Math From Scratch Lesson 12: Bases Other Than 10

W. Blaine Dowler

December 8, 2010

Contents

1 Other (Relatively) Common Bases

In our original bases representation theorem, there were only two conditions imposed on k:

1. $k \in \mathbb{N}$
2. $k > 1$

Ten is certainly *not* the only number which satisfies these two conditions. While the overwhelming majority of work in this world is done to base ten, there are some fields (notably computer science) in which other bases are used. We will examine three of those bases now.

Before moving forward, notation must be established. If we are writing numbers outside the usual base ten system, how do we represent the digits? Do we create new symbols specifically for each new base? Well, while that might eliminate ambiguity, it is a royal pain and is not the system which has been adopted. For cases in which $k < 10$, we simply use the symbols we have for the first $k-1$ digits, and then move to 10 after that. (For example, in base eight, $7+1=10$.) For cases in which $k > 10$, we need to use new symbols, typically drawn from the alphabet, such that $9+1=a$, $a+1=b$, and so forth. While this allows for familiar symbols to be used, it can also cause confusion. So, for the course of this lesson, subscripts indicating k will be used to eliminate ambiguity. For example, we can express a single number in base 16, 10, 8 and 2 as 100_{16}, 256_{10}, 400_8 and 100000000_2 respectively.

1.1 Base 2 - Binary

The base 2 or binary system is probably the uncommon base which garners the most attention from professional mathematicians, particularly as it relates to logic and logical systems. The fact that it allows only two possible values for each digit is surprisingly useful in defining Boolean algebra. However, binary number systems are best known for their applications to computer science. Mathematically, all of the groundwork for basic operations has already been laid for all bases. Nonetheless, there are two significant and common misconceptions based on computer science applications which I feel inclined to address.

Firstly, it is often believed that binary numbers must have eight digits (or a multiple of eight.) This is not the case from the mathematical perspective. Rather, it is a side effect of computer science using "bytes" as the fundamental size for data storage. It is a byte which must have eight digits. Numbers too large to fit in eight binary digits are assigned two bytes, or three bytes, etc. and, therefore, end up with a multiple of eight digits. They are thus expressed with leading zeroes, so that the mathematical binary number 1111_2 is expressed as 00001111 in a computer's memory. This is a necessary component of the computer science involved, but it is not a mathematical requirement.

The second misconception relates to the expression of negative numbers. While mathematicians simply use the $-$ symbol in front of a number as usual, computer scientists have no such luxury. Computer scientists are forced to use a single digit, or bit, to express the difference between positive and negative. They typically take the "high" bit, or leftmost bit in an eight digit representation, to express this quantity. Thus, in computer science, 10001111 may be interpreted as either $128_{10} + 8_{10} + 4_{10} + 2_{10} + 1_{10} = 143_{10}$, or as -113_{10}, depending upon whether the programmer informed the computer that negative numbers were allowed or not. Notice that this is most decidedly *not* -143_{10}! While that representation is possible, and has been implemented in some systems, computer science is far easier if negative numbers and positive numbers can be added using a single algorithm, regardless of sign. Thus, a different tack is taken.

A side effect of eight digit representation is that, should the sum of two numbers require regrouping into the ninth digit, that digit is lost.[1] So, the sum of these two numbers is calculated (with the computer science leading zeroes) as:

$$\begin{array}{rr} & 10001111 \\ + & 01110001 \\ \hline & 00000000 \end{array}$$

where numbers are regrouped noting that $1_2 + 1_2 = 10_2$. In other words, if negative numbers are not permitted by the programmer, these two numbers will add up to $256_{10} = 2^9$, which overflows the eight digit allowance and is erroneously interpreted as 0. If negative numbers are allowed, they still overflow, but the result is correctly interpreted as 0.

Failure to recognize this overflow (or an underestimation of a player's skill and/or ingenuity) has led to bugs in some video games. Perhaps most famously, the original *Super Mario Bros.* for either the Famicom or Nintendo Entertainment System (depending upon the reader's home continent) had an issue with the player's lives. In world 3-1, it is possible to repeatedly bounce of the shell of a turtle on a staircase like structure near the end of the level, escalating in points until one gains additional lives. That game counted *extra* lives using a variable which was allowed to be negative. If a player had zero extra lives and died, the game subtracted an extra life, reducing 0 to -1, and the game ended. However, there was no defense against a player accumulating over 127 lives, which would then be interpreted as negative, so that the loss of one of 130 extra lives also ended the game. The highest "safe" value for the lives one could store in reserve was $10000000 = 128_{10}$, which was interpreted by the computer as -128_{10}. Although negative, this was still safe to have, as the deduction of one life resulted in $01111111 = 127_{10}$ extra lives remaining, which was a positive number, so the game continued. Monitoring this was difficult on the part of the player, as the programmers were not prepared for a number of lives in excess of 99 at all, resulting in symbols appearing in place of the expected digits in the life display, making it almost impossible for the player to track extra lives without manually building a symbol key and counting manually.[2] Later releases of the game have corrected this bug, allowing up to 255 extra lives, presumably by checking to see if the count of extra lives is 0 before deducting for loss of life, rather than checking to see of the count is negative after the deduction.

