Quantum Computing: A Full Introduction for the Non-Physicist

Riley Scott Jacob*
Cornell University, Department of Physics

I. BACKGROUND & MOTIVATION

The computing revolution can without a doubt be considered one of the greatest transformations in all of history. It enables us to solve some of the world's trickiest problems. There are some problems, however, too tricky even for them. This is not always a bad thing – secure encryption across the internet, for example, is only possible because it is excruciatingly difficult to factor large numbers on a computer. Other times, though, we encounter problems whose solutions would benefit us immensely. Protein folding is a problem notoriously difficult to solve for even the simplest of structures on classical computers, but solving it would mean the ability to develop targeted drugs; ones cheaper, more effective, safer, and developed faster.

From there comes the idea of the *quantum computer* – a machine capable of calculating the complex dynamics of quantum-mechanical systems (such as protein folding) by using quantum mechanics itself to do so.

Here, I hope to familiarize the reader with the basic ideas underlying quantum computing without resorting to the trope descriptions of popular-science media, and only occasionally falling back on analogy. The intention is that one leaves not knowing how to do theoretical math, but appreciating and understanding qualitatively the basics of how quantum computation functions.

II. FUNDAMENTALS OF COMPUTATION

One is likely to be familiar with the *bit*. It is the most fundamental unit of information in classical computation. It is binary – it represents a state having one of two values. Those values can be ascribed meaning through a representation; for example, either true or false (T/F), on or off, or perhaps most commonly as one or zero (1/0). The bit itself has no fundamental attachment to any of these representations, they are simply conventions based on utility in a given moment.

Here we learn a new notation, the Dirac notation. It consists of symbols called *bras*, which look like this: $\langle x|$; *kets*, which look like this: $|x\rangle$; and *bra-kets*, which are a combination of the two, sharing their line: $\langle x|x\rangle$.

We will use $|0\rangle$'s and $|1\rangle$'s to represent our bits, occasionally switching between it and another, like so:

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

This new notation is called vector notation, and it should come as no surprise that the bracketed objects containing multiple numbers are themselves called *vectors*. Having a 1 in the top position indicates the vector represents a 0, and having a 1 in the bottom position indicates the vector represents a 1.

However, bits are not very useful if we cannot do anything with them. The magic of computation arises from the ability to manipulate bits, not simply have them. How can we change our bits?

Think of all the possible things you could ever want to do to a single bit. How many things can one even do? After all, the object itself only has two states! It turns out there are four unique operations we can perform on a single bit, and we will start with the simplest of the bunch: doing nothing. More properly, this operator is called the *identity* – it simply maps the object to itself. In vector notation, this is how we show it acting on both a $|0\rangle$ and a $|1\rangle$:

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

We have a new type of object in these expressions, the bracketed square of numbers $\left[\begin{smallmatrix}1&0\\0&1\end{smallmatrix}\right]$ is a type of *matrix*. In particular, it is the identity matrix. The characteristic feature of the identity matrix is the string of 1's running from the top-left corner to the bottom-right corner – the so-called *main diagonal*.

Much as there are four distinct operations we can perform on our single bit, the astute may notice there are four ways one can uniquely arrange two 0's and two 1's in our 2×2 matrix. This is not a coincidence! Some behaviors even feel intuitive. For example, what if we simply swap the two rows of the matrix, giving us a new one: $\left[\begin{smallmatrix}0&1\\1&0\end{smallmatrix}\right]$? It will look like so:

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

Swapping the rows of the matrix makes it swap the rows of the vector! That means that this matrix turns a 0 into a 1, and a 1 into a 0 – this operation is called *negation*, as it flips the value of the bit.

The other two operations are likewise simple – they are called *set*, and *clear*. The set operation looks as follows:

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

And the clear operation:

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

With that, we have fully covered all the ways in which it is possible to manipulate a single bit! One may feel the logical continuation, then, is to ask in what ways we may manipulate two.

