IBM Quantum Experience Documentation

If quantum physics sounds challenging to you, you are not alone. All of our intuitions are based on day-to-day experiences and are defined by classical physics — so most of us find the concepts in quantum physics counterintuitive at first. In order to comprehend the quantum world, you must let go of your beliefs about our physical world, and develop an intuition for a completely different (and often surprising) set of laws.

Our goal with the IBM Quantum Experience is to introduce this world through a set of short tutorials in our User Guide, and by providing the hands-on opportunity to experiment with operations on a real quantum computing processor. This way, we hope to foster a quantum intuition in the greater community, and spark further interest in those who are curious. By making quantum concepts more widely understood—even on a general level— we can more deeply explore all the possibilities quantum computing offers, and more rapidly bring its exciting power to a world whose perspective is limited by classical physics.

> Prepare yourself for a wild and fascinating journey -- and it all starts with a qubit.
>
> – Jay Gambetta and Jerry Chow

> Get ready to think outside a box you didn't know existed.
>
> – Charlie Bennett

Beginners Guide

> Prepare yourself for a wild and fascinating journey -- and it all starts with a qubit.
>
> – Jay Gambetta and Jerry Chow

> Get ready to think outside a box you didn't know existed.
>
> – Charlie Bennett

FAQ for Beginners

`What does "quantum" mean?`

Quantum theory, developed in the early 1900's, revolutionized physics and chemistry by successfully explaining the weird behavior of tiny particles like atoms and electrons. In the late twentieth century it was discovered that it applied not just to these particles, but to information itself. This led to a revolution in the science and technology of information processing, making possible previously unimagined kinds of computing and communication.

`What is a quantum computer?`

A quantum computer is a device able to manipulate delicate quantum states in a controlled fashion, the way an ordinary computer manipulates its bits.

`What is a qubit?`

A qubit is the quantum version of a bit, and its quantum state can take values of |0>, |1>, or **both at once**, a phenomenon known as **superposition**. The half angle bracket notation |> is conventionally used to distinguish qubits from ordinary bits.

`What is a superposition?`

A superposition is a weighted sum or difference of two or more states; for example, the state of the air when two or more musical tones are sounding at once. Ordinary, or "classical," superpositions commonly occur in everyday phenomena involving waves.

`How are quantum superpositions different?`

Quantum theory predicts that a computer with N qubits can exist in a superposition of all 2^N of its distinct logical states |00...0> through |11...1>. This is exponentially more than a classical superposition. Playing N musical tones at once can only produce a superposition of N states.

`How is superposition different from probability?`

A row of N coins, each of which might be heads or tails, has 2^N **possible**states, but it actually **is** in only one of them—we just don't know which. For this reason, quantum superposition is a more powerful resource than classical probabilism.

`How is a quantum superposition different from massive parallelism?`

Though superposition is stronger than a probabilism, it is weaker than actually having an army of 2^N real computers all working on the problem at once. Superposition is strictly weaker than full parallelism, and strictly stronger than probabilism.

`What is entanglement?`

Entanglement is a property of most quantum superpositions and does not occur in classical superpositions. In an entangled state, the whole system is in a definite state, even though the parts are not. Observing one of two entangled particles makes it behave randomly, but tells the observer exactly how the other particle would act if a similar observation were made on it. Because entanglement involves a correlation between individually random behaviors of the two particles, it cannot be used to send a message.

Therefore, the term “instantaneous action at a distance,” sometimes used to describe entanglement, is a misnomer. There is no **action**, only **correlation**, which, though uncannily perfect, can only be detected afterward when the two observers compare notes. The ability of quantum computers to exist in entangled states is responsible for much of their extra computing power, as well as many other feats of quantum information processing that cannot be performed, or even described, classically.

```
What is a quantum gate?
```

Quantum gates are the elementary building blocks for quantum computation, acting on qubits the way classical logic gates act on bits, one and two at a time, to change their state in a controllable way.

Introduction

We’re at the start of a new stage in the information revolution. The first stage began around 1950 with a handful of expensive room-sized computers, used only by specialists. Today there are more computers in the world than people, and we rely on them for everything from communication to transportation, commerce, and (of course!) the Internet. All of our computers’ varied abilities are produced by manipulating zeros and ones using simple operations like **AND**, **OR**, and **NOT**, which are called *logic gates*. By doing so billions of times per second in billions of places at once, they keep our world humming along in the manner to which we have become accustomed.
For over 35 years, IBM has been researching an utterly different kind of information and information processing, as different from ordinary “classical” information as a dream is from a book. Unlike dreams, this new kind of information, called *quantum information*, is both well-understood and useful. The basic unit of quantum information is called a **qubit** (pronounced CUE-bit), and a machine for storing and processing qubits is called a **quantum computer**. We’ve been building and testing increasingly powerful quantum computers for several years, and last year we made a 5-qubit one, housed at our Yorktown lab, available over the Internet to the general public. In other words, YOU now have a programmable quantum computer at your fingertips! We’ll soon be upgrading our public quantum computer, but even five qubits are enough to get a feel for quantum computing.
Quantum theory, developed in the early 1900’s, revolutionized physics and chemistry by successfully explaining the weird behavior of tiny particles like atoms and electrons. In the late twentieth century it was discovered that it applied not just to these particles, but to information itself. This led to a revolution in the science and technology of information processing, opening the door to new types of computing and communication.
By going through this Beginner’s Guide, we hope you will learn what’s different about quantum computing, and the new possibilities that it opens up. Some of these might include designing new materials and drugs, searching databases faster, and solving systems of linear equations with breathtaking efficiency, in ways that are currently impossible. To do all of this, quantum computers will use two fundamental properties of the quantum world: **superposition** and **entanglement**.
So, what is **superposition**? Qubits can be in the “$|0\rangle$” state (called a zero-ket), the “$|1\rangle$” state (called the one-ket), or a linear combination of the two (superposition). The half-angle bracket notation $|\rangle$ is conventionally used to indicate qubits, as opposed to ordinary bits. When you measure the $|0\rangle$ quantum state, you get a classical 0, and when you measure the $|1\rangle$ quantum state, you get a classical 1. The $|0\rangle$ state is sometimes called the ground state because in many physical implementations of quantum computing, including ours, it is the lowest energy state.

Now, for **entanglement**. Entanglement is a property of many quantum superpositions and does not have a classical analog. In an entangled state, the whole system can be described definitively, even though the parts cannot. Observing one of two entangled qubits causes it to behave randomly, but tells the observer exactly how the other qubit would act if observed in a similar manner. Entanglement involves a correlation between individually random behaviors of the two qubits, so it cannot be used to send a message. Some people call it “instantaneous action at a distance,” but this is a misnomer. There is no **action,** but rather **correlation**; the correlation between the two qubits’ outcomes is detected only after the two measurements when the observations are compared. The ability of quantum computers to exist in entangled states is responsible for much of their extra computing power, as well as many other feats of quantum information processing that cannot be performed, or even described, classically.

To learn more about the concepts of superposition and entanglement, see (https://www.youtube.com/watch?v=Hi0BzqV_b44) actor Paul Rudd defeat Stephen Hawking in a game of Quantum Chess. The rules can be found online (https://www.youtube.com/watch?v=jJoDKHKE2gA), and you can watch (https://www.youtube.com/watch?v=LikdmXfWO2A&t=24s) the inventor of the game, Chris Cantwell, play a friendly game with chess expert Anna Rudolf. You can also find both a short and longer introduction to quantum concepts by IBM Fellow Charlie Bennett (http://www.research.ibm.com/quantum/expertise.html) on our website.

If you want to see more of the math and theory behind the concepts, we encourage you to dig into our full User Guide!

We anticipate that you will spend about 10-30 min on each section (with the later sections requiring more time) as you read through this guide and explore the sample quantum scores on the IBM Q experience.

Getting Started

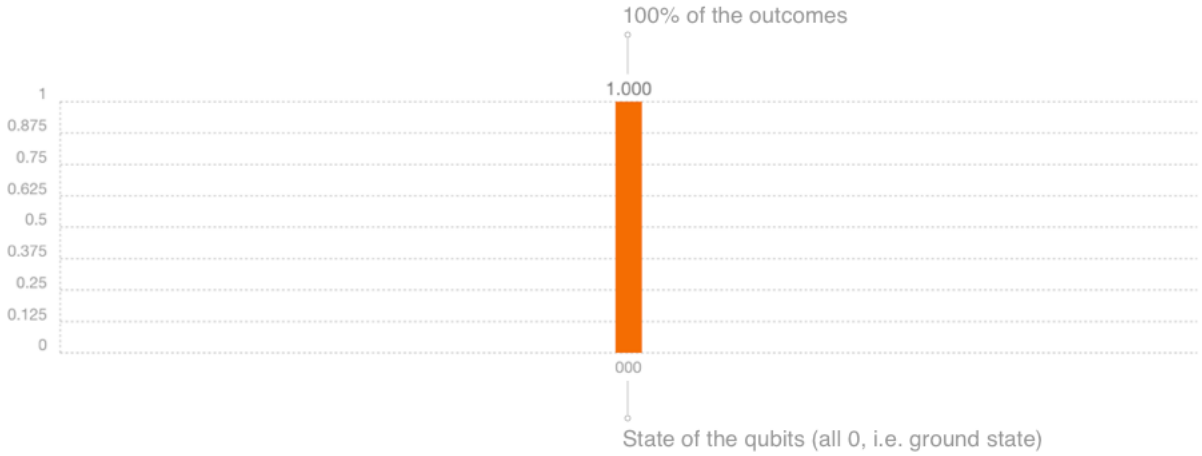
Histogram representation (Bar graph)

In the histogram/bar graph representation, the combination of 0’s and 1’s at the bottom of each bar represents the measured state of the qubit(s), and the height of the bar represents how frequently the outcome occured in the different runs of the experiment. Note that the more qubits you use, the more 0’s and 1’s it takes to represent the outcome(s). To save space, outcomes that did not occur are omitted from the histogram, and several low frequency outcomes may be combined into a bar labeled “other values.”

To try this out for yourself, go into the Composer and create a new experiment with three qubits. Drag a pink measurement gate over to each of them and click “Run” or “Simulate.” At this point, there should be only one possible outcome, 000. That’s because all of the qubits began in the $|0\rangle$ state and we haven’t changed them with any gates. Now, drag a blue *H* gate to each of the qubits before the measurement. The *H* gate puts each of the qubits into an equal superposition state. Now you should see that there are many more possible outcomes. We’ll go into more detail about the *H* gate in a later section.

3-qubit measurement, ground state

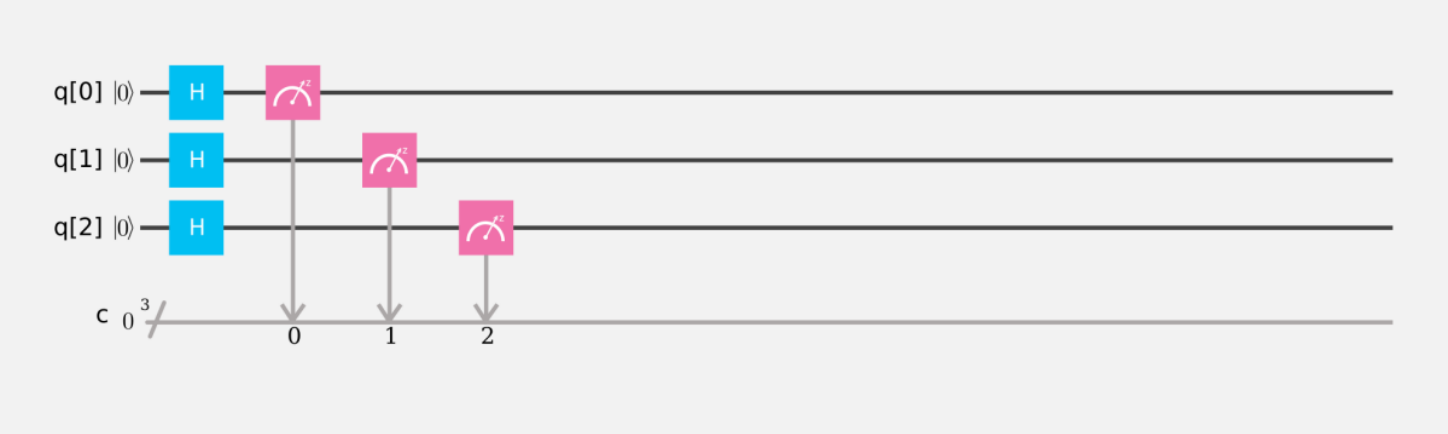
(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=f45e53d09688085a2363c7fc60406e9c&sharedCode=true) [Open in composer]
(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=f45e53d09688085a2363c7fc60406e9c&sharedCode=true)



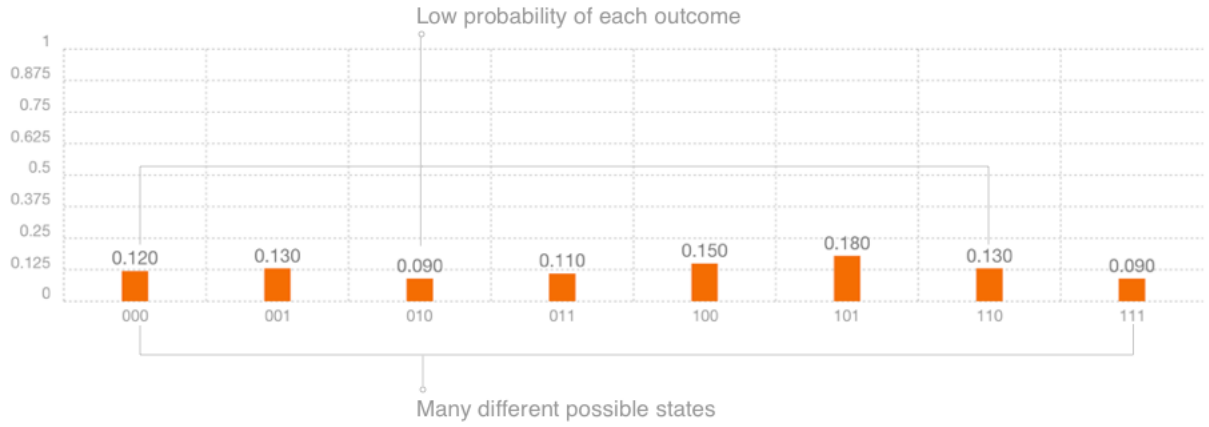
QISKit SDK python (https://qiskit.org) example

```
# use QISKit.org
from qiskit import QuantumProgram

# useful additional packages
from tools.visualization import plot_histogram

# Define the QProgram and the Quantum and Classical Registers
qp = QuantumProgram()
q = qp.create_quantum_register("q", 3)
c = qp.create_classical_register("c", 3)

# Define the circuit
threeQ = qp.create_circuit("threeQ", [q], [c])
threeQ.measure(q[0], c[0])
threeQ.measure(q[1], c[1])
threeQ.measure(q[2], c[2])

# Execute the circuit
result = qp.execute(["threeQ"])

# Plot result
plot_histogram(result.get_counts("threeQ"))
```

3-qubit measurement, full superposition of states



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=db03d34bf6ad5c4f27777fc3c8adb769&sharedCode=true) [Open in composer]
(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=db03d34bf6ad5c4f27777fc3c8adb769&sharedCode=true)



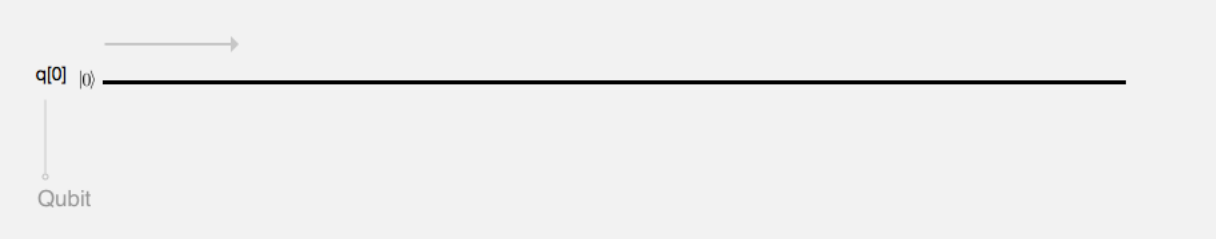
QISKit SDK python (https://qiskit.org) example

```
# use QISKit.org
from qiskit import QuantumProgram

# useful additional packages
from tools.visualization import plot_histogram

# Define the QProgram and the Quantum and Classical Registers
qp = QuantumProgram()
q = qp.create_quantum_register("q", 3)
c = qp.create_classical_register("c", 3)

# Define the circuit
threeQ = qp.create_circuit("threeQ", [q], [c])

threeQ.h(q[0])
threeQ.h(q[1])
threeQ.h(q[2])

threeQ.measure(q[0], c[0])
threeQ.measure(q[1], c[1])
threeQ.measure(q[2], c[2])

# Execute the circuit
result = qp.execute(["threeQ"])

# Plot result
plot_histogram(result.get_counts("threeQ"))
```

The **Quantum Composer** is a graphical user interface for programming a quantum processor. Think of it as a tool to construct quantum algorithms using a library of well-defined measurements and gates (operations that change the state of the qubit).

Throughout this guide, you'll try out many different experiments (feel free to explore on your own as well). When you first click on the "Composer" tab, you'll name your experiment and choose between running a *real* quantum processor or a *custom* quantum processor (simulator). If you choose custom, you'll select the number of qubits in the experiment and the number of classical bits in the classical bit register (you can keep this the same as the number of qubits). In the real processor, the possible connectivity between the qubits is limited by the experimental setup; there are also some errors in the measurement due to experimental imperfections. In the custom processor, however, quantum gates can be placed anywhere. While there are no experimental errors, you will still observe that random outcomes can occur due to the nature of quantum information. In this guide, we will show the results of running experiments on the custom processor (to avoid confusion about deviations that occur due to experimental errors). We encourage you to run some experiments on both the custom and real processors so that you can understand the differences.

The Composer enables you to create a quantum score – not a score as in a sporting match, but rather in the musical sense. In a quantum score, just as with music, time progresses from left to right. Each line represents a qubit (as well as what happens to that qubit over time). Just as with musical notes, each qubit has a different frequency. A quantum algorithm (circuit) begins by preparing the qubits in well-defined states (for example, "$|0\rangle$" in the picture below), then executing a series of single- and two-qubit gates in time from left to right.



Quantum gates are represented by square boxes; they play a frequency for different durations, amplitudes, and phases. These are called single-qubit gates. To apply a gate to a qubit, simply drag the gate box into the qubit staff. To delete, simply double-tap the box or drag it to the trash bin.

Once you've populated the staff with all of the gates and measurements you'd like, click "Run" (only available for the real processor) or "Simulate" to generate results for your experiment. Each circuit must end with a measurement gate in order to run the experiment.

Single Qubit Measurement:

In the above example, we created a single-qubit score with one classical bit in the classical bit register. We measured Qubit "0" and stored the measurement result in the 0th position of the classical bit register (this is the line below the score labeled 'c').

After performing a quantum measurement, a qubit's information becomes a classical bit, meaning it loses the quantum properties of superposition and entanglement. Each qubit in the measurement either takes the value 0, e.g., if the qubit is measured in state $|0\rangle$; or 1, e.g., if the qubit is measured in state $|1\rangle$. Sometimes your qubit has an equal chance of being a $|0\rangle$ or a $|1\rangle$, such as when it's in an equal superposition state. In those cases, when you repeat the experiment many times on the real device (what we call "shots" in the pull-down menu of Simulate, e.g., 1024 times), you'll find that about half the time you measure 0 and half the time you measure 1.

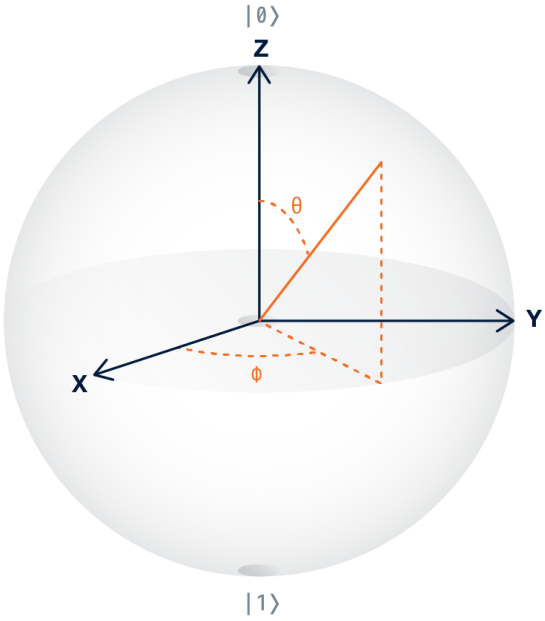
In the IBM Q experience, the results of your quantum scores are shown in a standard histogram/bar graph representation.

The Weird and Wonderful World of the Qubit

A qubit is a quantum system consisting of two energy levels, labeled $|0\rangle$ and $|1\rangle$. The $|0\rangle$ state is often called the ground state because it is the lower of the two energies. Together, $|0\rangle$ and $|1\rangle$ make up what we call "standard basis vectors". Like all vectors, they point in a direction and have a magnitude. Defining basis vectors is a really useful trick we have borrowed from linear algebra. The basic idea is that once you have defined these vectors, you can construct any other vector from a linear combination of the basis vectors.

Additionally, qubits also have a "phase", which results from the fact that superpositions can be complex. To represent these superpositions, we put a coefficient such as a or b in front of the state like so: $a|0\rangle + b|1\rangle$. Here's what the formula is saying: "The state is made up of a linear combination of $|0\rangle$ and $|1\rangle$, where the proportion of each depends on the coefficients a and b." The coefficients a and b could be positive, negative, or even complex. If we take the absolute value of a or b and square it (e.g. $|a|^2 `or : math : `|b|^2$), we get the probability of measuring the **0** or **1** outcome, respectively.

The basis states $|0\rangle$ and $|1\rangle$ and their linear combinations $a|0\rangle + b|1\rangle$ describe the state of a single qubit. But because the coefficients a and b are not just real numbers, but can be imaginary or even complex, visualizing a qubit requires a special tool called the Bloch Sphere. The Bloch Sphere is a sphere with a radius of one and a point on its surface represents the state of a qubit. Like a globe uses longitude and latitude to describe points on the surface, the Bloch sphere can also use angles to describe the state of a qubit. This representation allows any qubit state, including those with complex coefficients, to be represented as a point on the surface of the Bloch sphere. Points on the surface of the Bloch sphere which lie along the X, Y, or Z axis correspond to special states as described below.



The qubit state is shown by the orange line in the picture. In the picture, the state at the top of the sphere represents $|0\rangle$ and the state at the bottom of the sphere represents $|1\rangle$.

When the qubit is in a superposition of $|0\rangle$ and $|1\rangle$, the vector will point somewhere between the two on the sphere (i.e. the angle θ angle becomes somewhere between 0 (pointing straight up) – and 180 degrees, or π radians (pointing straight down). Note that in radians, 1/4 of a turn is represented by $\pi/2$).

We have another degree of freedom on the sphere: rotations around the Z axis, which can be described by the angle ϕ. When ϕ is non-zero, this indicates a change in the *phase* of the qubit. This Bloch Sphere depiction works only for single-qubit representations. For the purpose of the schematic, we'll assume that the length of the Bloch vector is equal to the radius of the Bloch sphere.

Single-Qubit Gates

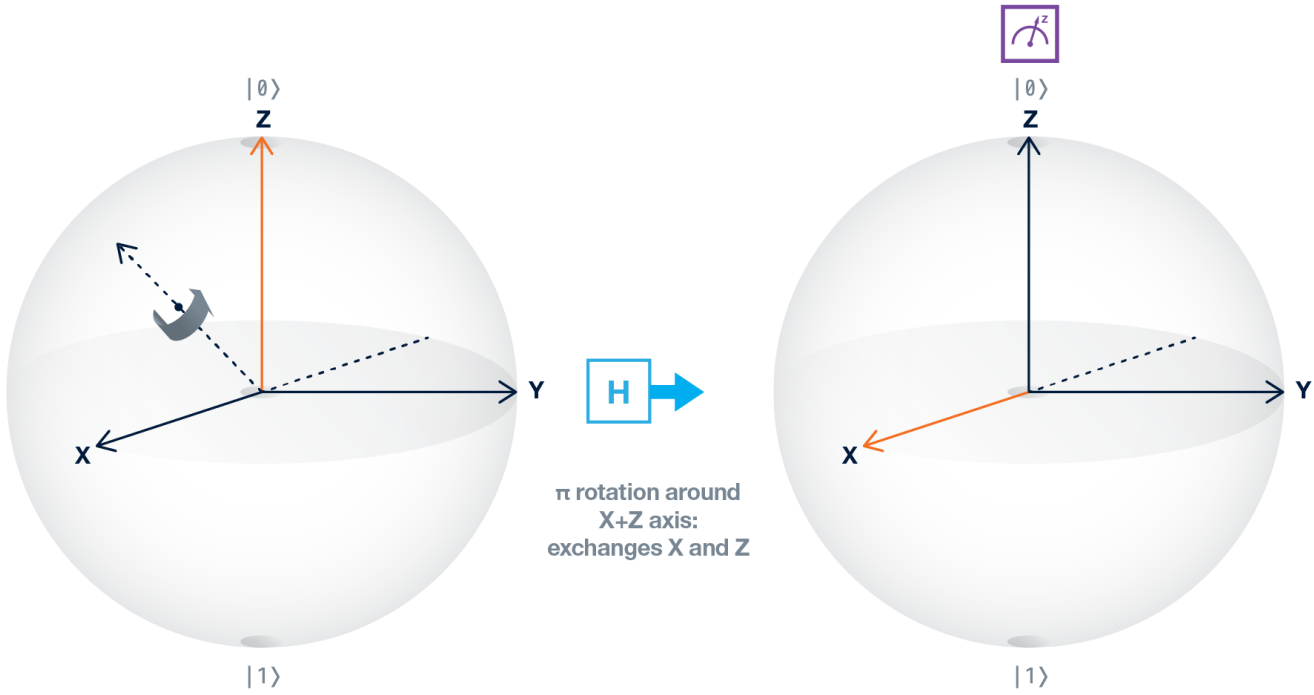
Creating superposition

Now that we know how to switch between $|0\rangle$ and $|1\rangle$, let's explore superposition, which is the concept of generating a new quantum state which is a combination of the basis states $|0\rangle$ and $|1\rangle$. To make superposition states, we will expand our set of gates to include *H* (which you saw earlier in the guide). In the Quantum Composer, this is the blue gate labeled *H*.

Place the *H* gate, known as the Hadamard gate, on one of the qubits (which starts in the $|0\rangle$ state) and run the standard measurement. Did you find that the qubit behaves half the time like a $|0\rangle$ and half the time like a $|1\rangle$? Before the measurement forced the qubit to choose a final state, the qubit's state was neither $|0\rangle$ nor $|1\rangle$ but rather a uniquely quantum state, a superposition, consisting of an equal-weighted combination of these two states.
The special case where the *H* gate is applied to the $|0\rangle$ state is given its own symbol and definition: $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. This is a fancy way of saying that this new state that we call "$|+\rangle$" has a 1/2 probability of giving the outcome 0 and 1/2 probability of giving the outcome 1 (remember how the coefficient squared gives the probability of the outcome?). You can think of the *H* gate as a rotation around the X+Z axis, shown below with a dotted line. This is the standard representation of a superposition state, and **points along +X on the Bloch sphere**.
$|+\rangle$ superposition state, the standard representation of a superposition:

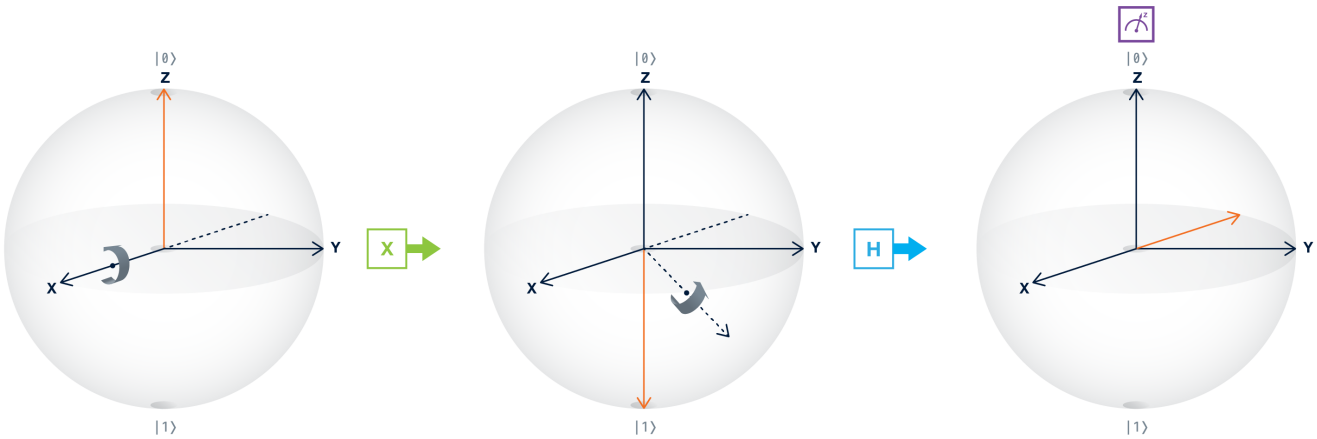
(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=3033b263493f8f4eacad676036b70f22&sharedCode=true)
Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=3033b263493f8f4eacad676036b70f22&sharedCode=true)

Below is a sample histogram showing the results of running the above circuit 100 times. Although on average we expect this circuit to yield 0 and 1 with equal probabilities, any finite set of trials is unlikely to produce this result exactly, just as 100 fair coin tosses generally won't yield exactly 50 heads and 50 tails.



Together with the state $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$, which is **a vector pointing along –X on the Bloch Sphere**, we can define a new basis (or measurement direction) called the superposition basis. The $|-\rangle$ state is made using the circuit below. The *X* gate flips the $|0\rangle$ to a $|1\rangle$, and then the *H* gate rotates the qubit around the X+Z direction to produce the $|-\rangle$ state. When you run this circuit you will find that, like before, the outcomes are equal. Different states give the same outcomes!

$|-\rangle$ superposition state



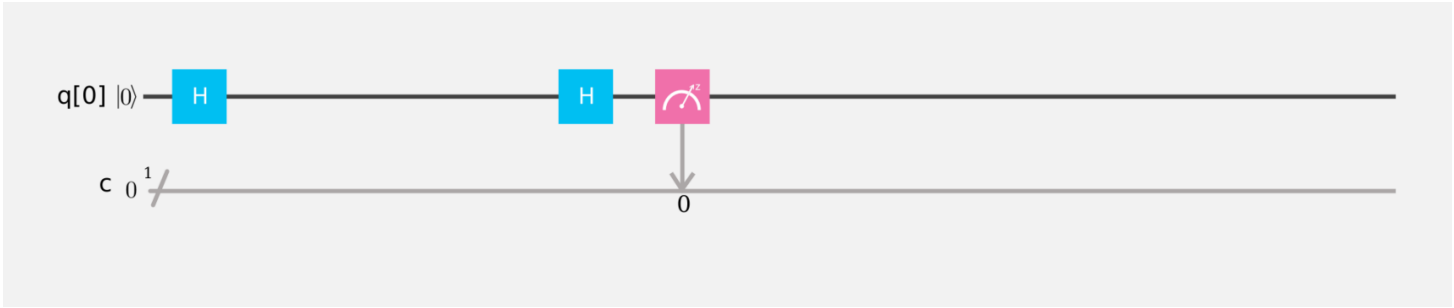
(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=69105edada81f30718a6872b1133ee2d&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=69105edada81f30718a6872b1133ee2d&sharedCode=true)

When you measure along the standard basis Z (which is the only direction we can access with our pink Z measurement gate), we are not able to access information about the qubit's phase.

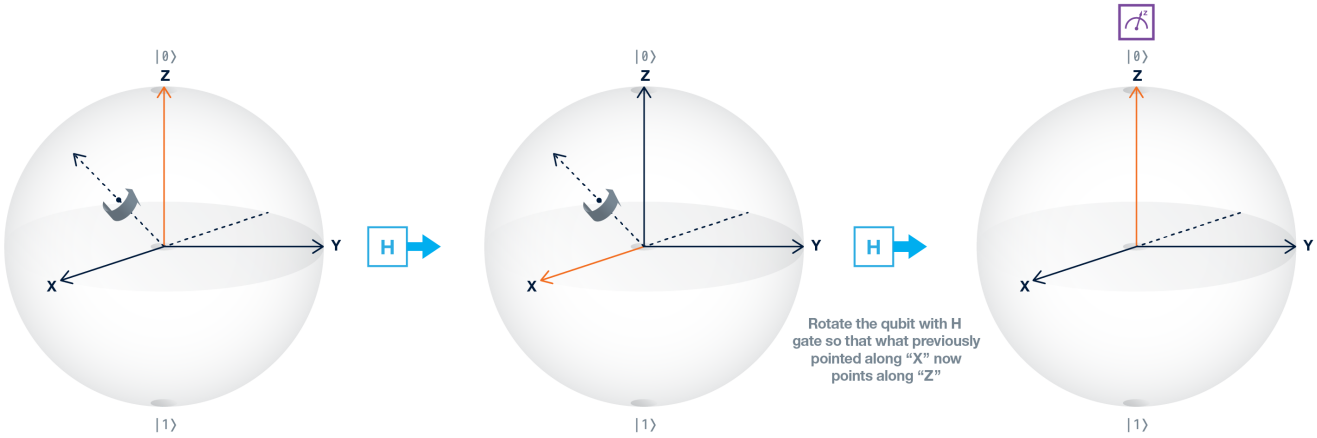
To tell the difference between these states, $|+\rangle$ and $|-\rangle$, we need to *measure* in the superposition basis. Experimentally, we cannot physically measure along the different directions of the Bloch Sphere; however, we can make it look like we're changing the measurement by rotating our qubit state using gates before performing the standard measurement (which can only measure along "+Z"). To measure in the X basis, we rotate the qubit's state until the qubit's component that previously pointed along X points now in the "+Z" direction, which is accomplished by applying a Hadamard gate before the measurement.

$|+\rangle$ state measured in X basis:



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=61eed74ff0b30d2476bdac5d454f0024&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=61eed74ff0b30d2476bdac5d454f0024&sharedCode=true)



$|-\rangle$ state measured in X basis:

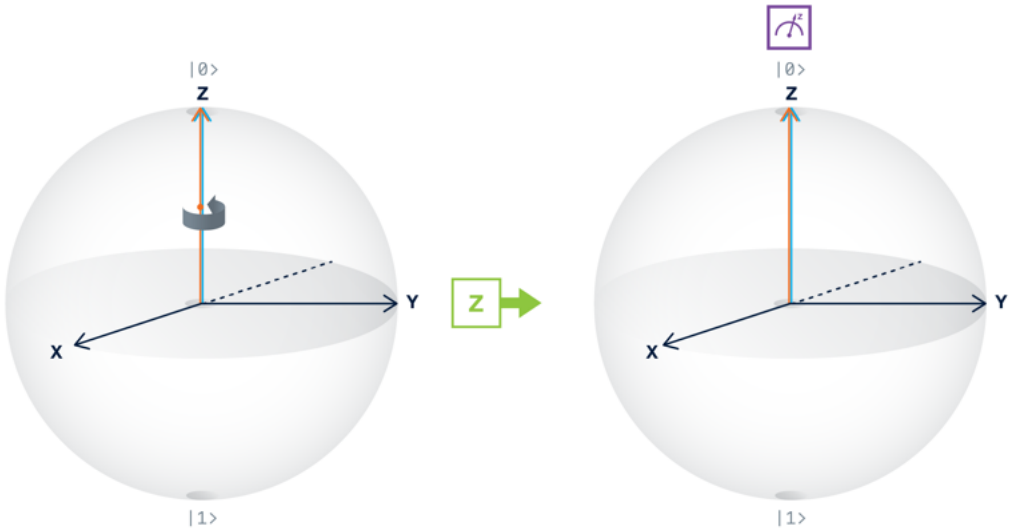
(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=b8c9dbdafa24ade6bc96be2916487493&sharedCode=true)
Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=b8c9dbdafa24ade6bc96be2916487493&sharedCode=true)
Try the above measurements of the superposition $|+\rangle$ and $|-\rangle$ states in the X basis. You should find that 100% of the time the outcome is 0 and 1 respectively. That is, if we make a measurement in the standard (“Z”) basis, the outcome is completely random. But, in the X basis, it has a deterministic outcome!

Introducing qubit phase

Now that we have seen how to make $|0\rangle$, $|1\rangle$, and superposition states, let’s investigate how we can change the phase of the superposition. We have added the following gates: Z, S, $S^\dagger$, T, and $T^\dagger$.

The *:math:`Z`* gate is a π rotation around the Z axis, the S gate is a rotation of $\pi/2$ around Z, and the T gate is a $\pi/4$ rotation around Z. $S^\dagger$ is the inverse of S (does a $-\pi/2$ around Z; $SS^\dagger$ returns the original state), and $T^\dagger$ is the inverse of T. These rotations give the qubit a component along the Y direction of the Bloch sphere, which is a representation of the complex information in the qubit state.

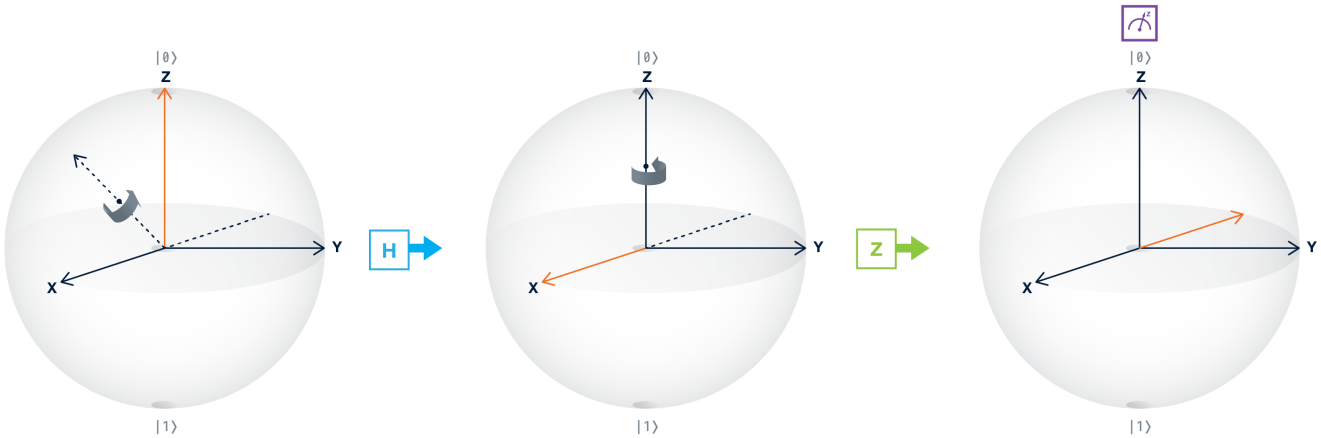
When the qubit is in the |0> state, Z doesn’t do much. But when the qubit is in the $|+\rangle$ state, you can see that Z flips the state from $|+\rangle$ to $|-\rangle$.



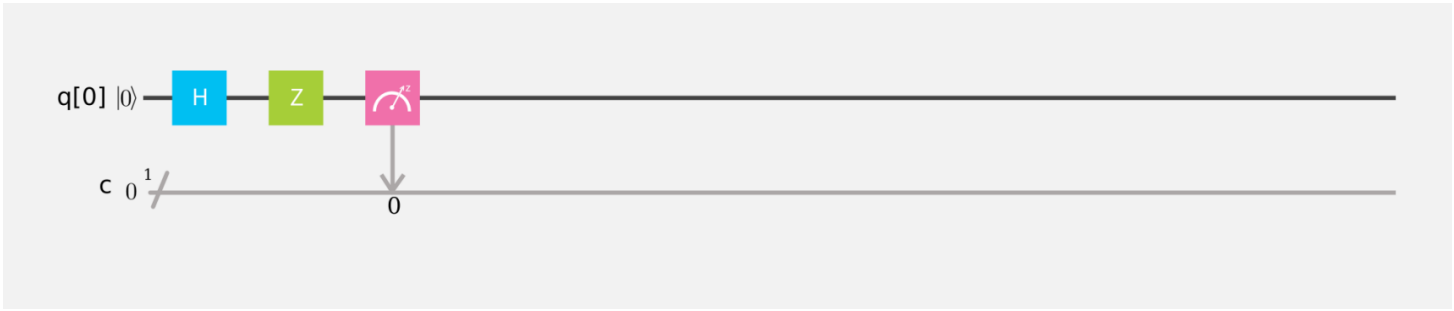
Z gate: when qubit is in the |0> state

(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=95d0c305399b7db58c1c6189b6a333e5&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=95d0c305399b7db58c1c6189b6a333e5&sharedCode=true)



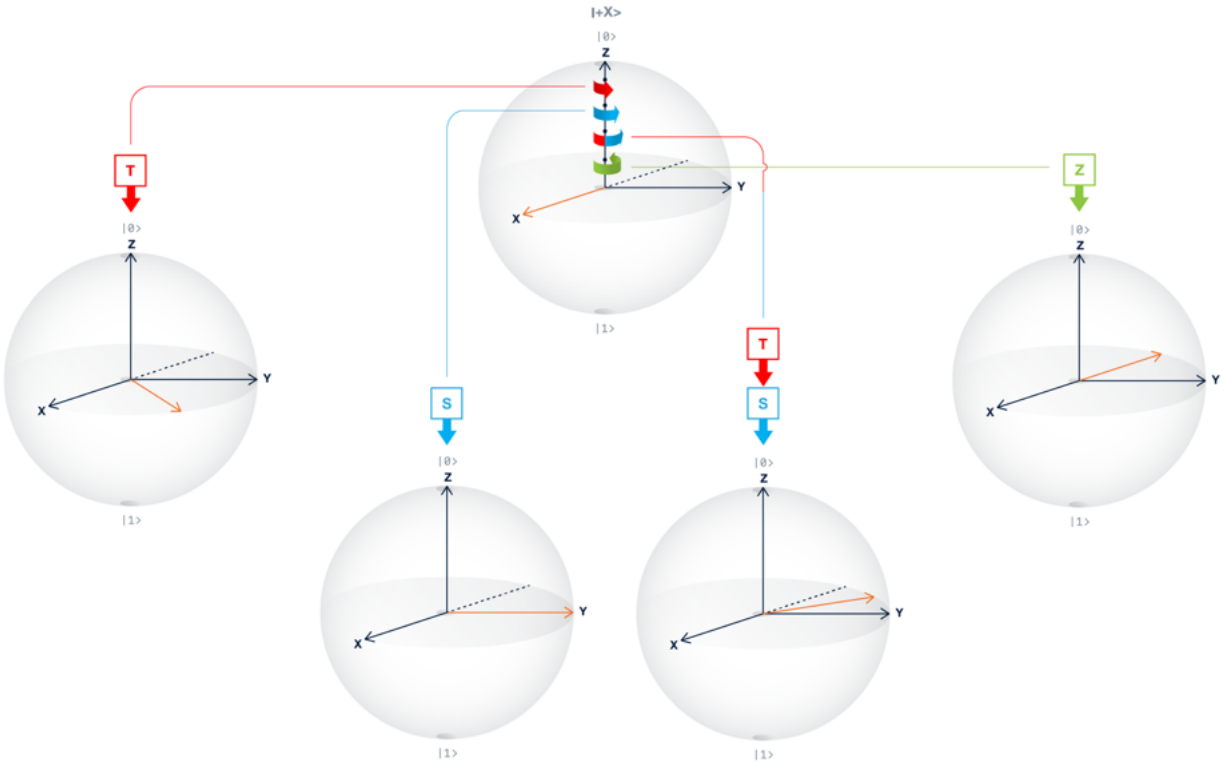
Z gate: when the qubit is in the |+> state



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=95d0c305399b7db58c1c6189b6c3e493&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=95d0c305399b7db58c1c6189b6c3e493&sharedCode=true)

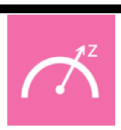
Let's now examine intermediate rotations around the Z axis, starting in the $|+\rangle$ superposition state. Below is a summary of how different rotations around the Z axis impact measurements in the superposition basis.



Gate sequence	Rotation around Z	Probability of 0	Probability of 1
H H	0	1.0	0
H T H	$\pi/4$	0.85	0.15
H S H	$\pi/2$	0.50	0.50
H S T H	$3\pi/4$	0.15	0.85
H Z H	π	0	1

Create +X state | Change qubit phase | Measure in superposition basis

Summary of quantum gates

Table of Quantum Gates and what they do				
State	Gate sequence to prepare the state	Transformation on Bloch sphere	Gates to measure in the respective basis	Name of measurement basis
|0>	(none, ground state)	None		Z, “standard”
|1>		π rotation around X		Z, “standard”
|+>		π rotation around X + Z		X
|->		π rotation around X + π rotation around X + Z		X

Gate	Transformation on Bloch sphere (defined for single qubit)
X	π-rotation around the X axis, Z→−Z. Also referred to as a bit-flip.
Z	π-rotation around the Z axis, X→−X. Also referred to as a phase-flip.
H	maps X→Z, and Z→X. This gate is required to make superpositions.
S	maps X→Y. This gate extends H to make complex superpositions. (π/2 rotation around Z axis).
$S^\dagger$	inverse of S. maps X→−Y. (-π/2 rotation around Z axis).
T	π/4 rotation around Z axis.
$T^\dagger$	-π/4 rotation around Z axis.

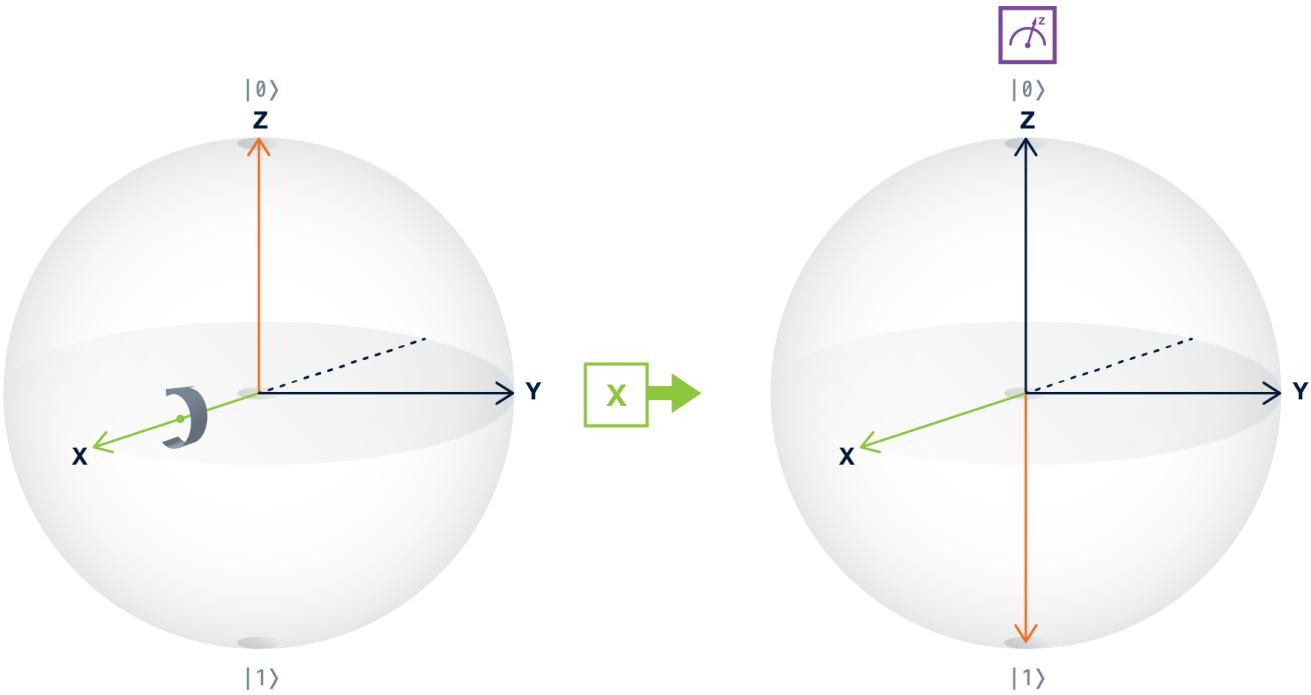
Just as classical computers perform calculations by manipulating bits between the 0 and 1 states, we manipulate qubits to perform calculations on a quantum computer. In this section, we'll show you how to use some important single-qubit gates.

