Quantum Computing, Part 2

Andrew Glassner

The computers we use today are engineering and technical marvels. They employ powerful mathematical abstractions about information and computation to create virtual machines that can do everything from simulating a thunderstorm to engaging you in real-time battle with a fire-breathing dragon.

But for all their power, today's computers can only do one thing at a time. Sure, we can put two computers side by side and carry out out *parallel computing*, but to have N streams of parallel computation, we need to have N computers. And parallel computing is hard to do efficiently: if we double the number of processors, the speed of computation goes up, but very rarely by a factor of 2. These are the realities of *classical computing*.

There's a whole new approach on the horizon, called *quantum computing*. The idea is to harness some of the strange properties from the world of quantum physics to build a new breed of computers. The scientific and engineering challenges to building a real quantum computer are formidable. But tiny, 3-bit quantum computers have been built and they prove that the theory works.

In the last issue, I introduced some of the ideas of quantum computing. Here I'll begin with a quick recap of those ideas, and then dig into the notation and terminology. In the next issue we'll see some tools and algorithms central to quantum computing.

You'll probably find the notation here a little unusual, but basically all we'll be doing is manipulating vectors and matrices. I'll use physics language and symbology here because it's the language of quantum computing.

A quick review

As a quick refresher, let's look at the characteristics of the quantum world I discussed last time. Then I'll quickly review some linear algebra, because we'll be talking about familiar ideas with unfamiliar notation.

In the following, electrons and photons are typical examples of quantum particles:

- A quantum particle can exist simultaneously in many incompatible configurations, or *states*. We call this *superposition*.
- We can operate on a quantum particle while it's in a superimposed state and affect all the states at once.
- When we *observe* a quantum particle, the very act of observation causes the particle to take on one and only one state. If we repeat the same observation before otherwise affecting the particle, we'll see the same state.
- The particle and the measuring apparatus determine the possible states that result from a measurement.

As we saw last time, quantum mechanics often contradicts our intuition. After all, how can a particle exist simultaneously in several incompatible states? What could that possibly mean? Nothing in our everyday experience works like that—a bird is flying or it isn't, a tree is alive or dead, and a bit is one or zero. Certainly a tree can be thriving or dying, but it can't be alive and dead simulaneously. But in the quantum realm it's a different story, and that opens the door to quantum computing.

Linear mathematics describes quantum mechanics. That means that linear algebra is our friend when talking about this field. Central to linear algebra is the idea of *complex numbers*. If you're familiar with these, you might want to skim the next paragraph, but slow down for the next section so you can pick up the bra-ket notation.

Briefly, a complex number is a two-part assembly constructed by taking two real numbers, multiplying one of them by the square root of –1, and then adding them together. A complex number z is written as $z = a + bi$, where a and b are real numbers, and $i=\sqrt{-1}$ (mathematicians often use the letter i for the square root of negative one, while physicists often use the letter j). We can think of z as a vector in a 2D coordinate system that begins at (0,0) and ends at (a,b). The *magnitude* of z, written $|z|$, is the length of this vector: $|z|=\sqrt{a^2+b^2}$. Finally, the *complex conjugate* of z, written $\overline{z}$ (or sometimes z^*), is simply the reflection of z around the X axis: $\overline{z}=a-bi$. If a complex number has no imaginary component (such as $z = a + 0i$), we say that z is a *pure real*. Similarly, if there's no real component (such as $z = 0 + bi$), we say z is a *pure imaginary*. If $z = 0 + 0i$, we simply write $z = 0$ and call it zero. If z is a pure real, then the complex conjugate is just z itself: $\overline{z}=z$. Note that conjugation doesn't change the length of a complex number: $|z| = |\overline{z}|$. If z has a length of 1, we say it's *normalized*, or is a *unit vector*. For more information on complex numbers, see my column in the January–February 1999 issue.

Bra-ket notation

In computer graphics we generally use a straightforward notation for vectors. A list of numbers in square

brackets (or sometimes parentheses), like $[a\ b\ c]$ is a row vector of three elements. To represent a column vector, we simply *transpose* a row vector by rotating it 90 degrees. We indicate that with a superscript T—thus $[abc]^T$ is a column vector. Physicists use a slightly different notation due to P.A.M. Dirac, called the *bra-ket notation*. A *ket* is a column vector, and it's written $|abc\rangle$. A *bra* is a column vector, written $\langle abc|$, but we create it by forming the complex conjugate of the listed elements. So in tableau form

$$|abc\rangle = \begin{bmatrix} a \\ b \\ c \end{bmatrix}, \quad \langle abc| = \begin{bmatrix} \bar{a} & \bar{b} & \bar{c} \end{bmatrix}$$

Note that $\langle$ and $\rangle$ are not less-than and greater-than signs; they're taller and skinnier. If we want to form the traditional *dot product* (or *inner product*), we just stick these two guys together: $\langle abc||def\rangle = \langle abc|def\rangle = \bar{a}d + \bar{b}e + \bar{c}f$. Note that we only need to use one vertical bar when we form this type of expression. When all the entries are purely real this is the familiar dot product we use during shading and other graphics calculations.

If we put a bra and a ket together in the other order, we create a matrix (also called the *outer product*):

$$|def\rangle\langle abc| \rightarrow \begin{array}{c|ccc} & \bar{a} & \bar{b} & \bar{c} \\ \hline d & d\bar{a} & d\bar{b} & d\bar{c} \\ e & e\bar{a} & e\bar{b} & e\bar{c} \\ f & f\bar{a} & f\bar{b} & f\bar{c} \end{array} \rightarrow \begin{bmatrix} d\bar{a} & d\bar{b} & d\bar{c} \\ e\bar{a} & e\bar{b} & e\bar{c} \\ f\bar{a} & f\bar{b} & f\bar{c} \end{bmatrix}$$

Qubits

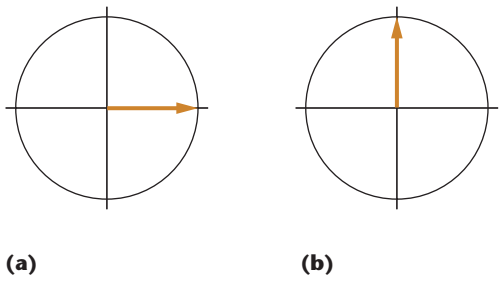
The fundamental element of classical computing is the *bit*. It's an abstract quantity that can be in one of two states: 0 and 1. I say it's abstract because in theoretical discussions we don't care about the details of how bits are implemented in different hardware: a bit could be a magnetic core on a pair of wires, a trapped quantity of electrical charge, or even a mousetrap (which can be sprung or unsprung). Similarly, in quantum computing the fundamental unit of computation is the *qubit* (pronounced q-bit). The qubit also has two basic states, but they're a little different than the classical states.

