

Last lecture I finished off by asking how to convert from a decimal representation of a number to a binary representation. There is a simple algorithm for doing this. To understand the algorithm, we need to consider the following basic property of decimal and binary numbers.

Suppose we multiply a number  $m$  by 10, where  $m$  is written in decimal. There is a simple way to get the result,  $10 m$ , namely shift the digits left by one place and put a 0 in the rightmost position. So,  $238 * 10 = 2380$  and the reason is

$$238 * 10 = (2 * 10^2 + 3 * 10^1 + 8 * 10^0) * 10 = 2 * 10^3 + 3 * 10^2 + 8 * 10^1 + 0 * 10^0.$$

Similarly, to divide a number  $m$  by 10, we shift the digits to the right

$$238/10 = (2 * 10^2 + 3 * 10^1 + 8 * 10^0)/10 = 2 * 10^1 + 3 * 10^0$$

We have dropped the rightmost digit 8 (which becomes the remainder) since we are doing integer division, and thus ignoring terms with negative powers of 10 i.e  $8 * 10^{-1}$ .

In binary, the same idea holds. If we represent a number  $m$  in binary and multiply by 2, we shift the bits to the left by one position and put a 0 in the rightmost position. So, e.g. if

$$m = 11010 = 1 * 2^4 + 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0$$

then multiplying by 2 gives

$$110100 = 1 * 2^5 + 1 * 2^4 + 0 * 2^3 + 1 * 2^2 + 0 * 2^1 + 0 * 2^0.$$

If we divide by 2, then we shift right by one position and drop the rightmost bit (which becomes the remainder).

For any positive integer  $m$ , we can write

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

for example,

$$549 = 540 + 9 = 10 * (549/10) + (549 \% 10)$$

where  $\%$  is the mod operator. More generally, if  $m$  is written a sum of powers of a base  $b$ , then

$$m = b * (m/b) + (m \% b),$$

and in particular, in binary  $b = 2$ , so

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

Representing a positive integer  $m$  in binary means that we write it as

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

where  $b_i$  is a bit, i.e either 0 or 1. So we write  $m$  in binary as a bit sequence  $(b_{n-1} b_{n-2} \dots b_2 b_1 b_0)$ .

Combining the above ideas, we write our  $n$ -bit number  $m$  as:

$$m = \sum_{i=0}^{n-1} b_i 2^i = \sum_{i=1}^{n-1} b_i 2^i + b_0 = (2 \sum_{i=1}^{n-1} b_i 2^{i-1}) + b_0 = (2 \sum_{i=0}^{n-2} b_{i+1} 2^i) + b_0.$$

Thus

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

The following algorithm is based on the above idea. We repeatedly divide by 2 and the “remainder” bits gives us our  $b_i$  values.

---

**Algorithm 1** Convert decimal to binary

---

**INPUT:** a number  $m$  expressed in base 10 (decimal)

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

```

i ← 0
while m > 0 do
  b[i] ← m%2
  m ← m/2
  i ← i + 1
end while

```

---

**Example: Convert 241 to binary**

<u>i</u>	<u>m</u>	<u>b[ ]</u>
0	<span style="border: 1px solid black; padding: 2px;">241</span>	
1	120	1
2	60	0
3	30	0
4	15	0
5	7	1
6	3	1
7	1	1
8	0	1
9	0	0
10	:	:

and so the answer is 11110001. Note that there are an infinite number of 0’s on the left which you can truncate.

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. (I did not do this in class, though.) Suppose our number is  $m = 241$ , which is 11110001 in binary.

<u>i</u>	<u>m</u>	<u>b[i]</u>
0	<span style="border: 1px solid black; padding: 2px;">11110001</span>	
1	1111000	1
2	111100	0
3	11110	0
4	1111	0
5	111	1
6	11	1
7	1	1
8	0	1

We see that the remainders are just the bits used in the binary representation of the number.

## Binary fractions

Up to now we have only talked about integers. We next talk about binary representations of fractional numbers, that is, numbers that lie between the integers. Take a decimal number such as 26.375. We write this as:

$$(26.375)_{\text{ten}} = 2 * 10^1 + 6 * 10^0 + 3 * 10^{-1} + 7 * 10^{-2} + 5 * 10^{-3}.$$

The “.” is called the *decimal point*.

One uses an analogous representation using binary numbers, *e.g.*

$$(11010.011)_{\text{two}} = 1 * 2^4 + 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0 + 0 * 2^{-1} + 1 * 2^{-2} + 1 * 2^{-3}$$

where “.” is called the *binary point*. Check for yourself that this is the same number as above, namely

$$16 + 8 + 2 + 0.25 + 0.125 = 26.375.$$

How do we convert from decimal to binary for such fractional numbers in general? In particular, suppose we have a number  $x$  that is written in decimal and has a finite number of digits to the (left and) right of the binary point. Let’s look at a particular example. This is different from the one given in the slides.

Let  $x = 4.63$  and let’s convert it to binary. Since  $x = 4 + .63$ , we know the answer will have the form  $100.\_\_\_$  since  $.63$  is a sum of powers of 2 with negative exponents. So we just need to find the bits to the right of the binary point. To get the first say five bits, we multiply and divide by 2 five times.

$$\begin{aligned} 0.67 &= 1.34 * 2^{-1} \\ &= 2.68 * 2^{-2} \\ &= 5.36 * 2^{-3} \\ &= 10.72 * 2^{-4} \\ &= 21.44 * 2^{-5} \end{aligned}$$

We convert 21 to binary (10101.) and shift the binary point left by five places. Thus  $.67 = .10101\_\_\_$ , and so  $4.63 = 100.10101\_\_\_$ . Note that this is just an approximation.