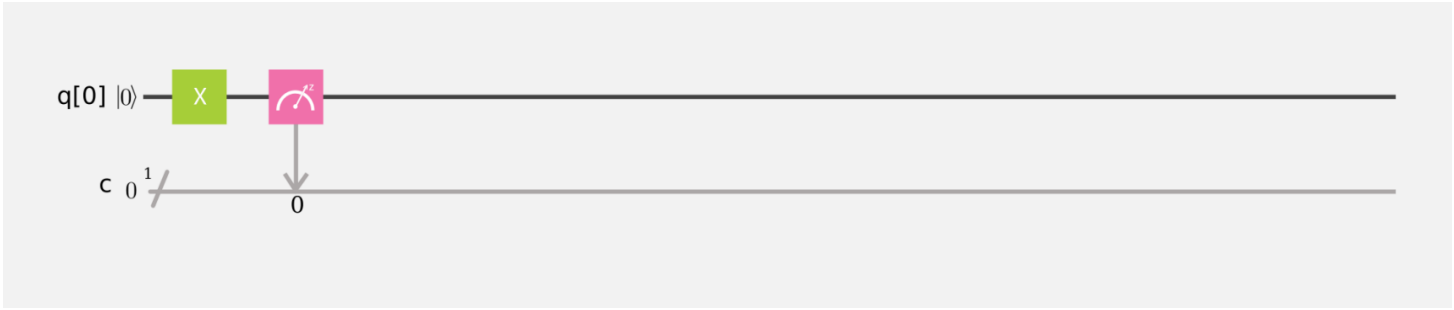
To understand what these do mathematically, go ahead and check out the full User Guide.

X gate

Let's start with the *X* gate, which is known as a "bit-flip", since it flips the 0 to 1 and vice versa. This is similar to a classical NOT gate.

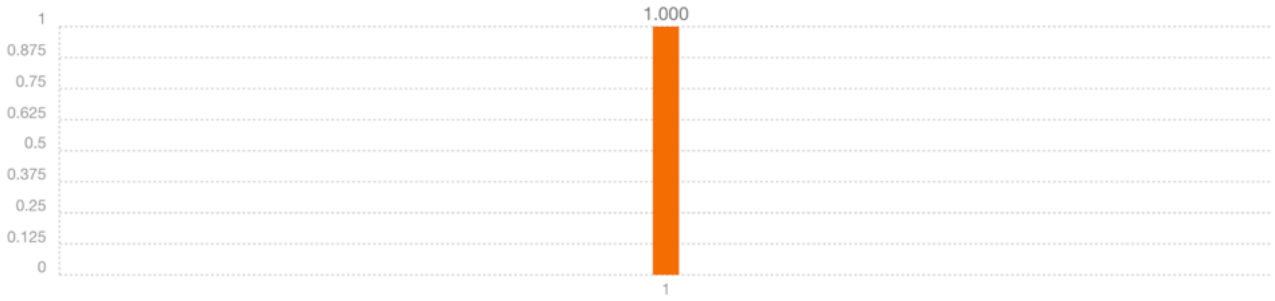
It is also known as an X-rotation, since it rotates the state by π radians around the X-axis. If you start in the $|0\rangle$ state at the top of the Bloch sphere, the X gate rotates you to the bottom of the Bloch sphere ($|1\rangle$). See the below schematic and also try the X gate in the Composer, using the score below.





(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=1a35de9b1f3b236cf98fc3e0960da72e&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=1a35de9b1f3b236cf98fc3e0960da72e&sharedCode=true)

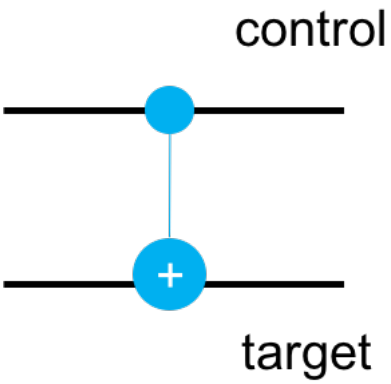


Multi-Qubit Gates

The notation for the state of a machine with multiple qubits is similar to what we have been using, but now there are multiple numbers inside the $|\rangle$ 'ket'. For a two-qubit processor, the qubits can be in four possible states: $|00\rangle$, $|01\rangle$, $|10\rangle$, and $|11\rangle$. Reading from left to right, the first number represents the state of the second qubit and the second number represents the state of the first qubit. That is to say: **the first qubit (q0) is always listed at the far right**. We chose this notation to be consistent with the classical binary representation of numbers. Just like the single qubit, there can also be superpositions of these states like: $\frac{1}{\sqrt{2}}(|00\rangle - |11\rangle)$. When this state is measured, **both** qubits will have the same value, but 50% of the time **both** will be 0 and 50% of the time both will be 1.

To do interesting things and make use of those many configurations in the quantum world, we need gates that perform *conditional* logic between qubits, meaning the state of one qubit depends on the state of another.

The conditional gate we will use is the Controlled-NOT, or CNOT. It is represented by the element:



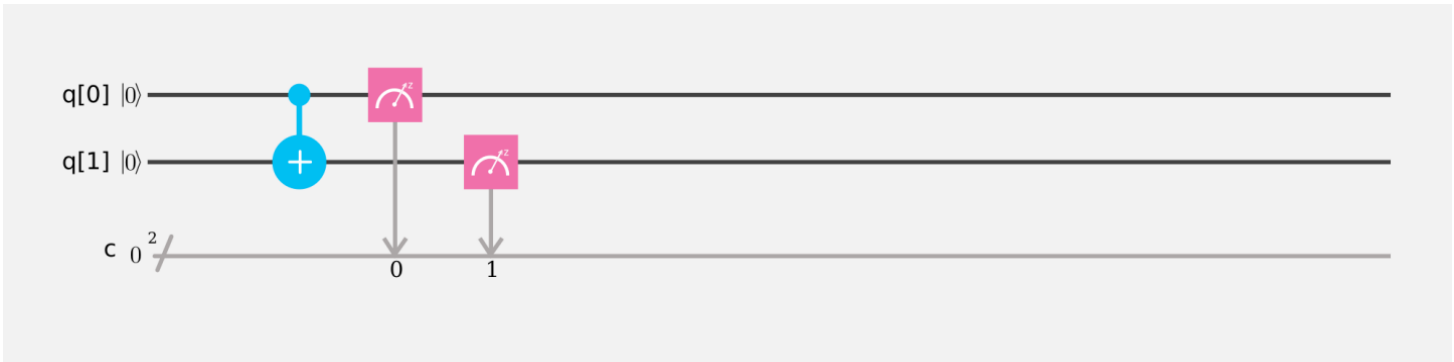
The CNOT gate's action on classical basis states is to flip (apply a NOT or X gate to) the target qubit *only* if the control qubit is $|1\rangle$; otherwise it does nothing.

Below is how the CNOT gate transforms a set of 2 qubits (where **the first qubit – the one on the right – is the control**):

Starting state		Ending State
|00>	→	|00>
|10>	→	|10>
|01>	→	|11>
|11>	→	|01>

Try the "CNOT Circuits" example below with different input states. Drag the CNOT gate to the target qubit and then click on the control qubit to add the link between them. **Note that the X gates have prepared the qubits in a different configuration for each example.**

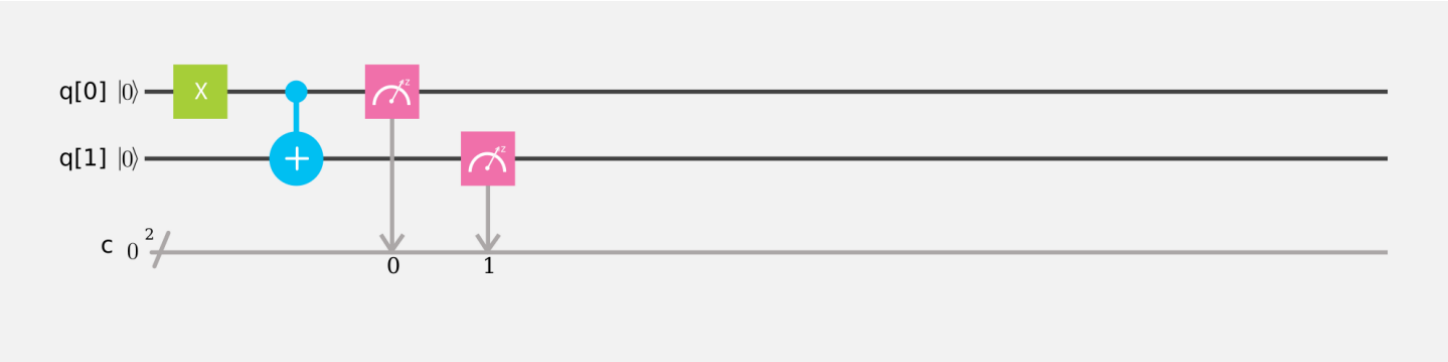
CNOT (input 00)



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=1ae7fa71f3ad1e6f8fbe269fafcc7ca9&sharedCode=true)

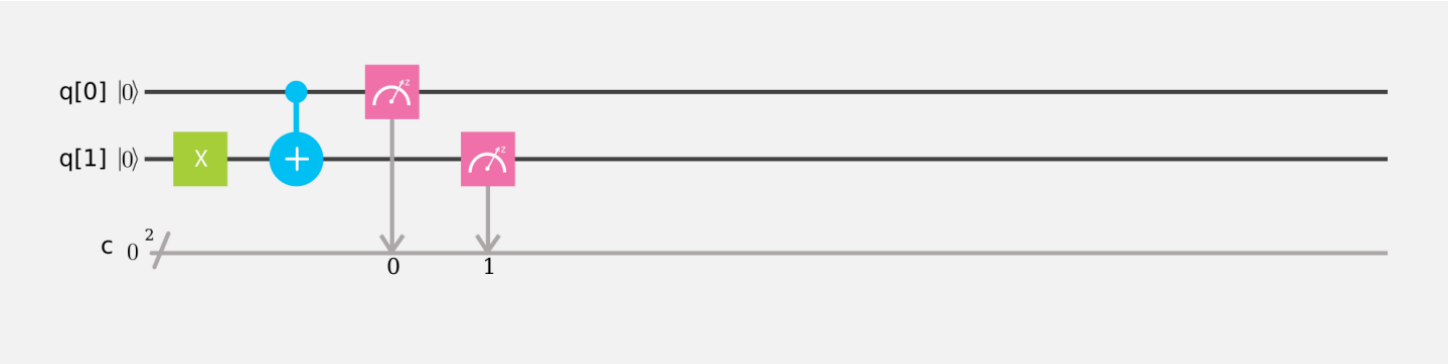
Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=1ae7fa71f3ad1e6f8fbe269fafcc7ca9&sharedCode=true)

CNOT (input 01)



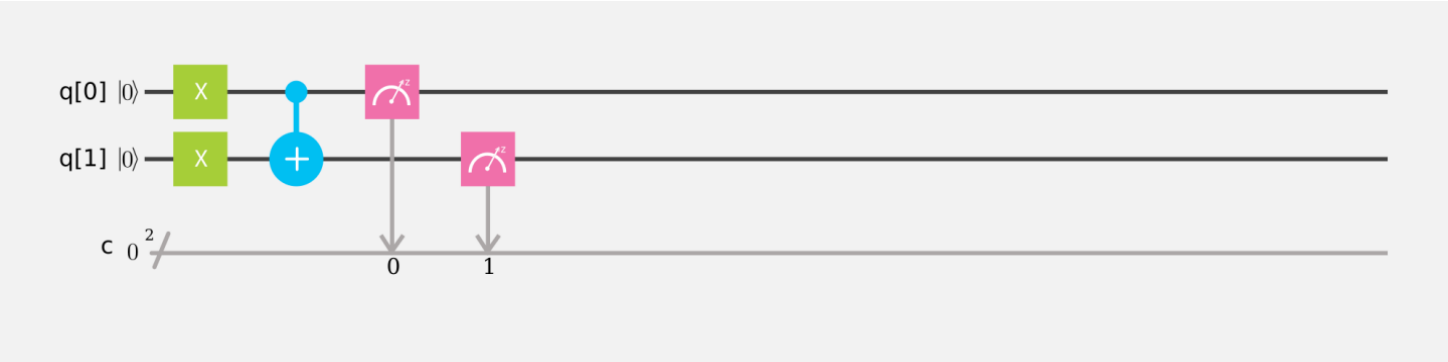
(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=1ae7fa71f3ad1e6f8fbe269fafd9abf4&sharedCode=true)
Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=1ae7fa71f3ad1e6f8fbe269fafd9abf4&sharedCode=true)

CNOT (input 10)



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=15d54765a0eeedcf83c173d5a6b80166&sharedCode=true)
Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=15d54765a0eeedcf83c173d5a6b80166&sharedCode=true)

CNOT (input 11)



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=7177ed173b5d2e3124c405339b9b6f15&sharedCode=true)
Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=7177ed173b5d2e3124c405339b9b6f15&sharedCode=true)

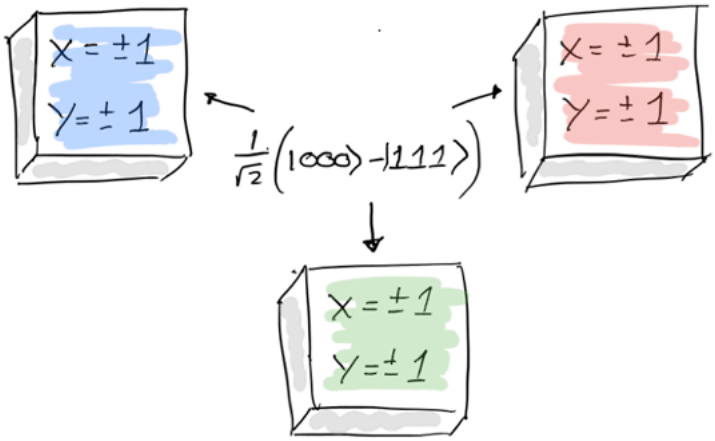
Entanglement

Bell and GHZ Tests

To finally settle the matter, John Bell designed a test called the Bell Test (which was later expanded into the CHSH inequality). He realized that measuring both of the two entangled qubits along the same axis (e.g. the Z direction) always gave complementary responses, but that this could have a classical explanation (i.e. a concrete rule that pre-determined the outcome). Instead, Bell realized he could perform the measurements of the two qubits along different measurement directions and by collecting statistics of the outcomes, he could show that the particles were not behaving classically. Describing how this works with two particles requires an understanding of statistics, so we will not go through it here. Instead we will introduce a more advanced multi-qubit entangled state.

Let's suppose that instead of two qubits, there are now three. We prepare a superposition of all qubits in the 0 state and all qubits in the 1 state: $\frac{1}{\sqrt{2}}(|000\rangle - |111\rangle)$. This state is called a GHZ state. GHZ states are named after Greenberger, Horne, and Zeilinger, who were the first to study them in 1997. By performing a series of measurements on this state, we can prove that this state violates certain assumptions that most people consider to be "truths" about our world.

Classically, the particles (in our case qubits) can only have one of two outcomes when they are measured – either 0 or 1. Additionally, if they were classical particles, there would be "hidden variables" that pre-determine the outcome of the measurements. In order to show the quantum nature of the GHZ state, and to prove that no such hidden variables exist, we need to consider different combinations (called correlations) of the possible outcomes of the measurements along the different bases (X and Y). This is well described in the 1990 paper by N. David Mermin, called "*What's wrong with these elements of reality?*"

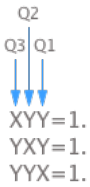


Imagine you have three independent systems which we denote by a blue, red, and green box. You are asked to solve the following problem: in each box there are two questions, labeled X and Y that each have only two possible outcomes, 1 or -1. The three systems could be the three entangled qubits of the GHZ state. We want to closely examine the *correlations* between the qubits.

Instead of mapping the $|0\rangle$ quantum state to the number 0 and the $|1\rangle$ quantum state to the number 1, for the purpose of this illustration we will map $|0\rangle$ to 1 and $|1\rangle$ to -1. (This will prevent us from having to multiply by 0 and lose information in the process). Each measurement of a qubit gives us either a 1 or -1, and then we'll multiply the results (i.e. the 1's and -1's) together and look at the correlations.

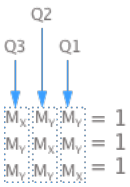
Here's how the notation works in practice. Let's say the three qubits together are in the state $|000\rangle$. When we measure all three of them in the Z basis, we would represent that quantum measurement as ZZZ. Based on the above mapping, the results would be 1, 1, 1 and the value of ZZZ would be 1. If the state was $|001\rangle$, then measuring ZZZ would give you -1.

To measure in another basis, such as XXX, we need to apply gates to the qubits to rotate them to the Z basis before measurement (as discussed previously). For XXX, that means applying a Hadamard gate to each of the qubits. We can also perform the following measurements of the three qubit system with the below results:



Bell and GHZ Tests (Cont.)

Here's where we show that our classical intuition is wrong.
Let's say that instead of performing quantum measurements on a quantum system, we perform "classical" measurements on a similar "classical" system. To distinguish the two, we'll call measurements along the X direction "M_X" for the classical system and measurements along Y "M_Y".
If the outcomes (M_X or M_Y) were pre-determined for each qubit ahead of the measurement, then we could already plug in a "1" or "-1" for each of the M_X and M_Y measurements for each qubit (though we don't yet know if we should use a 1 or -1). Since each of the above represents a measurement on *all 3 qubits*, we need to multiply down the columns to get the results we would have obtained for each of the individual qubits (again, assuming the outcomes were pre-determined and independent of each other!).



Multiply down all columns

Let's multiply down one of the columns, e.g. the one for qubit 3. As you can see, we get $M_X * M_Y * M_Y$ for qubit 3. We already learned that these measurements can only be one of two numbers: 1 or -1. In either case, $M_Y * M_Y$ will turn out to be 1. So, this expression can be simplified to just M_X. When we do this for all of the columns, we find that the above system of equations turns out to be equivalent to $M_{X,Q3}M_{X,Q2}M_{X,Q1} = 1$. This is the outcome we'd get if the system was classical. However, **quantum mechanics and experiments** both tell us that the result of an XXX measurement on the GHZ state (the quantum equivalent of the classical $M_{X,Q3}M_{X,Q2}M_{X,Q1}$) is **XXX** = -1!

Based on the results of these experiments, we have to accept what quantum mechanics teaches us: there is not hidden information in each independent qubit's state which predetermines the outcomes of measurements X and Y, independent of the other two qubits. **The outcomes of X or Y measurements for any given qubit are fundamentally tied to the outcomes of the X and/or Y measurements of the qubits with which it is entangled.** For example, if the Y measurements on qubits 2 and 3 give +1, then the X measurement of qubit 1 will give **+1**. If the X measurements of qubits 2 and 3 give +1, then the X measurement of qubit 1 will give **-1**. Strange but true!

Important note: the **combined** properties of the three-qubit system (e.g. the outcome of XYY) can be predicted, but the **individual** outcome of measuring each qubit (e.g. X,

Y) cannot! This is an important property of entanglement!

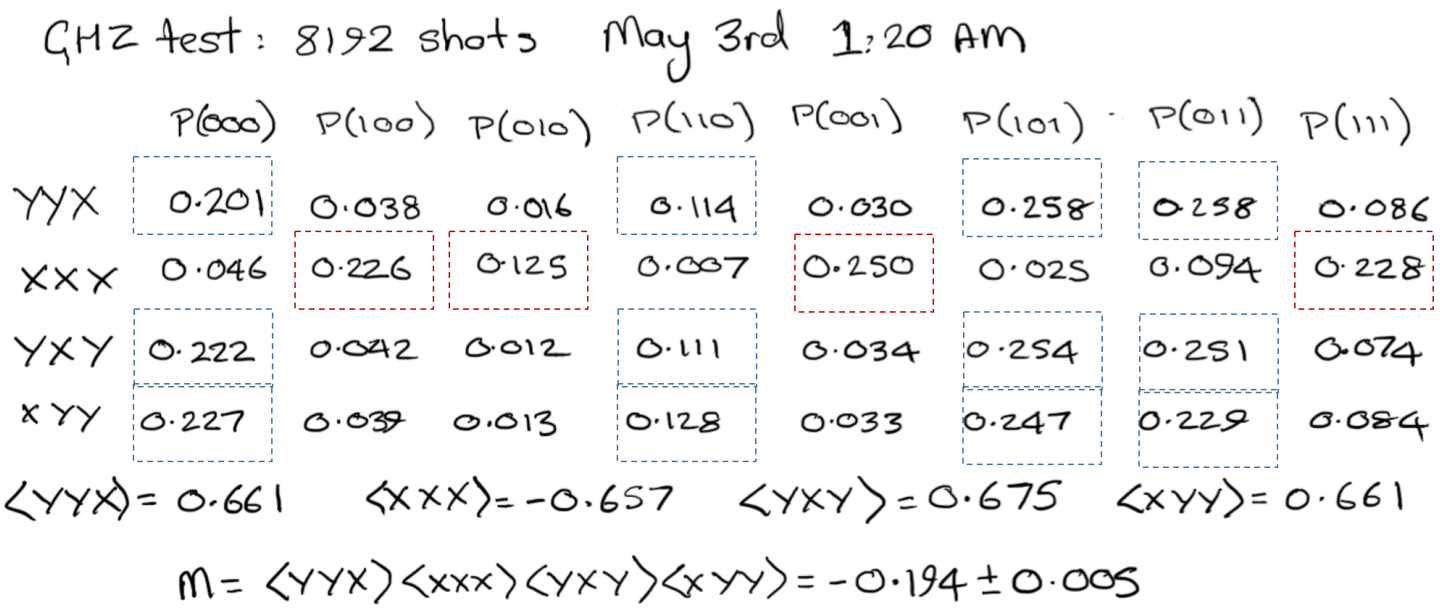
Results from the GHZ test in the Quantum Experience

Amazingly enough, we can test this ourselves in the Quantum Experience by creating a GHZ state, applying gates to the three qubits to put them in a state of superposition and entanglement, and performing the above combination of measurements. When we measure all three along the X basis, we get the equivalent of this "-1" condition compared to when we measure two of the qubits along the Y basis and the third along the X basis.

Our observation tells us that the system behaves as quantum mechanics predicts, and not how our classical intuition would have predicted!

Measurement (Q3/Q2/Q1)	**States that are observed**
YYX	000, 011, 101, 110
YXY	000, 011, 101, 110
XYY	000, 011, 101, 110
XXX	111, 100, 010, 001

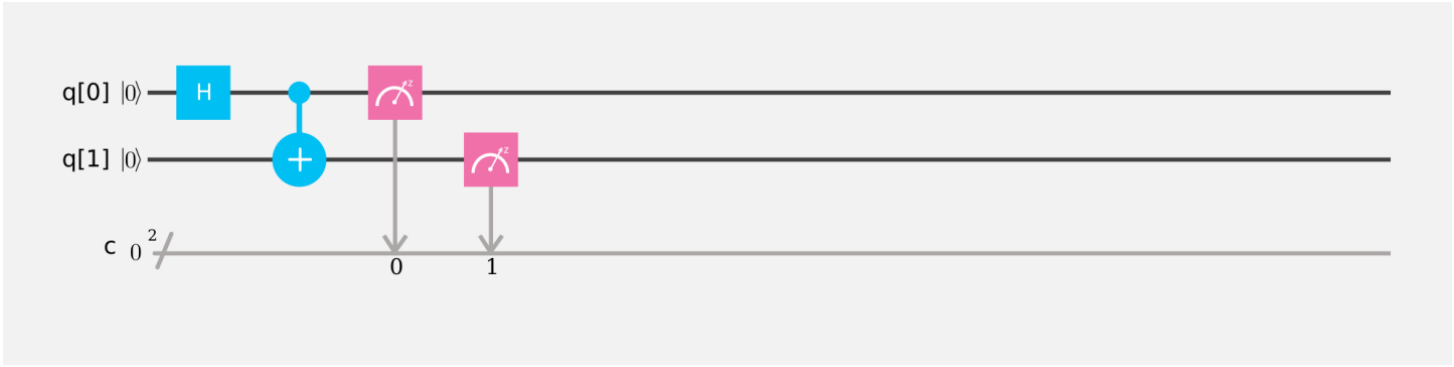
Note once again: if we measure in the standard basis (along Z), we get only 000 and 111 (with a 50-50 split!). The noise (small errors) in the below table comes from experimental error due to the fact that we used a real quantum processor, which is imperfect.

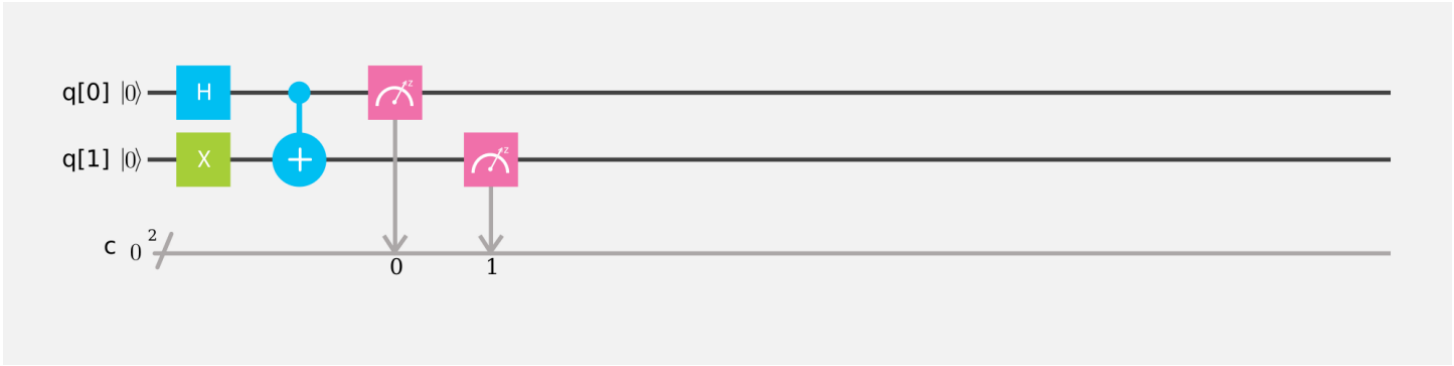


We will soon be adding a section to this Beginner's Guide about how to implement a GHZ test yourself in the QX! So watch this space.

We now come to entanglement, the weirdest of all quantum phenomena. Two or more quantum objects are entangled when, despite being too far apart to influence one another, they behave in ways that are 1) individually random, but also 2) too strongly correlated to be explained by supposing that each object is independent from the other. An entangled state is a state consisting of multiple qubits which cannot be expressed as a list of the individual qubits. For example, none of these two-qubit states: $|00\rangle$, $|01\rangle$, $|10\rangle$, or $|11\rangle$, is entangled because they can all be described by attributing a definite state to each qubit. The state $(|00\rangle + |01\rangle)/\sqrt{2}$, is also not entangled, because it can be expressed by saying the first qubit is in a superposed single-qubit state $(|0\rangle + |1\rangle)/\sqrt{2}$ and the second in the $|0\rangle$ state. However, the state $(|01\rangle + |10\rangle)/\sqrt{2}$ is entangled, because there is no way of expressing it as a list of one-qubit states. When you measure one of the qubits in this state, along any measurement axis at all, it behaves randomly, but its random behavior allows you to perfectly predict how the other qubit would behave if measured along the same axis. No unentangled state exhibits this distinctive combination of perfect correlation with perfect individual randomness.

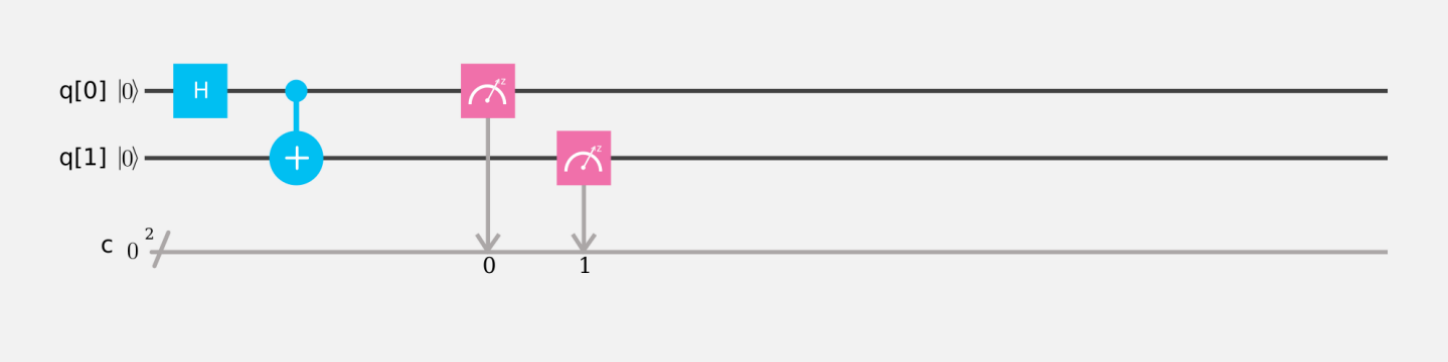
Two examples of Bell, or entangled, states:





Some people may find this hard to believe. Our classical intuition wonders if rather than two particles existing in a superposition of both states until measured, one of the particles was 1 all along and the other was 0 all along. This is called a “local hidden variable” theory.

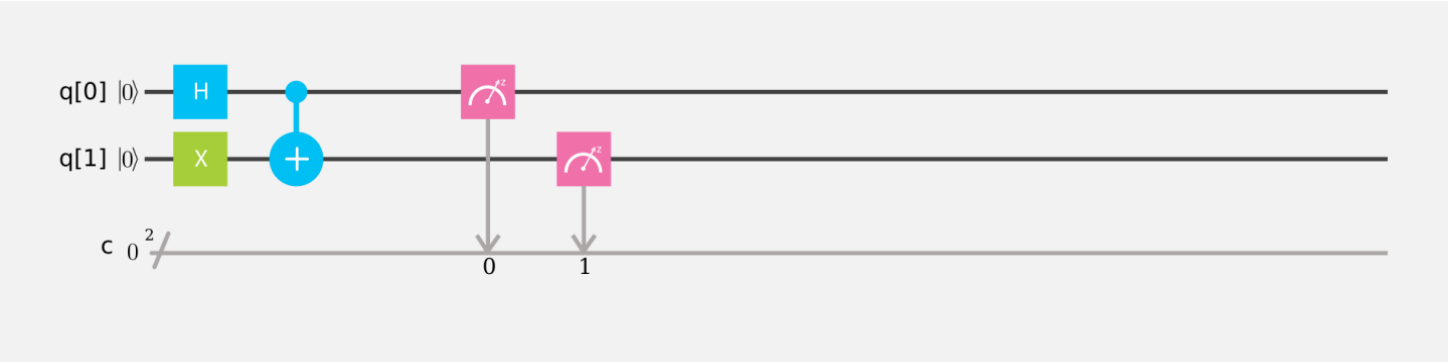
Example of Bell, or entangled, states



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=1fcc3d6878c984ce05e69298a76abe36&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=1fcc3d6878c984ce05e69298a76abe36&sharedCode=true)

Example of Bell, or entangled, states (2)



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=136177cb0d92579d4630b69d7812ea54&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=136177cb0d92579d4630b69d7812ea54&sharedCode=true)

|

Full User Guide

If quantum physics sounds challenging to you, you are not alone. All of our intuitions are based on day-to-day experiences and are defined by classical physics — so most of us find the concepts in quantum physics counterintuitive at first. In order to comprehend the quantum world, you must let go of your beliefs about our physical world, and develop an intuition for a completely different (and often surprising) set of laws.

Our goal with the IBM Quantum Experience is to introduce this world through a set of short tutorials in our User Guide, and by providing the hands-on opportunity to experiment with operations on a real quantum computing processor. This way, we hope to foster a quantum intuition in the greater community, and spark further interest in those who are curious. By making quantum concepts more widely understood—even on a general level— we can more deeply explore all the possibilities quantum computing offers, and more rapidly bring its exciting power to a world whose perspective is limited by classical physics.

Frequently Asked Questions

General questions about quantum information science

What does “quantum” mean?

Quantum theory is a revolutionary advancement in physics and chemistry that emerged in the early twentieth century. It is an elegant mathematical theory able to explain the counterintuitive behavior of subatomic particles, most notably the phenomenon of **entanglement**. In the late twentieth century it was discovered that quantum theory applies not only to atoms and molecules, but to bits and logic operations in a computer. This realization has been bringing about a revolution in the science and technology of information processing, making possible kinds of computing and communication hitherto unknown in the Information Age.

What is a quantum computer?
A quantum computer is a device able to manipulate delicate quantum states in a controlled fashion, not dissimilar to the way an ordinary computer manipulates its bits.

Can a quantum computer solve NP-complete problems?

We do not believe that quantum computers are likely to provide any more than a quadratic speedup to NP-complete problems. This is the consensus view of the quantum information research community, and in fact the idea that quantum computers can solve NP-complete problems efficiently has been described as “the *central* crock about quantum computing (http://www.scottaaronson.com/democritus/lec10.html)”.

What is a qubit?

A qubit (pronounced “cue-bit” and short for quantum bit) is the physical carrier of quantum information. It is the quantum version of a bit, and its quantum state can take values of $|0\rangle$, $|1\rangle$, or both at once, which is a phenomenon known as **superposition**.

What is a superposition?

A superposition is a weighted sum or difference of two or more states; for example, the state of the air when two or more musical tones are sounding at once. Ordinary, or “classical,” superpositions commonly occur in macroscopic phenomena involving waves.

How are quantum superpositions different?

Quantum theory predicts that a computer with n qubits can exist in a superposition of all 2^n of its distinct logical states $|000..0\rangle$, through $|111\ldots1\rangle$. This is exponentially more than a classical superposition. Playing n musical tones at once can only produce a superposition of n states.

How is superposition different from probability?

A set of n coins, each of which might be heads or tails, can be described as a probabilistic mixture of 2^n states, but it actually **is** in only one of them—we just don't know which. For this reason, quantum superposition is more powerful than classical probabilism. Quantum computers capable of holding their data in superposition can solve some problems exponentially faster than any known deterministic or probabilistic classical algorithm. A more technical difference is that while probabilities must be positive (or zero), the weights in a superposition can be positive, negative, or even complex numbers.

How is a quantum superposition different from exponential parallelism?

Just as a superposition is stronger than a probabilistic mixture, so it is weaker than actually having an exponentially large army of real computers (which is an unrealistic proposition in any case, since the observable universe isn't big enough to hold 2^{100} of anything). The power of quantum computers remains to be explored, but it is considered to be strictly weaker than exponential parallelism, and strictly stronger than probabilism.

What is entanglement?

Entanglement is a property of most quantum superpositions and does not occur in classical superpositions. In an entangled state, the whole system is in a definite state, even though the parts are not. Observing one of two entangled particles causes it to behave randomly, but tells the observer how the other particle would act if a similar observation were made on it. Because entanglement involves a correlation between individually random behaviors of the two particles, it cannot be used to send a message. Therefore, the term "instantaneous action at a distance," sometimes used to describe entanglement, is a misnomer. There is no **action** (in the sense of something that can be used to exert a controllable influence or send a message) – only **correlation**, which, though uncannily perfect, can only be detected afterward when the two observers compare notes. The ability of quantum computers to exist in entangled states is responsible for much of their extra computing power, as well as many other feats of quantum information processing that cannot be performed, or even described, classically.

What is the uncertainty principle?

In quantum physics, we cannot simultaneously know two non-commuting variables (like the position and momentum of a particle). This implies that a quantum system in a perfectly definite state can be certain under one measurement and completely random under another. Moreover, if a quantum system starts out in an arbitrary unknown state, no measurement can reveal complete information about that state; and the more information the measurement reveals, the more the state is disturbed. This is a underlying principle of quantum cryptography.

What is a quantum gate?

A quantum gate is an operation that is applied to a qubit to change its state. To generate entanglement you must have at least a two-qubit gate equivalent to the CNOT.

General questions about the IBM Quantum Experience

What is the IBM Quantum Experience?

The IBM Quantum Experience is a cloud-based platform where you can learn, research, and interact with a real quantum computer housed in an IBM Research lab.

What is the Quantum Composer?

The Quantum Composer is a graphical interface tool where you can drag and drop different operations to control qubits. The Quantum Composer permits you to develop your own quantum algorithms, which we call Quantum Scores.

What is a Quantum Score?

A Quantum Score is the set of instructions, or algorithm, to a quantum computer. It is a series of gates versus time played on different qubits, much like a musical score.

What is the Quantum Sphere?

The Quantum Sphere is a graphical representation of the output of a Quantum Score. It provides the user a way to easily visualize properties of the measurements performed on a number of qubits, all in one diagram.

Questions about quantum computers

What is a universal quantum computer?

A universal quantum computer is a machine that can simulate an arbitrary quantum state from an arbitrary input quantum state.

What is a universal fault-tolerant quantum computer?

A universal fault-tolerant quantum computer is the grand challenge of quantum computing. It is device that can properly perform universal quantum operations using unreliable components.

When will I have a quantum computer?

You have access to one now with the Quantum Experience. It is small at the moment, with a five-qubit processor, but it is a work-in-progress that we are continually improving.

What does a quantum computer look like?

A quantum computer looks like nothing you have on your desk, or in your office, or in your pocket. It is housed in a large unit known as a dilution refrigerator and is supported by multiple racks of electronic pulse-generating equipment. However, you can access our quantum computer with very familiar personal computing devices, such as laptops, tablets, and smartphones.

Questions about our qubits and experiments

What is the qubit that you are physically using?

The qubit we use is a fixed-frequency superconducting transmon qubit. It is a Josephson-junction-based qubit that is insensitive to charge noise. For more information on this type of qubit please see here (Koch *et al.* 2007 (http://journals.aps.org/pra/abstract/10.1103/PhysRevA.76.042319)). We use fixed-frequency qubits, as opposed to tunable qubits, to minimize our sensitivity to external magnetic field fluctuations that could corrupt the quantum information.

How do you make the qubits?

The superconducting qubits are fabricated at IBM. The devices are made on silicon wafers with superconducting metals such as niobium and aluminum. Details about the fabrication processes are given in these references (Chow *et al.* 2014 (http://www.nature.com/ncomms/2014/140624/ncomms5015/full/ncomms5015.html), Córcoles *et al.* 2015 (http://www.nature.com/ncomms/2015/150429/ncomms7979/full/ncomms7979.html)).

What are the properties of these qubits?

The properties of the qubits can be seen below the Quantum Composer. Properties such as relaxation time (T_1), coherence time (T_2), readout errors, and gate errors are given, posted from the last calibration experiment run on the actual quantum processor device.

Where do the qubits live?

The quantum processor itself is housed inside of a printed circuit board package. This package is mounted inside of a light-tight, magnetic-field shielding can, which sits at the coldest stage at the bottom of a dilution refrigerator, housed in one of IBM's Quantum Computing labs.

What's a dilution refrigerator?

A dilution refrigerator is the machine we use to cool down our quantum processor device. The refrigerator cools the device down to around 15 miliKelvin. It works by circulating a mixture of two helium isotopes, ^{3} He and ^{4} He, in a closed cycle within a complex system of pipes and chambers.

What happens when I hit the "Run" button?

The graphical quantum score is first interpreted, then compiled to run efficiently on a particular qubit configuration. The compiled version is translated into a sequence of operations performed by equipment in our lab to control the qubits. The output is then passed back to the user, and a note is sent to your email address to alert you that the quantum hardware has run your experiment.

How are quantum gates performed in the system?

Quantum gates are performed by sending electromagnetic impulses at microwave frequencies to the qubits through coaxial cables. These electromagnetic pulses have a particular duration, frequency, and phase that determine the angle of rotation of the qubit state around a particular axis of the Bloch sphere. For single-qubit operations, only one pulse type needs to be calibrated, namely $X_{\pi/2}$. From that pulse, we can create all the gates you find in the Quantum Composer library. For example, we implement the S gate with the help of a phase transform performed purely in software, which affects the physical implementation of subsequent gates; however, this does not mean that the gate is necessarily error-free, as doing a software phase transform still requires very good phase stability from our instruments! Another example is the Hadamard gate performed by the sequence $SX_{\pi/2}S$.

What about two-qubit gates?

Two-qubit gates typically require tuning to calibrate the interaction between the two qubits during the gate duration, and minimizing the interaction at any other time. Since our qubits of choice are fixed-frequency transmons, we cannot tune the interaction by bringing them closer in frequency during the two-qubit gate. Instead, we exploit the cross-resonance effect (Chow *et al*., 2011 (http://journals.aps.org/prl/abstract/10.1103/PhysRevLett.107.080502)), by driving one of the qubits (called **control**) with a microwave pulse tuned at the frequency of the second qubit (called **target**). By doing this, we can actively increase the strength of the coupling between them. The nature of the cross-resonance effect also allows us to perform rotations in the target qubit conditioned on the state of the control qubit, a key characteristic of the CNOT operation required for a universal quantum gate set. From our cross-resonance microwave pulse, we only need to perform an additional frame change S on the control qubit and a $X_{\pi/2}$ on the target to implement a CNOT.

How are measurements performed in the system?

We must perform the qubit measurements in a way that does not destroy the qubit quantum state. One method is to weakly couple each qubit to a microwave resonator whose resonance characteristics depend on the state of the qubit. Once the qubit operations are completed in your score, you can measure the qubits by sending a microwave tone to their resonators and analyzing the signal it reflects back. The phase and amplitude of this reflected signal will be different depending on the qubit state. These signals in the resonator are boosted via a chain of amplifiers inside of our dilution refrigerator, including a quantum-limited amplifier at 15 mK, and a high-electron mobility transistor amplifier at 4 K.

Do not forget to include measurements in your score! Because the measurement of a qubit in a superposition state seems random – the outcome is sometimes 0 and sometimes 1 – you must repeat the measurement multiple times to determine the likelihood of a qubit being in a particular state. When performing the experiment, you will be asked how many "shots" or experiments to run in order to determine the qubit state probabilities.

How often is the system tuned up?

We perform a full round of single- and two-qubit calibrations, as well as measurements of relaxation time, coherence time, and gate errors two times a day at 8AM and 8PM EST. Each full calibration round takes about one hour. During calibration, you will notice that the device will be "Down for Maintenance."

Questions about running the simulator versus the experiment

What does the simulator do?

The simulator computes the quantum state we expect a circuit to produce.

What is the difference between ideal and realistic simulator?

The ideal simulator treats each gate as a unitary matrix and composes the gates to find the output state. This simulator tells us what to expect when all of the operations are perfect. The realistic simulator, on the other hand, numerically solves a system of differential equations known as a master (https://en.wikipedia.org/wiki/Master_equation) equation. The master equation we solve includes dissipation and phase noise, as well as time-dependent terms for gates and interactions between adjacent qubits. The various interaction strengths are computed from realistic parameters and an effective Hamiltonian (https://en.wikipedia.org/wiki/Hamiltonian_%28quantum_mechanics%29). The results are qualitatively similar to what is observed in a typical experiment.

How many experiments can I run?

You can run as many experiments as you have Units to run; each experiment execution requires between 3 and 5 Units. No Units are required to perform simulations or to recall results of an experiment that was run previously.

How do I gain more Units?

Depending on your usage pattern and by requesting upgraded status, the team will consider converting users from Standard to Expert.

What happens when I run out of Units?

You can still run simulations or recall the result of the experiments that have been run previously, but you must wait for the Units to replenish, which happens either once your execution has run off the queue, or 24 hours, whichever is greater.

What is the difference between "Current Units" and "Promo Units"?

The maximum number of “Current Units” available is based on your user level. A Standard User has a maximum of 15 credits, and these credits are replenished upon the greater of 24 hours or when your execution has run off the queue. On the other hand, “Promo Units” are extra units that allow you to run extra executions but, once used, these credits won’t be replenished. If you have “Promo Units”, these will be used first rather than “Current Units”.

What are the different user levels?

Everyone begins as a Standard User. This level of user is advised to work through the entire User Guide, running scores in either Simulation mode or viewing cached results (both options require no Units). At the beginning, you will have 5 Units and 6 additional promo Units. Upon completion of the User Guide, the Standard User is given 15 Units for use on the actual quantum hardware. Request a replenishment of Units via the Account page. Once you have gotten familiar with the tool, tell us about yourself and request an upgrade to the Expert User level.

How do I join IBM Quantum Computing?

We are always interested in meeting the brightest and most inquisitive minds! Are you a quantum thinker (http://ibm-research.jobs/l/recruiting/jobsearchaction/1ed388e3-9ff9-11e4-b295-bc764e10782d/4fc7376b-11b4-11e6-bc9f-bc764e11b6f3/false?term=quantum&tags=&jobTypes=&locations=&postalCode=&distance=1000000)?

Introducing the IBM Quantum Experience

The Quantum World

Today’s computers perform calculations and process information using the standard (or as a physicist would say, “classical”) model of computation, which dates back to Turing (https://en.wikipedia.org/wiki/Alan_Turing) and von Neumann (https://en.wikipedia.org/wiki/John_von_Neumann). In this model, all information is reducible to bits, which can take the values of either 0 or 1 – and all processing can be performed via simple logic gates (https://en.wikipedia.org/wiki/Logic_gate) (AND, OR, NOT, NAND) acting on one or two bits at a time. At any point in its computation, a classical computer’s state is entirely determined by the states of all its bits, so that a computer with n bits can exist in one of 2^n possible states, ranging from $00...0$ to $11\ldots1$.

The power of the quantum computer, meanwhile, lies in its much richer repertoire of states. A quantum computer also has bits. But instead of 0 and 1, its quantum bits, or *qubit*s, can represent a 0, 1, or both at once, which is a property known as **superposition**. This on its own is no special thing, since a computer whose bits can be intermediate between 0 and 1 is just an analog computer, scarcely more powerful than an ordinary digital computer. However, a quantum computer takes advantage of a special kind of superposition that allows for * exponentially many* logical states at once, all the states from $|00...0\rangle$ to $|11\ldots1\rangle$. This is a powerful feat, and no classical computer can achieve it. The vast majority of these quantum superpositions, and the ones most useful for quantum computation, are **entangled**—they are states of the whole computer that do not correspond to any assignment of digital or analog states of the individual qubits. While not as powerful as exponentially many classical computers, a quantum computer is significantly more powerful than any one classical computer – whether it be deterministic, probabilistic, or analog. For a few famous problems (such as factoring large numbers), a quantum computer is clearly the winner over a classical computer. A working quantum computer could factor numbers in a day that would take a classical computer millions of years.

One might think that the difficulty in understanding quantum computing or quantum physics lies in “hard math”... but mathematically, quantum concepts are only a bit more complex than high school algebra. Quantum physics is hard because, like Einstein’s theory of relativity, it requires internalizing ideas that are simple but counter-intuitive. What is strange about relativity is the concept that time and space are interconnected, when common sense tells us they should act independently. If you begin to explain relativity to a person new to the idea by jumping straight to time and space, you will likely be greeted by a blank stare. A better way to start is as Einstein did, by explaining that relativity follows from a simple physical principle: the speed of light is the same for all uniformly moving observers. This one modest idea then becomes extremely profound and leads, by inexorable logic, to Einsteinian spacetime.

The counter-intuitive ideas of quantum physics that one must accept are 1) *a physical system in a perfectly definite state can still behave randomly*, and 2) *two systems that are too far apart to influence each other can nevertheless behave in ways that, though individually random, are somehow strongly correlated*. Unfortunately, unlike relativity, there is no single simple physical principle from which these conclusions follow. The best we can do is to distill quantum mechanics down to a few abstract-sounding mathematical laws, from which all the observed behavior of quantum particles (and qubits in a quantum computer) can be deduced and predicted. And, as with relativity, we must guard against attempting to describe quantum concepts in classical terms.

Quantum Laws in Black, White, and Blackandwhite

Quantum laws are, as far as we know, the most fundamental physical laws; they are inviolable. Here is our version of quantum physics distilled into five key laws**.** If the math is new to you, just print yourself a copy of these laws and skip them for now; while you work through the rest of the tutorial, return to them every now and then to see how they tie into what you are learning.

Quantum is a system like everything else.

To each physical system there corresponds a Hilbert space (1) of dimensionality equal to the system’s maximum number of reliably distinguishable states (2).

A quantum state is a configuration of the system.

Each direction (ray) in the Hilbert space corresponds to a possible state of the system (3), with two states being reliably distinguishable if and only if their directions are orthogonal (inner product is zero).

A quantum state changes; it naturally wants to evolve, but it can always be undone.

Evolution of a closed system is a unitary (4) transformation on its Hilbert space.

Scaling - how parts make a whole.

The Hilbert space of a composite system is the tensor product of the Hilbert space of the parts (5).

Quantum measurements are probabilistic.

Each possible measurement (6) on a system corresponds to a resolution of its Hilbert space into orthogonal subspaces $\{\Pi_j\}$ where $\sum_j \Pi_j = 1$. On state $|\psi\rangle$ the result j occurs with probability $P(j) = \langle\psi|\Pi_j|\psi\rangle$ and the state after the measurement is $|\psi_j\rangle = \Pi_j|\psi\rangle/\sqrt{P(j)}$.

