Circle Paper: Visualizing Quantum Computation

(Revised January 25, 2015)

Fine Print: This document is not finished, but you're welcome to read it anyway. It has not been peer-reviewed, or even thoroughly researched. These are my own notes and speculations, with shiny pictures added. If you have questions, comments, or really good jokes, please contact me at ej@machinelevel.com

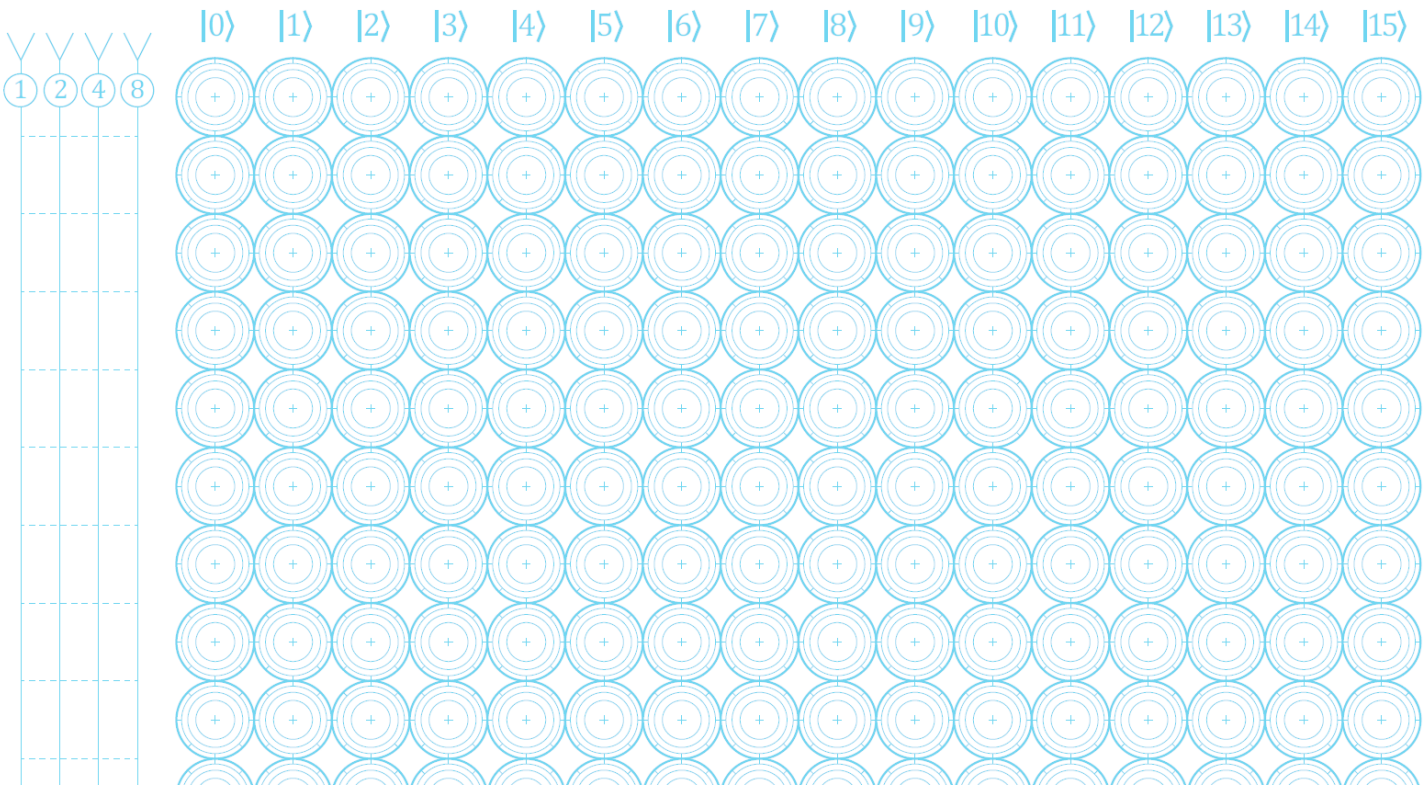
Overview

To an engineer, the notation used in quantum computation can be disorienting. Additionally, equations written in ket-notation become elaborate as the complexity of the operation grows. While a Bloch Sphere is a useful single-qubit visualization, it does not scale easily to multiple-qubit systems.

This document describes a simple handwritten system I find useful for getting an intuitive feel for QC operations, the same way one does for everyday arithmetic. While this still doesn't make it easy to solve arbitrarily complex math without a computer, it does allow ballpark estimates to be done, and also a quick assessment of whether a result seems reasonable.

The 'circle-paper' notation will be introduced, and then some common QC operations will be mapped onto it, to demonstrate the process of actually performing QC operations by hand, or even in your head.

You can print your own here.



Finally, I'll use this method to demonstrate quantum teleportation. This is a great way to take the 'spooky' out of 'spooky action at a distance'.

The Information in a System of Qubits

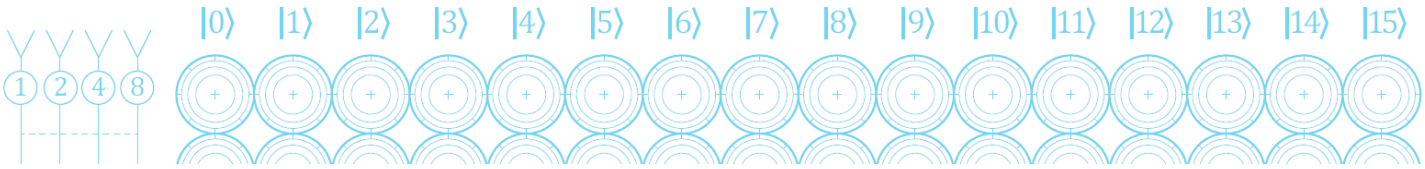
As engineers, we're going to be building devices with qubit-based information cores. We've all heard that qubits can hold multiple values at the same time, and that's why they're amazing. That doesn't describe the nature of qubits very completely, so here it is:

- A system of *n* qubits can be represented exactly by a 2^{n+1}-dimensional unit vector.

That's it, really. If you've worked with computer graphics, then a normal vector is an example of a 3-dimensional unit vector. It has scalar components *x*, *y*, and *z*, with $x^2 + y^2 + z^2 = 1$.

A *single* qubit is effectively a 4D unit vector (that's $2^{n+1} = 2^2 = 4$ dimensions), with the length always equal to 1. Four-dimensional visualizations are mind-bending, so I handle them by drawing a pair of 2D slices. That's convenient as well, because each pair of dimensions can be considered to be a complex number (*real* and *imaginary* if you like, though one isn't actually more real than the other). A complex number is, of course, just another term for a 2-dimsional vector.

On the standard 4-qubit circle paper, you can see space for 16 2-dimensional vectors. When you read a 4-bit number, there are only 16 possible values which can be read. The total information, 16 2D vectors (same as 16 complex numbers) is a 32-dimensional vector ($2^{n+1} = 2^5 = 32$ dimensions), but it's easy to see all at once when sliced up this way.



Critical note: With actual physical qubits, these circles aren't visible at all. This is a functional model, and it's useful, but the *only* information we can physically read is one bit per qubit. Reading this 4-qubit system will just return a number in the range [0, 15] every time, and the process of reading is destructive, collapsing qubit system to just a 4-bit value. We'll get into that soon.

Qubit Operator Pairs

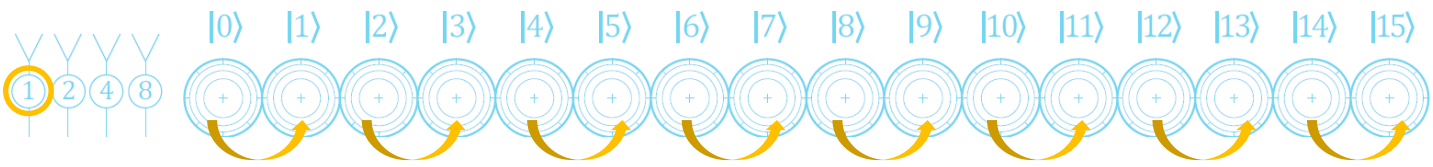
An important concept to get familiar with now is the idea of *qubit operator pairs*. We're going to use this in every single step coming up, and it's fairly simple. When a quantum operator acts on a specific qubit, each circle in the row will interact as a matched pair, with one of the others.

- Each ***qubit operator pair*** is a pair of circles whose value differs only by bit represented by the operation's target qubit.

