*Last revised 4/5/06*

# LECTURE NOTES ON QUANTUM COMPUTATION

*Cornell University, Physics 481-681, CS 483; Spring, 2006*



## II. Quantum Computation: General features and some simple examples

### A. The general computational process

We would like a suitably programmed quantum computer to act on a number $x$ to produce another number $f(x)$ for some specified function $f$. Appropriately interpreted, with an accuracy that increases with increasing $k$, we can treat all such numbers as non-negative integers less than $2^k$. Each integer is represented in the quantum computer by the corresponding computational-basis state of $k$ Qbits.

If we specify $x$ as an $n$-bit integer and $f(x)$ as an $m$-bit integer, then we shall need at least $n+m$ Qbits: a set of $n$-Qbits, called the *input register*, to represent $x$, and another set of $m$-Qbits, called the *output register*, to represent $f(x)$. Qbits being a scarce commodity, you might wonder why we need separate registers for input and output. One important reason is that if $f(x)$ assigns the same value to different values of $x$, as many interesting functions do, then the computation cannot be inverted if its only effect is to transform the contents of a single register from $x$ to $f(x)$. Having separate registers for input and output is standard practice in the classical theory of reversible computation. Since (as remarked in Chapter 1 and expanded on in Section C of this chapter) quantum computers must operate reversibly (except for measurement gates) to perform their magic, quantum computers are generally designed to operate with both input and output registers. We shall find that this dual-register procedure can be exploited by a quantum computer in some strikingly nonclassical ways.

The computational process will generally require many Qbits besides the $n + m$ in the input and output registers, but we shall ignore these additional Qbits for the moment, viewing a computation of $f$ as doing nothing more than applying a unitary transformation, $\mathbf{U}_f$ to the $n+m$ Qbits of the input and output registers. We shall return to the fundamental question of why the additional Qbits can be ignored in Section C, only noting for now that it will be the reversibility of the computation that makes it possible for them to be ignored.

To define unitary transformations $\mathbf{U}$ it is enough to define their action on any basis, since any other state $|\Psi\rangle$ must be a linear superposition of basis states, and therefore, since unitary transformations are linear, their action on $|\Psi\rangle$ is entirely determined by their action on the basis. In most cases of interest one takes the basis in which $\mathbf{U}$ is defined to be the computational (classical) basis. In many cases (such as the definition below of $\mathbf{U}_f$) the action of $\mathbf{U}$ on each computational basis state is simply to produce another computational basis state. Since $\mathbf{U}$ must have an inverse, it therefore acts on the entire

computational basis as a permutation, just as a classical transformation does. Since linear transformations that take orthonormal bases into other orthonormal bases are necessarily unitary, and since a permutation of an orthonormal basis trivially remains an orthonormal basis, any invertible linear transformation that is defined to take computational basis states into other computational basis states, is automatically unitary. Putting the point another way, *any classically meaningful reversible transformation on Cbits, has a linear extension to a unitary transformation on Qbits that acts as the classical transformation when restricted to states of the computational basis.* I shall use this fact repeatedly in defining unitary transformations on Qbits, without explicitly reminding you each time of this justification for it.

The standard quantum computational protocol defines the action of $\mathbf{U}_f$ on every computational basis state $|x\rangle_n|y\rangle_m$ of the $n+m$ Qbits making up the input and output registers as follows:

$$\mathbf{U}_f\big(\,|x\rangle_n|y\rangle_m\big) = |x\rangle_n|y\oplus f(x)\rangle_m, \tag{2.1}$$

where $\oplus$ indicates modulo-2 bitwise addition (without carrying), a straightforward generalization of the single-bit $\oplus$ defined in Chapter 1. If $x$ and $y$ are $m$-bit integers whose $j$th bits are $x_j$ and $y_j$, then $x\oplus y$ is the $m$-bit integer whose $j$-th bit is $x_j\oplus y_j$. Thus $1101\oplus 0111 = 1010$.

If the initial value represented by the output register is $y=0$ then we have

$$\mathbf{U}_f\big(\,|x\rangle_n|0\rangle_m\big) = |x\rangle_n|f(x)\rangle_m \tag{2.2}$$

and we do indeed end up with $f(x)$ in the output register. Regardless of the initial value of $y$ the input register ends up back in its initial state $|x\rangle_n$.

The transformation (2.1) is clearly invertible. Indeed $\mathbf{U}_f$ is its own inverse:

$$\mathbf{U}_f\mathbf{U}_f\big(\,|x\rangle|y\rangle\big) = \mathbf{U}_f\big(\,|x\rangle|y\oplus f(x)\rangle\big) = |x\rangle|y\oplus f(x)\oplus f(x)\rangle = |x\rangle|y\rangle, \tag{2.3}$$

since $z\oplus z = 0$ for any $z$. (From now on I shall keep the subscripts reminding you of the numbers of Qbits only when it is important to emphasize what those numbers are.) Since $\mathbf{U}$ is invertible and takes computational basis states into other computational basis states, its linear extension to all Qbit states is indeed unitary.

The form (2.2) inspires one of the most important tricks of the quantum computational repertoire. If we apply to each Qbit in the two-Qbit state $|0\rangle|0\rangle$ the 1-Qbit Hadamard transformation $\mathbf{H}$ (introduced in Chapter I, Section B), then we get

$$\big(\mathbf{H}\otimes\mathbf{H}\big)\big(|0\rangle|0\rangle\big) = \big(\mathbf{H}|0\rangle\big)\big(\mathbf{H}|0\rangle\big) = \tfrac{1}{\sqrt{2}}\big(|0\rangle+|1\rangle\big)\tfrac{1}{\sqrt{2}}\big(|0\rangle+|1\rangle\big) =$$

$$\tfrac{1}{2}\big(|0\rangle|0\rangle+|0\rangle|1\rangle+|1\rangle|0\rangle+|1\rangle|1\rangle\big) = \tfrac{1}{2}\big(|0\rangle_2+|1\rangle_2+|2\rangle_2+|3\rangle_2\big). \tag{2.4}$$

This generalizes to the $n$-fold tensor product of $n$ Hadamard transforms (written $\mathbf{H}^{\otimes n}$) applied to the $n$-Qbit state $|0\rangle_n$:

$$\mathbf{H}^{\otimes n}|0\rangle_n = \frac{1}{2^{n/2}}\sum_{0\le x<2^n}|x\rangle_n. \tag{2.5}$$

So if the initial state of the input register is $|0\rangle_n$ and we apply an $n$-fold Hadamard transformation to that register, its state becomes an equally weighted superposition of all possible $n$-Qbit inputs. If we then apply $\mathbf{U}_f$ to that superposition, with 0 initially in the output register, then by linearity we get from (2.5) and (2.2)

$$\mathbf{U}_f\big(\mathbf{H}^{\otimes n}\otimes \mathbf{1}_m\big)\big(\,|0\rangle_n|0\rangle_m\big) = \frac{1}{2^{n/2}}\sum_{0\le x<2^n}\mathbf{U}_f\big(\,|x\rangle_n|0\rangle_m\big) = \frac{1}{2^{n/2}}\sum_{0\le x<2^n}|x\rangle_n|f(x)\rangle_m. \tag{2.6}$$

This contains an important part of the magic that underlies quantum computation. If before letting $\mathbf{U}_f$ act, we merely apply a Hadamard transformation to every Qbit of the input register, initially in the standard state $|0\rangle_n$, the result of the computation is described by a state whose structure cannot be explicitly specified without knowing the result of all $2^n$ evaluations of the function $f$. So if we have a mere hundred Qbits in the input register, initially all in the state $|0\rangle_{100}$ (and $m$ more in the ouput register), if a hundred Hadamard gates act on the input register before the application of $\mathbf{U}_f$, then the form of the final state contains the results of $2^{100}\approx 10^{30}$ evaluations of the function $f$. A billion billion trillion evaluations! This apparent miracle is called *quantum parallelism.*

But a major part of the miracle is only apparent. I did not say that the result of the calculation *is* $2^n$ evaluations of $f$, though many practitioners of quantum computation are rather careless about making such a claim. All I said (and all one can say) is that those explicit evaluations occur in the form of the state that describes the output of the computation. Before drawing extravagant practical or only, as many practitioners of quantum computation are wont to do, metaphysical conclusions from quantum parallelism, it is important to remember that *when you have a collection of Qbits in a definite but unknown state, there is in general no way to find out what that state is.*

If there *were* a way to learn the state of such a set of Qbits, then I myself would join in the rhapsodic chorus. (Typical verses: "Where were all those calculations done? In parallel universes!" "The possibility of quantum computation has established the existence of the multiverse." "Quantum computation achieves its power by dividing the computational task among huge numbers of parallel worlds.") But there is no way to learn the state. The only way to extract information from Qbits is to subject them to a *measurement.*

When we send all $n+m$ Qbits through measurement gates, the Born rule tells us that if the state of the registers is (2.6), then with equal probability the result of measuring the Qbits in the input register will be any one of the values of $x$ less than $2^n$, while the result of measuring the Qbits in the ouput register will be the value of $f$ for that particular value of $x$. So by measuring the Qbits we can learn a single value of $f$ as well as learning a single (random) $x_0$ at which $f$ has that value. After the measurement the state of the registers reduces to $|x_0\rangle|f(x_0)\rangle$ and we are no longer able to learn anything about the values of $f$ for any other values of $x$.

So although we can learn something from the output of the "parallel computation", it is nothing more than what we would have learned had we simply run the computation

starting with a classical state $|x\rangle$ in the input register, with the value of $x$ chosen randomly. That, of course, could have been done with an ordinary classical computer. To be sure, a hint of a miracle remains — hardly more than the smile of the Cheshire cat — in the fact that in the quantum case the random selection of the $x$, for which $f(x)$ can be learned, is only made *after* the computation has been carried out. (To assert that the selection was made *before* the computation was done, is to make the same error as asserting that a Qbit in a superposition of the states $|0\rangle$ and $|1\rangle$ is actually in one or the other of them, as discussed in Chapter 1, Section E.) This is a characteristic instance of what journalists like to call "quantum weirdness", in that (a) it is indeed vexing to contemplate the fact that the choice of the value of $x$ for which $f$ can be learned is made only after — quite possibly long after — the computation has been finished, but (b) since that choice is inherently random — beyond anyone's power to control in any way whatever — it does not matter for any practical purpose whether the selection was made miraculously after or boringly before the calculation was executed.

If, of course, there were an easy way to make copies of the output state prior to making the measurement, without running the whole computation over again, then one could, with high probability, learn the values of $f$ for several different (random) values of $x$. But such copying is prohibited by an elementary result called the "no-cloning theorem", which states that there is no such duplication procedure: there is no unitary transformation that can take a state $|\psi\rangle_n|0\rangle_n$ into the state $|\psi\rangle_n|\psi\rangle_n$ for arbitrary $|\psi\rangle_n$.

The no-cloning theorem is an immediate consequence of linearity. If

$$\mathbf{U}\big(|\psi\rangle|0\rangle\big) = |\psi\rangle|\psi\rangle \quad \text{and} \quad \mathbf{U}\big(|\phi\rangle|0\rangle\big) = |\phi\rangle|\phi\rangle, \tag{2.7}$$

then it follows from linearity that

$$\mathbf{U}\big(a|\psi\rangle + b|\phi\rangle\big)|0\rangle = a\mathbf{U}|\psi\rangle|0\rangle + b\mathbf{U}|\phi\rangle|0\rangle = a|\psi\rangle|\psi\rangle + b|\phi\rangle|\phi\rangle. \tag{2.8}$$

But if $\mathbf{U}$ cloned arbitrary inputs, we would have

$$\begin{aligned}\mathbf{U}\big(a|\psi\rangle + b|\phi\rangle\big)|0\rangle &= \big(a|\psi\rangle + b|\phi\rangle\big)\big(a|\psi\rangle + b|\phi\rangle\big) \\ &= a^2|\psi\rangle|\psi\rangle + b^2|\phi\rangle|\phi\rangle + ab|\psi\rangle|\phi\rangle + ab|\phi\rangle|\psi\rangle,\end{aligned} \tag{2.9}$$

which differs from (2.8) unless one of $a$ or $b$ is zero. Surprisingly, this theorem was not proved until half a century after the discovery of quantum mechanics, presumably because it took that long for it to occur to somebody that it was an interesting proposition to formulate.

Of course the ability to clone to a reasonable degree of approximation would also be useful. But even this is impossible. Suppose $\mathbf{U}$ approximately cloned both $|\phi\rangle$ and $|\psi\rangle$:

$$\mathbf{U}\big(|\psi\rangle|0\rangle\big) \approx |\psi\rangle|\psi\rangle \quad \text{and} \quad \mathbf{U}\big(|\phi\rangle|0\rangle\big) \approx |\phi\rangle|\phi\rangle. \tag{2.10}$$

Then since unitary transformations preserve inner products, since the inner product of a tensor product of states is the (ordinary) product of their inner products, and since $\langle 0|0\rangle = 1$, it follows from (2.10) that

$$\langle\phi|\psi\rangle \approx \langle\phi|\psi\rangle^2. \tag{2.11}$$

But this requires that $\langle\phi|\psi\rangle$ is either close to 1 or close to 0. Hence a unitary transformation can come close to cloning both of two states $|\psi\rangle$ and $|\phi\rangle$ only if the states are very nearly the same, or very close to being orthogonal. In all other cases at least one of the two states will be badly copied.

If this were the full story, nobody would be interested in quantum computation except philosophers. The Department of Defense and the National Security Agency are interested because there are more clever things one can do. Typically these involve applying additional unitary transformations to one or both of the input and output registers before measuring the Qbits, sometimes applying such unitaries both before and after applying $\mathbf{U}_f$. These transformations are cleverly chosen so that when one finally does measure the Qbits, one extracts useful information about *relations* between the values of $f$ for several different values of $x$, which a classical computer could only get by making several independent evaluations. The price one pays for this relational information is to lose the possibility of learning the actual value $f(x)$ for any individual $x$ whatever. This tradeoff of one kind of information for another is typical of quantum computation, and typical of quantum physics in general, where it is called the *uncertainty principle*. The principle was first enunciated by Werner Heisenberg in the context of mechanical information — the position of a particle vs. its momentum.

So it is wrong and deeply misleading to say that in the process that assigns the state (2.6) to the Qbits, the quantum computer has evaluated the function $f(x)$ for all $x$ in the range $0 \le x < 2^n$. Such assertions are based on the mistaken view that the quantum state encodes a property inherent in the Qbits. All the state encodes are the possibilities available for the extraction of information from those Qbits. You might keep this in mind as we now examine some of the specific ways in which this nevertheless permits a quantum computer to perform tricks that no classical computer can accomplish.

