Chapter 1

Cbits and Qbits

1.1 What is a quantum computer?

It is tempting to say that a quantum computer is one whose operation is governed by the laws of quantum mechanics. But since the laws of quantum mechanics govern the behavior of all physical phenomena, this temptation must be resisted. Your laptop operates under the laws of quantum mechanics, but it is not a quantum computer. A quantum computer is one whose operation exploits certain very special transformations of its internal state, whose description is the primary subject of this book. The laws of quantum mechanics allow these peculiar transformations to take place under very carefully controlled conditions.

In a quantum computer the physical systems that encode the individual logical bits must have no physical interactions whatever that are not under the complete control of the program. All other interactions, however irrelevant they might be in an ordinary computer – which we shall call *classical* – introduce potentially catastrophic disruptions into the operation of a quantum computer. Such damaging encounters can include interactions with the external environment, such as air molecules bouncing off the physical systems that represent bits, or the absorption of minute amounts of ambient radiant thermal energy. There can even be disruptive interactions between the computationally relevant features of the physical systems that represent bits and other features of those same systems that are associated with computationally irrelevant aspects of their internal structure. Such destructive interactions, between what matters for the computation and what does not, result in *decoherence*, which is fatal to a quantum computation.

To avoid decoherence individual bits cannot in general be encoded in physical systems of macroscopic size, because such systems (except under very special circumstances) cannot be isolated from their own irrelevant internal properties. Such isolation can be achieved if the bits are encoded in a small number of states of a system of atomic size, where extra internal features do not matter, either because they do not exist, or because they require unavailably high energies to come into play. Such atomic-scale systems must also be decoupled from their surroundings except for the completely controlled interactions that are associated with the computational process itself.

Two things keep the situation from being hopeless. First, because the separation between the discrete energy levels of a system on the atomic scale can be enormously larger than the separation between the levels of a large system, the dynamical isolation of an atomic system is easier to achieve. It can take a substantial kick to knock an atom out of its state of lowest energy. The second reason for hope is the discovery that errors induced by extraneous interactions can actually be corrected if they occur at a sufficiently low rate. While error correction is routine for bits represented by classical systems, quantum error correction is constrained by the formidable requirement that it be done without knowing either the original or the corrupted state of the physical systems that represent the bits. Remarkably, this turns out to be possible.

Although the situation is therefore not hopeless, the practical difficulties in the way of achieving useful quantum computation are enormous. Only a rash person would declare that there will be no useful quantum computers by the year 2050, but only a rash person would predict that there will be. Never mind. Whether or not it will ever become a practical technology, there is a beauty to the theory of quantum computation that gives it a powerful appeal as a lovely branch of mathematics, and as a strange generalization of the paradigm of classical computer science, which had completely escaped the attention of computer scientists until the 1980s. The new paradigm demonstrates that the theory of computation can depend profoundly on the physics of the devices that carry it out. Quantum computation is also a valuable source of examples that illustrate and illuminate, in novel ways, the mysterious phenomena that quantum behavior can give rise to.

For computer scientists the most striking thing about quantum computation is that a quantum computer can be vastly more efficient than anything ever imagined in the classical theory of computational complexity, for certain computational tasks of considerable practical interest. The time it takes the quantum computer to accomplish such tasks scales up much more slowly with the size of the input than it does in any classical computer. Much of this book is devoted to examining the most celebrated examples of this speed-up.

This exposition of quantum computation begins with an introduction to quantum mechanics, specially tailored for this particular application. The quantum-mechanics lessons are designed to give you, as efficiently as possible, the conceptual tools needed to delve into quantum computation. This is done by restating the rules of quantum mechanics, not as the remarkable revision of classical Newtonian mechanics required to account for the behavior of matter at the atomic and subatomic levels, but as a curious generalization of rules describing an ordinary classical digital computer. By focusing exclusively on how quantum mechanics enlarges the possibilities for the physical manipulation of digital information, it is possible to characterize how

the quantum theory works in an elementary and quite concise way, which is nevertheless rigorous and complete for this special area of application.