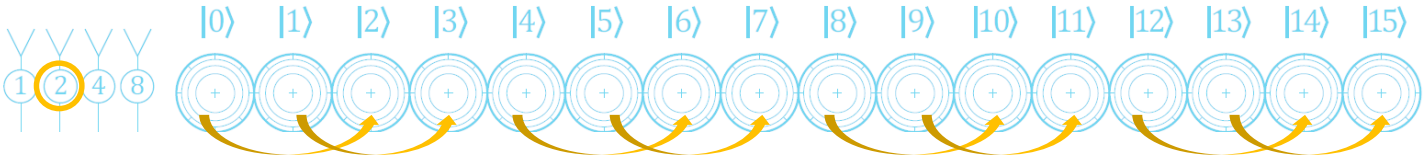
Here's how to find them. Take any circle's value ("7" for example), and flip the bit which corresponds to the operation being performed. So for an operation on the 2's place bit, 7 is 0111 in binary, so flipping t e 2 bit gives 0101, which is 5. For an operation on the 2-bit, 5 and 7 are an operator pair.

As there are only four bits in this version of the circle paper, we can just show all of the pairs explicitly:

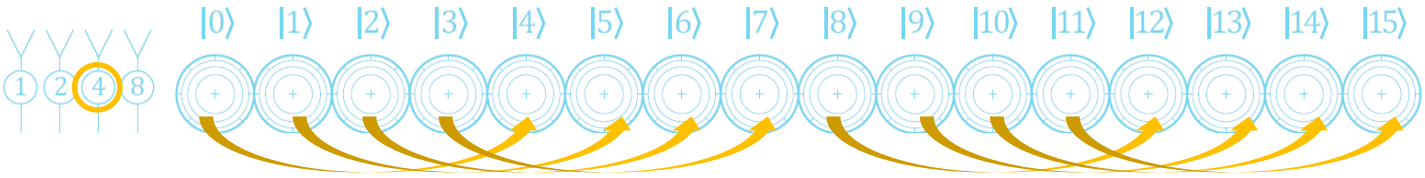
Qubit Operator Pairs for bit 1:



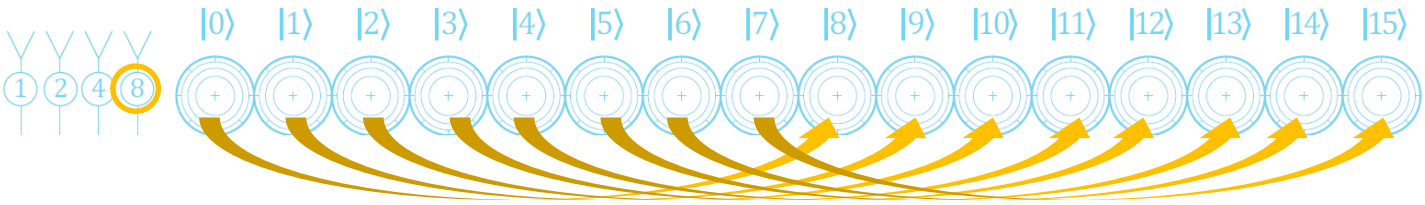
Qubit Operator Pairs for bit 2:



Qubit Operator Pairs for bit 4:



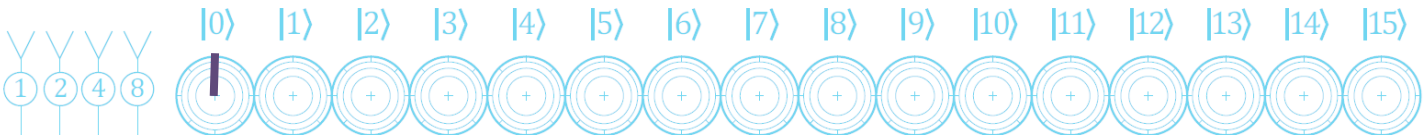
Qubit Operator Pairs for bit 8:



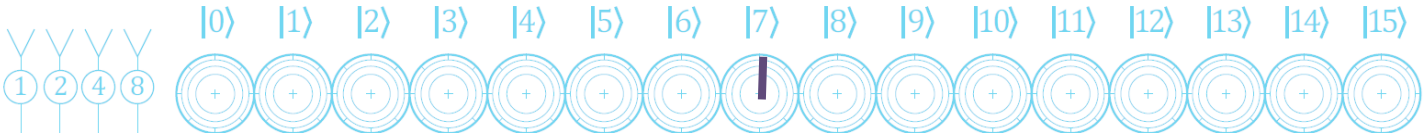
Barely Starting: Initialize the System

We'll get into reading and writing values later, but just initializing this system with a simple value is easy. Pick the circle which the value you want (that's often zero), and draw a line from the center to the rim. Any direction is fine, but I usually go vertically to keep things looking neat.

Here's the very common case where out qubits have been initialized to zero:



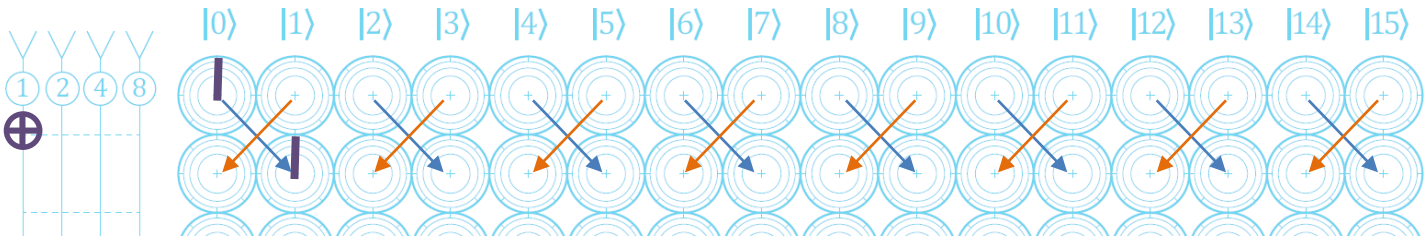
…and here's the case where we've chosen the binary value 0111, or the number 7:



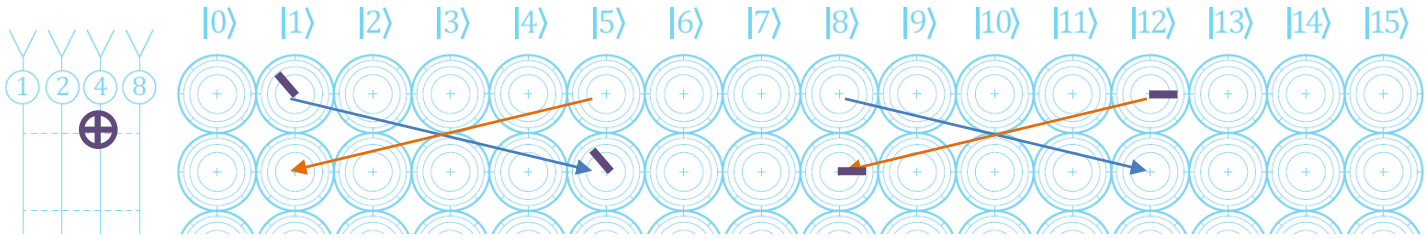
Basic Operations: NOT

Let's jump right in and start performing operations. The most basic operation in QC is a NOT. In digital logic, this is just the equivalent of flipping a bit from 0 to 1, or vice versa.

To perform a quantum-NOT using circle paper, simply identify the pairs for the bit being flipped, and swap the circles. For example, if we want to perform a NOT on bit 1, we'll just exchange the bit-1 pairs.



In this case, there was only one mark to swap, but take a look at what happened. We took a register with value 0, and flipped the 1 bit, which resulted in the value 1, just as it would in digital logic. Let's take a more interesting quantum example:



In this case, we started with a register which was holding *both* 1 and 12, in quantum superposition, with a relative phase of about 135 degrees. If we had read the bits in a real QC, they would have collapsed to return either "1" or "12", with 50% probability of either.

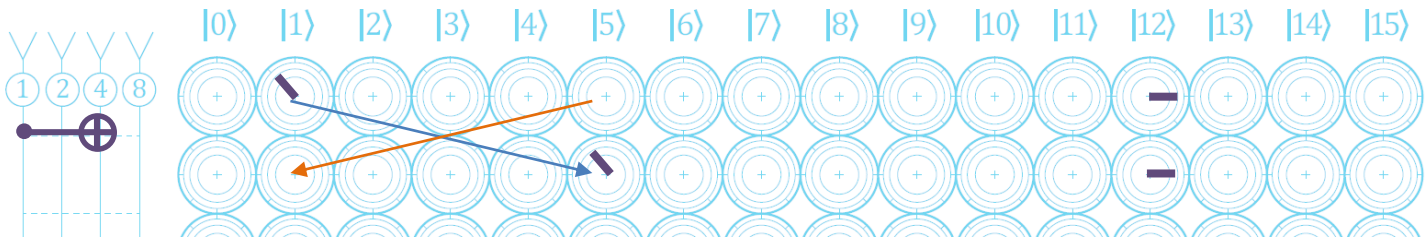
By flipping the 4 bit, we swapped the circle-pairs again, and end up with a register containing the values 5 and 8. From a digital logic standpoint, this makes perfect sense. Once again, we've just flipped the 4 bit.

