

# BINARY NOTATION\*

Don Johnson

This work is produced by The Connexions Project and licensed under the  
Creative Commons Attribution License †

## Abstract

A short course on our friend, Base 2.

We need to concentrate on how computers represent numbers and perform arithmetic. Suppose we have twenty-six "things." This number is an abstract quantity that we need to represent so that we can perform calculations on it. We represent this number with **positional notation** using base 10:  $26 = 2 \times 10^1 + 6 \times 10^0$ . We could also use base-8 notation ( $32_8 = 3 \times 8^1 + 2 \times 8^0$ ), base 16 ( $1A_{16} = 1 \times 16^1 + A_{16} \times 16^0$ , with  $A_{16}$  representing ten), or base 2 ( $11010_2 = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$ ). Each of these represent the **same** quantity; all numbers can be so represented and arithmetic can be performed in each system. Which we chose is a matter of convenience and cultural heritage. Each position is occupied by an integer between 0 and the base minus 1, and the integer's position implicitly represents how many times the base raised to an exponent is needed. The integer  $N$  is succinctly expressed in base- $b$  as  $N_b = d_n d_{n-1} \dots d_0$ , which mathematically means  $N_b = d_n b^n + \dots + d_0 b^0$ . No matter what base you might choose, addition, subtraction, multiplication, and division all follow the same rules you learned as a child. These rules can be derived mathematically from the positional notation conventions defined by this formula. (Non-positional systems are very strange; take Roman numerals for example. Try adding (or, more extremely, dividing) two numbers using this number representation system). To extend the positional representation to signed integers, we add a symbol that represents whether the number is positive or negative.

Humans have ten fingers, and so it's not so surprising that many cultures throughout history have used base 10. Digital computers use base 2 or **binary** number representation, each digit of which is known as a **bit** (binary dig it). Here, each bit is represented as a voltage that is either "high" or "low," thereby representing "1" or "0", respectively. To represent signed values, we tack on a special bit — the **sign bit** — to express the sign. The binary addition and multiplication tables are

$$0 + 0 = 0 \tag{1}$$

$$1 + 1 = 10$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$0 \times 0 = 0$$

---

\*Version 2.7: Jul 22, 2005 10:52 am GMT-5

†<http://creativecommons.org/licenses/by/1.0>

$$0 \times 1 = 0$$

$$1 \times 1 = 1$$

$$1 \times 0 = 0$$

A **carry** means that a computation performed at a given position affects other positions as well. Here,  $1_2 + 1_2 = 10_2$  is an example of a computation that involves a carry. Note that if carries are ignored, subtraction of two single-digit binary numbers yields the same bit as addition. Computers use high and low voltage values to express a bit, and an array of such voltages express numbers akin to positional notation. Logic circuits perform arithmetic operations.

**Exercise 1***(Solution on p. 3.)*

Add twenty-five and seven in base 2. Note the carries that might occur. Why is the result "nice"?

## Solutions to Exercises in this Module

### Solution to Exercise 1 (p. 2)

$25 = 11001_2$  and  $7 = 111_2$ . We find that  $11001_2 + 111_2 = 100000_2 = 32$ .