A quantum state is represented by a ket (recall that it's a column vector of complex numbers). By convention, the two basic states are called $|0\rangle$ and $|1\rangle$, and correspond to the X and Y axes:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Figure 1 illustrates these states. These two states are called the *standard basis*. Note that in both of the standard bases, both numbers are purely real. These vectors are perpendicular, or orthogonal: $\langle 0|1\rangle = 0$.

In my last column, we saw that a quantum particle can be in more than one state simultaneously. We say that multiple states can exist in *superposition* in a single qubit. To capture this idea, we write a qubit q in a superposition as the sum of two or more different states, each scaled by a different amount.



1 Visual interpretation of the quantum bases. (a) State |0⟩ corresponds to (1,0). (b) State |1⟩ corresponds to (0,1).

The quantum basis vectors behave just like the bases we're used to in linear algebra. In the 2D plane, we usually use the basis vectors $\mathbf{x} = [1\ 0]$ and $\mathbf{y} = [0\ 1]$. Then any vector $\mathbf{v} = (a, b)$ can be written as a combination of the bases: $\mathbf{v} = a\mathbf{x} + b\mathbf{y}$. Similarly, a qubit in superposition is written as a combination of the two basis vectors: $q = a|0\rangle + b|1\rangle$.

As we've just seen, the notation for a superimposed state uses the standard addition sign +. When we see an expression like 3 + 5, we often mentally discard the two elements and just think about their sum, 8. Similarly, when working with the vector $\mathbf{v}$ above, we usually forget about the bases $\mathbf{x}$ and $\mathbf{y}$, and just think of $\mathbf{v}$ as (a, b).

Although we use the addition sign in quantum computing notation just as it's used in linear algebra, I find it's useful to keep the component states distinct in my mind. I think of + used in this case like playing two songs on my stereo simultaneously. The result is a mix of the two songs, but I can mentally tune in to either one as easily as I can focus on the composite—the two songs are distinct but coexisting.

Using the standard basis introduced above, we write a superimposed state as $q = c_0|0\rangle + c_1|1\rangle$, where c_0 and c_1 are complex and scaled so that $|c_0|^2 + |c_1|^2 = 1$. The latter condition keeps the magnitude of q normalized, which is important when carrying out calculations on actual hardware.

The *measurement postulate* of quantum mechanics says that when we measure (or observe) a superimposed qubit q, it will instantly snap (or collapse or be projected) into one of the states allowed by the measurement. We typically measure qubits by evaluating them with respect to the standard basis, so we'll find a qubit in either state $|0\rangle$ or $|1\rangle$.

When we observe a superimposed qubit, which state do we see? The answer is probabilistic and depends on the square of the weights on the different states. So for our qubit above, our measurement has a probability of $|c_0|^2$ of being found in state $|0\rangle$, and a probability of $|c_1|^2$ of state $|1\rangle$.

Just as there are many ways to represent a classical bit, we can make physical qubits with a variety of physical implementations. In the last issue we saw how photons passing through a sheet of polarizing material become polarized either parallel or perpendicular to the material's polarization direction. We could say these two directions correspond to the $|0\rangle$ and $|1\rangle$ states, so polarized photons are one realization of qubits. Another implementation example is based on a property of elec-

(a) **(b)**

2 Assembling registers. (a) Three separate classical bits. (b) A 3-bit register created with the Cartesian product.

trons called *spin* (despite the colorful name, don't think of electrons literally as little spinning balls). Spin comes in only two flavors: up and down. So it's natural to associate those spin directions with qubit states. These (and other) particle properties bridge the gap between theory and practice: polarization and spin are real, physical phenomena we can measure.

Remember in the following sections that when you see an expression like $q=c_0|0\rangle+c_1|1\rangle$, we're referring to a superimposed quantum particle q that is simultaneously in two distinct states, $|0\rangle$ and $|1\rangle$. Only when we finally measure it will the particle snap into a single state. The probability of seeing each of the states is the corresponding weight's square.

Multibit registers

There's not much you can do with a single classical bit. Typically we pack up bits together into larger units, such as bytes and words. In general, we often speak of a *register*, which is a sequence of bits of any length.

To create a register on any kind of computer, we need some kind of mathematical glue to stick the bits together. The simplest glue is called the *Cartesian product*, represented by the multiplication sign $\times$. For our purposes, this simply takes two elements and abuts them side by side.

We build up a register on a traditional computer using the Cartesian product. If we have three bits, b_0, b_1, and b_2, then we can create a 3-bit register $R = b_0 \times b_1 \times b_2$. Figure 2 shows the idea. Note here that we aren't multiplying together the bits in the numerical sense—we're gluing them side-by-side.

Because R consists of three bits, to specify the state of R we need only identify the state of each of its three components. In other words, three numbers completely identify the state of R: the values of its three bits.

To be more general, suppose we have a system described by three 2D variables. For example, we could describe a song with three pairs of values: (tempo, volume), (style, orchestration), and (date-composed, date-recorded).

More abstractly, let's denote the first pair of axes as $U=\{u_0, u_1\}$, the second as $V=\{v_0, v_1\}$, and the third as $W=\{w_0, w_1\}$. To describe a song, we paste these three 2D systems together. The first two go together like this:

$$U \times V = \{u_0, u_1\} \times \{v_0, v_1\} = \{u_0, u_1, v_0, v_1\}$$

All three give us:

$$\begin{aligned} U \times V \times W &= \{u_0, u_1\} \times \{v_0, v_1\} \times \{w_0, w_1\} \\ &= \{u_0, u_1, v_0, v_1, w_0, w_1\} \end{aligned}$$

In other words, we describe the complete system by specifying six numbers. In general, if we have two spaces X and Y with dimensions $\dim(X)$ and $\dim(Y)$, then the number of dimensions in the Cartesian product is their sum:

$$\dim(X \times Y) = \dim(X) + \dim(Y)$$

In quantum computing, things get a little more interesting. The big difference comes about because we need to account for superposition. In the quantum world, we put qubits together with a more complex glue called the *tensor product*, written $\otimes$. A convenient way to think about the tensor product of two items is that it's a little machine that unpacks its arguments into their components, and then creates all the combinations that result, in the order in which they're named.