These five principles are the foundation for the whole quantum world.
Clarifications

1. A Hilbert space is a linear vector space with complex coefficients and inner products $\langle\phi|\psi\rangle = \sum_i \phi_i^* \psi_i$.
2. For a single qubit, there are two standard orthogonal states (*computational basis states*) that are conventionally denoted $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$.
3. Other qubit states include $|+\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \end{pmatrix}$, $|-\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ -1 \end{pmatrix}$, $|\circlearrowright\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ i \end{pmatrix}$ and $|\circlearrowleft\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ -i \end{pmatrix}$
4. *Unitary* means linear and inner-product-preserving.
5. A two-qubit system can exist in a product state such as $|00\rangle$ or $|0+\rangle$ but also in an *entangled* state $(|00\rangle + |11\rangle)/\sqrt{2}$, in which neither qubit has a definite state, even though the pair together does.
6. Measurement causes the system to behave probabilistically and forget its pre-measurement state, unless that state happens to lie entirely with one of the subspaces Π_j.

The Quantum Composer

The **Quantum Composer** is our graphical user interface for programming a quantum processor. Those familiar with quantum computing may recognize the composer as a tool to construct *quantum circuits* using a library of well-defined gates and measurements. For those not familiar, we will explain a few of the key parts.

When you first click on the "Composer" tab above, you will have a choice between running a *real* quantum processor or a *custom* quantum processor. In the custom processor, gates can be placed anywhere, whereas in the real processor, the topology is set by the physical device running in our lab (note that this restricts the usability of some of the two-qubit gates).

Once you are in the "Composer" tab, you can start making your very own quantum circuits!

With the Composer, you can create a *quantum score,* which is analogous to a musical score in several respects. Time progresses from left to right. Each line represents a qubit (as well as what happens to that qubit over time). Each qubit has a different frequency, like a different musical note. Quantum gates are represented by square boxes that play a frequency for different durations, amplitudes, and phases. Gates on just one line are called single-qubit gates. The gates made with vertical lines connecting two qubits together are known as CNOT gates; these two-qubit gates function like an exclusive OR gate in conventional digital logic. The qubit at the solid-dot end of the CNOT gate controls the whether or not the target qubit at the $\oplus$-end of the gate is inverted (hence controlled NOT, or CNOT). Some gates, like the CNOT, have hardware constraints; the set of allowed connections is defined by the schematic of the device located below the Quantum Composer, along with recently calibrated device parameters.

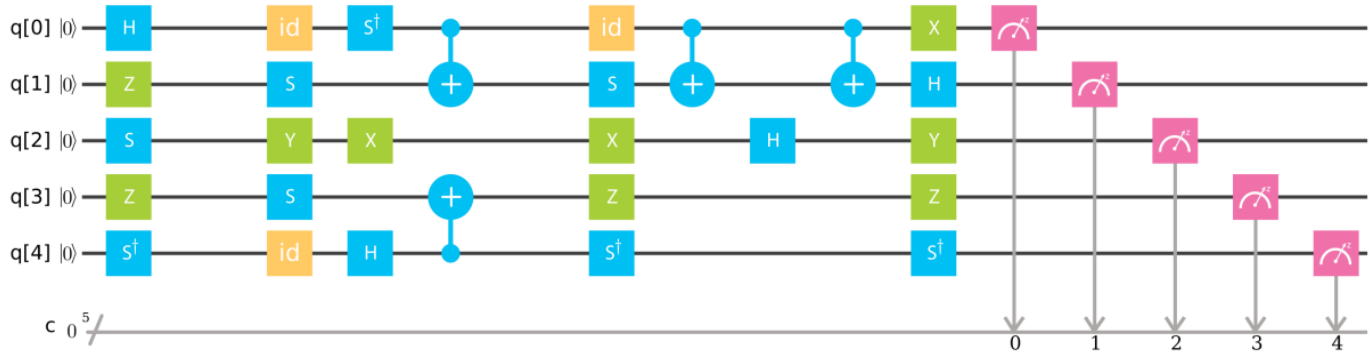
The Quantum Composer's library (located to the right of the qubit stave) contains many different classes of gates: single-qubit gates, such as the **yellow** idle operation; the **green** class of *Pauli operators*, which represent bit-flips (X, equivalent to a classical NOT); phase-flips (Z); and a combined bit-flip and phase-flip (Y). We also offer *Clifford operations*, which are the **blue** class of gates, such as H, S, and $S^\dagger$ gates for generating quantum superpositions, as well as the two-qubit entangling gate CNOT previously mentioned. The **red** gates are two-phase gates that are not in the Clifford group and are important for giving quantum computing its power. To measure the state of any qubit, use the **pink** standard measurement operation, which is a simple Z projection that is assigned to a classical bit in a classical bit register. If you ever need a reminder, hit the Help button (the question mark near the gates heading) for a quick summary of all available gates. A quantum algorithm (circuit) begins by preparing the qubits in well-defined states (here the ground state, $|0\rangle$, which we've automatically done for you), then executing a series of one- and two-qubit gates in time, followed by a measurement of the qubits.

If you are feeling brave, you can hit the **Advanced** button to view an additional set of gate operations and sub-routines; these are described in further detail here (https://quantumexperience.ng.bluemix.net/qstage/#/tutorial?sectionId=89ada8b1aa9e798ce6aa9a705feab237&pageIndex=0).

To use the Composer, simply drag the gate boxes onto the qubit stave to place them. Double-tap the boxes to delete, or drag them to the trash bin. To place a CNOT gate, drag first onto the target qubit (a $\oplus$ symbol will appear), then click on the control qubit (a solid dot will appear). Note that on the real quantum processor, you cannot add more gates to a circuit after placing a measurement; this feature will be added in the future.

Load the quantum score below and try out a simulation, or start composing your own!

My First Score



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=1f234d4750fe47817393d8e1c801c478&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=1f234d4750fe47817393d8e1c801c478&sharedCode=true)

Running your Quantum Scores

Now that we've gotten familiar with the Composer, let's go over how to run a quantum score.

When you begin an experiment, you'll be prompted to give it a name, so that you can recognize it later. You will also see two choices: Real Quantum Processor, or Custom Topology. In both cases, you create your score by dragging gates onto the stave, adding a measurement, and then hitting "Run" for the score to execute.

Running on an Custom Quantum Processor

If you select “Custom Topology,” your only option is to run your score in simulation. This is because the custom processor permits all-to-all connectivity; the real device, in contrast, is limited by physical connectivity. When you select Custom Topology, a dialogue box will ask you to select the number of qubits and classical bits assigned to different registers. We have set the maximum number of qubits to 20. The execution of your circuit happens immediately (unless the number of qubits is large) and the output can then be viewed in the Results (see the next section). Try out the “Single Qubit Measurement” below.

Running on a Real Quantum Processor (Requires Units)

If you select “Real Quantum Processor,” the score you compose will be placed into the experimental queue when you hit “Run,” and you will be notified via email when it has been executed on the actual quantum computer in our lab.

You must have Units in your Quantum Experience account to use the Real Quantum Processor. If the score you are trying to run has previously been run, you will be given the choice of seeing the results from the cached execution right away (which costs no Units), or spending the Units to re-run the score as a new execution (meaning it will go into the experimental queue, and you will receive an email notification when it is done).

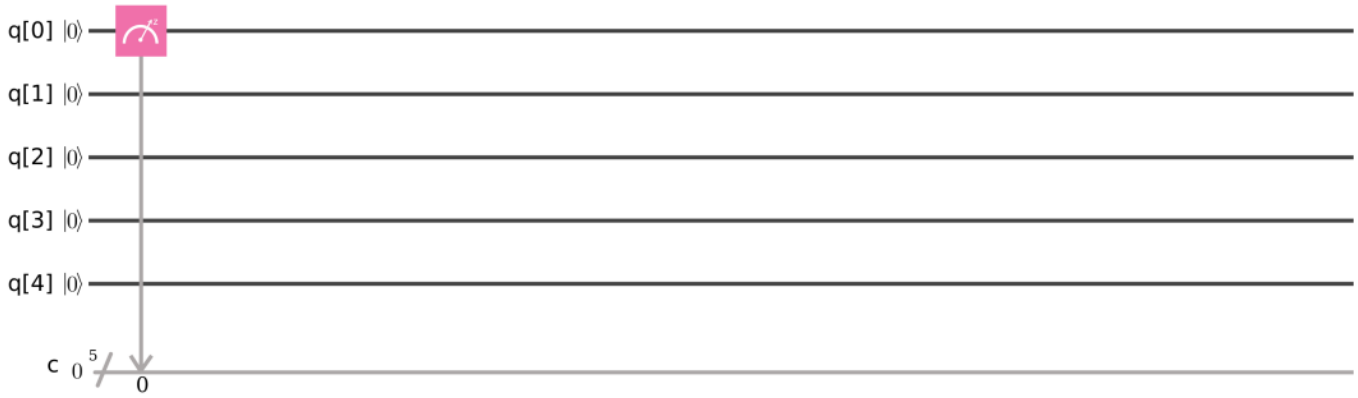
Note that many of the score examples found in this User Guide have previous executions available for you to view and experiment with!

Once you hit “Run,” your score’s progress will be visible in the “My Scores” tab, sorted by date executed. When your results are ready, you will be able to view them from this tab. You can also re-edit your score and execute it on the simulator while you wait for your results to return.

Single Qubit Measurement

(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=c0e1386ab6ad50c7464096012f656334&sharedCode=true)
Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=c0e1386ab6ad50c7464096012f656334&sharedCode=true)

Single Qubit Measurement (Real)



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=68d7454c2e1e9e6dfd17d2c0289a387f&sharedCode=true)
Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=68d7454c2e1e9e6dfd17d2c0289a387f&sharedCode=true)

The Results

In our Quantum Experience, the results from launching your quantum scores can be visualized in two different ways: a standard histogram/bar graph, and as a Quantum Sphere, or QSphere. The QSphere, unique to the Quantum Experience, represents quantum circuit measurement outcomes in a visually striking and information-dense graphic.

After performing a quantum measurement, a qubit’s information becomes a classical bit, and in our system (as is standard) the measurements are performed in the computational basis. For each qubit the measurement either takes the value 0 if the qubit is measured in state $|0\rangle$ and value 1 if the qubit is measured in state $|1\rangle$.

In a given run of a quantum circuit with n measurements, the result will be one of the 2^n possible n-bit binary strings. If the experiment is run a second time, even if the measurement is perfect and has no error, the outcome may be different due to the fundamental randomness of quantum physics. The results of a quantum circuit executed many different times can be represented as a distribution over the full 2^n possible outcomes. It is not scalable to represent all possible outcomes; therefore, we keep only those outcomes that happen in a given experiment and represent them in two different ways: as bars or as a Quantum Sphere.

Histogram representation (Bars)

The histogram/bar graph representation is the simplest to understand. The height of the bar represents the fraction of instances the outcome comes up in the different runs on the experiment. Only those outcomes that occurred with non-zero occurrences are included. If all the bars are small for visualization only (not if you download the data) they are collected into single bar called other values. In general this is not a problem as a good quantum circuit should not have many outcomes only circuits that have the final state in a large superposition will give many outcomes and these would take exponential measurements to measure.

The Quantum Sphere representation (QSphere)

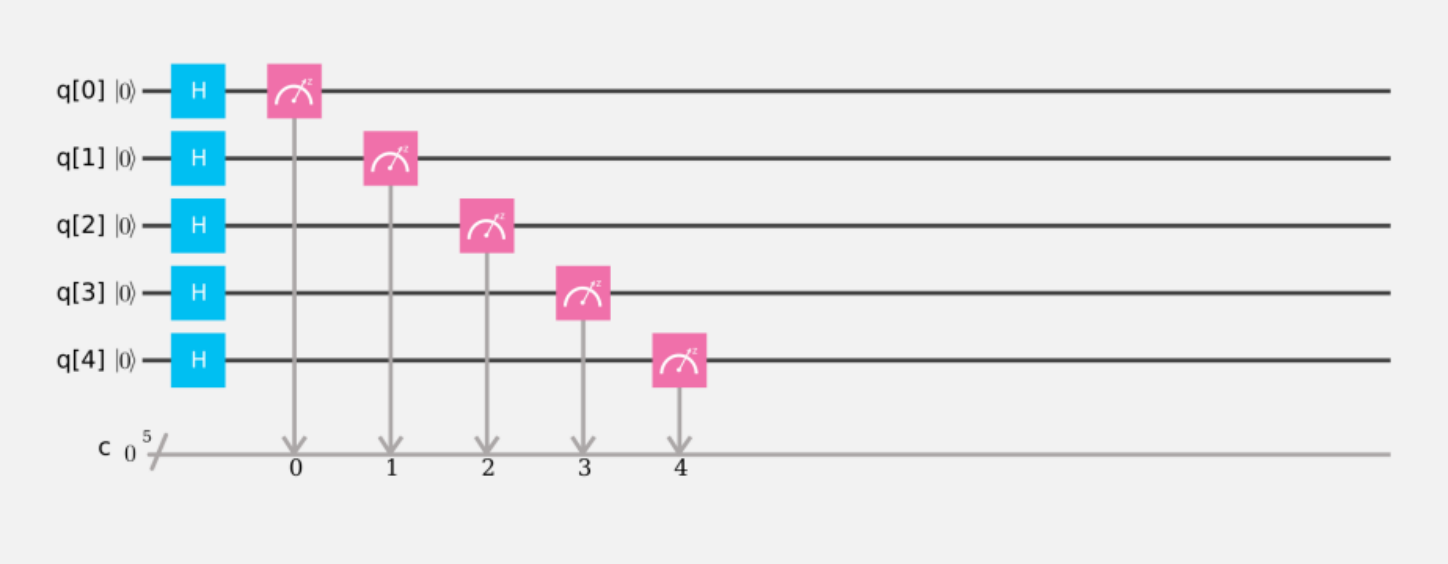
The QSphere is our new alternative representation to visually show the same data as the bar graph neatly and strikingly. Each line from the center represents a possible outcome of the experiment, and the weight (darkness of the line) represents the likelihood of each outcome. As with the histogram, only those outcomes are included that occurred in a given experiment. The QSphere is divided into $n+1$ levels, and each section represents the weight (total number of 1s) of the binary outcome. The top is the $|0\ldots0\rangle$ outcome, the next line is all the outcomes with a single 1 ($|10\ldots0\rangle$, $|01\ldots0\rangle$, etc), the line after that is all outcomes with two 1s, and so on until the bottom is the outcome $|1\ldots1\rangle$.

For a single qubit there are two outcomes, and the sphere has only two levels; for two qubits, it has three sections with the middle section separated into two parts; for three qubits, it has four sections with the middle two being broken into three sections, and so on, following Pascal's triangle.

$$
\begin{array}{ccccccccccc}
 & & & & 1 & & & & & & \\
 & & & 1 & & 1 & & & & & \\
 & & 1 & & 2 & & 1 & & & & \\
 & 1 & & 3 & & 3 & & 1 & & & \\
1 & & 4 & & 6 & & 4 & & 1 & & \\
1 & 5 & & 10 & & 10 & & 5 & & 1 &
\end{array}
$$

The usefulness of this representation is for distinguishing classical states from entangled states. A computational basis state will have a single line pointing in one direction. Under the assumption the state is pure, a superposition of two basis states will have two lines pointing in two directions of half weight. If these directions are on opposite sides of the QSphere we have a state that is maximally entangled (for $n > 1$) in the computation bases. Finally if there are faint lines in every direction we have made a uniform superposition state.

Superposition +++++



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=8ea323460b6f03cb9002b802139af008&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=8ea323460b6f03cb9002b802139af008&sharedCode=true)

Quantum computing is here. While today's quantum processors are modest in size, the complexity grows continuously. The time is ripe to build and engage a community of new quantum learners, so that we change the way we think about computing. Our goal with the IBM Quantum Experience is for you to learn about the quantum world by reading this User Guide, composing your own experiments, running them in simulation, and executing them on the world's first fully-controllable quantum processor through the IBM Cloud.

The IBM Quantum Experience consists of:

- a set of tutorials (this User Guide) that will lead you from the basics of simple single-qubit experiments (Section II (/qstage/#/tutorial?sectionId=71972f437b08e12d1f465a8857f4514c)) to more complicated multi-qubit experiments (Section III (/qstage/#/tutorial?sectionId=050edf961d485bfcd9962933ea09062b)), and then toward more advanced ideas in the area of quantum algorithms (Section IV (/qstage/#/tutorial?sectionId=8443c4f713521c10b1a56a533958286b)) and quantum error correction (Section V (/qstage/#/tutorial?sectionId=bfd2a30ad6c5c915da3a696d76c474d7));
- the **Quantum Composer,** which is a graphical user interface where you can create your very own quantum circuit (which we call a **quantum score),** much like a composer composes a music score;
- a simulator that can test your quantum scores;
- access to an actual quantum processor running in an IBM Quantum Computing lab, where your quantum scores will be executed; and
- a **quantum community** where your quantum scores, ideas, and experiences can be shared and discussed.

Please note that the IBM Quantum Experience is a “living experiment,” and we are constantly making updates to it. We hope our users will help us refine its functionality and improve the overall interface by providing feedback. In the community forum, please let us know your thoughts for improvement – and please share any cool scores and results you come across! If you find any bugs, please report them with our bug tracker, accessible via the little bug icon in the lower right-hand corner of each page.

To make sure everyone has a chance to use the real device in our lab via the Cloud, we have established a **Units** currency system. If you have joined the IBM Quantum Experience as a **Standard User,** you have full access to our simulation capabilities and to previously-run cached results from the real device and a small number of Units to run real experiments on the quantum processor hardware. Once you have read through the User Guide you will be rewarded with extra Units to run more real-time experiments. This system allows our experiment queue to run smoothly. When your Units are used up, you will be replenished once you have viewed the results of the completed execution. We also invite Standard Users to request an upgrade of your User status to **Expert User**, which provides access to more Units and other advanced features as we introduce them. Contact us and let us know why you would like to become an Expert User.

The quantum processor in our lab requires frequent calibration; during these short periods, you will receive a “Down for Calibration” notice and if we need to perform maintenance a “Down for Maintenance” message will be displayed. In both cases the simulation will be available for you to keep learning and designing new experiments.

For those who wish to jump right into creating your own experiments, you can skip right to the Quantum Composer (/qstage/#/tutorial?sectionId=75a85f7e14ae3fd4329ad5c3e59466ea&pageIndex=3). If you wish to improve your understanding of the quantum world first, please continue reading this User Guide, starting with the Quantum World (/qstage/#/tutorial?sectionId=75a85f7e14ae3fd4329ad5c3e59466ea&pageIndex=1)section on the next page. Use the numbers at the top and bottom of each page to navigate through each section.

Prepare yourself for a wild and fascinating journey – and it all starts with a qubit.

We would like to acknowledge the work done under the IARPA Multi-Qubit Coherent Operations program, the Logical Qubits program, the LPS Quantum Characterization Validation & Verification program, and the IBM Research Frontiers Institute. The research performed in those programs has contributed to making this Quantum Experience possible.

Thank you,

Jay Gambetta, Jerry Chow, and the IBM Quantum team

The Weird and Wonderful World of the Qubit

The Quantum Bit (Qubit)

In this section you will meet the qubit. You will also see a bit of mathematical notation, including some concepts from linear algebra.
A qubit is a quantum system consisting of two levels, labeled $|0\rangle$ and $|1\rangle$ (here we are using Dirac’s bra-ket notation), and is represented by a two-dimensional vector space over the complex numbers $\mathbb{C}^2$. This means that a qubit takes two complex numbers to fully describe it. The computational (or standard) basis corresponds to the two levels $|0\rangle$ and $|1\rangle$, which correspond to the following vectors:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

The qubit does not always have to be in either $|0\rangle$ or $|1\rangle$; it can be in an arbitrary quantum state, denoted $|\psi\rangle$, which can be any *superposition* $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, of the basis vectors. The superposition quantities α and β are complex numbers; together they obey $|\alpha|^2 + |\beta|^2 = 1$.
Interesting things happen when quantum systems are measured, or observed. Quantum measurement is described by the Born rule (https://en.wikipedia.org/wiki/Born_rule). In particular, if a qubit in some state $|\psi\rangle$ is measured in the standard basis, the result **0** is obtained with probability $|\alpha|^2$, and the result **1** is obtained with the complementary probability $|\beta|^2$. Interestingly, a quantum measurement takes any superposition state of the qubit, and projects it to either the state $|0\rangle$ or the state $|1\rangle$ with a probability determined from the parameters of the superposition.
What we have described here is the abstract notion of a qubit. The prototype quantum computer you interact with in the IBM Quantum Experience uses a physical type of qubit called a *superconducting transmon qubit*, which is made from superconducting materials such as niobium and aluminum, patterned on a silicon substrate.
Physically, for this superconducting qubit to behave as the abstract notion of the qubit, the device must be at drastically low temperatures. In the IBM Quantum Lab, we keep the temperature so cold (15 milliKelvin in a dilution refrigerator) that there is no ambient noise or heat to excite the superconducting qubit. Once our system has gotten cold enough, which takes several days, the superconducting qubit reaches equilibrium at the ground state $|0\rangle$.

To get a sense for what “ground state” means, try running the first score file below in simulation mode (or look at some cached runs on the real device). Here, the qubit is initially prepared in the ground state $|0\rangle$, then is followed by the standard measure. From your execution results, you should find in the ideal case (and with very high probability for the cached runs) that the qubit is still in the ground state. In the real device runs, you can observe that there is some error, with some shots giving a $|1\rangle$ instead, which is due to imperfect measurements and some residual heating of the qubit.

The output for every score you run will be in the My Scores tab. Click the little bar graph icon next to the time stamp for your quantum score to see the results (if the results are not yet ready, or if there has been an error, you will see a yellow or red symbol on the bar graph icon). This will take you to the Results screen, where you can view the outcomes.

1Q Measurement

(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=f03e4cfa53ccf70b3bea5e0955b6f458&sharedCode=true)
Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=f03e4cfa53ccf70b3bea5e0955b6f458&sharedCode=true)

Excited State and Pauli Operators

As you may have guessed, a qubit does more than just sit around in the $|0\rangle$ ground state. To put it into the $|1\rangle$ *excited* state, we need a *quantum gate*. In this section we will introduce gates, and how to use them in the Composer.

Quantum gates, or operations, are typically represented as matrices. A gate that acts on one qubit is represented by a 2×2 unitary matrix (https://en.wikipedia.org/wiki/Unitary_matrix). Since quantum operations must be reversible and preserve probability amplitudes, the matrices must be unitary. The result of the quantum gate is found by multiplying the matrix representing the gate with the vector representing the quantum state.

$|\psi'\rangle = U|\psi\rangle$ where $U^\dagger U = 1$ ($A^\dagger$ represents the complex conjugation and transpose of any matrix A).

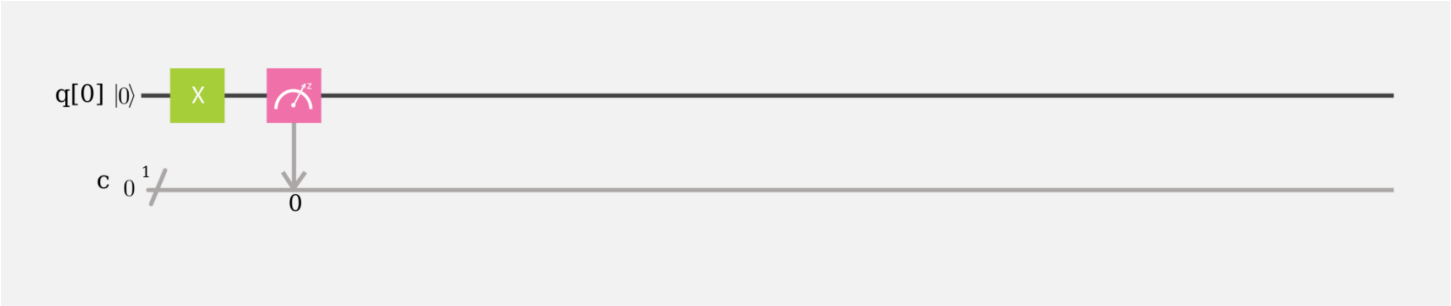
A common group of gates, known as the Pauli Operators (https://en.wikipedia.org/wiki/Pauli_matrices), are represented by the matrices

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \qquad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \qquad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

The Pauli X gate is known as an X_π-rotation. It takes $|0\rangle \rightarrow X|0\rangle = |1\rangle$; in other words, it flips the zero to a one, or vice versa (this is why it is also commonly referred to as a bit-flip). Try the Pauli X gate in the Composer, using the score file below. Did you find that, unlike in the example on the previous page, the qubit ended up in the excited state $|1\rangle$ with high probability? Like before, any deviation from the excited state is likely due to decoherence and imperfect measurements.

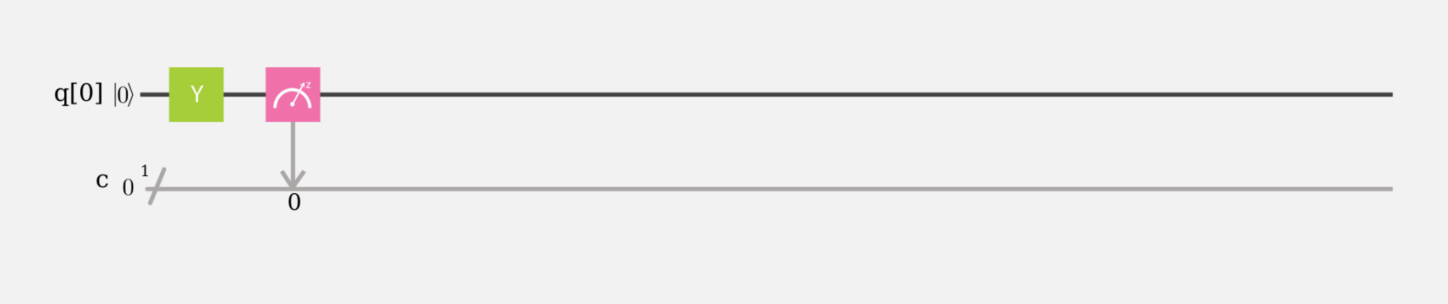
In the other examples below, explore what the Pauli Operators do. What do you get when you try a Y or Z gate? Did you find that Y gave you an excited state and Z did not do anything?

1Q Pauli X



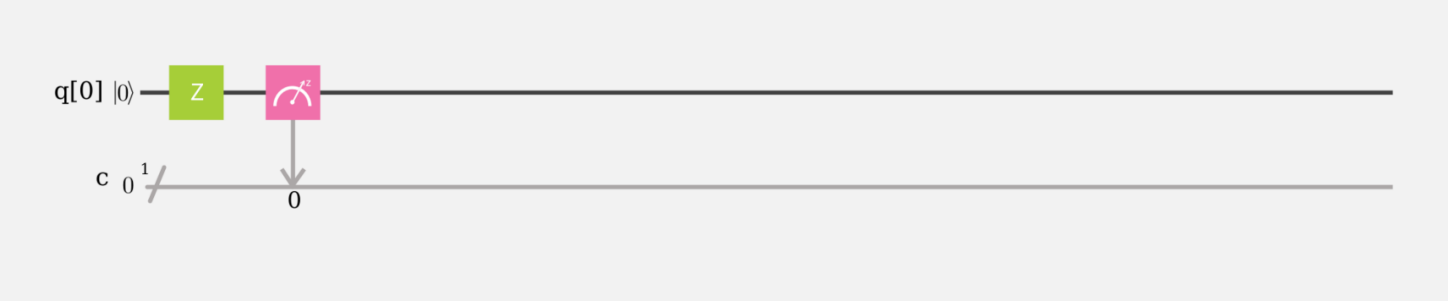
(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=d45a9317f90b97fddfda9f15f31eb14f&sharedCode=true)
Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=d45a9317f90b97fddfda9f15f31eb14f&sharedCode=true)

1Q Pauli Y



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=d45a9317f90b97fddfda9f15f33478de&sharedCode=true)
Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=d45a9317f90b97fddfda9f15f33478de&sharedCode=true)

1Q Pauli Z



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=8afcf3a276348f6c37ee3246a5e3c561&sharedCode=true)
Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=8afcf3a276348f6c37ee3246a5e3c561&sharedCode=true)

Superposition

Now that we have the $|0\rangle$ and $|1\rangle$ states under our belt, let's explore superposition, which is the concept that adding quantum states together (similar to overlaying two waves) results in a new quantum state. To make superpositions, we will expand our set of gates to include $\{H, S, S^\dagger\}$. In the Quantum Composer, these are the blue set of gates. They are represented by the matrices

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \qquad S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} \qquad S^\dagger = \begin{pmatrix} 1 & 0 \\ 0 & -i \end{pmatrix}$$

In the first score below, we apply H, known as the Hadamard gate, on one of the qubits that has been prepared in the $|0\rangle$ state; we then follow up with the standard measurement. Run the circuit and observe the result. The qubit should spend half its time in the $|0\rangle$ state and the other half in the $|1\rangle$ state. Before the measurement forced the qubit to choose a final state, the qubit was in both states at once. This superposition effect is often over-emphasized, and has become part of an often-misused analogy that a quantum computer is more powerful because it somehow "does everything at once."

So what is happening mathematically?

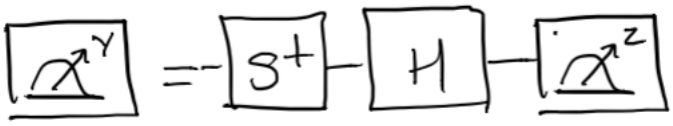
When we apply the H gate to $|0\rangle$, we make the state $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ (we've performed a size-2 discrete Fourier transform). This is the standard representation of a superposition state. We can define the state $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$, which with $|+\rangle$ forms a new basis called the *diagonal* (or conjugate) basis. It is made using the second circuit below. The H creates the above superposition, and then the Z, which previously did nothing, now flips the phase ($|1\rangle$ to $-|1\rangle$). When you run this circuit you will find that, like before, the outcomes are equal. Different states give the same outcomes!

To tell the difference between these states, we need to measure in the diagonal basis. In our experiments we cannot physically change the measurement; however, we can effectively change the measurement using gates before measurement. To measure in the diagonal basis, the standard basis (Z) is rotated to the diagonal basis (X) with a Hadamard gate before the measurement.

Try Superposition (+) X Measurement and Superposition (-) X Measurement below in simulation. You should find that 100% of the time the outcome is 0 and 1 respectively. That is, if we make a measurement in the standard basis, the outcome is completely uniform – but in the diagonal basis, it has deterministic outcome. No measurement can distinguish all four kinds of states $|0\rangle$, $|1\rangle$, $|+\rangle$, $|-\rangle$. This is not a limitation of the measurement, but a fundamental consequence of the uncertainty principle. (This limitation gives rise to the possibility of quantum money and quantum cryptography.)

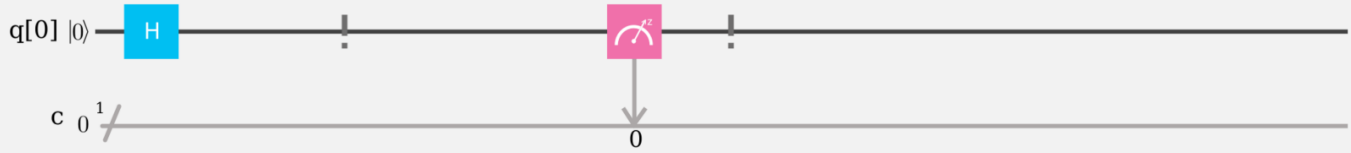
A third commonly-used basis is the *circular* (or Y) basis: $|\circlearrowright\rangle = \frac{1}{\sqrt{2}}(|0\rangle + i|1\rangle)$, $|\circlearrowleft\rangle = \frac{1}{\sqrt{2}}(|0\rangle - i|1\rangle)$. To make the $|\circlearrowright\rangle$ state we will use S, the phase gate. This gate applies a complex phase to $|1\rangle$. By applying an H followed by an S gate to the $|0\rangle$ state, you can obtain $|\circlearrowright\rangle$. Try to figure out how to get the $|\circlearrowleft\rangle$ on your own.

Like the above example, measurement in the standard basis will not give you different statistics. Even measurement in the diagonal basis will be random. To measure in this basis, we must rotate the standard basis (Z) to the circular basis (Y). To do this, use an $S^\dagger$ followed by H before your measurement.



Try out the last example. It should give you 1, since it is a measurement of $|\circlearrowright\rangle$ in the circular basis.

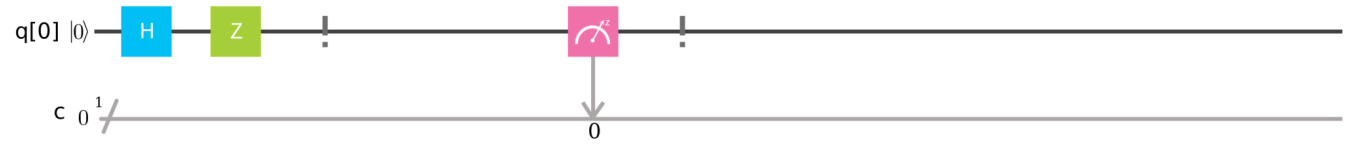
Superposition (+) Z-Measurement



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=5306b82998001797c6ef04345641bab8&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=5306b82998001797c6ef04345641bab8&sharedCode=true)

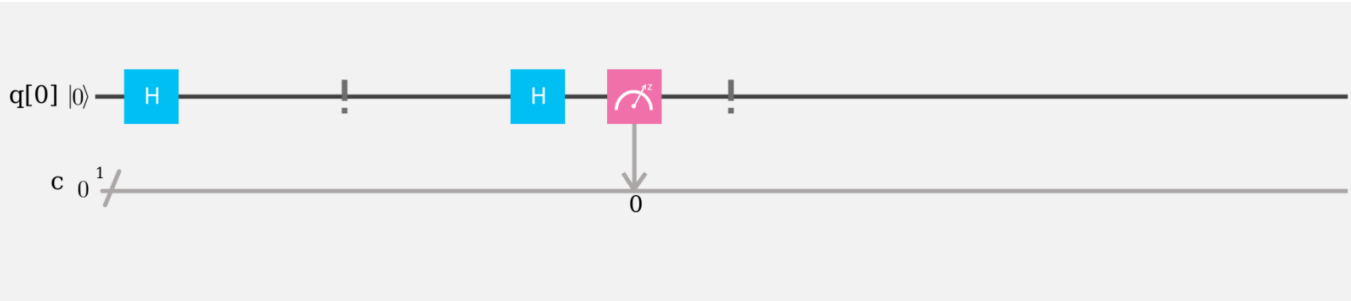
Superposition (-) Z-Measurement



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=5306b82998001797c6ef0434564767f0&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=5306b82998001797c6ef0434564767f0&sharedCode=true)

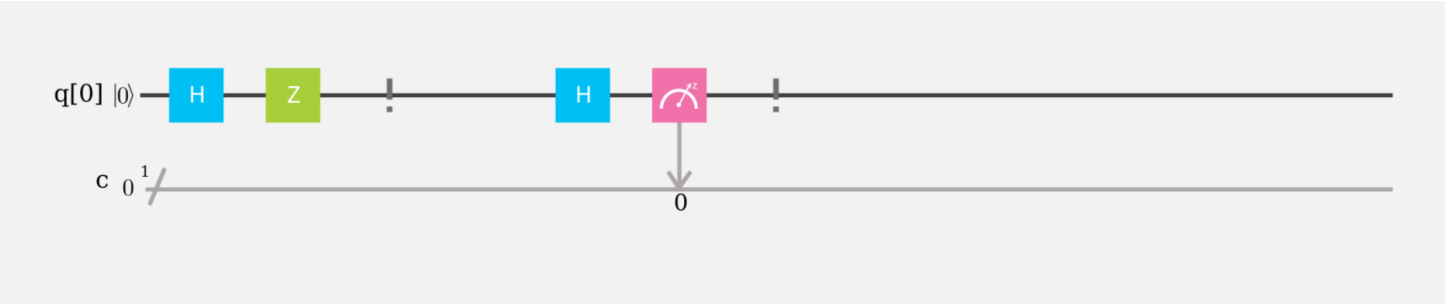
Superposition (+) X-Measurement



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=7c2bba34e2503c02aa981dcca8fa718f&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=7c2bba34e2503c02aa981dcca8fa718f&sharedCode=true)

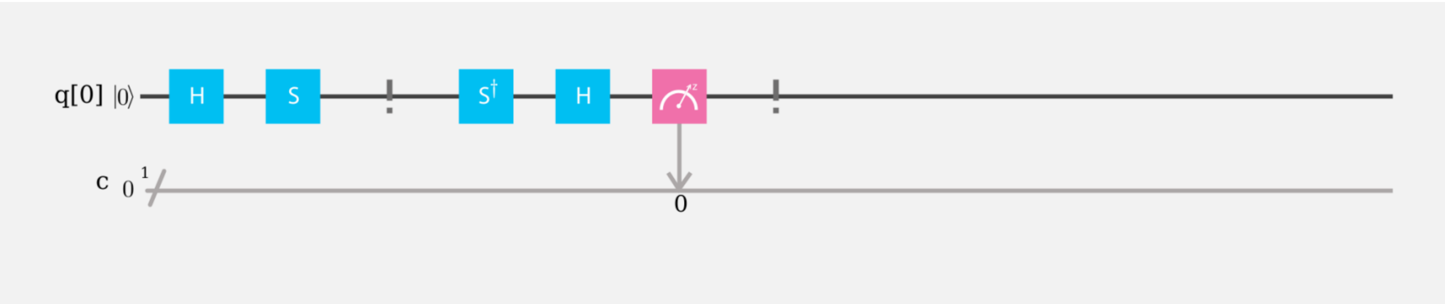
Superposition (-) X-Measurement



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=ba9448cba01a6e174b2cbccad444da1a&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=ba9448cba01a6e174b2cbccad444da1a&sharedCode=true)

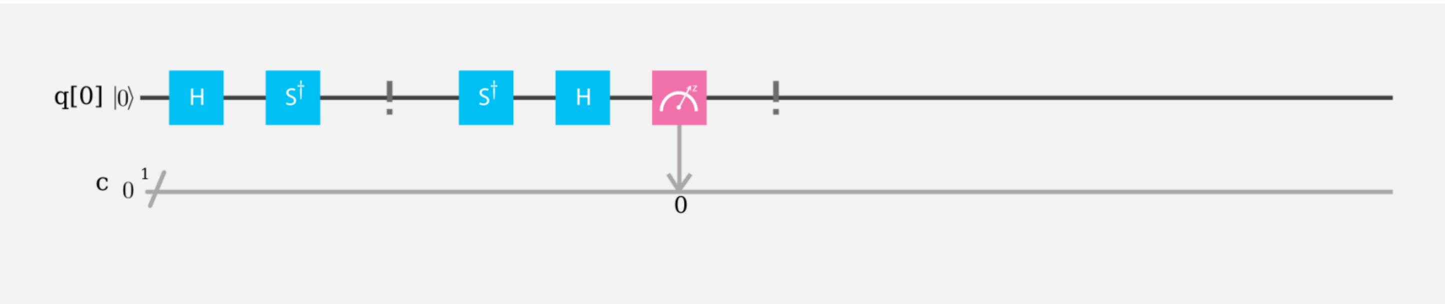
Superposition (+i) Y-Measurement



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=5306b82998001797c6ef0434561ebdeb&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=5306b82998001797c6ef0434561ebdeb&sharedCode=true)

Superposition (-i) Y-Measurement

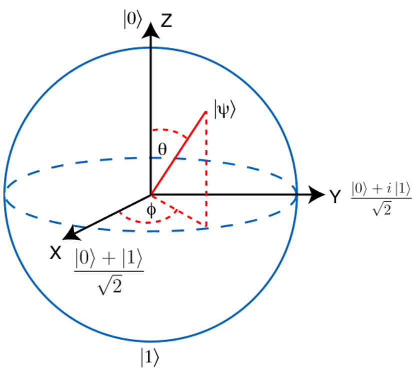


(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=89524da231758e94d5784382510c722d&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=89524da231758e94d5784382510c722d&sharedCode=true)

The Bloch Sphere

Probabilities in the standard basis are not enough to specify a quantum state because it cannot capture the phase of the superposition. A convenient representation for a qubit is the *Bloch sphere*; this depiction makes it easier to visualize qubit states and gates.



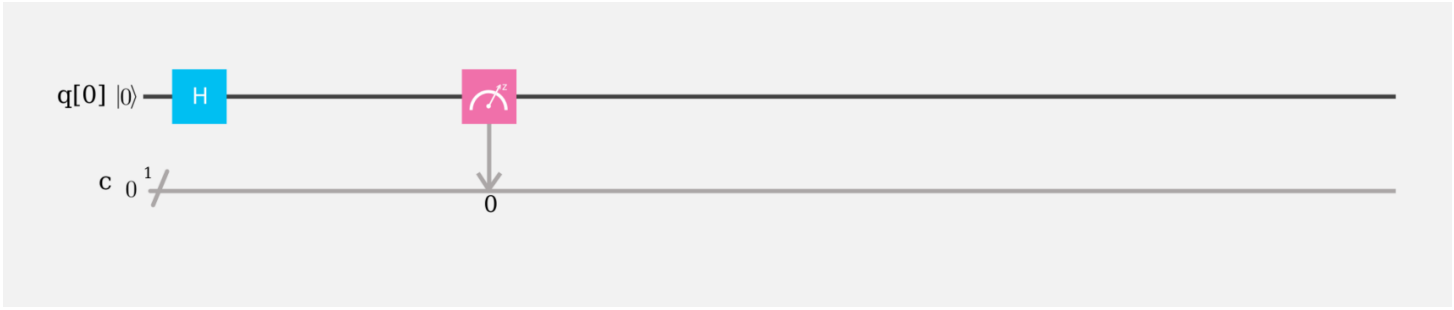
(

If we define a qubit state by $|\psi\rangle = \cos(\theta/2)|0\rangle + e^{i\phi}\sin(\theta/2)|1\rangle$, where θ and ϕ are defined in the picture, we see that there is a one-to-one correspondence between pure qubit states ($\mathbb{C}^2$) and the points on the surface of a unit sphere ($\mathbb{R}^3$). We can reconstruct an arbitrary unknown qubit state $|\psi\rangle$ by measuring the *Bloch vector*, whose vector components are the expectation values (https://en.wikipedia.org/wiki/Expectation_value_%28quantum_mechanics%29) of the three Pauli operators, given by $\langle X\rangle = \mathrm{tr}(|\psi\rangle\langle\psi|X)$, $\langle Y\rangle = \mathrm{tr}(|\psi\rangle\langle\psi|Y)$, and $\langle Z\rangle = \mathrm{tr}(|\psi\rangle\langle\psi|Z)$. The state is given by $|\psi\rangle\langle\psi| = (I + \langle X\rangle X + \langle Y\rangle Y + \langle Z\rangle Z)/2$.

Each expectation value $\langle Q\rangle$ can be obtained experimentally by first preparing the state, rotating the standard basis frame to lie along the corresponding axis Q, and making a measurement in the standard basis. The probabilities of obtaining the two possible outcomes 0 and 1 are used to evaluate the desired expectation value via $\langle Q\rangle = P(0) - P(1)$. As an example, let's look at measuring the expectation value of X, $\langle X\rangle = \mathrm{tr}(|\psi\rangle\langle\psi|X)$, depicted in the circuit below. Once $|\psi\rangle$ is prepared, we implement the gate H that exchanges Z to X, then we make a measurement in the standard basis. The desired expectation value is given by $\langle X\rangle = P(0) - P(1)$. Similarly, we can use the $S^\dagger - H$ gate to measure $\langle Y\rangle$.

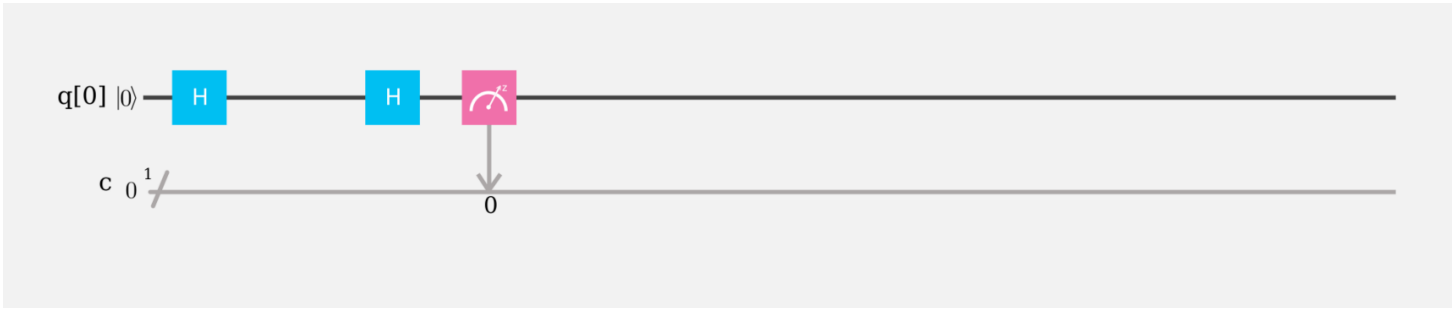
As you can see, a series of measurements is required to reconstruct the full Bloch vector, which then can be combined. This type of reconstruction is often referred to as state tomography (https://en.wikipedia.org/wiki/Quantum_tomography). Efficiently running routines for tomography requires running batches of quantum scores at a time, and can be set up by more advanced users through our API examples.

Superposition (+) Z-Measurement



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=1ead387800554184bd04529c736909b9&sharedCode=true)
Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=1ead387800554184bd04529c736909b9&sharedCode=true)

Superposition (+) X-Measurement



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=94a56f11d3a2e42f781d39b3d0f5f619&sharedCode=true)
Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=94a56f11d3a2e42f781d39b3d0f5f619&sharedCode=true)

Superposition (+) Y-Measurement

(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=576dbd3ecfc231148b777728c9bc2c03&sharedCode=true)
Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=576dbd3ecfc231148b777728c9bc2c03&sharedCode=true)

Decoherence

Real quantum computers must deal with decoherence (https://en.wikipedia.org/wiki/Quantum_decoherence), or the loss of information due to environmental disturbances (noise). The Bloch vector formalism we introduced in the previous section is sufficient to describe the state of the system under decoherence processes. The pure states (https://en.wikipedia.org/wiki/Quantum_state#Pure_states) we have studied so far have a Bloch vector of length 1, touching the surface of the Bloch sphere, and can be represented in density matrix form as $\rho = |\psi\rangle\langle\psi|$. Decoherence causes a change in our quantum states from pure to mixed states (https://en.wikipedia.org/wiki/Quantum_state#Mixed_states), which have a density matrix (https://en.wikipedia.org/wiki/Density_matrix) ρ that can be written as a sum over pure states

$$\rho = \sum_k p_k |\psi_k\rangle\langle\psi_k|$$

and a Bloch vector that sits inside the Bloch sphere

$$|\langle X\rangle|^2 + |\langle Y\rangle|^2 + |\langle Z\rangle|^2 < 1.$$

Energy relaxation and T_1

One important decoherence process is called *energy relaxation*, where the excited $|1\rangle$ state decays toward the ground state $|0\rangle$. The time constant of this process, T_1, is an extremely important figure-of-merit for any implementation of quantum computing, and one in which IBM has made great progress in recent years, ultimately leading to the prototype quantum computer you are now using. Experiment with the circuits below to see how adding many repetitions of additional do-nothing Idle Id gates (or Identity gates; these are gates that do nothing but wait) before measurement causes the state to gradually decay towards $|0\rangle$.