### B. Deutsch's Problem

Deutsch's problem is the simplest example of a quantum tradeoff that sacrifices particular information to acquire relational information. A crude version of it appeared in a 1985 paper by David Deutsch that, together with a 1982 paper by Richard Feynman, launched the whole field. In that early version the trick could only be successfully executed half the time. It took a while for people to realize that the trick could be accomplished every single time. Here is how it works:

Let both input and output registers contain only a single Qbit, so we are exploring

TABLE 2.1: The 4 distinct functions $f_j(x)$ that take one bit into one bit.

| | $x=0$ | $x=1$ |
|---|---|---|
| $f_0$ | 0 | 0 |
| $f_1$ | 0 | 1 |
| $f_2$ | 1 | 0 |
| $f_3$ | 1 | 1 |

---

functions $f$ that take a single bit into a single bit. There are two rather different ways to think about such functions:

(1) The first way is to note that there are just four such functions, as shown in Table 2.1. Suppose we are given a black box that calculates one of these four functions in the usual quantum computational format, by performing the unitary transformation

$$\mathbf{U}_f\big(\,|x\rangle|y\rangle\big) = |x\rangle|y \oplus f(x)\rangle, \tag{2.12}$$

where the state on the left is that of the 1-Qbit input register ($i$), and the state on the right is that of the 1-Qbit output register ($o$). Using the forms in Table 2.1 and the explicit structure (2.12) of $\mathbf{U}_f$ you can easily confirm that

$$\mathbf{U}_{f_0} = \mathbf{1}, \;\; \mathbf{U}_{f_1} = \mathbf{C}_{io}, \;\; \mathbf{U}_{f_2} = \mathbf{C}_{io}\mathbf{X}_o, \;\; \mathbf{U}_{f_3} = \mathbf{X}_o, \tag{2.13}$$

where $\mathbf{1}$ is the (2-Qbit) unit operator, $\mathbf{C}_{io}$ is the controlled-NOT with the input Qbit as control and the output as target, $\mathbf{X}_i$ flips only the input register, and $\mathbf{X}_o$ flips only the output register. These possibilities are illustrated in the circuit diagram of Figure 2.1.

Suppose we are given a black box that executes $\mathbf{U}_f$ for one of the four functions, but are not told which of the four operations (2.13) the box carries out. We can, of course, find out by letting the black box act twice — first on $|0\rangle|0\rangle$ and then on $|1\rangle|0\rangle$. But suppose we are only allowed to let the box act once. What can we learn about $f$?

In a classical computer, where we are effectively restricted to letting the black box act on Qbits in one of the four computational basis states, we can learn either the value of $f(0)$ (if we let $\mathbf{U}_f$ act on either $|0\rangle|0\rangle$ or $|0\rangle|1\rangle$) or the value of $f(1)$ (if we let $\mathbf{U}_f$ act on either $|1\rangle|0\rangle$ or $|1\rangle|1\rangle$). If we chose to learn the value of $f(0)$. then we can restrict $f$ to being either $f_0$ or $f_1$ (if $f(0) = 0$) and we can restrict $f$ to being either $f_2$ or $f_3$ (if $f(0) = 1$). If we chose to learn the value of $f(1)$, then we can restrict $f$ to being either $f_0$ or $f_2$ (if $f(1) = 0$) or to being either $f_1$ or $f_3$ (if $f(1) = 1$).

Suppose, however, we want to learn whether $f$ is constant ($f(0) = f(1)$, satisfied by $f_0$ and $f_3$) or not constant ($f(0) \neq f(1)$, satisfied by $f_1$ and $f_2$). We then have no choice with

a classical computer but to evaluate both $f(0)$ and $f(1)$ and compare them. In this way we determine whether or not $f$ is constant, but we have to extract complete information about $f$ to do so. We have to run $\mathbf{U}_f$ twice.

Remarkably, it turns out that with a quantum computer we do not have to run $\mathbf{U}_f$ twice to determine whether or not $f$ is constant. We can do this in a single run. Interestingly, when we do this we learn nothing whatever about the individual values of $f(0)$ and $f(1)$, but we are nevertheless able to answer the question about their relative values: whether or not they are the same. Thus we get less information than we get in answering the question with a classical computer, but by renouncing the possibility of acquiring that part of the information which is irrelevant to the question we wish to answer, we can get the answer with only a *single* application of the black box.

(2) There is a second way to look at Deutsch's problem, which gives it some nontrivial mathematical content. One can think of $x$ as specifying a choice of two different inputs to an elaborate subroutine that requires many additional Qbits, and one can think of $f(x)$ as characterizing a two-valued property of the output of that subroutine. For example $f(x)$ might be the value of the millionth bit in the binary expansion of $\sqrt{2+x}$ so that $f(0)$ is the millionth bit in the expansion of $\sqrt{2}$ while $f(1)$ is the millionth bit of $\sqrt{3}$. In this case the input register feeds data into the subroutine and the subroutine reports back to the output register.

In the course of the calculation the input and output registers will in general become entangled with the additional Qbits used by the subroutine. If the entanglement persists to the end of the calculation, the input and output registers will have no final states of their own, and it will be impossible to describe the computational process as the simple unitary transformation (2.1). We shall see in Section C, however, that it is possible to set things up so that at the end of the computation the additional Qbits required for the subroutine are no longer entangled with the input and output registers, and that the additional Qbits can indeed be ignored, with the simple linear transformation (2.1) correctly characterizing the net effect of the computation on those two registers.

Under interpretation (1) of Deutsch's problem, the question of whether $f$ is or is not constant is the somewhat artificial question of learning something about the nature of the black-box that executes $\mathbf{U}_f$ without actually opening it up and looking inside. Under interpretation (2) it becomes the nontrivial question of whether the millionth bits of $\sqrt{2}$ and $\sqrt{3}$ agree or disagree. Under either interpretation, to answer the question with a classical computer we can do no better than to run the black box twice, with both 0 and 1 as inputs, and compare the two outputs. In the quantum case we could try the standard trick, preparing the input register in the superposition $\frac{1}{\sqrt{2}}\big(|0\rangle + |1\rangle\big)$. The final state of the 1-Qbit input and output registers would then be

$$\mathbf{U}_f\big(\mathbf{H}\otimes\mathbf{1}\big)\big(|0\rangle|0\rangle\big) = \tfrac{1}{\sqrt{2}}|0\rangle|f(0)\rangle + \tfrac{1}{\sqrt{2}}|1\rangle|f(1)\rangle, \tag{2.14}$$

as described in (2.6). If we measured the input and ouput registers we could learn the millionth bit of either $\sqrt{2}$ or $\sqrt{3}$ (and learn which it is we have learned). The choice of

which we did learn would be completely random, so this procedure offers no improvement on the classical situation.

It was first noticed, however, that there are unitary transformations one can apply to the state (2.14) before carrying out the measurement, that, depending on the outcome, enable you half the time state with assurance whether or not $f(0) = f(1)$. Some time after that, it was realized that you can *always* answer the question, provided you apply appropriate unitary transformations before as well as after running the computation. Here is how the trick is done:

To get the output (2.14) we took the input to $\mathbf{U}_f$ to be the state

$$\big(\mathbf{H}\otimes\mathbf{1}\big)\big(|0\rangle|0\rangle\big). \tag{2.15}$$

Instead of doing this, we again start with both input and output registers in the state $|0\rangle$, but then we apply the NOT operation $\mathbf{X}$ to both registers, followed by an application of the Hadamard transformt to both. Since $\mathbf{X}|0\rangle = |1\rangle$ and $\mathbf{H}|1\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$, the input to $\mathbf{U}_f$ is now described by the state

$$\begin{aligned}\big(\mathbf{H}\otimes\mathbf{H}\big)\big(\mathbf{X}\otimes\mathbf{X}\big)\big(|0\rangle|0\rangle\big) = \big(\mathbf{H}\otimes\mathbf{H}\big)\big(|1\rangle|1\rangle\big) &= \Big(\tfrac{1}{\sqrt{2}}|0\rangle - \tfrac{1}{\sqrt{2}}|1\rangle\Big)\Big(\tfrac{1}{\sqrt{2}}|0\rangle - \tfrac{1}{\sqrt{2}}|1\rangle\Big)\\ &= \tfrac{1}{2}\Big(|0\rangle|0\rangle - |1\rangle|0\rangle - |0\rangle|1\rangle + |1\rangle|1\rangle\Big).\end{aligned} \tag{2.16}$$

If we take the state (2.16) as input to $\mathbf{U}_f$, then by linearity the resulting state is

$$= \tfrac{1}{2}\Big(\mathbf{U}_f\big(|0\rangle|0\rangle\big) - \mathbf{U}_f\big(|1\rangle|0\rangle\big) - \mathbf{U}_f\big(|0\rangle|1\rangle\big) + \mathbf{U}_f\big(|1\rangle|1\rangle\big)\Big). \tag{2.17}$$

It follows from the explicit form (2.12) of the action of $\mathbf{U}_f$ on the computational basis states that this is simply

$$\tfrac{1}{2}\Big(|0\rangle|f(0)\rangle - |1\rangle|f(1)\rangle - |0\rangle|\overline{f}(0)\rangle + |1\rangle|\overline{f}(1)\rangle\Big), \tag{2.18}$$

where, as earlier, $\overline{x} = 1\oplus x$ so that $\overline{1} = 0$ and $\overline{0} = 1$, and $\overline{f}(x) = 1\oplus f(x)$. So if $f(0) = f(1)$ the ouput state (2.18) is

$$\tfrac{1}{2}\Big(|0\rangle - |1\rangle\Big)\Big(|f(0)\rangle - |\overline{f}(0)\rangle\Big), \quad f(0) = f(1), \tag{2.19}$$

but if $f(0) \neq f(1)$ then $f(1) = \overline{f}(0)$, $\overline{f}(1) = f(0)$, and the output state (2.18) becomes

$$\tfrac{1}{2}\Big(|0\rangle + |1\rangle\Big)\Big(|f(0)\rangle - |\overline{f}(0)\rangle\Big), \quad f(0) \neq f(1). \tag{2.20}$$

If, finally, we apply a Hadmard transformation to the input register these become

$$|1\rangle\tfrac{1}{\sqrt{2}}\Big(|f(0)\rangle - |\overline{f}(0)\rangle\Big), \quad f(0) = f(1), \tag{2.21}$$

$$|0\rangle\tfrac{1}{\sqrt{2}}\Big(|f(0)\rangle - |\overline{f}(0)\rangle\Big), \quad f(0) \neq f(1). \tag{2.22}$$

Putting together all the operations in a form we can compare with the more straightforward computation (2.14), we have

$$\big(\mathbf{H}\otimes\mathbf{1}\big)\mathbf{U}_f\big(\mathbf{H}\otimes\mathbf{H}\big)\big(\mathbf{X}\otimes\mathbf{X}\big)\big(|0\rangle|0\rangle\big) = \begin{cases} |1\rangle\tfrac{1}{\sqrt{2}}\Big(|f(0)\rangle - |\overline{f}(0)\rangle\Big), & f(0) = f(1) \\ |0\rangle\tfrac{1}{\sqrt{2}}\Big(|f(0)\rangle - |\overline{f}(0)\rangle\Big), & f(0) \neq f(1). \end{cases} \tag{2.23}$$

Thus the state of the input register has ended up as $|1\rangle$ or $|0\rangle$ depending on whether or not $f(0) = f(1)$, so by measuring the *input* register we can indeed answer the question of whether $f(0)$ and $f(1)$ are or are not the same!

Notice also that in either case the output register is left in the state $\frac{1}{\sqrt{2}}|f(0)\rangle - \frac{1}{\sqrt{2}}|\overline{f}(0)\rangle$. Because both terms in the superposition have amplitudes with exactly the same magnitude, if one measures the output register the result is equally likely to be $f(0)$ or $\overline{f}(0)$, so one learns absolutely nothing about the actual value of $f(0)$. The output register contains no useful information at all.

Another way to put it is that the final state of the output register is $\pm\frac{1}{\sqrt{2}}\big(|0\rangle - |1\rangle\big)$ depending on whether $f(0) = 0$ or $f(0) = 1$. Since a change in the overall sign of a state (or, more generally, the presence of an overall complex factor of modulus 1) has no effect on the statistical distribution of measurement outcomes, there is no way to distinguish between these two cases.

Thus the price one has paid to learn whether $f(0)$ and $f(1)$ are or are not the same, is the loss of any information whatever about the actual value of either one of them. One has still eliminated only two of the four possible forms for the function $f$. What the quantum computer has given us is the ability to make this particular discrimination with just a single invocation of the black box, which a classical computer cannot do.

There is a rather neat circuit-theoretic way of seeing why this trick enables one to learn whether or not $f(0) = f(1)$ in just one application of $\mathbf{U}_f$, without going through any of the above algebraic manipulations. This quite different way of looking at Deutsch's problem is illustrated in Figures 2.1-2.3 below.

When one thinks of applying this to learn whether the millionth bits of $\sqrt{2}$ and $\sqrt{3}$ are the same or different, as in the second interpretation of Deutsch's problem, it is really quite startling that one can do this with no more effort (except for a simple modification of the initial and final states) than one uses to calculate the millionth bit of either $\sqrt{2}$ or $\sqrt{3}$. In this case, however, there is an irritating catch, which we return to at the end of Section C.

### C. Why additional subroutine Qbits needn't mess things up.

Now that we have a specific example of a quantum computation to keep in mind, we can address an important issue mentioned in Section A. The computational process

generally requires the use of many Qbits besides the $n+m$ in the input and output registers. In the second way of thinking about Deutsch's problem, it may need to use a great many more. The action of the computer is then described by a unitary transformation $\mathbf{W}_f$ that acts on the space associated with *all* the Qbits: those in the input and output registers, together with the $r$ additional Qbits used in evaluating the function $f$. Only under very special circumstances will this global unitary transformation $\mathbf{W}_f$ on all $n+m+r$ Qbits induce a transformation on the input and output registers that can be be described by a unitary transformation $\mathbf{U}_f$ that acts only on those two registers, as in (2.1). In general the input and output registers will become entangled with the states of the additional $r$ Qbits, and will not even have states of their own.

But if the action of the computer on all $n+m+r$ Qbits has a very special form, then the input and output registers can indeed end up in states of their own, which are related to their initial states through the desired unitary transformation $\mathbf{U}_f$. Let the additional $r$ Qbits start off in some standard initial state $|\psi\rangle_r$, so that the initial state of input register, output register, and additional Qbits is

$$|\Psi\rangle_{n+m+r} = |x\rangle_n |y\rangle_m |\psi\rangle_r. \tag{2.24}$$

Although the $r$ additional Qbits may well become entangled with those in the input and output registers in the course of the calculation — indeed they will have to if they are to serve any useful purpose — we require that when the calculation is finished the final state of the computer must be of the form

$$\mathbf{W}|\Psi\rangle_{n+m+r} = |x\rangle_n |y \oplus f(x)\rangle_m |\phi\rangle_r, \tag{2.25}$$

where the additional $r$ Qbits have a state $|\phi\rangle_r$ of their own, which is independent of the initial state of the input and ouput registers (and therefore offers no hint of the value of $x$ for which $f(x)$ was actually calculated).