You can perform multiple not operations simultaneously if desired, but I find I make errors when I try to do that, especially if the circle-vectors are already in a non-simple state. Doing them one at a time is easy.

Basic Operations: CNOT

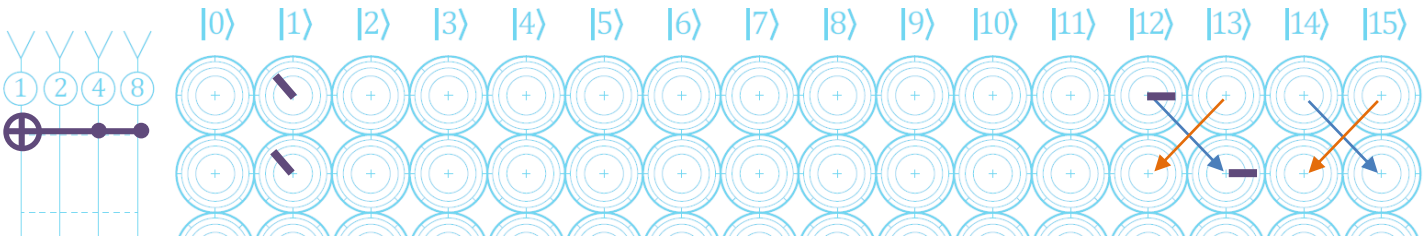
The conditional-not operation is a mainstay of QC. Most quantum programs are full of them, and the digital analog is "the condition bit is 1, flip the target bit." This operation is used in entanglement, and many other interesting QC operations.

On circle paper, this is simple. Swap the pairs, **just like the NOT** operation, except that you'll swap only the ones which have the condition qubit set to 1. Here's an example:



In this case, we have a CNOT with condition 1 and target 4, so we're going to do a bit-4 pair swap, but ***only*** for values with bit-1 set. So 1 (binary 000**1**) gets swapped to become 5, but 12 (binary 110**0**) is left alone.

A Toffoli gate, along with other multi-condition gates, can be executed by only swapping circles where ***all*** of the condition bits are 1. Here's an example of that:



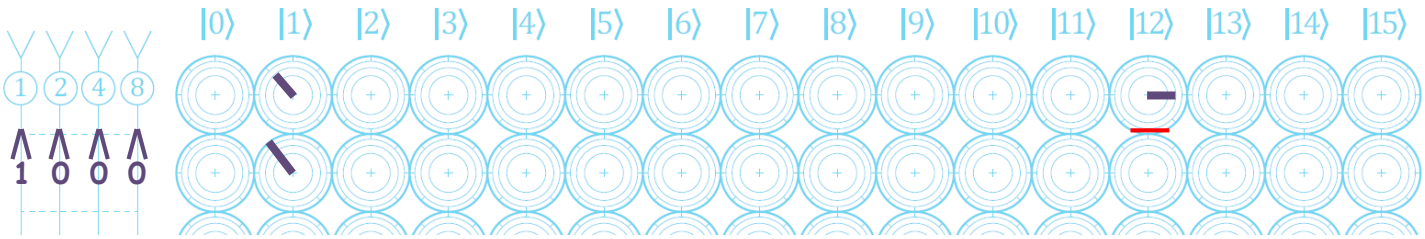
In this case, 1 (binary **00**01) does not get swapped, but 12 (binary **11**00) has both condition bits set, so the target bit 1 is flipped, and it becomes 13.

Reading Qubit Values

When you read one or more qubits, their values 'collapse' (the exact meaning of this depends on your interpretation, and gets philosophical) into digital values.

On circle paper, it's an easy process to implement. The length of each circle vector represents he *magnitude*, or energy, or probability, of that particular value. On a physical QC, the result will be random, with probability tied to the magnitude of each value.

To read ***all*** qubits at once, simply pick any value with a non-zero magnitude (if there's a clear winner, one vector longer than the rest, picking that one will give you the most likely outcome). Extend that vector to full-magnitude (center-to-rim on the circle) and clear out all of the others. Yep, that's the 'collapse' we were talking about.

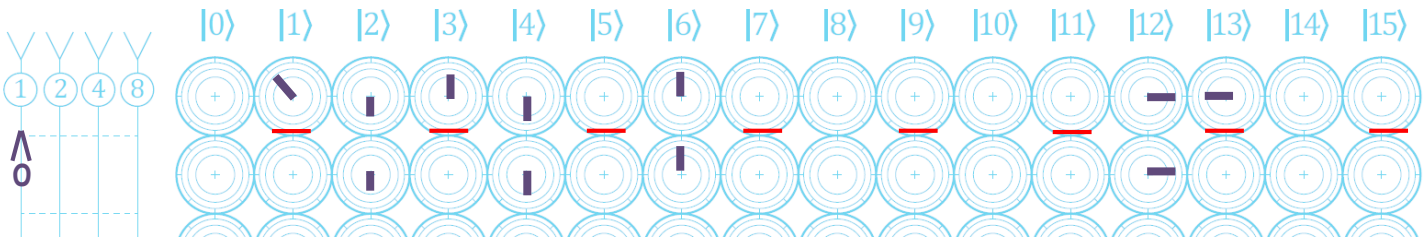


In this case, the probabilities looked about even between 1 and 12, so we picked 1, and collapsed out the 12. If we read it again right away, we can only get the same value, since all of the other circles are empty.

What happens if we only read one qubit? We *should* only collapse part of the quantum state. It turns out that's pretty easy as well. We start the same way, by picking any non-zero value, and generally preferring the ones with the largest magnitudes. Then, we zero out ***only*** the circles where the bits we're reading don't agree with the chosen value. After that, we re-normalize by expanding the remaining values so that the total vector length is about 1.

Note: We're being pretty casual with the magnitudes. That's not because they're unimportant, but because for many operations we only need their approximate values, and this is all sort of a napkin-sketch method anyway.

For example, suppose we read *just* bit 1 in the following system of qubits. Looking at the circles, the value 6 looks likely, so we select it. Then, we simply zero out any values where bit 1 disagrees with out chosen value. (The red lines are just a visual aid, to show what's being collapsed. I don't always draw them, but they're visually useful.)



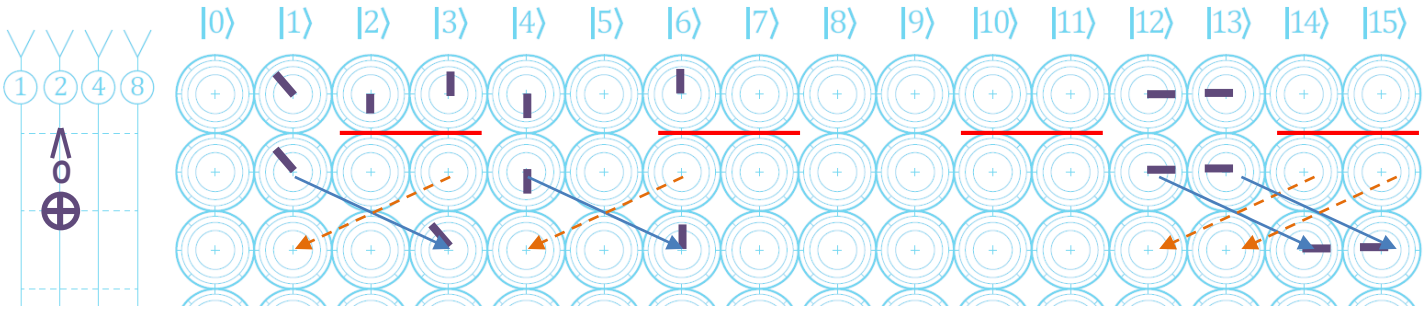
…so now the only remaining values are ones **consistent with the observation** that bit 1 is zero. I didn't bother re-normalizing this time, just removed the collapsed values.

Writing Qubit Values

Earlier when we "initialized" the system of qubits, we were just conveniently placing values as we wanted to. That's not really a physical process, so let's take a look at a valid way to do this. It turns out this is simple. to write qubit values, first we ***read*** those qubits, and then we simply ***not*** the ones we want to change.