For a first example, let's look at just two of the musical axes we encountered earlier. The tensor product $U \otimes V$ breaks these spaces apart and puts them back together again:

$$\begin{aligned} U \otimes V &= \{u_0, u_1\} \otimes \{v_0, v_1\} \\ &= \{(u_0,v_0),(u_0,v_1),(u_1,v_0),(u_1,v_1)\} \end{aligned}$$

Note that we have four elements—the same number as from the Cartesian product. But this time we have four pairs, one for each combination of the source elements.

If we carry this operation out on all three spaces, we get

$$\begin{aligned} U \otimes V \otimes W &= \{u_0, u_1\} \otimes \{v_0, v_1\} \otimes \{w_0, w_1\} \\ &= \{(u_0, v_0, w_0), (u_0, v_0, w_1), (u_0, v_1, w_0), \\ &\quad (u_0, v_1, w_1), (u_1, v_0, w_0), (u_1, v_0, w_1), \\ &\quad (u_1, v_1, w_0), (u_1, v_1, w_1)\} \end{aligned}$$

Now we see the big difference between the Cartesian product and the tensor product. The Cartesian product gave us six elements, while the tensor product gave us eight, one for each combination of the underlying inputs. The number of elements in the tensor product is the product of the number of elements in each of the spaces:

$$\dim(X \otimes Y) = \dim(X) \times \dim(Y)$$

where here the $\times$ is for multiplication. The important thing to remember about the tensor product is that it doesn't just glue together its arguments. Rather, it breaks each one down one step (if possible), and then glues together all the combinations that result. The order of the elements in those combinations is the same as the order in which they're named.

The Cartesian example we saw for three 2D systems basically creates a new 6D space. To describe a point in this space, we need six numbers, one for each element in the product. We scale together the elements in the Cartesian product by the associated weight, and then sum the results together.

Using the tensor product, we need eight numbers. We identify the points in this space by weighting and then summing the eight elements in the tensor product forms, each element of which represents a particular combination of the input states. So in the tensor prod-

uct, we can't adjust the input elements individually, as in the Cartesian product—we can only scale together composite states. It's as though the elements are rebundled into sealed packages, and we can only scale how much of each package is in the final state. Figure 3 shows this idea graphically.

Let's look at another example, involving two matrices, A and B:

$$A=\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix} \quad B=\begin{bmatrix}5\\ 6\end{bmatrix}$$

Here's the tensor product $A \otimes B$:

$$A\otimes B=\begin{bmatrix}1B & 2B\\ 3B & 4B\end{bmatrix}=\begin{bmatrix}1\begin{bmatrix}5\\6\end{bmatrix} & 2\begin{bmatrix}5\\6\end{bmatrix}\\ 3\begin{bmatrix}5\\6\end{bmatrix} & 4\begin{bmatrix}5\\6\end{bmatrix}\end{bmatrix}=\begin{bmatrix}5 & 10\\ 6 & 12\\ 15 & 20\\ 18 & 24\end{bmatrix}$$

Compare this to $B \otimes A$:

$$B\otimes A=\begin{bmatrix}5A\\ 6A\end{bmatrix}=\begin{bmatrix}5\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}\\ 6\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}\end{bmatrix}=\begin{bmatrix}5 & 10\\ 15 & 20\\ 6 & 12\\ 18 & 24\end{bmatrix}$$

Clearly the order of the operation matters: $A \otimes B \neq B \otimes A$.

We often speak of *preparing* a qubit, meaning that we put it into a desired state. We can use prepared qubits to create prepared registers. Suppose that we make some qubits that aren't superimposed; that is, some are completely in state $|0\rangle$ and others are completely in state $|1\rangle$. Then, as in classical computing, we can build multiqubit registers by simply gluing them together. We use the tensor product, of course, but it doesn't do anything fancy for us in this situation because the qubits have no internal composite structure to break down.

For example, if we wanted to represent the decimal number 6 in binary, we'd set up a 3-bit register to read 110. Using qubits, we write this as

$$|1\rangle\otimes|1\rangle\otimes|0\rangle=|110\rangle=|6\rangle$$

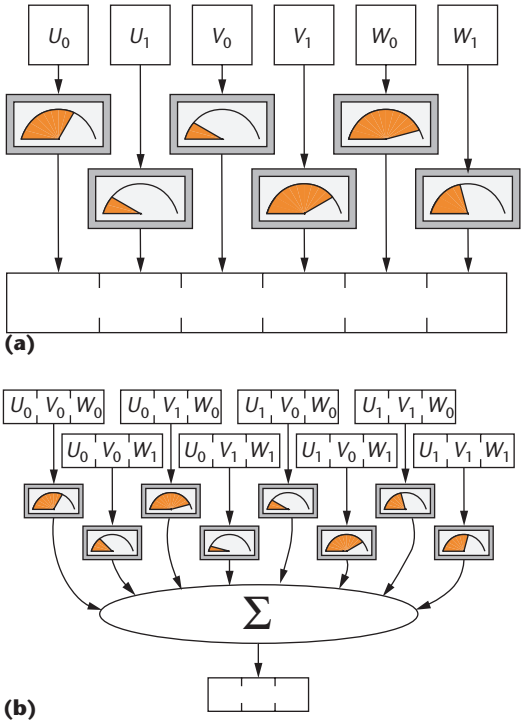
So $|110\rangle$ and $|6\rangle$ are both just notational shorthand for the tensor product of three qubits prepared in these particular states and glued together in this particular order.

By the way, sometimes there's confusion about the base of a register. A register described as $|10\rangle$ can mean the 2-bit register $|1\rangle\otimes|0\rangle$, or the 4-bit register representing the value ten: $|1\rangle\otimes|0\rangle\otimes|1\rangle\otimes|0\rangle$. We can disambiguate these cases either from context or with a subscript: $|10\rangle_2 \neq |10\rangle_{10}$.