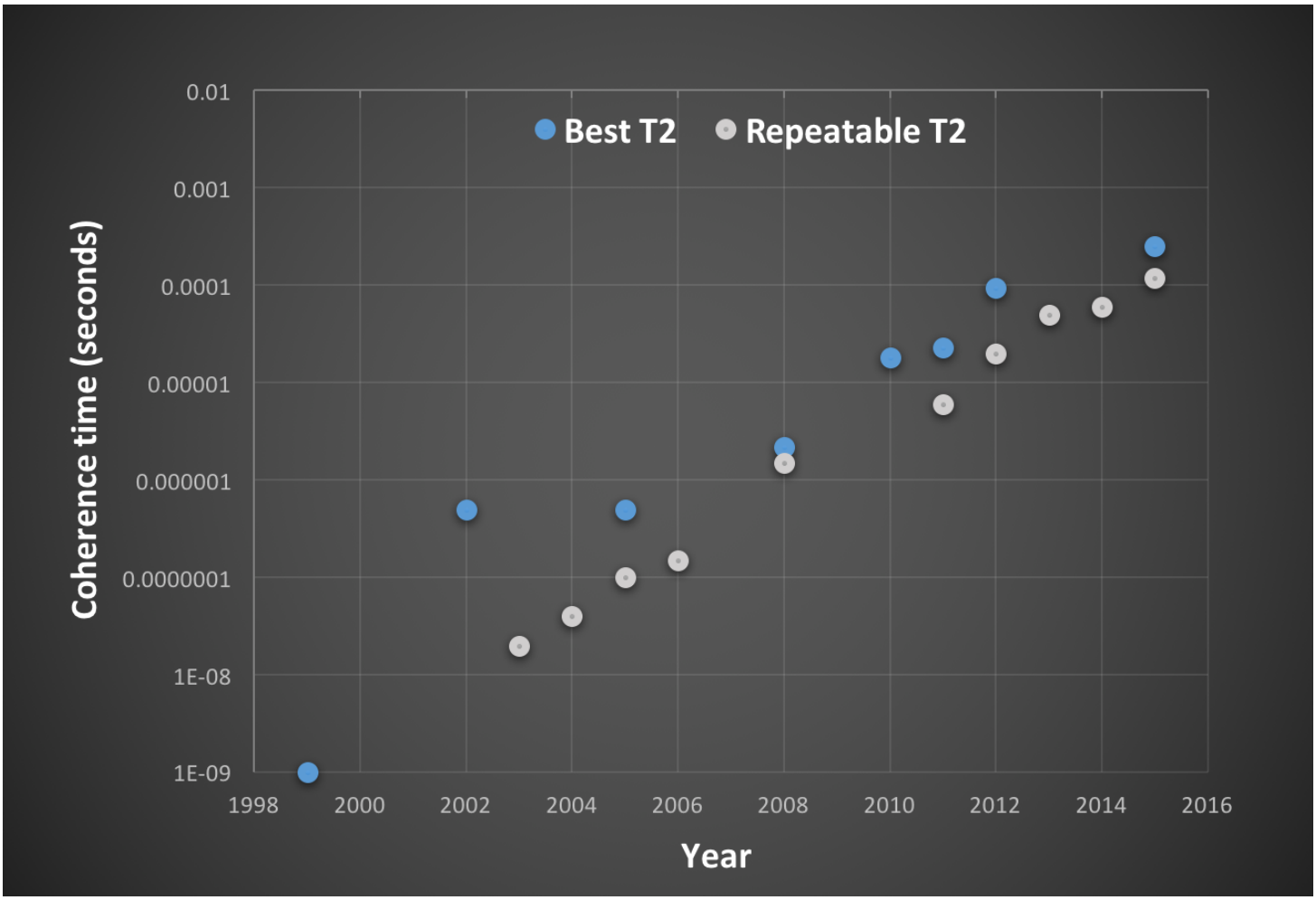
Dephasing and T_2

Dephasing is another decoherence process, and unlike energy relaxation, it affects only superposition states. It can be understood solely in a quantum setting as it has no classical analog. The time constant T_2 includes the effect of dephasing as well as energy relaxation, and is another crucial figure-of-merit. Again, IBM has some of the world's best qubits by this metric. Experiment with the circuits below to see that with a Ramsey experiment, essentially separating two $\pi/2$ rotations with an Idle, there is a decay in

the final expected excited state signal. When pointing along the equator, the qubit is subjected to more decay channels than when it starts in the computational state $|1\rangle$.

Progress in decoherence with superconducting qubits

Because T_2 is such an important quantity, it is interesting to chart how far the community of superconducting qubits has come over the years. Here is a graph depicting T_2 versus time.

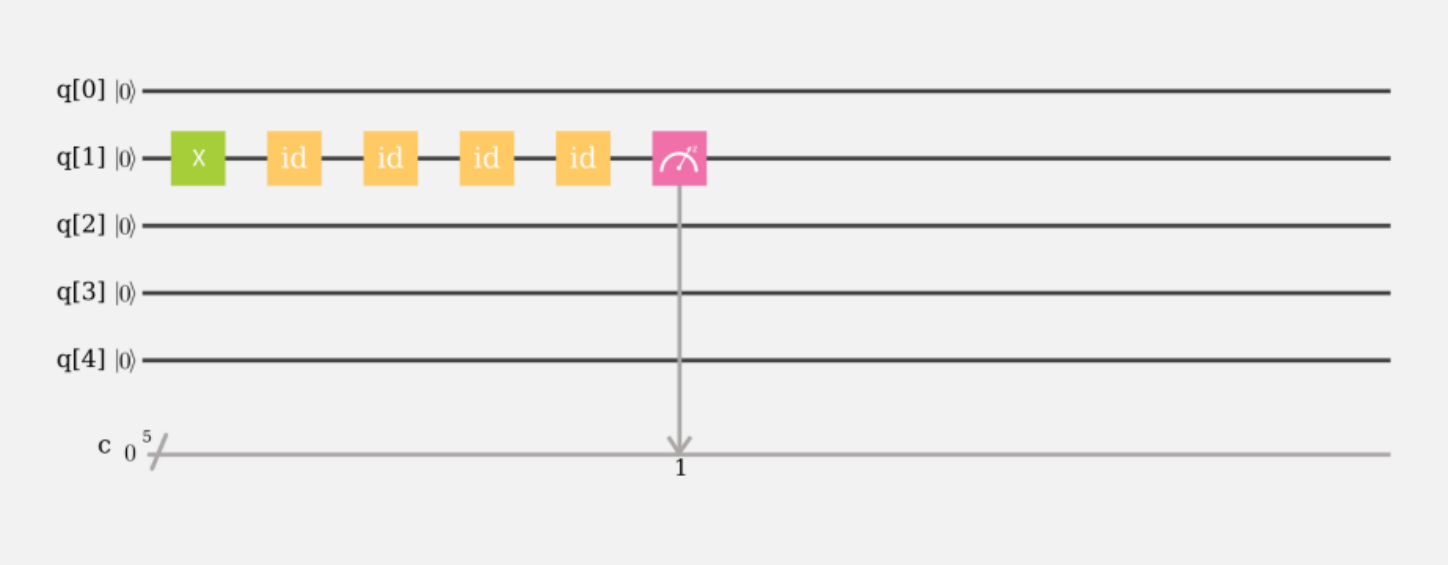


Excited state (No idle)

(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=f0c8e1f96638ef39ca67a6378231d4a4&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=f0c8e1f96638ef39ca67a6378231d4a4&sharedCode=true)

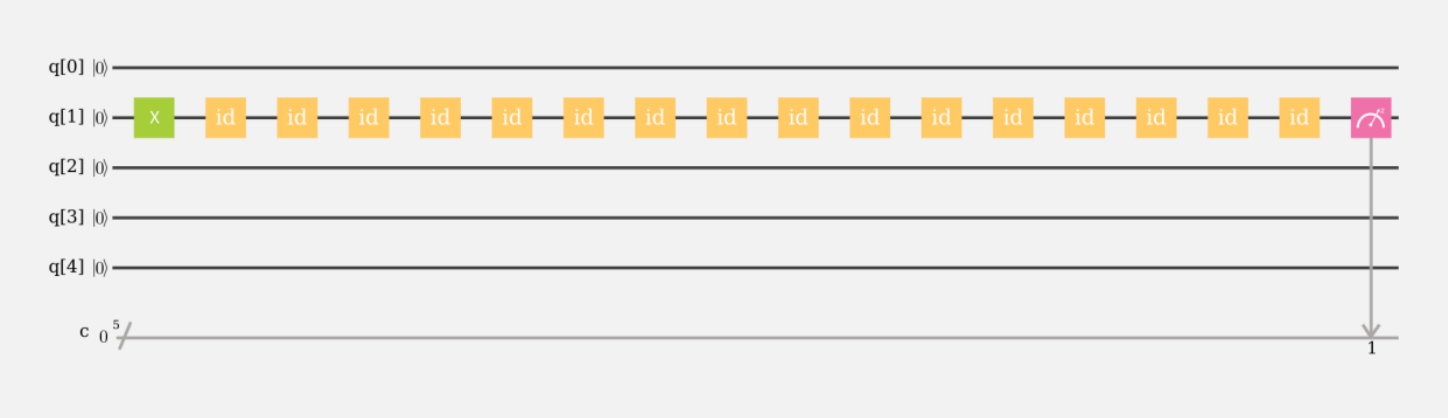
Excited state (4 idle)



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=f0c8e1f96638ef39ca67a63782490887&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=f0c8e1f96638ef39ca67a63782490887&sharedCode=true)

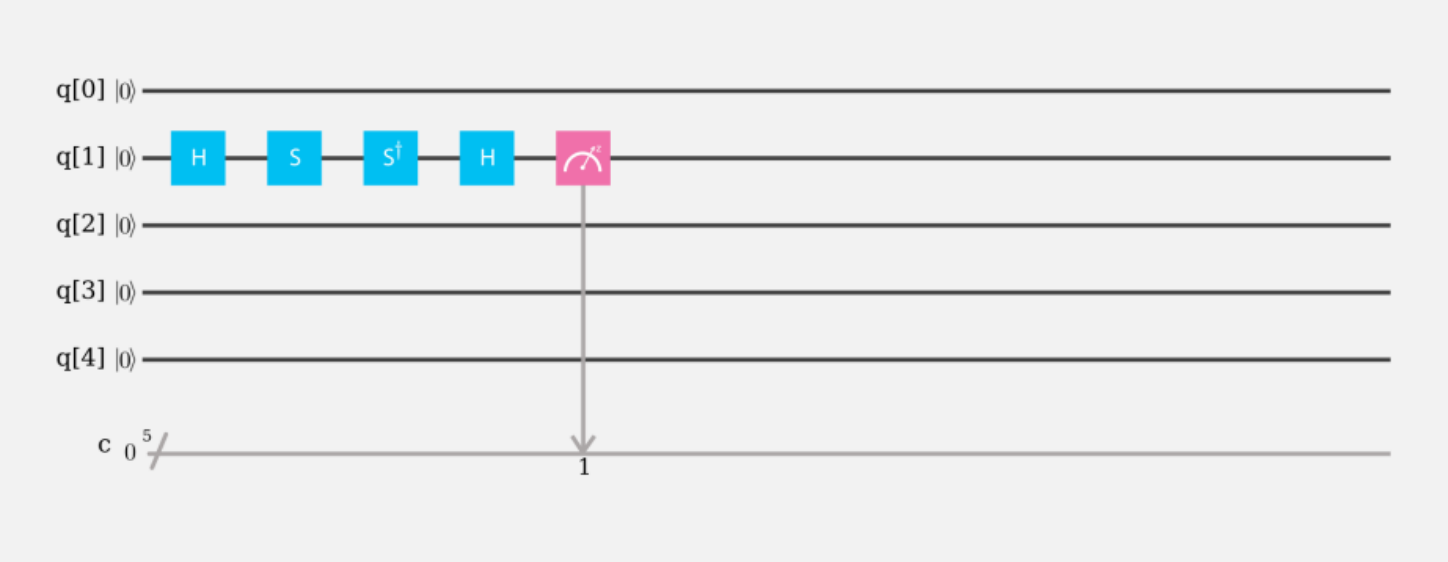
Excited state (16 idle)



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=8d36692e07e55a8ef2688a910e62a9b4&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=8d36692e07e55a8ef2688a910e62a9b4&sharedCode=true)

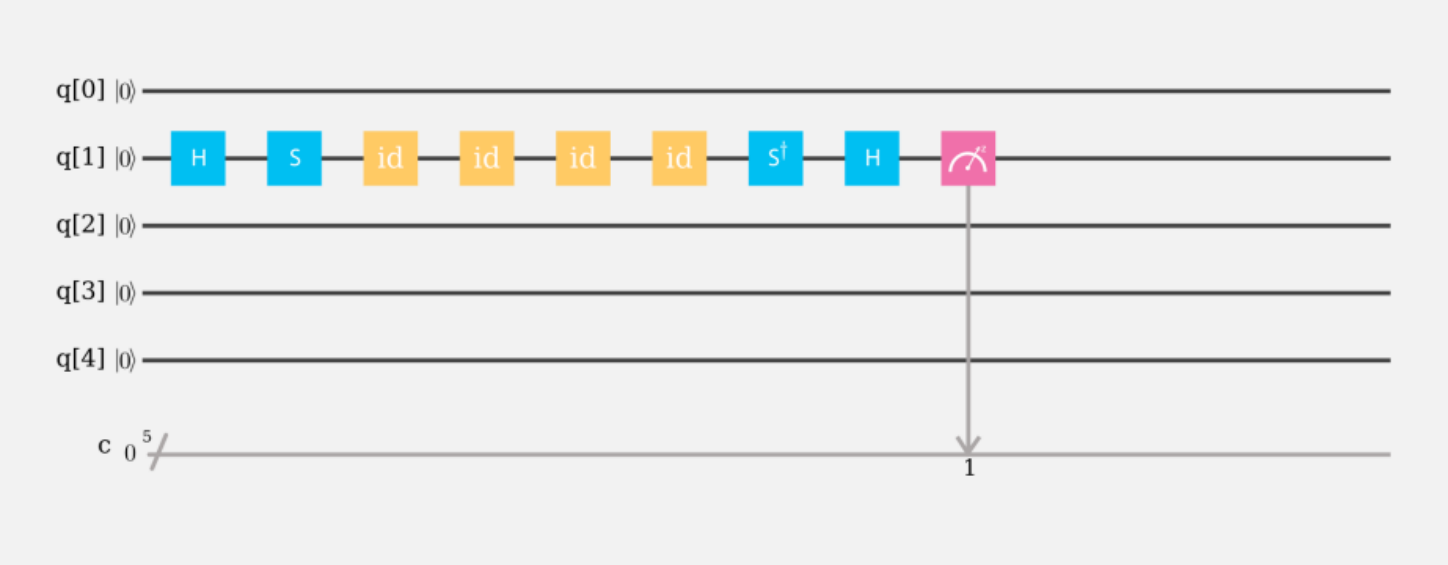
Ramsey (no idle)



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=117bf13ae0d48351f58fca70f11206d6&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=117bf13ae0d48351f58fca70f11206d6&sharedCode=true)

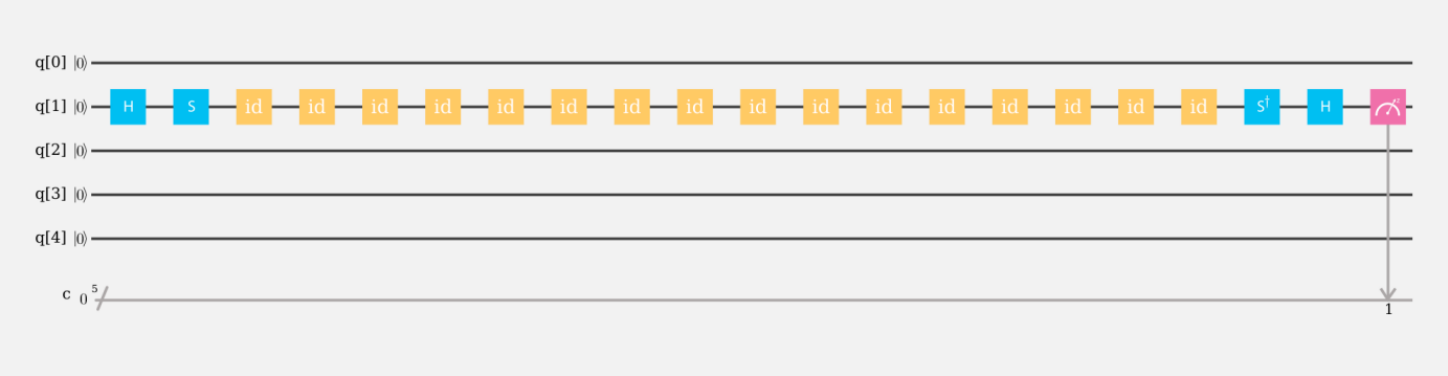
Ramsey (4 idle)



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=8d36692e07e55a8ef2688a910e938fe2&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=8d36692e07e55a8ef2688a910e938fe2&sharedCode=true)

Ramsey (16 idle)



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=b44dce9f84c53d75c991e794006a83f2&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=b44dce9f84c53d75c991e794006a83f2&sharedCode=true)

In this section we will introduce you to the qubit. A qubit (pronounced “cue-bit” and short for quantum bit) is the physical carrier of quantum information; in other words, It is the quantum version of a bit, and quantum computing revolves around it.

By the end of this section you will have a basic understanding of

- Qubits and their measurements
- Excited states
- Superpositions of qubit states
- The Bloch Sphere
- Decoherence

Multiple Qubits, Gates, and Entangled States

Multiple Qubits

Until this point, we have only considered a single qubit; let’s now consider more than one qubit in a system. The complex vector space of an n qubit system has a dimension equal to 2^n, which we denote $\mathbb{C}^{2^n}$. The standard basis is the set of all binary strings for $k \in \{0, 2^n - 1\}$. For example, the basis for two qubits is

$$\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\};$$

for three qubits,

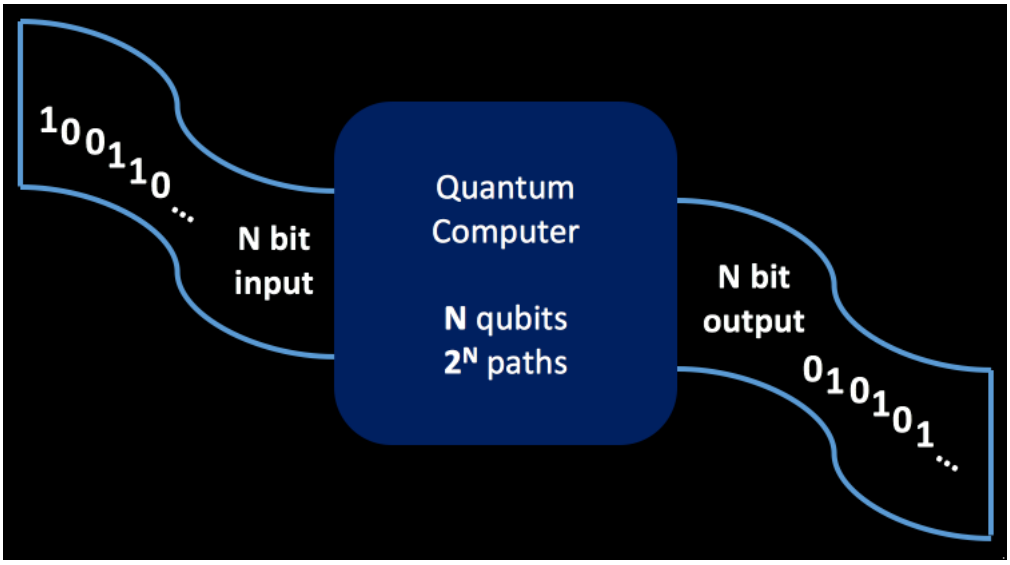
$$\{|000\rangle, |001\rangle, |010\rangle, |011\rangle, |100\rangle, |101\rangle, |110\rangle, |111\rangle\};$$

and for four qubits,

$$\{|0000\rangle, |0001\rangle, |0010\rangle, |0011\rangle, |0100\rangle, |0101\rangle, |0110\rangle, |0111\rangle, |1000\rangle, |1001\rangle, \ |1010\rangle, |1011\rangle, |1100\rangle, |1101\rangle, |1110\rangle, |1111\rangle\}.$$

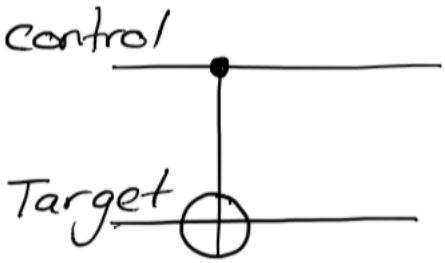
You may notice the number of terms is increasing exponentially. Try writing out all 32 terms for five qubits, then try writing them all out for 64 qubits! (This is like the famous wheat and chessboard problem (https://en.wikipedia.org/wiki/Wheat_and_chessboard_problem).) This exponential increase is one of the reasons why a large quantum system becomes impossible to simulate on a conventional computer.

Note, however, that the complexity is subtler than just the absolute number of terms growing exponentially. A classical computer that has n-bits also has 2^n possible configurations; at any one point in time, the computer state is in one and only one of these configurations. For example, a classical computer takes an n bit number, say 00000, and performs bit-wise operations on it, mapping the input though an n-bit intermediate state such as 00001, which is then output as another n-bit number 10101. Interestingly, a quantum computer also takes in an n-bit number and outputs an n-bit number; but because of the superposition principle and the possibility of entanglement, the intermediate state is very different. To describe it requires 2^n complex numbers, giving a lot more room for maneuvering.



As an example, try running the "Random Classical Circuit" provided below. It takes the initial state $|0\rangle^{\otimes n}$ and should produce the output $|10101\rangle$. By using X operations (NOTs) you can take the $|0\rangle^{\otimes n}$ to any classical state.

To do interesting things that make use of those many configurations in the quantum world, we need gates that perform conditional logic between qubits. The conditional gate we have provided is the Controlled-NOT, or CNOT. It is represented by the element



The CNOT gate's action on classical basis states is to flip (apply a NOT or X gate to) the target qubit if the control qubit is $|1\rangle$; otherwise it does nothing. The CNOT plays the role of the classical XOR gate, but unlike the XOR, it is a two-output gate in order to be reversible (https://en.wikipedia.org/wiki/Reversible_computing)(as all quantum gates must be). It is represented by the matrix

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

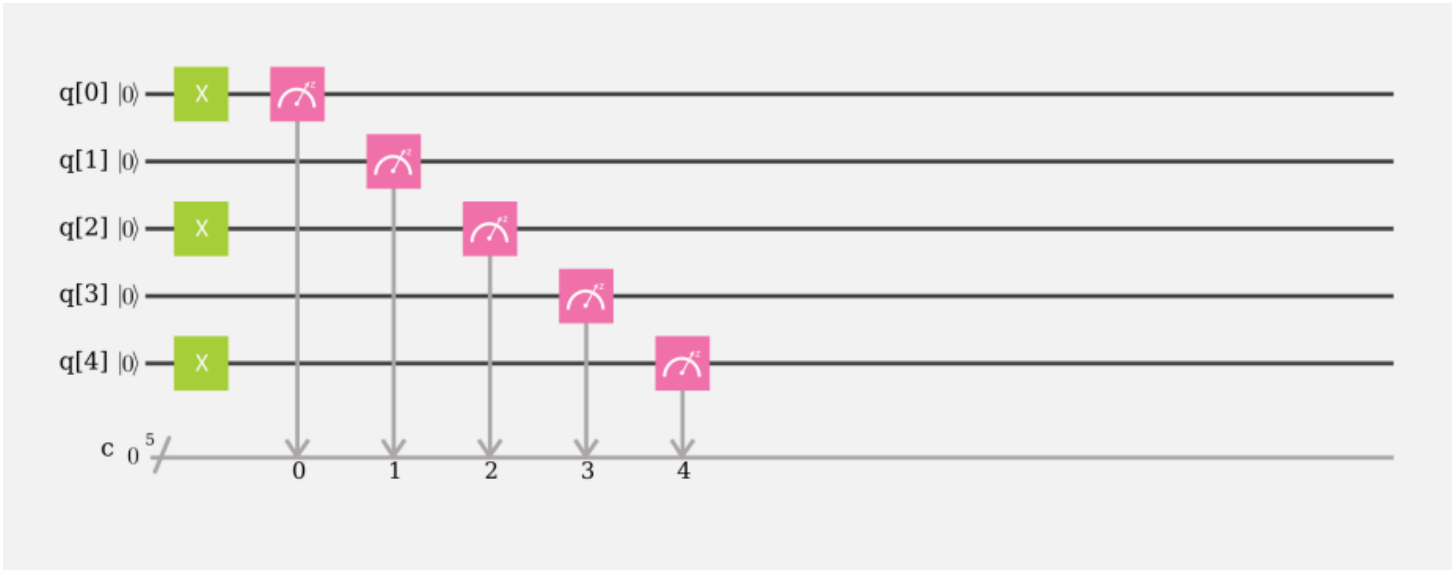
Try the "CNOT Circuits" example below with different input states. Note that the X gates have prepared the qubits in a different configuration for each example. Here you can see the results we got when we ran these experiments on the processor:

CNOT test: May 3rd, 2016 9:21 PM

	00	01	10	11
00	0.965	0.014	0.007	0.014
01	0.040	0.943	0.009	0.008
10	0.010	0.026	0.038	0.926
11	0.026	0.010	0.947	0.017

Finally, many quantum algorithms (https://en.wikipedia.org/wiki/Quantum_algorithm) start out by applying a Hadamard gate, because it maps :math:`n` qubits prepared in state $|0\rangle^{\otimes n}$ to a superposition of all 2^n orthogonal states with equal weight. Try out the five-qubit version. You should see that it has made a quantum sphere that points in all directions with a small weight $1/(2^5)$. Try adding the CNOT gate to make your own complex quantum states. In the next sections we will show you how quantum computers take advantage of a certain peculiarity known as *entangled* states.

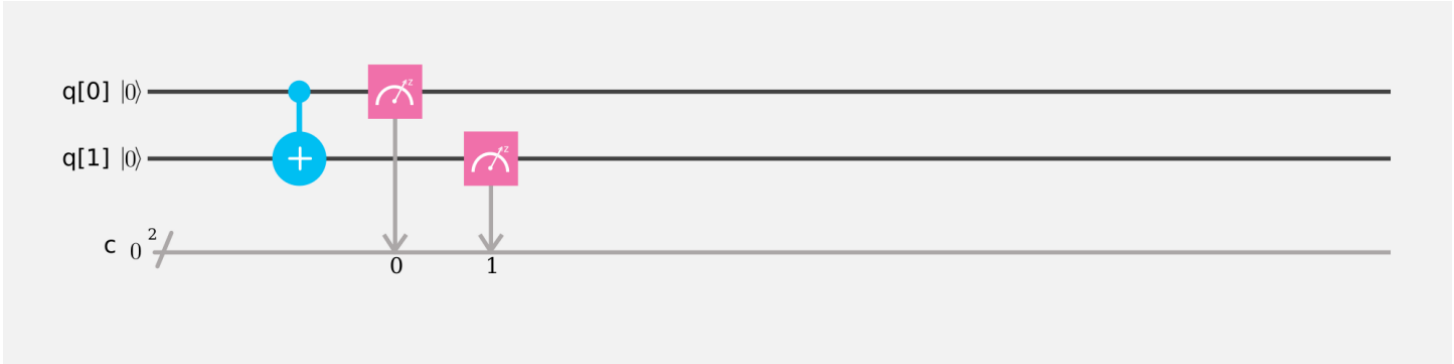
A Random Classical Circuit



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=eb66af75098f190ea9e7c6f7df658b1b&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=eb66af75098f190ea9e7c6f7df658b1b&sharedCode=true)

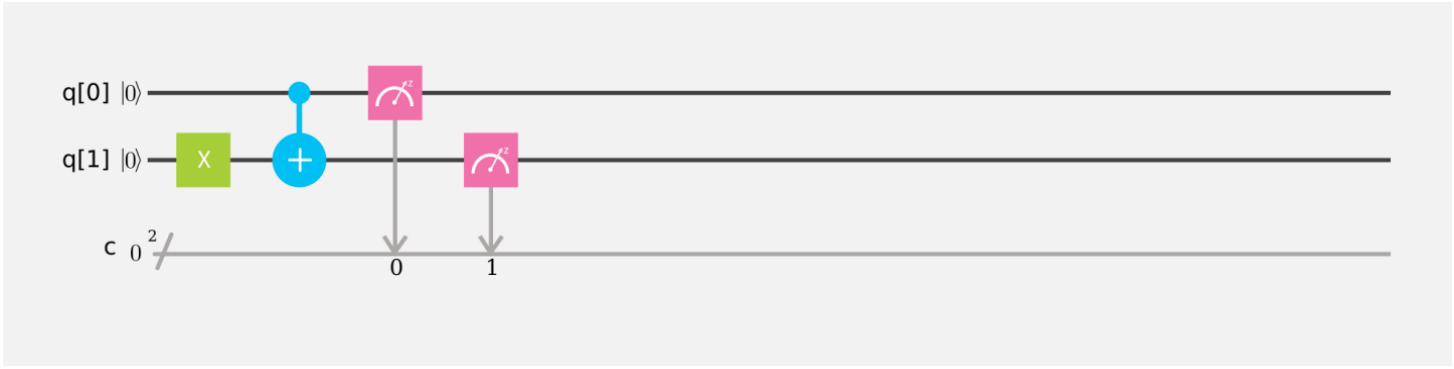
CNOT (input 00)



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=a95f11718a78d8ff64bdd87cecc8ba63&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=a95f11718a78d8ff64bdd87cecc8ba63&sharedCode=true)

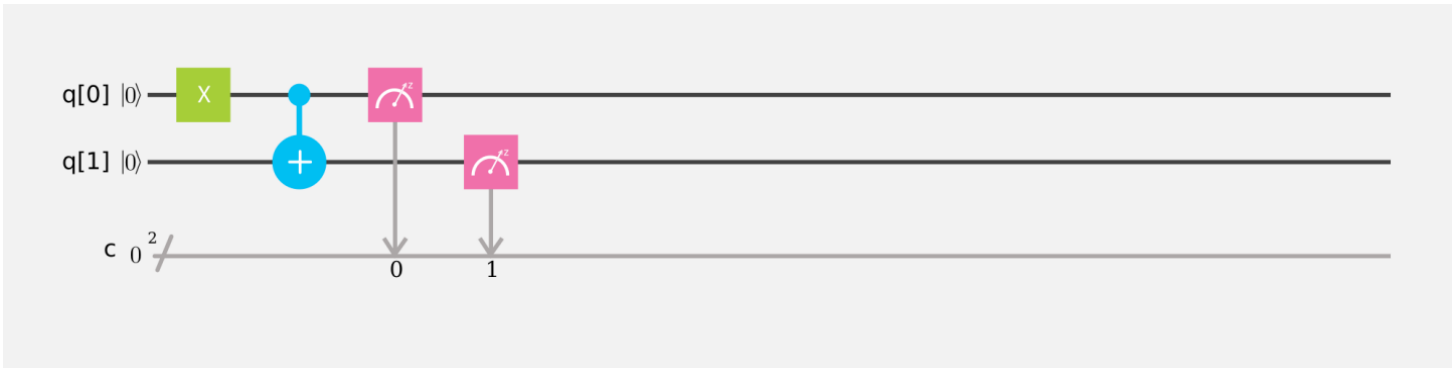
CNOT (input 01)



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=6c3621d910c99f3d4670a5ea5285e9d9&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=6c3621d910c99f3d4670a5ea5285e9d9&sharedCode=true)

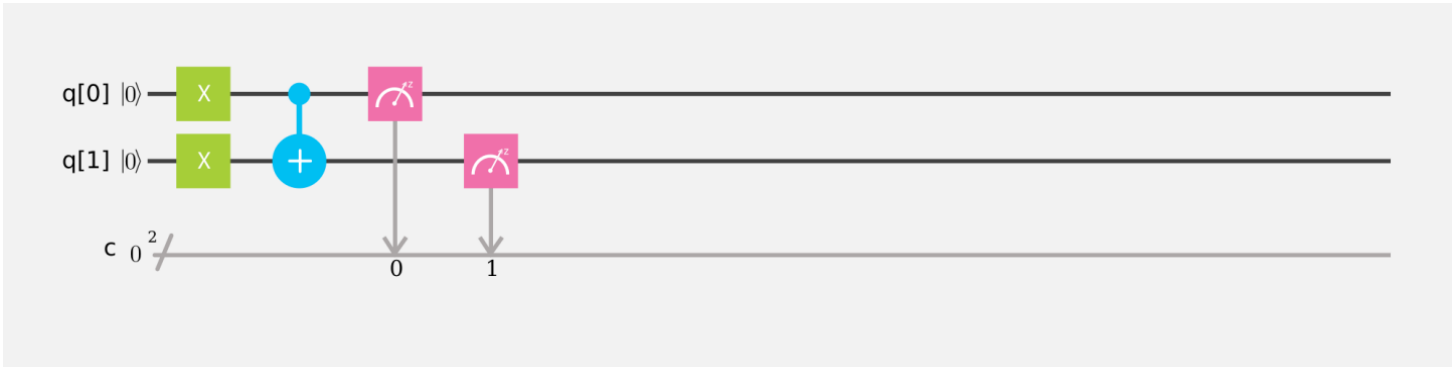
CNOT (input 10)



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=6dcb75a9dd05dca9f0504ec02153c124&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=6dcb75a9dd05dca9f0504ec02153c124&sharedCode=true)

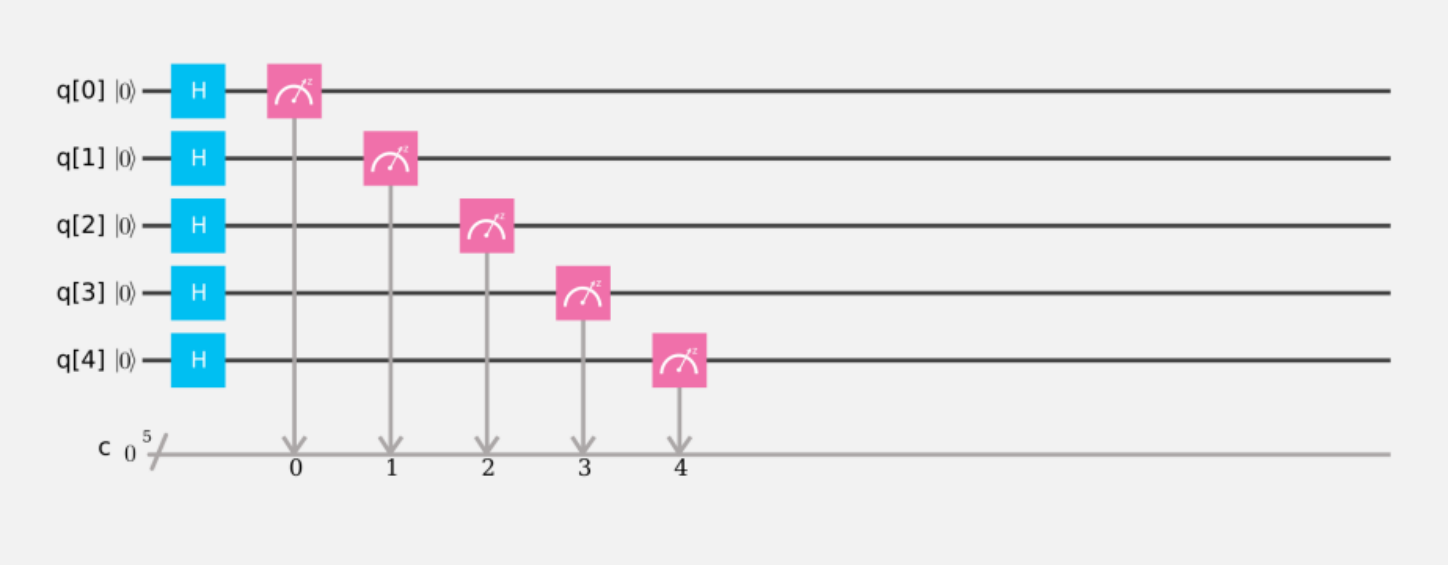
CNOT (input 11)



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=6dcb75a9dd05dca9f0504ec0216cd1b6&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=6dcb75a9dd05dca9f0504ec0216cd1b6&sharedCode=true)

5Q Complete Superposition Circuit



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=e21164be4db4b6f5698f254a300838ec&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=e21164be4db4b6f5698f254a300838ec&sharedCode=true)

Non-Clifford Gates

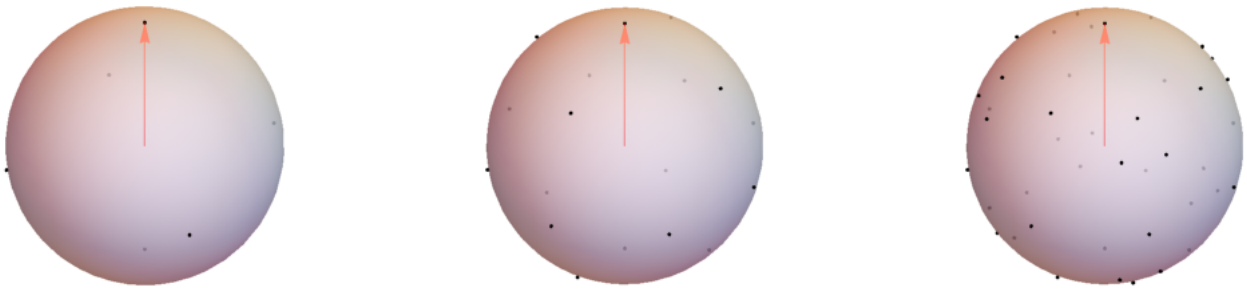
The most general operation that a quantum computer can perform is a unitary matrix in 2^n dimensions. A finite set of gates that can approximate any unitary matrix arbitrarily well is known as a universal gate set (https://en.wikipedia.org/wiki/Quantum_gate#Universal_quantum_gates). This is similar to how certain sets of classical logic gates, such as {AND, NOT}, are functionally complete (https://en.wikipedia.org/wiki/Functional_completeness) and can be used to build any Boolean function.

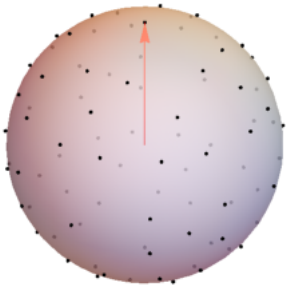
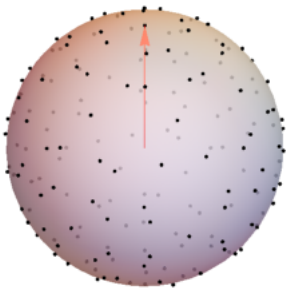
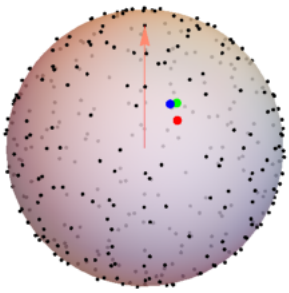
Up to this point, all the gates we have discussed (X, Y, Z, H, S, $S^\dagger$, and CNOT) are members of a special group of gates known as the Clifford group. These gates can be simulated efficiently on a classical computer (see the Gottesman-Knill (https://en.wikipedia.org/wiki/Gottesman%E2%80%93Knill_theorem) theorem). Therefore, the Clifford group is not universal. It cannot harness the full power of quantum computation; for that, we must include at least one non-Clifford gate in our circuits.

Any unitary matrix can be written as a combination of single- and two-qubit gates [*Barenco et al., 1995* (http://journals.aps.org/pra/abstract/10.1103/PhysRevA.52.3457?cm_mc_uid=43781767191014577577895&cm_mc_sid_50200000=1460741020)]. (This is unlike classical reversible computing, where 3-bit gates such as Toffoli (https://en.wikipedia.org/wiki/Toffoli_gate) are additionally required for functional completeness.) It turns out that adding almost any non-Clifford gate to single-qubit Clifford gates and CNOT gates is universal. There are several popular choices for non-Clifford gates, but we implement T as well as $T^\dagger$. These are given by

$$T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix} \qquad T^\dagger = \begin{pmatrix} 1 & 0 \\ 0 & e^{-i\pi/4} \end{pmatrix}$$

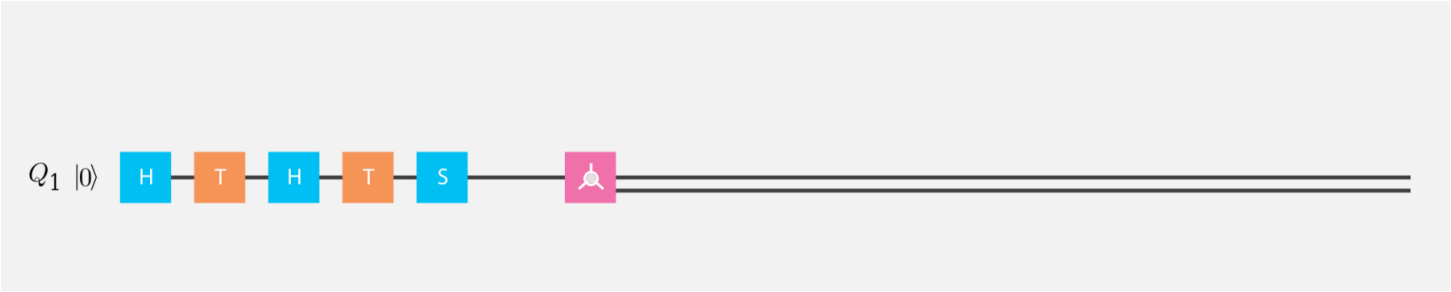
The T gate essentially makes it possible to reach all different points of the Bloch sphere. We can see that by increasing the number of T-gates in our circuit (the so-called *:math:`T` -depth*) we start to cover the Bloch sphere more densely with states we can reach. The following figures depict the attainable states by increasing T-depth from 0, 1, ... up to 5. In the final Bloch sphere for T-depth 5, we have highlighted a few points in red, green, and blue, which correspond to the Clifford+:math:*T* scores given below. Run these circuits to see if you end up at those points!



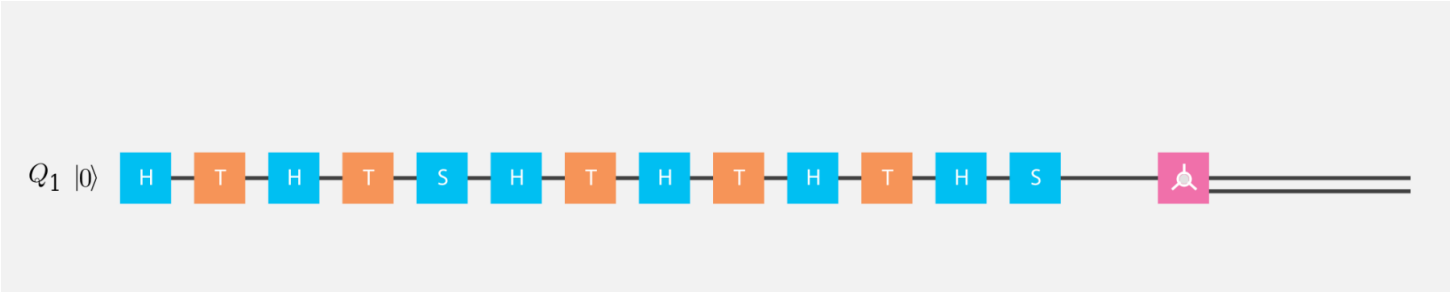
T-Depths are 0, 1, 2, 3, 4, and 5.

Red State



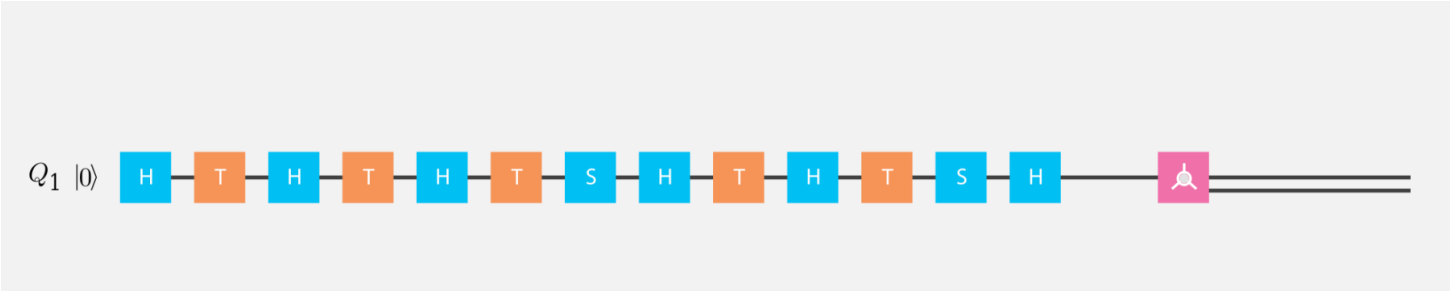
(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=4c0c46c51bc95c280e3cecc7140121c0&sharedCode=true)
Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=4c0c46c51bc95c280e3cecc7140121c0&sharedCode=true)

Green State



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=4c0c46c51bc95c280e3cecc71432020c&sharedCode=true)
Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=4c0c46c51bc95c280e3cecc71432020c&sharedCode=true)

Blue State

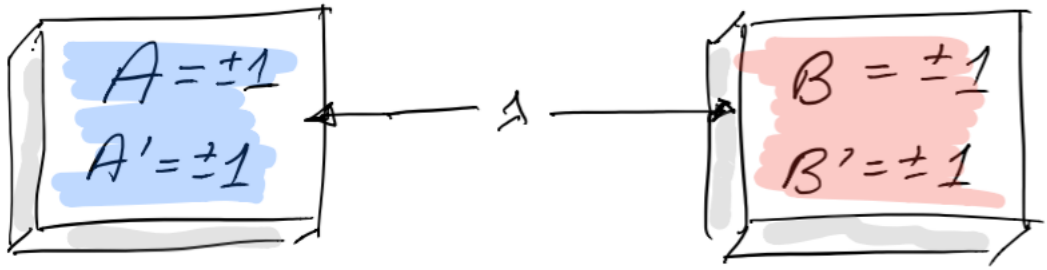


(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=82f8e7eaf1640aed285a4e982a12571f&sharedCode=true)
Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=82f8e7eaf1640aed285a4e982a12571f&sharedCode=true)

Entanglement and Bell Tests

One of the infamous counterintuitive ideas of quantum mechanics is that two systems that appear too far apart to influence each other can nevertheless behave in ways that, though individually random, are too strongly correlated to be described by any classical local theory.

To understand this, we have outlined a simple Bell test experiment here. Imagine you have two systems (see blue and red systems below). Within each there are two measurements performed: A, A', B and B', that have outcomes 1 (or -1). Bell showed that if these measurements are chosen correctly for a given entangled state, the statistics can not be explained by any local hidden variable theory, and that there must be correlations that are beyond classical.



In 1969 John Clauser (https://en.wikipedia.org/wiki/John_Clauser), Michael Horne (https://en.wikipedia.org/w/index.php?title=Michael_Horne&action=edit&redlink=1), Abner Shimony (https://en.wikipedia.org/wiki/Abner_Shimony), and Richard Holt (https://en.wikipedia.org/w/index.php?title=Richard_Holt_%28physicist%29&action=edit&redlink=1)derived the following CHSH inequality $|\mathrm{C}| \leq 2$ where

$$C = \langle AB \rangle - \langle AB' \rangle + \langle A'B \rangle + \langle A'B' \rangle$$

and the correlated expectation is given by

$$\langle AB \rangle = P_r(1,1) + P(0,0) - P(0,1) - P(0,1)$$

with 0 giving outcome $+1$ and 1 giving outcome -1. A correlation of 1 means both observables have even parity, and a correlation of -1 means both observables have odd parity.

It is simple to show that this inequality must be true if the theory obeys the following two assumptions, *locality* and *realism*:

** Locality**: No information can travel faster than the speed of light. There is a hidden variable λ that defines all the correlations so that

$$\langle AB \rangle = \sum_{\lambda} P_r(\lambda) A(\lambda) B(\lambda)$$

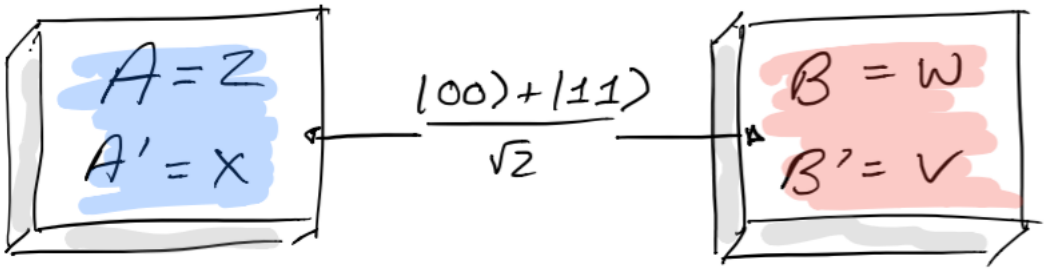
and C becomes

$$C = \sum_{\lambda} P(\lambda) \left[A(\lambda)\left(B(\lambda) - B'(\lambda)\right) + A'(\lambda)\left(B(\lambda) + B'(\lambda)\right) \right]$$

Realism: All observables have a definite value independent of the measurement (+1 or -1). This implies that either $|B(\lambda) + B'(\lambda)| = 2$ (or 0) while $|B(\lambda) - B'(\lambda)| = 0$ (or 2) respectively. That is, $|\mathrm{C}| = 2$, and noise will only make this smaller.

Perfectly reasonable, right? However, as you see, $|C| > 2$. How is this possible? The above assumptions must not be valid, and this is one of those astonishing counterintuitive ideas one must accept in the quantum world. Before you launch the scores below, let's try to understand what is happening and how each observable is measured and combined to give $|C|$.

The Bell experiment we have provided uses the entangled state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ and the two measurements for system A are Z and X, while the two for B are $W = \frac{1}{\sqrt{2}}(Z + X)$ and $V = \frac{1}{\sqrt{2}}(Z - X)$.