While I assume no prior familiarity with quantum physics (or any other kind of physics), I do assume familiarity with elementary linear algebra and, in particular, with the theory of finite-dimensional vector spaces over the complex numbers. Appendix A summarizes the relevant linear algebra. It is worth examining even if you are well acquainted with the mathematics of such vector spaces, since it also provides a compact summary of the mathematically unconventional language – *Dirac notation* – in which linear algebra is couched in all treatments of quantum computation. Dirac notation is also developed, more informally, throughout the rest of this chapter.

1.2 Cbits and their states

We begin with an offbeat formulation of what an ordinary classical computer does. I frame the elementary remarks that follow in a language which may look artificial and cumbersome, but is designed to accommodate the richer variety of things that a computer can do if it takes full advantage of the possibilities made available by the quantum-mechanical behavior of its constituent parts. By introducing and applying the unfamiliar nomenclature and notation of quantum mechanics in a familiar classical context, I hope to make a little less strange its subsequent extension to the broader quantum setting.

A classical computer operates on strings of zeros and ones, such as 110010111011000, converting them into other such strings. Each position in such a string is called a *bit*, and it contains either a 0 or a 1. To represent such collections of bits the computer must contain a corresponding collection of physical systems, each of which can exist in two unambiguously distinguishable physical states, associated with the value (0 or 1) of the abstract bit that the physical system represents. Such a physical system could be, for example, a switch that could be open (0) or shut (1), or a magnet whose magnetization could be oriented in two different directions, "up" (0) or "down" (1).

It is a common practice in quantum computer science to use the same term "bit" to describe the two-state classical system that represents the value of the abstract bit. But this use of a single term to characterize both the abstract bit (0 or 1) and the physical system whose two states represent the two values is a potential source of confusion. To avoid such confusion, I shall use the term *Cbit* ("C" for "classical") to describe the two-state classical physical system and *Qbit* to describe its quantum generalization. This terminology is inspired by Paul Dirac's early use of *c-number* and *q-number* to describe classical quantities and their quantum-mechanical generalizations. "Cbit" and

"Qbit" are preferable to "c-bit" and "q-bit" because the terms themselves often appear in hyphenated constructions.

Unfortunately the preposterous spelling *qubit* currently holds sway for the quantum system. The term *qubit* was invented and first used in print by the otherwise admirable Benjamin Schumacher.[1] A brief history of the term can be found in the acknowledgments at the end of his paper. Although "qubit" honors the English (German, Italian, . . .) rule that *q* should be followed by *u*, it ignores the equally powerful requirement that *qu* should be followed by a vowel. My guess is that "qubit" has gained acceptance because it visually resembles an obsolete English unit of distance, the homonymic *cubit*. To see its ungainliness with fresh eyes, it suffices to imagine that Dirac had written *qunumber* instead of *q-number*, or that one erased transparencies and cleaned one's ears with *Qutips*.

Because clear distinctions among bits, Cbits, and Qbits are crucial in the introduction to quantum computation that follows, I shall use this currently unfashionable terminology. If you are already addicted to the term *qubit*, please regard *Qbit* as a convenient abbreviation.

To prepare for the extension from Cbits to Qbits, I introduce what may well strike you as a degree of notational overkill in the discussion of Cbits that follows. We shall represent the state of each Cbit as a kind of box, depicted by the symbol $|\ \rangle$, into which we place the value, 0 or 1, represented by that state. Thus the two distinguishable states of a Cbit are represented by the symbols $|0\rangle$ and $|1\rangle$. It is the common practice to call the symbol $|0\rangle$ or $|1\rangle$ itself the *state* of the Cbit, thereby using the same term to refer to both the physical condition of the Cbit and the abstract symbol that represents that physical condition. There is nothing unusual in this. For example one commonly uses the term "position" to refer to the symbol x that represents the physical position of an object. I call this common, if little noted, practice to your attention only because in the quantum case "state" refers *only* to the symbol, there being *no* internal property of the Qbit that the symbol represents. The subtle relation between Qbits and their state symbol will emerge later in this chapter.