Quantum registers get more interesting when they contain one or more qubits that aren't in a pure state. Suppose that we have two qubits,

$$q_0 = |1\rangle$$

and



3 (a) The Cartesian product of three 2D spaces. We can weight each of the six values independently (u_0, u_1, v_0, v_1, w_0, w_1). The result is a six-element vector. (b) The tensor product of the same spaces. We can weight each of the eight combinations independently. In a quantum register, the eight combinations reside simultaneously in a single 3-qubit register.

$$q_1=\frac{1}{\sqrt{2}}\left(|0\rangle+|1\rangle\right)$$

To form a register r from their tensor product, we write

$$r=q_0\otimes q_1=|1\rangle\otimes\frac{1}{\sqrt{2}}\left(|0\rangle+|1\rangle\right)=\frac{1}{\sqrt{2}}\left(|10\rangle+|11\rangle\right)$$

Before we measure r, it's simultaneously representing $|10\rangle$ and $|11\rangle$, with equal probability. As we've seen, it doesn't actually project into one state or the other until we look at it.

Entanglement

When we combine states using superposition, something interesting occurs: we sometimes implicitly measure other bits in the register, even if we don't look at them. Let's see how this works.

First we'll look at a case where nothing terribly strange happens. I'll take two 2-bit registers $a_0 = |01\rangle$ and $a_1 = |11\rangle$ and build a single superimposed state $\alpha = w|01\rangle + w|11\rangle$ where each of the two states has equal weight w (as we've seen, normalization requires that $w^2 + w^2 = 1$, so $w = 1/\sqrt{2}$).

Since the two states have equal weight (and thus equal probabilities), when we observe a quantum register in this superimposed state we will find, upon observation, that it appears in one state or the other with equal probability. So if we were to look at thousands of systems created in state α, we'd find roughly half of them in state $|01\rangle$ and half in $|11\rangle$.

Suppose we had a way to measure just the second qubit of α and leave the first bit unobserved. Looking

at the states a_0 and a_1 that make up α, we can see that the second qubit of both states is $|1\rangle$. If we measure the second qubit of α, then the probability that it will be in state $|1\rangle$ is 1.

What does this tell us about the first qubit of α? Nothing. Because the two qubits are unrelated, measuring the second one reveals nothing about the first.

Suppose that instead we measure just the first qubit. Looking at α, we can see that the first qubit is in an equal superposition of $|0\rangle$ and $|1\rangle$. So half the time the first qubit will come up in state $|0\rangle$, and the other half in $|1\rangle$. The second qubit, of course, will always be $|1\rangle$.

Now let's look at a more interesting case. I'll start with two new states $b_0 = |00\rangle$ and $b_1 = |11\rangle$ and build a new superimposed state $\beta = w|00\rangle + w|11\rangle$.

Suppose now we measure the second qubit of β. Because the states b_0 and b_1 are mixed with equal weights, roughly half the time the second qubit of β will come up in state $|0\rangle$, and half the time in state $|1\rangle$.

Suppose we observe that the second qubit is in state $|0\rangle$. What does this tell us about the first qubit of β? Everything! We now know the state of this first qubit with complete certainty: it's also $|0\rangle$. That's because the superposition that created β is made up of the complete states b_0 and b_1. Since only b_0 has a second qubit in state $|0\rangle$, then the entire register β must be in state b_0.

If the second qubit had come up in state $|1\rangle$, then we would again be completely certain that the entire register was in state b_1 and thus the first qubit was in state $|1\rangle$ as well.

The same thing holds true if we measure the first qubit. Because of the states' structure that make up β, measuring either qubit completely determines the state of the other. In effect, measuring one of the qubits causes the other qubit to be implicitly observed. This unobserved qubit is immediately projected even though we haven't looked at it.

We say that the two qubits of register β are *entangled*.

Entanglement is a curious property that we never see in the everyday world. But it's a key ingredient to many important applications, so let's look at it a little more closely and try to get a better handle on it.

The key to entanglement is that the two qubits have some kind of unseen bond. Is it possible to break this bond? If we could write two entangled qubits as a sum of unentangled qubits, then we could avoid entanglement. Let's try to do this by hypothesizing that β could be created by gluing together two independent qubits, each one in an equally superimposed state. We can write this as

$$\begin{aligned}\beta &= wb_0 + wb_1 = w|00\rangle + w|11\rangle \\ &= \left(p_0|0\rangle + p_1|1\rangle\right) \otimes \left(q_0|0\rangle + q_1|1\rangle\right)\end{aligned}$$

If we could find values for p_0, p_1, q_0, and q_1 that satisfy this expression for β, then we could find a way to tease apart the qubits of β and avoid the implicit measurement created by entanglement. So let's expand out the tensor product and see if we can find such values:

$$\begin{aligned}&\left(p_0|0\rangle + p_1|1\rangle\right) \otimes \left(q_0|0\rangle + q_1|1\rangle\right) \\ &\quad = p_0q_0|00\rangle + p_0q1|01\rangle + p_1q_0|10\rangle + p_1q_1|11\rangle\end{aligned}$$

By definition β doesn't contain state $|01\rangle$ or $|10\rangle$, so $p_0q_1 = p_1q_0 = 0$. The first term, $p_0q_1 = 0$, tells us that either $p_0 = 0$ or $q_1 = 0$. But p_0 can't be 0, because then $p_0p_1 = 0$, and then state $|00\rangle$ would be eliminated from β, contradicting the definition of β. Similarly, if $q_1 = 0$, then $p_1q_1 = 0$, and state $|11\rangle$ would be eliminated. So we know that $p_0 \neq 0$ and $q_1 \neq 0$. So state $|01\rangle$ has a nonzero weight, again contradicting the definition of β. The other possibility, that $p_1q_0 = 0$, leads to the same problems.

So there's no way to get rid of the states that aren't in β without also getting rid of the states that do belong. Thus our hypothesis that β could be built from independent qubits must be false. In other words, the bits must be entangled, or dependent on each other.

This analysis shows us that we can't build up the 2-qubit system β by assembling two independent qubits. The state is a combination of composites. And we've seen that because of the states I used to build β, observing either qubit tells us the status of the other.

