

Chapter 1 – Information Coding

Modern computers are implemented using digital electronic technology, with basic building blocks such as transistors and logic gates. In the machine, every piece of information (numbers, text, pictures, programs) is encoded in terms of boolean values i.e. zeroes and ones. In other words, the computer thinks in binary. The goal for today is to get familiar with binary.

1 Counting in base 2

You're used to counting in base 10:

- decimal digits take their value in $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- values bigger than 9 are represented by concatenating digits for successive powers of 10:
- $234 = 2 \times 10^2 + 3 \times 10^1 + 4 \times 10^0$
- $45678 = 4 \times 10^4 + 5 \times 10^3 + 6 \times 10^2 + 7 \times 10^1 + 8 \times 10^0$

Binary numbers are the exact same idea but in base 2:

- binary digits (**bits**) take their value in $\{0, 1\}$
- values bigger than 1 are represented by concatenating bits for successive powers of 2:
- binary $110 = 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = \text{decimal } 6$
- binary $110011 = 1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = \text{decimal } 99$

When the context is not enough to distinguish “binary 110” from “decimal 110” we will use a “0b” prefix to indicate binary numbers, e.g. 0b1010=10.

Binary-to-Decimal Conversion Let x be a sequence of n bits $x_{n-1}, x_{n-2}, \dots, x_1, x_0$ (warning: we always write bits x_i with **descending** values of i , down to zero). To interpret the value of x as a natural number, we apply the same formula as in decimal, but with a radix of 2.

$$x = \sum_{i=0}^{n-1} x_i \cdot 2^i$$

Example: 0b110010 reads as $32+16+0+0+2+0 = 50$. (Please don't learn the powers of 2 by heart ! We provide all you need on the following page.)

Exercise Convert 0b101010 to decimal. Same question with 0b111110100.

Exercise (10 min max) Show that with n bits, the largest integer that can be represented is $2^n - 1$.

Binary numbers: bit length vs numeric range A sequence of n bits can take any one of 2^n configurations. The smallest natural number encoded on n bits is 0. The largest natural number encoded on n bits is $2^n - 1$.

Examples:

- $n = 4: x \in [0..15] \Rightarrow 16$ different values
- $n = 8: x \in [0..255] \Rightarrow 256$ different values
- $n = 16: x \in [0..65535] \Rightarrow 65536$ different values
- $n = 32: x \in [0..4294967295] \Rightarrow 4294967296$ different values
(that's about $4 \times 10^9 \sim 4$ Billion ~ 4 Giga)
- $n = 64: x \in [0..18446744073709551615] \Rightarrow 18446744073709551616$ different values
(that's about $1.8 \times 10^{19} \sim 18$ Quintillion)

Useful: Powers of 2

$2^0 = 1$	$2^{16} = 65\,536$	$2^{32} = 4\,294\,967\,296$	$2^{48} = 281\,474\,976\,710\,656$
$2^1 = 2$	$2^{17} = 131\,072$	$2^{33} = 8\,589\,934\,592$	$2^{49} = 562\,949\,953\,421\,312$
$2^2 = 4$	$2^{18} = 262\,144$	$2^{34} = 17\,179\,869\,184$	$2^{50} = 1\,125\,899\,906\,842\,624$
$2^3 = 8$	$2^{19} = 524\,288$	$2^{35} = 34\,359\,738\,368$	$2^{51} = 2\,251\,799\,813\,685\,248$
$2^4 = 16$	$2^{20} = 1\,048\,576$	$2^{36} = 68\,719\,476\,736$	$2^{52} = 4\,503\,599\,627\,370\,496$
$2^5 = 32$	$2^{21} = 2\,097\,152$	$2^{37} = 137\,438\,953\,472$	$2^{53} = 9\,007\,199\,254\,740\,992$
$2^6 = 64$	$2^{22} = 4\,194\,304$	$2^{38} = 274\,877\,906\,944$	$2^{54} = 18\,014\,398\,509\,481\,984$
$2^7 = 128$	$2^{23} = 8\,388\,608$	$2^{39} = 549\,755\,813\,888$	$2^{55} = 36\,028\,797\,018\,963\,968$
$2^8 = 256$	$2^{24} = 16\,777\,216$	$2^{40} = 1\,099\,511\,627\,776$	$2^{56} = 72\,057\,594\,037\,927\,936$
$2^9 = 512$	$2^{25} = 33\,554\,432$	$2^{41} = 2\,199\,023\,255\,552$	$2^{57} = 144\,115\,188\,075\,855\,488$
$2^{10} = 1\,024$	$2^{26} = 67\,108\,864$	$2^{42} = 4\,398\,046\,511\,104$	$2^{58} = 288\,230\,376\,151\,711\,744$
$2^{11} = 2\,048$	$2^{27} = 134\,217\,728$	$2^{43} = 8\,796\,093\,022\,208$	$2^{59} = 576\,460\,752\,303\,423\,488$
$2^{12} = 4\,096$	$2^{28} = 268\,435\,456$	$2^{44} = 17\,592\,186\,044\,416$	$2^{60} = 1\,152\,921\,504\,606\,846\,976$
$2^{13} = 8\,192$	$2^{29} = 536\,870\,912$	$2^{45} = 35\,184\,372\,088\,832$	$2^{61} = 2\,305\,843\,009\,213\,693\,952$
$2^{14} = 16\,384$	$2^{30} = 1\,073\,741\,824$	$2^{46} = 70\,368\,744\,177\,664$	$2^{62} = 4\,611\,686\,018\,427\,387\,904$
$2^{15} = 32\,768$	$2^{31} = 2\,147\,483\,648$	$2^{47} = 140\,737\,488\,355\,328$	$2^{63} = 9\,223\,372\,036\,854\,775\,808$
			$2^{64} = 18\,446\,744\,073\,709\,551\,616$

Exercise (hands-on) Open a terminal window and type `python` (or maybe `python3` ; ask for help until you get to the standard prompt i.e. "`>>>`"). Type a binary number e.g. `0b110010` then press Return. Observe how python prints numbers in decimal by default (it can be changed, as we'll see later)

Exercise (hands-on) Besides usual arithmetic operations (+, −, × etc) python also offers a built-in exponentiation operator: writing `x**y` will compute x^y i.e. x to the power of y . Try computing a few examples of powers of two e.g. `2**10`, or `2**32` etc. (btw, this is how we generated the table above !)