Along the same lines, we shall characterize the states of the five Cbits representing 11001, for example, by the symbol

$$|1\rangle|1\rangle|0\rangle|0\rangle|1\rangle, \tag{1.1}$$

and refer to this object as the *state* of all five Cbits. Thus a pair of Cbits can have (or "be in") any of the four possible states

$$|0\rangle|0\rangle,\ |0\rangle|1\rangle,\ |1\rangle|0\rangle,\ |1\rangle|1\rangle, \tag{1.2}$$

1 Benjamin Schumacher, "Quantum coding," *Physical Review* A **51**, 2738–2747 (1995).

three Cbits can be in any of the eight possible states

$$|0\rangle|0\rangle|0\rangle,\ |0\rangle|0\rangle|1\rangle,\ |0\rangle|1\rangle|0\rangle,\ |0\rangle|1\rangle|1\rangle,\ |1\rangle|0\rangle|0\rangle,\ |1\rangle|0\rangle|1\rangle,\ |1\rangle|1\rangle|0\rangle,\ |1\rangle|1\rangle|1\rangle, \tag{1.3}$$

and so on.

As (1.4) already makes evident, when there are many Cbits such products are often much easier to read if one encloses the whole string of zeros and ones in a single bigger box of the form $|\quad\rangle$ rather than having a separate box for each Cbit:

$$|000\rangle,\ |001\rangle,\ |010\rangle,\ |011\rangle,\ |100\rangle,\ |101\rangle,\ |110\rangle,\ |111\rangle. \tag{1.4}$$

We shall freely move between these two equivalent ways of expressing the state of several Cbits that represent a string of bits, boxing the whole string or boxing each individual bit. Whether the form (1.3) or (1.4) is to be preferred depends on the context.

There is also a third form, which is useful when we regard the zeros and ones as constituting the binary expansion of an integer. We can then replace the representations of the 3-Cbit states in (1.4) by the even shorter forms

$$|0\rangle,\ |1\rangle,\ |2\rangle,\ |3\rangle,\ |4\rangle,\ |5\rangle,\ |6\rangle,\ |7\rangle. \tag{1.5}$$

Note that, unlike the forms (1.3) and (1.4), the form (1.5) is ambiguous, unless we are told that these symbols express states of three Cbits. If we are not told, then there is no way of telling, for example, whether $|3\rangle$ represents the 2-Cbit state $|11\rangle$, the 3-Cbit state $|011\rangle$, or the 4-Cbit state $|0011\rangle$, etc. This ambiguity can be removed, when necessary, by adding a subscript making the number of Cbits explicit:

$$|0\rangle_3,\ |1\rangle_3,\ |2\rangle_3,\ |3\rangle_3,\ |4\rangle_3,\ |5\rangle_3,\ |6\rangle_3,\ |7\rangle_3. \tag{1.6}$$

Be warned, however, that, when there is no need to emphasize how many Cbits $|x\rangle$ represents, it can be useful to use such subscripts for other purposes. If, for example, Alice and Bob each possess a single Cbit it can be convenient to describe the state of Alice's Cbit (if it has the value 1) by $|1\rangle_a$, Bob's (if it has the value 0) by $|0\rangle_b$, and the joint state of the two by $|1\rangle_a|0\rangle_b$ or $|10\rangle_{ab}$.

Dirac introduced the $|\ \rangle$ notation (known as Dirac notation) in the early days of the quantum theory, as a useful way to write and manipulate *vectors*. For silly reasons he called such vectors *kets*, a terminology that has survived to this day. In Dirac notation you can put into the box $|\ \rangle$ anything that serves to specify what the vector is. If, for example, we were talking about displacement vectors in ordinary three-dimensional space, we could have a vector

$$|5 \text{ horizontal centimeters northeast}\rangle. \tag{1.7}$$

In using Dirac notation to express the state of a Cbit, or a collection of Cbits, I'm suggesting that there might be some utility in thinking of the states as vectors. Is there? Well, in the case of Cbits, not very much, but maybe a little. We now explore this way of thinking about Cbit states, because when we come to the generalization to Qbits, it becomes absolutely essential to consider them to be vectors – so much so that the term *state* is often taken to be synonymous with *vector* (or, more precisely, "vector that represents the state").

We shall briefly explore what one can do with Cbits when one takes the two states $|0\rangle$ and $|1\rangle$ of a single Cbit to be represented by two *orthogonal unit vectors in a two-dimensional space.* While this is little more than a curious and unnecessarily elaborate way of describing Cbits, it is fundamental and unavoidable in dealing with Qbits. Playing unfamiliar and somewhat silly games with Cbits will enable you to become acquainted with much of the quantum-mechanical formalism in a familiar setting.

