1, otherwise go to step 4.
4. The binary numeral is the sequence of remainders in left-to-right order.

Example
Let x = 1000.

x	Arithmetic	Quotient of x div 2	Remainders
1000	1000/2 = 500 r 0	500	0
500	500/2 = 250 r 0	250	00
250	250/2 = 125 r 0	125	000
125	125/2 = 62 r 1	62	1000
62	62/2 = 31 r 0	31	01000
31	31/2 = 15 r 1	15	101000
15	15/2 = 7 r 1	7	1101000
7	7/2 = 3 r 1	3	11101000
3	3/2 = 1 r 1	1	111101000
1	1/2 = 0 r 1	0	1111101000

The method stops when the quotient is 0, and the answer is 1111101000. You can verify that this is correct by using the binary-to-decimal conversion table and adding up the powers of two.

1.2.2.2 Subtraction Method

The procedure is as follows.
Let x be the decimal numeral you want to write in binary.

Repeat the following until x equals 0:

1. Find the largest power of 2 that is less than or equal to x. Suppose it is 2^n. Let y = x - 2^n. Write 2^n on the side.
2. If y is not 0, set x = y and repeat step 1. Otherwise go to step 3
3. Write the powers of 2 set aside in descending order. If an exponent is missing, fill its place with 0. Replace the powers by 1's.

It helps to know the powers of 2. Here are the first 17 of them:

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2^n	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768	65536

Example

To convert 1000 to binary,
Find the largest power of 2 less than or equal to 1000:
1000 - 512 = 488. Using the table, 512 = $\mathbf{2^9}$.

Find the largest power of 2 less than or equal to 488:
488 - 256 = 232. Using the table, 256 = $\mathbf{2^8}$.

Find the largest power of 2 less than or equal to 232:
232 - 128 = 104. Using the table, 128 = $\mathbf{2^7}$.

Find the largest power of 2 less than or equal to 104:
104 - 64 = 40. Using the table, 64 = $\mathbf{2^6}$.

Find the largest power of 2 less than or equal to 40:
40 - 32 = 8. Using the table, 32 = $\mathbf{2^5}$.

Find the largest power of 2 less than or equal to 8:
8 - 8 = 0. Using the table, 8 = $\mathbf{2^3}$.

Since the remainder is 0, 1000 we stop.
Write the powers, filling in with 0's. $\mathbf{2^9}$ $\mathbf{2^8}$ $\mathbf{2^7}$ $\mathbf{2^6}$ $\mathbf{2^5}$ 0 $\mathbf{2^3}$ 0 0 0. Replace the powers by 1's: 1 1 1 1 1 0 1 0 0 0. The binary is this 1111101000:

2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	1	1	1	0	1	0	0	0

Exercises.

1. Find the binary representations of
   a. 1023
   b. 81

   c. 224
2. Find the decimal representations of these 8-bit binary numbers.
   a. 11111111
   b. 11110000
   c. 11001100
   d. 10101010
3. Without converting these numbers to decimals, find their products
   a. 101 × 110
   b. 111 × 111

1.2.3 Negative Numbers

You may be wondering how to represent negative binary numbers. There are two different methods, but the most common one is called ***two's complement***, and it depends how many bits are used to represent numbers. In most computers, 32 bits are used to represent whole numbers. Sticking with the 32-bit representation, the number 1 looks like this:

0000 0000 0000 0000 0000 0000 0000 0001

the number 32,769 looks like this:

0000 0000 0000 0000 1000 0000 0000 0001

and the number 2,147,483,647 looks like

0111 1111 1111 1111 1111 1111 1111 1111

There is a reason that I put spaces between every four bits. It makes them easier to read, and it will come into play when I introduce hexadecimal numbers, below. The last number above is the largest whole number that can fit in 32 bits. If the leftmost bit had a 1 in it, the number would be negative. So,

1111 1111 1111 1111 1111 1111 1111 1111

is surprisingly the number -1. If a binary number has a 1 in the 32nd bit, to find its value, subtract 1 from it and then turn every 1 into a 0 and every 0 into a 1:

subtract 1:	1111 1111 1111 1111 1111 1111 1111 1110
switch 1's and 0's:	0000 0000 0000 0000 0000 0000 0000 0001

The number is the negative of the binary number you get from this. So the last row is 1, and the number with all 1 bits is -1.

To find what the binary numeral for -2 would be, reverse the process: start with positive 2, switch 1's and 0's, then add 1:

positive 2:	0000 0000 0000 0000 0000 0000 0000 0010
switch 1's and 0's:	1111 1111 1111 1111 1111 1111 1111 1101
add 1:	1111 1111 1111 1111 1111 1111 1111 1110

This is -2 in binary.

So what is the value of 1000 0000 0000 0000 0000 0000 0000 0000?