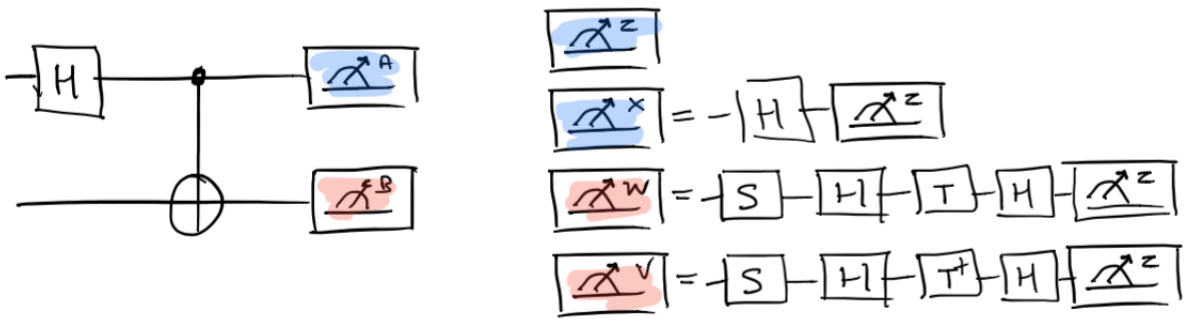


For an ideal implementation the four correlated expectation values give,

$\langle ZW \rangle = 1/\sqrt{2}$ $\langle ZV \rangle = 1/\sqrt{2}$ $\langle XW \rangle = 1/\sqrt{2}$ $\langle XV \rangle = -1/\sqrt{2}$

which gives $|C| = 2\sqrt{2}$.

To run this experiment with our hardware we need the following quantum score and 4 measurements.



In the first part of the experiment the qubits are initially prepared in the ground state $|00\rangle$. The H takes the first qubit to the equal superposition $\frac{|00\rangle+|10\rangle}{\sqrt{2}}$ and the CNOT gate flips the second qubit if the first is excited, making the state $\frac{|00\rangle+|11\rangle}{\sqrt{2}}$. This is the entangled state (commonly called a *Bell state*) required for this test. In the first experiment the measurements are of the observable Z and $W = \frac{1}{\sqrt{2}}(X+Z)$. To rotate the measurement basis to the W axis, use the sequence of gates S-H-T-H and then perform a standard measurement. The correlator $\langle ZW \rangle$ should be close to $1/\sqrt{2}$ and is found using the above equation.

In the second experiment the two observables are Z and $V = \frac{1}{\sqrt{2}}(-X+Z)$. To rotate to this basis we use the sequence of gates S-H-$T^\dagger$-H and then perform a standard measurement. The correlator $\langle ZV \rangle$ is found in a similar way as before and should be close to $1/\sqrt{2}$.

Finally, in the third and fourth experiment the correlators $\langle XW \rangle$ and $\langle XV \rangle$ ‘$aremeasuredandshouldbecloseto : math : ‘1/\sqrt{2}$ and $-1/\sqrt{2}$ respectively. The W and V measurement is performed the same way as above and the X via a Hadamard gate before a standard measurement.

Here you can see the results we got when we ran this experiment on the processor:

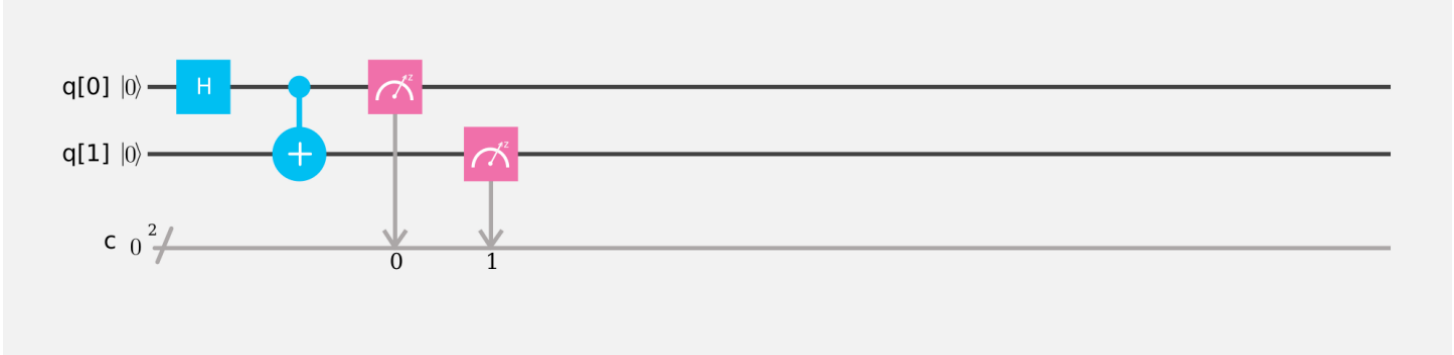
Bell test: 8192 shots May 2nd 11:44pm

	P(00)	P(11)	P(01)	P(10)	⟨AB⟩
ZW	0.434	0.380	0.070	0.116	0.629
ZV	0.409	0.415	0.100	0.076	0.648
XW	0.452	0.375	0.090	0.083	0.654
XV	0.110	0.077	0.451	0.36	−0.626

$|C| = 2.56 \pm 0.03$

Try it out for yourself! Compare what we got with the simulations (with both ideal and realistic parameters).

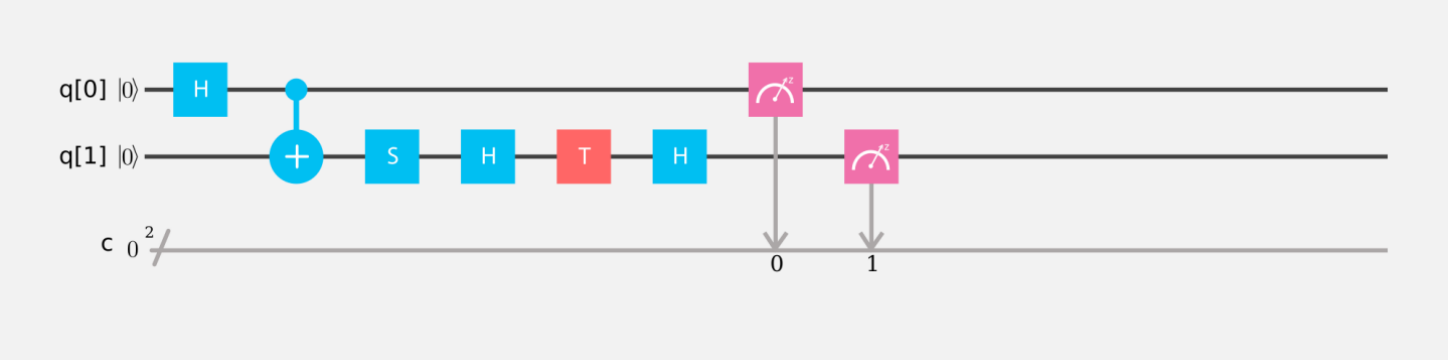
Bell State ZZ-Measurement



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=479450d7f0d95a28e4fa155576a9bcbd&sharedCode=true)

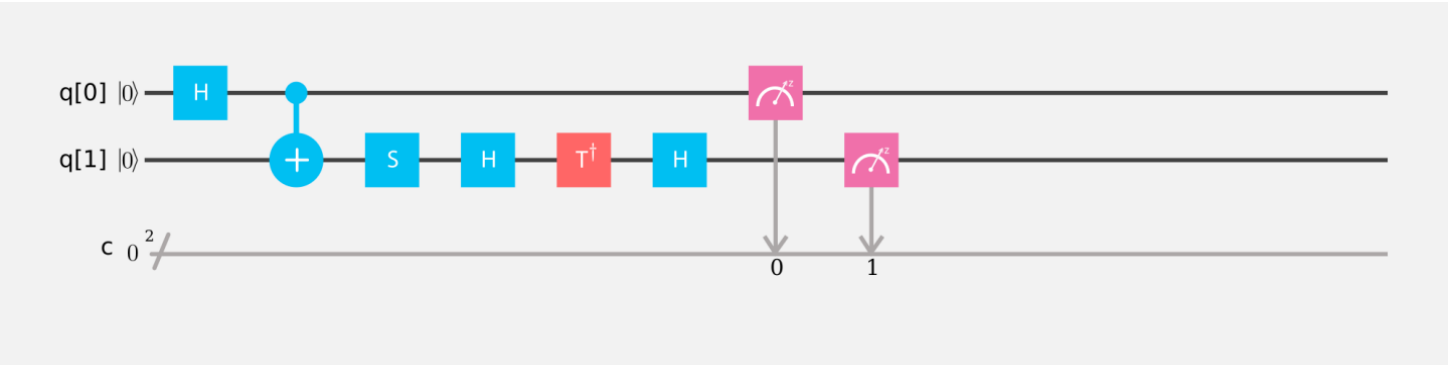
Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=479450d7f0d95a28e4fa155576a9bcbd&sharedCode=true)

Bell State ZW-Measurement



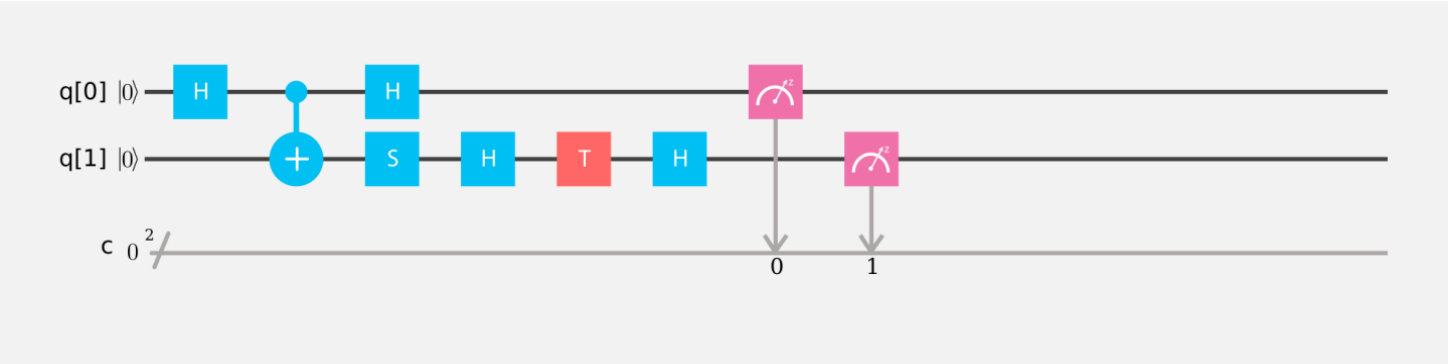
(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=e21164be4db4b6f5698f254a30184862&sharedCode=true)
Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=e21164be4db4b6f5698f254a30184862&sharedCode=true)

Bell State ZV-Measurement



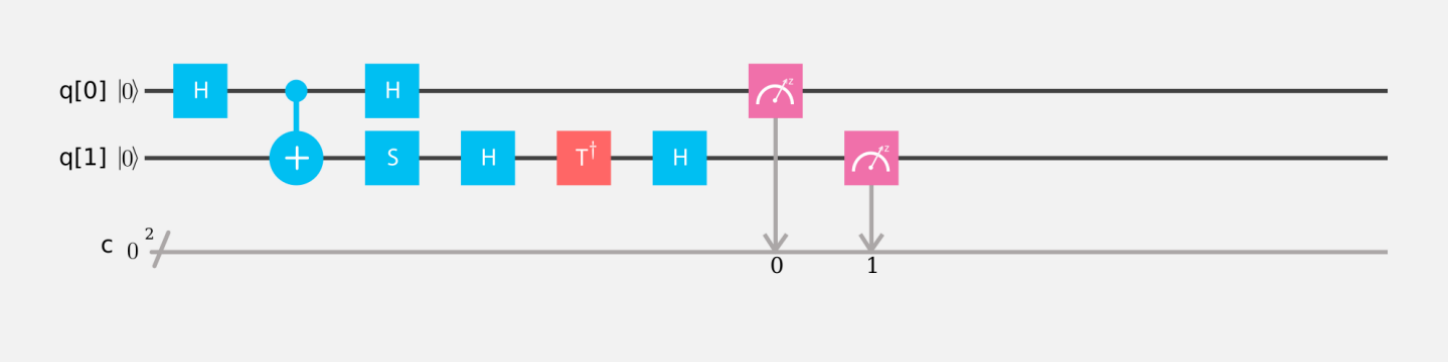
(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=e21164be4db4b6f5698f254a30191fac&sharedCode=true)
Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=e21164be4db4b6f5698f254a30191fac&sharedCode=true)

Bell State XW-Measurement



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=e21164be4db4b6f5698f254a301a9363&sharedCode=true)
Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=e21164be4db4b6f5698f254a301a9363&sharedCode=true)

Bell State XV-Measurement

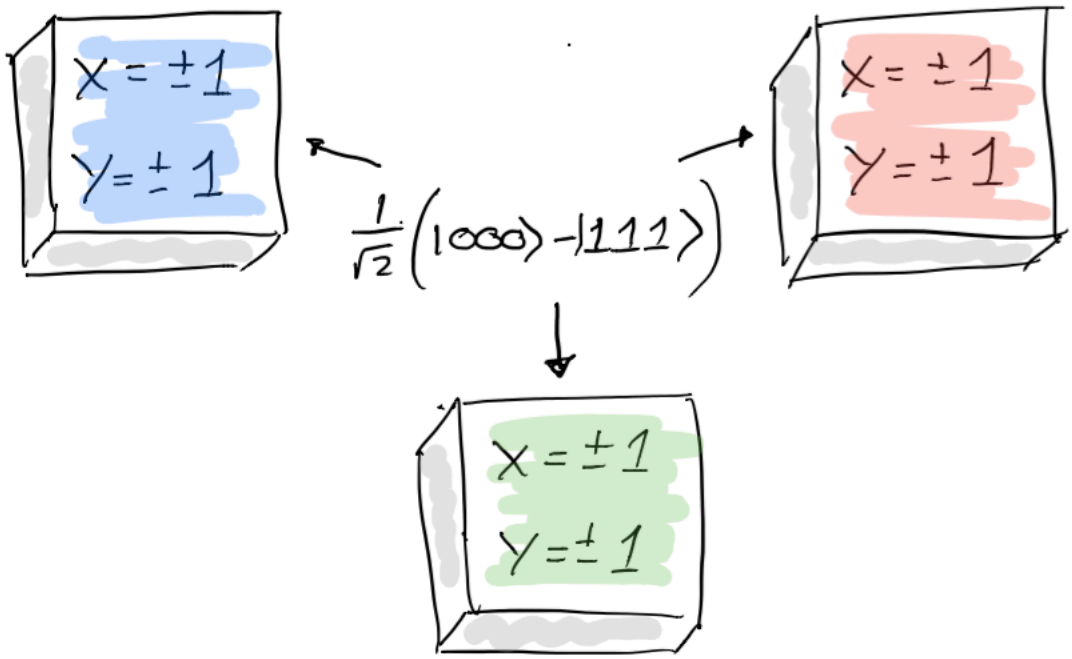


(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=479450d7f0d95a28e4fa155576ae55fb&sharedCode=true)
Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=479450d7f0d95a28e4fa155576ae55fb&sharedCode=true)

GHZ States

Perhaps even stranger than Bell states are their three-qubit generalization, the *GHZ states*. An example of one of these states is $\frac{1}{\sqrt{2}}(|000\rangle - |111\rangle)$. The measured results should be half $|000\rangle$ and half $|111\rangle$. GHZ states are named after Greenberger, Horne, and Zeilinger, who were the first to study them in 1997. GHZ states are also known as "Schroedinger cat states" or just "cat states."

In the 1990 paper by N. David Mermin, *What's wrong with these elements of reality?*, the GHZ states demonstrate an even stronger violation of local reality than Bell's inequality. Instead of a *probabilistic* violation of an inequality, the GHZ states lead to a *deterministic* violation of an equality.



Imagine you have three independent systems which we denote by a blue, red, and green box. You are asked to solve the following problem: in each box there are two questions, labeled X and Y, that have only two possible outcomes, $+1$ or -1. You must come up with a solution to the following set of identities.

$XYY = 1.$

$YXY = 1.$

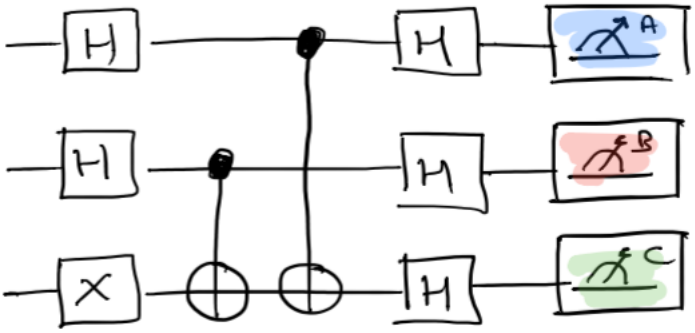
$YYX = 1.$

$XXX = -1.$

Try it!

After a while you will realize this is not possible. The simple way to show this is the following: if we multiply the first three equations together, we can simplify squared quantities and obtain $XXX = 1$, which contradicts the fourth identity.

Amazingly enough, a GHZ state can provide a solution to this problem. Then we have to accept what quantum mechanics teaches us: there are not local hidden elements of reality associated with each qubit which predetermine the outcomes of measurements in the X and Y bases. So, as Mermin pointed out, the GHZ test described above contradicts the possibility of physics being described by local reality! As opposed to the Bell test, which provides only a statistical evidence for the contradiction, the GHZ test can rule out the local reality description with certainty after a single run of the experiment (not including the effects of noise and imperfections in our system).



To make this state we use the following circuit, which is slightly different to the standard way of creating a GHZ (in our hardware the CNOT gates are constrained in their orientation). In the first part of the circuit, the ground state is taken to a superposition $\frac{1}{2}(|001\rangle + |011\rangle + |101\rangle + |111\rangle)$. The two CNOTs now entangle all the qubits into the state $\frac{1}{2}(|001\rangle + |010\rangle + |100\rangle + |111\rangle)$. The final three Hadamard gates map this to the GHZ state $\frac{1}{\sqrt{2}}(|000\rangle - |111\rangle)$.

To make the measurements in the X and Y basis we again rotate the measurement using the circuits you have seen before. For example, consider the XXX measurement. Note that flipping all three qubits of the GHZ state gives the same state with the minus sign. In other words, the GHZ state is a -1 eigenvector of a three-qubit Pauli operator XXX. This implies

$XXX = -1$

for each realization of the experiment. Next consider the Pauli operator XYY. One can check that the GHZ state is a $+1$ eigenevector of XYY. Therefore,

$XYY = 1$

for each realization of the experiment. Likewise,

$YXY = 1$, and $YYX = 1$.

One can verify this by running the experiments using the circuits provided below.

Here you can see the results we got when we ran this experiment on the processor:

GHZ test: 8192 shots May 3rd 1:20 AM

	P(000)	P(001)	P(010)	P(011)	P(100)	P(101)	P(110)	P(111)
YYX	0.201	0.038	0.016	0.114	0.030	0.258	0.258	0.086
XXX	0.046	0.226	0.125	0.007	0.250	0.025	0.094	0.228
YXY	0.222	0.042	0.012	0.111	0.034	0.254	0.251	0.074
XYY	0.227	0.039	0.013	0.128	0.033	0.247	0.229	0.084

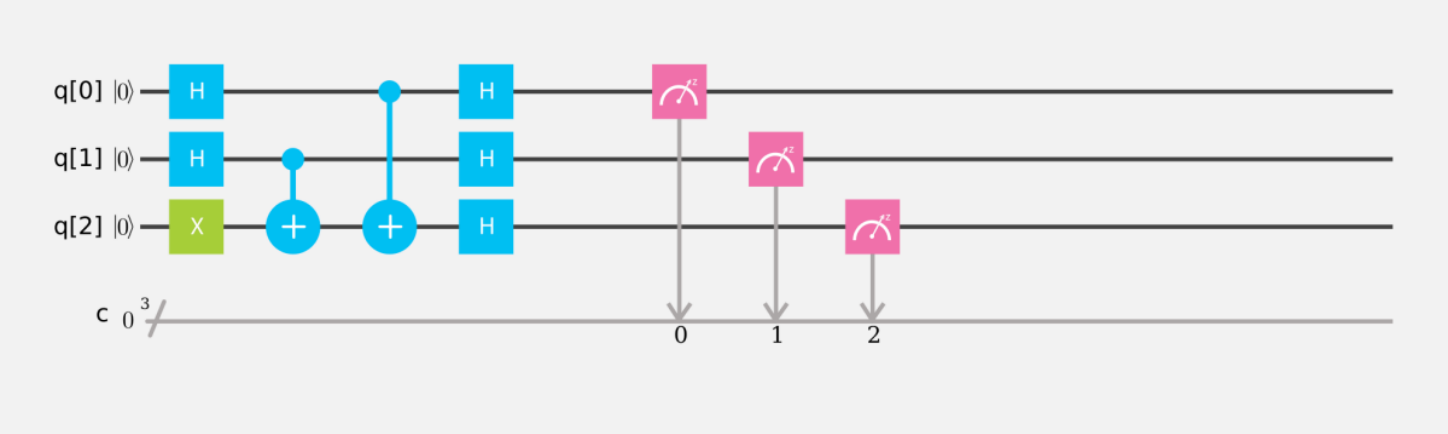
$\langle YYX \rangle = 0.661$ $\langle XXX \rangle = -0.657$ $\langle YXY \rangle = 0.675$ $\langle XYY \rangle = 0.661$

$M = \langle YYX \rangle \langle XXX \rangle \langle YXY \rangle \langle XYY \rangle = -0.194 \pm 0.005$

EXAMPLE CIRCUITS:

The first circuit shown below creates a GHZ state and then measures all qubits in the standard basis. The measured results should be half 000 and half 111. The remaining four circuits describe the GHZ test. Each circuit prepares the GHZ state and then measures the three qubits by choosing the measurement bases according to YYX, YXY, XYY, and XXX respectively.

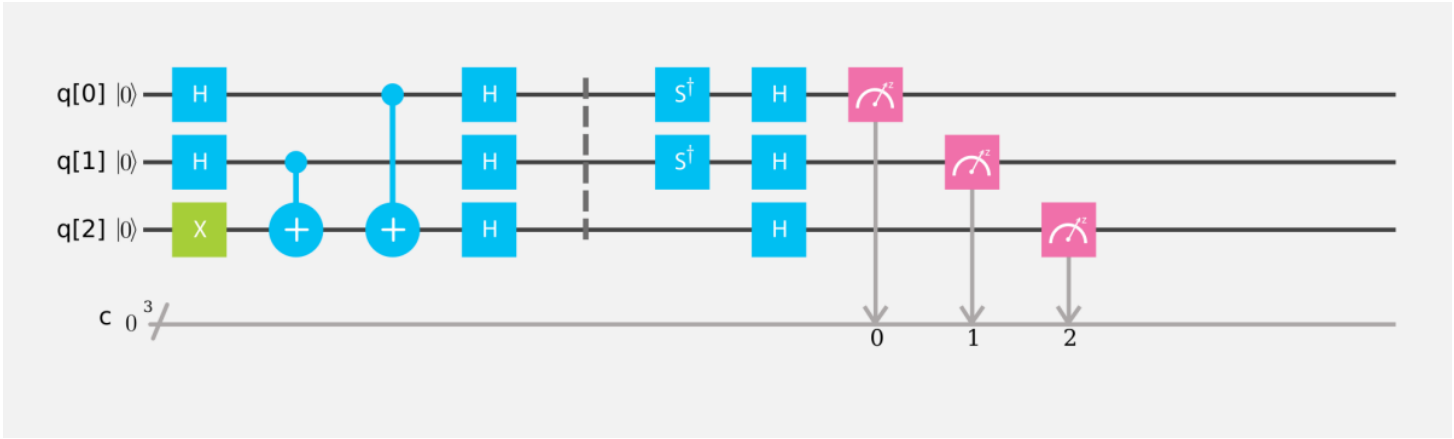
3Q GHZ State



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=479450d7f0d95a28e4fa155576c25c03&sharedCode=true)

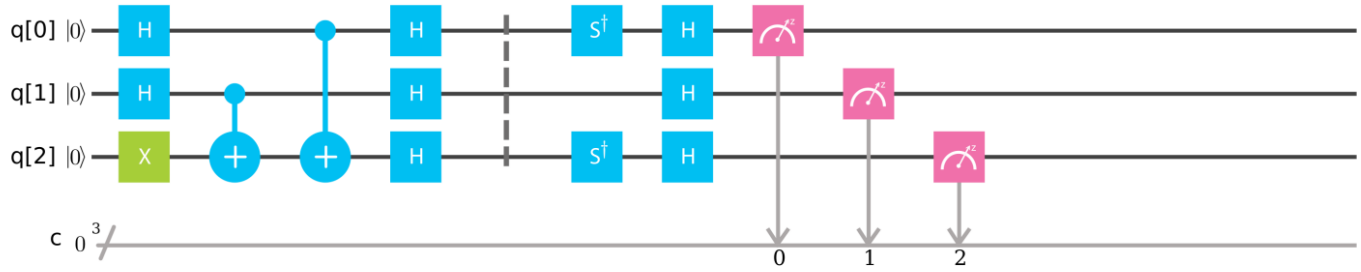
Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=479450d7f0d95a28e4fa155576c25c03&sharedCode=true)

3Q GHZ State YYX-Measurement



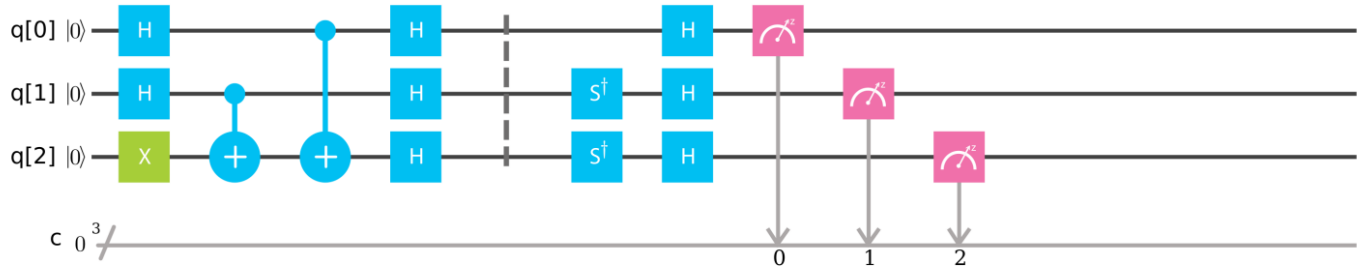
(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=0e6a68c3cd23d638b8093ad4d067d45f&sharedCode=true)
Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=0e6a68c3cd23d638b8093ad4d067d45f&sharedCode=true)

3Q GHZ State YXY-Measurement



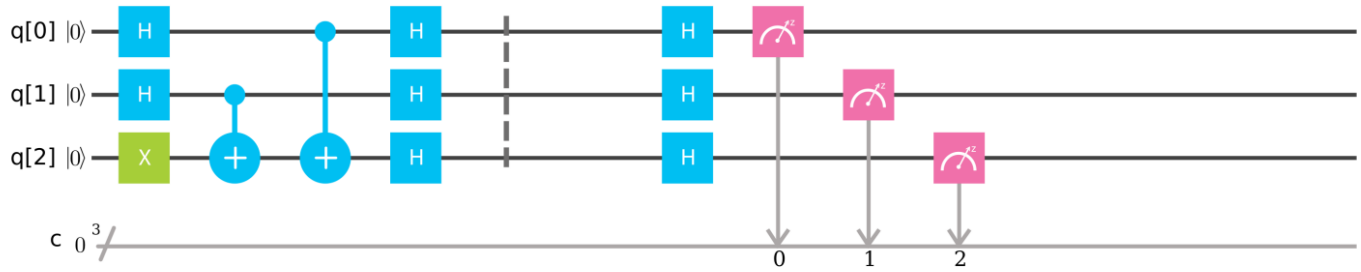
(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=09ed38c0975b6f51d6945603177805c3&sharedCode=true)
Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=09ed38c0975b6f51d6945603177805c3&sharedCode=true)

3Q GHZ State XYY-Measurement



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=ae3e60e47e4c28f29edcb5a9a49519c3&sharedCode=true) Open in composer
(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=ae3e60e47e4c28f29edcb5a9a49519c3&sharedCode=true)

3Q GHZ State XXX-Measurement



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=ba07eaf7c8c75a9f9c5cf12a947ce1e0&sharedCode=true)
Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=ba07eaf7c8c75a9f9c5cf12a947ce1e0&sharedCode=true)
In this section we will introduce you to multi-qubit states and operations. You will create an entangled state, known as a Bell state, and test some of the strangest properties of quantum physics.

The tutorials are broken down into the follow topics:

- CNOT and multi-qubit states
- T-gates (non-Clifford gates)
- Bell States
- GHZ states

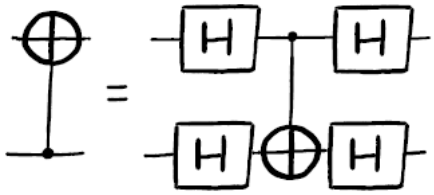
Quantum Algorithms

Basic Circuit Identities and Larger Circuits

There are several facts about quantum circuits that can be used to express more complicated unitary transformations (https://en.wikipedia.org/wiki/Unitary_transformation), write circuits more concisely, or adapt circuits to experimental constraints.

Changing the direction of a CNOT gate

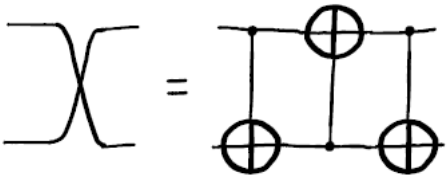
In the first example "CNOT (Reverse)," we consider how to implement a CNOT gate from control qubit 2 to target qubit 1 (notated $CNOT_{21}$) using a CNOT gate that acts in the opposite direction, from control qubit 1 to target qubit 2, $CNOT_{12}$. By multiplying the matrices for each gate, you can convince yourself that $(H \otimes H)CNOT_{12}(H \otimes H) = CNOT_{21}$.



The Kronecker product (https://en.wikipedia.org/wiki/Kronecker_product) $H \otimes H$ in this equation is equal to a four-by-four matrix $\frac{1}{\sqrt{2}}\begin{pmatrix} H & H \\ H & -H \end{pmatrix}$. If you run the example, you will confirm that this combination of Hadamard and CNOT gates implements a CNOT gate in the opposite direction. The Pauli (https://en.wikipedia.org/wiki/Quantum_gate#Pauli-X_gate) X acts to invert the control qubit 2, and the result is $|11\rangle$ as expected for $CNOT_{21}$.

Swapping the states of qubits

Our second example, "Swap," demonstrates a building block that allows you to permute the information stored in a set of qubits. Suppose we want to exchange the states of a pair of qubits by implementing a SWAP gate on the pair. There is no SWAP gate in our basis, but we can construct one from three CNOT gates $SWAP_{12} = CNOT_{12}CNOT_{21}CNOT_{12}$.

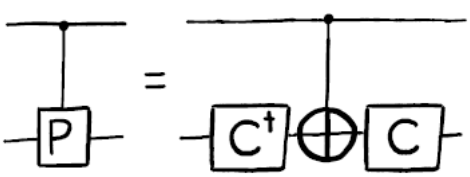


To see that this is true, it is enough to look at what happens to each classical state $00, 01, 10$, and 11. Let's consider 01. The first gate $CNOT_{12}$ does nothing since the control is 0. The second gate $CNOT_{21}$ flips the first qubit, so we have 11. Finally, the last $CNOT_{12}$ flips the second qubit and we get 10. The 1 has moved from the second qubit to the first. The other cases can be worked out similarly. Now you can see this for yourself by running the "Swap" example below. Notice that we have used the "CNOT (Reverse)" identity to change the direction of the $CNOT_{21}$ gate, since this is necessary to run the example on the real device. Try deleting the Pauli X and placing a Pauli X on qubit 1 instead.

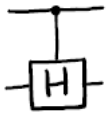
In the experimental device, not all qubits are connected to each other; therefore, some two-qubit gates cannot be applied directly. In the third example, "Swap Q0 with Q1," we show how to swap a pair of qubits that are not directly connected to each other but share a common neighbor (in this case Q2). The state $|+\rangle$, prepared by the first Hadamard gate on Q_0, is swapped into Q_1 by three successive SWAP gates.

Adding a control qubit to a gate

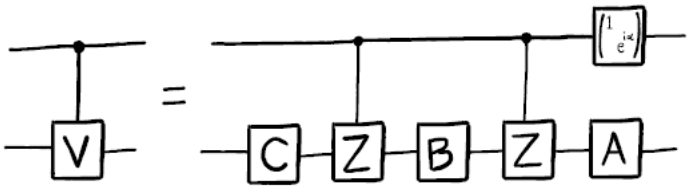
Some unitary transformations can be constructed exactly using gates in our basis. One set of transformations we use regularly are controlled-Pauli operations, where we apply a Pauli gate to a target qubit if a control qubit is $|1\rangle$. The CNOT gate is a controlled-X gate. Since we know that $HXH = Z$ and $SXS^\dagger = Y$, and furthermore that $HH = I$ and $SS^\dagger = I$, it is straightforward to construct circuits for controlled-Z and controlled-Y. This is illustrated in the following figure, where P is a Pauli gate X, Y, or Z, and C is a Clifford operation such as I, S, or H.



For a more involved example, let's add a control qubit to a Hadamard gate to implement a controlled-Hadamard operation:



It turns out that we can write down a circuit for a controlled-V operation if we can find three circuits A, B, and C such that $ABC = I$ and $e^{i\alpha}AZBZC = V$ [Barenco et al., 1995 (http://journals.aps.org/pra/abstract/10.1103/PhysRevA.52.3457?cm_mc_uid=43781767191014577577895&cm_mc_sid_50200000=1460741020)]. There is a general recipe for doing this, but we will just write down a solution for when $V = H$ that you can check for yourself: $A = e^{i3\pi/8}XSHTHS^\dagger$, $B = e^{-i\pi/8}SHT^\dagger HS^\dagger HSH$, $C = e^{-i\pi/4}HSH$, and $e^{i\alpha} = -i$. Combining these circuits as shown here



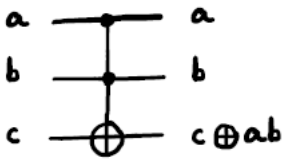
and making some simplifications, we get the result shown in the fourth example, "controlled-Hadamard." This example applies a Hadamard gate to the control qubit, Q_0, and then applies the controlled-Hadamard circuit from Q_0 to Q_2. This creates the output state $\frac{1}{\sqrt{2}}(|00\rangle + |1+\rangle) = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{2}|10\rangle + \frac{1}{2}|11\rangle$. Try deleting the first Hadamard gate on Q_0 and replacing it with a bit-flip (X) to see what happens. Can you implement circuits for other controlled gates, such as a controlled-S?

Approximating a unitary transformation

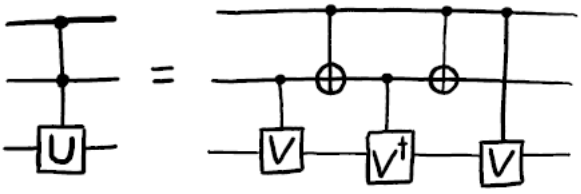
Most unitary transformations cannot be written exactly using the gates we have in our basis; but because our basis is a discrete universal set, it *is* possible to approximate any unitary transformation to any accuracy. Let's see an example of this. The $\sqrt{T}$ unitary transformation cannot be written exactly using our basis. Since $\sqrt{T}$ is a Z-rotation, the identity gate (which does nothing) is an approximate $\sqrt{T}$ gate, but not a very good one. The fifth example "Approximate sqrt(T)" gives a much better approximation using 17 Hadamard, S, and T gates. This example first puts the qubit Q_0 on the equator of the Bloch sphere using a Hadamard gate, then applies the 17 gate sequence. We use state tomography (https://en.wikipedia.org/wiki/Quantum_tomography) to observe the final state on the Bloch sphere. Had we applied an exact $\sqrt{T}$ gate, the final state would correspond to the point $(\frac{\sqrt{2+\sqrt{2}}}{2}, \frac{\sqrt{2-\sqrt{2}}}{2}, 0) \approx (0.92388, 0.38268, 0)$ on the Bloch sphere. How good is the 17 gate approximation? Arbitrarily good approximations exist, so can you find a better one? How might you use these circuits to construct an approximate controlled-T unitary transformation?

The Toffoli gate

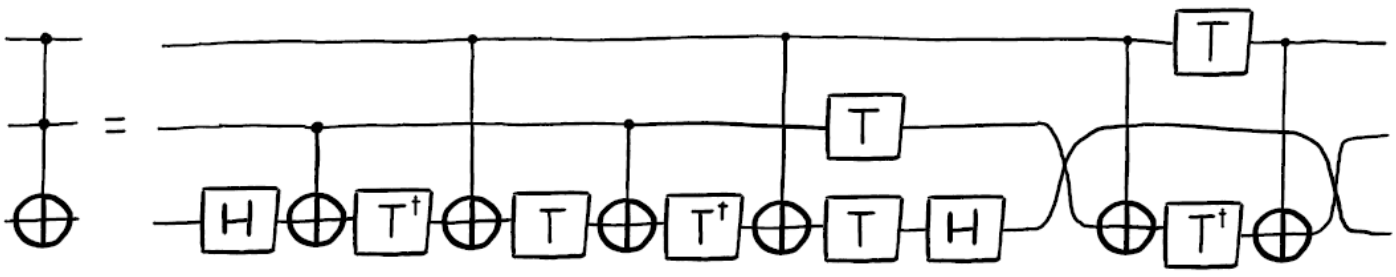
Our final examples, "Toffoli with flips" and "Toffoli state" demonstrate how to implement the reversible circuit equivalent of the (irreversible, classical) AND gate using gates from our basis. An AND gate has two inputs and one output, and outputs 1 if and only if both inputs are 1. One reason the AND gate is important for computation is that it is a non-linear transformation (a multiplication). Why do we say AND is irreversible? Notice that all three inputs 00, 01, and 10 result in an output of 0. If you see that the output of the gate is 0, you can't undo the gate because you don't know which of these three inputs gave you the result. Since there are three possible answers, even if you add another output qubit, you won't have enough information to undo the gate, since you must distinguish 3 cases, and there are only two choices for the state of the new qubit. However, it is possible to implement AND reversibly using 3 wires. This reversible AND gate is called the Toffoli gate (https://en.wikipedia.org/wiki/Toffoli_gate) $TOF|a, b, c\rangle = |a, b, (a \text{ AND } b) \text{ XOR } c\rangle$.



It is not obvious how to build a Toffoli gate from gates in our basis [Barenco et al., 1995 (http://journals.aps.org/pra/abstract/10.1103/PhysRevA.52.3457?cm_mc_uid=43781767191014577577895&cm_mc_sid_50200000=1460741020)]. The main idea is illustrated in the following figure



where $V = \sqrt{U}$. By tracing through each value of the two control qubits, you can convince yourself that a U gate is applied to the target qubit if and only if both controls are 1. Using ideas we have already described, you could now implement each controlled-V gate to arrive at some circuit for the doubly-controlled-U gate. It turns out that the minimum number of CNOT gates required to implement the Toffoli gate is 6 [Shende and Markov, 2009 (http://dl.acm.org/citation.cfm?id=2011799)]:

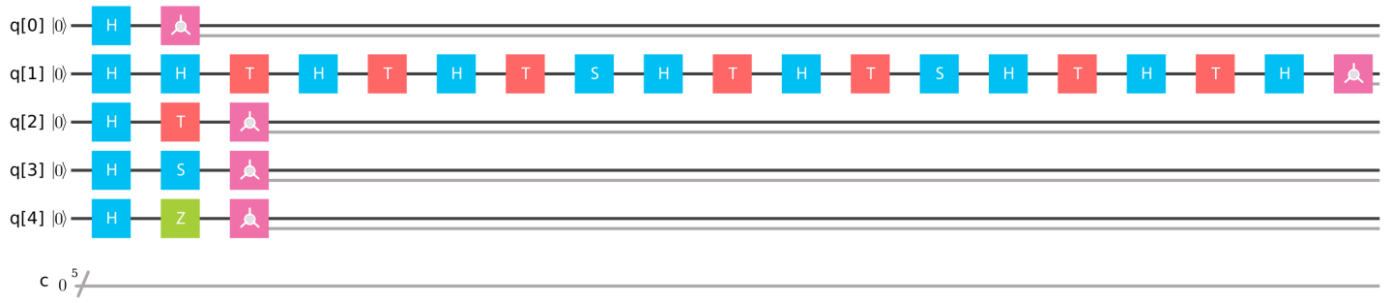


The "Toffoli with flips" example below allows you to choose an input state by changing the single-qubit gates at the beginning of the circuit. Right now they are set to Pauli X

on qubits 0 and 1 and identity on qubit 2, so the default output is $|111\rangle$. You will notice that the example circuit uses 9 CNOT gates instead of the optimal number 6. This is because we wrote the circuit so it can run on the real device, which requires inserting a SWAP gate on qubits 1 and 2 near the end of the circuit. Note that we do not undo this swap, so if the input qubit labels are $0, 1, 2$, the output labels are actually $0, 2, 1$. This means, for example, that an input of 010 produces output 001 (not 010). Finally, the "Toffoli state" example demonstrates the creation of an interesting 3-qubit entangled state $SWAP_{1,2}TOF|++0\rangle_{0,1,2} = \frac{1}{2}(|000\rangle + |001\rangle + |100\rangle + |111\rangle)$ that encodes the truth table (https://en.wikipedia.org/wiki/Truth_table) of the Toffoli gate.

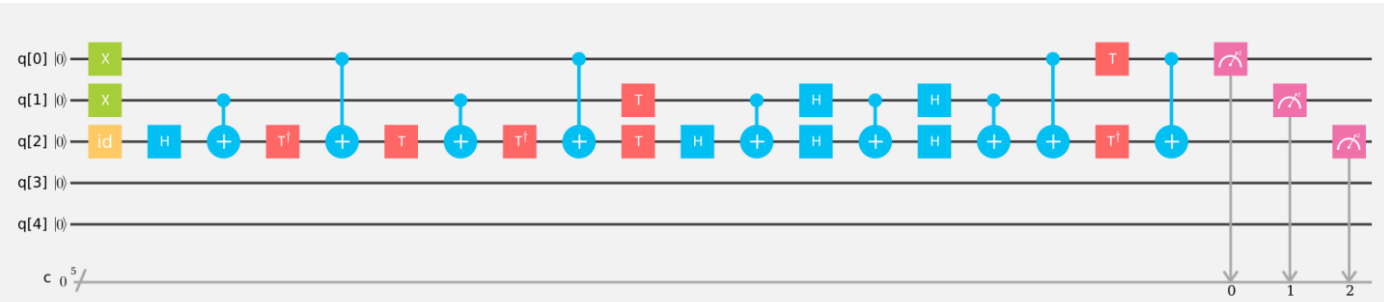
Using the Toffoli gate, it is possible to construct more complex circuits. A single Toffoli gate is sufficient to implement a modulo-4 addition operation between a pair of 2-bit registers or an increment-by-one operation from one 2-bit register to another. Can you find circuits for these operations and run them in the Quantum Composer?

Approximate sqrt(T)



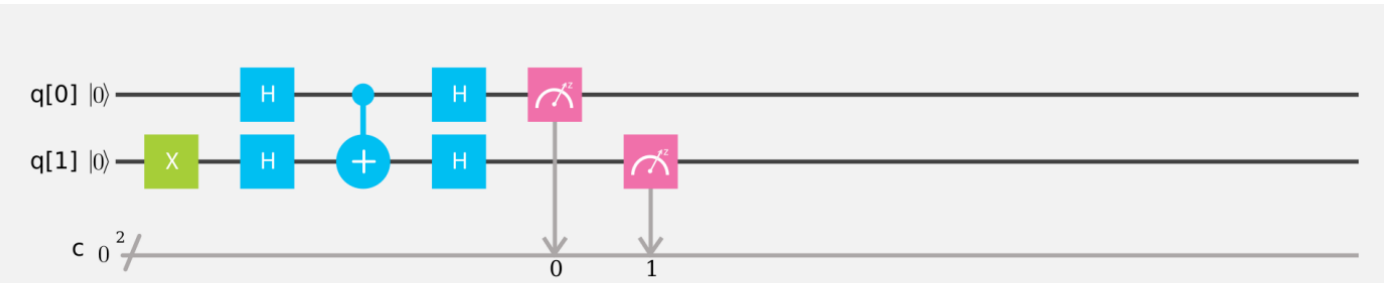
(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=b5a0e7376ded40cd7dc1022e778fd799&sharedCode=true)
Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=b5a0e7376ded40cd7dc1022e778fd799&sharedCode=true)

Toffoli with flips



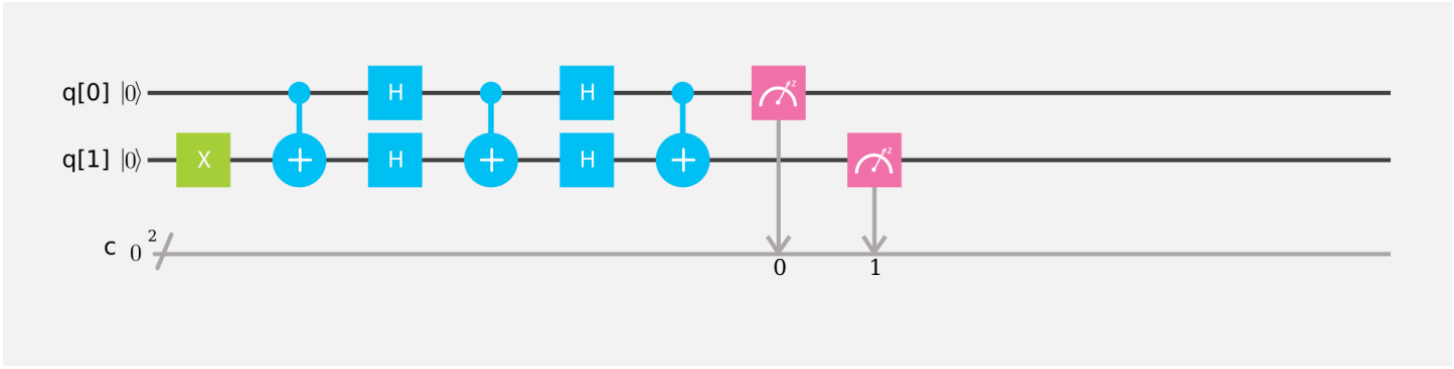
(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=4de5b472cdbc8f33b422ee097e26a3f1&sharedCode=true)
Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=4de5b472cdbc8f33b422ee097e26a3f1&sharedCode=true)

CNOT (Reversed)



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=56bfed556c0d64e8b92649d5fb206a21&sharedCode=true)
Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=56bfed556c0d64e8b92649d5fb206a21&sharedCode=true)

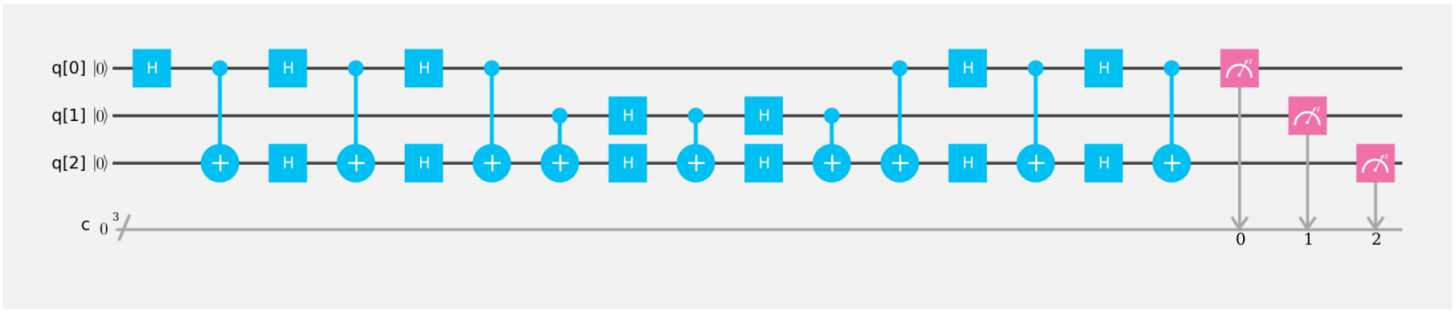
SWAP Gate



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=5537ba08c4b9b9369c47ea0fd67d26ea&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=5537ba08c4b9b9369c47ea0fd67d26ea&sharedCode=true)

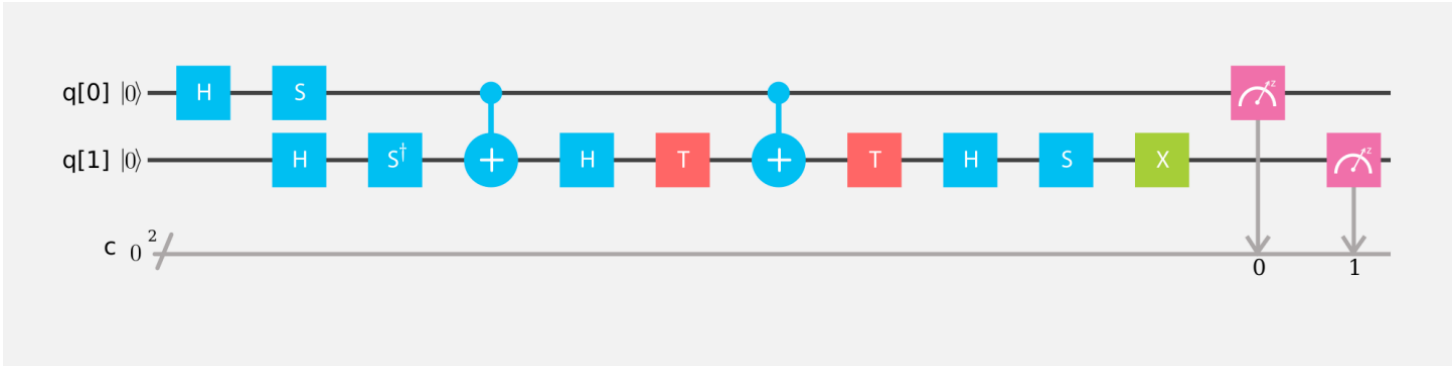
Swap q[0] and q[1]



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=5537ba08c4b9b9369c47ea0fd67c616a&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=5537ba08c4b9b9369c47ea0fd67c616a&sharedCode=true)

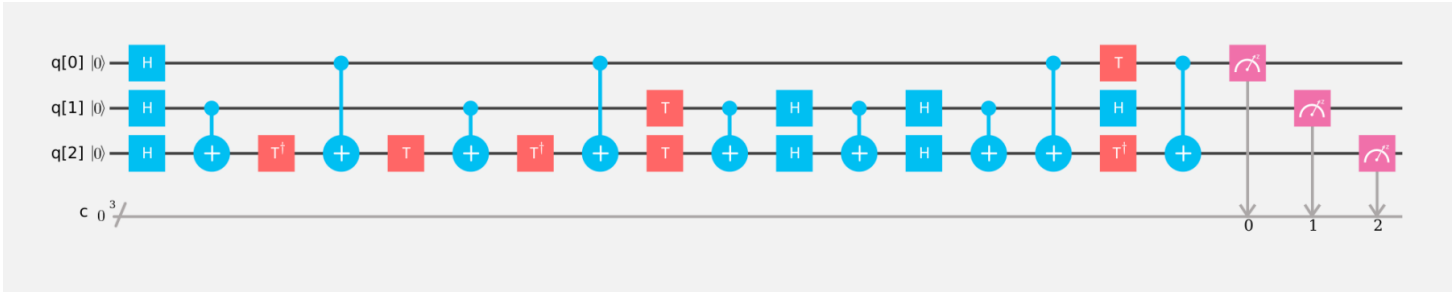
Controlled-Hadamard



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=5537ba08c4b9b9369c47ea0fd6706724&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=5537ba08c4b9b9369c47ea0fd6706724&sharedCode=true)

3Q Toffoli state



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=a685beebe7614db5a17b8eddbdcd7e90&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=a685beebe7614db5a17b8eddbdcd7e90&sharedCode=true)

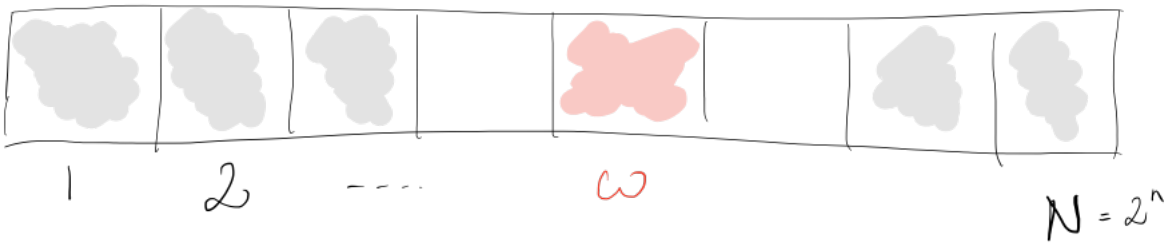
Grover's Algorithm

We are now in a good place to discuss our first quantum algorithms and subroutines. Let's begin with Grover's search algorithm (http://arxiv.org/abs/quant-ph/9605043) and the *amplitude amplification trick*.