Because the final state (2.25) is an unentangled product state between the additional Qbits and the input and output registers, the two registers do indeed have a state of their own at the end of the computation. And because the initial and final states $|\psi_r\rangle$ and $|\phi_r\rangle$ of the additional Qbits are independent of the initial state of the input and output registers, one easily verifies that the final states of the input and output registers will indeed be related to their initial states by a transformation $\mathbf{U}_f$ that is linear on their $(n+m)$-Qbit subspace, as a consequence of the linearity of $\mathbf{W}_f$ on the whole $(n+m+r)$-Qbit subspace. That the linear transformation $\mathbf{U}_f$ is also unitary then follows from the fact that it permutes the states of the computational basis.

Therefore we can indeed ignore the additional $r$ Qbits needed to compute the function $f$ provided both the initial and the final states of the additional Qbits are entirely independent of the initial state of the input and output registers. This independence be achieved by taking advantage of the fact that unitary transformations are *reversible*. It

is this need to disentangle additional registers from the input and output registers that is the fundamental reason why quantum computation must be reversible.

To do the trick, begin the computation by applying a unitary transformation $\mathbf{V}$ that acts only on the input register and the $r$ additional Qbits, doing nothing to the output register. This unitary transformation is designed to construct $f(x)$, given $x$. Next change the $y$ initially in the output register to $y \oplus f(x)$ (as (2.1) requires) without altering any Qbits outside the output register, by using $m$ cNOT gates that combine to make up a unitary transformation $\mathbf{C}_m$. The $m$ control bits are those among the $n + r$ that represent the result of the computation $f(x)$; the $m$ target bits are the ones in the corresponding positions of the output register. Since the state of the $n + r$ Qbits is not altered by the application of $\mathbf{C}_m$, we can then apply the inverse transformation $\mathbf{V}^\dagger$ to restore them to their original state. In this way we have produced the required unitary transformation $\mathbf{W}$, with the final states of the $r$ additional Qbits being the same as their initial state. This whole construction is illustrated in the circuit diagrams of Figures 2.4-2.7.

The need for a procedure like this negates some of the hype one encounters in discussions of Deutsch's problem. It is often said that by using a quantum computer one can learn whether or not $f(x) = f(y)$ in no more time than it takes to perform a single evaluation of $f$. This is true only under the first, arithmetically uninteresting, view of Deutsch's problem. If, however, one is thinking of $f$ as a function of mathematical interest evaluated by an elaborate subroutine, then to evaluate $f$ for a single value of $x$ there is no need to undo the effect of the unitary transform $\mathbf{V}$ on the additional registers. But for the trick that determines whether or not $f(x) = f(y)$ it is absolutely essential to apply $\mathbf{V}^\dagger$ to undo the effect of $\mathbf{V}$, which doubles the time of the computation. This misrepresentation of the situation is not entirely dishonorable, since in almost all other examples the speed-up is considerably more than a factor of two, and the necessary doubling of computational time is an insignificant price to pay.

### D. Some more substantial speed-ups with a quantum computer

#### *1. Bernstein-Vazirani problem.*

Like many of the examples discovered before Shor's factoring algorithm, this has a somewhat artificial character. Its significance lies not in the intrinsic arithmetical interest of the problem, but in the fact that the problem can be solved dramatically and unambiguously faster on a quantum computer.

Let $a$ be an unknown non-negative integer less than $2^n$. Let $f(x)$ take any other such integer $x$ into the modulo 2 sum of the products of corresponding bits of $a$ and $x$, which we denote by $a \cdot x$ (in recognition of the fact that it is a kind of bitwise modulo-2 inner product):

$$a \cdot x = a_0 x_0 \oplus a_1 x_1 \oplus a_2 x_2 \cdots . \tag{2.26}$$

Suppose we have a subroutine that evaluates $f(x) = a \cdot x$. If we want to determine the value of the integer $a$ how many times do we have to call that subroutine? Here and in

all subsequent examples, we shall assume that any Qbits acted on by such subroutines, except for the Qbits of the input and output registers, are returned to their initial state at the end of the computation, as discussed in Section C.

The $m$-th bit of $a$ is $a \cdot 2^m$, since the binary expansion of $2^m$ has 1 in position $m$ and 0 in all the other positions. So with a classical computer we can learn the $n$ bits of $a$ by applying $f$ to the $n$ values $x = 2^m, 0 \le m < n$. This, or any other classical method one can think of, requires $n$ different invocations of the subroutine. But with a quantum computer a *single* invocation is enough to determine $a$ completely, regardless of how big $n$ is!

I first describe the conventional way of seeing how this can be done, and then describe a much simpler way to understand the process. The conventional way exploits a trick (implicitly exploited in our solution to Deutsch's problem) that is useful in dealing with functions like $f$ that act on $n$-Qbits with output to a single Qbit. If the 1-Qbit output register is initially prepared in the state $\mathbf{HX}|0\rangle = \mathbf{H}|1\rangle = \frac{1}{\sqrt{2}}\big(|0\rangle - |1\rangle\big)$ then, since $\mathbf{U}_f$ applied to the computational basis state $|x\rangle_n|y\rangle_1$ flips the value $y$ of the output register if and only if $f(x) = 1$, we have

$$\mathbf{U}_f|x\rangle_n \tfrac{1}{\sqrt{2}}\big(|0\rangle - |1\rangle\big) = (-1)^{f(x)}|x\rangle_n \tfrac{1}{\sqrt{2}}\big(|0\rangle - |1\rangle\big). \tag{2.27}$$

So by taking the state of the 1-Qbit output register to $\frac{1}{\sqrt{2}}\big(|0\rangle - |1\rangle\big)$ we convert a bit flip to an overall change of sign. This becomes useful because of a second trick, which exploits a generalization of the action (2.5) of $\mathbf{H}^{\otimes n}$ on $|0\rangle_n$.

The action of $\mathbf{H}$ on a single Qbit can be compactly summarized as

$$\mathbf{H}|x\rangle_1 = \tfrac{1}{\sqrt{2}}\big(|0\rangle + (-1)^x|1\rangle\big) = \tfrac{1}{\sqrt{2}}\sum_{y=0}^{1}(-1)^{xy}|y\rangle. \tag{2.28}$$

If we apply $\mathbf{H}^{\otimes n}$ to an $n$-Qbit computational basis state state $|x\rangle_n$ we can therefore express the result as

$$\mathbf{H}^{\otimes n}|x\rangle_n = \frac{1}{2^{n/2}}\sum_{y_{n-1}=0}^{1}\cdots\sum_{y_0=0}^{1}(-1)^{\sum_{j=0}^{n-1}x_jy_j}|y_{n-1}\rangle\cdots|y_0\rangle = \frac{1}{2^{n/2}}\sum_{y=0}^{2^n-1}(-1)^{x\cdot y}|y\rangle_n, \tag{2.29}$$

where the product $x \cdot y$ is the one defined in (2.26). (Because $-1$ is raised to the power $\Sigma x_j y_j$, all that matters about that sum is its value modulo 2,)

So if we start with the $n$-Qbit input register in the standard initial state $\mathbf{H}^{\otimes n}|0\rangle$, put the 1-Qbit output register into the state $\mathbf{H}|1\rangle$, apply $\mathbf{U}_f$, and then again apply $\mathbf{H}^{\otimes n}$ to the input register, we get

$$\big(\mathbf{H}^{\otimes n}\otimes\mathbf{1}\big)\mathbf{U}_f\big(\mathbf{H}^{\otimes n}\otimes\mathbf{H}\big)|0\rangle_n|1\rangle_1 =$$

$$\big(\mathbf{H}^{\otimes n}\otimes\mathbf{1}\big)\mathbf{U}_f\Big(\frac{1}{2^{n/2}}\sum_{x=0}^{2^n-1}|x\rangle\Big)\frac{1}{\sqrt{2}}\big(|0\rangle - |1\rangle\big) =$$

$$\frac{1}{2^{n/2}}\Big(\mathbf{H}^{\otimes n}\sum_{x=0}^{2^n-1}(-1)^{f(x)}|x\rangle\Big)\frac{1}{\sqrt{2}}\big(|0\rangle-|1\rangle\big)=$$

$$\frac{1}{2^n}\sum_{x=0}^{2^n-1}\sum_{y=0}^{2^n-1}(-1)^{f(x)+x\cdot y}|y\rangle\frac{1}{\sqrt{2}}\big(|0\rangle-|1\rangle\big). \tag{2.30}$$

We can do the sum over $x$ first. If the function $f(x)$ is $a\cdot x$ then this sum produces the factor

$$\sum_{x=0}^{2^n-1}(-1)^{(a\cdot x)}(-1)^{(y\cdot x)}=\prod_{j=1}^{n}\sum_{x_j=0}^{1}(-1)^{(a_j+y_j)x_j}. \tag{2.31}$$

At least one term in the product vanishes unless every bit $y_j$ of $y$ is equal to the corresponding bit $a_j$ of $a$ — i.e. unless $y=a$. Therefore the entire computational process (2.30) reduces to

$$\mathbf{H}^{\otimes(n+1)}\mathbf{U}_f\mathbf{H}^{\otimes(n+1)}|0\rangle_n|1\rangle_1=|a\rangle_n|1\rangle_1, \tag{2.32}$$

where I have applied a final $\mathbf{H}$ to the 1-Qbit output register to make the final expression look a little neater and more symmetric. (I've also restored subscripts to the state symbols for the $n$-Qbit input register and 1-Qbit output register.)

So by putting the input and output registers into the appropriate initial states, after a single invocation of the subroutine followed by an application of $\mathbf{H}^{\otimes n}$ to the input register, the state of the input register becomes $|a\rangle$. All $n$ bits of the number $a$ can now be determined by measuring the input register, even though we have called the subroutine only once!

There is a second, complementary way to look at the Bernstein-Vazirani problem that bypasses all of the preceding analysis, making it evident why (2.32) holds, by examining a few circuit diagrams. The idea is to note, just as we did for the black box of Deutsch's problem in (2.13), that the actions of the black boxes that implement $\mathbf{U}_f$ for the different available choices of $f$ are identical to the actions of some simple circuits.

When $f(x)=a\cdot x$, $\mathbf{U}_f$ flips the one-Qbit output register once, whenever a bit of $x$ and the corresponding bit of $a$ are both 1. When the state of the input register is $|x\rangle_n$ this action can be performed by a collection of cNOT gates all targeted on the output register. There is one cNOT for each non-zero bit of $a$, controlled by the Qbit representing the corresponding bit of $x$. The combined effect of these cNOT gates on every computational basis state is precisely that of $\mathbf{U}_f$. Therefore the effect of any other transformations preceding and/or following $\mathbf{U}_f$ can be understood by examining their effect on this equivalent collection of cNOT gates, even though $\mathbf{U}_f$ may actually be implemented in a completely different way.

The enconding of the sought-for value of $a$ in the disposition of the equivalent cNOT gates, is illustrated in Figure 2.8. The application (2.32) of $\mathbf{H}$ to every Qbit in the input and output registers both before and after the application of $\mathbf{U}_f$, pictured in Figure 2.9, converts every cNOT gate in the equivalent representation of $\mathbf{U}_f$ from $\mathbf{C}_{ij}$ to $\big(\mathbf{H}_i\mathbf{H}_j\big)\mathbf{C}_{ij}\big(\mathbf{H}_i\mathbf{H}_j\big)=$

$\mathbf{C}_{ji}$, as shown in Chapter 1 and pictured in Fig. 10. After this reversal of target and control Qbits, the output register controls every one of the cNOT's, and since it contains a 1, every one of the NOT operators acts. That action is to flip just those Qbits of the input register for which the corresponding bit of $a$ is 1. Since the input register starts in the state $|0\rangle_n$, this changes the state of each Qbit of the input register to to $|1\rangle$, if and only if it corresponds to a non-zero bit of $a$. As a result, the state of the input register is converted from $|0\rangle_n$ to $|a\rangle_n$, just as (2.32) asserts.

Note (as part of your education in quantum mechanics) how different these two explanations are. The first applies $\mathbf{U}_f$ to the quantum superposition of all possible inputs, and then applies operations which lead to perfect destructive interference of all states in the superposition except for the one in which the input register is in the state $|a\rangle$. The second suggests a specific mechanism for representing the subroutine that executes $\mathbf{U}_f$ and then shows that sandwiching such a mechanism between Hadamards automatically imprints $a$ on the input register. Quantum mechanics appears in the second method only through the possibility it allows for reversing the control and target Qbits of a cNOT operation solely through the use of one-Qbit (Hadamard) gates.

One also can reverse control and target bits of a cNOT classically, but this requires the use of 2-Qbit SWAP gates, rather than 1-Qbit Hadamards. You should convince yourself that this solution to the Bernstein-Vazirani problem using Hadamards no longer works if you try to replace all the Hadamard gates by any arrangement of SWAP gates.

*2. Simon's problem.*

Simon's problem, like the Bernstein-Vazirani problem, has an $n$-bit non-zero number $a$ built into the action of a subroutine $\mathbf{U}_f$, and the aim is to learn the value of $a$ with as few invocations of the subroutine as possible. In the Bernstein-Vazirani problem a classical computer must call the subroutine $n$ times to determine the value of $a$, while a quantum computer need call the subroutine only once. The number of calls grows linearly with $n$ in the classical case, while being independent of $n$ in the quantum case. In Simon's problem the speed-up with a quantum computer is substantially more dramatic. With a classical computer the number of times one must call the subroutine to solve Simon's problem grows exponentially with increasing $n$, but with a quantum computer it only grows linearly.

This spectacular speed-up involves a probabilistic element characteristic of many quantum computations. The characterization of how the number of calls of the subroutine scales with the number of bits in $a$ applies not to calculating it directly, but to learning $a$ with high probability.

The subroutine $\mathbf{U}_f$ in Simon's problem evaluates a function $f$ on $n$ bits that is two to one — i.e. it is a function from $n$ to $n-1$ bits. It is constructed so that $f(x) = f(y)$ if and only if the $n$-bit integers $x$ and $y$ are related by $x = y \oplus a$ or, equivalently and more symmetrically, $x \oplus y = a$, where $\oplus$ again denotes bitwise modulo-2 addition. One can think of this as a period-finding problem. One is told that $f$ is periodic under bitwise

modulo-2 addition,

$$f(x \oplus a) = f(x) \tag{2.33}$$

for all $x$, and the problem is to find the period $a$. Simon's problem is thus a precursor of Shor's much more subtle and spectacularly more useful period-finding algorithm — the heart of his factoring procedure — where one finds the unknown period $a$ of a function that is periodic under ordinary addition: $f(x + a) = f(x)$.

To find the value of $a$ in (2.33) with a classical computer all you can do is feed the subroutine different $x_1, x_2, x_3, \ldots$, listing the resulting values of $f$ until you stumble on an $x_j$ that yields one of the previously computed values $f(x_i)$. You then know that $a = x_j \oplus x_i$. At any stage of the process prior to success, if you have picked $m$ different values of $x$, then all you know is that $a \neq x_i \oplus x_j$ for all pairs of previously selected values of $x$. You have therefore eliminated at most $\frac{1}{2}m(m-1)$ values of $a$. (You would have eliminated fewer values of $a$ if you were careless enough to pick an $x$ equal to $x_i \oplus x_j \oplus x_k$ for three values of $x$ already selected.) Since there are $2^n - 1$ possibilities for $a$, your chances of success will not be appreciable while $\frac{1}{2}m(m-1)$ remains small compared with $2^n$. You are unlikely to succeed until $m$ becomes of the order of $2^{n/2}$, so the number of times the subroutine has to be run to give an appreciable probability of determining $a$ grows with the number of bits $n$ as $2^{n/2}$ — i.e. exponentially. If $a$ has 100 bits a classical computer would have to run the subroutine about $2^{50} \approx 10^{15}$ times to have a significant chance of determining $a$. At ten million calls per second it would take about three years.

