K

keenonscience

Positional Notation and Base Conversion: An Algebraic Approach

The successive-division algorithm and the irrelevance of base ten

An in-depth guide bridging elementary algebra and number theory

by **Nicola Maglione** – keenonscience

@keenonmaths

September 2026

Contents

Introduction 2

1 Positional notation 3

2 Existence and uniqueness of the representation 4

3 The successive-division method 6

4 The starting base is irrelevant 8

5 Operational comparison: direct versus via base ten 9

6 Direct conversion between arbitrary bases 10

7 An application: bytes, binary, and hexadecimal 12

Bibliography 14

Introduction

Take a number written in a non-decimal base – say $(253)_7$ – and ask for it to be rewritten in another base, also non-decimal, say base 4. The problem looks harmless: the most common reaction is to convert the number to base ten first, the base one is most comfortable with, and proceed from there to the required base.

This route always works, but it raises a question worth pursuing: *is it actually necessary*? The number $(253)_7$ represents a well-defined quantity, one that exists independently of how it is written. Changing base means changing the representation of that quantity, not carrying out a computation that must necessarily transit through some privileged base.

This article shows that the successive-division algorithm – the same one used to convert a decimal number into binary – works in exactly the same way regardless of the starting base and the target base, with no intermediate base required. Base ten plays no special algebraic role in this mechanism: it is convenient because it is the base we are used to computing in, but it is by no means necessary.

The path is as follows. Section 1 formalises positional notation. Section 2 proves that every integer admits, in every base, a positional representation, and that this representation is unique; the proof, constructive, introduces the successive-division algorithm. Section 3 applies this algorithm directly to the desired target base, with the article's central worked example. Section 4 shows why the starting base – base ten in particular – is irrelevant to the mechanism. Section 5 quantitatively compares the direct method with passing through base ten. Section 6 generalises everything to arbitrary bases. Section 7 closes with an immediate computing application: binary–hexadecimal conversion of a byte.

1. Positional notation

Definition 1.1 (positional representation in base b). *Let $b \geq 2$ be an integer, called the **base**. An integer $N \geq 0$ admits a **positional representation in base** b if there exist an integer $k \geq 0$ and digits $d_0, d_1, \ldots, d_k$, with*

$$0 \leq d_i < b \quad \text{for every } i = 0, \ldots, k,$$

such that

$$N = \sum_{i=0}^{k} d_i \, b^i. \tag{1}$$

One then writes $N = (d_k \, d_{k-1} \cdots d_1 \, d_0)_b$.

Remark. Digit d_i carries weight b^i: the position of each digit determines its weight, hence the name "positional notation". For $b = 10$ one recovers ordinary decimal notation, normally written without the subscript.

Example 1.1. Consider the number written in base 7 as $N = (253)_7$. Applying Definition 1.1 with $b = 7$, $d_2 = 2$, $d_1 = 5$, $d_0 = 3$ (all digits are admissible, since $0 \leq 2, 5, 3 < 7$):

$$N = 2 \cdot 7^2 + 5 \cdot 7^1 + 3 \cdot 7^0 = 2 \cdot 49 + 5 \cdot 7 + 3 = 98 + 35 + 3 = 136.$$

So $(253)_7$ admits, in base ten, the value 136. We will meet this same number again as the central example of Sections 3–5.

2. Existence and uniqueness of the representation

Theorem 2.1 (existence and uniqueness). *Let $b \geq 2$ be an integer. Every integer $N \geq 0$ admits a positional representation in base b in the sense of Definition 1.1, and this representation is unique (up to leading zero digits).*

Proof.

Part 1 – Existence. We proceed by strong induction on N.

Base case ($N = 0$). Set $k = 0$ and $d_0 = 0$: then $\sum_{i=0}^{0} d_i b^i = 0 \cdot b^0 = 0 = N$, and the digit $d_0 = 0$ trivially satisfies $0 \leq d_0 < b$ (since $b \geq 2 > 0$). The base case holds.

Inductive step. Let $N > 0$, and suppose (inductive hypothesis) that every integer M with $0 \leq M < N$ already admits a positional representation in base b. Apply Euclidean division of N by b: there exist unique integers q, d_0 such that

$$N = q\,b + d_0, \qquad 0 \leq d_0 < b, \tag{2}$$

(this is an elementary fact of integer arithmetic: the Euclidean division theorem always guarantees existence and uniqueness of quotient and remainder, for any dividend $N \geq 0$ and any divisor $b \geq 1$).

