

1. Convert the following decimal numbers to binary and back again.
 - (a) 34
 - (b) 82
2. The algorithm for addition of two positive integers is as easy with the binary representation as it is with the decimal representation.

Show that $26 + 27 = 53$, by converting 26 and 27 to binary, computing the sum, and then converting back to decimal.

(Hint: you may need to peek at the solution to see what is being asked here.)

3. A *byte* is $n = 8$ bits. When the bits are interpreted as 8 bit numbers, the values range from 0 to 255.

Write out the bytes (binary numbers) that correspond to numbers 127 to 130.

4. Typically when we have binary numbers with n bits (where n is large), we do not write them out as such since bits are difficult to look at and we can easily make mistakes. Instead, it is quite common to group the bits into 4-tuples and represent each possible 4-tuple as a single symbol. Since we have $2^4 = 16$ possibilities, we have a base 16 representation. This representation is called *hexadecimal*. Hexadecimal uses 16 symbols 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F to represent numbers from 0 to 15. Thus, the hexadecimal number $23D$ is, in decimal, $2 \times 16^2 + 3 \times 16 + 13 \times 1 = 317$.

[ASIDE: For example, in HTML colour intensities are often given in hexadecimal, as in **FF FF FF**, the colour white, with $(\text{FF})_{16} = (255)_{10}$, the maximum level of Red, Green, and Blue (RGB). Observe that the base of hexadecimal, 16, is a power of 2, namely 2^4 . Thus a ‘full’ hexadecimal number, like **FF**, corresponds to a ‘full’ binary number, in this case, **11111111**.]

Convert the following hexadecimal numbers to decimal:

- (a) 11
 - (b) 25
 - (c) FF
 - (d) 10A
5. In question 2 above, you had to figure out the addition algorithm for binary. Figure out the addition algorithm for hexadecimal, and find the following sums:
 - (a) $9 + 3$
 - (b) $3D + 10$
 - (c) $FA + D0$
 - (d) $DD + DD$

6. Convert the following binary fractions to decimal.
- (a) 110.1001
 - (b) .0011
7. Convert the following decimal numbers to binary. For (c),(d), you will need to use an approximation.
- (a) 5.5
 - (b) 25.25
 - (c) 2.2
 - (d) 2.3
8. If we have a number m which is an integer,
- (a) how many digits (base 10) do we need to represent this integer?
 - (b) how many bits (base 2) do we need to represent this integer?

Solutions

1. (a) $100010 = 2^5 + 2^1 = 32 + 2$
 (b) $1010010 = 2^6 + 2^4 + 2^1 = 64 + 16 + 2$
2. $00011010 \leftarrow 26$
 $+ \quad \underline{00011011} \leftarrow 27$

To perform the addition, use the same algorithm that you use with decimal, except that you are only allowed 0's and 1's. Whenever the sum in a column is 2 or more, you carry to the next column, since it contributes to the next power of 2:

$$2^i + 2^i = 2^{i+1}$$

So,

$$\begin{array}{r} 00110100 \quad \leftarrow \text{carry bits} \\ 00011010 \quad \leftarrow 26 \\ + \quad \underline{00011011} \quad \leftarrow 27 \\ \hline 00110101 \quad \leftarrow 53 \end{array}$$

In fact, this is basically how computers do arithmetic as you will learn in COMP 273 (or ECSE 221).

3. 127 is 01111111, 128 is 10000000, 129 is 10000001, 130 is 10000010
4. (a) 17
 (b) 37
 (c) 255
 (d) 266
5. (a) C
 (b) 4D
 (c) 1CA
 (d) 1BA
6. (a) $2^2 + 2^1 + 2^{-1} + 2^{-4} = 4 + 2 + .5 + .0625 = 6.5625$
 (b) $2^{-3} + 2^{-4} = .125 + .0625 = .1875$
7. (a) 101.1
 (b) 11001.01

- (c) You need an approximation here since .2 cannot be written as a finite sum of powers of 2. (You may be surprised to realize this.) To get such an approximation, we use the trick we saw in lecture 2. Here I describe it slightly differently. We rewrite:

$$2.2 = 2.2 * 2^n / 2^n$$

for some chosen n (which will correspond to the number of bits to the right of the binary point). Let's choose $n = 5$. Let

$$2.2 * 2^5 / 2^5 = (2.2 * 32) / 2^5 = 70.4 / 2^5$$

where we calculated $2.2 * 32 = 70.4$ "by hand", i.e. using our grade school method for multiplication.

So we have re-expressed our problem: we want to approximate $70.4 / 2^5$ in binary. Noting that

$$70 < 70.4 < 71$$

we see that

$$(70.4)_{ten} = (\underline{\hspace{1cm}}.\underline{\hspace{1cm}})_{two}$$

where the part to the left of the binary point is 70 (in binary) and the part to the right of the binary point is 0.4 in binary. For our approximation, we will chop off the part to the right of the point.

Converting 70 to binary (using the algorithm given in class) gives 1000110 and so 70.4 must be $(1000110.\underline{\hspace{1cm}})_{two}$ for some unknown part on the right side which is strictly less than 1. Our approximation is to chop off the part to the right of the binary point. So, $70.4 \approx (1000110)_{two}$.

To finish our approximation of 2.2, we must remember to divide our approximation of 70.4 by 2^5 , which (by inspection) means shifting the binary point to the left by 5 places. So, we get

$$2.2 \approx (10.00110)_{two}$$

Notice that we can get an approximation to as much precision as we want by choosing the number of bits n to be as large as we want and then re-doing the above.

- (d) Let's approximate this one to 6 bits of precision (beyond the binary point). Then

$$2.3 = 2.3 * 2^6 / 2^6 = 147.2 / 2^6$$

But 147 is $(10010011)_{two}$, so $2.3 \approx 10.010011$ where we have chopped off all terms of power $2^{-7}, 2^{-8}, \dots$

8. (a) We need to consider the biggest power of ten greater than or equal to m and then take the \log_{10} of that number. Since $\log_{10} m$ is typically not an integer though, we need to round off. By inspection, we need $\lfloor \log_{10} m \rfloor + 1$ digits, where the notation $\lfloor \]$ means "floor" or rounding down. There is a corresponding notation $\lceil \]$ which means "ceiling" or rounding up.

Note that the number of digits to represent m is not $\lceil \log_{10} m \rceil$, in general, for example take $m = 1, 10, 100, \dots$

- (b) Similarly, we need $\lfloor \log_2 m \rfloor + 1$ bits.