In contrast, a quantum computer can determine $a$ with high probability (say less than one chance in a million of failing) by running the subroutine not very much more than $n$ times — e.g. with about 120 invocations of the subroutine if $a$ has 100 bits. This remarkable feat can be accomplished with the following strategy:

As usual, we apply the unitary transformation $\mathbf{U}_f$ only after the state of the input register has been transformed into the uniformly weighted superposition (2.5) of all possible inputs by the application of $\mathbf{H}^{\otimes n}$, so that the effect of $\mathbf{U}_f$ is to take the input and output registers into the entangled state

$$\frac{1}{2^{n/2}} \sum_{x=0}^{2^n-1} |x\rangle|f(x)\rangle. \tag{2.34}$$

If we now subject only the output register to a measurement, then the measurement gate is equally likely to indicate each of the $2^{n-1}$ different values of $f$. Since each value is associated in (2.34) with precisely two values of $x$, the generalized Born rule tells us that the input register will be left in the state

$$\tfrac{1}{\sqrt{2}}\big(|x_0\rangle + |x_0 \oplus a\rangle\big) \tag{2.35}$$

for the (random) value of $x_0$ for which $f(x_0)$ agrees with the (random) value of $f$ given by the measurement.

At first glance this looks like great progress. We have produced a superposition of just two computational basis states, associated with two $n$-bit integers that differ (in the sense of $\oplus$) by $a$. If we knew those two integers their bitwise modulo 2 sum would be $a$. But unfortunately, as we have noted earlier, when a register is in a given quantum state there is in general no way to learn what that state is. To be sure if we could clone the state, then by measuring a mere 10 copies of it in the computational basis we could with a probability of about 0.998 learn both $x_0$ and $x_0 \oplus a$ and therefore $a$ itself. But unfortunately, as we have also noted earlier, one cannot clone an unknown quantum state. Nor does it help to run the algorithm many times, since we are likely to get states of the form (2.35) for different (random) values of $x_0$.

By subjecting (2.35) to a direct measurement all we can learn is either $x_0$ — a random number — or $x_0 \oplus a$ — another random number. The number $a$ that we would like to know appears only in the *relation* between these two random numbers, only one of which we can find out. Nevertheless, as in Deutsch's problem, if we renounce the possibility of learning either individual number (which alone is of no interest), we can get some useful partial information about their relationship (which is potentially of great interest). The trick now is this:

With the input register in the state (2.35), before subjecting it to a measurement we apply the $n$-fold Hadmard transformation $\mathbf{H}^{\otimes n}$. Eq. (2.29) then gives:

$$\mathbf{H}^{\otimes n}\tfrac{1}{\sqrt{2}}\big(|x_0\rangle + |x_0 \oplus a\rangle\big) = \frac{1}{2^{(n+1)/2}} \sum_{y=0}^{2^n-1} \big((-1)^{x_0\cdot y} + (-1)^{(x_0\oplus a)\cdot y}\big)|y\rangle. \tag{2.36}$$

Since $(-1)^{(x_0\oplus a)\cdot y} = (-1)^{x_0\cdot y}(-1)^{a\cdot y}$, the coefficient of $|y\rangle$ in (2.36) is 0 if $a\cdot y = 1$ and $2(-1)^{x_0\cdot y}$ if $a\cdot y = 0$. Therefore (2.36) becomes

$$\frac{1}{2^{(n-1)/2}} \sum_{a\cdot y=0} (-1)^{x_0\cdot y}|y\rangle, \tag{2.37}$$

where the sum is now restricted to those $y$ for which the modulo-2 bitwise inner product $a\cdot y$ is 0 rather than 1. So measuring the input register now gives us (with equal probability) any of the values of $y$ for which $a\cdot y = 0$ — i.e. for which

$$\sum_{i=0}^{n-1} y_i a_i = 0 \pmod 2, \tag{2.38}$$

where $a_i$ and $y_i$ are corresponding bits in the binary expansions of $a$ and $y$.

Note the spectacular improvement on the classical case, where a single invocation of $\mathbf{U}_f$ tells us nothing whatever about $a$. In the quantum case, we learn nothing only if we are unlucky enough to get $y = 0$ (which happens with the very small probability $1/2^{n-1}$). Otherwise we learn a non-zero value of $y$, and therefore a non-trivial subset of the $n$ bits

of $a$ whose modulo-2 sum vanishes. One of those bits is thus entirely determined by the others in the subset, so we have cut the number of possible choices for $a$ in half, from $2^n - 1$ (the $-1$ reflecting the fact that we are told that $a \neq 0$) to $2^{(n-1)/2} - 1$. In one invocation of the subroutine we can, with very high probability, eliminate half of the candidates for $a$!

Furthermore, if we repeat the whole procedure once more, then with very high probability the new value of $y$ that we learn will be neither 0 nor the same as the value we learned the first time. We will therefore learn a new non-trivial relation among the bits of $a$ which enables us to reduce the number of candidates by another factor of 2, removing 3/4 of the possibilities available for $a$ with our two invocations of the subroutine (compared to the classical situation in which only a single value of $a$ is removed).

If every time we repeat the procedure we have a good chance of reducing the number of choices for $a$ by another factor of 2, then perhaps with $n$ invocations of the subroutine we might have a significant chance of learning $a$ precisely. And indeed, one can show that if we repeat the procedure a fixed number of times (for example 20 times) more than $n$, then the odds are high (more than a million to one if we do $n + 20$ runs) that we will learn $a$, no matter how large $n$ may be.

Our description of the quantum computation and what we learn from it is now finished: each time we run the computation we learn a random $y$ satisfying $a \cdot y = 0$. To get a quantitative measure of how many runs we need to have a high probability of learning $a$, there remains some slightly subtle but purely mathematical analysis. If we invoke $\mathbf{U}_f$ $m$ times, we learn $m$ independently selected random numbers $y$ whose bits $y_i$ satisfy (2.38). It is then just a matter of thinking about familiar properties of the (somewhat unfamiliar) space of $n$-dimensional vectors with components restricted to the modulo 2 integers 0 and 1, to establish that a number of such vectors not very much larger than $n$ suffices with high probability to determine $a$ completely. The point is that if we have $n - 1$ relations (2.38) for $n - 1$ *linearly independent* sets of $y_i$, then this gives us enough equations to determine a unique non-zero $a$. "Linearly independent" in this context means linear independence over the integers modulo 2; i.e. no subsets of the $y$'s should satisfy $y \oplus y' \oplus y'' \oplus \cdots = 0 \pmod 2$. We have to invoke the subroutine enough times to give us a high probability of coming up with $n - 1$ linearly independent values of $y$. But regardless of the size of $n$, for not terribly large $x$ the probability becomes extremely close to 1 that a set of $n + x$ random vectors from such an $n - 1$ dimensional subspace contains a linearly independent subset. This is obvious for ordinary vectors with continuous components, since the probability of a randomly selected vector in an $n - 1$ dimensional space lying in a subspace of lower dimensionality is zero — it is certain to have a non-zero component outside of the lower dimensional subspace. The argument is trickier here because components are restricted to only two values: 1 or 0.

Introduce a basis in the full $n - 1$ dimensional subspace of all vectors $y$ with $a \cdot y = 0$, so that a random vector in the subspace can be expressed as a linear combination of the

basis vectors with coefficients that are randomly and independently 1 or 0. Arrange the resulting $(n + x)$ random vectors of 1's and 0's into a matrix of $n + x$ rows and $n - 1$ columns. Since the row rank (the number of linearly independent rows) of a matrix is the same as the column rank, even when arithmetic is confined to the integers modulo 2, the probability that some subset of $n - 1$ of the $n + x$ $(n - 1)$-dimensional row vectors is linearly independent is the same as the probability that all $n-1$ of the $(n+x)$ dimensional column vectors are linearly independent. But it is easy to find a lower bound for this last probability:

Pick a column vector at random. The probability that it is non-zero is $1 - \frac{1}{2^{n+x}}$. Take it as the first member of a basis in which we expand the remaining column vectors. The probability that a second, randomly selected column vector is independent of the first is $1 - \frac{1}{2^{n+x-1}}$, since it will be independent unless every one of its (random) components along the remaining $n + x - 1$ vectors is zero. Continuing in this way, we conclude that the probability $q$ of all $n - 1$ column vectors being linearly independent is

$$q = \Big(1 - \frac{1}{2^{n+x}}\Big)\Big(1 - \frac{1}{2^{n+x-1}}\Big) \cdots \Big(1 - \frac{1}{2^{x+2}}\Big). \tag{2.39}$$

(If you're suspicious of this argument, reassure yourself by checking that it gives the right $q$ when $n = 3$, $a = 111$, and $x = 0$, by explicitly enumerating which of the 64 different sets of three $y$'s, all satisfying $a \cdot y = 0$, contain two linearly independent vectors.)

Finally, to get a convenient lower bound on the size of $q$, note that if we have a set of non-negative numbers $a, b, c, \ldots$ whose sum is less than 1, the product $(1-a)(1-b)(1-c)\cdots$ exceeds $1 - (a + b + c + \cdots)$. (Easily proved by induction on the number of numbers in the set.) The probability $q$ is therefore greater than

$$1 - \frac{1}{2^{x+2}} - \frac{1}{2^{x+3}} - \cdots - \frac{1}{2^{x+n}}, \tag{2.40}$$

and this, in turn, is greater than

$$1 - \frac{1}{2^{x+1}}. \tag{2.41}$$

So if we want to be able to determine $a$ with less than one chance in a million of failure, it is enough to run the subroutine $n + 20$ times.

This intrusion of some mildly arcane arithmetic arguments, to confirm that the output of the quantum computer provides the needed information in the advertised number of runs, is characteristic of many quantum-computational algorithms. The action of the quantum computer itself is rather straightforward, but we must engage in more strenuous mathematical exertions to show that the outcome of the quantum computation does indeed enable us to accomplish the required task.

### E. The importance of cNOT gates

As noted in Chapter I, constraints on what is physically feasible have pretty much limited people to considering unitary transformations that can be built up entirely out of

1-Qbit and 2-Qbit gates — unitary transformations that act only on a single Qbit at a time or on a single pair of Qbits. It is generally assumed that arbitrary 1-Qbit unitary gates are relatively straightforward to construct, though even this can be difficult for many of the systems that have been proposed for Qbits. Two-Qbit gates present an even tougher challenge to the quantum engineer, since they require one to manipulate the dynamical interaction between two physical Qbits. The controlled-NOT (cNOT) gate is the great workhorse of 2-Qbit gates because any arithmetical operation on bits can be carried out on computational-basis states with operations built up out of 1-Qbit gates acting in suitable combination with 2-Qbit cNOT gates. No other 2-Qbit gates are required. This happy state of affairs follows from three facts.

(a) Any arithmetical operation can be built up on a reversible classical computer out of *three*-Cbit controlled-controlled-not gates (ccNOT gates, called *Toffoli gates*).

(b) The quantum ccNOT gate — the linear extension of ccNOT from computational basis states to arbitrary three-Qbit states — can be built up out of cNOT gates $\mathbf{C}$ and appropriate controlled-$U$ gates $\mathbf{cU}$, defined in analogy to the cNOT gate: acting on computational basis states $\mathbf{cU}_{ij}$ does nothing to the target Qbit $j$ if the state of the control Qbit $i$ is $|0\rangle$, but it applies the unitary transformation $\mathbf{U}$ to the target Qbit if the state of the control Qbit is $|1\rangle$.

(c) Controlled-$U$ gates can be built out of controlled-NOT gates and appropriate 1-Qbit unitary gates.

We take up these three points in turn:

(a) The 3-bit Toffoli gate $\mathbf{T}$ is the great workhorse of reversible classical computation. It flips the third (target) bit if and only if *both* of the first two (control bits) are 1:

$$x, y, z \rightarrow x, y, z \oplus xy. \tag{2.42}$$

Its extension to a unitary transformation on Qbits is correspondingly defined in the computational basis by

$$\mathbf{T}|x\rangle|y\rangle|z\rangle = |x\rangle|y\rangle|z \oplus xy\rangle. \tag{2.43}$$

Since $\mathbf{T}$ is its own inverse, it is clearly reversible, so its action on the computational basis is simply to permute those states, and therefore its linear extension to arbitrary Qbit states is unitary (by the argument given in Section D of Chapter 1). The Toffoli gate enables one to calculate the logical AND of two bits (i.e. their product) since $\mathbf{T}|x\rangle|y\rangle|0\rangle = |x\rangle|y\rangle|xy\rangle$. Since all Boolean operations can be built up out of AND and NOT, and since all arithmetic can be built up out of Boolean operations, using Toffoli gates one can build up all of classical computation through reversible operations. (One can even produce NOT with a Toffoli gate: $\mathbf{T}|1\rangle|1\rangle|x\rangle = |1\rangle|1\rangle|\overline{x}\rangle$, but this would be a ridiculous way to implement NOT on a quantum computer, since 3-Qbit gates are much harder to implement than 1-Qbit gates.)

(b) Although there is no way to construct the Toffoli gate out of 1-Cbit and 2-Cbit classical gates, the trick *can* be done with the corresponding quantum gates. The first

thing to note is that there is a quantum gate whose square is the NOT operator $\mathbf{X}$. It is called "the square root of NOT" and we shall write it as $\sqrt{\mathbf{X}}$. Its matrix is just

$$\sqrt{\mathbf{X}} = \frac{1}{1+i}\begin{pmatrix} 1 & i \\ i & 1 \end{pmatrix}. \tag{2.44}$$

One easily verifies that

$$(\sqrt{\mathbf{X}})^2 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \mathbf{X}, \tag{2.45}$$

and that it is indeed unitary,

$$\sqrt{\mathbf{X}}^{\,\dagger}\sqrt{\mathbf{X}} = 1. \tag{2.46}$$

To build the action on computational basis states of the 3-Qbit Toffoli gate out of 1-Qbit and 2-Qbit gates we require a *controlled* square root of NOT gate. More generally, I describe in (c) below how to construct a controlled-$U$ gate for *any* 1-Qbit unitary $\mathbf{U}$. Given such a controlled-$U$ gate $\mathbf{cU}_{ij}$ with Qbit $i$ the control and $j$ the target, a doubly controlled $\mathbf{U}^2$ gate, controlled by Qbits 2 and 1 and targeting Qbit 0, can be constructed out of three controlled-$U$ gates and two cNOT gates:

$$\mathbf{cU}_{10}\mathbf{C}_{21}\mathbf{cU}_{10}^{\dagger}\mathbf{C}_{21}\mathbf{cU}_{20}. \tag{2.47}$$

The corresponding circuit diagram is shown in Figure 2.11. One can verify that, acting on the 3-Qbit computational basis states, (2.47) acts as $\mathbf{1}$ unless Qbits 2 and 1 are both in the state $|1\rangle$, in which case it acts on Qbit 0 as $\mathbf{U}^2$.