We want to apply the inductive hypothesis to the quotient q, and to do so we must first check that $0 \leq q < N$. From (2), since $d_0 \geq 0$, we have $q\,b \leq N$, hence $q \leq N/b$; and since $b \geq 2$, $N/b \leq N/2 < N$ (the last inequality holds because $N > 0$). Thus $q < N$, and clearly $q \geq 0$ as the quotient of two non-negative integers: the inductive hypothesis applies to q.

By the inductive hypothesis, there exist an integer $k' \geq 0$ and digits $e_0, \ldots, e_{k'}$, with $0 \leq e_i < b$, such that $q = \sum_{i=0}^{k'} e_i b^i$. Substituting into (2):

$$N = qb + d_0 = b \sum_{i=0}^{k'} e_i b^i + d_0 = \sum_{i=0}^{k'} e_i b^{i+1} + d_0 b^0 = \sum_{i=1}^{k'+1} e_{i-1} b^i + d_0 b^0.$$

Setting $k := k' + 1$ and $d_i := e_{i-1}$ for $i = 1, \ldots, k$ (keeping d_0 as already defined by the Euclidean division), we obtain $N = \sum_{i=0}^{k} d_i b^i$ with $0 \leq d_i < b$ for every i: for $i = 0$ by construction, for $i \geq 1$ because $d_i = e_{i-1}$ satisfies $0 \leq e_{i-1} < b$ by the inductive hypothesis. This completes the inductive step, and with it the proof of existence for every $N \geq 0$. ■ (Part 1)

Part 2 – Uniqueness. We proceed by contradiction, showing that two distinct representations of the same N cannot exist: the contradiction already emerges at the least significant digit, and propagates by induction to the subsequent digits.

Suppose, for contradiction, that N admits two distinct positional representations in base b,

$$N = \sum_{i=0}^{k} d_i b^i = \sum_{i=0}^{k} e_i b^i, \qquad 0 \leq d_i, e_i < b,$$

(up to padding the shorter one with leading zero digits, we may always assume both have the same number $k+1$ of digits; this does not change the represented value), with $(d_0, \ldots, d_k) \neq (e_0, \ldots, e_k)$.

Base step – the least significant digit. Reduce both sides modulo b. For every $i \geq 1$, the term $d_i b^i$ (and likewise $e_i b^i$) is a multiple of b, since b^i is; these terms vanish modulo b, leaving only the $i = 0$ contribution:

$$d_0 \equiv N \equiv e_0 \pmod{b}.$$

Since $0 \le d_0, e_0 < b$, and the interval $[0, b)$ contains exactly one representative of each residue class modulo b (this is, again, uniqueness of the remainder in Euclidean division), we conclude $d_0 = e_0$.

Inductive step – the subsequent digits. Subtracting $d_0 = e_0$ from both sides of the original equality and dividing by b (valid: every remaining term, for $i \ge 1$, is a multiple of b) gives, for the number $N' := (N - d_0)/b$, a new pair of representations with one fewer digit:

$$N' = \sum_{i=1}^{k} d_i b^{i-1} = \sum_{i=0}^{k-1} d_{i+1} b^i \qquad \text{and} \qquad N' = \sum_{i=0}^{k-1} e_{i+1} b^i.$$

Applying the same argument as the base step to this pair (with N' in place of N) gives $d_1 = e_1$; iterating on $N'' := (N' - d_1)/b$ gives $d_2 = e_2$, and so on. After $k+1$ applications of this argument (formally, by induction on the number of remaining digits, with an immediate base case: a number with a single digit is, trivially, that digit) we obtain $d_i = e_i$ for every $i = 0, \dots, k$.

But then $(d_0, \dots, d_k) = (e_0, \dots, e_k)$, contradicting the assumption that the two representations were distinct. ■ (Part 2)

Connection with the practical method (Section 3). The proof of Part 1 is, digit by digit, the description of an algorithm: at each step one performs a single Euclidean division by b (equation (2)), records the remainder as a digit, and repeats the procedure on the quotient – exactly what Section 3 isolates and calls the *successive-division method*. The Theorem guarantees that this procedure always terminates after finitely many steps (the quotient strictly decreases at each step, as shown in the inductive step of Part 1) and produces the only possible representation (Part 2): there is thus no need to ask whether a different choice of digits, or a different order, could have given a different result.