Entanglement can get pretty interesting. Let's look at a 4-qubit system δ made up of three equally weighted states:

$$\begin{aligned}\delta &= \frac{1}{\sqrt{3}}(d_0 + d_1 + d_2) \\ &= \frac{1}{\sqrt{3}}\left(|0010\rangle + |1100\rangle + |1101\rangle\right)\end{aligned}$$

Before we measure anything, the odds of finding the second qubit in state $|1\rangle$ are two out of three. Now suppose that we measure the first qubit of δ and find it in state $|0\rangle$. Then we know the whole thing: $\delta = d_0$. But if the qubit is $|1\rangle$, then δ is projected into a new state that's a 50/50 combination of the other two states: $\delta = (1/\sqrt{2})(d_1 + d_2)$. Note that the second qubit has now been projected as well, because it's entangled with the first.

We might capture this interpretation by writing δ as a combination of states that depend on the first qubit:

$$\delta = \frac{1}{\sqrt{3}}\left(|0\rangle \otimes |010\rangle\right) + \frac{2}{\sqrt{3}}\left(|1\rangle \otimes \left(|100\rangle + |101\rangle\right)\right)$$

Just for fun, suppose that instead we measure just the fourth qubit. If it comes up in state $|0\rangle$, then δ is in a 50/50 mix of states d_0 and d_1. If qubit four is in state $|1\rangle$, then δ is in state d_2. We might then choose to write δ this way:

$$\delta = \frac{2}{\sqrt{3}}\left(\left(|001\rangle + |110\rangle\right) \otimes |0\rangle\right) + \frac{1}{\sqrt{3}}\left(|110\rangle \otimes |1\rangle\right)$$

These formulations for δ are equivalent, but they express different relationships among the qubits. The physical system isn't changed by how we choose to write its components, but these different expressions help to highlight different entangled relationships.

More entanglement

Entanglement can lead to some surprising practical applications. Many derive from the fact that entanglement doesn't have a limited lifetime. Particles created in an entangled situation will stay that way until they're measured. Another surprising fact is that two entangled qubits don't have to be near one another. In fact, they can be on opposite sides of the room, or opposite ends of the universe. And finally and most strangely, when we measure one particle of an entangled pair, we also implicitly measure the other and it simultaneously collapses into the necessary state.

As one example of entanglement's features, let's look at a famous thought experiment called the Einstein-Podolsky-Rosen, or EPR, experiment. We'll create an entangled 2-qubit system $\delta = (1/\sqrt{2})\,(|00\rangle + |11\rangle)$. We'll take the first qubit (call it q_0) and put it in a safe here at home, and we'll send the other qubit (q_1) away at the speed of light.

Let's now freeze ourselves for 10 million years and then wake up again. First things first: we have lunch. Then we return to our experiment.

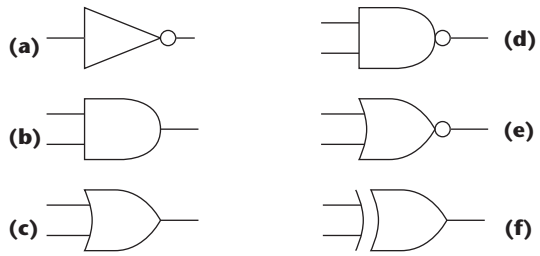
In the time that's passed, particle q_1 has traveled to a spot 10 million light years away. Let's suppose that we have a friend out there who can catch the particle and measure it, if need be.

On our side, we'll open up the safe and take out particle q_0. Before we measure it, it has a 50/50 chance of being in state $|0\rangle$ or $|1\rangle$, and of course q_1 is in the same condition. Now we measure q_0, and suppose that we find it's in state $|0\rangle$. Instantly, on the other side of the universe, entangled particle q_1 snaps into state $|0\rangle$. It takes no time, and it's absolutely certain. Somehow, just by looking at a particle here on Earth, we have instantly and measurably affected a particle 10 million light years far away!

This happens because the particles are entangled: when one is projected, so is the other. Nothing in the physics we've looked at (or even in the physics we haven't looked at) requires time to communicate this information. It's just how quantum mechanics works. No matter where they are in the universe, when we measure one entangled particle, the other one instantly snaps into the state demanded of it.

This conclusion has bothered a lot of people for a lot of reasons, and scientists have carried out many experiments to prove or disprove this result. It turns out that every test of the theory has confirmed this quantum behavior, and no test has ever yet contradicted it.

At first blush, entanglement seems to imply the opportunity for instant communication across arbitrary distances, which could lead to all kinds of marvelous technologies. It turns out that the results are more subtle, as we'll see next time. The bottom line is that yes, it appears that you can instantly alter the state of a particle on the opposite side of the universe, but that operation alone isn't enough to transmit useful information. As far as we know now, you can indeed send information instantly via entanglement, but it's useless until you transmit at least a little bit more information over classical channels. We'll see the details behind this in the next issue.



4 Basic logic components. The (a) NOT , (b) AND, (c) OR, (d) NAND, (e) NOR, and (f) XOR gates.

Although this means that we can't run out and build *Star Trek* style teleportation devices based on this theory, entanglement is of great practical importance for many applications, from quantum cryptography to our current focus of quantum computing.

Quantum gates

Now that we've seen how to create quantum bits and registers, and we've seen a little bit about how these registers behave, let's look at actually using these ideas for computation. I'll start by showing how to generalize today's common classical circuits into quantum-style versions, and then discuss what happens when we think of the inputs as quantum states rather than zeros and ones.

Classical computers are built out of primitive logic elements called *gates*. Basically, a gate is a little machine that takes one or more bits of input and produces one or more bits of output. For the following discussion, I'll name the inputs starting with the letter a, and outputs starting with x. I'll write the negation of a bit a as $\overline{a}$, so if $a = 0$ then $\overline{a} = 1$, and vice versa.