The Tensor Product

Just as we represent a single 0 as $|0\rangle$, so too do we represent two 0's, 00, as $|00\rangle$ in Dirac notation. But how do we meaningfully extend this to vector notation? We employ a new operation, called the *tensor product*, symbolized as $\otimes$. For two vectors, $|a\rangle$ and $|b\rangle$, we find their tensor product $|a\rangle \otimes |b\rangle$ as follows:

$$|a\rangle \otimes |b\rangle = \begin{bmatrix}a_0\\a_1\end{bmatrix} \otimes \begin{bmatrix}b_0\\b_1\end{bmatrix} = \begin{bmatrix}a_0\begin{bmatrix}b_0\\b_1\end{bmatrix}\\a_1\begin{bmatrix}b_0\\b_1\end{bmatrix}\end{bmatrix} = \begin{bmatrix}a_0b_0\\a_0b_1\\a_1b_0\\a_1b_1\end{bmatrix}$$

In words – imagine the first vector $|a\rangle$ is n elements long. Above, $n = 2$. Then, for each integer $i < n$ (above, for $i = 0, 1$), we refer to the i-th element of $|a\rangle$ as a_i. For each of these elements, we create a new vector $|i\rangle$ which is simply a copy of $|b\rangle$, but with each of its elements multiplied by a_i. That is, $|i\rangle = a_i|b\rangle$. Then, we simply stack each of these vectors on top of one another in the order we made them.

This is the operation we use to extend vector notation to multiple bits. We define $|00\rangle = |0\rangle\otimes|0\rangle$, $|10\rangle = |1\rangle\otimes|0\rangle$ and so on. For all four two-bit states, this is what the vectors look like:

$$|00\rangle = \begin{bmatrix}1\\0\end{bmatrix} \otimes \begin{bmatrix}1\\0\end{bmatrix} = \begin{bmatrix}1\begin{bmatrix}1\\0\end{bmatrix}\\0\begin{bmatrix}1\\0\end{bmatrix}\end{bmatrix} = \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$$

$$|01\rangle = \begin{bmatrix}1\\0\end{bmatrix} \otimes \begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}1\begin{bmatrix}0\\1\end{bmatrix}\\0\begin{bmatrix}0\\1\end{bmatrix}\end{bmatrix} = \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$$

$$|10\rangle = \begin{bmatrix}0\\1\end{bmatrix} \otimes \begin{bmatrix}1\\0\end{bmatrix} = \begin{bmatrix}0\begin{bmatrix}1\\0\end{bmatrix}\\1\begin{bmatrix}1\\0\end{bmatrix}\end{bmatrix} = \begin{bmatrix}0\\0\\1\\0\end{bmatrix}$$

$$|11\rangle = \begin{bmatrix}0\\1\end{bmatrix} \otimes \begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}0\begin{bmatrix}0\\1\end{bmatrix}\\1\begin{bmatrix}0\\1\end{bmatrix}\end{bmatrix} = \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$$

Conveniently, it is also always[1] possible to factor these longer vectors back out into their original form. Now, we have complete vector representations for all of our 2-bit states. What kind of manipulations can we perform on them, though?

The Controlled NOT

One such operation is of particular importance: `CNOT` – the *controlled `NOT`*. This operator operates on 2-bit states. One of the bits is called the *control bit*, and the other the *target bit*. The operator functions by negating the target bit depending on the state of the control bit. If the control bit is 1, the target bit will be flipped – it will have the negation operation described above applied to it. Otherwise, the identity will apply, in which case nothing changes. `CNOT` never changes the value of the control bit. In our notation, we consider the first bit the be the control bit. That is, if the control bit is labeled C, and the target bit T, then our 2-bit states are labeled $|CT\rangle$.