Example 2.1. Let us apply, step by step, the algorithm (2) just used in the proof, to convert $N = 47$ into base $b = 3$:

$$\begin{aligned}
47 &= 15 \cdot 3 + 2 && \Rightarrow\ d_0 = 2,\\
15 &= 5 \cdot 3 + 0 && \Rightarrow\ d_1 = 0,\\
5 &= 1 \cdot 3 + 2 && \Rightarrow\ d_2 = 2,\\
1 &= 0 \cdot 3 + 1 && \Rightarrow\ d_3 = 1,\ N_4 = 0:\ \text{we stop.}
\end{aligned}$$

Reading the digits in the order $d_3 d_2 d_1 d_0$ (that is, in *reverse* order with respect to extraction, from the last remainder computed to the first):

$$\mathbf{47 = (1202)_3}$$

Check: $1 \cdot 27 + 2 \cdot 9 + 0 \cdot 3 + 2 \cdot 1 = 27 + 18 + 0 + 2 = 47$. ✓

3. The successive-division method

The constructive proof of Theorem 2.1 is, in effect, already a ready-to-use algorithm: to convert an integer N into the target base b, apply (2) repeatedly starting from $N_0 = N$, until a zero quotient is reached. It is worth isolating and formalising it explicitly.

Successive-division method. *To convert $N \geq 0$ into base b: set $N_0 = N$ and, for $j = 0, 1, 2, \ldots$, compute $N_j = q_j\, b + d_j$ (quotient and remainder of N_j divided by b), setting $N_{j+1} = q_j$, until $N_{k+1} = 0$. The digits $d_0, \ldots, d_k$ obtained – read **in reverse order** of extraction – give $N = (d_k \cdots d_1 d_0)_b$.*

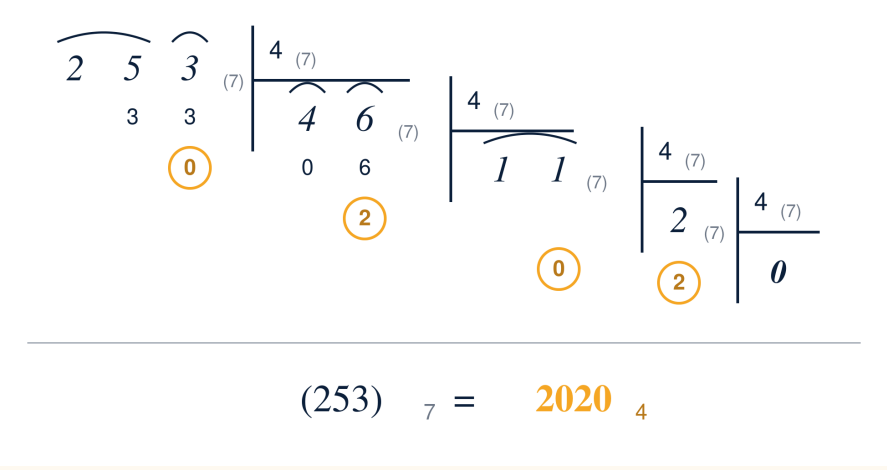
Remark. The method produces exactly the representation guaranteed by Theorem 2.1: it is the very same construction used in the existence proof, here simply isolated as a stand-alone procedure. The digits emerge in *reverse* order relative to how Definition 1.1 indexes them, because d_0 (the lowest-weight digit, b^0) is the *first* remainder computed, while d_k (the highest-weight digit) is the *last*.

Example 3.1. Let us take again the number from Example 1.1: $N = (253)_7$. We want to convert it **directly** into base $b = 4$, applying the method just formalised – but this time entirely in **base-7 arithmetic**: dividend, quotient and remainder, at every step, are written in base 7, and long division follows the rules of base 7 (a borrow from one digit is worth 7 units of the next digit, not 10). Since $4 < 7$, the divisor $(4)_7$ remains a single digit: the long division reduces, at each step, to the usual single-digit-divisor procedure, applied however to numbers expressed in base 7.