2 Decimal-to-Binary Conversion

Method A: divide by two, repeatedly Given a number x , the remainder of the euclidean division of x by 2 gives the right-most digit of its binary representation (in other words $x_0 = x \bmod 2$):

$$\begin{aligned}
 x &= x_{n-1} \cdot 2^{n-1} + x_{n-2} \cdot 2^{n-2} + \dots + x_2 \cdot 2^2 + x_1 \cdot 2^1 + x_0 \\
 &= \underbrace{(x_{n-1} \cdot 2^{n-2} + x_{n-2} \cdot 2^{n-3} + \dots + x_2 \cdot 2^1 + x_1)}_{\text{quotient}} \cdot 2 + \underbrace{x_0}_{\text{remainder}}
 \end{aligned}$$

If we repeat this procedure to the quotient until we reach 0, then we get all successive digits x_1, x_2 and so on (from right to left). Example with $x = 423$:

$$\begin{aligned}
 423 &= 211 \times 2 + 1 \\
 211 &= 105 \times 2 + 1 \\
 105 &= 52 \times 2 + 1 \\
 52 &= 26 \times 2 + 0 \\
 26 &= 13 \times 2 + 0 \\
 13 &= 6 \times 2 + 1 \\
 6 &= 3 \times 2 + 0 \\
 3 &= 1 \times 2 + 1 \\
 1 &= 0 \times 2 + 1
 \end{aligned}$$

From this we conclude that: $423 = 0b110100111$

Method B: subtract powers of two Given a number x , we search the biggest i such that $2^i < x$. This tells us that bit $x_i = 1$. Then we subtract 2^i from x and repeat until x reaches one or zero, which tells us our final bit x_0 .

Example with $x = 423$ again:

- what's the biggest power of 2 smaller than 423 ?
 - that's $256 = 2^8$
 - $423 - 256 = 167$
- what's the biggest power of 2 smaller than 167 ?
 - that's $128 = 2^7$
 - $167 - 128 = 39$
- what's the biggest power of 2 smaller than 39 ?
 - that's $32 = 2^5$
 - $39 - 32 = 7$
- what's the biggest power of 2 smaller than 7 ?
 - that's $4 = 2^2$
 - $7 - 4 = 3$
- what's the biggest power of 2 smaller than 3 ?
 - that's $2 = 2^1$
 - $3 - 2 = 1$

In conclusion, we find that $423 = 2^8 + 2^7 + 2^5 + 2^2 + 2^1 + 1 = 0b110100111$

Exercise (pen & paper) Convert 23 to binary. Same question with 200.

Exercise (hands-on) Play with the `bin()` function of python to convert a few numbers to binary notation. For instance, `bin(423)` returns `0b110100111`.

Remark Throughout this course, you should never hesitate to “cheat” (by using a computer) to save time and/or to check your results. But you should still be able to do things by hand when required !

3 Hexadecimal Notation

Significant Digits On paper we usually only write significant digits (i.e. 0011 really is the same as 11) but within the computer, bits are stored in bundles of 8 bits called **bytes**. A byte can take 256 distinct values, from 00000000 to 11111111. Larger numbers are usually represented on 32 bits (4 bytes aka a **word**) or sometimes on 16 bits (2 bytes).

Counting in base 16 is actually easier than it sounds. Indeed $16=2^4$, so one digit in base 16 represents 4 digits in base 2. This means that we can represent each bundle of 4 bits with one **hexadecimal digit**, as illustrated below.

Dec	Hex	Bin
0	0	0
1	1	1
2	2	10
3	3	11
4	4	100

Dec	Hex	Bin
5	5	101
6	6	110
7	7	111
8	8	1000
9	9	1001

Dec	Hex	Bin
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110

Dec	Hex	Bin
15	F	1111
16	10	10000
17	11	10001
18	12	10010
19	13	10011

Hexadecimal is very useful as a compact notation for binary numbers, but we will never compute in base 16, or do direct conversions. To avoid confusion, hexadecimal notation uses the “0x” prefix, e.g. $423=0x1A7$.

Exercise (pen & paper) Encode your first name in ASCII.

Exercise (hands-on) Use python to play with ASCII encoding and decoding. The relevant functions are `chr()` to go from integers to characters, and `ord()` to go the other way.

5 RGB

Binary numbers can also represent colors. In the RGB model, a color is described by indicating how much of each of the red, green, and blue light is included. The color is expressed as an RGB triplet (r,g,b), where each component can vary from zero to a defined maximum value. If all the components are at zero the result is black; if all are at maximum, the result is the brightest representable white.

Exercise (hands-on) Download `rgb.py` from Moodle, and type `python3 rgb.py FFFFFFFF` to execute it. The last argument on the command-line is a **hex triplet** describing the desired color with three byte values (from 00 to FF in hexadecimal). In other words 000000 means “black” and FFFFFFFF means “white”. Try both. Then, using trial and error, find out the hex triplets corresponding to standard colors like yellow, purple, orange, cyan, etc.