We want to be able to represent this operator in its matrix form, like we did in the case of the single-bit operators. The matrix looks like this:

$$\texttt{CNOT} = \begin{bmatrix}1 & 0 & 0 & 0\\0 & 1 & 0 & 0\\0 & 0 & 0 & 1\\0 & 0 & 1 & 0\end{bmatrix}$$

Notice the first two rows are simply the identity. This is exactly what we want, because the control bit corresponds to the first two rows of the vector, and we never want to change the control bit. However, we always flip the bottom two rows, as the bottom two rows of the matrix are the same as the negation matrix described previously. How are we able to always flip the bottom two rows when we only want to negate the target bit when the control bit is 1?

This turns out to not be an issue! Notice that the possible 2-bit vectors only have a 1 in the two bottom rows when the control bit is 1. If the control bit is 0, then both bottom rows only contain 0's – so "flipping" them leaves them exactly the same! This is exactly the change we were hoping to perform. As an example: $\texttt{CNOT}|10\rangle = |11\rangle$ and $\texttt{CNOT}|01\rangle = |01\rangle$. This operation will serve as our most important building block in quantum computing. [1]

---

[1] In quantum mechanics, sometimes it is not possible to factor the vectors. When this happens, we say that the qubits are *entangled*. They have no meaningful value when considered on their own – they must be treated as a single entity.

III. QUANTUM COMPUTATION

Much as the bit is the classical unit of information, the quantum bit – called the *qubit* – is the fundamental unit of information in quantum computing. You will be pleased to learn that all of the notation described above is precisely the same notation used for representing qubits! Our classical bits can be thought of as simply being special cases of qubits. What makes qubits special is that the value of the numbers in the matrices and vectors are not required to be 0 or 1. They can take other values, such as 1/2.

Superposition

Although the values of the vector elements can be things other than 0 or 1, this does not mean they can take just any value. There is a special relationship which much be preserved between them. That relationship is that the sum of the squares of all elements in the vector must equal one. For some two-element vector like we have been working with, $\left[\begin{smallmatrix} a \\ b \end{smallmatrix}\right]$, that means this:

$$|a|^2 + |b|^2 = 1$$

Notice the absolute values – they are not there for no reason. They are there because our vector elements can even be negative! Why this relationship must be true will become clear later.

Interesting things start to happen here. How are we meant to interpret these objects? It is simple enough to have $\left[\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right]$ represent a 0, and $\left[\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right]$ represent a 1. But what does an object like $\begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$ even mean? Herein we see one of properties of quantum states: this is called *superposition*. It tells us that this object represents a state which is neither a 0 nor a 1. It is instead both at the same time. The relative values of the elements tells us to what degree the state has some degree of "zeroness" as opposed to "oneness".

The top number encodes the amount of zeroness that the qubit has, and likewise the bottom number encodes the amount of oneness that the qubit has. A qubit that has complete zeroness and no oneness is then just simply a pure zero, effectively equivalent to the classical counterpart in meaning – just as we would expect from discussion in the previous section. These classical analog states with pure zeroness or oneness are termed *pure states*. All others are known generally as *mixed states*.

Superposition in reality is not so weird as popular media often makes it sound, and there are plenty of analogies to that end. For example, imagine dropping two stones in a pond some distance apart. As the ripples from both radiate outward, it is easy enough to say "these ripples are from the first stone" or "these are from the second". But at some point, the ripples start to intersect. Looking at a point where they do so, you now must say "the ripples here are from both stones at the same time". It is much the same with our qubits.

Measurement

When it comes time to look at the result of our quantum computer's work, we have something of an issue. Our outputs are qubits; but, because of the nature of the universe, we are disallowed from ever observing a mixed state directly. We find that when we look, the qubits collapse to a pure state – either $\left[\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right]$ or $\left[\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right]$. Which one it ends up as depends on the values of the elements. For a qubit $\left[\begin{smallmatrix} a \\ b \end{smallmatrix}\right]$, it will collapse to a 0 with probability $|a|^2$, and 1 with probability $|b|^2$.