For convenience, here are the multiples of $(4)_7$ needed in the steps below (obtained by repeated addition in base 7):

d	0	1	2	3	4	5	6
$(4)_7 \times d$	$(0)_7$	$(4)_7$	$(11)_7$	$(15)_7$	$(22)_7$	$(26)_7$	$(33)_7$

Native base-7 long division of $(253)_7$ by $(4)_7$. The diagram below chains the four successive divisions into a single staircase: the little arcs at the top show, at each step, how digits are brought down and grouped for comparison with the multiples of $(4)_7$; the remainder of each step, circled, is a digit of the result; the quotient (in bold) becomes the dividend of the next step.



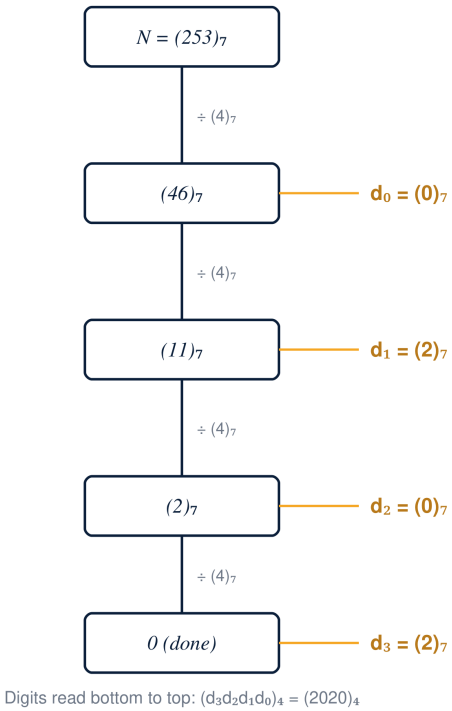


Figure 1: *Tree of successive divisions, entirely in base-7 arithmetic, for the direct conversion of* $N = (253)_7$ *into base* 4*: the root is the starting number, each vertical branch is a division by* $(4)_7$*, each side branch (in orange) the corresponding remainder, which becomes a result digit.*

Reading the digits in the order $d_3d_2d_1d_0$ (Section 3):

$$\mathbf{(253)_7 = (2020)_4}$$

(Check, as a final sanity test: $(253)_7$ equals 136 in base ten – Example 1.1 – and $2 \cdot 4^3 + 0 \cdot 4^2 + 2 \cdot 4^1 + 0 \cdot 4^0 = 128 + 0 + 8 + 0 = 136$. ✓)

This conversion required exactly **4 divisions** (one for each digit produced in base 4), carried out entirely in base-7 arithmetic: no base other than 7 (for the starting point) and 4 (for the result) ever appeared in the computation – not even implicitly, as an intermediate value.

4. The starting base is irrelevant

The successive-division method, as formalised in Section 3, operates exclusively on the *value* of N: Definition 2.1 (algorithm (2)) never refers to the base in which N was originally written. N is an object – a quantity – that exists independently of its representation: $(253)_7$ and 136 are two different names for the same number, just as 47 and $(1202)_3$ from Example 2.1 are. Changing base changes the name, not the object.

It follows immediately that the direct method works regardless of the starting base – **including, trivially, base ten itself**. And here the question anticipated in the Introduction arises: if so, why is the "via base ten" method, which does always work, not necessary?

The answer is that base ten, in this mechanism, enjoys no distinctive algebraic property: it is simply another possible target base among the infinitely many admitted by Theorem 2.1, no more and no less privileged than 4, 7, or 9. By transitivity of equality – the same N, however it is computed, remains the same N – converting N to base ten and then from base ten to base b necessarily produces the same result as converting directly to base b: both procedures do nothing more than apply Theorem 2.1 (in its uniqueness) to the same quantity N. This is not a lucky coincidence, but a direct consequence of the uniqueness proved in Section 2.

Example 4.1 (the same example, via base ten). Let us take again $N = (253)_7 = 136$ from Example 3.1, and convert it to base 4 explicitly via base ten.

Step A – 136 in base ten. Apply the successive-division method with target base 10:

$$\begin{aligned} 136 &= 13 \cdot 10 + 6 && \Rightarrow\ d_0 = 6, \\ 13 &= 1 \cdot 10 + 3 && \Rightarrow\ d_1 = 3, \\ 1 &= 0 \cdot 10 + 1 && \Rightarrow\ d_2 = 1,\ N_3 = 0 :\ \text{we stop.} \end{aligned}$$

That is, $136 = (136)_{10}$ – obviously, since 136 is already written in base ten. The point, however, is that *this* step too is, to all effects, an application of Theorem 2.1: it took **3 divisions** to produce it.