If you prefer your vectors to be expressed in terms of components, note that we can represent the two orthogonal states of a single Cbit, $|0\rangle$ and $|1\rangle$, as column vectors

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \qquad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \tag{1.8}$$

In the case of two Cbits the vector space is four-dimensional, with an orthonormal basis

$$|00\rangle,\ |01\rangle,\ |10\rangle,\ |11\rangle. \tag{1.9}$$

The alternative notation for this basis,

$$|0\rangle|0\rangle,\ |0\rangle|1\rangle,\ |1\rangle|0\rangle,\ |1\rangle|1\rangle, \tag{1.10}$$

is deliberately designed to suggest multiplication, since it is, in fact, a short-hand notation for the *tensor product* of the two single-Cbit 2-vectors, written in more formal mathematical notation as

$$|0\rangle \otimes |0\rangle,\ \ |0\rangle \otimes |1\rangle,\ \ |1\rangle \otimes |0\rangle,\ \ |1\rangle \otimes |1\rangle. \tag{1.11}$$

In terms of components, the tensor product $\mathbf{a} \otimes \mathbf{b}$ of an M-component vector $\mathbf{a}$ with components a_μ and an N-component vector $\mathbf{b}$ with components b_ν is the (MN)-component vector with components indexed by all the MN possible pairs of indices (μ, ν), whose (μ, ν)th component is just the product $a_\mu b_\nu$. A broader view can be found in the extended review of vector-space concepts in Appendix A. I shall freely move back and forth between the various ways (1.9)–(1.11) of writing the tensor product and their generalizations to multi-Cbit states, using in each case a form that makes the content clearest.

Once one agrees to regard the two 1-Cbit states as orthogonal unit vectors, the tensor product is indeed the natural way to represent

multi-Cbit states, since it leads to the obvious multi-Cbit generalization of the representation (1.8) of 1-Cbit states as column vectors. If we express the states $|0\rangle$ and $|1\rangle$ of each single Cbit as column vectors, then we can get the column vector describing a multi-Cbit state by repeatedly applying the rule for the components of the tensor product of two vectors. The result is illustrated here for a three-fold tensor product:

$$\begin{pmatrix} x_0 \\ x_1 \end{pmatrix} \otimes \begin{pmatrix} y_0 \\ y_1 \end{pmatrix} \otimes \begin{pmatrix} z_0 \\ z_1 \end{pmatrix} = \begin{pmatrix} x_0y_0z_0 \\ x_0y_0z_1 \\ x_0y_1z_0 \\ x_0y_1z_1 \\ x_1y_0z_0 \\ x_1y_0z_1 \\ x_1y_1z_0 \\ x_1y_1z_1 \end{pmatrix}. \tag{1.12}$$

On applying this, for example, to the case $|5\rangle_3$, we have

$$|5\rangle_3 = |101\rangle = |1\rangle|0\rangle|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}. \tag{1.13}$$

If we label the vertical components of the 8-vector on the right 0, 1, . . ., 7, from the top down, then the single nonzero component is the 1 in position 5 – precisely the position specified by the state vector in its form on the left of (1.13). This is indeed the obvious multi-Cbit generalization of the column-vector form (1.8) for 1-Cbit states.

This is quite general: the tensor-product structure of multi-Cbit states is just what one needs in order for the 2^n-dimensional column vector representing the state $|m\rangle_n$ to have all its entries zero except for a single 1 in the mth position down from the top.