Start with

1000 0000 0000 0000 0000 0000 0000 0000

Subtract 1 and you get

0111 1111 1111 1111 1111 1111 1111 1111

Then switch bits: 1000 0000 0000 0000 0000 0000 0000 0000

You get back the same numeral you started with! This tells you that the binary number that has a 1 in the leftmost position followed by all zeros, is -2^{31}, which is -2,147,483,648.

Food for Thought. *How do you think fractions can be represented in this system?*

1.3 Hexadecimal Number System

It is difficult to read binary numerals. They are long, compared to their equivalent decimal numerals. (In fact, the binary representation is a little over three times as long as the decimal equivalent, on average.) For this reason, binary numerals are often transformed to either octal or hexadecimal numerals.

The word "hexadecimal" took on life in the early 1950's within the halls of IBM.[2] The word means "16". It was originally ***sexadecimal***, since that is closer to pure Latin, but for evident reasons, they replaced the "sex" with the Greek "hex".

The hexadecimal number system is base 16. Since it uses 16 digits but there are only 10 digits in the decimal system, the first six letters of the alphabet are used as the next six digits. Thus, the digits of the hexadecimal system are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F, in which A is ten, B is eleven, C is twelve, D, thirteen, E, fourteen, and F, fifteen.

The hexadecimal number system solves two problems.

- Each hexadecimal digit uses exactly four bits, making the conversion from binary to hexadecimal easy, and
- Hexadecimal numbers are very compact and easy to read.

The reason that hexadecimal digits use four bits is that $16 = 2^4$. The four-bit codes for the hexadecimal digits are in the table below.

[2] According to Wikipedia.

Hexadecimal	Binary Code
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Notice that these codes are just the 4-bit binary numerals that represent the numbers from 0 to 15.

Why is this helpful? Take any binary number to illustrate. Suppose it is a long, hard-to-read number, such as

```
01101100011111111000110011110010
```

Break it up into 4-bit groups:

```
0110 1100 0111 1111 1000 1100 1111 0010
```

Now use the table to the left and replace each 4-bit group by the corresponding hexadecimal number:

```
  6    C    7    F    8    C    F    2
```

Put these together and you have

```
    6C7F8CF2
```

This is the hexadecimal representation of the 32-bit binary number above. Usually, hexadecimal numbers are preceded by "0x" to let people know what they are:

```
    0x6C7F8CF2
```

is the right way to write the above number.

It is very easy to reverse this process, i.e., to go from hexadecimal to binary. For example, to convert the hexadecimal number 0x7FF011A4 to binary, use the table to replace each hexadecimal digit by the 4-bit code for it:

0x 7FF011A4 = 0111 1111 1111 0000 0001 0001 1010 0100

It is also possible to convert decimal numbers to hexadecimal numbers but the method is different than it was for converting from base 10 to base 2. Again it is helpful to have a table of the powers of 16 (or a calculator).

n	0	1	2	3	4	5	6	7	8
16^n	1	16	256	4,096	65,536	1,048,576	16,777,216	268,435,456	4,294,967,296

Example

To convert 100,000 to binary,

Find the largest power of 16 less than or equal to 100,000. Divide 100,000 by this number and save the quotient and the remainder:

100,000 ÷ 65,536 = 1 r 34,464. Put a 1 in position 4 and continue.

Find the largest power of 16 less than or equal to 34,464 and divide 34,464 by this number:

34,464 ÷ 4096 = 8 r 1696. Put an 8 in position 3 and continue.

Find the largest power of 16 less than or equal to 1666 and divide 1666 by this number::

1696 ÷ 256 = 6 r 160. Put a 6 in position 2 and continue.

Find the largest power of 16 less than or equal to 130 and divide again:

160 ÷ 16 = 10 r 0. 10 is A in base 16, so put an A in position 1. Since the remainder is 0, all positions to the right are filled with 0's. In this case it is just position 0.

Thus, 100,000 in hexadecimal is 186A0.

1.4 Octal Number System

The word "octal" comes from the Latin root of "eight". The octal number system is an 8-digit number system. Because it has 8 digits, octal numerals are shorter than their binary equivalents (but longer than their decimal and hexadecimal equivalents.) It is easy to convert binary to octal. The principle is the same as it is in converting to hexadecimal, except that we decompose the binary number into groups of 3 bits each, since $8 = 2^3$. If the binary numeral does not have a multiple of 3 bits, the leftmost group may be 1 or 2 bits long.

The codes for the octal numbers are below:

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

Example

Convert the following to octal:

```
01101100011111110001100111110010
```

Break it up into 3-bit groups:

```
01 101 100 011 111 111 000 110 011 110 010
```

Now use the table to the left and replace each 3-bit group by the corresponding octal number:

```
 1   5   4   3   7   7   0   6   3   6   2
```

Put these together and you have

```
        15437706362
```

Exercises.

4. Convert the following binaries to octal:
    a. 1001011111011100110010
    b. 100010111001011001
5. Figure out how to convert octal to binary on your own and write these octal numerals in binary:
    a. 37001
    b. 177523