Step B – from base ten to base 4. Now convert 136 (whose starting base is, this time, 10) into base 4:

$$136 = 34 \cdot 4 + 0, \quad 34 = 8 \cdot 4 + 2, \quad 8 = 2 \cdot 4 + 0, \quad 2 = 0 \cdot 4 + 2,$$

i.e. exactly the **same 4 divisions** already carried out in Example 3.1, with the same result $(2020)_4$.

Remark. Step B is nothing but a literal repetition of the computation from Section 3: knowing that 136 "is written in base ten" made that division neither simpler nor different. The only divisions actually necessary for the final result are those in Step B; those in Step A are, as quantified in the next section, an additional cost.

5. Operational comparison: direct versus via base ten

Let us count the divisions required by each of the two routes on the example from Sections 3–4:

Method	Divisions	Total
Direct ($136 \to$ base 4, Example 3.1)	4	4
Via base ten ($136 \to$ base 10, then base $10 \to$ base 4, Example 4.1)	$3 + 4$	7

The direct method saves exactly the 3 divisions of Step A – a step which, as shown in Section 4, contributes nothing to the final result in base 4.

In general, if N admits k_{10} digits in base ten and k_b digits in the target base b, the direct method requires k_b divisions, while the via-base-ten method requires $k_{10} + k_b$: an overhead of exactly k_{10} divisions, which depends only on the size of N and grows independently of the target base b. The larger N is (and hence the more decimal digits it has), the greater the waste of passing through base ten before reaching the desired base.

6. Direct conversion between arbitrary bases

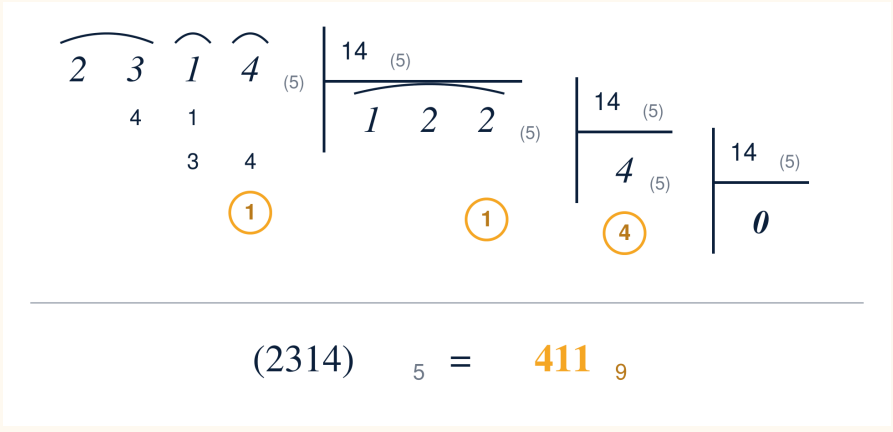
Nothing in the argument of Sections 3–4 depended on the specific values 7 and 4: the same principle applies to a direct conversion between any two bases a and b, $a, b \geq 2$, with no need to pass through base ten or through any other intermediate base. One only needs the value of N (obtained by applying Definition 1.1 to its representation in base a), to which the successive-division method with target base b is then applied.

Example 6.1. Let $N = (2314)_5$, a number written in base 5. We want to convert it directly into base 9. Note that 9, written in base 5, is $(14)_5$ (since $9 = 1 \cdot 5 + 4$): the divisor is thus expressed with **two digits** in base 5, and long division requires – as with the usual division by a two-digit divisor in base ten – estimating the quotient digit, followed by multiplication and subtraction (with a possible borrow), all in base-5 arithmetic.

Multiples of $(14)_5$ useful for the estimation (computed by repeated addition in base 5):

d	1	2	3	4
$(14)_5 \times d$	$(14)_5$	$(33)_5$	$(102)_5$	$(121)_5$

Native base-5 long division of $(2314)_5$ by $(14)_5$. The diagram below chains the three rounds into a single staircase, with the same logic as Section 3: at each sub-step, the minimum number of leading digits needed to reach the divisor is grouped (a wide arc, as on "23" and on the whole of "122" at the second step), or a single digit is brought down at a time when the first digit is already enough on its own (separate small arcs, as on "1" and "4" in the first step); the remainder of each sub-step is rewritten next to the following digit brought down, forming the new value to be divided. The difference from Section 3 is that the divisor here has two digits: each sub-step therefore requires estimating the quotient digit from the table of multiples above, rather than an immediate choice.