An efficient way to see this is first to note that the part in the middle,

$$\mathbf{C}_{21}\mathbf{cU}_{10}^{\dagger}\mathbf{C}_{21} \tag{2.48}$$

leaves the computational-basis states of Qbits 2 and 1 unchanged, while acting on Qbit 0 as $\mathbf{1}$ if those states are both $|0\rangle$ or both $|1\rangle$, and acting on Qbit 0 as $\mathbf{U}^{\dagger}$ if one of the states is $|0\rangle$ and the other, $|1\rangle$. But the $\mathbf{cU}_{10}$ and $\mathbf{cU}_{20}$, between which the gates in (2.48) are sandwiched in (2.47), both act on the the target Qbit 0 as $\mathbf{1}$ when the states of both control Qbits are $|0\rangle$; when the states of the control Qbits are $|0\rangle$ and $|1\rangle$, they provide a single $\mathbf{U}$ on the target Qbit, which combines with the $\mathbf{U}^{\dagger}$ to give $\mathbf{1}$: and they both act on the target as $\mathbf{U}$ when the states of both control Qbits are $|1\rangle$. So the circuit does indeed give a doubly controlled $\mathbf{U}^2$.

(c) Finally, we must show that a controlled-$U$ gate can be built out of controlled-NOT gates and 1-Qbit unitary gates. Let $\mathbf{V}$ and $\mathbf{W}$ be arbitrary 1-Qbit unitary transformations and consider the product (see also Figure 2.12, noting there some of the irritating consequences of the convention that operators on the right act first in equations, but operators on the left act first in circuit diagrams)

$$\mathbf{V}_0\mathbf{C}_{10}\mathbf{V}_0^{\dagger}\mathbf{W}_0\mathbf{C}_{10}\mathbf{W}_0^{\dagger}. \tag{2.49}$$

If the state of the control Qbit 1 is $|0\rangle$ then $\mathbf{C}_{10}$ acts as $\mathbf{1}$ on the target Qbit 0 and the action of the unitary transformations on the target reduces to $(\mathbf{V}\mathbf{V}^\dagger)(\mathbf{W}\mathbf{W}^\dagger) = \mathbf{1}$. But if the state of the control Qbit is $|1\rangle$, then $\mathbf{C}_{10}$ acts as $\mathbf{X}$ on the target Qbit and the action of the unitary transformations on the target becomes $(\mathbf{V}\mathbf{X}\mathbf{V}^\dagger)(\mathbf{W}\mathbf{X}\mathbf{W}^\dagger)$. To see that an arbitrary 1-Qbit unitary transformation can be produced by appropriate choices for $\mathbf{V}$ and $\mathbf{W}$ it is best to think of the operator $\mathbf{X}$ in its equivalent form as the Pauli operator $\mathbf{X} = \sigma_x = \mathbf{x}\cdot\sigma$, so that the action on the target Qbit when the state of the control Qbit is $|1\rangle$ becomes

$$\big(\mathbf{V}(\mathbf{x}\cdot\sigma)\mathbf{V}^\dagger\big)\big(\mathbf{W}(\mathbf{x}\cdot\sigma)\mathbf{W}^\dagger\big). \tag{2.50}$$

As explained in Appendix A2 of Chapter I, it is possible to pick unitary $\mathbf{U}$ so that $\mathbf{U}(\mathbf{x}\cdot\sigma)\mathbf{U}^\dagger = \mathbf{c}\cdot\sigma$ for any unit vector $\mathbf{c}$ by taking $\mathbf{U}$ to be the unitary transformation associated with any rotation that takes $\mathbf{x}$ into $\mathbf{c}$. It is therefore possible to choose $\mathbf{V}$ and $\mathbf{W}$ so that

$$\big(\mathbf{V}(\mathbf{x}\cdot\sigma)\mathbf{V}^\dagger\big)\big(\mathbf{W}(\mathbf{x}\cdot\sigma)\mathbf{W}^\dagger\big) = (\mathbf{a}\cdot\sigma)(\mathbf{b}\cdot\sigma) = \mathbf{a}\cdot\mathbf{b} + i\mathbf{a}\otimes\mathbf{b}\cdot\sigma, \tag{2.51}$$

for any desired unit vectors $\mathbf{a}$ and $\mathbf{b}$. But an arbitrary 1-Qbit unitary transformation has, to within a multiplicative numerical phase factor, the form

$$\mathbf{u}(\mathbf{n},\theta) = e^{i\frac{1}{2}\theta(\mathbf{n}\cdot\sigma)} = \cos\tfrac{1}{2}\theta + i(\mathbf{n}\cdot\sigma)\sin\tfrac{1}{2}\theta. \tag{2.52}$$

Hence by taking $\mathbf{a}$ and $\mathbf{b}$ to be in the plane perpendicular to $\mathbf{n}$ and taking the angle between them to be $\frac{1}{2}\theta$, we can reproduce any unitary transformation of the form (2.52).

If the unitary transformation has an additional phase factor $e^{i\alpha}$, then we need to apply an additional controlled-change-of-phase operation, that acts in the computational basis to take $|x\rangle|y\rangle$ into $|x\rangle\big(e^{i\alpha x}|y\rangle\big)$. But this state is identical to $\big(e^{i\alpha x}|x\rangle\big)|y\rangle$, which can be produced by applying the 1-Qbit unitary transformation $e^{i\alpha\mathbf{n}}$ directly to the control Qbit.

Eight cNOT gates are required to construct a Toffoli gate in this way: the construction of the doubly controlled $\mathbf{U}^2$ gate in (2.47) (illustrated in Figure 2.11) requires two cNOT gates and three controlled-unitary gates, and each controlled-unitary gate, as described in point (c) above (illustrated in Figure 2.12) requires two more cNOT gates. There is a more direct construction that uses only six cNOT gates, illustrated in Figure 2.13. As the figure makes clear, it is quite straightforward to construct a doubly-controlled $i\mathbf{X}$ gate, using only four cNOT gates. To get rid of the extra factor of $i$, however, requires an additional controlled-phase gate, which can be constructed using two more cNOT gates. If quantum computation ever becomes a working technology, it might well be easier to construct controlled-phase gates as fundamental gates in their own right — as basic a piece of 2-Qbit hardware as a cNOT gate.

Two other comments about cNOT gates:

(1) As demonstrated in Figure 2.7, $n$ cNOT gates can act jointly to take the $2n$ Qbit state $|x\rangle_n|0\rangle_n$ into $|x\rangle_n|x\rangle_n$. But although this clones computational basis states, it

does not violate the no-cloning theorem, since it fails to clone nontrivial *superpositions* of computational basis states.

(2) There is a way in which cNOT gates resemble measurements: the transformation of $|x\rangle_n|0\rangle_n$ into $|x\rangle_n|x\rangle_n$ is reminiscent of the way in which applying a measurement gate to $n$ Qbits in the state $|x\rangle_n$ results in the number $x_n$ appearing on the display of the measurement gate, while the original $n$ Qbits remain in the state $|x\rangle_n$. But the analogy between the target Qbits and the display of the measurement gate breaks down when we consider superpositions. If we want to measure $n$ Qbits in the state $a|x\rangle_n + b|y\rangle_n$ a measurement gate will display $x$ with probability $|a|^2$ and $y$ with probability $|b|^2$, leaving the $n$ Qbits in the state $|x\rangle_n$ in the first case and $|y\rangle_n$ in the second. In contrast, the $n$ controlled-NOT gates take $\big(a|x\rangle_n + b|y\rangle_n\big)|0\rangle_n$ into

$$a|x\rangle_n|x\rangle_n + b|y\rangle_n|y\rangle_n. \tag{2.53}$$

The $n$ control Qbits and the $n$ target Qbits are now in an entangled state. Neither set of Qbits has any state of its own. In particular the control Qbits are neither in the state $|x\rangle_n$ nor $|y\rangle_n$ as they would be after a measurement. Nor are the target Qbits associated with either of these two states. Indeed, in contrast to a measurement gate, the action of the $n$ cNOT gates can be reversed by simply applying them a second time to the $2n$ Qbits.

If, however, one subjects the *target* Qbits to a measurement, then, according to the generalized Born rule, the joint state (2.53) of control and target Qbits, reduces to $|x\rangle_n|x\rangle_n$ with probability $|a|^2$ or to $|y\rangle_n|y\rangle_n$ with probability $|b|^2$, and we are in a situation in which the control and target Qbits behaving *exactly* like measured system and measuring device. What the cNOT gates can do is to shift the measurement process from applying the measurement gate to the control Qbits, to applying it to the target Qbits.

## Appendix to Chapter 2

This more technical appendix is addressed to physicists, curious about how one might, at least in principle, construct a cNOT gate, exploiting physically plausible interactions between two Qbits. Those with no background in quantum physics are invited to inspect it, but will find many parts quite obscure. It is of relevance only to those interested in quantum computational hardware, and plays no role in subsequent developments.

As noted in Chapter 1 (and as easily confirmed directly) the controlled-NOT gate $\mathbf{cX}_{12}$ with control Qbit 1 and target Qbit 2 can be written as

$$\mathbf{cX}_{12} = \tfrac{1}{2}\big(\mathbf{1} + \mathbf{Z}_1 + \mathbf{X}_2 - \mathbf{Z}_1\mathbf{X}_2\big). \tag{2.54}$$

Since $\mathbf{Z} = \mathbf{HXH}$ and $\mathbf{1} = \mathbf{H}^2$, this can also be written as

$$\mathbf{cX}_{12} = \mathbf{H}_2\mathbf{cZ}_{12}\mathbf{H}_2, \tag{2.55}$$

where the more symmetric controlled-Z operation is given by[1]

$$\mathbf{cZ} = \tfrac{1}{2}\big(\mathbf{1} + \mathbf{Z}_1 + \mathbf{Z}_2 - \mathbf{Z}_1\mathbf{Z}_2\big). \tag{2.56}$$

So to within one-Qbit Hadamard transformations, the problem of constructing a **cX** gate is the same as that of constructing a **cZ** gate.

Since $(\mathbf{cZ})^2 = \mathbf{1}$, **cZ** satisfies the identity[2]

$$\exp(i\mathbf{cZ}\theta) = \cos\theta + i\mathbf{cZ}\sin\theta. \tag{2.57}$$

We can therefore rewrite (2.56) as the less straightforward operator identity:

$$\begin{aligned}\mathbf{cZ} &= -i\exp(i\tfrac{\pi}{2}\mathbf{cZ}) = -i\exp\big(i(\tfrac{\pi}{4})(\mathbf{1} + \mathbf{Z}_1 + \mathbf{Z}_2 - \mathbf{Z}_1\mathbf{Z}_2)\big)\\ &= e^{-i(\pi/4)}\exp\big(i(\tfrac{\pi}{4})(\mathbf{Z}_1 + \mathbf{Z}_2 - \mathbf{Z}_1\mathbf{Z}_2)\big).\end{aligned} \tag{2.58}$$

The point of writing **cZ** in this clumsy way is that the unitary transformations one can construct physically are those of the form

$$\mathbf{U} = \exp(i\mathcal{H}t), \tag{2.59}$$

where $\hbar\mathcal{H}$ is the Hamiltonian that describes the external fields acting on the Qbits, and the interactions between Qbits. So to within an overall constant (and therefore irrelevant) phase factor we can realize a **cZ** gate by letting the two Qbits interact through a Hamiltonian proportional to $\mathbf{Z}_1 + \mathbf{Z}_2 - \mathbf{Z}_1\mathbf{Z}_2$ for a precisely specified interval of time. If each Qbit is a spin-$\frac{1}{2}$, then (since $\mathbf{Z} = \sigma_z$) this Hamiltonian describes two such spins with a highly anisotropic interaction that couples only their $z$-components (*Ising interaction*) subject to a uniform magnetic field with a magnitude appropriately proportional to the strength of their coupling. This is perhaps the simplest example of how to make a cNOT gate.

Ising interactions, however, are rather hard to arrange. A much more natural interaction between two spins is the *exchange interaction*

$$\sigma^{(1)}\cdot\sigma^{(2)} = \sigma_x^{(1)}\sigma_x^{(2)} + \sigma_y^{(1)}\sigma_y^{(2)} + \sigma_z^{(1)}\sigma_z^{(2)} = \mathbf{X}_1\mathbf{X}_2 + \mathbf{X}_1\mathbf{Z}_1\mathbf{Z}_2\mathbf{X}_2 + \mathbf{Z}_1\mathbf{Z}_2, \tag{2.60}$$

which is invariant under spatial rotations. (See Appendix A2 of Chapter 1.)

One can also build a **cZ** gate out of two spins interacting through (2.60), provided one applies magnetic fields to each spin that are along the same direction but have different magnitudes and signs.[3]

---

[1] Because of its symmetry we may write **cZ** without the subscripts distinguishing control and target Qbits.

[2] Both sides have identical expansions as power series in $\theta$.

[3] What follows was inspired by G. Burkard, D. Loss, D. P. DiVincenzo, and J. A. Smolin, `http://arxiv.org/abs/cond-mat/9905230`.

What we must show is that to within an overall constant phase factor it is possible to express **cZ** in the form

$$\mathbf{cZ} = \exp(i\mathcal{H}t), \tag{2.61}$$

with a Hamiltonian $\mathcal{H}$ of the form

$$\mathcal{H} = \; J\,\sigma^{(1)}\cdot\sigma^{(2)} + H_1\,\sigma_z^{(1)} + H_2\,\sigma_z^{(2)}, \tag{2.62}$$

for appropriate choices of $J$ (known as the exchange coupling), of $H_1$ and $H_2$ (proportional to the magnetic fields acting on the two spins — hereafter we ignore the proportionality constant and refer to them simply as the "magnetic fields"), and of the time $t$ during which the spins interact with each other and with the magnetic fields.

To see that the parameters in (2.62) can indeed be chosen so that $\mathcal{H}$ gives rise to **cZ** through (2.61), recall first[4] that the operator $\frac{1}{2}(1+\sigma^{(1)}\cdot\sigma^{(2)})$ acts as the swap operator on any two-Qbit computational-basis state:

$$\tfrac{1}{2}(1+\sigma^{(1)}\cdot\sigma^{(2)})|xy\rangle = |yx\rangle. \tag{2.63}$$

It follows from (2.63) that the three states (called *triplet states*)

$$|11\rangle, \quad |00\rangle, \quad \tfrac{1}{\sqrt{2}}\big(|01\rangle + |10\rangle\big) \tag{2.64}$$

are eigenstates of $\sigma^{(1)}\cdot\sigma^{(2)}$ with eigenvalue 1, while the state

$$\tfrac{1}{\sqrt{2}}\big(|01\rangle - |10\rangle\big) \tag{2.65}$$

(called the the *singlet state*) is an eigenstate of $\sigma^{(1)}\cdot\sigma^{(2)}$ with eigenvalue $-3$.[5]

The four states (2.64) and (2.65) are also eigenstates of $\frac{1}{2}(\sigma_z^{(1)}+\sigma_z^{(2)})$, the three triplet states (in the order in which they appear in (2.64)) having eigenvalues $-1$, 1, and 0, and the singlet state having eigenvalue 0.

