Little-Endian vs Big-Endian in Quantum Computing and the importance of conventions

Danylo Yakymenko

July 2020

1 Introduction

People that are new to quantum computing may be wondering why both the big-endian and the little-endian conventions are in use, why popular quantum frameworks tend to use little-endian convention and why we have to wrap our heads around it. In this writing we discuss not just this particular convention but also other conventions so we can see the full picture.

2 Writing conventions

The majority of the world's population writes text from left to right and from top to bottom. Writing is such a common activity that we are not conscious of the fact that the direction is just a convention. A significant part of the world is using a different convention - they write symbols from right to left (but still from top to bottom). The examples of right-to-left writing systems are Arabic, Hebrew, Persian, and others. Also, there are writing systems that use bottom-to-top writing convention, though they are not in active use today.

From the abstract point of view, if we have a finite linearly ordered set of items then there are only two ways to put them on a horizontal line without order disturbing – either the first item is on the right and the last item is on the left, or vice versa. Similarly for a vertical line.

3 0 vs 1 starting index convention

In mathematics and everyday life we use 1 (or the word *first*) for labelling the first item in an ordered set (notice a self-reference here). But in computer systems, it appears convenient from the engineering point of view to use 0 as a starting index. It's hard to prefer one over the other. The author's preference is to consider the 1-based indexing convention as the default and to use 0-based indexing whenever it's convenient.

4 Endianness in classical computing

In the binary number system every number can be represented as an ordered collection of 0s and 1s. For example, $13 = 1+4+8 = 1\cdot 2^0 + 0\cdot 2^1 + 1\cdot 2^2 + 1\cdot 2^3$, so the corresponding ordered collection is $\{1, 0, 1, 1\}$. But this is not just an ordered collection. Binary decomposition gives us a map from numbers to the set $\{0, 1\}$, that is, 13 is the map $\{2^0 \to 1, 2^1 \to 0, 2^2 \to 1, 2^3 \to 1\}$. This map may suggest to call the first digit the one that corresponds to 2^0, the second digit the one that corresponds to 2^1, and so forth. In other words, the first digit is the least significant digit and the last digit is the most significant digit. But, in fact, this is just a convention called *little-endian*. The other convention, called *big-endian*, is to call the first digit the one that corresponds to 2^3 (the most significant digit in this case), and the last digit corresponds to 2^0.

You may wonder why use the big-endian convention while little-endian looks more natural. In fact, our ordinary decimal system is big-endian. We write the most significant digit from the left, i.e. at first. It's the same for the binary system, e.g. we write $13 = (1101)_2$. However, if we will use the right-to-left writing system then the ordinary number system may be considered as little-endian since the least significant digit will be written first (it's also pronounced first for the numbers under 100 in the Arabic language, for example).

Nowadays most computer systems use the little-endian convention, though big-endian also has some advantages. For example, it's faster to compare numbers in particular big-endian systems.

5 Vectors in linear spaces

If we have an n-dimensional linear space it's convenient to pick a particular basis $\{e_1, e_2, ..., e_n\}$ in it and consider it as a standard. Then any vector can be uniquely written as $v = \alpha_1 e_1 + \alpha_2 e_2 + ... + \alpha_n e_n$, where α_i are from the base field. So, a vector can be described as an ordered collection of the field elements $\{\alpha_1, \alpha_2, ..., \alpha_n\}$ where the first element corresponds to a coefficient near the first basis vector, etc. There is no sense in considering the last coefficient α_n as the first element and so on since this is equivalent to picking a different (reversed) standard basis.

Note, however, that it's convenient to represent vector as a so-called *vector-column* which is a $n \times 1$ matrix. We will see later why this is useful.

6 Operators and matrices

Another basic notion in Linear Algebra is a linear operator (linear map) between linear spaces. If we have two linear spaces L_1 and L_2 with the bases $\{e_1, e_2, ..., e_n\}$ and $\{g_1, g_2, ..., g_m\}$ respectively then a linear operator A from L_1 to L_2 can be uniquely described by the images of the basis vectors of L_1, i.e. by the vectors $\{A(e_1), A(e_2), ..., A(e_n)\}$. In turn, every vector $A(e_i)$ is described by

an ordered collection of m field elements. So, an operator A can be described by an ordered collection of n ordered collections containing m field elements each.

There is another viewpoint, however. We can write $A(v) = \beta_1(v)g_1 + \beta_2(v)g_2 + ... + \beta_m(v)g_m$, where $\beta_i(\cdot)$ are some linear maps from the L_1 to the base field. Note that every β_i is uniquely described by an ordered collection of n elements. So, from this point of view, an operator A can be described by an ordered collection of m ordered collections containing n field elements each. This is, in fact, a "transposed" version of our first view (the total of nm field elements will be the same).

When we have a collection of collections (of the same size) it's convenient to arrange elements in a rectangular fashion, which is called a *matrix* in our context. If we use left-to-right and top-to-bottom writing system it's natural to put the first collection in the first row (with the first element on the left), the second collection in the second row and so on.