Reversibility

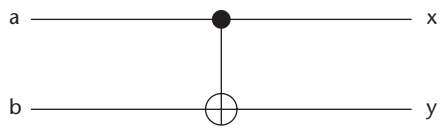
The simplest gate is the NOT gate, which has a single input and output. It just flips its input bit: $x = \overline{a}$. Figure 4a shows the standard symbolic picture for the NOT gate. Its truth table is very simple:

	a	x
NOT:	0	1
	1	0

The next most common gates are the 2-input, 1-output gates called AND and OR. The AND gate outputs a 1 only if both inputs are 1, while OR outputs a 1 if either (or both) of its inputs are 1. Different authors sometimes use different symbols to indicate the logical-and and logical-or operations. I use a common notation that writes AND as $x = a \wedge b$ and OR as $x = a \vee b$. Figures 4b and 4c illustrate these gates. Here are their truth tables:

	a	b	x
	0	0	0
AND:	0	1	0
	1	0	0
	1	1	1

	a	b	x
	0	0	0
OR:	0	1	1
	1	0	1
	1	1	1



5 The controlled-OR or C_{not} gate; *a* is the signal, *b* is the target. Output *x*=*a*, and if *x*=0, then *y*=*b*, else *y* = $\bar{b}$.

If we place a NOT gate at the end of either of these, we create NAND and NOR gates, as shown in Figures 4d and 4e. The truth tables are exactly the opposite of the ones for AND and OR:

	a	*b*	*x*
NAND:	0	0	1
	0	1	1
	1	0	1
	1	1	0

	a	*b*	*x*
NOR:	0	0	1
	0	1	0
	1	0	0
	1	1	0

The last elementary gate we'll look at is the 2-input, 1-output XOR gate. This outputs a 1 if exactly 1 of its inputs is a 1. Written $x = a \oplus b$, Figure 4f shows the symbol.

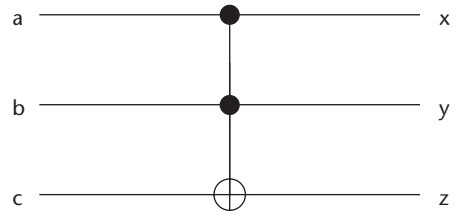
	a	*b*	*x*
XOR:	0	0	0
	0	1	1
	1	0	1
	1	1	0

The AND and NOT gates form what's called a *complete set* of gates. You can prove that if you have enough of these two gates, you can build a circuit that can evaluate any computable function. Other sets of complete gates exist: for example, OR and NOT taken together do the job. The NAND gate is called a *universal gate* because it's a complete set all by itself.

Quantum gates are like classical gates, but they have an extra requirement that changes things. Because of the underlying physics, quantum gates must be *reversible*. This means you can deduce the inputs from the outputs.

Classical gates generally aren't reversible. The NOT gate is an exception: its input is just the opposite of its output. But notice that the other gates we discussed have fewer outputs than inputs. This means that some information gets lost, and we can't recover that. For example, if an AND gate has an output of 1, we know its inputs are both 1. But if an AND has an output of 0, then we only know that at least one of the inputs is also 0—we can't get any more precise than that. Quantum gates, in contrast, need to be completely reversible. This means that there are always at least as many outputs as inputs. If a gate has *n* inputs and outputs, we sometimes call it an *n*-wire gate.

We can build reversible gates in several ways. Let's look at one of the simplest, which implements a reversible form of XOR: this is the *controlled-NOT* gate, written C_{not}. The controlled-NOT is a 2-wire gate



6 The Toffoli gate. Output *x*=*a* and *y*=*b*. If *x*=1 and *y*=1, then *z* = $\bar{c}$, else *z*=*c*.

(see Figure 5); here's the truth table:

	a	*b*	*x*	*y*
C_{not}:	0	0	0	0
	0	1	0	1
	1	0	1	1
	1	1	1	0

We typically call input *a* the *source*, and it passes through unchanged: $x = a$. We call input *b* the *target*, and it turns into the XOR of *a* and *b*: $y = a \oplus b$.

The C_{not} gate is reversible because the inputs can be completely determined from the outputs. It's called the controlled-NOT because you can think of it as negating the target if and only if the source is 1. That is, if the source is 0, then $y = b$, but if the source is 1, then $y = \bar{b}$. The result is still just $y = a \oplus b$, but sometimes this functional viewpoint is more useful when thinking about circuits.

It would be nice to find a universal gate for reversible computation, just as the NAND is universal for classical computation. The C_{not} gate doesn't fill the bill, but something close to it does. In 1980, Toffoli introduced the 3-wire gate (Figure 6), now called the *Toffoli gate*. Inputs *a* and *b* pass through unchanged: $x = a$ and $y = b$. Input *c* is transformed by computing the XOR of *c* with the logical AND of *a* and *b*: $z = c \oplus (a \wedge b)$. The Toffoli gate toggles input *c* if inputs *a* and *b* are both 1, otherwise *c* passes through unchanged. You can think of this like a series circuit with two switches and a light: the light only goes on (that is, *c* only flips) if both switches are on (both *a* and *b* are 1).

Quantum logic

So far I've only discussed classical values of 0 and 1 for each input and output bit. Let's now move into the quantum realm.

It will probably be no surprise that the quantum standard bases $|0\rangle$ and $|1\rangle$ correspond to the classical 0 and 1. Of course, the big change is that quantum bits can be superpositions of these states.

Let's see what happens when we use superpositions for input qubits in the 2-wire C_{not} gate.

Think of the values of *a* and *b* together as a 2-qubit input register α, and *x* and *y* together as a 2-qubit output register β. We'll first create qubits *a* and *b* by mixing equal amounts of each basis state, and then form their tensor product to create the input register α:

$$\alpha = \frac{1}{\sqrt{2}}\left(|0\rangle \otimes |1\rangle\right), \quad b = \frac{1}{\sqrt{2}}\left(|0\rangle \otimes |1\rangle\right)$$

$$\alpha = a \otimes b = \frac{1}{2}\left(|00\rangle + |01\rangle + |10\rangle + |11\rangle\right)$$

To keep things generalized, let's change the weights on α so that the states aren't equally probable:

$$\alpha = v_0|00\rangle + v_1|01\rangle + v_2|10\rangle + v_3|11\rangle$$

where the sum of the squares of the weights v_i is 1. Because the controlled-NOT swaps the value of the last qubit based on the value of the first, it effectively swaps the weights on the last two states to create the output register β:

$$\beta = v_0|00\rangle + v_1|01\rangle + v_3|10\rangle + v_2|11\rangle$$

In fact, the C_{not} gate is a great way to manufacture entangled pairs. Let's prepare a qubit in state $|0\rangle - |1\rangle$ and feed it into input a and put a qubit in state $|1\rangle$ into b. The result is an entangled pair that's an equally blended register in the entangled state $(1/\sqrt{2})(|01\rangle - |10\rangle)$. Entangled pairs are a basic building block for many quantum algorithms, as we'll see next time.