From this, it should also become clearer why the special relationship the elements have must be true. Recall that valid states must obey

$$|a|^2 + |b|^2 = 1$$

This is precisely because the probability of measuring a 0 is $|a|^2$, 1 is $|b|^2$, and those are the only two options. You must measure *something*, so these probabilities must add to 1.

As an example, a very common vector is $\begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$. With our new knowledge of how the probabilities work, can you see why? Squaring the elements means that we have a probability of 1/2 for both a 0 and 1 – the state is evenly split between the two.

Reversibility

Quantum mechanics has the property that it is *unitary*. Roughly, this means that all operations must be reversible. We are not allowed to lose information through our operations – there should always be a "reverse operation" we can do to bring our state right back to where it was.

This has a few implications for our operations. Think back to the single-bit operators. Some of them are reversible, like the identity, or negation. This is because we know that the negation operator always flips the bit. If one has the output bit, it is simple enough to just flip it back (indeed, it will turn out that all operators will be their own inverses). The set operator, however, does not have this property. If one has the output bit (which will always be a 1), one has no way of knowing whether it used to be a 0, or had still just been a 1. This will be important to keep in mind when we start to design our quantum operators.

The Hadamard Gate

The time has come to introduce our first truly quantum operator, or gate. It acts on a single qubit. In particular, when acting on a pure 0 or 1, it puts the qubit into a state of equal superposition:

$$H|0\rangle = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$$

$$H|1\rangle = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{-1}{\sqrt{2}} \end{bmatrix}$$

Notice the only difference is that the output state when the input is a 1 has an extra minus sign. This does not change measurement outcomes if we were to measure the qubit (because -1 squared is just 1), but is necessary for reversibility (to make them distinguishable) and can have implications in other computations down the line. Notice as well that the matrix representing the operation can also be written more compactly as H.

As hinted at earlier, all quantum gates are their own inverses. So, the Hadamard gate can also be used to recover pure states from these perfectly mixed superpositions. That is, $H^2|0\rangle = |0\rangle$ and $H^2|1\rangle = |1\rangle$. Applying the gate twice simply does nothing.

THE DEUTSCH ORACLE

To conclude this overview of the basics of quantum computation, we will solve an actual problem of the type discussed in the introduction: a problem whose quantum-mechanical solution is objectively faster than the classical one.[2]

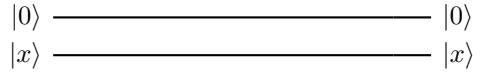
In this problem, we have some black box, called the oracle, which implements one of the four single-bit operations from section II. We do not know which. The four functions can be categorized into two types: constant, and balanced. The constant functions are set and clear (because they will always return the same thing regardless of input). The balanced functions are the identity and negation (because they will always return a balanced mix of 1's and 0's). The challenge is to determine whether the function in the box is constant or balanced by querying the oracle as few times as possible.

It is easy to consider how this would be done classically. Simply query the oracle twice, inputting a 0 one time, and a 1 the other. The set of both outputs then uniquely identifies the function, and therefore also whether it is constant or balanced. There is no way to know from one query alone.

What is the minimum number of queries to complete the task with a quantum computer? First, we must implement each of the single-bit operators quantum-mechanically. When we attempt to do so, however, a problem arises immediately. The constant functions, set and clear, are not reversible.

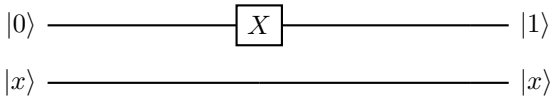
This is relatively simple to fix, though. Just add another qubit, which is always input as being a 0. This qubit will be used to store the output from our calculation, and the input qubit will be left alone. That way, we have the output we wanted, while also retaining what the input had been. Reversibility is preserved!