Note also that the first two triplet states in (2.64) are eigenstates of $\frac{1}{2}(\sigma_z^{(1)}-\sigma_z^{(2)})$ with eigenvalue 0, while $\frac{1}{2}(\sigma_z^{(1)}-\sigma_z^{(2)})$ takes the third of the triplet states into the singlet state, and vice-versa.

So the eigenstates of the Hamiltonian

$$\begin{aligned}\mathcal{H} &= J\,\sigma^{(1)}\cdot\sigma^{(2)} + H_1\sigma_z^{(1)} + H_2\sigma_z^{(2)} \\ &= J\,\sigma^{(1)}\cdot\sigma^{(2)} + H_+\tfrac{1}{2}(\sigma_z^{(1)}+\sigma_z^{(2)}) + H_-\tfrac{1}{2}(\sigma_z^{(1)}-\sigma_z^{(2)})\,,\end{aligned} \tag{2.66}$$

[4] This was established in Chapter 1, Section B. It is why the interaction is called the exchange interaction.

[5] If $|0\rangle$ is the state $|\uparrow\rangle$ of spin-up along $\mathbf{z}$, and $|1\rangle$ is $|\downarrow\rangle$, then the singlet state is the state of zero total angular momentum and the three triplet states are the states of angular momentum 1 with $z$-components $-\hbar$, 0, and $\hbar$.

where

$$H_\pm = H_1 \pm H_2, \tag{2.67}$$

can be taken to be the first of the two triplet states (2.64) and two appropriately chosen orthogonal linear combinations of the third triplet state and the singlet state (2.65). The eigenvalues of $\mathcal{H}$ associated with the first and second triplet states are $J - H_+$ and $J + H_+$ ; those associated with the last two states are the eigenvalues of the matrix

$$\begin{pmatrix} J & H_- \\ H_- & -3J \end{pmatrix}$$

of $\mathcal{H}$ in the space spanned by the last two; i.e. $-J \pm \sqrt{4J^2 + H_-^2}$.

Now the four states (2.64) and (2.65) are also eigenstates of **cZ**, the first of the three triplet states having eigenvalue $-1$ and the other three having eigenvalue 1. Consequently these eigenstates of $\mathcal{H}$ are also eigenstates of **cZ** with respective eigenvalues $-1$, 1, 1, and 1. We will therefore produce **cZ** (to within a constant phase factor) if we can chose the exchange coupling $J$, the magnetic fields $H_1$ and $H_2$, and the time $t$ during which $\mathcal{H}$ acts to satisfy:

$$-e^{it(J-H_+)} = e^{it(J+H_+)} = e^{it(-J+\sqrt{4J^2+H_-^2})} = e^{it(-J-\sqrt{4J^2+H_-^2})}. \tag{2.68}$$

The last equality is equivalent to

$$e^{2it\sqrt{4J^2+H_-^2}} = 1, \quad \text{or} \quad e^{it\sqrt{4J^2+H_-^2}} = \pm 1; \tag{2.69}$$

the first is equivalent to

$$e^{2itH_+} = -1, \quad \text{or} \quad e^{itH_+} = \pm i; \tag{2.70}$$

and the second is equivalent to

$$e^{-2itJ} = e^{itH_+} e^{-it\sqrt{4J^2+H_-^2}}. \tag{2.71}$$

The identities (2.69) and (2.70) require the right side of (2.71) to be $\pm i$. For the (positive) time $t$ for which the gate acts to be as small as possible we should choose $-i$, which gives

$$Jt = \pi/4. \tag{2.72}$$

With this value of $t$ we can satisfy (2.69) (with the minus sign) and (2.70) (with the plus sign) by taking $\sqrt{4J^2 + H_-^2} = 4J$ and $H_+ = 2J$. So we can produce the gate **cZ** (to within an overall constant phase factor) by taking the magnetic fields in the Hamiltonian (2.66) and the time for which it acts to be related to the exchange coupling by

$$H_+ = 2J, \quad H_- = 2\sqrt{3}J, \quad t = \tfrac{1}{4}\pi/J, \tag{2.73}$$

or, in terms of the fields on each spin,

$$H_1 = (1+\sqrt{3})J, \quad H_2 = (1-\sqrt{3})J, \quad t = \tfrac{1}{4}\pi/J. \tag{2.74}$$

Note the curious fact that although, as (2.56) makes explicit, the gate **cZ** acts symmetrically on the two spins, the realization of **cZ** by the unitary transformation $e^{i\mathcal{H}t}$ requires the fields acting on the spins to break that symmetry. Of course the symmetry survives in the fact that the alternative choice of fields $H_1 = (1-\sqrt{3})J, \quad H_2 = (1+\sqrt{3})J$ works just as well.

# Figure 2.1

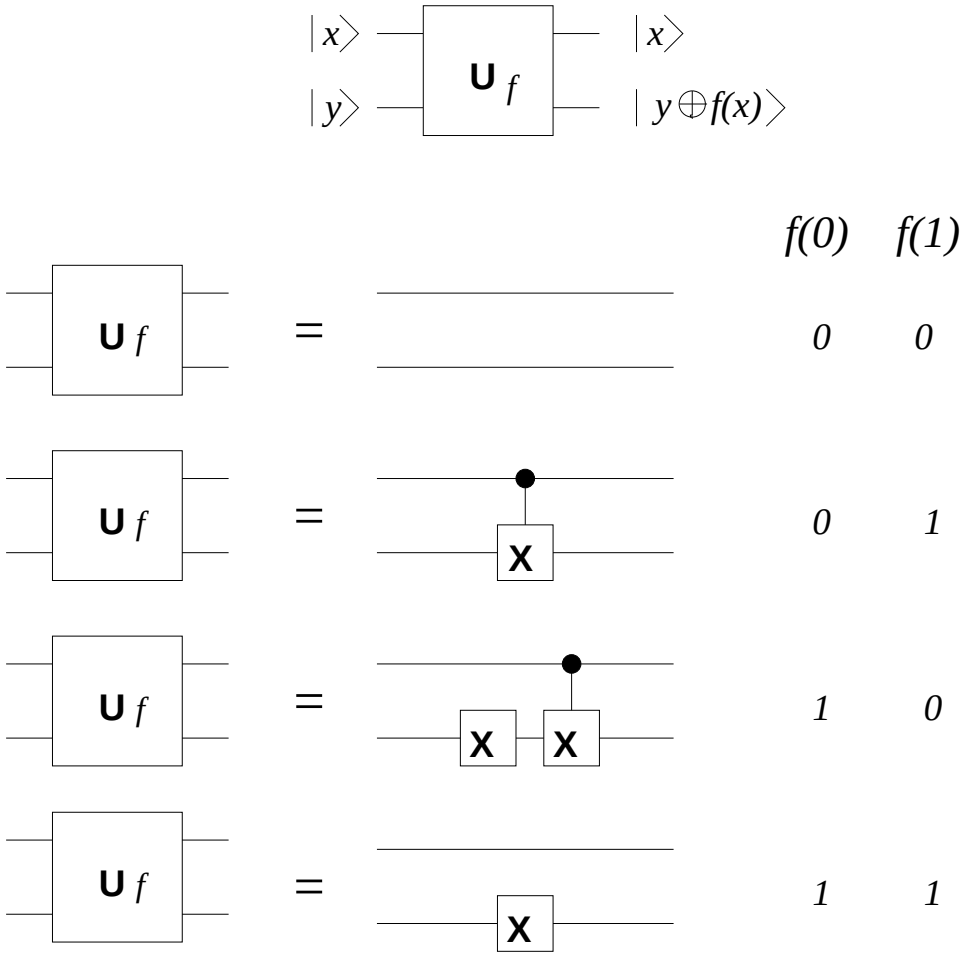


Figure 2.1. A way to construct, with elementary gates, each of the black boxes $\mathbf{U}_f$ that realize the four possible functions $f$ that appear in Deutsch's problem. In case 00 $f$ is identically 0 and it is evident from the general form at the top of the figure that $\mathbf{U}_f$ acts as the identity. In case 01 $f$ acts as the identity so $\mathbf{U}_f$ acts as cNOT with the input register as the control bit. In case 10 $f$ interchanges 0 and 1, so $\mathbf{U}_f$ flips the target bit if and only if the control bit is 0. This is equivalent to combining a cNOT with an unconditional flip of the target bit. And in case 11 $f$ is identically 1, and the effect of $\mathbf{U}_f$ is just to flip the output register, independent of the input register. Note the diagramatic convention for controlled operations: the control Qbit is represented by the wire with the black dot on it; the target Qbit is connected to the control by a vertical line ending in a box containing the controlled operation. An alternative representation for cNOT appears in Figure 2.7.

# Figure 2.2

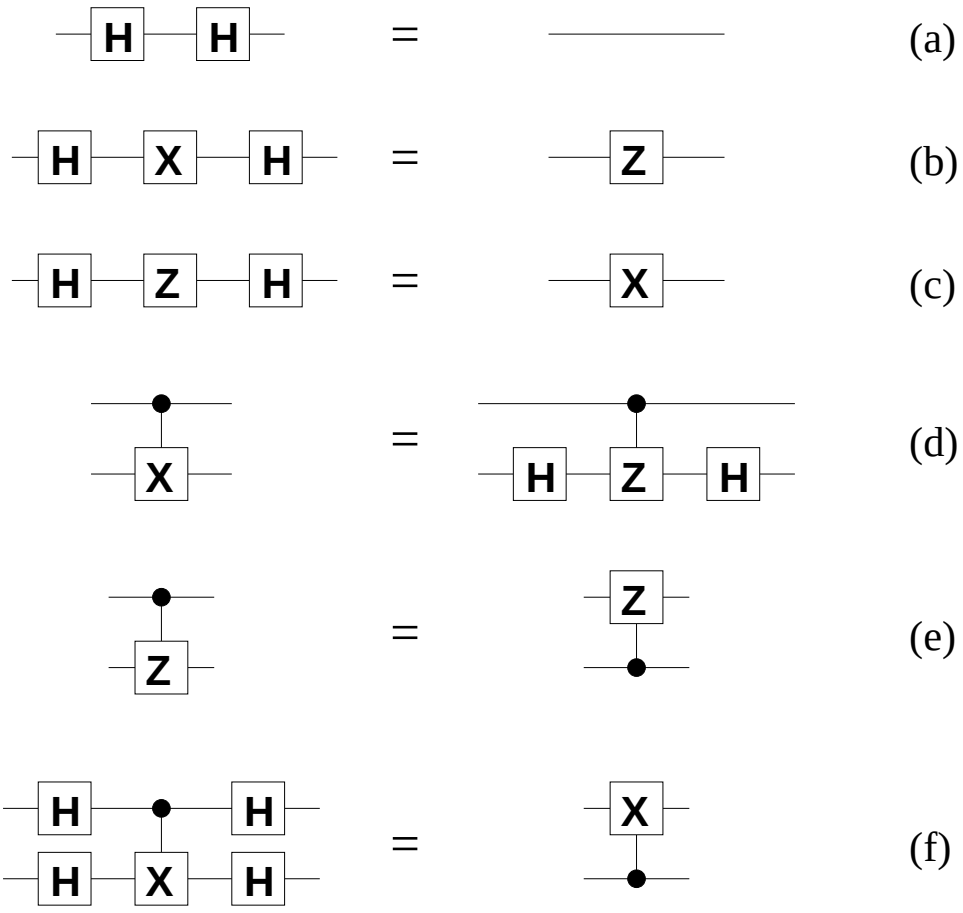


Figure 2.2. Some elementary circuit identities that are useful in seeing why it is possible to determine whether or not $f(0) = f(1)$ with a single invocation of $\mathbf{U}_f$. (a) $\mathbf{H}^2 = \mathbf{1}$. (b) $\mathbf{HXH} = \mathbf{Z}$. (c) A consequence of (a) and (b). (d) A consequence of (a) and (c). (e) The action of the controlled-$Z$ gate does not depend on which Qbit is control and which is target, since it acts as the identity on each of the state $|\,00\rangle, |\,01\rangle, |\,10\rangle$ and multiplies the state $|\,11\rangle$ by $-1$. (f) This follows from (d), (a), and (e).

# Figure 2.3

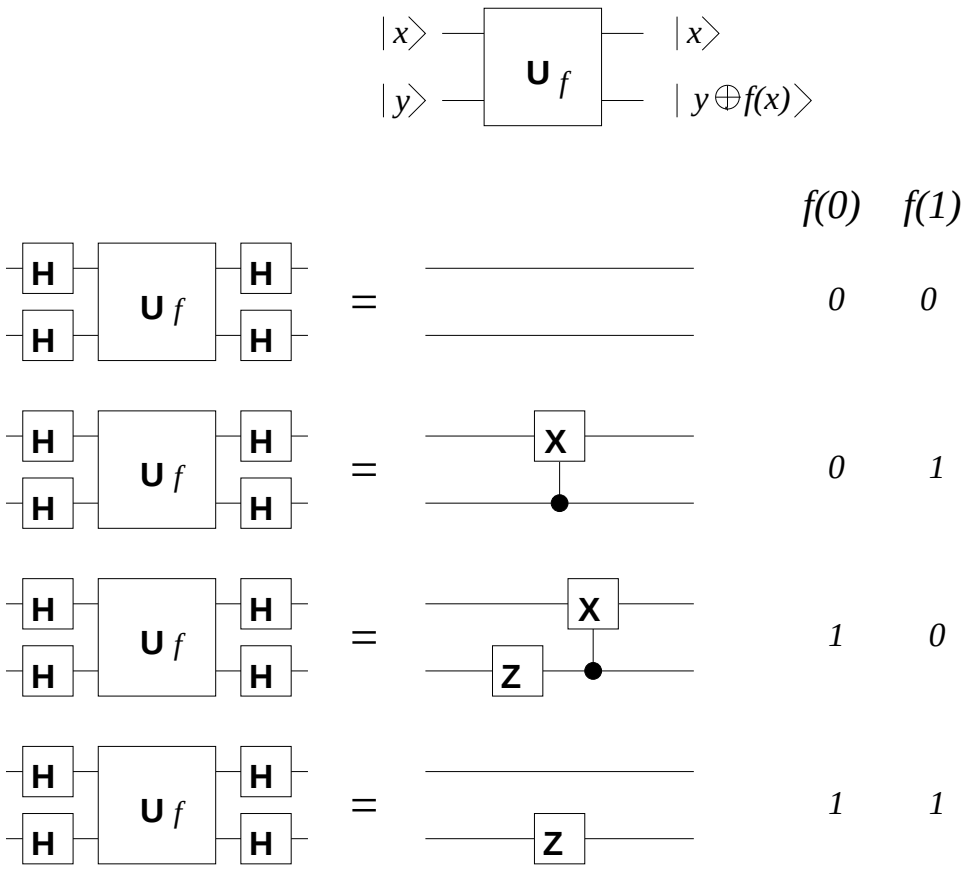


Figure 2.3. We can get the action of $\mathbf{U}_f$, when it is preceded and followed by Hadamards on both Qbits, by applying the appropriate identities of Figure 2.2 to the diagrams of Figure 2.1. Case 00 is unchanged because of Figure 2(a). In case 01 the target and control Qbits of the cNOT are interchanged because of Figure 2.2(f). The form in case 10 follows from the corresponding form in Figure 1 because of Figure 2.2(f) and 2.2(b). And the form in case 11 follows from Figure 2.2(a) and 2.2(b). If the initial state of the output register (lower wire) is $|1\rangle$ and the initial state of the input register is either of the two computational-basis states, then the initial state of the input register will be unchanged in cases 00 and 11, and flipped in cases 01 and 10, so by measuring the input register after the action of $(\mathbf{H} \otimes \mathbf{H})\mathbf{U}_f(\mathbf{H} \otimes \mathbf{H})$ one can determine whether or not $f(0) = f(1)$.