How should we describe how quantum gates work? In the classical domain, truth tables do a fine job—you look up the inputs and read off the outputs. But a quantum gate takes several superimposed input bits and produces superimposed outputs, which isn't the sort of thing that we can capture with a truth table.

The expressions for the C_{not} gate point suggest that the gate simply changes the weights on the inputs. We're already used to a mathematical device that takes a collection of related values and transforms them: the matrix. We can think of the four v weights mentioned previously as a four-element vector that gets transformed. Here's one way to write C_{not}:

$$\begin{bmatrix} v'_0 \\ v'_1 \\ v'_2 \\ v'_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

The matrix form is a natural for representing quantum gates, but we need to establish a convention for representing states. For example, when we write a 2D point as (3, 5) we're implicitly writing it as a combination of the two basis vectors in 2D: $3\mathbf{x} + 5\mathbf{y} = 3[1, 0] + 5[0, 1]$. We can only leave off the vectors because we know by convention that they should be applied in the order **x** and then **y**. Without that convention, we wouldn't know if we meant $(x = 3, y = 5)$ or $(y = 3, x = 5)$.

In quantum computing, a similar convention exists. If there are n qubits, then we'll have 2^n combinations of the bases. Thus an n-qubit register corresponds to a ket with 2^n entries. Now that we know the size of the kets, we need to establish a convention for ordering them.

The convention used in quantum computing works this way: write out the state of the ket as a binary string and put a 1 in the column position corresponding to that binary number, starting with 0 at the top. For example, here is the 4-tuple basis, corresponding to a 2-qubit register:

$$|00\rangle = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad |01\rangle = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \quad |10\rangle = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \quad |11\rangle = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

There's nothing magical about this convention, just as there's no reason we order 2D points as (x, y) rather than (y, x); it's just the agreed-upon procedure.

You probably won't be too surprised to see that this is the convention I used in the description of the C_{not} gate above. To summarize that, here's the definition of the C_{not} gate both in terms of how it transforms states, and as a matrix:

$$C_{\text{not}}: \begin{array}{lcl} |00\rangle & \rightarrow & |00\rangle \\ |01\rangle & \rightarrow & |01\rangle \\ |10\rangle & \rightarrow & |11\rangle \\ |11\rangle & \rightarrow & |10\rangle \end{array} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

More compactly, we can build the matrices as outer products:

$$C_{\text{not}} = |0\rangle\langle 0| \otimes I_2 + |1\rangle\langle 1| \otimes X_2$$

where

$$I_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \; X_2 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

To see why this compact form is the same as the matrix above, we first expand the outer products:

$$|0\rangle\langle 0| = \begin{array}{c|cc} & 1 & 0 \\ \hline 1 & 1 & 0 \\ 0 & 0 & 0 \end{array}, \quad |1\rangle\langle 1| = \begin{array}{c|cc} & 0 & 1 \\ \hline 0 & 0 & 0 \\ 1 & 0 & 1 \end{array}$$

Plugging in the definitions of I_2 and X_2 from above, this gives us:

$$\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Then we expand the tensor products:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

which is the definition of C_{not}.

Let's use this convention to represent the 3-wire Toffoli gate. We can write this as a big matrix:

$$T:\ \begin{array}{rcl} |000\rangle & \rightarrow & |000\rangle \\ |001\rangle & \rightarrow & |001\rangle \\ |010\rangle & \rightarrow & |010\rangle \\ |011\rangle & \rightarrow & |011\rangle \\ |100\rangle & \rightarrow & |100\rangle \\ |101\rangle & \rightarrow & |101\rangle \\ |110\rangle & \rightarrow & |111\rangle \\ |111\rangle & \rightarrow & |110\rangle \end{array} = \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&1&0 \end{bmatrix}$$

The outer-product version is much simpler:

$$T = |0\rangle\langle 0| \otimes I_2 \otimes I_2 \ + \ |1\rangle\langle 1| \otimes C_{\text{not}}$$

We can capture the effect of the Toffoli gate succinctly by showing what happens to particular input patterns:

$$T(|x,\ y,\ 0\rangle) = |x,\ y,\ x \wedge y\rangle$$

$$T(|1,\ 1,\ z\rangle) = |1,\ 1,\ \bar{z}\rangle$$

$$T(|x,\ y,\ z\rangle) = |x,\ y,\ z \oplus x \wedge y\rangle$$

I said earlier that the Toffoli gate is universal, in the sense that if you have enough of them, you can build a circuit to evaluate any quantum-computable function. Another complete quantum gate is the *Fredkin*, or *controlled-swap*, gate:

$$F = |0\rangle\langle 0| \otimes I_2 \otimes I_2 \ + \ |1\rangle\langle 1| \otimes S_2$$

where S_2 is the 2-qubit *swap operator*:

$$S_2 = |00\rangle\langle 00| \ + \ |01\rangle\langle 10| \ + \ |10\rangle\langle 01| \ + \ |11\rangle\langle 11|$$

If you want to expand these out to see the matrices for yourself, remember to use the standard bases as defined previously. For example,

$$|01\rangle\langle 10| = \begin{array}{c|cccc} & 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array}$$

A purely quantum gate

The gates we've seen so far bear a close similarity to classical logic. Let's look at a gate that has no counterpart in the classical world: the one-wire *square-root-of-not* gate.

Here's the definition of this gate:

$$\sqrt{\text{NOT}}:\ \begin{array}{l} |0\rangle \rightarrow \frac{1}{\sqrt{2}}\left(|0\rangle - |1\rangle\right) \\ |1\rangle \rightarrow \frac{1}{\sqrt{2}}\left(|0\rangle + |1\rangle\right) \end{array} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$$

What this gate tells us is that if we feed in a qubit in a pure state, we'll get back a state that's an equal superposition of the two bases. That gives us a way to create superimposed states from pure ones, which does come in handy. But the really interesting thing about this gate is apparent when we apply it twice. If you apply this transformation to a single input twice in a row, or equivalently square the above matrix, you'll get this result:

$$\sqrt{\text{NOT}}^2:\ \begin{array}{l} |0\rangle \rightarrow |1\rangle \\ |1\rangle \rightarrow |0\rangle \end{array} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

Applying $\sqrt{\text{NOT}}$ to a pure input creates a perfectly unpredictable blend of the standard bases, while a second application inverts the input qubit (note that because probabilities are squares of the weights, the probability of observing a qubit in state $-|1\rangle$ is $-1^2 = 1$).