You have likely heard that one of the many advantages a quantum computer has over a classical computer is its superior speed searching databases. Grover's algorithm demonstrates this capability. This algorithm can speed up an unstructured search problem quadratically, but its uses extend beyond that; it can serve as a general trick or subroutine to obtain quadratic run time improvements for a variety of other algorithms. This is called the amplitude amplification trick. But before we start the simulations, let's look at the unstructured search problem.

Unstructured search

Suppose you are given a large list of N items. Among these items there is one item with a unique property that we wish to locate; we will call this one the winner w. Think of each item in the list as a box of a particular color. Say all items in the list are gray except the winner w, which is pink.



To find the pink box – the *marked item* – using classical computation, one would have to check on average $N/2$ of these boxes, and in the worst case, all N of them. On a quantum computer, however, we can find the marked item in roughly $\sqrt{N}$ steps with Grover's amplitude amplification trick. It was proven (even before Grover's algorithm was discovered!) that this speedup is in fact the most we can hope for [Bennett, 1997 (http://arxiv.org/abs/quant-ph/9701001)]. A quadratic speedup is indeed a substantial time-saver for finding marked items in long lists. Additionally, the algorithm does not use the list's internal structure, which makes it *generic;* this is why it immediately provides a quadratic quantum speed-up for many classical problems.

The Oracle

How will the list items be provided to the quantum computer? A common way to encode such a list is in terms of a function f which returns $f(x) = 0$ for all unmarked items x and $f(w) = 1$ for the winner. To use a quantum computer for this problem, we must provide the items in superposition to this function, so we encode the function into a unitary matrix called an *oracle*. First we choose a binary encoding of the items x, $w \in \{0, 1\}^n$ so that $N = 2^n$; now we can represent it in terms of qubits on a quantum computer. Then we define the oracle matrix U_f to act on any of the simple, standard basis states $|x\rangle$ by

$$U_f|x\rangle = (-1)^{f(x)}|x\rangle.$$

We see that if x is an unmarked item, the oracle does nothing to the state. However, when we apply the oracle to the basis state $|w\rangle$, it maps $U_f|w\rangle = -|w\rangle$. Geometrically, this unitary matrix corresponds to a reflection about the origin for the marked item in an $N = 2^n$ dimensional vector space.

Amplitude amplification

So how does the algorithm work? Before looking at the list of items, we have no idea where the marked item is. Therefore, any guess of its location is as good as any other, which can be expressed in terms of a quantum state called a *uniform superposition*:

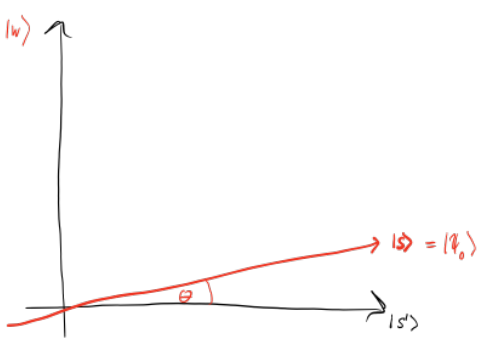
$$|s\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle.$$

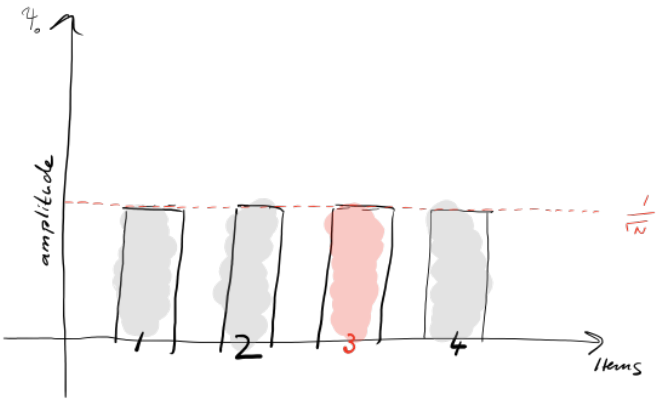
If at this point we were to measure in the standard basis $\{|x\rangle\}$, this superposition would collapse, according to the fifth quantum law, to any one of the basis states with the same probability of $\frac{1}{N} = \frac{1}{2^n}$. Our chances of guessing the right value w is therefore 1 in 2^n, as could be expected. Hence, on average we would need to try about $N = 2^n$ times to guess the correct item.

Enter the procedure called amplitude amplification, which is how a quantum computer significantly enhances this probability. This procedure stretches out (amplifies) the amplitude of the marked item, which shrinks the other items' amplitude, so that measuring the final state will return the right item with near-certainty.

This algorithm has a nice geometrical interpretation in terms of two reflections, which generate a rotation in a two-dimensional plane. The only two special states we need to consider are the winner $|w\rangle$ and the uniform superposition $|s\rangle$. These two vectors span a two-dimensional plane in the vector space $\mathbb{C}^N$. They are not quite perpendicular because $|w\rangle$ occurs in the superposition with amplitude $N^{-1/2}$ as well. We can, however, introduce an additional state $|s'\rangle$ that is in the span of these two vectors, which is perpendicular to $|w\rangle$ and is obtained from $|s\rangle$ by removing $|w\rangle$ and rescaling.

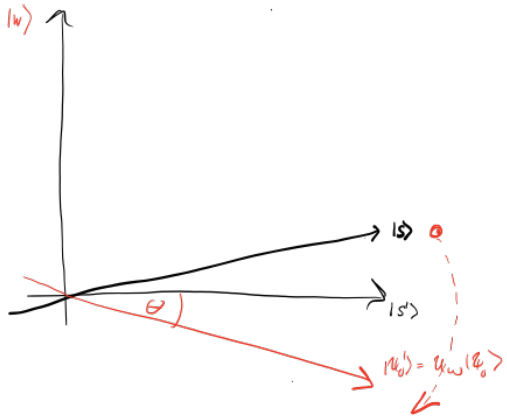
step 0 The amplitude amplification procedure starts out in the uniform superposition $|s\rangle$. (The uniform superposition is easily constructed from $|s\rangle = H^{\otimes n}|0\rangle^n$, as was shown in a previous section.) At $t = 0$ the initial state is $|\psi_0\rangle = |s\rangle$.

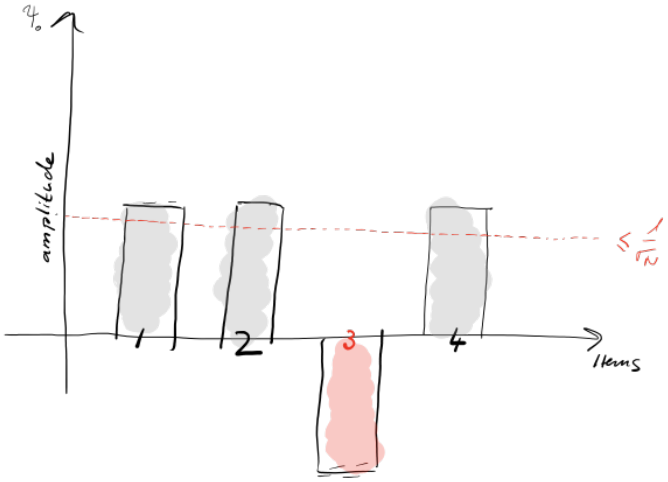




The left graphic corresponds to the two-dimensional plane spanned by $|w\rangle$, $|s\rangle$. The right graphic is a bar graph of the amplitudes of the state $|\psi_t\rangle$ for the case $N = 2^2 = 4$. The average amplitude is indicated by a dashed line.

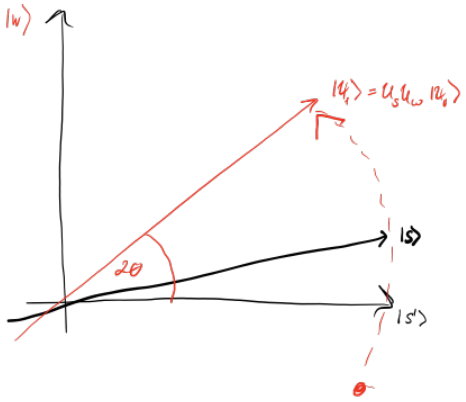
step 1 We apply the oracle reflection U_f to the state $U_f|\psi_t\rangle = |\psi_{t'}\rangle$.

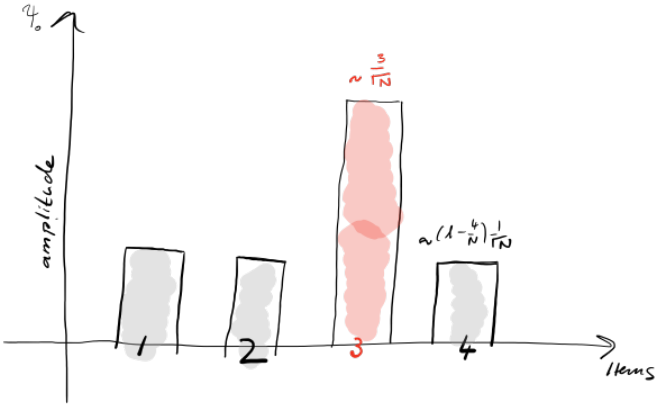




Geometrically this corresponds to a reflection of the state $|\psi_t\rangle$ about $-|w\rangle$. This transformation means that the amplitude in front of the $|w\rangle$ state becomes negative, which in turn means that the average amplitude has been lowered.

step 2 We now apply an additional reflection U_s about the state $|s\rangle$. In the bra-ket (https://en.wikipedia.org/wiki/Bra%E2%80%93ket_notation) notation this reflection is written $U_s = 2|s\rangle\langle s| - \mathbb{1}$. This transformation maps the state to $U_s|\psi_{t'}\rangle$ and completes the transformation $|\psi_{t+1}\rangle = U_s U_f|\psi_t\rangle$.



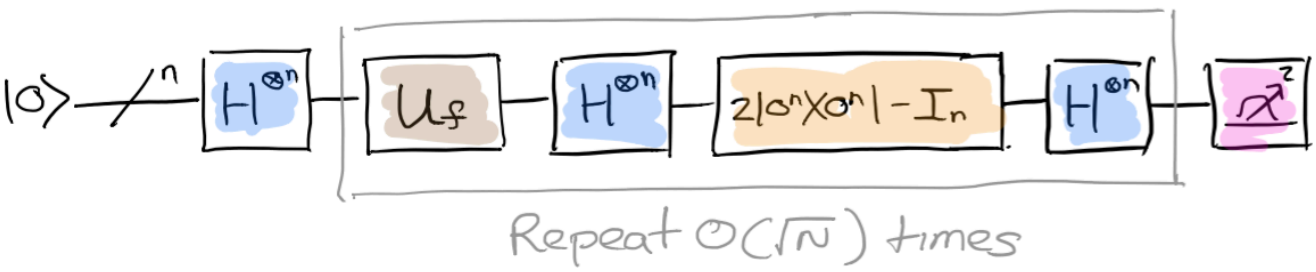


Two reflections always correspond to a rotation. The transformation $U_s U_f$ rotates the initial state $|s\rangle$ closer towards the winner $|w\rangle$. The action of the reflection U_s in the amplitude bar diagram can be understood as a reflection about the average amplitude. Since the average amplitude has been lowered by the first reflection, this transformation boosts the negative amplitude of $|w\rangle$ to roughly three times its original value, while it decreases the other amplitudes. We then go to **step 1** to repeat the application. This procedure will be repeated several times to zero in on the winner.

After t steps the state will have transformed to

$$|\psi_t\rangle = (U_s U_f)^t |\psi_0\rangle.$$

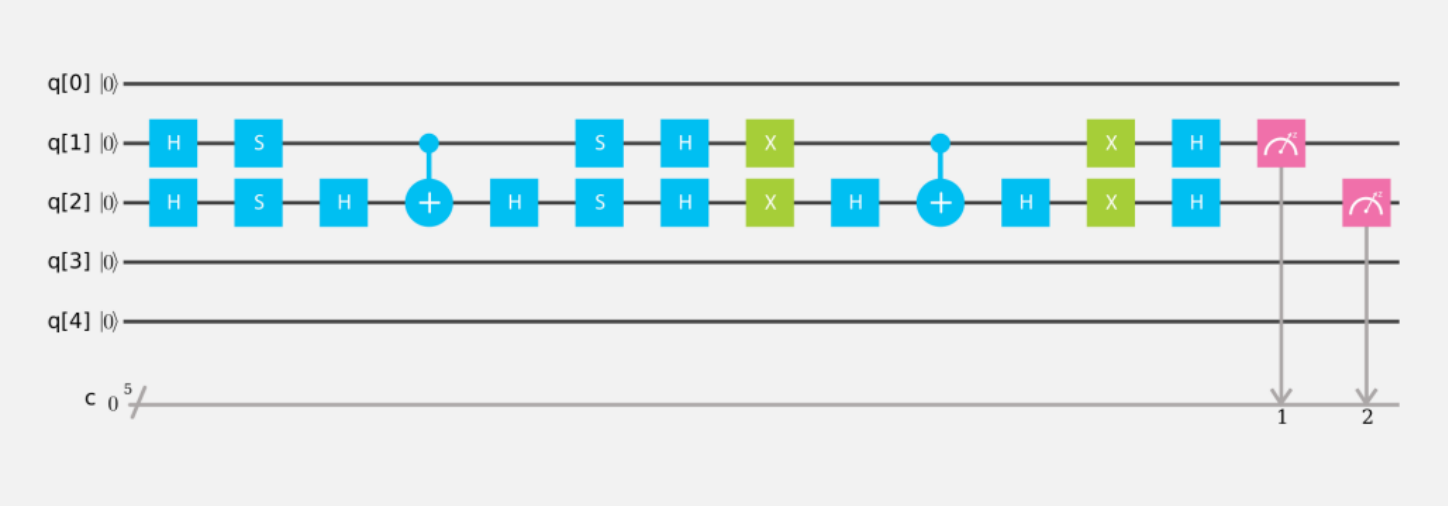
How many times do we need to apply the rotation? It turns out that roughly $\sqrt{N}$ rotations suffice. This becomes clear when looking at the amplitudes of the state $|\psi_t\rangle$. We can see that the amplitude of $|w\rangle$ grows linearly with the number of applications $\sim tN^{-1/2}$. However, since we are dealing with amplitudes and not probabilities, the vector space's dimension enters as a square root. Therefore it is the amplitude, and not just the probability, that is being amplified in this procedure.



Example circuits

Let us now examine a simple example. The smallest circuit for which this can be implemented involves 2 qubits, i.e. $N = 2^2$, which means there are four possible oracles U_f, one for each choice of the winner. The first part of this example creates the uniform superposition. The second part tags the state with U_f, which is made from a control-Z gate, made from a CNOT (as described in the last section). The final part of the circuit performs U_s.

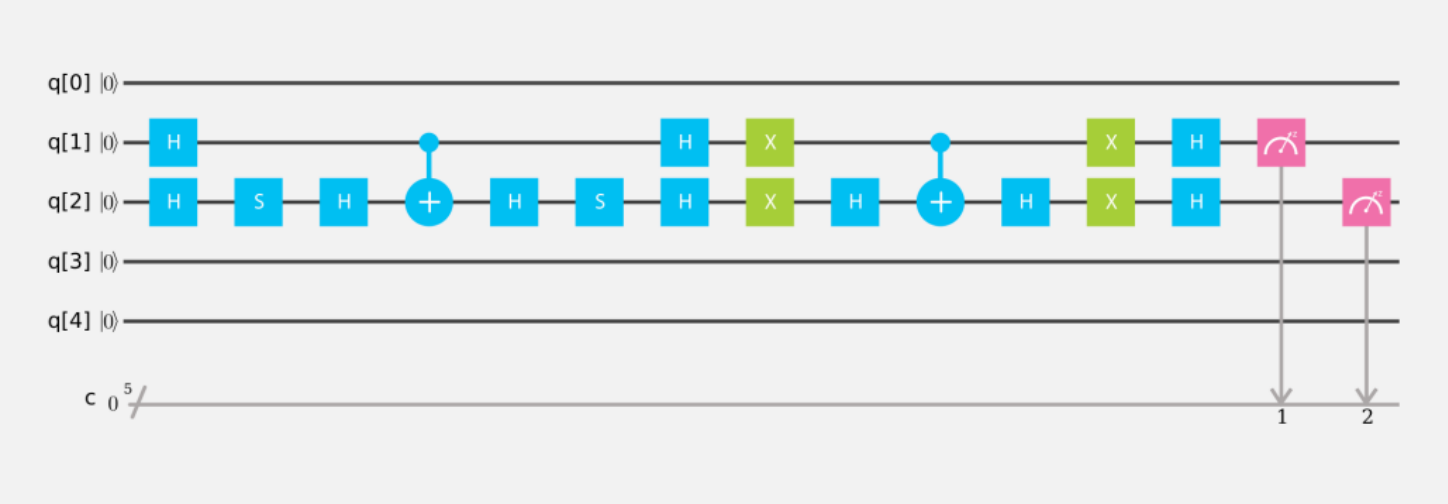
Grover N=2 A=00



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=ac042c16f1d2bf7312503e842e7ffbcc&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=ac042c16f1d2bf7312503e842e7ffbcc&sharedCode=true)

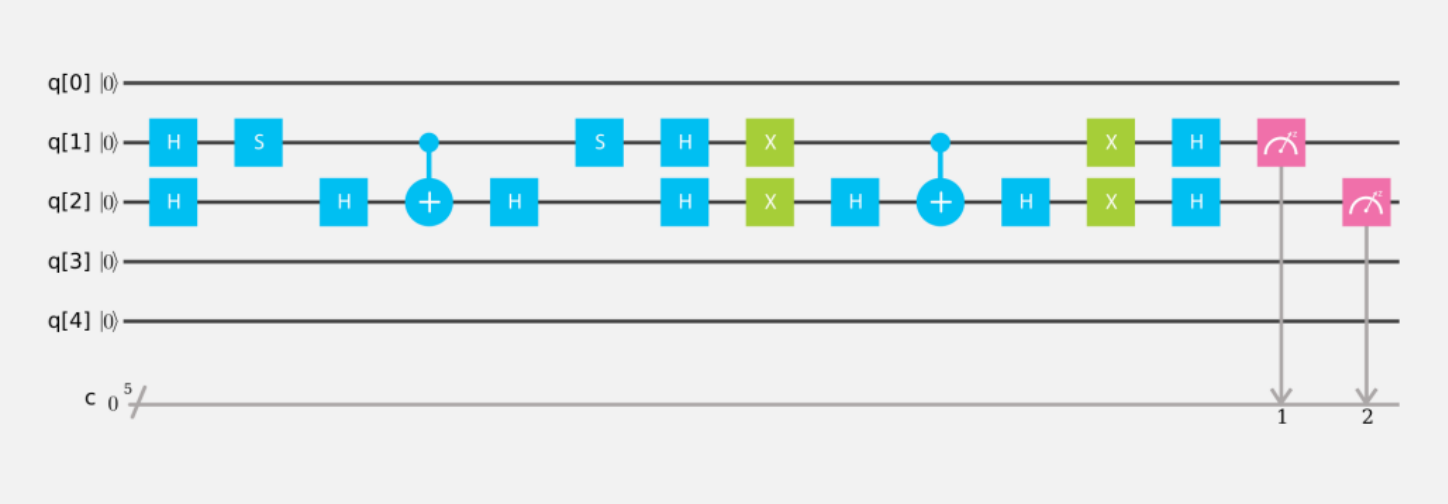
Grover N=2 A=01



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=34bf6d8078127a4cbb85dcba18d71547&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=34bf6d8078127a4cbb85dcba18d71547&sharedCode=true)

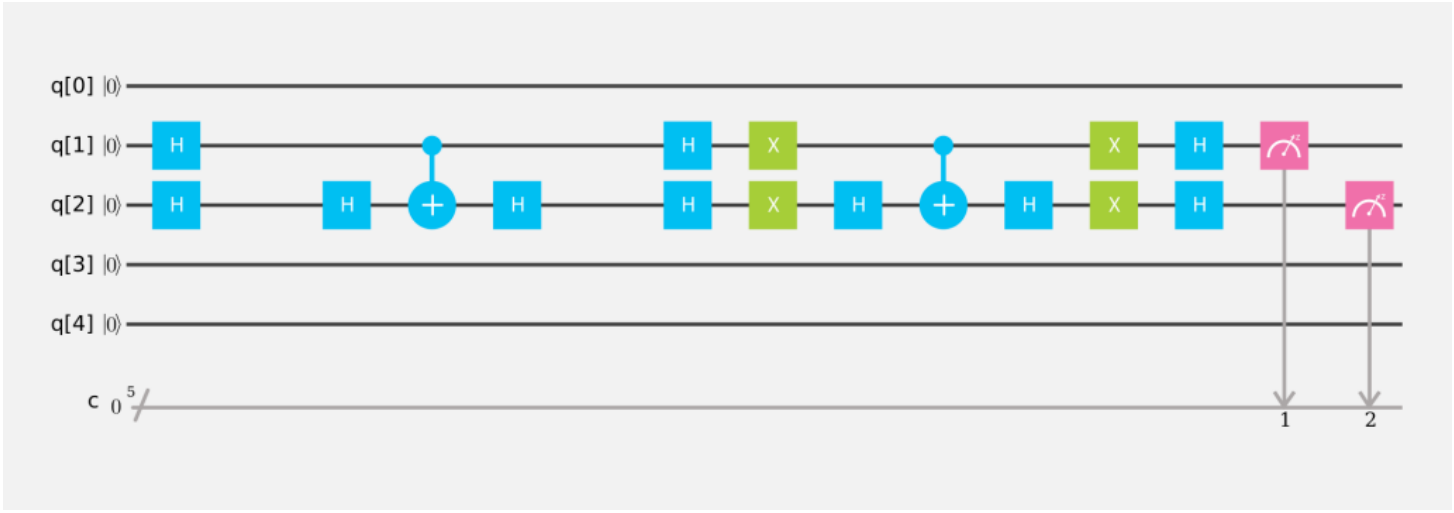
Grover N=2 A=10



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=e554e35d344f1e346c3b7cd30c5d1939&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=e554e35d344f1e346c3b7cd30c5d1939&sharedCode=true)

Grover N=2 A=11



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=764735cc22581811f9f382d3b3c644f0&sharedCode=true)

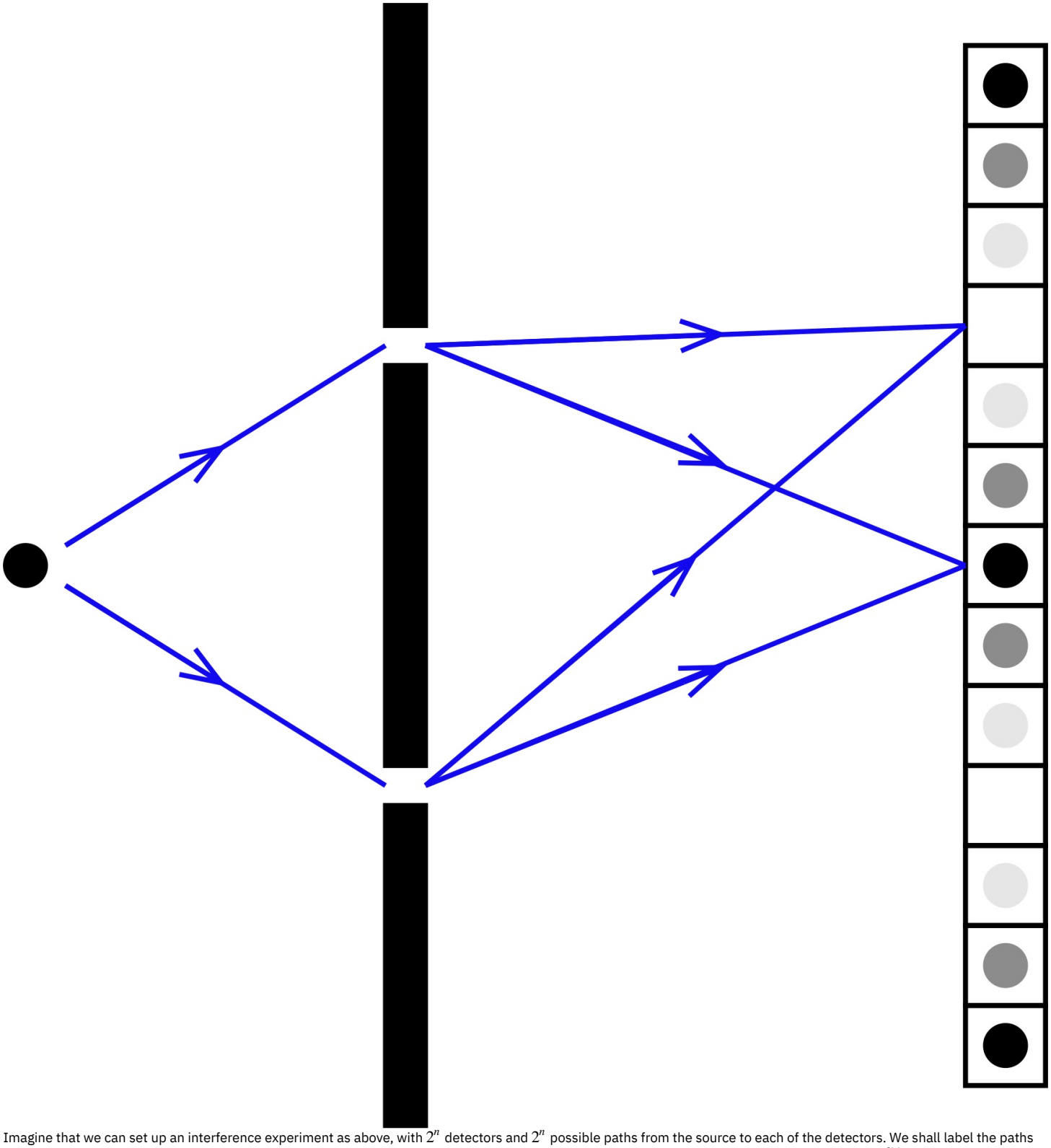
Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=764735cc22581811f9f382d3b3c644f0&sharedCode=true)

Deutsch-Jozsa Algorithm

The Deutsch-Jozsa algorithm was the first to show a separation between the quantum and classical difficulty of a problem. This algorithm demonstrates the significance of allowing quantum amplitudes to take both positive and negative values, as opposed to classical probabilities that are always non-negative.

The Deutsch-Jozsa problem is defined as follows. Consider a function $f(x)$ that takes as input n-bit strings x and returns 0 or 1. Suppose we are promised that $f(x)$ is either a **constant** function that takes the same value $c \in \{0, 1\}$ on all inputs x, or a **balanced** function that takes each value 0 and 1 on exactly half of the inputs. The goal is to decide whether f is constant or balanced by making as few function evaluations as possible. Classically, it requires $2^{n-1} + 1$ function evaluations in the worst case. Using the Deutsch-Jozsa algorithm, the question can be answered with just one function evaluation. In the quantum world the function f is specified by an oracle circuit U_f (see the previous section on Grover's algorithm, such that $U_f|x\rangle = (-1)^{f(x)}|x\rangle$).

To understand how the Deutsch-Jozsa algorithm works, let us first consider a typical interference experiment: a particle that behaves like a wave, such as a photon, can travel from the source to an array of detectors by following two or more paths simultaneously. The probability of observing the particle will be concentrated at those detectors where most of the incoming waves arrive with the same phase.



Imagine that we can set up an interference experiment as above, with 2^n detectors and 2^n possible paths from the source to each of the detectors. We shall label the paths and the detectors with n-bit strings x and y respectively. Suppose further that the phase accumulated along a path x to a detector y equals $C(-1)^{f(x)+x\cdot y}$, where

$$x \cdot y = \sum_{i=1}^{n} x_i y_i$$

is the binary inner product and C is a normalizing coefficient. The probability to observe the particle at a detector y can be computed by summing up amplitudes of all paths x arriving at y and taking the absolute value squared:

$$\Pr(y) = |C \sum_x (-1)^{f(x)+x\cdot y}|^2$$

Normalization condition $\sum_y \Pr(y) = 1$ then gives $C = 2^{-n}$. Let us compute the probability $\Pr(y = 0^n)$ of observing the particle at the detector $y = 0^n$ (all zeros string). We have $\Pr(y = 0^n) = |2^{-n} \sum_x (-1)^{f(x)}|^2$
If $f(x) = c$ is a constant function, we get $\Pr(y = 0^n) = |(-1)^c|^2 = 1$. However, if $f(x)$ is a balanced function, we get $\Pr(y = 0^n) = 0$, since all the terms in the sum over x cancel each other.

We can therefore determine whether f is constant or balanced with certainty by running the experiment just once.
Of course, this experiment is not practical since it would require an impossibly large optical table! However, we can simulate this experiment on a quantum computer with just n qubits and access to the oracle circuit U_f. Indeed, consider the following algorithm:
Step 1. Initialize n qubits in the all-zeros state $|0, \dots, 0\rangle$.
Step 2. Apply the Hadamard gate H to each qubit.

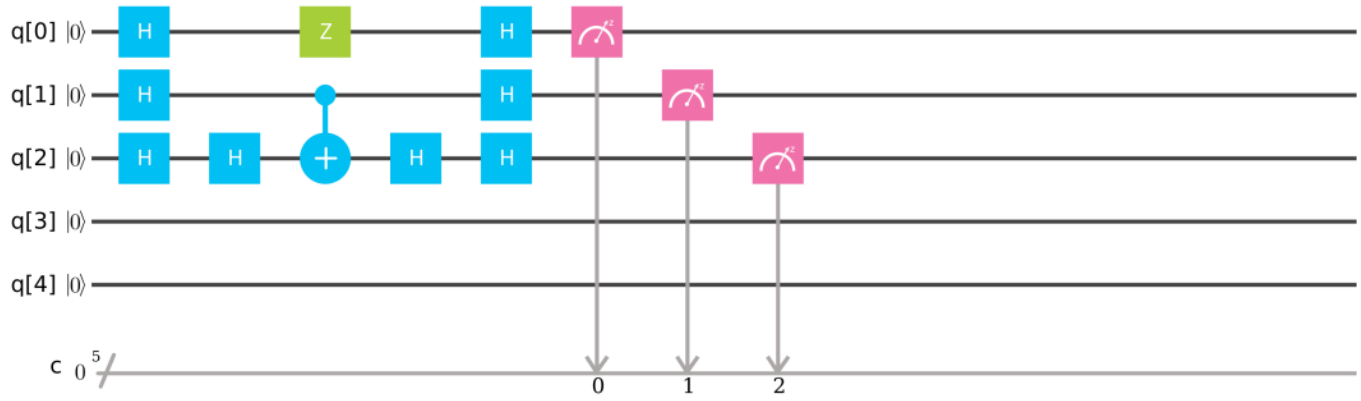
Step 3. Apply the oracle circuit U_f.
Step 4. Repeat Step 2.
Step 5. Measure each qubit. Let $y = (y_1, \ldots, y_n)$ be the list of measurement outcomes.
We find that f is a constant function if y is the all-zeros string. Why does this work? Recall that the Hadamard gate H maps $|0\rangle$ to the uniform superposition of $|0\rangle$ and $|1\rangle$. Thus the state reached after Step 2 is $2^{-n/2} \sum_x |x\rangle$, where the sum runs over all n-bit strings. The oracle circuit maps this state to $2^{-n/2} \sum_x (-1)^{f(x)}|x\rangle$. Finally, let us apply the layer of Hadamards at Step 4. It maps a basis state $|x\rangle$ to a superposition $2^{-n/2} \sum_y (-1)^{x \cdot y}|y\rangle$. Thus the state reached after Step 4 is $|\psi\rangle = \sum_y \psi(y)|y\rangle$, where $\psi(y) = 2^{-n} \sum_x (-1)^{f(x)+x \cdot y}$.

This is exactly what we need for the interference experiment described above. The final measurement at Step 5 plays the role of detecting the particle. As was shown above, the probability to measure $y = 0^n$ at Step 5 is one if f is a constant function and zero if f is a balanced function. Thus we have solved the Deutsch-Jozsa problem with certainty by making just one function evaluation.

Example circuits

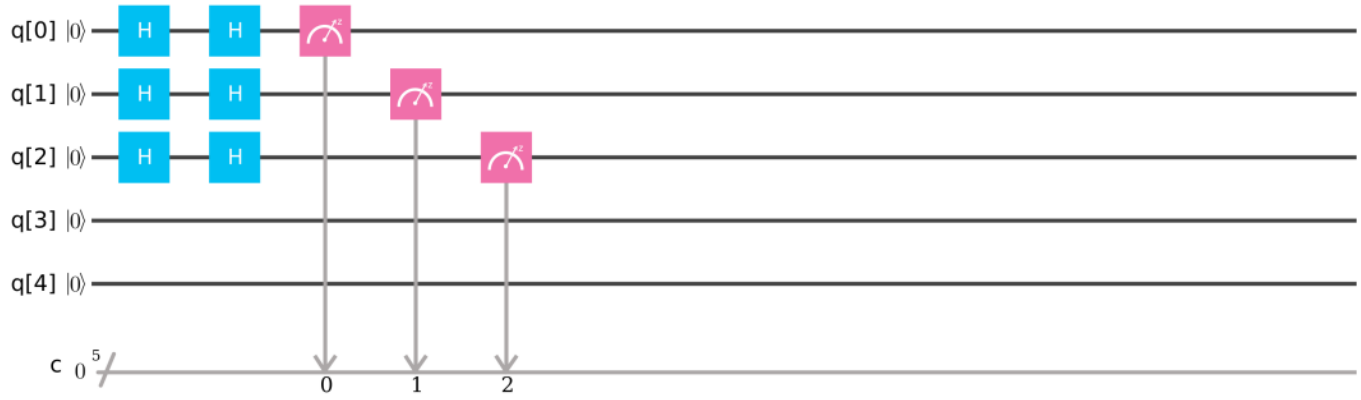
Suppose $n = 3$ and $f(x) = x_0 \oplus x_1 x_2$. This function is balanced since flipping the bit x_0 flips the value of $f(x)$ regardless of x_1, x_2. To run the Deutsch-Jozsa algorithm we need an explicit description of the oracle circuit U_f as a sequence of quantum gates. To this end we need a Z_0 gate such that $Z_0|x\rangle = (-1)^{x_0}|x\rangle$ and a controlled-Z gate $CZ_{1,2}$ such that $CZ_{1,2}|x\rangle = (-1)^{x_1 x_2}|x\rangle$. Using basic circuit identities (see the *Basic Circuit Identities and Larger Circuits* section), one can realize the controlled-Z gate as a CNOT sandwiched between two Hadamard gates.

DJ N=3 Example



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=dda7fb160013ea06bc75d0204439c9a6&sharedCode=true)
Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=dda7fb160013ea06bc75d0204439c9a6&sharedCode=true)

DJ N=3 Constant



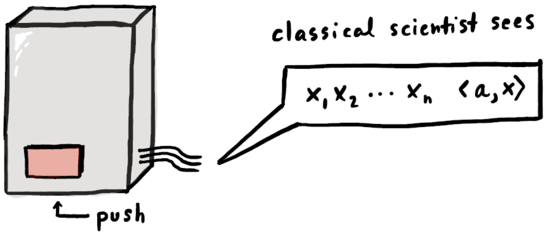
(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=1f234d4750fe47817393d8e1c8f8943d&sharedCode=true)
Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=1f234d4750fe47817393d8e1c8f8943d&sharedCode=true)

Learning Parity with Noise

Imagine you discover an alien artifact. It has a single button on it which you can push. Every time you push the button the artifact emits $n + 1$ qubits. A scientist who does not know any quantum mechanics determines that the (to him) bits emitted have a specific form. The first n bits are completely random.

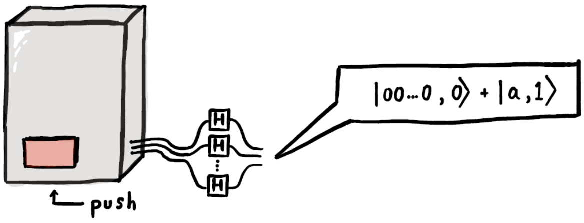
The final bit is determined by $\langle a, x \rangle$ where a is an n-bit string which he does not yet know. $a \cdot x$ means the inner product between a and x defined as $\sum_{j=1}^{n} a_j x_j \mod 2$ where a_j and x_j are the j'$thbitsof$: $math$: 'a and x. We say that the artifact is an *example oracle for the parity function*.
The goal in this example is to learn the value of a. The classical scientist can do this by obtaining just a few more than n examples from the oracle. To see this, imagine the scientist got lucky and read out a string like $0000100000|1$. The bit to the right of the $|$ is the special "last" bit. This then immediately tells him that the 5th bit of a is 1. If the last bit had been 0, you would know that the 5th bit of a was 0. Now, because the whole problem is *linear*, any example yields one bit of information about a. Each additional

example will yield another bit of information about a, provided it is linearly independent of the previous examples. For instance, if you get the same bitstring out of the oracle that you have seen before, it obviously doesn't teach you anything new. Actually calculating a can be done efficiently from the $O(n)$ examples using Gaussian elimination (https://en.wikipedia.org/wiki/Gaussian_elimination).



(

Unlike the classical scientist, we know quantum mechanics. The state that actually emerges from the oracle is $\sum_x |x, a \cdot x\rangle$. If we were to measure in the computation basis, this would result in the distribution seen classically. But if we simply apply the Hadamard gate to all the qubits, the resulting state is $\frac{1}{\sqrt{2}}(|0^n, 0\rangle + |a, 1\rangle)$. Thus, if we measure the first n bits, we can just read out the value of a half the time! Furthermore, the state of the last bit tells us whether to expect to read out 0^n or a. So we need only $O(2)$ queries and no special Gaussian elimination classical computation at the end!



(

That's already a pretty stark difference between the classical and quantum case. In fact, this is the example oracle version of the Bernstein-Vazirani algorithm, itself a refinement of the Deutsch-Jozsa algorithm found earlier in the tutorial. But when the oracle is noisy, as all practical systems are, the difference becomes even stranger. Usually, noise makes things worse for quantum computers much more than for their classical counterparts. In this problem, the noise makes things much worse for the classical scientist, but not so much for us quantum scientists.

The problem for the classical scientist is that when some of the bits are wrong, the Gaussian elimination algorithm no longer works. The problem becomes equivalent to the problem of decoding a random linear code, which is believed to be outside of P. The best known algorithm becomes computationally intractable for a few hundred bits. This is true even for relatively tiny amounts of noise.

The quantum scientist can just carry on as before, needing just a few more examples from the oracle. Provided the noise isn't too large, each bit of the output is still correct more than half the time. By keeping track of the results for each bit separately and using whichever result (0 or 1) is most common for each bit, the correct answer can be determined in a straightforward manner.

Implementing this algorithm is simple. Making a parity oracle on our quantum computer is very easy, requiring only CNOT gates, and the learning algorithm requires only Hadamards and measurements. See if you can determine the hidden value of a in the following circuit. Note that the special "last" bit is in the middle due to the limitations of our current actual architecture. The example oracle is everything up to the final set of Hadamard gates, which themselves are the quantum algorithm.

LPN circuit 2

(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=d76fea8f02071aff9e33ff4a325a9ecb&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=d76fea8f02071aff9e33ff4a325a9ecb&sharedCode=true)

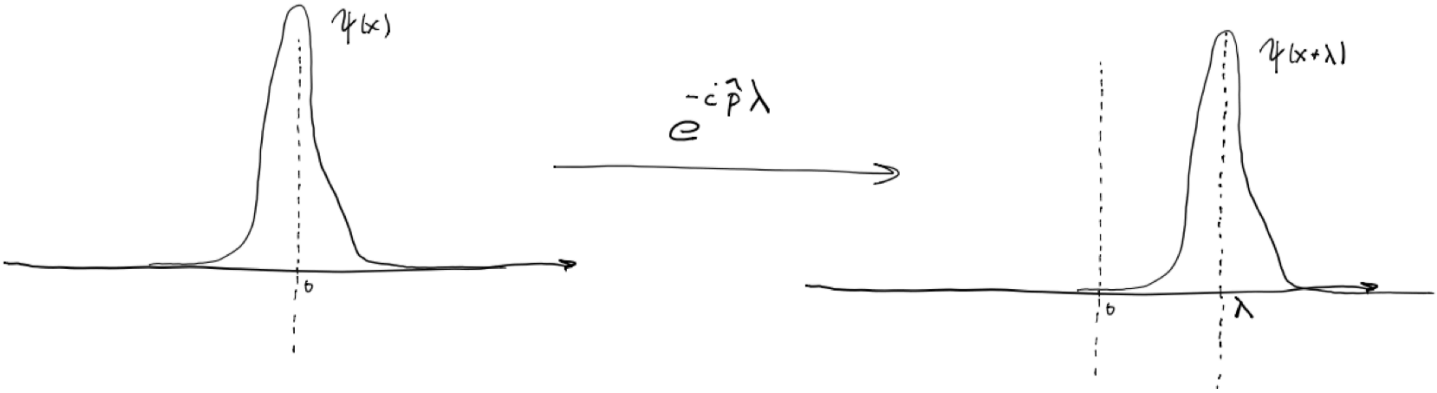
Quantum Phase Estimation

Quantum phase estimation is one of the most important subroutines in quantum computation. It serves as a central building block for many quantum algorithms and implements a measurement for essentially any Hermitian operator (https://en.wikipedia.org/wiki/Hermitian_adjoint). Recall that a quantum computer initially only permits us to measure individual qubits. If we want to measure a more complex observable, such as the energy described by a Hamiltonian H, we resort to quantum phase estimation. The routine prepares an eigenstate of the Hermitian operator in one register and stores the corresponding eigenvalue in a second register.

John von Neumann's measurement scheme

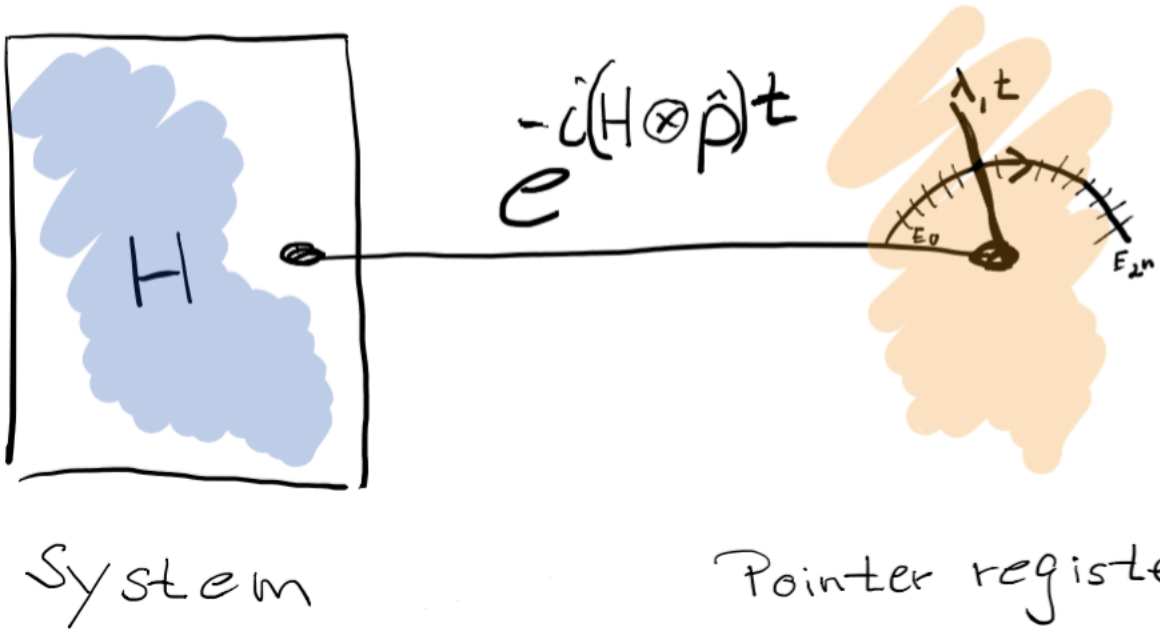
Quantum phase estimation is a discretization of von Neumann's prescription to measure a Hermitian observable $H = \sum_j E_j |\psi_j\rangle\langle\psi_j|$. The scheme that von Neumann envisioned is the following. We consider a quantum system that supports the observable H, which we want to measure. We assume that we are only able to measure simpler observables, in our case single qubits, or as in the original setting the location of a single particle. It is therefore our goal to reduce the measurement of the complex

observable H to a measurement of the simpler observable, e.g. the location. This simple observable is then referred to as the pointer. To map the complex observable on to the simpler one we'll make use of a convenient observation from quantum mechanics. It is known that the momentum operator $\hat{p}$ generate shifts for single particles.



That is, if we apply the unitary $\exp(-ip\hat{\lambda})$ to some wave packet $\psi(x)$, then this wave packet will be shifted by λ in the positive direction.