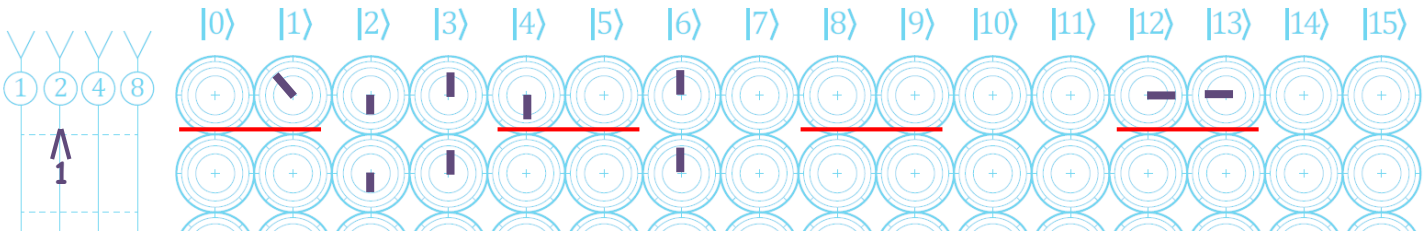
So in the reading example above, suppose we wanted to write a 1 into the 2-bit. First, we read that bit. It looks like the 2-bit is about a 3/7 chance of being 1 (three of the seven visible magnitudes are for values there the 2-bit is a one).

Since it could go either way, let's look at both probabilities. If we read the 2-bit and get a zero, we clear out inconsistent values, and then do a NOT operation, since we want that bit to be a 1.



…so now the only valid values are ones for which the 2-bit is set. Also, notice that we've lost a lot of information in the process. Reading and writing do that. We're left with a quantum system containing values 3, 6, 14, 15 in superposition.

What if reading the 2-bit had returned a 1? If so, we'd collapse the inconsistent values, but there's no need to flip the 2-bit, because it's already what we want.



The result is a register containing values 2, 3, 6 in superposition.

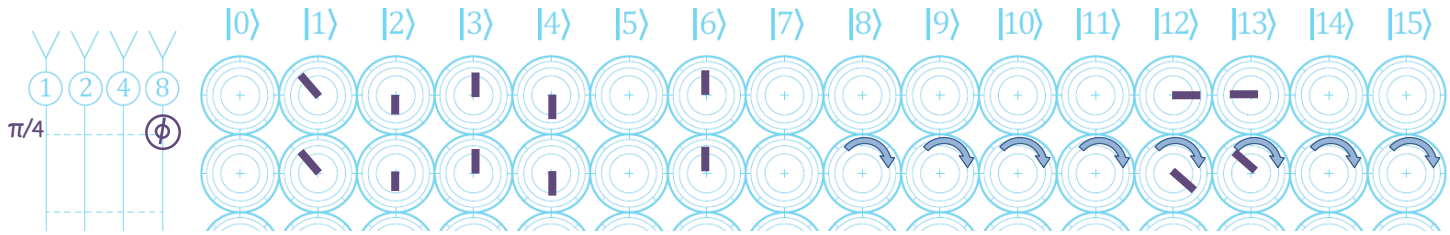
Notice something troubling: writing a bit using this "read and flip" method *changes* the result randomly, depending on the random result of the read, even though the resulting values are all consistent with having written a one into the 2-bit position.

The strangest part of this is that the value 2 drops out of the first case entirely. Here's why that happens: We have a register which can possibly contain the value 2 (binary 0010), but *not* zero (binary 0000). If we read the 2-bit and get a zero, then neither 0010 or 0000 is consistent with the register's state, so flipping the 2-bit can't possibly result in the value 0010, or 2. Yes, it's strange that writing the 2-bit onto the value 2 doesn't always result in 2, but that's the fun of QC's. ***[TODO: re-word this paragraph and fix the terrible 2's.]***

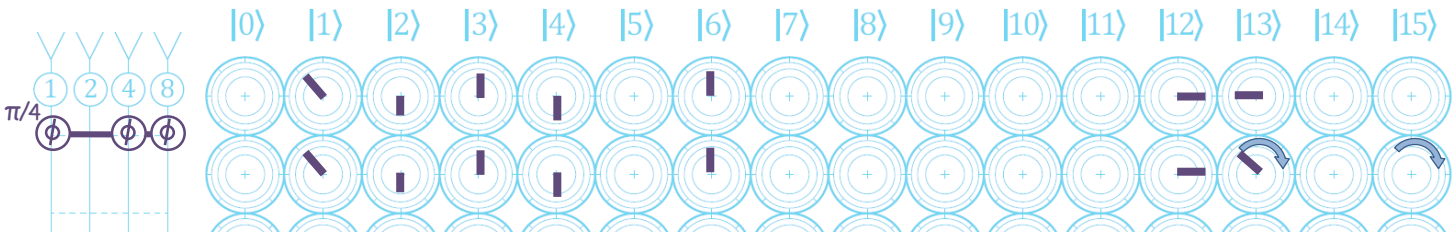
One way to view this anomaly is to view the superposition as containing some implicit logic of its own. “If the 8-bit and 4-bit are zero, and the 2-bit is 1, then the 1-bit must also be 1.” It’s actually pretty neat that complex logical connections can be stored this way. By writing to the bit, we destroy (collapse) some of this logic.

Easy Operation: Phase Gate

Phase gates are perhaps the easiest operation to perform on circle paper. Phase gates don’t really have a target bit, they just have condition bits. So to change the phase of some bits by 45°, just rotate the circles which have all of those bits set to 1. Here are some examples:



Using a π/4 (45°) phase gate on the 8 bit causes a rotation of all of the circles whose values have the 8 bit set, which is values 8 to 15. If we phase-shift more than one bit, our rotation just gets more selective:



A phase gate which operates on the 1, 4, 8 bits can be done by rotating all of the circles which have those three bits set. In this case, that’s just 13 (binary **11**0**1**) and 15 (binary **11**1**1**).

Important Note About Phase: It appears that only ***relative*** phase matters, not absolute phase. As far as I have observed, there is no physical way to tell the absolute phase, so you’re free to spin those circles with no effect, as long as you spin them all by the same amount.

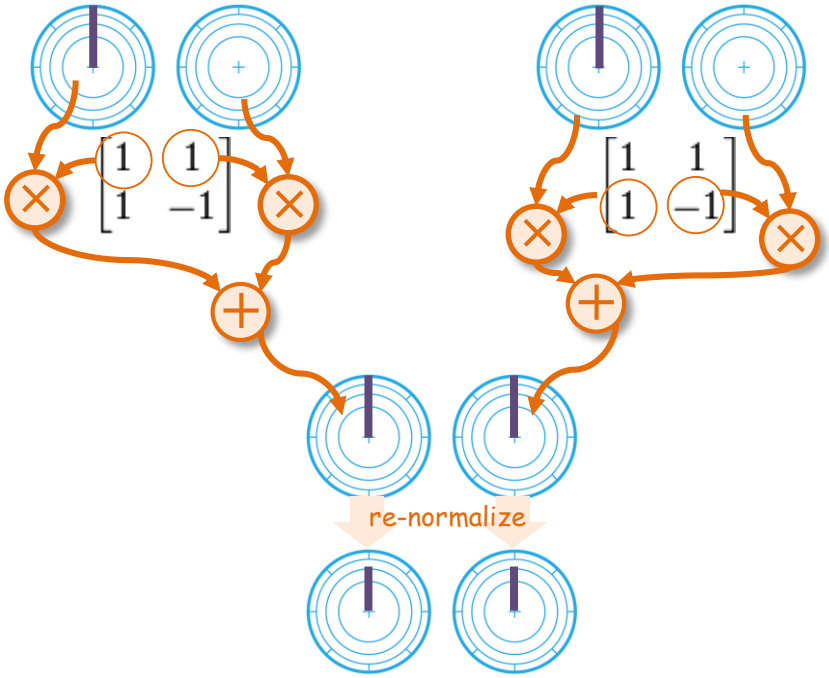
Common Operations: Arbitrary 2x2 Real-only Matrix Op

While reading QC literature (for example, the gates listed in this Wikipedia summary), you’re going to come across operators which look like this:

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

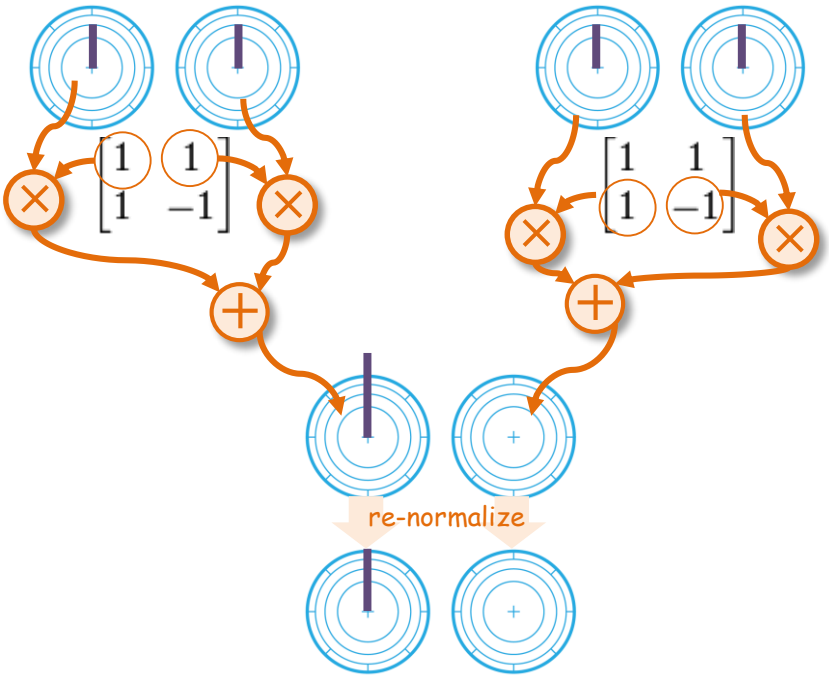
No worries, we can do that pretty easily in most cases. Here’s how it works:

- In the very common case where the whole operator is multiplied by a constant (such as the 1/sqrt(2) here), just ignore the constant and apply it at the end of the operation. It’s there to keep the vector normalized, so you can throw it out. Remember, we’re sketching, not solving exactly.
- Then, find your qubit operator pairs, and apply the matrix like this:

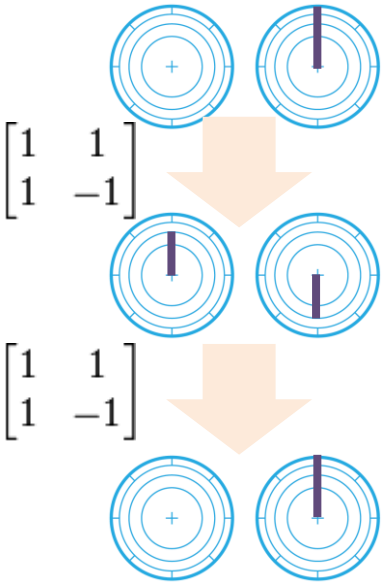


We're just multiplying 2D vectors by scalars and then adding them, but now we're finally getting into operations which are tricky to eyeball. Still, taking them in pairs makes it simple.

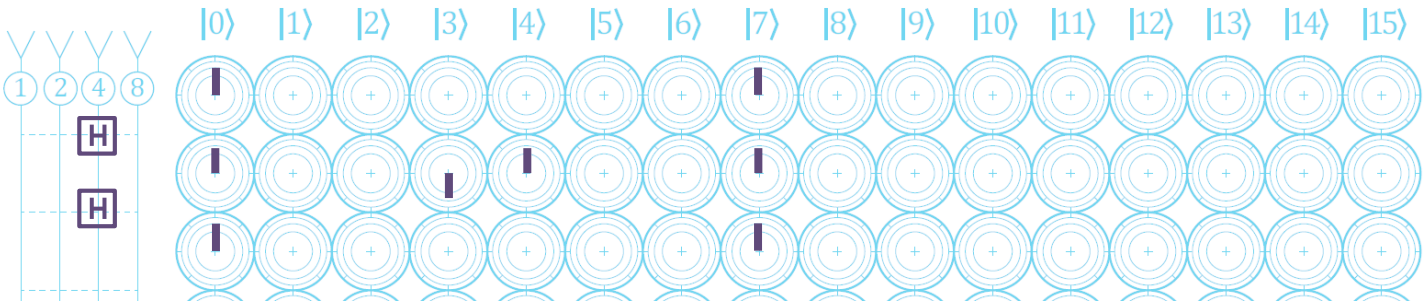
The operation above is a Hadamard gate, so performing this again *should* return the pair to its original state. Let's try it out:



Success! We transformed this pair to a 50/60 superposition and back. Note that if we try this starting with the pair reversed, we see the "phase memory" the Hadamard has, which enables it to return to the original state. This is something we can't do with digital bits at all, since they have no phase, and no fractional magnitudes.



Applied to the pairs of a quantum register, we get something like this (note the pairs are 0-4 and 3-7, as those values differ only in the 4-bit, which is what we're operating on):



Now we're really ignoring the magnitudes, but we still get a sense of what's going on, and what values are likely. Also, this is a ***great*** way to mentally sort out "When I entangle, does it matter if I C-NOT before or after the Hadamard? What if I NOT everything first?" After some practice, you can work these out quickly on paper.

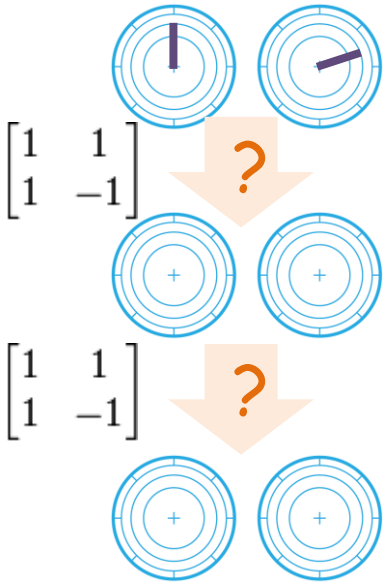
Here's something fun to try: On that same Wikipedia page, there's a Pauli-X gate, which the author *claims* is equivalent to a NOT gate. Here's the matrix:

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

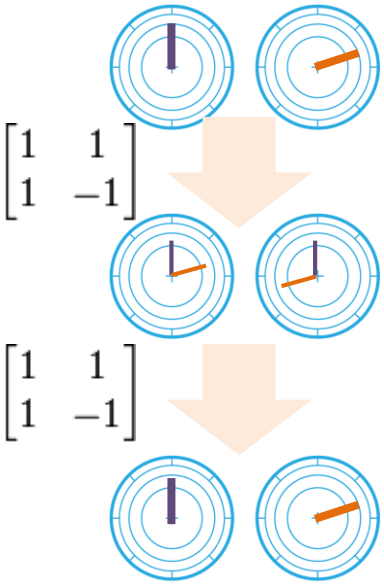
Using the matrix operation described above, you should be able to confirm that this matrix performs a simple pair-swap, which is the same as our NOT-op shortcut mentioned earlier.

Spiffy Trick: Multi-Vector Separation Sketches

Suppose you're doing a Hadamard gate on these qubits:



The phase makes this tricky (it's not even clear what the phase angle is), and we're going to end up with errors if we don't get the angles just right. The Hadamard gate won't really introduce any *new* angles, it's just going to negate parts. It turns out that you can simply do these separately, as though they were two totally different operations, draw both vectors, and carry forward the work. Happily, in this case the multi-vectors cancel out entirely in the second operation.



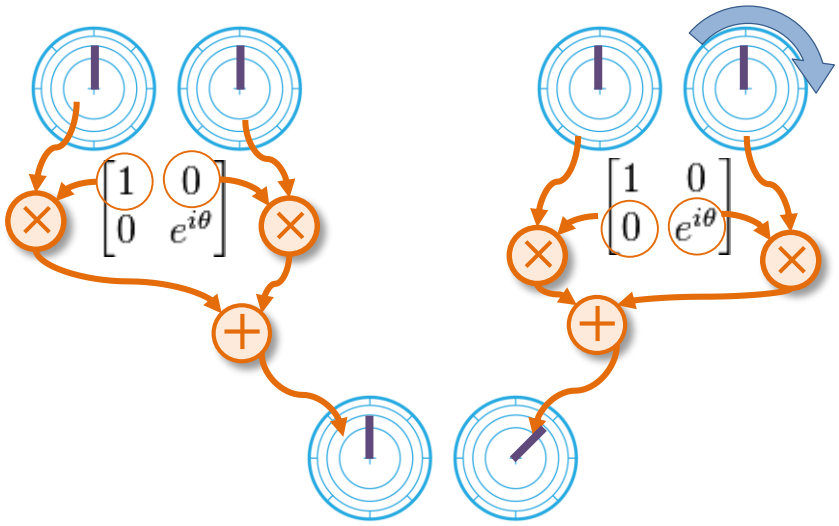
So it actually doesn't matter that we treated one hairy-looking Hadamard as two separate, simple ones. We get the same answer, and it's much easier to sketch.