One can turn this development upside down, taking as one's starting point the simple rule that an integer x in the range $0 \le x < N$ is represented by one of N orthonormal vectors in an N-dimensional space. One can then pick a basis so that 0 is represented by an N-component column vector $|0\rangle$ that has 0 in every position except for a 1 in the top position, and x is to be represented by an N-component column vector $|x\rangle$ that has 0 in every position except for a 1 in the position x down from the top. It then follows from the nature of the tensor product that if $N = 2^n$ and x has the binary expansion $x = \sum_{j=0}^{n-1} x_j 2^j$, then the column vector $|x\rangle_n$ is the tensor product of the n 2-component column vectors $|x_j\rangle$:

$$|x\rangle_n = |x_{n-1}\rangle \otimes |x_{n-2}\rangle \otimes \cdots \otimes |x_1\rangle \otimes |x_0\rangle. \tag{1.14}$$

In dealing with n-Cbit states of the form (1.14) we shall identify each of the n 1-Cbit states, out of which they are composed, by giving the power of 2 associated with the individual bit that the Cbit represents. Thus the 1-Cbit state on the extreme right of (1.14) represents Cbit 0, the state immediately to its left represents Cbit 1, and so on.

This relation between tensor products of vectors and positional notation for integers is not confined to the binary system. Suppose, for example, one represents a decimal digit $x = 0, 1, \ldots, 9$ as a 10-component column vector $\mathbf{v}^{(x)}$ with all components 0 except for a 1, x positions down from the top. If the n-digit decimal number $X = \sum_{j=0}^{n-1} x_j 10^j$ is represented by the tensor product $\mathbf{V} = \mathbf{v}^{(x_{n-1})} \otimes \mathbf{v}^{(x_{n-2})} \otimes \cdots \otimes \mathbf{v}^{(1)} \otimes \mathbf{v}^{(0)}$, then $\mathbf{V}$ will be a 10^n-component column vector with all components 0 except for a 1, x positions down from the top.

Although the representation of Cbit states by column vectors clearly shows why tensor products give a natural description of multi-Cbit states, for almost all other purposes it is better and much simpler to forget about column vectors and components, and deal directly with the state vectors in their abstract forms (1.3)–(1.6).

1.3 Reversible operations on Cbits

Quantum computers do an important part of their magic through *reversible* operations, which transform the initial state of the Qbits into its final form using only processes whose action can be inverted. There is only a single *irreversible* component to the operation of a quantum computer, called *measurement*, which is the only way to extract useful information from the Qbits after their state has acquired its final form. Although measurement is a nontrivial and crucial part of any quantum computation, in a classical computer the extraction of information from the state of the Cbits is so conceptually straightforward that it is not viewed as an inherent part of the computational process, though it is, of course, a nontrivial concern for those who design digital displays or printers. Because the only computationally relevant operations on a classical computer that can be extended to operations on a quantum computer are reversible, only operations on Cbits that are reversible will be of interest to us here.

In a reversible operation every final state arises from a unique initial state. An example of an irreversible operation is ERASE, which forces a Cbit into the state $|0\rangle$ regardless of whether its initial state is $|0\rangle$ or $|1\rangle$. ERASE is irreversible in the sense that, given only the final state and the fact that it was the output of the operation ERASE, there is no way to recover the initial state.

The only nontrivial reversible operation we can apply to a single Cbit is the NOT operation, denoted by the symbol **X**, which interchanges

the two states $|0\rangle$ and $|1\rangle$:

$$\mathbf{X} : |x\rangle \to |\tilde{x}\rangle; \quad \tilde{1} = 0, \quad \tilde{0} = 1. \tag{1.15}$$

This is sometimes referred to as *flipping* the Cbit. NOT is reversible because it has an inverse: applying $\mathbf{X}$ a second time brings the state of the Cbit back to its original form:

$$\mathbf{X}^2 = \mathbf{1}, \tag{1.16}$$

where $\mathbf{1}$ is the unit (identity) operator. If we represent the two orthogonal states of the Cbit by the column vectors (1.8), then we can express NOT by a linear operator $\mathbf{X}$ on the two-dimensional vector space, whose action on the column vectors is given by the matrix

$$\mathbf{X} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}. \tag{1.17}$$

So the two reversible things you can do to a single Cbit – leaving it alone and flipping it – correspond to the two linear operators $\mathbf{X}$ and $\mathbf{1}$,

$$\mathbf{1} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \tag{1.18}$$

on its two-dimensional vector space.

A pedantic digression: since multiplication by the scalar 1 and action by the unit operator $\mathbf{1}$ achieve the same result, I shall sometimes follow the possibly irritating practice of physicists and not distinguish notationally between them. I shall take similar liberties with the scalar 0, the zero vector $\mathbf{0}$, and the zero operator $\mathbf{0}$.