Reading the digits in the order $d_2 d_1 d_0$ (Section 3):

$$\mathbf{(2314)_5 = (411)_9}$$

Check (entirely in base 5, reconstructing the last non-trivial round backwards): quotient times divisor plus remainder must give back the dividend of that round, i.e. $(4)_5 \times (14)_5 + (1)_5$. From the table, $(4)_5 \times (14)_5 = (121)_5$; adding the remainder, $(121)_5 + (1)_5 = (122)_5$ (units $1 + 1 = 2$, the other digits unchanged) – exactly the N_1 found in the first round. ✓

Once again no intermediate base appeared in the computation: from base 5 to base 9, directly – this time with a multi-digit divisor, a case that Section 3 (where the divisor had a single digit) had not yet presented.

7. An application: bytes, binary, and hexadecimal

Base ten admits, as such, no privileged mathematical role: by Theorem 2.1, every integer $b \geq 2$ provides an equally legitimate positional system. The choice of base ten is a **cultural** convention, plausibly tied to the number of fingers on human hands, not an algebraic property of the number 10.

An immediate application of this principle, particularly natural in computing, is conversion between base **2** (binary) and base **16** (hexadecimal) for a byte.

Example 7.1. The byte $(10110110)_2$ admits, by Definition 1.1, the value

$$1 \cdot 128 + 0 \cdot 64 + 1 \cdot 32 + 1 \cdot 16 + 0 \cdot 8 + 1 \cdot 4 + 1 \cdot 2 + 0 \cdot 1 = 128 + 32 + 16 + 4 + 2 = 182.$$

Direct conversion of 182 **into base** 16**.**

$$\begin{aligned} 182 &= 11 \cdot 16 + 6 && \Rightarrow\ d_0 = 6, \\ 11 &= 0 \cdot 16 + 11 && \Rightarrow\ d_1 = 11 = (\mathrm{B})_{16},\ N_2 = 0 :\ \text{we stop.} \end{aligned}$$

$$\mathbf{(10110110)_2 = (B6)_{16}}$$

Check: $11 \cdot 16 + 6 = 176 + 6 = 182$. ✓

The shortcut commonly used to convert between binary and hexadecimal – grouping binary digits four at a time – is not an isolated trick: it is itself a special case of Theorem 2.1, and deserves a formal algebraic proof, not just a numerical illustration.

Proposition 7.1 (grouping into blocks of 4). *Let $N = (b_n \cdots b_1 b_0)_2 = \sum_{i=0}^{n} b_i 2^i$ be a number written in base 2, with $0 \leq b_i < 2$, and suppose the number of binary digits $n+1$ is a multiple of 4, i.e. $n+1 = 4m$ for some integer $m \geq 1$ (see the Note on padding below for the general case). Then*

$$N = \sum_{k=0}^{m-1} c_k \, 16^k, \qquad c_k := \sum_{j=0}^{3} b_{4k+j} \, 2^j,$$

and each c_k satisfies $0 \leq c_k \leq 15$: the c_k are thus exactly the hexadecimal digits of N.

Proof. Every index $i \in \{0, \ldots, n\} = \{0, \ldots, 4m-1\}$ can be written uniquely as $i = 4k + j$ with $k = \lfloor i/4 \rfloor \in \{0, \ldots, m-1\}$ and $j = i \bmod 4 \in \{0, 1, 2, 3\}$ – itself an instance of Euclidean division by 4 (Theorem 2.1). Rewriting the sum with this change of index:

$$N = \sum_{i=0}^{n} b_i 2^i = \sum_{k=0}^{m-1} \sum_{j=0}^{3} b_{4k+j} \, 2^{4k+j} = \sum_{k=0}^{m-1} \left(\sum_{j=0}^{3} b_{4k+j} 2^j \right) 2^{4k} = \sum_{k=0}^{m-1} c_k \, (2^4)^k = \sum_{k=0}^{m-1} c_k \, 16^k,$$

where we collected $2^{4k} = (2^4)^k = 16^k$ and set $c_k := \sum_{j=0}^{3} b_{4k+j} 2^j$.