The scheme now assumes that we can apply the unitary evolution $\exp(-iH \otimes p\hat{i})$ to both the system and the pointer register as illustrated in the following picture



This picture essentially describes von Neumann"s measurement scheme. We now follow the steps and first adjoin an ancilla – the pointer – which is a continuous quantum variable initialized in the state $|0\rangle$ (the origin), so that the system+pointer is initialized in the state $|\psi\rangle|0\rangle$, where $|\psi\rangle$ is the initial state of the system. Then we evolve according to the new Hamiltonian $K = H \otimes \hat{p}$ for a time t, so the evolution is given by

$$e^{-itH\otimes\hat{p}} = \sum_{j=1}^{2^N} |\psi_j\rangle\langle\psi_j| \otimes e^{-itE_j\hat{p}}.$$

We now observe the action of this measurement apparatus. Suppose that $|\psi\rangle$ is an eigenstate (https://en.wikipedia.org/wiki/Introduction_to_eigenstates) $|\psi_j\rangle$ of H, we find that the system evolves to $e^{-itH\otimes\hat{p}}|\psi_j\rangle|0\rangle = |\psi_j\rangle|x = tE_j\rangle$. A measurement of the position of the pointer with sufficiently high accuracy will provide an approximation to E_j.

The quantum algorithm

To carry out the above operation efficiently on a quantum computer, we discretize the pointer using r qubits, replacing the continuous quantum variable with a 2^r-dimensional space, where the computational basis states $|z\rangle$ of the pointer represent the basis of momentum eigenstates of the original continuous quantum variable. The label z is the binary representation of the integers 0 through $2^r - 1$. In this representation, the discretization of the momentum operator becomes

$\hat{p} = \sum_{j=1}^{r} 2^{-j} \frac{\mathbb{1}-\sigma_j^z}{2}$. xx
With this normalization $\hat{p}|z\rangle = \frac{z}{2^r}|z\rangle$. Now the discretized Hamiltonian $K = H \otimes \hat{p}$ is a sum of terms involving at most $k + 1$ qubits, if H is a Hamiltonian involving terms of at most k qubits. Thus we can simulate the dynamics of K using standard methods. In terms of the momentum eigenbasis the initial (discretized) state of the pointer is written $|x = 0\rangle = \frac{1}{2^{r/2}} \sum_{z=0}^{2^r-1} |z\rangle$. This state can be prepared efficiently on a quantum computer by first initializing the qubits of the pointer in the state $|0\rangle \cdots |0\rangle$ and applying an (inverse) quantum Fourier transform (https://en.wikipedia.org/wiki/Quantum_Fourier_transform). Since we have a very simple initial state, the Fourier transform can be represented by a product of Hadamard matrices. The discretized evolution of the system+pointer now can be written

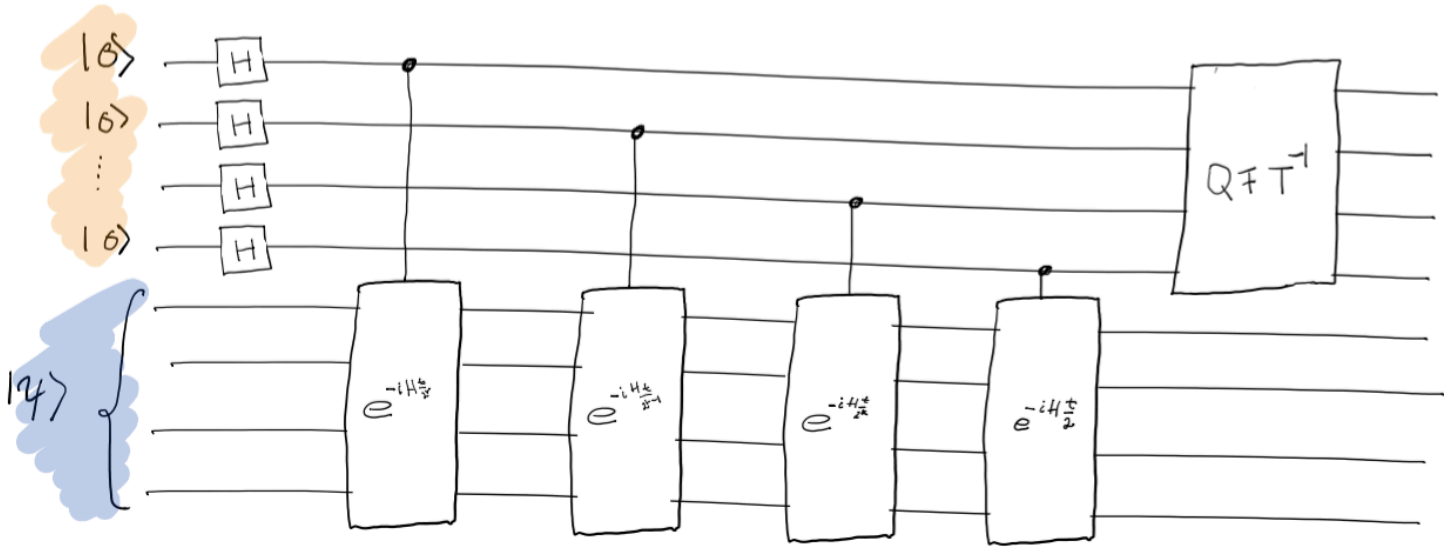
$$e^{-itH\otimes\hat{p}}|\psi_j\rangle|x = 0\rangle = \frac{1}{2^{r/2}} \sum_{z=0}^{2^r-1} e^{-iE_j z t/2^r} |\psi_j\rangle z\rangle.$$

Performing an inverse quantum Fourier transform on the pointer leaves the system in the state $|\psi_j\rangle \otimes |\phi\rangle$, where

$$|\phi\rangle = \sum_{x=0}^{2^r-1} \left(\frac{1}{2^r} \sum_{z=0}^{2^r-1} e^{\frac{2\pi i}{2^r}\left(x - \frac{E_j t}{2\pi}\right)z} \right) |x\rangle,$$

which is strongly peaked near the location $x = \lfloor \frac{E_j t}{2\pi} \rceil$. To ensure that there are no overflow errors we need to choose $t < \frac{2\pi}{\|H\|}$. (We assume here, for simplicity, that $H \geq 0$.)

It is easy to see that actually performing the simulation of K for $t = 1$ is a product of r simulations of the evolution according to $\frac{1}{2^r} H \otimes \frac{\mathbb{1}-\sigma_k^z}{2}$ for $1, 2, 2^2, \ldots, 2^{r-1}$ units of time, respectively. This results in the general circuit for quantum phase estimation:

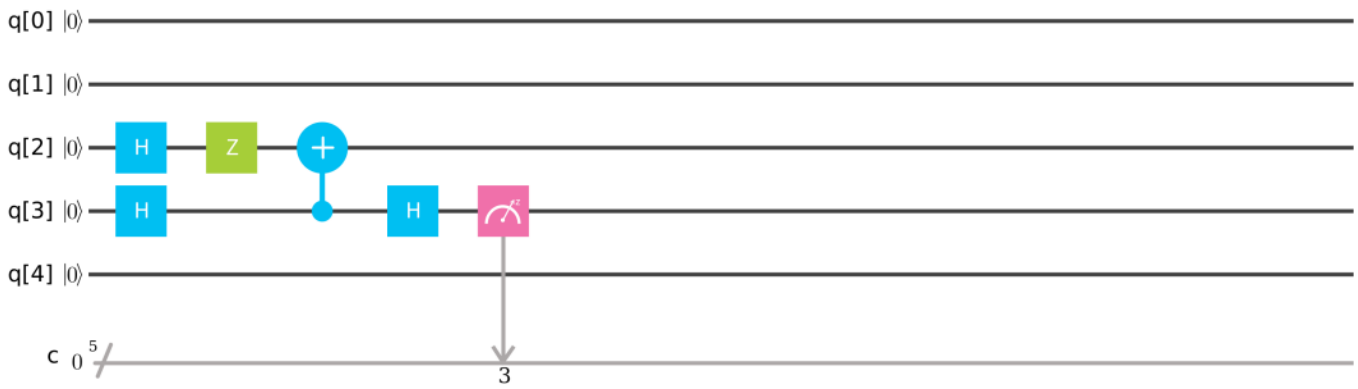


In order to implement the full circuit on a quantum computer, we still need to decompose the controlled unitaries $e^{-iH\frac{t}{2^k}}$ as well as the inverse quantum Fourier transform denoted by QFT^{-1} into our elementary gates.

Example circuit

The example below demonstrates quantum phase estimation for a toy single-qubit Hamiltonian σ^x acting on qubit Q_2. Qubit Q_3 serves as a pointer system. In this example the quantum Fourier transform on the pointer system is equivalent to the Hadamard gate H on Q_3. The discretized evolution of the system+pointer system is described by the CNOT gate. The final measurement outcome on the pointer qubit Q_3 is 0 or 1 depending on whether Q_2 is prepared in the $+1$ or -1 eigenstate of σ^x. In this example, qubit Q_2 is initialized in a state $ZH|0\rangle$ which is -1 eigenvector σ^x. Accordingly, the measurement outcome is 1.

Phase Estimation Circuit (-)



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=bc8e23382f4bb64da16c0a4579f9dc8a&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=bc8e23382f4bb64da16c0a4579f9dc8a&sharedCode=true)

Phase Estimation Circuit (+)

(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=b5a0e7376ded40cd7dc1022e778ebd71&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=b5a0e7376ded40cd7dc1022e778ebd71&sharedCode=true)

Shor's algorithm

Although any integer number has a unique decomposition into a product of primes, finding the prime factors is believed to be a hard problem. In fact, the security of our online transactions rests on the assumption that factoring integers with a thousand or more digits is practically impossible. This assumption has been challenged in 1995 when Peter Shor proposed a polynomial-time quantum algorithm for the factoring problem. Shor's algorithm is arguably the most dramatic example of how the paradigm of quantum computing changed our perception of which problems should be considered tractable. In this section we briefly summarize some basic facts about factoring, highlight main ingredients of the Shor's algorithm, and illustrate how it works using a toy factoring problem.

Complexity of factoring.

Suppose our task is to factor an integer N with d decimal digits. The brute force algorithm goes through all primes p up to $\sqrt{N}$ and checks whether p divides N. In the worst case, this would take time roughly $\sqrt{N}$ which is exponential in the number of digits d. A more efficient algorithm known as the quadratic sieve attempts to construct integers a, b such that $a^2 - b^2$ is a multiple of N. Once such a, b are found, one checks whether $a \pm b$ have common factors with N. The quadratic sieve method has asymptotic runtime exponential in $\sqrt{d}$. The most efficient classical factoring algorithm known as general number field sieve achieves an asymptotic runtime exponential in $d^{1/3}$. The exponential runtime scaling limits applicability of the classical factoring algorithms to numbers with a few hundred digits with the world record being $d = 232$ (which took roughly 2,000 CPU years). In contrast, Shor's factoring algorithm has runtime *polynomial* in d. The version of the algorithm described below due to Alexey Kitaev requires roughly $10d$ qubits and has runtime roughly d^3.

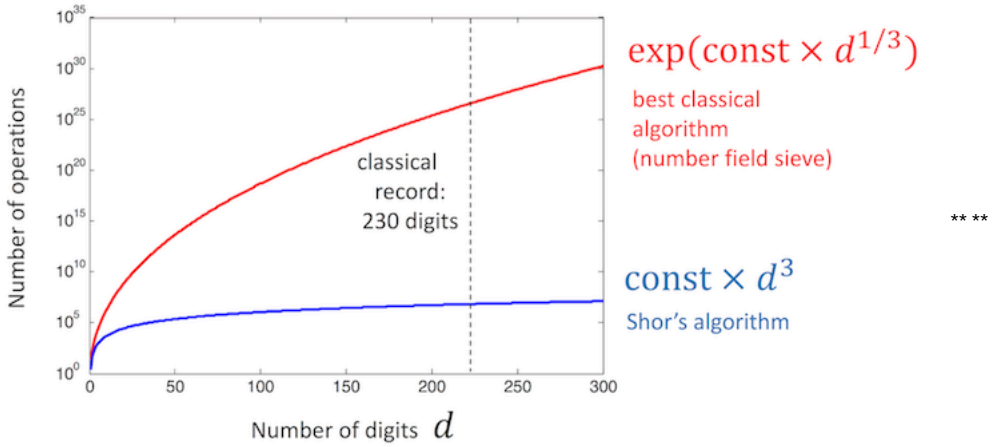


Figure 1: classical vs quantum factoring algorithms

Period finding.

It has been known to mathematicians since 1970's that factoring becomes easy if one can solve another hard problem: find a period of the modular exponential function. The period finding problem is defined as follows. Given integers N and a, find the smallest positive integer r such that $a^r - 1$ is a multiple of N. The number r is called the period of a modulo N. Recall that in modular arithmetics the remainder of a division a/N is called the value of a modulo N and denoted $a \pmod N$. For example, $1 = 16 = 91 \pmod{15}$. Thus the period of a modulo N is the smallest positive integer r such that $a^r = 1 \pmod N$. For example, suppose $N = 15$ and $a = 7$. Then

$$
\begin{aligned}
7^2 &= 4 \pmod{15} \\
7^3 &= 4 \cdot 7 = 13 \pmod{15} \\
7^4 &= 13 \cdot 7 = 1 \pmod{15}
\end{aligned}
$$

that is, 7 has period 4 modulo 15. Note that computing the higher powers of 7 would give rise to a periodic sequence: $7^{x+4} = 7^x \pmod{15}$ for any integer x. Thus $r = 4$ is the period of the modular exponential function 7^x. In general the period finding problem is well-defined if N and a are co-prime (have no common factors).

From factoring to period finding.

Assume for a moment that we are given a period finding machine that takes as input co-prime integers N, a and outputs the period of a modulo N. Let us show how to use the machine to find all prime factors of N. For simplicity, assume that N has only two distinct prime factors:

$$
N = p_1 p_2
$$

First, pick a random integer a between 2 and $N - 1$ and compute the greatest common divisor gcd:math:*(N,a)*. This can be done very efficiently using Euclid's algorithm (http://en.wikipedia.org/wiki/Euclidean_algorithm). If we are lucky, N and a have some common prime factors in which case gcd:math:*(N,a)* equals p_1 or p_2, so we are done. From now on let us assume that gcd:math:*(N,a)=1*, that is, N and a are co-prime. Let r be the period of a modulo N computed by the machine. Repeat the above steps with different random choices of a until r is even. It can be shown that a significant fraction of all integers a has even period, see Table 1 for examples, so on average one needs only a few repetitions. At this point we have found some pair r, a such that r is even and r is the smallest integer such that $a^r - 1$ is a multiple of N. Let us use the identity

$$
a^r - 1 = (a^{r/2} - 1)(a^{r/2} + 1)
$$

The above shows that $a^{r/2} - 1$ is not a multiple of N (otherwise the period of a would be $r/2$). Assume for a moment that $a^{r/2} + 1$ is not a multiple of N. Then neither of the integers $a^{r/2} \pm 1$ is a multiple of N, but their product is. This is possible only if p_1 is a prime factor of $a^{r/2} - 1$ and p_2 is a prime factor of $a^{r/2} + 1$ (or vice verse). Thus we can find p_1 and p_2 by computing gcd:math:*(N,a^{r/2}pm 1)*, see Table 1 for examples. In the remaining ``unlucky" case when $a^{r/2} + 1$ is a multiple of N we give up and try a different integer a. For example, $a = 14$ is the only unlucky integer in Table 1. In general, it can be shown that the unlucky integers a are not too frequent, so on average only two calls to the period finding machine are sufficient to factor N.

a	Period r	$\gcd(15, a^{r/2}-1)$	$\gcd(15, a^{r/2}+1)$
1	1		
2	4	3	5
4	2	3	5
7	4	3	5
8	4	3	5
11	2	5	3
13	4	3	5
14	2	1	15

Table 1: period of integers :math:`a` modulo :math:`15`

Shor's algorithm.

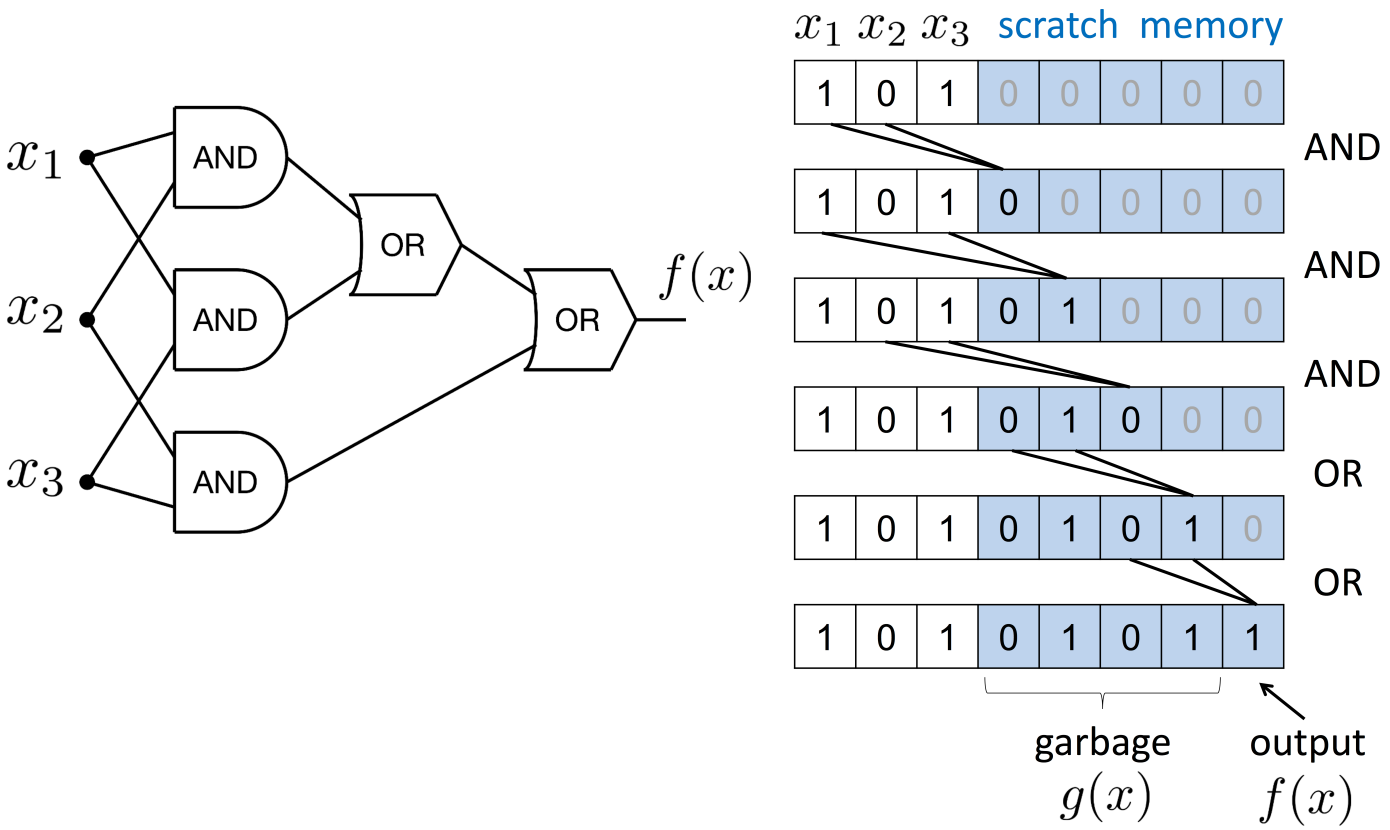
Let us now show that a quantum computer can efficiently simulate the period finding machine. As in the case of theDeutsch-Jozsa algorithm (/qstage/#/tutorial?sectionId=8443c4f713521c10b1a56a533958286b&pageIndex=3), we shall exploit quantum parallelism and constructive interference to determine whether a complicated function has a certain global property that cannot be learned by evaluating the function only at a few points. However, instead of detecting the property of being a balanced function, we seek to detect and measure periodicity of the modular exponentiation function. The fact that interference makes it easier to measure periodicity should not come as a big surprise. After all, physicists routinely use scattering of electromagnetic waves and interference measurements to determine periodicity of physical objects such as crystal lattices. Likewise, Shor's algorithm exploits interference to measure periodicity of arithmetic objects.

Suppose we are given co-prime integers a, N. Our goal is compute the period of a modulo N, that is, the smallest positive integer r such that $a^r = 1 \pmod N$. The basic idea is to construct a unitary operator U_a that implements the modular multiplication function $x \to ax \pmod N$. It can be shown that eigenvalues of U_a are closely related to the period of a. Namely, each eigenvalue of U_a has a form $e^{i\phi}$, where $\phi = 2\pi k/r$ for some integer k. Furthermore, as we saw in the previous section, eigenvalues of certain unitary operators can be measured efficiently using the phase estimation algorithm. Unfortunately, inferring r from the measured eigenvalues of U_a is only possible if the eigenvalues are measured *exactly* (or with an exponentially small precision). For example, factoring a 1,000-digit number would require measuring the eigenvalue of U_a with a precision 10^{-2000}. Such accuracy cannot be achieved by a direct application of the phase estimation algorithm as this would require too large pointer system. Here comes the main trick: we shall estimate the eigenvalue of U_a by applying the phase estimation algorithm to a family of unitary operators U_b with $b = a, a^2, a^4, a^8$ etc. We stop at $b = a^{2^p}$ with $2^p \approx N^2$. Why does it work? The first observation is that all operators U_b are integer powers of U_a. Namely, if $b = a^t$ then $U_b = (U_a)^t$. This implies that the operators U_b have the same eigenvectors. In particular, eigenvalues of the entire family U_b can be measured simultaneously. Second, implementing U_b is as easy as implementing U_a - one just need to precompute the powers $b = a, a^2, a^4, a^8, \ldots \pmod N$ by the repeated squaring method. Finally, even if the eigenvalues of U_b are measured with a poor precision (say 10%), each squaring of a reduces the error in the estimated eigenvalue of U_a by a factor $1/2$. Indeed, consider an eigenvector of U_a with an eigenvalue $e^{i\phi}$. If $b = a^2$ then the eigenvalue of U_b is $e^{2i\phi}$. If $b = a^4$ then the eigenvalue of U_b is $e^{4i\phi}$ etc. Thus we can estimate $\phi, 2\phi, 4\phi, \ldots, 2^p\phi$ with a constant precision (say 10%). We shall see that this is enough to estimate ϕ with a precision roughly 2^{-p}. For example, one can achieve a precision 10^{-2000} by a sequence of less than 10^6 lousy measurements of U_b with an error 10%. Furthermore, it can be shown that estimating a few randomly picked eigenvalues $\phi = 2\pi k/r$ with a precision less than $1/N^2$ is enough to determine the period r exactly (the idea is to find the best rational approximation to the estimate of k/r using continued fractions).

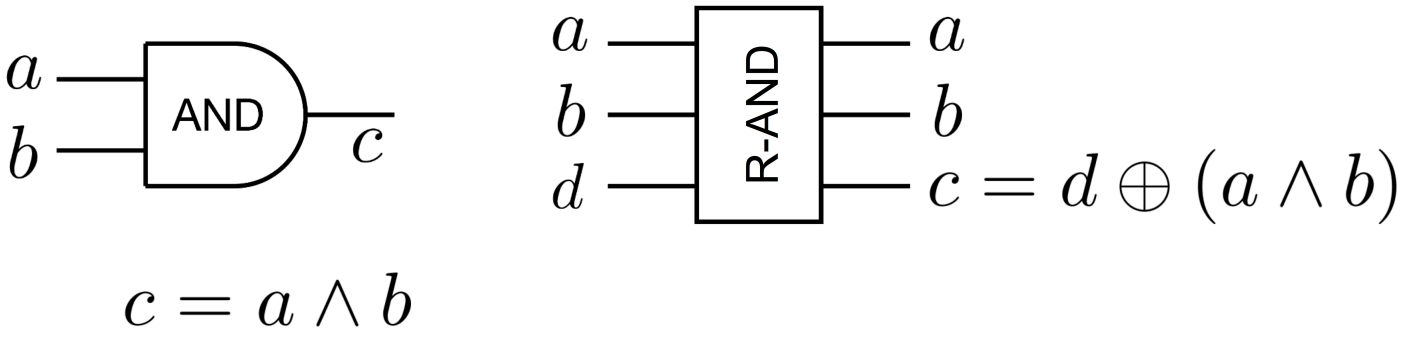
In order to use the phase estimation algorithm we need to construct a quantum circuit implementing the modular multiplication operator. By analogy with classical algorithms that can link standard library functions, a quantum algorithm is allowed to call classical subroutines, for example a subroutine for computing the modular multiplication. Importantly, before such classical subroutines are incorporated into a quantum circuit, they must be transformed into a *reversible form.*More precisely, a quantum algorithm can call a classical subroutine only if it is compiled into a sequence of reversible logical gates such as CNOT or Toffoli gate (in particular, the number of input and output wires in each gate must be the same). The subroutine is allowed to use a scratch memory similar to local variables used by the standard library functions. However, once the subroutine is completed, the scratch memory must be totally clean (say, all zeros). The reason is that a quantum algorithm operates on coherent superpositions of different classical states. Leaving information about the inputs or the outputs in the scratch memory could potentially destroy quantum coherence and prevent the algorithm from seeing interference between different states. Since the notion of reversible classical circuits plays an important role in the Shor's algorithm and many other quantum algorithms, below we briefly discuss methods for constructing such circuits.

Reversible classical circuits.

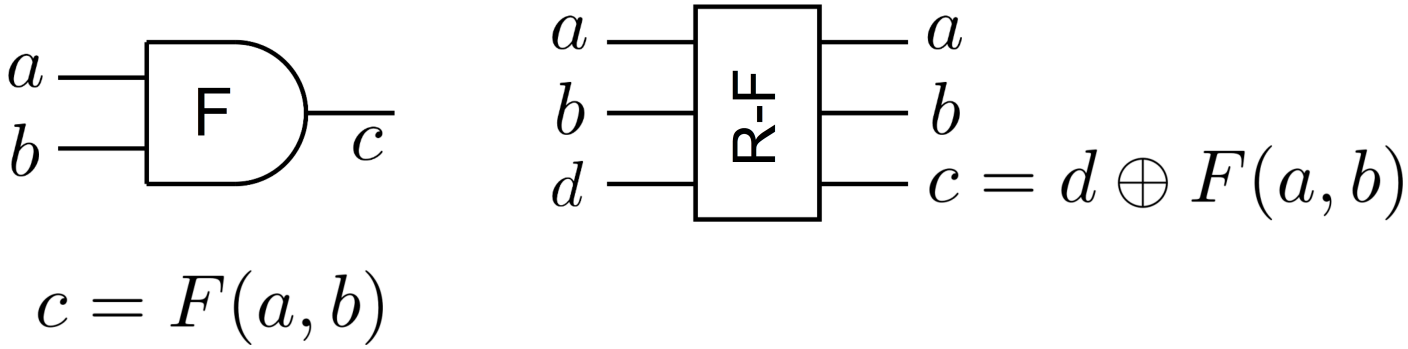
An important insight made in 1973 by our IBM colleague Charles Bennett is that any classical computation can be transformed into a reversible form. How does it work? Suppose $f(x)$ represents some classical computation that takes as input n-bit strings x and outputs m-bit strings $f(x)$. The first observation is that the answer $f(x)$ can be computed without erasing any intermediate data if we are allowed to use some extra memory. Indeed, let us write down an algorithm for computing $f(x)$ and compile it into a sequence of elementary logical gates such as AND, OR, etc. For concreteness, assume that each gate has two input wires and one output wire. Let L be the total number of gates. We shall extend the n-bit memory storing the input x by adding L bits initialized by zeros. These extra bits will serve as a scratch memory for storing intermediate data. We shall write the output of the i-th gate to the i-th bit of the scratch memory and keep the values of the input bits. Once the computation is completed, the final answer $f(x)$ is contained in some designated output register within the scratch memory. The remaining part of the scratch memory contains some "garbage" bit string $g(x)$ (intermediate data). Below we illustrate how it works for the example when $f(x)$ computes the 3-bit Majority function.



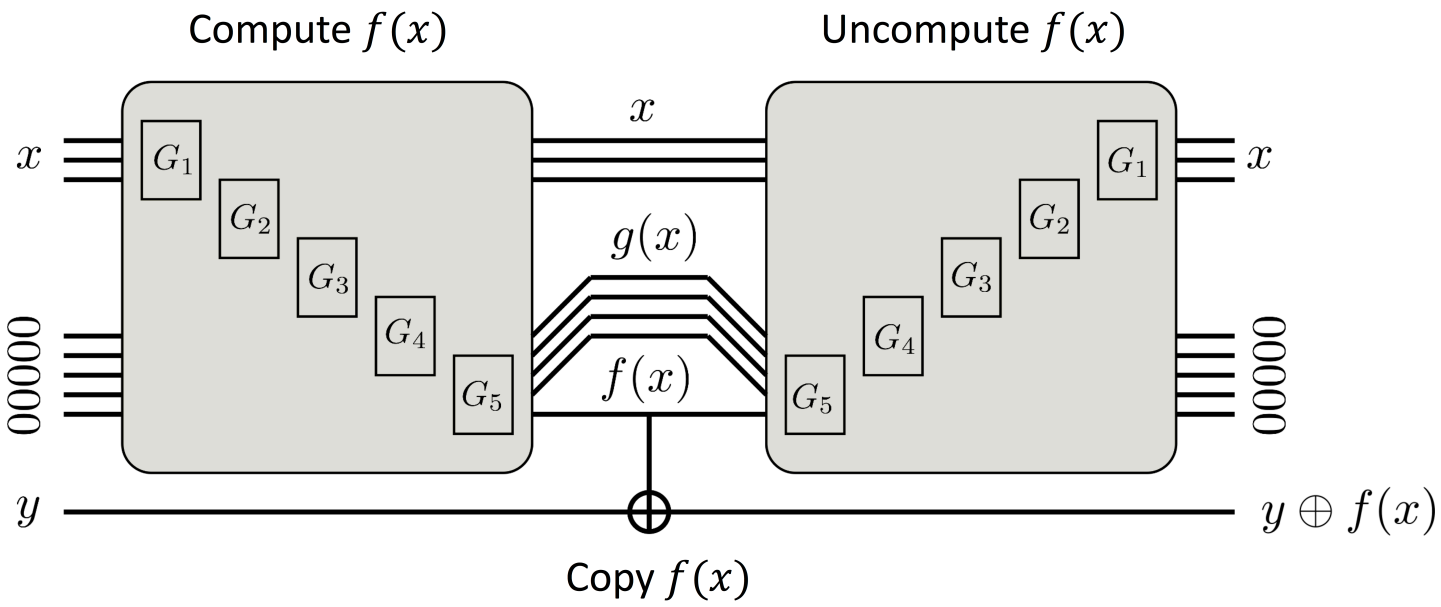
At this point the circuit is reversible as a whole, but its individual gates are still irreversible. The next step is to transform each gate into a reversible form. Consider as an example the AND gate with input wires a, b and output wire c such that $c = a \wedge b$. Let us define its reversible version R-AND. One of the output wires of R-AND must carry the output bit c of the standard AND gate. To avoid losing information, R-AND must have at least two other output wires (note that in the case $c = 0$ there are three possible input strings: $ab = 00, 01, 10$). The simplest version of R-AND has three input wires and three output wires as shown below.



Here d is a dummy input wire and $\oplus$ denotes XOR operation (addition modulo two). The gate expects to receive inputs with $d = 0$ in which case $c = a \wedge b$. If $d = 1$ then the output data bit if flipped. Note that all inputs of R-AND can be computed from its outputs since $d = c \oplus (a \wedge b)$. Thus R-AND indeed acts reversibly (technically, R-AND realizes a permutation on the set of 3-bit strings). Note also that R-AND coincides with the Toffoli gate (/qstage/#/tutorial?sectionId=8443c4f713521c10b1a56a533958286b&pageIndex=1). The same construction can be applied to any other gate with two input wires and one output wire. Namely, if a gate F computes some Boolean function $c = F(a, b)$ then its reversible version R-F would map inputs a, b, d to outputs a, b, c where $c = d \oplus F(a, b)$, see below. Note that applying R-F twice implements the identity gate, that is, R-F coincides with its own inverse.



Suppose the original circuit is described by a sequence of L gates $F_1, \ldots, F_L$. Replace each gate F_i by its reversible version $G_i = R\text{-}F_i$ constructed above. We shall connect the dummy input wire of G_i and its output wire c to the i-th bit of the scratch memory such that the gate always receives inputs with $d = 0$. The new circuit has $n + L$ input and $n + L$ output wires and is composed from reversible 3-bit gates. The final state generated by the circuit can be written as $x, g(x), f(x)$, where $f(x)$ is the final answer stored in the output register somewhere within the scratch memory and $g(x)$ represents ``garbage" (intermediate data). Here we assumed that the scratch memory is initially clean (all zeros). Thus we have constructed a reversible circuit that maps $x, 0^L$ to $x, g(x), f(x)$. The final step is to get rid of the garbage $g(x)$ without erasing any information (which would render the circuit irreversible). A solution is to copy the answer $f(x)$ to a clean ancillary register of m bits and then ``uncompute" $f(x)$ by applying the circuit backwards in time. Below we sketch how this works.



Ignoring for simplicity all ancillary bits that are initialized and returned in the zero state, we obtained a reversible circuit on $n + m$ bits that maps input strings x, y to output strings $x, y \oplus f(x)$. In the special case when the $f(x)$ is invertible one can use similar tricks to construct a reversible circuit that maps input strings x to output strings $f(x)$. In practice, one would never use the method described above since it requires too large scratch memory. Several optimization techniques for constructing reversible circuits have been proposed (such as uncomputing partial results more often and reusing scratch memory bits).

Quantum circuits for modular multiplication.

Suppose now that $f(x) = ax \pmod N$ is the modular multiplication function. Let n be the number of binary digits in N. Using n-bit strings to represent integers modulo N, one can implement $f(x)$ by a classical circuit U_a composed of 3-bit reversible gates with n input and output wires, as described above. The circuit U_a may also use ancillary bits that are initialized and returned in the 0 state. The state-of-the-art implementation would require roughly n^2 gates and roughly $2n$ ancillary bits. For simplicity, below we shall often ignore the ancillary bits. Let us convert U_a to a quantum circuit by replacing each classical gate with its quantum counterpart. This is possible because, by construction, each gate of U_a implements some permutation on the set of input bit strings $000, 001, \ldots, 111$. The corresponding quantum gate implements the same permutation on the set of basis states $|000\rangle, |001\rangle, \ldots, |111\rangle$. We obtained a quantum circuit U_a acting on a register of n qubits that maps a basis state $|x\rangle$ to $|f(x)\rangle$. An example for $f(x) = 7x \pmod{15}$ is shown below. Period finding algorithm requires modular multiplication circuits U_b for $b = a, a^2, a^4, \ldots, a^{2^p} \pmod N$, where $2^p \approx N^2$.

a	Basis vector
1	$\|0001\rangle$
7	$\|0111\rangle$
4	$\|0100\rangle$
13	$\|1101\rangle$

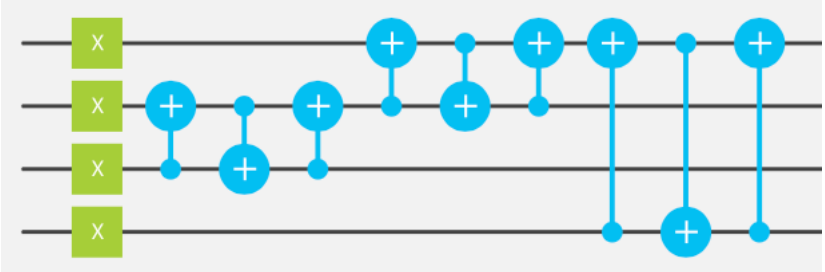
some basis states representing integers modulo :math:`15`

$$U_7 : \begin{cases} |1\rangle \rightarrow |7\rangle \rightarrow |4\rangle \rightarrow |13\rangle \rightarrow |1\rangle \\ |2\rangle \rightarrow |14\rangle \rightarrow |8\rangle \rightarrow |11\rangle \rightarrow |2\rangle \end{cases}$$

Multiplication by 7 modulo 15

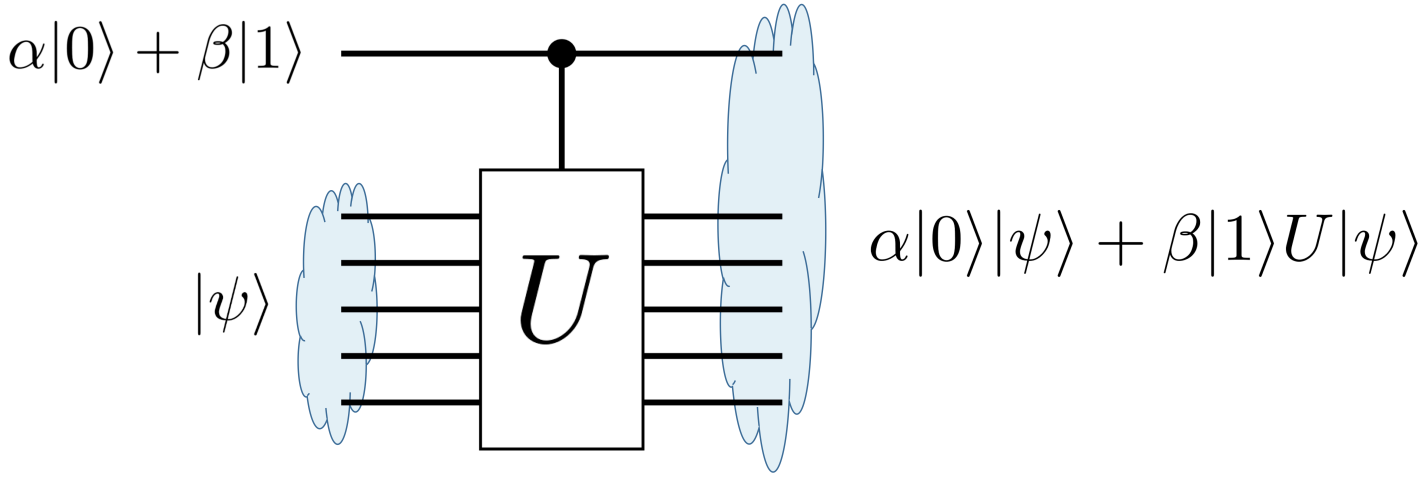
Modular multiplication operator maps :math:`|xrangle` to :math:`|7x pmod{15}rangle`

This quantum circuit implements U_7 (see Markov and Saeedi 2012 (http://arxiv.org/abs/1202.6614))

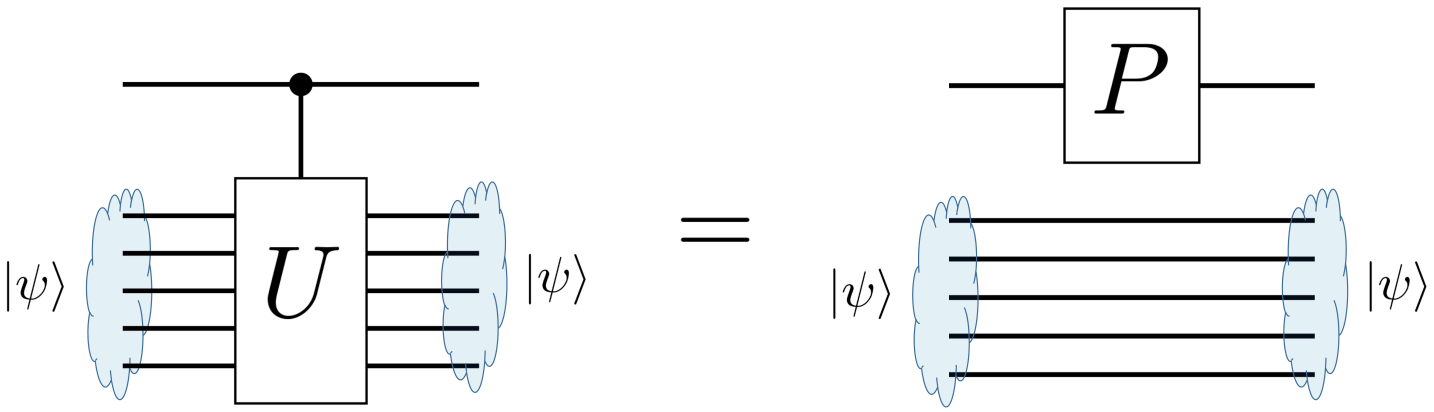


Controlled operations and phase estimation.

Let $U = U_a$ be the modular multiplication operator. At this point we know how to construct a quantum circuit implementing U as well as repeated squares of U such as U^2, U^4, U^8, etc. We also know that eigenvalues of U reveal information about the period of a modulo N. The final step is to measure the eigenvalues. For that we shall need a controlled version of U. A controlled unitary operator is a quantum analogue of classical conditional statements such as if-then-else. We already saw examples of controlled quantum gatesearlier in the tutorial (/qstage/#/tutorial?sectionId=8443c4f713521c10b1a56a533958286b&pageIndex=1). In general, suppose U is a quantum circuit acting on n qubits. A controlled version of U is a unitary operator acting on a larger system control+target, where control is a single qubit and target is a register of n qubits. Controlled-U applies U to the target register if the control qubit is $|1\rangle$ state and does nothing if the control qubit is $|0\rangle$.



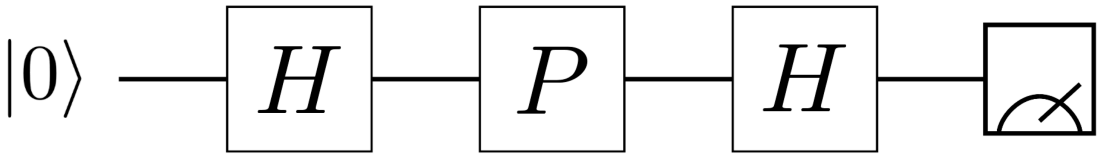
Like their classical counterparts, controlled quantum operations are used in almost any quantum algorithm. We note that if U can be realized by a short quantum circuit then so does controlled-U. Indeed, one can take the circuit realizing U and replace each gate by its controlled version (with the same control qubit). The main distinction from the classical if-then-else construct is that the controlled qubit can be in a superposition of state $\alpha|0\rangle + \beta|1\rangle$. One could say that in the quantum world two branches of a conditional statement can be executed "at the same time". Consider now a special case when the target register is prepared in some state ψ which is an eigenvector of U, that is $U|\psi\rangle = e^{i\phi}|\psi\rangle$. Then the only difference between the two branches of the controlled-U operation is the phase shift $e^{i\phi}$. In other words, the control qubit gets mapped from $\alpha|0\rangle + \beta|1\rangle$ to $\alpha|0\rangle + e^{i\phi}\beta|1\rangle$, while the target register remains in the state ψ. Thus we can describe that the action of controlled-U on the composite system control+target by a single-qubit phase shift gate P acting on the control qubit.

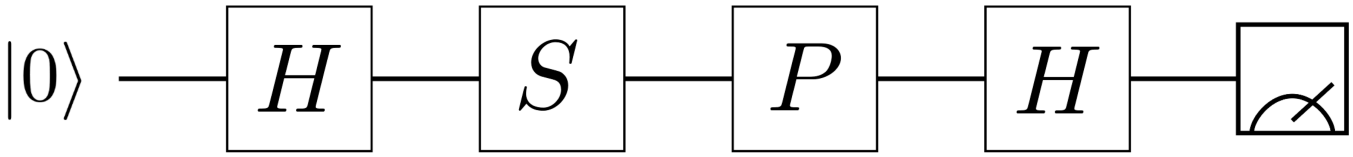


$$U|\psi\rangle = e^{i\phi}|\psi\rangle$$

$$P = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{bmatrix}$$

Below we focus on what happens with the control qubit only (keeping in mind that it is part of the larger system control+target). We shall measure the eigenvalue $e^{i\phi}$ using a pair of phase estimation circuits shown below.



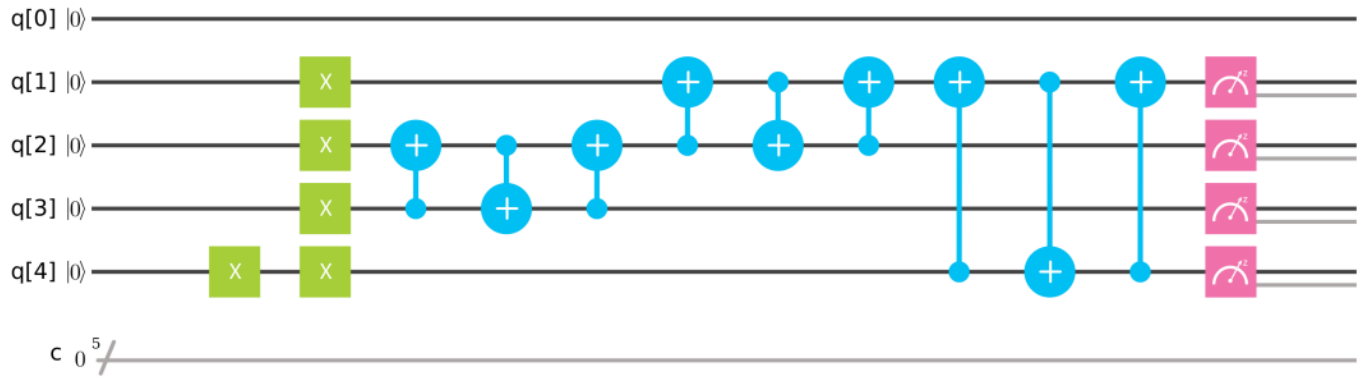


One can easily check that the probability of observing the measurement outcome 0 is $0.5(1+\cos(\phi))$ for the first circuit and $0.5(1-\sin(\phi))$ for the second circuit. One should keep in mind that P represents the controlled-U operator, so the circuit extracts information about the phase ϕ by measuring interference between two branches of controlled-U where one branch accumulates a phase factor $e^{i\phi}$ and the other branch accumulates no phase. By repeating each circuit several time and collecting the measurement statistics we can estimate the probabilities which gives us an estimate ϕ. For concreteness, assume that we are willing to perform at most 100 measurements. Then the statistical error in our estimate of ϕ is roughly 10%.

To factor a number N with 1,000 decimal digits the phase ϕ has to be estimated with a very high precision $\epsilon \sim 1/N^2 \sim 10^{-2000}$. To this end we shall perform the phase estimation for a family of unitary operators U^t, where $t = 1, 2, 4, 8$ etc. We stop at $t = 2^p$ such that $2^p \approx 1/\epsilon$. Recall that we can efficiently implement U^t for very large values of t by classically computing $b = a^t \pmod N$ and using the identity $U^t = (U_a)^t = U_b$. Since all operators U^t have the same eigenvector ψ, we can do all phase estimations with the same target register (initialized in the eigenvector $|\psi\rangle$). For simplicity, let us assume that the phase estimations are performed sequentially in which case only one control qubit is needed. The controlled-U^2 operator gives rise to a phase shift P^2 by angle 2ϕ on the control qubit. Thus we can estimate 2ϕ with a precision 10% by performing roughly 100 measurements. This gives an estimate of ϕ with a precision 5%. More precisely, since the phase ϕ lives on the unit circuit, we get a pair of candidate angles ϕ' and $\phi'' = \phi' + \pi$ such that one of them approximates ϕ with a precision 5% and the other is very far from ϕ (approximately by π). However, we have already estimated ϕ itself with a precision 10%. This is enough to select one of the candidate angles ϕ' and ϕ''. Applying this argument inductively several times shows that estimating $\phi, 2\phi, \ldots, 2^p\phi$ with a constant precision (say, 10%) is enough to estimate ϕ with a precision roughly $2^{-p} \sim \epsilon$. Overall we would need approximately $M = 100\log_2(1/\epsilon) \sim 10^6$ measurements which translates to 10^6 controlled modular multiplication operators. In general, M scales as $\log(N)$ with some extra factors doubly logarithmic in N. Since each controlled modular multiplication operator requires a quantum circuit of size $\log^2(N)$, the overall complexity of the factoring algorithm scales as $\log^3(N) \sim d^3$.

We have not explained yet how to initialize the target register in the eigenvector of U. Fortunately, all eigenvectors are equally good for our purposes: we are not interested in any particular eigenvalue but rather want to measure a random eigenvalue drawn from the uniform distribution. Thus one can initialize the target register in an arbitrary state that has equal weight on each eigenvector of U. For example, one can choose the initial state as the basis vector $|0\ldots01\rangle$ encoding the integer $x = 1$.