Possibilities for reversible operations get richer when we go from a single Cbit to a pair of Cbits. The most general reversible operation on two Cbits is any permutation of their four possible states. There are $4! = 24$ such operations. Perhaps the simplest nontrivial example is the *swap* (or *exchange*) operator $\mathbf{S}_{ij}$, which simply interchanges the states of Cbits i and j:

$$\mathbf{S}_{10}|xy\rangle = |yx\rangle. \tag{1.19}$$

Since the swap operator $\mathbf{S}_{10}$ interchanges $|01\rangle = |1\rangle_2$ and $|10\rangle = |2\rangle_2$, while leaving $|00\rangle = |0\rangle_2$ and $|11\rangle = |3\rangle_2$ fixed, its matrix in the basis $|0\rangle_2, |1\rangle_2, |2\rangle_2, |3\rangle_2$ is

$$\mathbf{S}_{10} = \mathbf{S}_{01} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{1.20}$$

The 2-Cbit operator whose extension to Qbits plays by far the most important role in quantum computation is the *controlled-NOT* or cNOT operator $\mathbf{C}_{ij}$. If the state of the ith Cbit (the *control Cbit*) is $|0\rangle$, $\mathbf{C}_{ij}$ leaves the state of the jth Cbit (the *target Cbit*) unchanged, but,

if the state of the control Cbit is $|1\rangle$, $\mathbf{C}_{ij}$ applies the NOT operator $\mathbf{X}$ to the state of the target Cbit. In either case the state of the control Cbit is left unchanged.

We can summarize this compactly by writing

$$\mathbf{C}_{10}|x\rangle|y\rangle = |x\rangle|y \oplus x\rangle, \qquad \mathbf{C}_{01}|x\rangle|y\rangle = |x \oplus y\rangle|y\rangle, \tag{1.21}$$

where $\oplus$ denotes addition modulo 2:

$$y \oplus 0 = y, \qquad y \oplus 1 = \tilde{y} = 1 - y. \tag{1.22}$$

The modulo-2 sum $x \oplus y$ is also called the "exclusive OR" (or XOR) of x and y.

You can construct SWAP out of three cNOT operations:

$$\mathbf{S}_{ij} = \mathbf{C}_{ij}\mathbf{C}_{ji}\mathbf{C}_{ij}. \tag{1.23}$$

This can easily be verified by repeated applications of (1.21), noting that $x \oplus x = 0$. We note some other ways of showing it below.

To construct the matrix for the cNOT operation in the four-dimensional 2-Cbit space, note that if the control Cbit is on the left then cNOT leaves $|00\rangle = |0\rangle_2$ and $|01\rangle = |1\rangle_2$ fixed and exchanges $|10\rangle = |2\rangle_2$ and $|11\rangle = |3\rangle_2$. Therefore the $4 \otimes 4$ matrix representing $\mathbf{C}_{10}$ is just

$$\mathbf{C}_{10} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \tag{1.24}$$

If the control Cbit is on the right, then the states $|01\rangle = |1\rangle_2$ and $|11\rangle = |3\rangle_2$ are interchanged, and $|00\rangle = |0\rangle_2$ and $|10\rangle = |2\rangle_2$ are fixed, so the matrix representing $\mathbf{C}_{01}$ is

$$\mathbf{C}_{01} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}. \tag{1.25}$$

The construction (1.23) of $\mathbf{S}$ out of cNOT operators also follows from (1.20), (1.24), and (1.25), using matrix multiplication. As a practical matter, it is almost always more efficient to establish operator identities by dealing with them directly as operators, avoiding matrix representations.

A very common kind of 2-Cbit operator consists of the tensor product $\otimes$ of two 1-Cbit operators:

$$(\mathbf{a} \otimes \mathbf{b})|xy\rangle = (\mathbf{a} \otimes \mathbf{b})|x\rangle \otimes |y\rangle = \mathbf{a}|x\rangle \otimes \mathbf{b}|y\rangle, \tag{1.26}$$

from which it follows that

$$(\mathbf{a} \otimes \mathbf{b})(\mathbf{c} \otimes \mathbf{d}) = (\mathbf{ac}) \otimes (\mathbf{bd}). \tag{1.27}$$