First, we will implement the clear function because it is simplest. It looks like so:

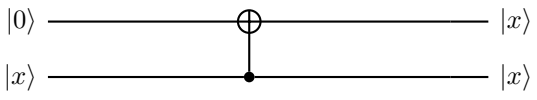


Here we introduce quantum *circuit notation*. The lines represent the physical qubits. Later, we will connect the lines and add gates, to allow them to interact and exchange information. It just so happens that for the clear operation, we do not wish to do anything to the qubits at all. The spare qubit we use to write the output too already contains a 0, which is what the clear operation always outputs. The input qubit is also left unchanged, as is required for reversibility.

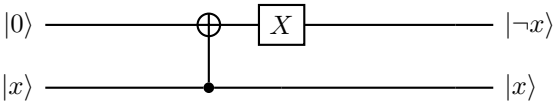
The set operation is similar. All we need to do is perform a bit flip (swap the rows of the vector) on the output, so that it always give a 1. In quantum circuit notation, the bit flip gate is represented by an X, making the implementation looks like this:



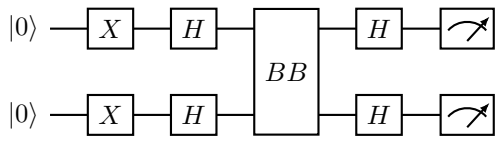
To implement the identity, we will need the `CNOT` operator. In the new notation, it is represented by a line connecting the two qubits. A dot is drawn on the control bit, and a circled cross is drawn on the target bit. We need to use this operator because we need to copy the input qubit to the output qubit only if the input qubit is 1. If it is 0, the output qubit already contains the correct value. Drawn, it looks like this:



Negation is simply the identity flipped, so we can add a bit flip to the previous circuit:



Now that we have implementations for each of the single-bit operators, we can finally query the black box, represented by BB. We do so like follows:



The meter symbols represent taking a measurement. Notice, we always input a 0, and then both our spare line for the output and our input line are given the same treatment. First they are flipped, to give 1's. Then, they are put into a superposition using the Hadamard gate, leaving each of the qubits in the state $\begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix}$. Let us then see what happens for each of the four functions.

Starting with the clear function – the function does not modify the qubits, so they pass through. They reach a second set of Hadamard gates, which takes them out of superposition, giving us a 1. Both lines are 1, so the system outputs the state $|11\rangle$.

For the set function, we flip the rows of our superposition state. Passing through the Hadamard gate takes the qubit out of superposition, leaving the state $\begin{bmatrix} 0 \\ -1 \end{bmatrix}$. This will collapse to a 1 upon measurement, and once again the system outputs the state $|11\rangle$.

The identity has a `CNOT`, which will make things complicated. There is no way around doing some math:

$$\texttt{CNOT}\left[\begin{bmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{bmatrix} \otimes \begin{bmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{bmatrix}\right] = \frac{1}{2}\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix}$$
$$= \frac{1}{2}\begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} \otimes \begin{bmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{bmatrix}$$

Thankfully, we know exactly how to apply the Hadamard gate to this. The top line will output 1, and the bottom line will output 0. So, the system is left in state $|01\rangle$.

The last one: negation. We can take the same result from the previous function, but bit flip the top line before performing the Hadamard gate. Like with the constant function, this gives us the state $\begin{bmatrix} 0 \\ -1 \end{bmatrix}$, which collapses to a 1 upon measurement. Therefore, the system is left in the state $|01\rangle$.

Notice there are only two possible states the system can be left in: $|01\rangle$ or $|11\rangle$. Notice that these states exactly correspond to whether the function inside the black box was constant or balanced. Therefore, we have shown that it is possible to solve the problem with only a single query of the oracle, beating the performance of the classical computer in the process.

* rsj56@cornell.edu

[1] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter, Phys. Rev. A **52**, 3457 (1995).

[2] N. Johansson and J.-Å. Larsson, Quantum Information Processing **16**, 233 (2017), arXiv:1508.05027 [quant-ph].