The square-root-of-not gate has no corollary in the classical world—it's truly a quantum gate. When applied to an input it creates pure randomness but when applied again to that result it eliminates the randomness to create a single, purely deterministic state that's the input state's opposite.

A quantum full adder

Let's wrap up this month with a simple but useful quantum circuit. We'll build a *full adder*. The classical version of this circuit takes three 1-bit inputs, typically called a, b, and C_{in}. The idea is that we want to add up two big binary numbers one bit at a time. So we add up first the least-significant bits, then the next-most-significant bits, and so on. At each step we add the incoming carry bit C_{in}, a, and b, and output a single sum bit s and a *carry-out* bit C_{out}. Here's the classical truth table:

	a	b	C_{in}	C_{out}	s
	0	0	0	0	0
	0	0	1	0	1
	0	1	0	0	1
ADD:	0	1	1	1	0
	1	0	0	0	1
	1	0	1	1	0
	1	1	0	1	0

In short, add up the values of a, b, and C_{in}, and treat C_{out} and s as the high- and low-order bits of their sum. The value of C_{out} becomes the *carry-in* bit C_{in} to the next stage.

To turn this into a quantum circuit, think of the inputs as qubits rather than regular bits. This doesn't require us to change anything directly, but let's adopt a quantum style and build the circuit using only C_{not} and Toffoli gates.

If you enjoy this sort of problem, you might want to take a shot at designing this circuit yourself before looking at the following text and figures. The solution doesn't require a lot of gates. If you find your circuit getting very complicated, you're probably trying too hard.

Figure 7 shows two ways to build a quantum full-

adder. You can verify for yourself that when the inputs aren't superpositioned, the outputs follow the truth table for a classical adder.

Here's the beautiful part: if the input qubits are in a superposition state, then so are the outputs. This means that if we prepare $a = (1/\sqrt{2})\,(|0\rangle + |1\rangle)$, and b and C_{in} to similar mixed states, then the adder presents at s and C_{out} all possible values from the truth table simultaneously.

When we examine, or measure, the input bits, then the output bits are immediately projected to the appropriate values for those inputs. In effect all the possible sums for all possible inputs are computed simultaneously and are just sitting there, waiting for us to pick one out by measuring some of the qubits. Because the circuit is reversible, we can measure the outputs instead if we want, and from them, deduce the inputs that created them.

The adder of Figure 7a has a characteristic that's common in quantum circuits: the inputs appear unchanged at the outputs. This is certainly one way to make reversibility easy, because we don't have to expend any effort to recover the input values.

We often express this by writing a quantum circuit's input as a concatenation of two registers: the input signal itself and a placeholder for the result. This placeholder is usually set to state $|00\ldots0\rangle$, but it can be in any prepared state. Then the circuit simply replaces those qubits in the output register.

If a circuit U applies a function f, then we write this as

$$U_f\left|x,\ 0\right\rangle \rightarrow \left|x, f(x)\right\rangle$$

Next time

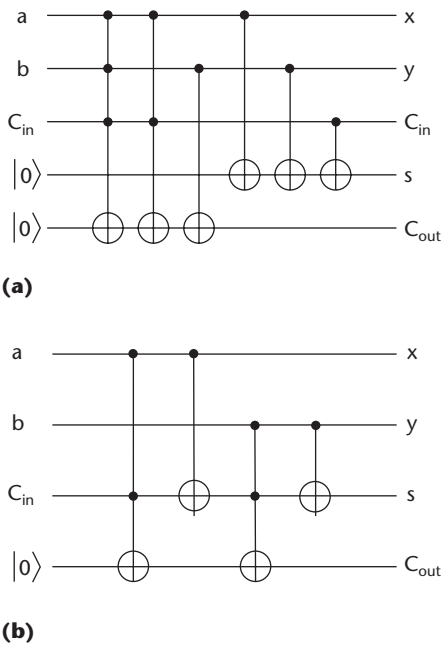
My goal here was to lay out the basics of quantum computing notation and computational techniques. Now that we can build gates, we can look at quantum computation's building blocks.

In the next issue, I'll continue our journey into quantum computing by presenting some simple quantum circuits, and then taking a look at some of the unexpected phenomena and computational and cryptographic techniques that result from this theory. ■

Acknowledgments

Thanks to Don Mitchell and Kirk Olynyk for comments and suggestions.

Readers may contact Glassner directly by email at andrew_glassner@yahoo.com.



7 (a) A quantum full adder that uses a total of six gates: three Toffoli gates and three C_{not} gates. (b) A different adder that uses only four gates: two Toffoli gates and two C_{not} gates.

Further Reading

As I mentioned in my last column, much of the research in the field is available in electronic form on the publicly accessible Los Alamos Physics Preprint Archive at Los Alamos National Labs, located at http://xxx.lanl.gov/abs/quant-ph. The LANL server contains mostly Postscript papers, but it can convert many of them into PDF and other output formats on demand.

My principal sources for this month's article included *An Introduction To Quantum Computing for Non-Physicists* by Eleanor Rieffel and Wolfgang Polak (LANL 9809016), *Quantum Computation* by Dorit Aharonov (LANL 9812037), *Quantum Gates and Circuits* by David P. DiVincenzo (LANL 9705009), and *Basic Concepts in Quantum Computation* by Artur Ekert, Patrick Hayden, and Hitoshi Inamori (LANL 0011013).

Quantum Networks for Elementary Arithmetic Operations by Vlatko Vedral, Adriano Barenco, and Artur Ekert (LANL 9511018) discusses more sophisticated approaches to quantum addition. The original discussion of the Toffoli gate appears in "Reversible Computing," by Tommaso Toffoli, in *Automata, Languages and Programming*, J.W. de Bakker and J. van Leeuwen, eds., Springer-Verlag, New York, 1980, pp. 632-644.