Common Operations: Arbitrary 2x2 $e^{i\theta}$ Matrix Op

The Matrix operations above use real numbers, but many others don't. Imaginary (or more generally, complex) numbers show up in quantum gate matrices all the time. Not to worry, we can take it. Consider this:

$$R_\theta = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{bmatrix}$$

All we need to do in this case is use exactly the same matrix-combo method, but *rotate* the circle by theta before we apply it. So this one just works like this (let's use 45° for theta):



Again, we're just combining vectors. If you said "Hey that's just a phase shift! We did that already." you're right. Additionally, if you see something like $1 + e^{i\vartheta}$ then all you need is to add the rotated version to the un-rotated version.

Common Operations: Arbitrary 2x2 Complex Matrix Op

Often, there are i terms without the e. For example, this sort of thing is pretty common:

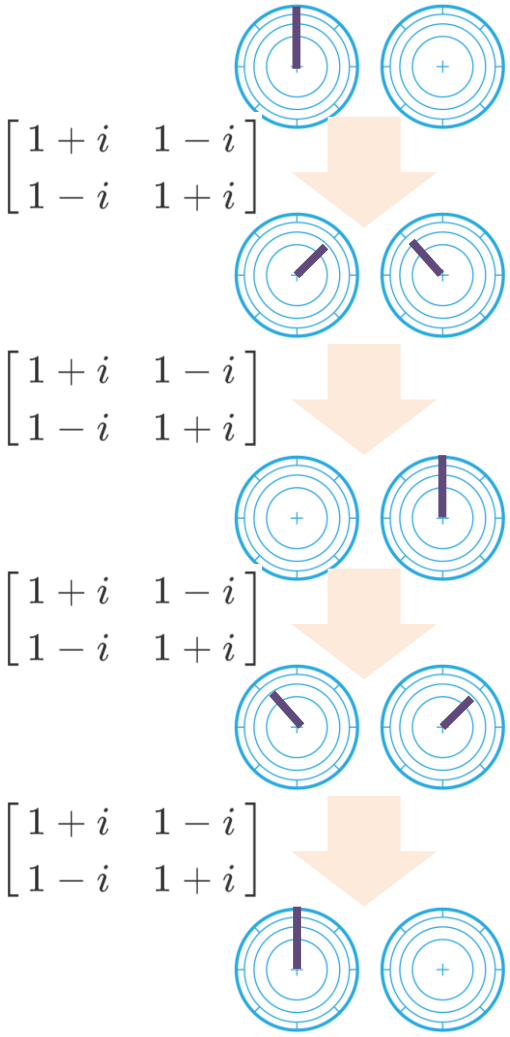
$$\begin{bmatrix} \frac{1+i}{\sqrt{2}} & \frac{1-i}{\sqrt{2}} \\ \frac{1-i}{\sqrt{2}} & \frac{1+i}{\sqrt{2}} \end{bmatrix}$$

First of all, we can throw out the root(2) items, as that's clearly just for normalization. So we're left with:

$$\begin{bmatrix} 1+i & 1-i \\ 1-i & 1+i \end{bmatrix}$$

There are many ways to tackle this, but my personal favorite is to realize that i is equal to $e^{i\,\pi/2}$, so I can treat those terms as just a 90° rotation of the circle. So this operator can be applied as follows (four times in a row, just for fun):

(Note that the 45-degree angles come from adding the un-rotated vector to the 90-degree rotated vector.)



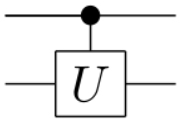
This is actually something called a “root of NOT” gate. Notice that when it’s applied twice, it performs a NOT operation. Like the Hadamard, this is something you just can’t do with digital logic. The quantum version is able to use “phase memory” to remember which direction it’s going. ***[TODO: Explain this 1-i business more clearly.]***

Controlled Operators

When you see a larger matrix in QC literature, it’s very often in this form:

$$\mathrm{C}(U) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & x_{00} & x_{01} \\ 0 & 0 & x_{10} & x_{11} \end{bmatrix}$$

That’s simply a “controlled gate”, like this…

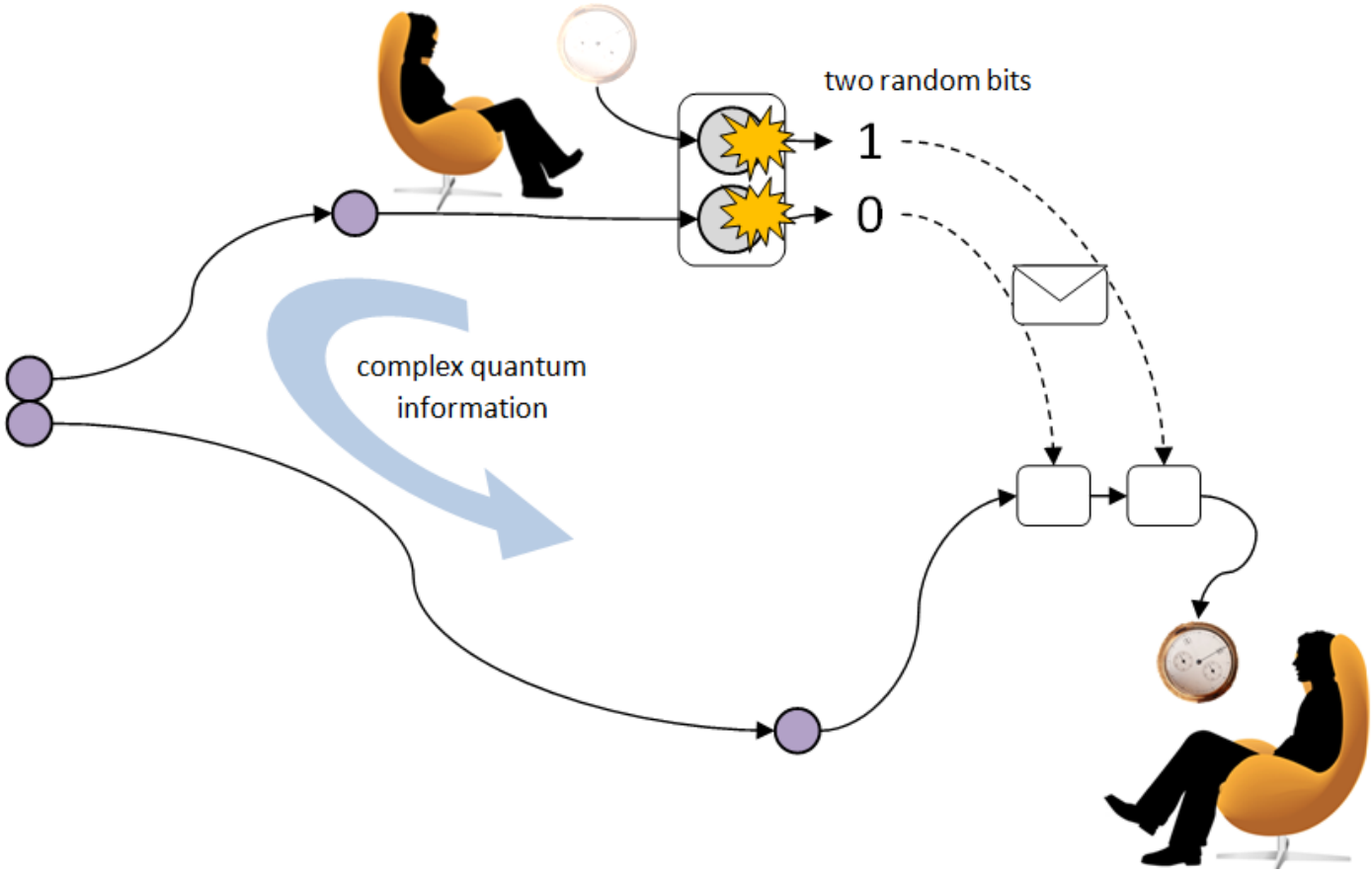


…and to apply it on circle paper, just do the 2x2 pair-op as you normally would, but don’t apply it to any pair which has the control bit set to zero. For quick reference, see how we handled the CNOT gate earlier.

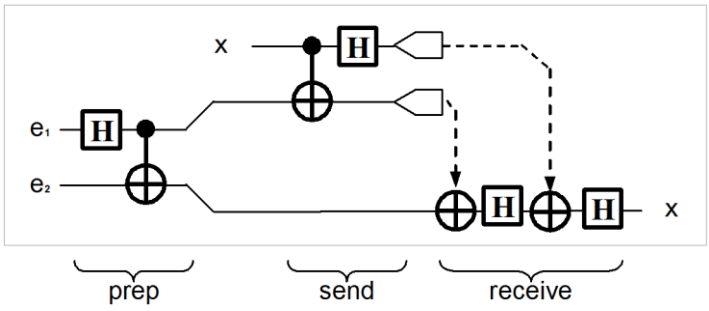
Finally, Let's Teleport Something

If you're not familiar with quantum teleportation, here's the 30-second version. Alice has a qubit (A) she wants to send to Bob. As we saw above, a qubit is a 4-dimensional unit vector, but if you read it, it will collapse into a single bit.