# Figure 2.4

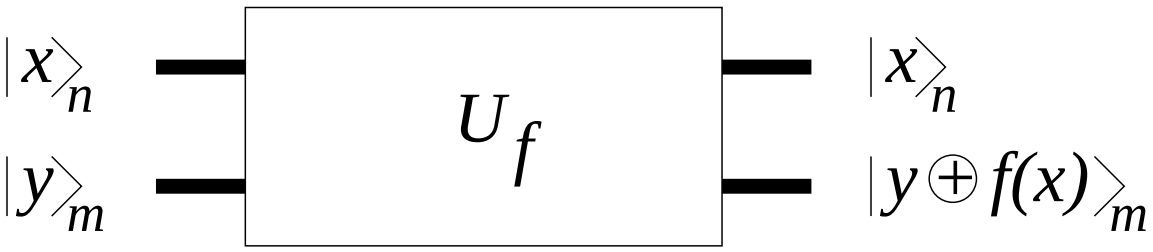


Figure 2.4. Schematic representation of the standard unitary transformation $\mathbf{U}_f$ for evaluating a function $f$ taking a number $0 \leq x < 2^n$ into a number $0 \leq f(x) < 2^m$. The heavy horizontal lines represent multiple Qbit inputs. In order for the computation to be reversible even when $f$ is not one-to-one, two multi-Qbit registers are used. After the computation is done the state of the input register again represents $x$ and the state of the output register represents $y \oplus f(x)$, where $\oplus$ is the operation of bitwise modulo 2 addition (without carry). If the output register initially represents 0, after the computation it represents $f(x)$.

# Figure 2.5

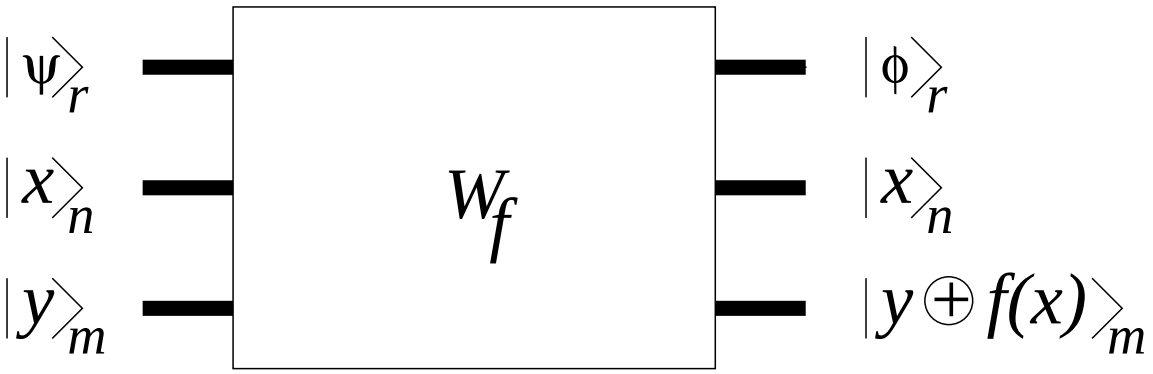


Figure 2.5. A more realistic picture of the computation represented in Figure 2.4. Many additional Qbits may be needed to carry out the calculation. These are represented by an $r$-Qbit bar in addition to the $n$- and $m$-Qbit bars representing the input and output registers. The computation is actually executed by a unitary transformation $\mathbf{W}_f$ that acts on the larger space of all $n+m+r$ Qbits. The representation of Figure 2.4 is correct only if the action of this larger unitary transformation $\mathbf{W}_f$ on the input and output registers alone can be represented by a unitary transformation $\mathbf{U}_f$. As discussed in the text, this will be true, for arbitrary initial superpositions of computational-basis states in the input and output registers, provided the action of $\mathbf{W}_f$ on the residual $r$ Qbits is to take them from an initial pure state $|\psi\rangle_r$ to a final pure state $|\phi\rangle_r$ that is independent of the initial contents of the input and output registers, as depicted in the figure.

# Figure 2.6

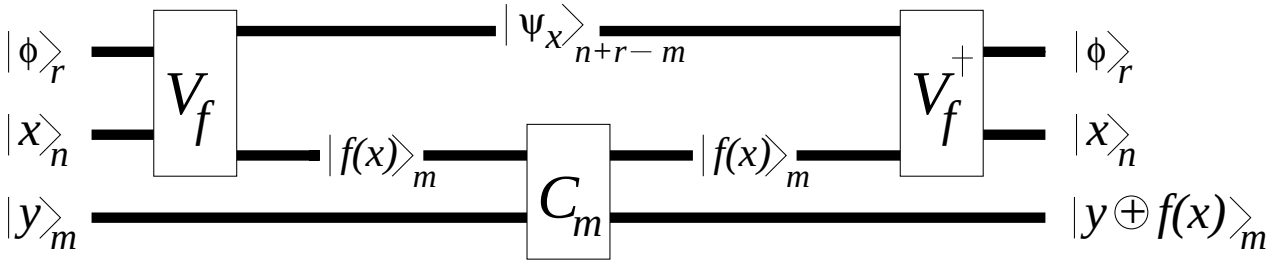


Figure 2.6. A more detailed view of the structure of the unitary transformation $\mathbf{W}_f$ of Figure 2.5. Algebraically, $\mathbf{W}_f = \mathbf{V}_f^\dagger \mathbf{C}_m \mathbf{V}_f$. First a unitary transformation $\mathbf{V}_f$ acts only on the input register and residual Qbits (i.e. it acts as the identity on the output register), transforming them into a state that contains among its $n+r$ Qbits an $m$-Qbit subset that represents the result of the calculation, $f(x)$. At this stage the remaining $n+r-m$ Qbits are in an entangled state. Next an $m$ controlled-NOT transformations (described in more detail in Figure 2.7) act only on the $m$ Qbits representng $f(x)$ and the $m$ Qbits of the output register, leaving the former $m$ unchanged but changing the number represented by the latter $m$ from $y$ to $y \oplus f(x)$. Finally, the inverse $\mathbf{V}_f^\dagger$ of $\mathbf{V}_f$ is applied to the residual Qbits and the input register to restore them to their (unentangled) initial states.

# Figure 2.7

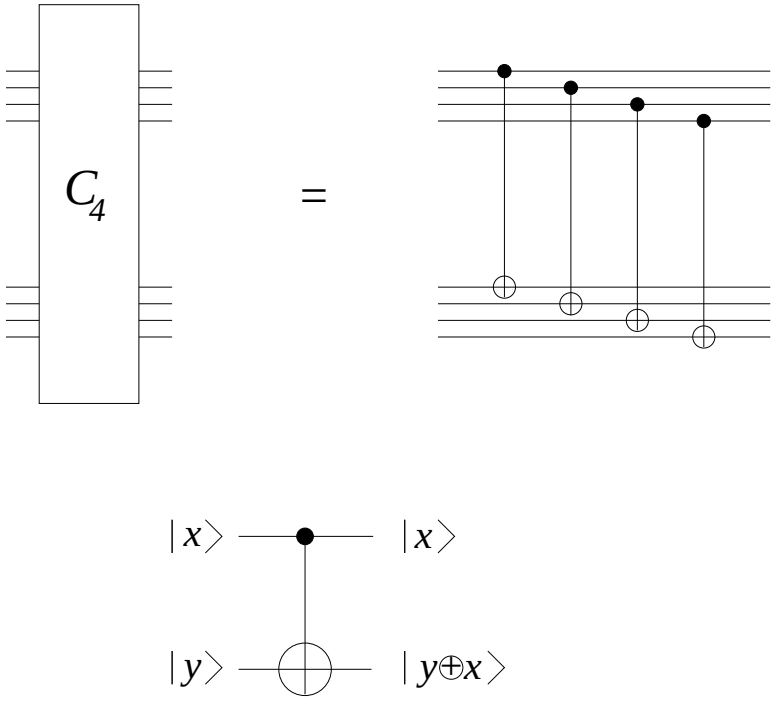


Figure 2.7. Not a simple Gregorian chant, but a more detailed picture of the $\mathbf{C}_m$ unitary transformation in Figure 2.6, for the case $m = 4$. Each of the input and output bars contains 4 Qbits, represented by sets of 4 thin lines (wires). Four different 2-Qbit controlled-not gates link the 4 upper wires representing $f(x)$ to the four lower wires representing the corresponding positions in the output register. The action of a single such cNOT gate is shown in the lower part of the figure. Note the alternative convention for a cNOT gate: the black dot on the wire representing the control Qbit is connected by a vertical line to an open circle on the wire representing the target Qbit. The alternative convention (used above in Figures 2.1-2.3) replaces the open circle by a square box containing the NOT operator $\mathbf{X}$ that may act on the target Qbit. The advantage of the circle representation is that it suggests the symbol $\oplus$ that represents the XOR operation, and that it is easier to draw quickly on a blackboard. The advantage of using $\mathbf{X}$ is that it makes the algebraic relations more evident when NOT operations $\mathbf{X}$, $\mathbf{Z}$ operations, or controlled-Z operations also appear, and that it follws the same form as is used for all other controlled unitaries.

# Figure 2.8

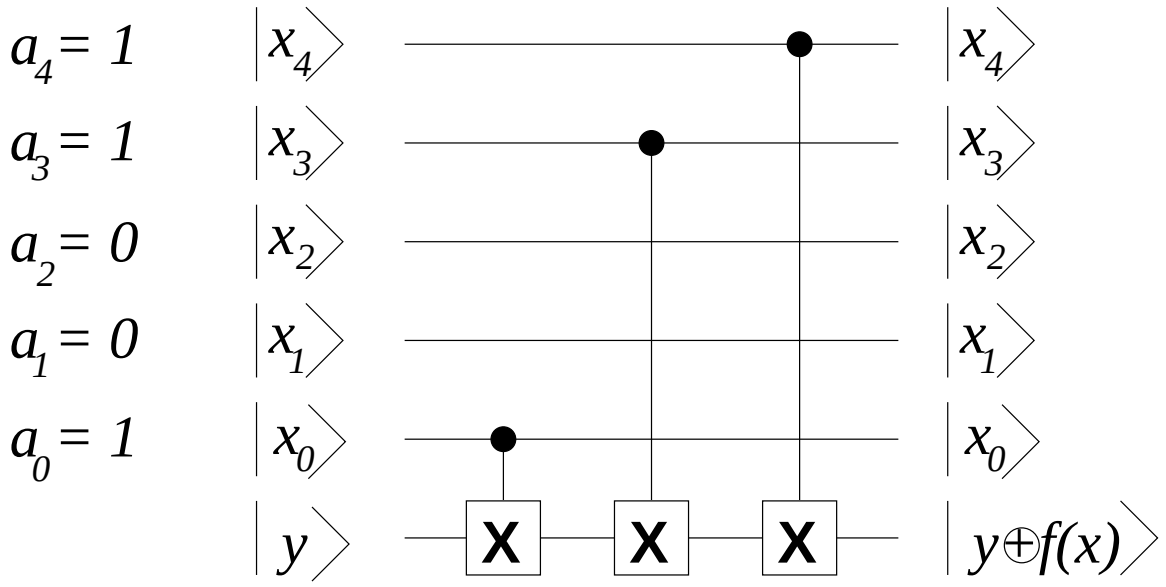


Figure 2.8. An illustration of a circuit that implements the unitary subroutine $\mathbf{U}_f$ taking $n$-Qbit input and 1-Qbit output registers, initially in the state $|\,x\rangle_n|\,y\rangle_1$, into $|\,x\rangle_n|\,y \oplus f(x)\rangle_1$, where $f(x) = a \cdot x = \sum_{j=0}^{n-1} a_j x_j \pmod 2$. The Bernstein-Vazarini problem is to determine all the bits of $a$ with a single invocation of the subroutine. In the illustration $n = 5$ and $a = 25 = 11001$. For $j = 0 \ldots n-1$ each of the cNOT gates adds 1 (mod 2) to the output register if and only if $a_j x_j = 1$. In addition to their normal labeling with the one-Qbit states they represent, the wires of the input register are labeled with the bits of $a$, to make it clear which (those associated with $a_j = 1$) act as control bits for a cNOT targeted on the output register.

# Figure 2.9

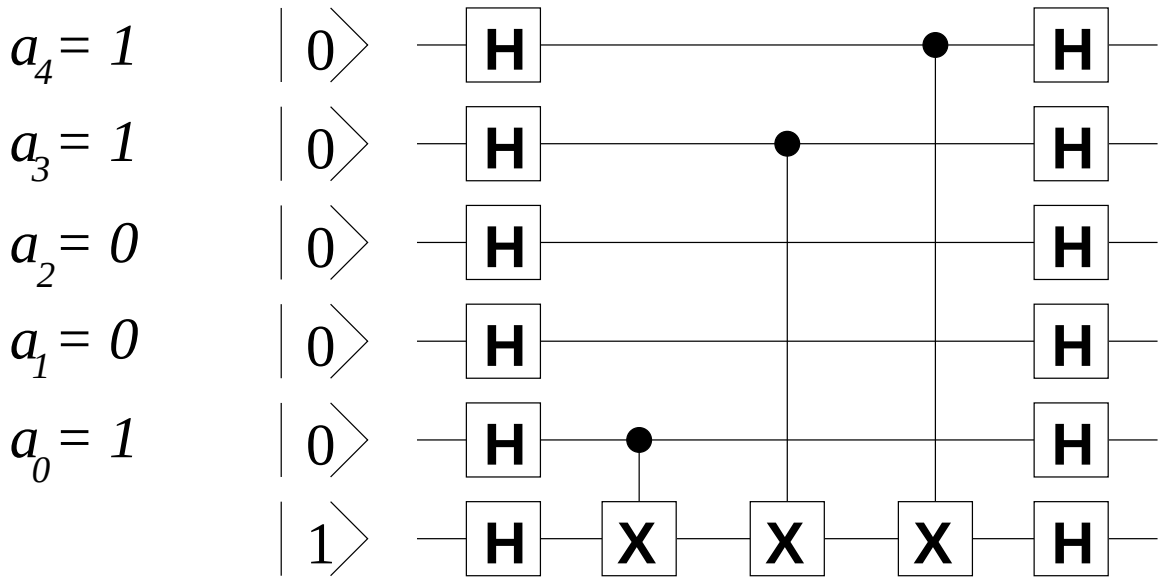


Figure 2.9. The solution to the Berstein-Vazirani problem is to start with the input register in the state $|0\rangle_n$ and the output register in the state $|1\rangle_1$ and apply Hadamard transforms to all $n+1$ registers before applying $U_f$. Another $n+1$ Hadamards are applied after $U_f$ acts. The cNOT gates reproduce the action of $\mathbf{U}_f$, as shown in Figure 2.10. The conventional analysis (as described above) deduces the final state by exploring the effect of the Hadamards on the initial state of the Qbits and on the state subsequently produced by the action of $\mathbf{U}_f$. A much easier way to understand what is going on is to examine the effect of the Hadamards on the collection of cNOT gates equivalent to $\mathbf{U}_f$. This is shown in Figure 2.10.

