Binary number representation

We humans represent numbers using decimal (the ten digits from 0,1, ... 9) or "base 10". The reason we do so is that we have ten fingers. There is nothing special otherwise about the number ten. Computers don't represent numbers using decimal. Instead, they are designed to represent numbers using binary, or "base 2". Let's make sure we understand what binary representations of numbers are. We'll start with positive integers.

In decimal, we write numbers using *digits* $\{0, 1, \ldots, 9\}$, in particular, as sums of powers of ten, for example,

$$(238)_{10} = 2 * 10^2 + 3 * 10^1 + 8 * 10^0$$

whereas, in binary, we represent numbers using *bits* $\{0, 1\}$, as a sum of powers of two:

$$(11010)_2 = 1 * 2^4 + 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0.$$

I have put little subscripts (10 and 2) to indicate that we are using a particular representation (decimal or binary). We don't need to always put this subscript in, but sometimes it helps.

You know how to count in decimal, so let's consider how to count in binary. You should verify that the binary representation is a sum of powers of 2 that indeed corresponds to the decimal representation in the leftmost column. In the left two columns below, I only used as as many digits or bits as I needed to represent the number. In the right column, I used a fixed number of bits, namely 8. 8 bits is called a *byte*.

decimal	binary	binary (8 bits)
0	0	00000000
1	1	00000001
2	10	00000010
3	11	00000011
4	100	00000100
5	101	00000101
6	110	00000110
7	111	00000111
8	1000	00001000
9	1001	00001001
10	1010	00001010
11	1011	00001011

etc

Converting from decimal to binary

It is trivial to convert a number from a binary representation to a decimal representation. You just need to know the decimal representation of the various powers of 2.

$$2^0 = 1,\ 2^1 = 2,\ 2^2 = 4,\ 2^3 = 8,\ 2^4 = 16,\ 2^5 = 32,\ 2^6 = 64,\ 2^7 = 128,\ 2^8 = 256,\ 2^9 = 512,\ 2^{10} = 1024,\ \ldots$$

Then, for any binary number, you write each of its '1' bits as a power of 2 using the decimal representation you are familiar with. Then you add up these decimal numbers, e.g.

$$11010_2 \;=\; 16 + 8 + 2 = 26.$$

The other direction is more challenging, however. How do you convert a decimal number to binary? I will give a simple algorithm for doing so soon which is based on the following idea. I first explain the idea in base 10 where we have a better intuition. Let m be a positive integer which is written in decimal. Then,

$$m = 10 * (m/10) + (m\%10).$$

Note that $m/10$ chops off the rightmost digit and multiplying by 10 tags on a 0. So dividing and the multiplying by 10 might not get us back to the original number. What is missing is the remainder part, which we dropped in the divison.

In binary, the same idea holds. If we represent a number m in binary and we divide by 2, then we chop off the rightmost bit which becomes the remained and we shift right each bit by one position. To multiply by 2, we shift the bits to the left by one position and put a 0 in the rightmost position. So, for example, if

$$m = (11011)_2 \;=\; 1 * 2^4 + 1 * 2^3 + 1 * 2^1 + 1 * 2^0$$

then dividing by 2 gives

$$(11011)_2/2 \;=\; (1101)_2$$

then multiplying by 2 gives

$$(11010)_2 \;=\; 1 * 2^4 + 1 * 2^3 + 1 * 2^1.$$

More generally, for any m,

$$m = 2 * (m/2) + (m \;\%\; 2).$$

Here is the algorithm for converting m to binary. It is so simple you could have learned it in grade school. The algorithm repeatedly divides by 2 and the "remainder" bits $b[i]$ are the bits of the binary representation. After I present the algorithm, I will explain *why* it works.

Algorithm 1 Convert decimal to binary

INPUT: a number m
OUTPUT: the number m expressed in base 2 using a bit array $b[\,]$

```
  $i \leftarrow 0$
  while $m > 0$ do
    $b[i] \leftarrow m \% 2$
    $m \leftarrow m / 2$
    $i \leftarrow i + 1$
  end while
```

Note this algorithm doesn't say anything about how m is represented. But in practice, since you are human, and so m is represented in decimal.

Example: Convert 241 to binary

i	m	b[i]
	$\boxed{241}$	
0	120	1
1	60	0
2	30	0
3	15	0
4	7	1
5	3	1
6	1	1
7	0	1
8	0	0
9	:	:

Thus, $(241)_{10} = (11110001)_2$. Note that there are an infinite number of 0's on the left which are higher powers of 2 which we ignore.

Now let's apply these ideas to the algorithm for converting to binary. Representing a positive integer m in binary *means* that we write it as a sum of powers of 2:

$$m = \sum_{i=0}^{n-1} b_i \, 2^i$$

where b_i is a bit, which has a value either 0 or 1. So we write m in binary as a bit sequence $(b_{n-1}\ b_{n-2}\ \ldots\ b_2\ b_1\ b_0)_2$. In particular,

$$\begin{aligned} m \ \% \ 2 &= b_0 \\ m \ / \ 2 &= (b_{n-1} \ldots b_2 b_1)_2 \end{aligned}$$

Thus, we can see that the algorithm for converting to binary, which just repeats the mod and division operations, essentially just read off the bits of the binary representation of the number!