But we still have the choice to pick a view from the two that we discussed above. So it will be either $n \times m$ or $m \times n$ (transposed) matrix. The common convention is to use the second view, that is, we describe a linear operator from n-dimensional space to m-dimensional space as a $m \times n$ matrix (m rows, n columns). We will see that another convention might have its own benefits.

7 Compositions

Let's say we have two linear operators A_1, A_2 that act from some linear space L to itself. Their composition $C = A_2 \circ A_1$ is a map from L to L defined by $C(v) = A_2(A_1(v))$, $v \in L$. It is also a linear operator. Notice that while we act by A_1 at first, in the expression $A_2(A_1(v))$ the first operator from the left is A_2 which acts last. This is due to another convention in which operators are written to the left of their operands. In the reverse Polish (postfix) notation arguments are written first (on the left) followed by operators. So we could write $v\ C = v\ A_1\ A_2$ in this postfix notation. Postfix/prefix notations and little/big endian conventions have some similarities between them. Prefix notation is more like big-endian because the first symbol denotes the operator that acts last (the most significant one). We can summarize by saying that in the prefix notation it's easier to analyze what is happening (what kind of results we get) but in the postfix notation the exact calculations are more straightforward.

8 Matrix products

As in the previous section suppose that A_1, A_2 acts from L to L, $C = A_2 \circ A_1$ is their composition and M_1, M_2 are the corresponding matrices of A_1, A_2. We can define the matrix product $M_2 \cdot M_1$ as a matrix that corresponds to the operator C. From this definition we can deduce the usual concrete definition of the matrix product that doesn't assume a connection to operators or vectors, that is, if $D = AB$ then $d_{ij} = \sum_k a_{ik}b_{kj}$.

Now we can see the usefulness of vector-columns. If A is an operator, M its matrix, v is some vector and v_{col} is the corresponding vector-column of v then the corresponding vector-column of $A(v)$ equals to $M \cdot v_{col}$.

But let's recall that we could use a different representation of an operator as a matrix, i.e. M^T instead of M. With this representation the concrete matrix product rule (as a consequence of operator composition definition) will be different. It will be $\text{prod}_2(M_2, M_1) = (M_2^T \cdot M_1^T)^T = M_1 \cdot M_2$ instead of $M_2 \cdot M_1$, where $(\cdot)$ − is the concrete usual matrix product rule. This has a funny effect. We could say that a matrix, that corresponds to a composition of operators A_2, A_1 that have matrices M_2, M_1, is an operator that has a matrix representation given by $M_1 \cdot M_2$ − here the first operator action is on the left. Moreover, this case is aligned with considering vector-rows instead of vector-columns. That is, vector-row of $A(v)$ equals to $v_{row} \cdot M$.

9 Tensor products

Suppose we have two linear spaces L_1 and L_2 with the bases $\{e_1, e_2, ..., e_n\}$ and $\{g_1, g_2, ..., g_m\}$ respectively. We can think of $L_1 \otimes L_2$ as a linear vector space of dimension nm with the basis $\{e_i \otimes g_j\}$, $i \in [1..n]$, $j \in [1..m]$. The rules of tensor products $v \otimes w$ will not be important to us. The problem we are dealing with is that to represent vectors and operators in $L_1 \otimes L_2$ as matrices we have to introduce a *linearly* ordered basis, but there is no unique natural way to linearly order the set $\{e_i \otimes g_j\}$. There are two natural options. Either we order it as

$$\{e_1 \otimes g_1, e_1 \otimes g_2, ..., e_1 \otimes g_m; e_2 \otimes g_1, e_2 \otimes g_2, ..., e_2 \otimes g_m; ...; e_n \otimes g_1, ..., e_n \otimes g_m\},$$

i.e. the first basis is the most significant, so we can call it *big-endian*, or we can order it as

$$\{e_1 \otimes g_1, e_2 \otimes g_1, ..., e_n \otimes g_1; e_1 \otimes g_2, e_2 \otimes g_2, ..., e_n \otimes g_2; ...; e_1 \otimes g_m, ..., e_n \otimes g_m\},$$

i.e. the first basis is the least significant, so we can call it *little-endian*. Depending on our choice the concrete rule of the matrix tensor product will be different. If you have read till this moment then you may guess that the usual matrix tensor product, also known as Kronecker product, corresponds to the big-endian ordering of the basis, which is another unfortunate choice. In other words, the Kronecker product rule implies the big-endian ordering of the basis. For example, for two qubits we have $e_1 = |0\rangle, e_2 = |1\rangle$ and $g_1 = |0\rangle, g_2 = |1\rangle$, and the whole ordering in this case is

$$f_1 = |00\rangle, f_2 = |01\rangle, f_3 = |10\rangle, f_4 = |11\rangle.$$

On the other hand, it could be seen that with the little-endian ordering the matrix tensor product rule will be the same as usual but with the swapped arguments, that is $\text{kron}_2(M_1, M_2) = M_2 \otimes M_1$.

10 Conclusions

We've discussed important conventions used in mathematics and quantum computing in particular, how they emerge, and how they relate to each other. We hope that it will help everyone involved in the field of quantum computing.