It remains to show $0 \leq c_k \leq 15$. Since $b_i \in \{0, 1\}$ for every i, we have $c_k = b_{4k} + 2b_{4k+1} + 4b_{4k+2} + 8b_{4k+3} \leq 1 + 2 + 4 + 8 = 15$ (the maximum is reached when all four bits of the block equal 1), and clearly $c_k \geq 0$: each c_k is thus an admissible digit in base 16 (Definition 1.1). By Theorem 2.1 (uniqueness), since $N = \sum_k c_k 16^k$ is a positional representation of N in base 16 with admissible digits, it is necessarily the *only* representation of N in base 16: the c_k are thus

exactly its hexadecimal digits, and not some equivalent but different writing of the canonical one. ■

Note (binary digit counts not a multiple of 4*).* If N has a number of binary digits that is not a multiple of 4, one pads its binary representation with non-significant leading zeros until reaching a multiple of 4: an extra 0 digit in the most significant position contributes $0 \cdot 2^i = 0$ to the sum in Definition 1.1, so it does not change the value of N, and it makes Proposition 7.1 applicable. The byte in Example 7.1 already has $8 = 4 \cdot 2$ digits, a multiple of 4: no padding is needed. It would be needed, for instance, to convert the number $(101)_2$ alone (three digits): one would first write it as $(0101)_2$, a complete block.

Application to Example 7.1. The byte $N = (10110110)_2$ has bits (from b_7 to b_0) $1, 0, 1, 1, 0, 1, 1, 0$, so $n + 1 = 8 = 4 \cdot 2$, $m = 2$, no padding needed. By Proposition 7.1:

$$\begin{aligned} &\text{block } k = 0 \ (b_3b_2b_1b_0 = 0110): \quad c_0 = b_0 \cdot 1 + b_1 \cdot 2 + b_2 \cdot 4 + b_3 \cdot 8 = 0 + 2 + 4 + 0 = 6, \\ &\text{block } k = 1 \ (b_7b_6b_5b_4 = 1011): \quad c_1 = b_4 \cdot 1 + b_5 \cdot 2 + b_6 \cdot 4 + b_7 \cdot 8 = 1 + 2 + 0 + 8 = 11. \end{aligned}$$

Hence $N = c_1 \cdot 16 + c_0 = 11 \cdot 16 + 6$, i.e., writing 11 with the hexadecimal symbol B: $N = (\mathrm{B6})_{16}$ – the same result found in Example 7.1 by successive division, now justified in general by Proposition 7.1, and not merely verified case by case.

In the opposite direction, $(\mathrm{2F})_{16} = 2 \cdot 16 + 15 = 47$, which in binary is $(00101111)_2$ (check: $32 + 8 + 4 + 2 + 1 = 47$ ✓; by the same shortcut: $2 \to 0010$, $\mathrm{F} = 15 \to 1111$).

The whole framework developed in this article – from Definition 1.1 to Theorem 2.1, down to the direct method of Section 3 – shows that converting between arbitrary bases never requires base ten as an intermediate stop: base ten remains one among infinitely many equally legitimate choices, rooted historically in the anatomy of the human hand rather than in any distinctive mathematical property.

Bibliography

[1] I. Niven, H. S. Zuckerman, H. L. Montgomery, *An Introduction to the Theory of Numbers*, 5th ed., Wiley, 1991.

[2] D. E. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, 3rd ed., Addison-Wesley, 1997.

keenonscience – @keenonmaths – in-depth material, not a substitute for the course textbook.

About the author. Nicola Maglione is an electronic engineer (degree in Electronic Engineering, University of Naples, 2001), with over twenty years of experience in the semiconductor industry as a design and program manager, specialized in NAND Flash memory and high-speed interfaces, and holder of several patents in memory architectures. Alongside his professional work, he has always cultivated a passion for mathematics – particularly Analysis, Abstract Algebra, and Number Theory – and for physics, which he shares through keenonscience, his science-outreach page, and on the website keenonmaths.org.

© 2026 Nicola Maglione. Distributed under a Creative Commons Attribution – NonCommercial – NoDerivatives 4.0 International license (CC BY-NC-ND 4.0): sharing is permitted with attribution; commercial use and derivative works are not permitted.
creativecommons.org/licenses/by-nc-nd/4.0