If you are still not convinced, let's run another example where we "know" the answer from the start and we'll see that the algorithm does the correct thing. Suppose our number is $m = 241$, which is $(11110001)_2$ in binary. The algorithm just reads off the rightmost bit of m each time that we divide it by 2.

i	m	b[i]
	$\boxed{(11110001)_2}$	
0	$(1111000)_2$	1
1	$(111100)_2$	0
2	$(11110)_2$	0
3	$(1111)_2$	0
4	$(111)_2$	1
5	$(11)_2$	1
6	$(1)_2$	1
7	0	1

Arithmetic in binary

Let's add two numbers which are written in binary. I've written the binary representation on the left and the decimal representation on the right.

```
11010    <-carries
 11010                26
+ 1011               +11
------              ----
100101                37
```

Make sure you see how this is done, namely how the "carries" work. For example, in column 0, we have $0+1$ and get 1 and there is no carry. In column 1, we have $1+1$ (in fact, $1*2^1+1*2^1$) and we get $2*2^1=2^2$ and so we carry a 1 over column 2 which represents the 2^2 terms. Make sure you understand how the rest of the carries work.

How many bits N do we need to represent m ?

Let N be the number of bits needed to represent an integer m, that is,

$$m = b_{N-1}2^{N-1} + b_{N-2}2^{N-2} + \ldots b_1 2 + b_0$$

where $b_{N-1}=1$, that is, *we only use as many bits as we need.* This is similar to the idea that in decimal we don't usually write, for example, 0000364 but rather we write 364. We don't write 0000364 because the 0's on the left don't contribute anything.

Let's derive an expression for how many bits N we *need* to represent n. I will do so by deriving a lower bound and an upper bound for N. First, the lower bound: Since $b_{N-1}=1$ and since each of the other b_i's is either 0 or 1 for $0 \le i < N-1$, we have

$$m \;\le\; 2^{N-1} + 2^{N-2} + \cdots + 2 + 1. \qquad (*)$$

To go further, I will next use of the following claim which many of you have seen from Calculus: for any real number number x,

$$\sum_{i=0}^{N-1} x^i = \frac{x^N - 1}{x-1}. \qquad (**)$$

The proof of this fact goes as follows. Take the sum on the left and multiply by $x-1$ and expand:

$$\sum_{i=0}^{N-1} x^i(x-1) = \sum_{i=1}^{N} x^i - \sum_{i=0}^{N-1} x^i$$

Note the indices of the first sum on the right go from 1 to N and the second go from 0 to $N-1$. Because we are taking a difference on the right, all terms cancel except for two, namely

$$\sum_{i=1}^{N} x^i - \sum_{i=0}^{N-1} x^i = x^N - 1.$$

Thus,

$$\sum_{i=0}^{N-1} x^i(x-1) == x^N - 1.$$

which is what I claimed above.

If we consider the case $x = 2$ we get

$$\sum_{i=0}^{N-1} 2^i = 2^N - 1.$$

For example, think of $N = 4$. We are saying that $1 + 2 + 4 + 8 = 16 - 1 = 15$.

Anyhow, getting back to our problem of deriving a lower bound on N, from equation (*) above, we have

$$\begin{aligned} m &\leq 2^{N-1} + 2^{N-2} + \cdots + 2 + 1 \\ &= 2^N - 1 \quad \text{which I just proved} \\ &< 2^N \quad \text{which makes the following step cleaner} \end{aligned}$$

Taking the $\log_2$ (base 2) of both sides gives:

$$\log_2 m < N.$$

Now let's derive an upper bound. Since we are only considering the number of bits that we need, we have $b_{N-1} = 1$, and since the b_i are either 0 or 1 for any $i < N - 1$, we can conclude

$$m \ \geq \ 1 * 2^{N-1} + 0 * 2^{N-1} + \ldots.0 * 2^1 + 0 * 2^0 \ = \ 2^{N-1}$$

and so

$$\log_2 m >= N - 1.$$

Rearranging and putting the two inequalities together gives

$$\log_2 m < N \leq \log_2 m + 1$$

Noting that N is an integer, we conclude that N is the largest integer that is less than or equal to $\log_2 m + 1$, that is N is $\log_2 m + 1$ rounded down. We write: $N = floor(\log_2 m + 1)$ where "floor" just *means* "round the number down, i.e. it is a definition. Thus, the number of bits in the binary representation of m is always:

$$N = floor(\log_2 m) + 1.$$

This is a rather complicated expression, and I don't expect you to remember it exactly. What I do expect you to remember is that N grows roughly as $\log_2 m$.

Other number representations

Today we have considered only positive integers. Of course, sometimes we want to represent negative integers, and sometimes we want to represent numbers that are not integers, e.g. fractions. These other number representations are taught in COMP 273. If you wish to learn about them now, please have a look at the lecture notes on my COMP 273 public web page.