# Figure 2.10

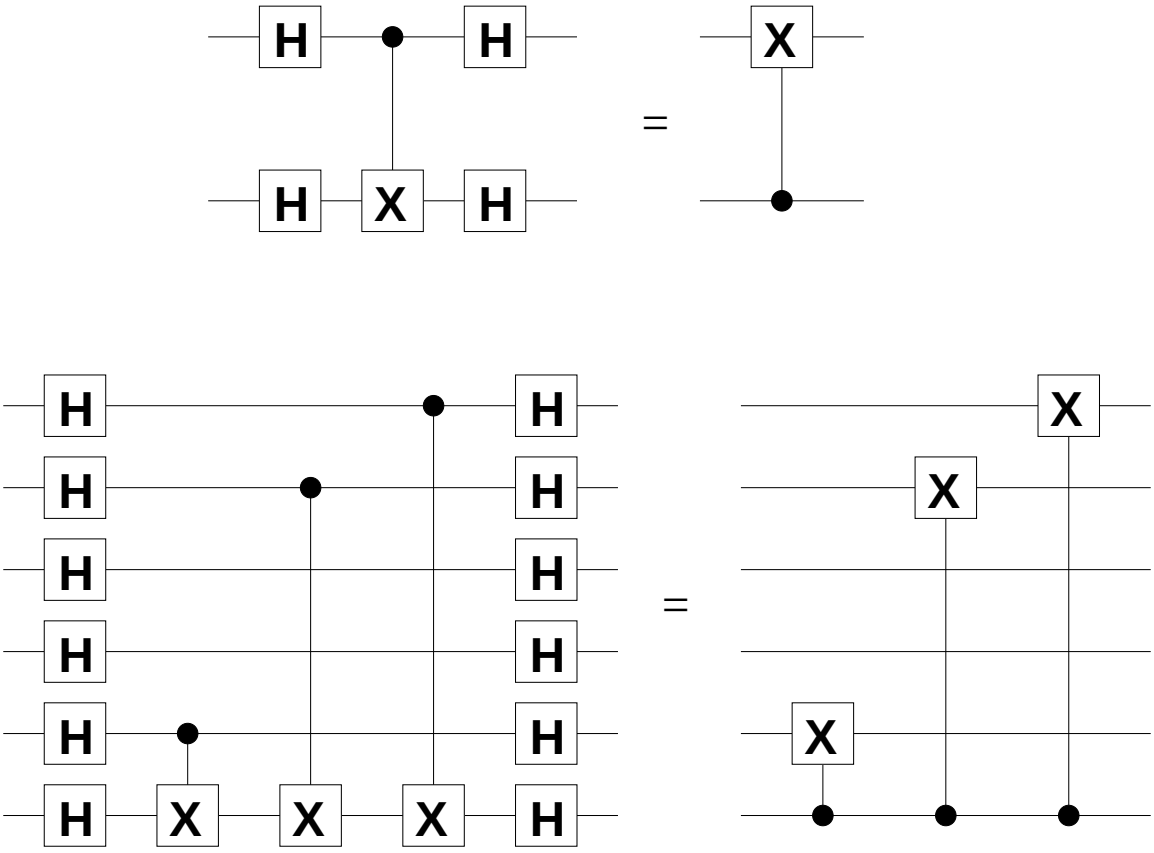


Figure 2.10. Sandwiching a cNOT gate between Hadamards that act on both control and target Qbits, has the effect of interchanging control and target, as shown at the top of the figure. (See also Chapter 1, Eq. (1.40) or Figure 2.2.) Consequently the action of all the Hadamards in Figure 2.9 on the cNOT gates between them is simply to interchange the control and target Qbits, as shown in the lower part of the figure. (One uses the fact that $\mathbf{H}^2 = 1$, so that (a) the **H** gates on wires that are not control bits have no action at all and (b) pairs of Hadamards can be introduced between every **X** on the lowest wire, converting **HXXXH** into $(\mathbf{HXH})(\mathbf{HXH})(\mathbf{HXH})$.) Because the cNOT gates are now controlled by the output register, if the output register is in the state $|\,1\rangle$, as it is in Figure 2.9, then all the **X** act on their input-register targets. If the initial state of the input register is $|\,0\rangle_n$, as it is in Figure 11, the effect of each **X** is to convert to $|\,1\rangle$ the state of each Qbit associated with a bit of $a$ that is 1. This converts the state of the input register to $|\,a\rangle_n$.

# Figure 2.11

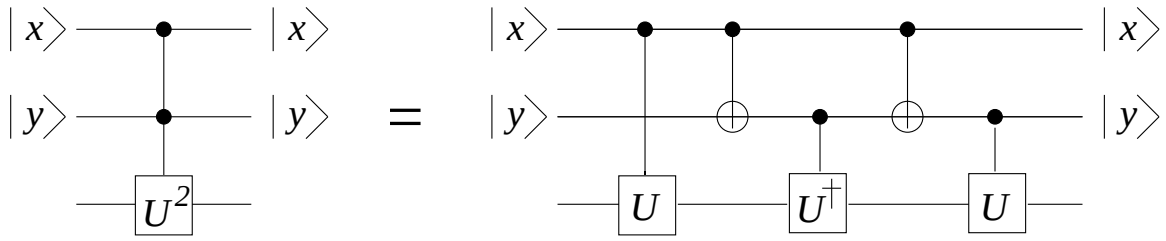


Figure 2.11. How to construct a three-Qbit controlled-controlled-$U^2$ gate from two-Qbit controlled-NOT, controlled-$U$, and controlled-$U^\dagger$ gates. The values $x$ and $y$ of the two control bits are indicated to the right and left of the wires that carry them.[6] Note that both are unchanged by this sequence of operations, since the one labeled $y$ is either flipped twice or not at all. To see that the circuit on the right acts as claimed, note first that its three central gates have the effect of applying $\mathbf{U}^\dagger$ to the bottom wire if and only if $x$ and $y$ are different. If $x$ and $y$ are different then just one of the two outermost controlled-$U$ gates acts on the bottom wire, which combines with the $\mathbf{U}^\dagger$ to give $\mathbf{1}$. If $x$ and $y$ are both 0, neither of the controlled-$U$ gates act on the bottom wire, so since $\mathbf{U}^\dagger$ also does not act, the total action on the bottom wire is $\mathbf{1}$. But if both $x$ and $y$ are 1, then both controlled-$U$ gates act on the bottom wire, and since $\mathbf{U}^\dagger$ does not act, the total action is $\mathbf{U}^2$.

[6] In keeping with the informal practice of talking about values of Qbits (or even bits) rather than the computational-basis state $|\,x\rangle$ (with $x = 0$ or 1) of the Qbit, one sometimes omits the state symbols $|\ \rangle$ from the input and output of a circuit diagram, specifying only the values of $x$, $y$, ....

# Figure 2.12

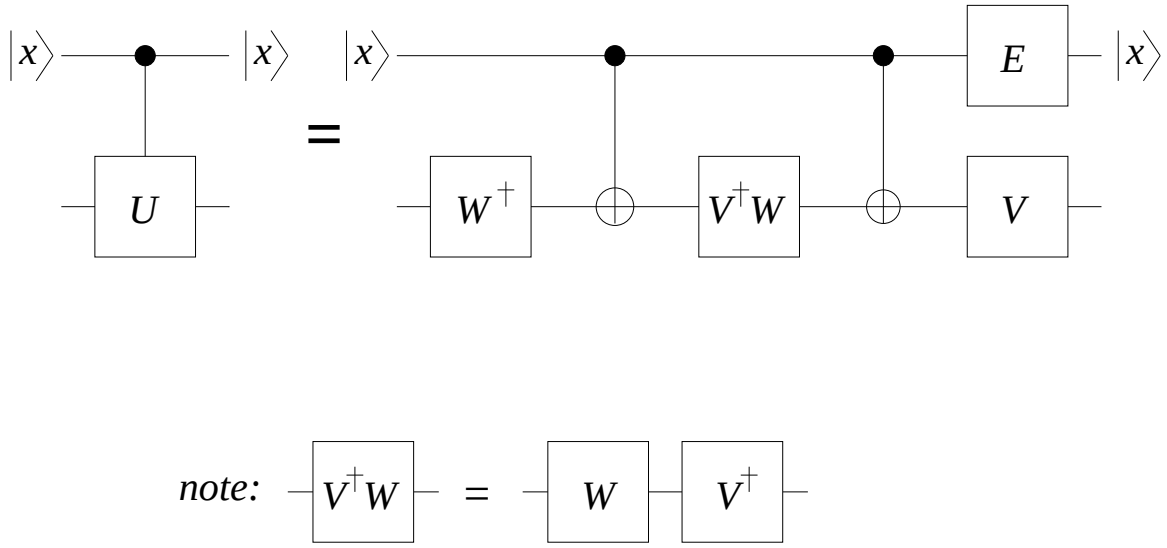


Figure 2.12. How to construct a controlled-$U$ gate $\mathbf{C}_U$ from the unitary one-Qbit gates and two controlled-NOT gates. If the control-Qbit is in the state $|\,0\rangle$, the operations on the target wire combine to give the identity $\mathbf{1}$. But if the control Qbit is in the state $|\,1\rangle$ then the operations combine to give $\mathbf{U} = \left(\mathbf{V}\sigma_x\mathbf{V}^\dagger\right)\left(\mathbf{W}\sigma_x\mathbf{W}^\dagger\right)$, where $\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \mathbf{X}$ is the flip operator. As noted in Section D above, a general two dimensional unitary transformation can always be put in this form for appropriate $\mathbf{V}$ and $\mathbf{W}$, to within an overall constant phase factor. The $\mathbf{E}$ on the control wire is the unitary transformation $\mathbf{E} = e^{i\mathbf{n}\alpha} = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\alpha} \end{pmatrix}$, which supplies such a phase factor when the control bit is 1. Note the irritating left-right interchange between the order of symbols in the formula for $\mathbf{U}$ and the ordering of gates in the diagram. Note also the example, in the lower half of the figure, of the even more irritating general feature of circuit diagrams that follows from this reversal, illustrated in Figure 1.3 of Chapter 1.

# Figure 2.13

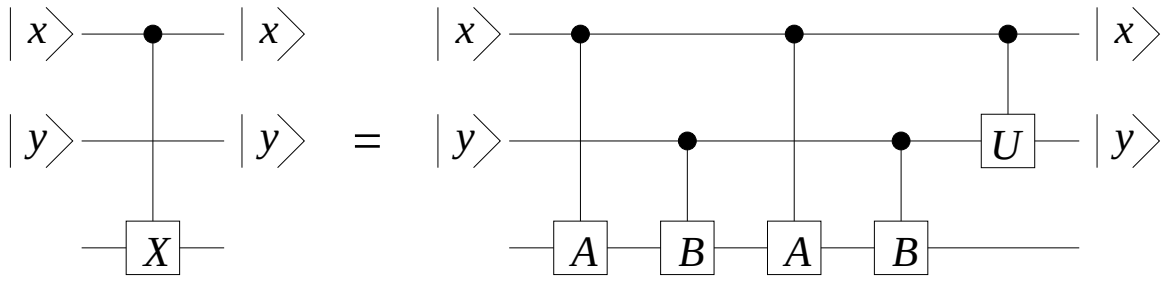


Figure 2.13. How to make a doubly-controlled NOT (Toffoli) gate using only 6 cNOT gates. The unitary operators $\mathbf{A}$ and $\mathbf{B}$ are given by $\mathbf{A} = \mathbf{a}\cdot\sigma$ and $\mathbf{B} = \mathbf{b}\cdot\sigma$ for appropriately chosen real unit vectors $\mathbf{a}$ and $\mathbf{b}$. Since $\mathbf{n}\cdot\sigma$ can be expressed as $\mathbf{V}^\dagger\big(\mathbf{x}\cdot\sigma\big)\mathbf{V}$ for appropriate unitary $\mathbf{V}$, and $\mathbf{X} = \mathbf{x}\cdot\sigma$, each controlled-$A$ and controlled-$B$ gate can be constructed with a single controlled-NOT gate and 1-Qbit unitaries. Since $\mathbf{A}^2 = \mathbf{B}^2 = \mathbf{1}$, it follows that the target Qbit (bottom wire) is not acted upon unless $x = y = 1$, in which case the target Qbit is acted on by $(\mathbf{BA})^2$. The product $\mathbf{BA}$ is the unitary $\big(\mathbf{b}\cdot\sigma\big)\big(\mathbf{a}\cdot\sigma\big) = \mathbf{b}\cdot\mathbf{a} + i\big(\mathbf{b}\times\mathbf{a}\big)\cdot\sigma$. Pick the unit vectors $\mathbf{b}$ and $\mathbf{a}$ so the angle between them is $\pi/4$ and they lie in the plane perpendicular to $\mathbf{x}$ with their vector product directed along $\mathbf{x}$, so that $\big(\mathbf{b}\cdot\sigma\big)\big(\mathbf{a}\cdot\sigma\big) = \cos(\pi/4) + i\sin(\pi/4)\mathbf{x}\cdot\sigma$. Then $(\mathbf{BA})^2 = \cos(\pi/2) + i\sin(\pi/2)\mathbf{x}\cdot\sigma = i\mathbf{x}\cdot\sigma = i\mathbf{X}$. So the alternating controlled-$A$ and -$B$ gates produce a doubly-controlled-NOT gate except for an extra factor of $i$ accompanying the NOT. The controlled-$U$ gate on the right corrects for this. Here $\mathbf{U}$ is the 1-Qbit unitary $e^{-i(\pi/2)\mathbf{n}}$, so controlled-$U$ acts as the identity unless $x = y = 1$ in which case it multiplies the state by $e^{-i\pi/2} = -i$. This cancels the unwanted factor of $i$ produced by the controlled-$A$ and -$B$ gates when $x = y = 1$. Since any controled-$\mathbf{U}$ gate can be constructed with two cNOT gates and 1-Qbit unitaries, using the construction in Figure 2.12, this adds two more cNOT gates to the construction for a total of six. Alternatively, one can view this as constructing a Toffoli gate from four cNOT gates and a single controlled-phase gate of precisely the kind that plays a central role in the quantum Fourier transform described in Chapter 3.