To teleport, she and Bob will need a pre-entangled pair of qubits (E_1 and E_2). Let's assume they planned ahead, and have these handy. Alice performs a two-bit operation on A and E_1, and then destroys them by reading both of them. She sends Bob the two resulting (non-quantum) bits.



Bob takes these, and uses them to turn E_2 into the qubit Alice teleported. It's not a copy, as the original was destroyed. That's quick and sketchy, but you get the idea. Here's the actual logic:

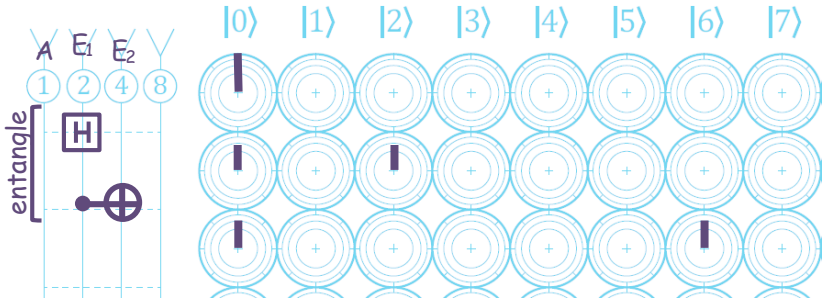


Wow! There's nothing there that we haven't covered already, and we only need three qubits, not all four. That means we can just use 8-column circle paper. There's no harm in using all 16 columns, but half of them will be blank.

So let's do it.

Step 1: Make an entangled pair

(way back in the past) Alice and Bob entangle two qubits E_1 and E_2, and they each keep one.

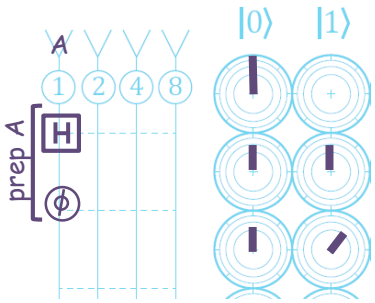


That's it, they can take E_1 and E_2 and lock them away, ready to use when Alice has something she wants to teleport. Can you tell E_1 and E_2 are entangled by looking at the circles? Sure. If we read them, E_1 and E_2 will always have the same value. The only two choices are 0, where they're both 0, or 6, where those two bits are both 1.

Notice we haven't touched A at all yet.

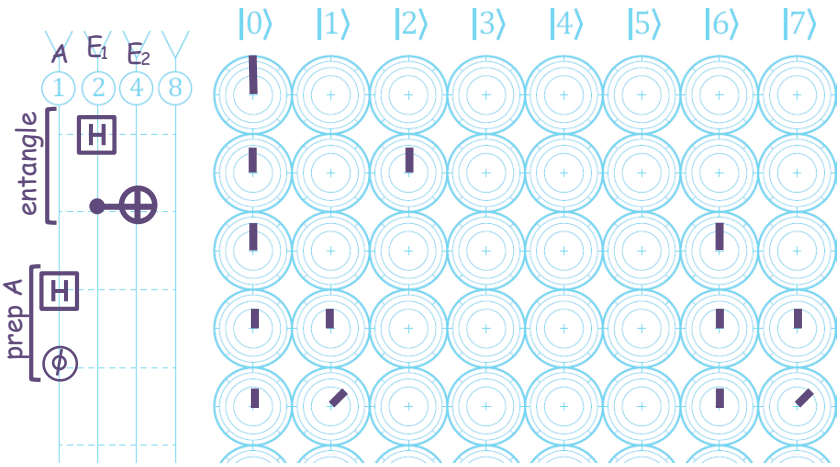
Step 2: Alice preps her payload

What's she going to send? Let's say it's a qubit with a secret phase angle. She sits in her lab, and does a Hadamard and a 45-degree phase shift. In the lab, just one qubit on its own, that looks like this:



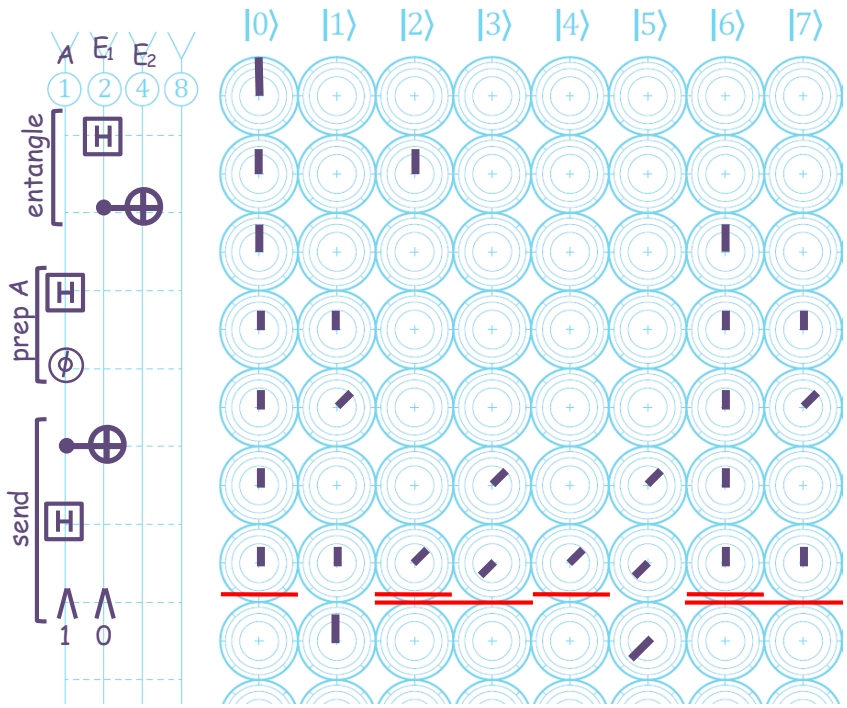
…so that's the package she's planning to send. **But** it's actually part of a larger system. There are three qubits in the story, and it doesn't matter that they haven't met yet, the fact that they're *going to meet* matters. Sort of like people in love, who maybe were always in love but just hadn't met yet. Not everything has to be time-forward to make sense.

So, here's the complete system so far, all on the same page.



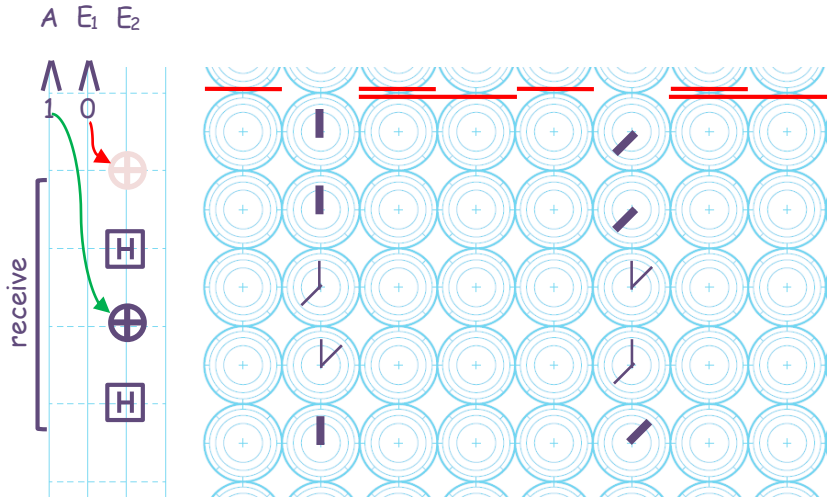
That's kind of interesting… if we read this register, we can only get the binary values (order E_2 E_1A) 000, 001, 110, 111. So each bit has exactly a 50/50 chance of being 1 or 0, but E_1 and E_2 will always be the same value. They're still entangled. A isn't entangles with anything, which makes sense, as we've kept it separate so far.

Now it's time for Alice to send her teleport package.



Note that the distribution before the read is totally even, so we can pick *any* value as our read value. We could choose 0 for both, which would be totally valid, but I decided to flip a coin. It came up tails and then heads, so I picked 1 and 0.