A similar bug is reported to be present in Namco's *Pac-Man*, in which the game would end if a player could successfully navigate over 255 levels of game

[1]Naturally, if the numbers being added are two bytes each instead of one, the digit will not be lost unless it is carried into the seventeenth digit, and so on.

[2]The author can offer personal assurance that this is entirely possible, though tedious and time consuming. He can also offer the assurance that, by the time the player has obtained enough practice to read these symbols as if they were standard numbers, the player will not require having so many extra lives in the first place.

play. The author hasn't come remotely close to offering a personal assurance on that one.[3]

1.2 Base 8 - Octal

A decreasingly common base in computer science is base eight, known as *octal.* Although everything internally handled by computers is to base 2, the numbers can get a little on the long side when being expressed on paper. (For example, $1024_{10} = 10000000000_2$, a full 7 digits longer. The smallest possible 25 digit binary number is an eight digit number in base 10.) To cope with this, programmers have adopted bases which can be obtained by calculating 2^n for some natural number n. In the case of $n = 3$, we obtain octal. The largest number one can fit in one byte of data is $255_{10} = 377_8$. In fact, generally speaking, with two bases j and k, if $j < k$, then the number of digits used to represent numbers in base j is at least as large as the number of digits in base k. Thus, using a base less than 10 means losing the economy of representation, which is likely a contributing factor in octal's supplantation by our next system.

1.3 Base 16 - Hexadecimal

The most common alternate base used by programmers, next to binary, is hexadecimal, or base $16_{10} = 2^4$. If one uses software that exposes the explicit contents of a computer file as they are written to the drive or stored in memory, rather than as they are interpreted by the appropriate software, the contents are typically represented in this fashion. The largest number which can be stored in one byte is $255_{10} = \text{ff}_{16}$, which is clearly a smaller number of digits than the base ten representation.[4] This one is economical, which may be a large factor in its popularity.

The author is compelled to point out another misconception. With hexadecimal numbers, there is the perception amongst some that digits come in pairs and are written with a space between them, so that $256_{10} = 00\ 01_{16}$. Again,

[3]The author's personal best performance is approximately 245 levels short of this record.

[4]It is worth noting that, in some conventions, the digits are represented by capital letters rather than lowercase letters. Again, this convention is borne of computer science rather than mathematics. The author suspects but has not confirmed that it dates back to the earlier generation of computers which supported capital letters but not lowercase letters. As an aside, when people were deciding whether to include the capital or lowercase letters in computers that could only support one or the other, the decision was based on religious reasons. Research had shown that using exclusively lowercase was easier to read than using exclusively capital letters, but the creators refused to allow a system that did not capitalize the name of their chosen deity, and so capital letters were selected instead.

Binary	Octal	Base 10	Hexadecimal	Number
0	0	0	0	
1	1	1	1	●
10	2	2	2	●●
11	3	3	3	●●●
100	4	4	4	●●●●
101	5	5	5	●●●●●
110	6	6	6	●●●●●●
111	7	7	7	●●●●●●●
1000	10	8	8	●●●●●●●●
1001	11	9	9	●●●●●●●●●
1010	12	10	a	●●●●●●●●●●
1011	13	11	b	●●●●●●●●●●●
1100	14	12	c	●●●●●●●●●●●●
1101	15	13	d	●●●●●●●●●●●●●
1110	16	14	e	●●●●●●●●●●●●●●
1111	17	15	f	●●●●●●●●●●●●●●●
10000	20	16	10	●●●●●●●●●●●●●●●●

Table 1: The first 17_{10} whole numbers in multiple bases.

this is a computer science side effect. Many computers store the bytes relating to lower digits earlier in the file than the bytes relating to higher digits. Mathematically, $256_{10} = 100_{16}$ in the natural order without additional spaces.

2 Example Representations

For a more concrete set of examples, the first seventeen whole numbers are shown with their representations in table 1.

3 Upcoming Lessons

We are now almost fully equipped to come up with a useful definition of division, and to start discussing factors and divisibility tests with modular arithmetic.

This content will occupy the lessons for most of the remaining year. After that has been established, we can complete the field axioms and define the system of rational numbers, including fractions, and begin examining the decimal representations of fractions. We will also be able to determine through inspection which fractions will result in terminating decimals, and which will result in repeating decimals.