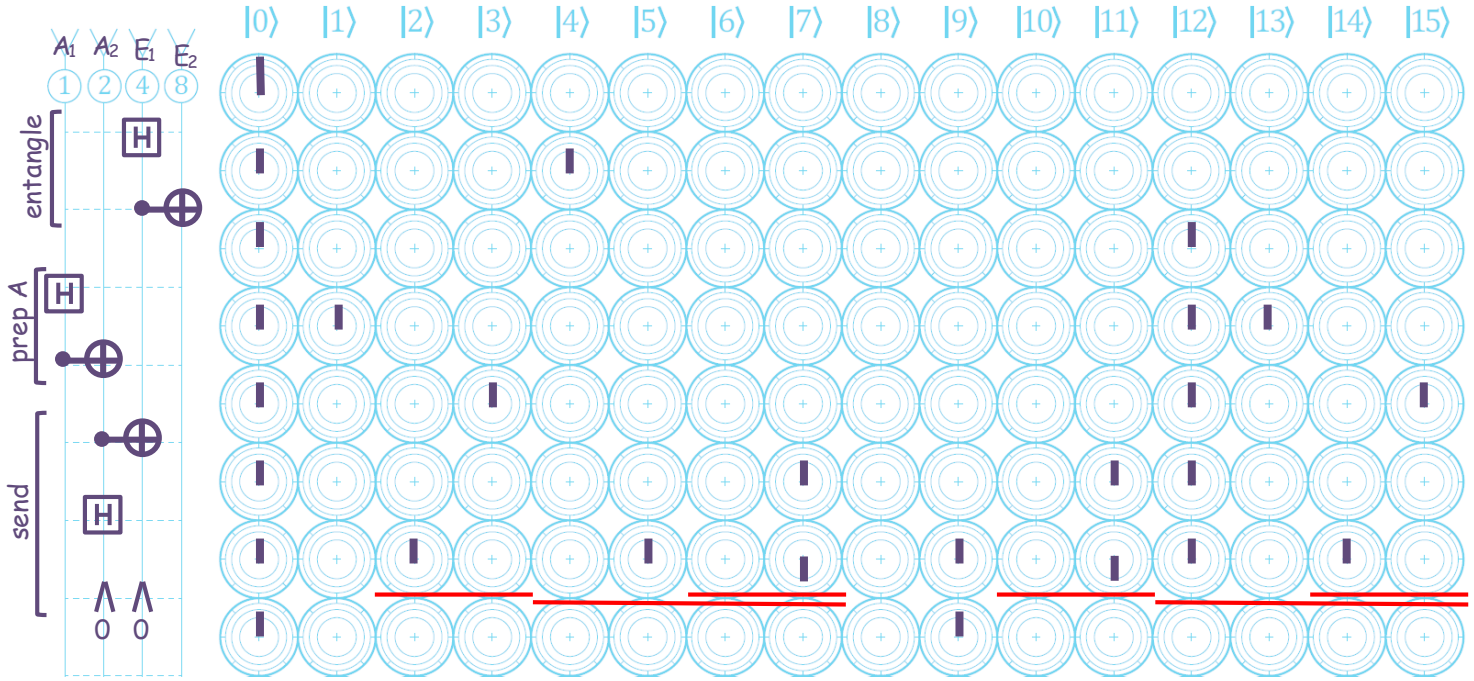
So Alice has destroyed her qubits. She can write down "1 and 0" on a piece of paper and send it to Bob. He already has E_2 kept safe, so when he gets her note, he can do the receive operation. This consists of two Hadamards, and two NOT gates which only get applied is the corresponding bit from Alice is 1.



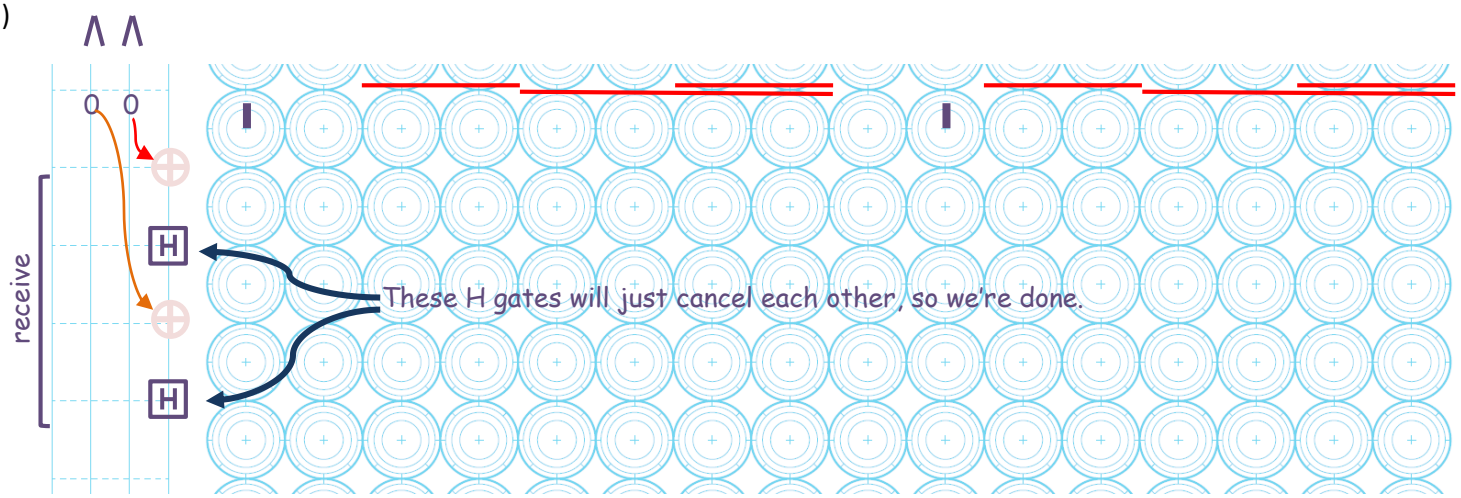
…and SUCCESS! Bob's qubit (E_2) is now an *exact* recreation of Alice's initial qubit. Even the phase transferred perfectly. **Teleportation system: activated.** Notice that I used the double-vector separation trick to avoid having to monkey with precise vector angles. Since we're not doing anything but NOT and H in this section, we're not worried about new phase components creeping in, and the dual-vector circles end up canceling out anyway.

Fun Test: Teleporting With Pre-Entanglement

Since we have four qubits on the page, let's use them all, to see what happens when Alice sends Bob a qubit which is ***already*** entangled with another qubit in her lab. That seems useful, if it works. Here's the entire thing, and I'll leave it to you to read through it. In this case, she just entangles A_1 and A_2, and then teleports A_2 to Bob. If it works, he should receive a qubit which is *still entangled* with A_1.



(I flipped a coin again. This time it was heads both times, so 0 and 0. That means Bob shouldn't apply either of his NOT gates.)

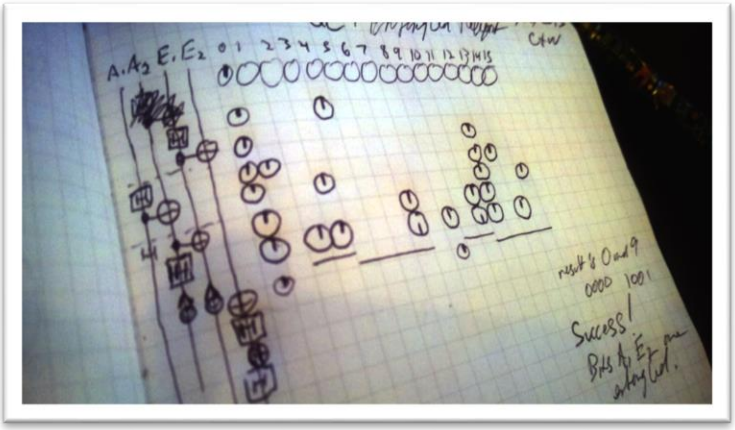


…so did it work? Looks like yes. Bob will be thrilled.

Conclusion

When it comes to understanding QC operations intuitively, I find circle paper very useful. If you do as well, please feel free to use it (print it from this link), and send me a note to let me know.

One final note, if actual circle paper is not available (or your favorite notebook is just quad-rule), you can of course do everything mentioned above pretty easily. It ends up feeling a bit like you're working on a challenging and fun logic puzzle.



Useful References

- Minds, Machines, and the Multiverse, by Julian Brown
- Elementary Gates for Quantum Computation (1995) arXiv:quant-ph/9503016v1
- Quantum Computing for Computer Scientists (Yanofsky and Mannucci), 2008
- IBM QC advances 2/28/2012:
    - http://ibmquantumcomputing.tumblr.com/
    - http://www.nytimes.com/2012/02/28/technology/ibm-inch-closer-on-quantum-computer.html
- [TODO: add other papers and books I've found useful]

About the Author

EJ and his muse live in a secret laboratory in San Francisco. Everything else you really need to know is either posted here, or can be obtained by sending an email to ej@machinelevel.com.