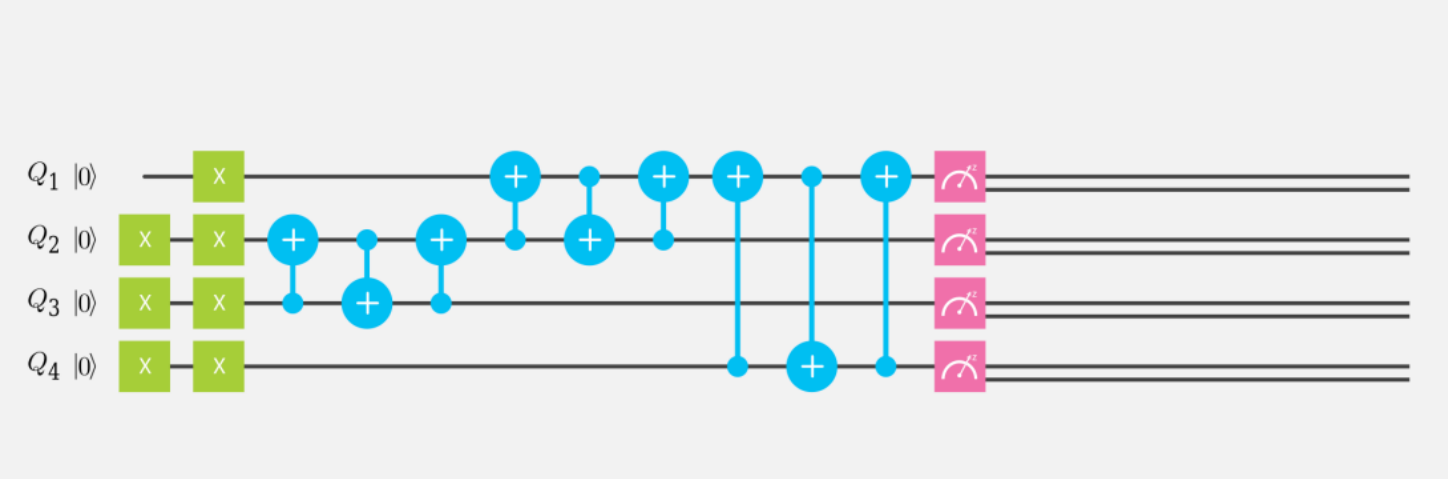
Multi7x1Mod15



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=858317af73c7a5ed31f676db5b15913f&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=858317af73c7a5ed31f676db5b15913f&sharedCode=true)

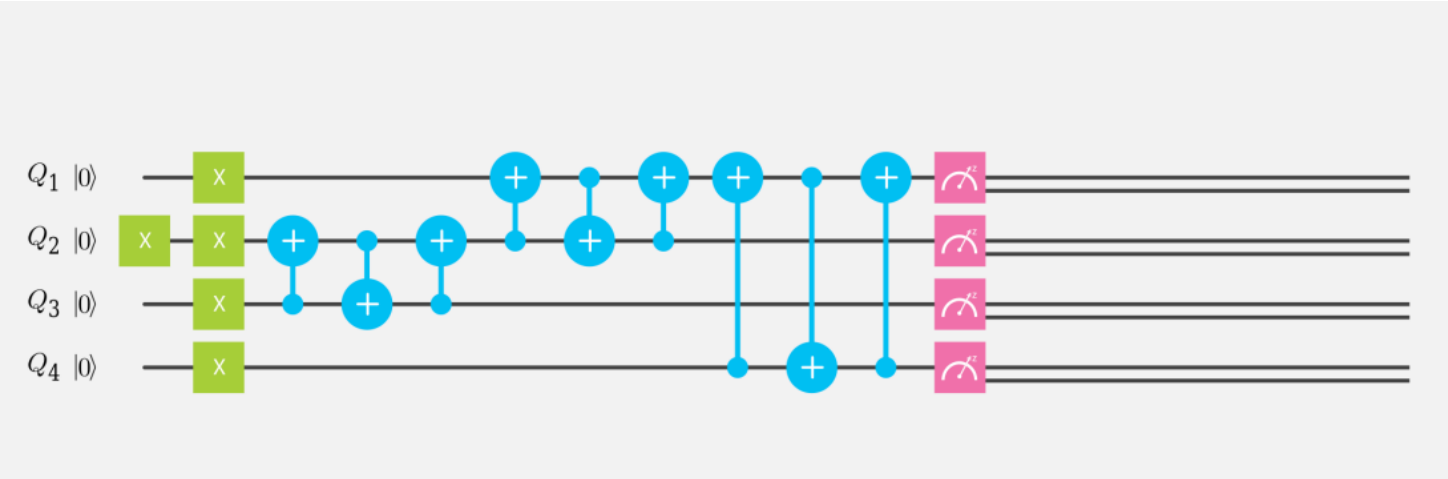
Multi7x7Mod15



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=c97d1b1f88e0615685200e6cd6d4b8d2&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=c97d1b1f88e0615685200e6cd6d4b8d2&sharedCode=true)

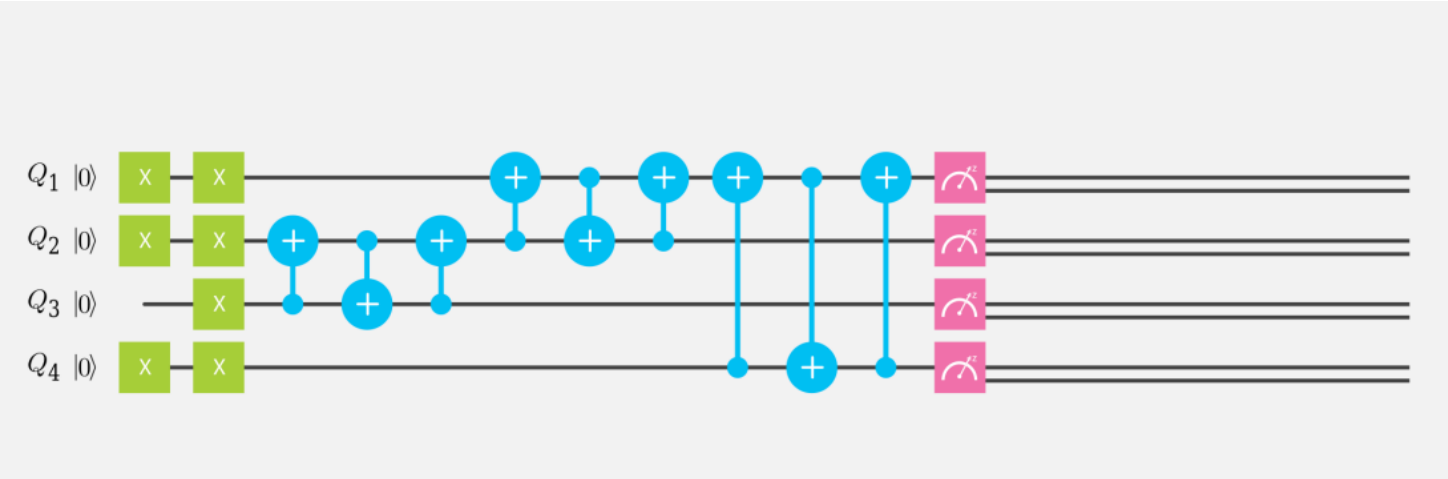
Multi7x4Mod15



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=858317af73c7a5ed31f676db5b62695e&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=858317af73c7a5ed31f676db5b62695e&sharedCode=true)

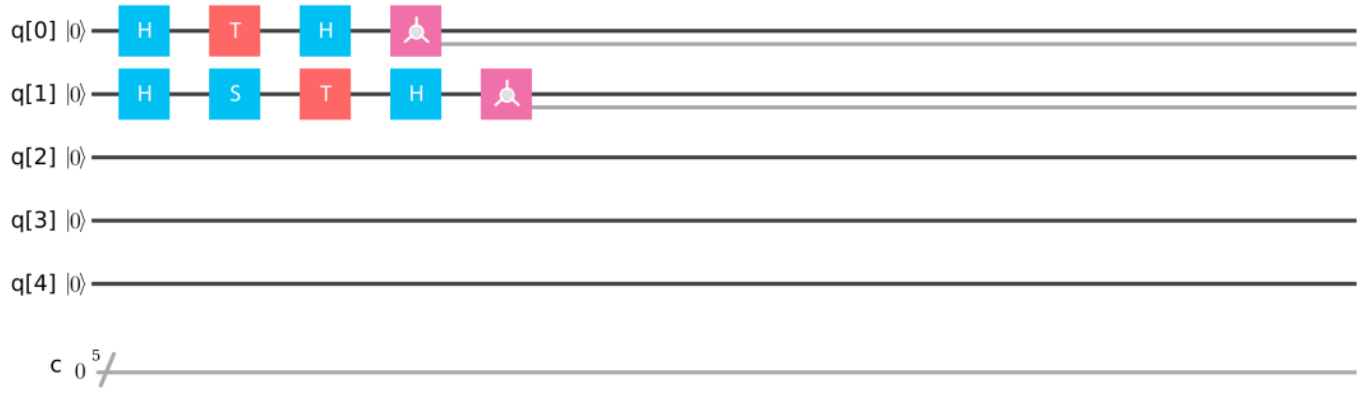
Multi7x13Mod15



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=858317af73c7a5ed31f676db5b9099fd&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=858317af73c7a5ed31f676db5b9099fd&sharedCode=true)

PhaseEstimationTgate



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=0a1742807714ccbf73df68bbef062fae&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=0a1742807714ccbf73df68bbef062fae&sharedCode=true)

In this section we embark on more complex scores and explore the components it takes to construct real quantum algorithms. We go beyond simply defining entanglement and begin to use it in computation, in order to perform some well-known algorithms (with more to come in the future):

- Grover's algorithm
- Deutsch-Jozsa algorithm
- Learning parity with noise
- Phase estimation algorithm
- Shor's Algorithm

Quantum Error Correction

Quantum Repetition Code

Rather than have a general discussion of codes and error-correction, we will instead look at some of the simplest examples. One of the most straightforward error-correcting codes is the *repetition code*. To encode a 1 bit message in the repetition code, we simply copy it several times (say 3 times): 0 encodes to 000 and 1 encodes to 111. Suppose now that we store the *codeword* 000 or 111 in a memory for a period of time and, during that time, errors occur. A simple model of errors is to suppose that each bit can flip randomly with some probability $p < 1/2$ and that the error process acts independently on each bit. If one error occurs, the stored codeword becomes one of $\{001, 010, 100\}$ or one of $\{110, 101, 011\}$, respectively. Given an encoded message abc, the original message is the value indicated by a majority of the bits $\mathrm{MAJ}(a, b, c) = ab \oplus bc \oplus ca$. For example, if we read 001 from the memory, the majority value is obviously 0, but we could also have computed it from the formula for $\mathrm{MAJ}(0, 0, 1)$. This recovery procedure works if only one error occurs; it fails otherwise. However, correcting even a single error is enough to reduce the probability of failure, since the probability of more than one error is $3p^2(1 - p) + p^3 = 3p^2 - 2p^3$, which is less than p whenever $p < 1/2$. The reduction in error rate can be surprisingly large: if p is 1 percent, the failure probability after encoding is less than 0.03 percent.

The repetition code is a classical error-correcting code, but there is a closely related quantum repetition code that is one of the simplest quantum codes. Again, rather than defining quantum codes in general, we will describe a 3-qubit quantum bit-flip code and look at how to encode, decode, and detect errors.

To encode a single-qubit message $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ in the 3-qubit quantum bit-flip code, we apply a quantum circuit that encodes the messages 0 and 1 in superposition so that $|\psi\rangle$ encodes to $\alpha|000\rangle + \beta|111\rangle$. A very important quantum result is that, for arbitrary α, we cannot create identical copies of $|\psi\rangle$, like $(\alpha|0\rangle + \beta|1\rangle)(\alpha|0\rangle + \beta|1\rangle)(\alpha|0\rangle + \beta|1\rangle)$, which is the way the classical repetition code would operate. Instead, we can make repetitions with the codewords. Now, when the message is an equal superposition $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, the encoded message happens to be an entangled state, since it cannot be written as a tensor product of two or more states. The "*Encoder into bit-flip code*" example encodes qubit 2 into the quantum bit-flip code. The first three gates in the example prepare qubit 2 in the state $\cos(\pi/8)|0\rangle - i\sin(\pi/8)|1\rangle$ and the remaining gates encode this state into the code.

Suppose that we store the quantum codeword for a period of time and the qubits begin to decohere. It is not obvious, but the error operators that arise from independent decoherence processes on each qubit can be written as linear combinations of the identity operator and the Pauli operators X, $Y = -iZX$, and Z. Since quantum mechanics is a linear theory, it suffices to correct only the bit-flip X and phase-flip Z errors [Shor (1995) (http://journals.aps.org/pra/abstract/10.1103/PhysRevA.52.R2493)]. This is a remarkable observation. By correcting only a discrete set of errors, weighted sums of those errors – and hence a continuum of errors – can be corrected as well.

That said, the quantum bit-flip code is unable to correct any phase-flip errors that occur (hence its name). A phase flip on any qubit changes the encoded message to $\alpha|000\rangle - \beta|111\rangle$, but this state is an encoding of the 1 qubit message $\alpha|0\rangle - \beta|1\rangle$, which is a valid codeword too. Hence, the quantum bit-flip code is not "strong enough" to correct realistic errors since it is unable to correct phase errors. While we do not discuss them here, there are quantum codes that can detect and correct the most general types of errors, such as Shor's code [Shor (1995) (http://journals.aps.org/pra/abstract/10.1103/PhysRevA.52.R2493)].

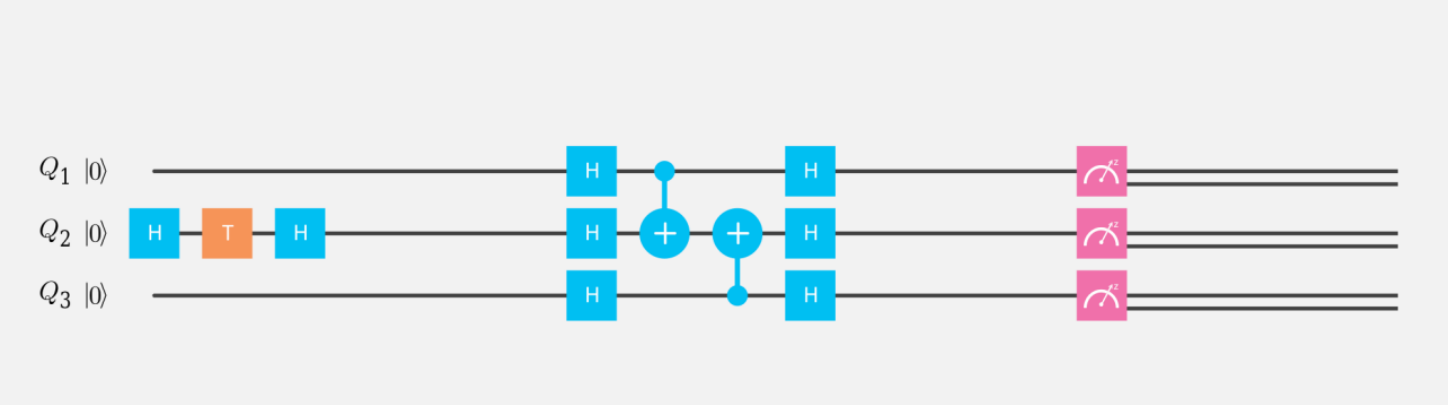
The quantum bit-flip code does correct bit-flip errors. We will briefly describe two procedures for detecting and correcting a bit-flip error using the 3-qubit code.

The second example below, "*Bit-flip encoder and decoder*," implements a reversible majority voter to decode the bit-flip code. The identity gates in the circuit represent noise, and you can replace them with various operations, such as bit flips X, to test the decoder. Following the identity gates, a pair of CNOT gates (with Hadamard gates) computes $abc \mapsto a \oplus b, b, c \oplus b$. The remaining T, CNOT, and H gates that target qubit 2 compute $\mathrm{MAJ}(a, b, c)$ into qubit 2, which we characterize using state tomography. Qubits 1 and 3 carry information about what errors occurred and remain unobserved. Even without explicitly inserting any errors, an experiment or realistic simulation will decode to a different point on the Bloch sphere because (a) the codeword is unprotected against phase errors and (b) the encoder and decoder are not ideal operations.

The third example below, "*Encoder into bit-flip code with parity checks*," implements parity measurements to detect errors. The first set of gates prepares the input state to the encoder, in this case $\cos(\pi/8)|0\rangle - i\sin(\pi/8)|1\rangle$. The second set of gates is the encoder followed by a SWAP, so that qubits 0, 1, and 3 contain the codeword. The third and final set of gates implements two parity computations: the parity of qubits 0 and 1 is computed into qubit 4, and the parity of qubits 1 and 3 is computed into qubit 2. Let's call the outcome of measuring qubit 4 by the name s_0 and the outcome of measuring qubit 2 by the name s_1. The pair of bits $s = s_0 s_1$ is called the *error syndrome*. If no error or just a single bit-flip error occurs, the error syndrome s reveals the location of the bit-flip error. Specifically, if $s = 00$ then no error occurred, if $s = 01$ then qubit 3 is

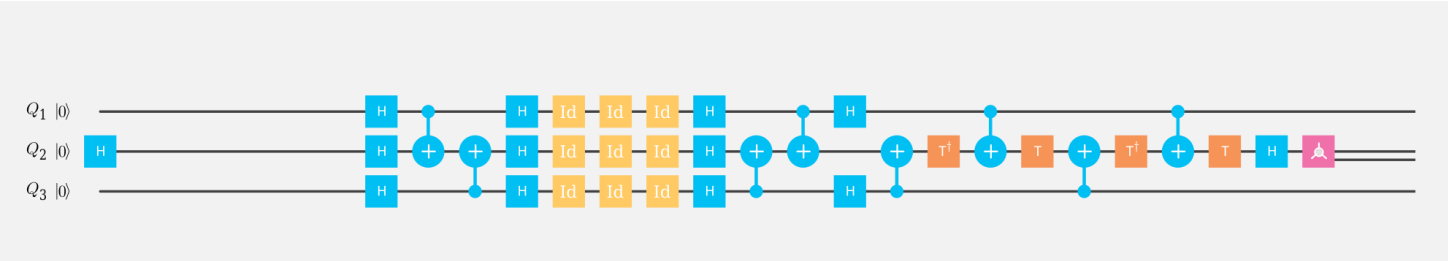
flipped, if $s = 10$ then qubit 0 is flipped, and if $s = 11$ then qubit 1 is flipped. Importantly, the act of measuring the error syndrome discretizes the error! After measuring quantum codeword in the standard basis, we obtain three outcome bits abc that are correlated with an error syndrome s and can use s to correct the outcome bits. For example, if we measure the outcome 001 (qubits 0, 1, 3) with error syndrome 01 (qubits 2 and 4) then we correct 001 to 000. Each three bit outcome abc is corrected to one of 000 or 111, and we expect to observe these with probability near $\cos^2(\pi/8)$ and $\sin^2(\pi/8)$, respectively, if the bit-flip error rate it not too high.

Encoder into bit-flip code (qubits 1-3)



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=74154f1a9b7afaa85e52795ab985e503&sharedCode=true)
Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=74154f1a9b7afaa85e52795ab985e503&sharedCode=true)

Bit-flip encoder and decoder (tomography)



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=c318faf262d3e567c71d1acfd981254f&sharedCode=true)
Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=c318faf262d3e567c71d1acfd981254f&sharedCode=true)

Encoder into bit-flip code with parity checks (qubits 0,1,3)

(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=b25b8a91a6f8355fff5a96b3a51078bd&sharedCode=true)
Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=b25b8a91a6f8355fff5a96b3a51078bd&sharedCode=true)

Stabilizer Measurements

Stabilizer codes (http://arxiv.org/abs/quant-ph/9705052) are a large and important family of quantum error-correcting codes. These codes are defined as joint eigenspaces of a collection of operators known as *stabilizers*. The eigenvalues of the stabilizers are used to detect and diagnose errors. Therefore, many quantum error-correction protocols make *stabilizer measurements* to obtain these eigenvalues. One way to implement a stabilizer measurement is to use extra qubits, known as *syndrome qubits*, to detect errors on *data qubits* that are part of your computation. Stabilizer measurements can identify a collective property of a set of data qubits; one such property is the qubits' *parity*.

Consider the quantum bit-flip code again. The bit-flip code is a stabilizer code, and one choice for its stabilizers is $Z \otimes Z \otimes I$ and $I \otimes Z \otimes Z$. A valid codeword is a $+1$ eigenstate of these stabilizer operators. For Z-type operators such as these, the stabilizer enforces that the parity of the first two qubits is even, and the parity of the second two qubits is even. The stabilizer measurement tells us whether the parity has changed, which we can use to diagnose the error.

In quantum codes such as the surface code (https://en.wikipedia.org/wiki/Toric_code), there are Z-type and X-type stabilizers to measure. The X-type stabilizers simply enforce a parity constraint in the $\{|+\rangle, |-\rangle\}$ basis. For example, the states $|++\rangle$ and $|--\rangle$ are $+1$ eigenstates of $X \otimes X$. The Z-type stabilizers in the bulk of the surface code have the form $Z \otimes Z \otimes Z \otimes Z$ and detect bit-flip errors on the four data qubits that are involved. The X-type stabilizers in the bulk have the form $X \otimes X \otimes X \otimes X$ and detect phase-flip errors. These stabilizer measurements are implemented by computing the associated bit-flip or phase-flip parities into a syndrome qubit.

In the Composer, we can test such stabilizer measurements using Q_2 as a syndrome qubit, and Q_0, Q_1, Q_3, and Q_4 as data qubits. The Z-type parity check is performed using a CNOT gate from each data qubit to the syndrome qubit. The X-type parity check is simply the conjugate of the Z-type check, obtained by applying Hadamard gates to the input and output data qubits. In a fault-tolerant implementation of the circuits, the order of gates is important to limit the spread of errors.

The scores below prepare different states of input parity. See if the processor returns the proper parity measurements.

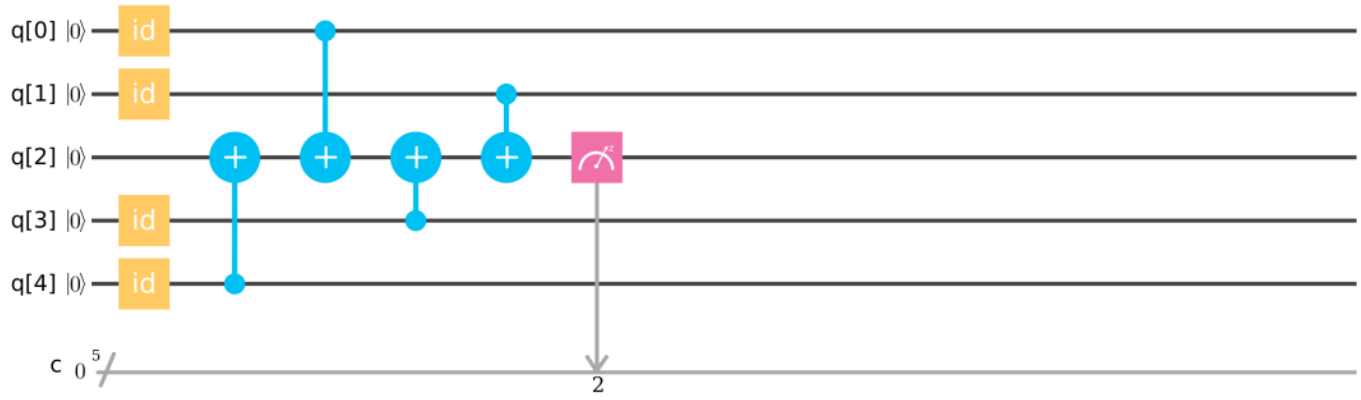
When we ran these experiments we got

Plaquette test: May 3rd 11:20 pm

$\|Q_4Q_0Q_3Q_1\rangle$	P_C	SE
$\|0\,0\,0\,0\rangle$	0.878	0.0036
$\|0\,0\,0\,1\rangle$	0.809	0.0043
$\|0\,0\,1\,0\rangle$	0.880	0.0036
$\|0\,0\,1\,1\rangle$	0.908	0.0032
$\|0\,1\,0\,0\rangle$	0.812	0.0043
$\|0\,1\,0\,1\rangle$	0.815	0.0043
$\|0\,1\,1\,0\rangle$	0.914	0.0031
$\|0\,1\,1\,1\rangle$	0.900	0.0033
$\|1\,0\,0\,0\rangle$	0.872	0.0037
$\|1\,0\,0\,1\rangle$	0.878	0.0036
$\|1\,0\,1\,0\rangle$	0.936	0.0027
$\|1\,0\,1\,1\rangle$	0.893	0.0034
$\|1\,1\,0\,0\rangle$	0.887	0.0035
$\|1\,1\,0\,1\rangle$	0.771	0.0046
$\|1\,1\,1\,0\rangle$	0.875	0.0037
$\|1\,1\,1\,1\rangle$	0.922	0.0030

$\|Q_4Q_0Q_3Q_1\rangle$	P_C	SE
$\|+\,+\,+\,+\rangle$	0.847	0.0040
$\|+\,+\,+\,-\rangle$	0.798	0.0044
$\|+\,+\,-\,+\rangle$	0.883	0.0036
$\|+\,+\,-\,-\rangle$	0.908	0.0032
$\|+\,-\,+\,+\rangle$	0.822	0.0042
$\|+\,-\,+\,-\rangle$	0.811	0.0043
$\|+\,-\,-\,+\rangle$	0.922	0.0030
$\|+\,-\,-\,-\rangle$	0.871	0.0037
$\|-\,+\,+\,+\rangle$	0.858	0.0039
$\|-\,+\,+\,-\rangle$	0.883	0.0036
$\|-\,+\,-\,+\rangle$	0.898	0.0033
$\|-\,+\,-\,-\rangle$	0.873	0.0037
$\|-\,-\,+\,+\rangle$	0.918	0.0030
$\|-\,-\,+\,-\rangle$	0.850	0.0039
$\|-\,-\,-\,+\rangle$	0.806	0.0044
$\|-\,-\,-\,-\rangle$	0.860	0.0038

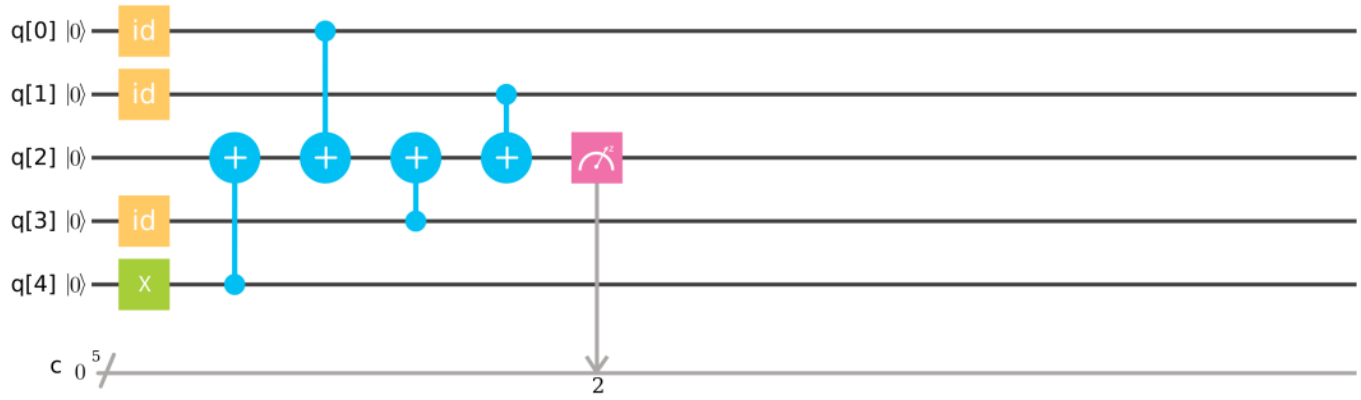
Plaquette Z 0000



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=618bfcf17927bad8e5a5f8d37568772d&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=618bfcf17927bad8e5a5f8d37568772d&sharedCode=true)

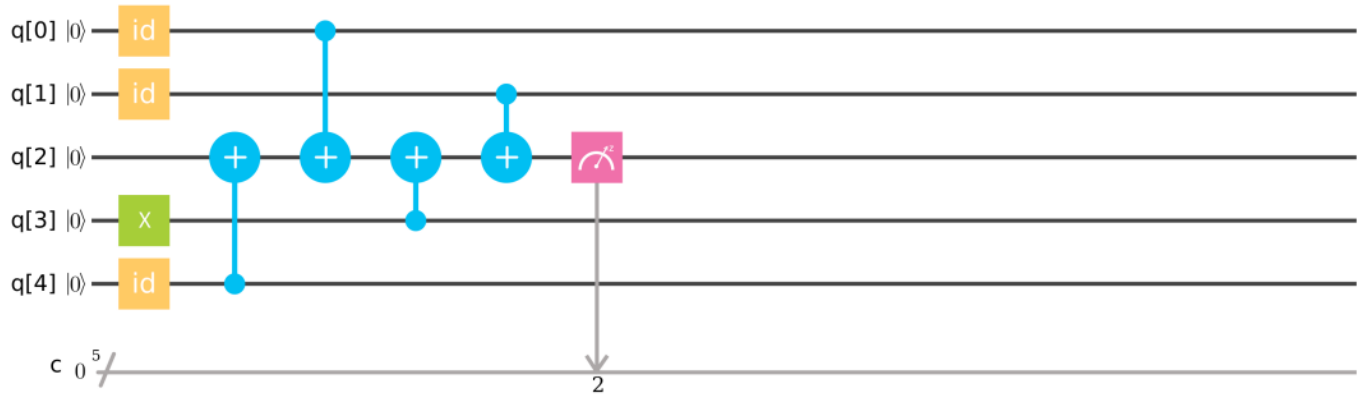
Plaquette Z 0001



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=5cba20db95acffdf97bf95af4f253de8&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=5cba20db95acffdf97bf95af4f253de8&sharedCode=true)

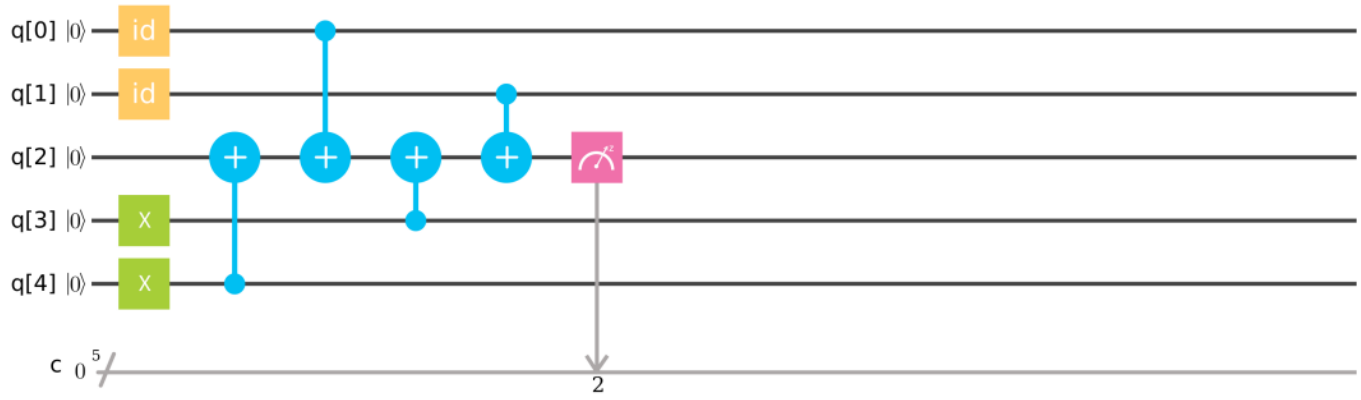
Plaquette Z 0010



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=e356856b01d8bc9f8db0d3edced40ec1&sharedCode=true)

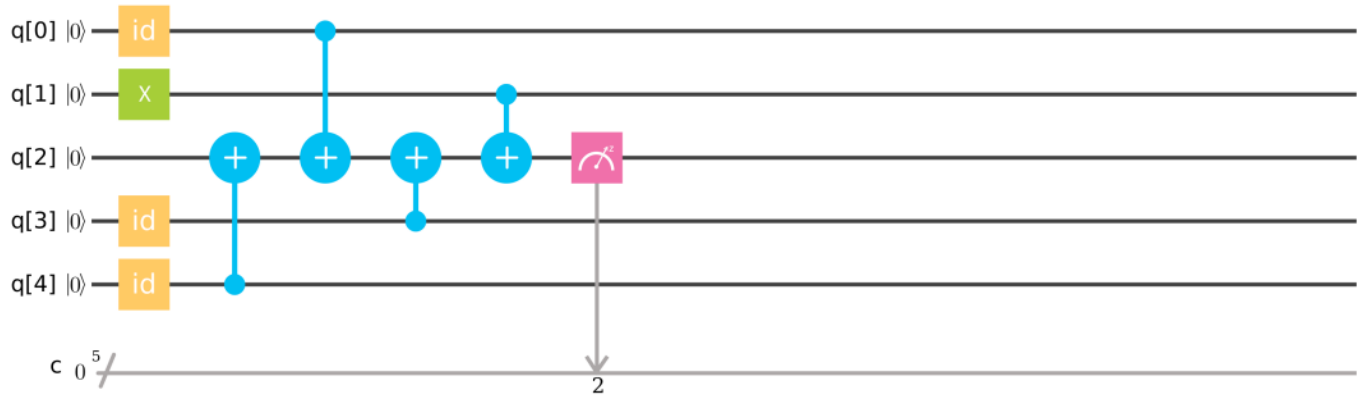
Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=e356856b01d8bc9f8db0d3edced40ec1&sharedCode=true)

Plaquette Z 0011



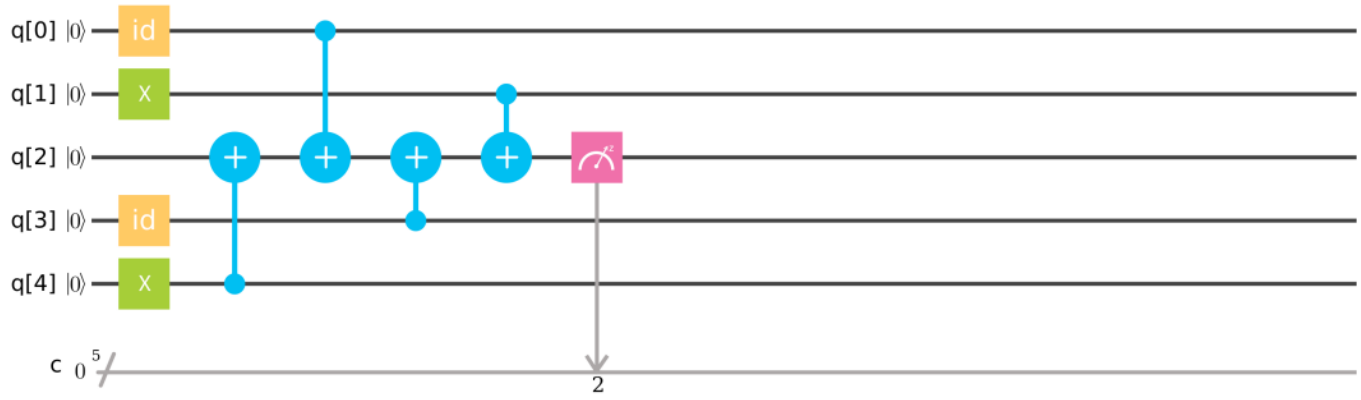
(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=35cfa432f306d500aa85941a932318bf&sharedCode=true)
Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=35cfa432f306d500aa85941a932318bf&sharedCode=true)

Plaquette Z 0100



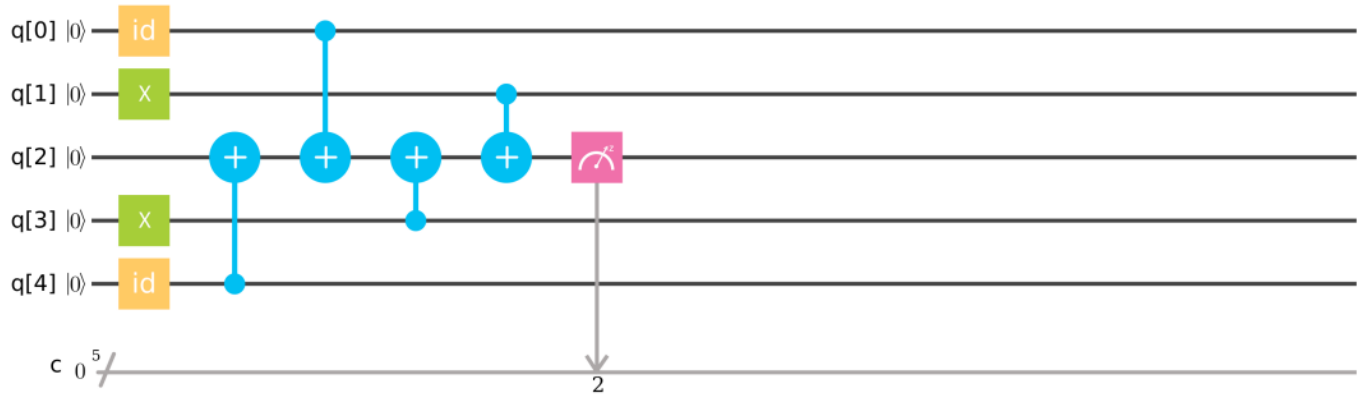
(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=35cfa432f306d500aa85941a93421b6f&sharedCode=true)
Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=35cfa432f306d500aa85941a93421b6f&sharedCode=true)

Plaquette Z 0101



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=9d8a5f01839f380c33855ba11714ad18&sharedCode=true)
Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=9d8a5f01839f380c33855ba11714ad18&sharedCode=true)

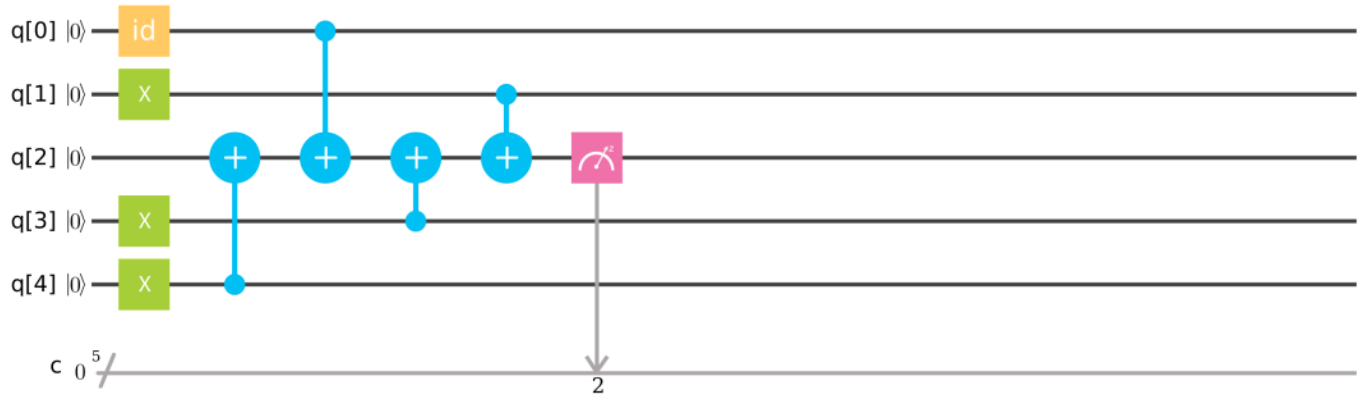
Plaquette Z 0110



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=618bfcf17927bad8e5a5f8d375b37f29&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=618bfcf17927bad8e5a5f8d375b37f29&sharedCode=true)

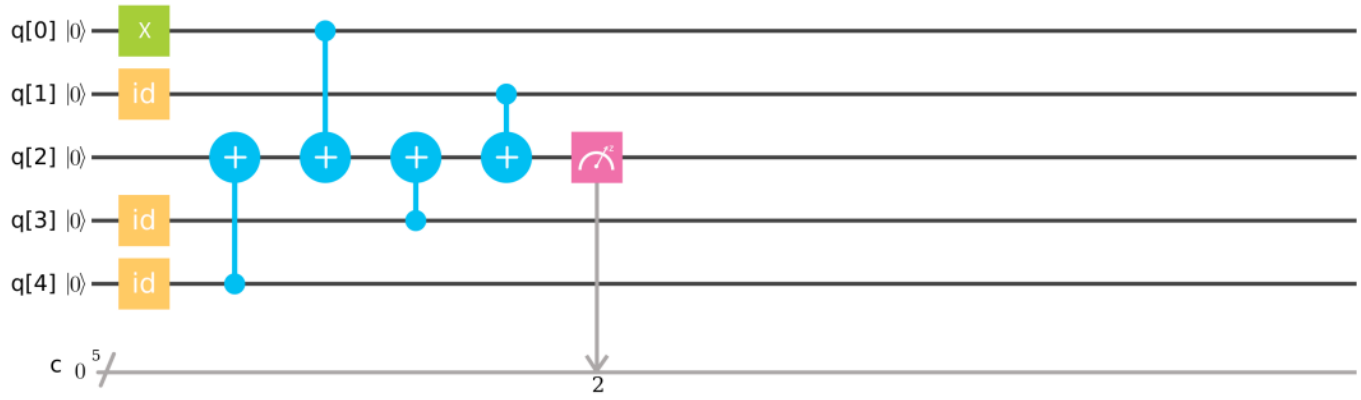
Plaquette Z 0111



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=c7177d58a05920ca438c872f1ce253b7&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=c7177d58a05920ca438c872f1ce253b7&sharedCode=true)

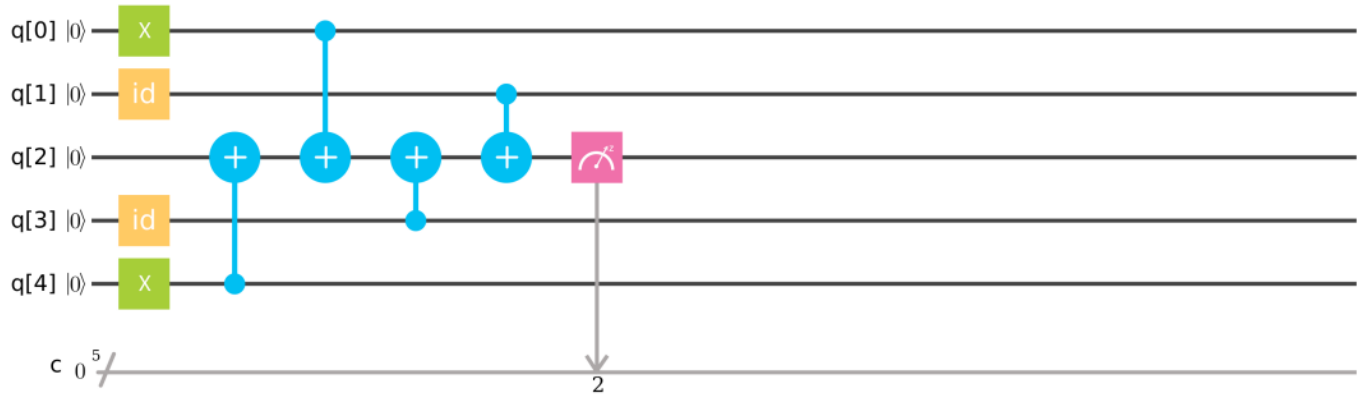
Plaquette Z 1000



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=deb1f2aa1f7c5718bfdb4029eb67c200&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=deb1f2aa1f7c5718bfdb4029eb67c200&sharedCode=true)

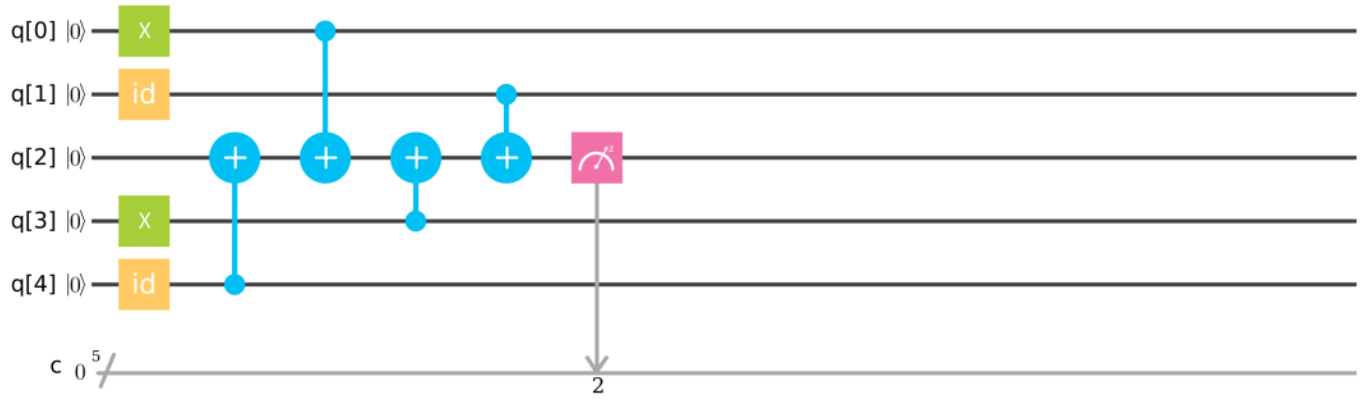
Plaquette Z 1001



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=618bfcf17927bad8e5a5f8d375c82131&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=618bfcf17927bad8e5a5f8d375c82131&sharedCode=true)

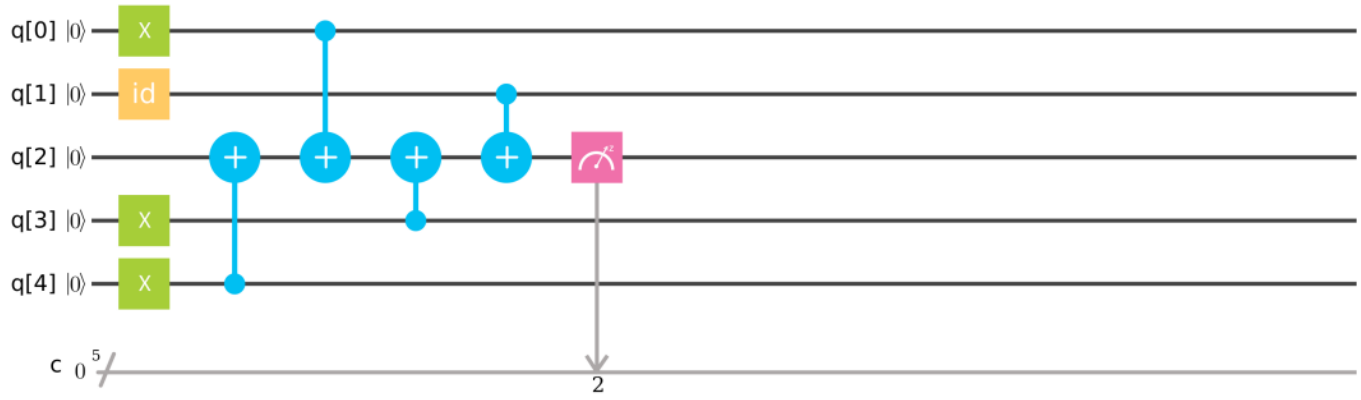
Plaquette Z 1010



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=c495ddc03a0d87c9c4c3a9b31a145813&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=c495ddc03a0d87c9c4c3a9b31a145813&sharedCode=true)

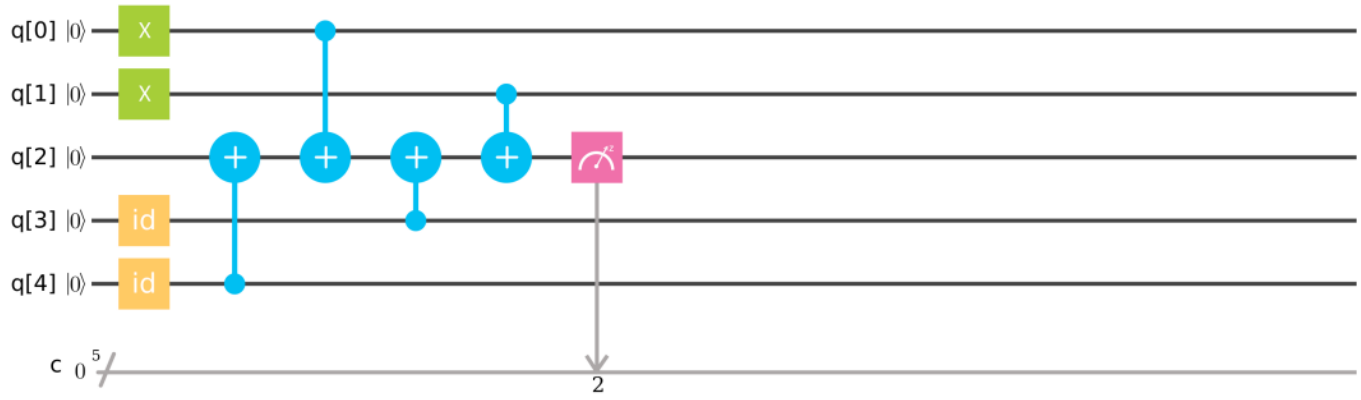
Plaquette Z 1011



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=7ce6f28163171c882dc7228f080ca3a1&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=7ce6f28163171c882dc7228f080ca3a1&sharedCode=true)

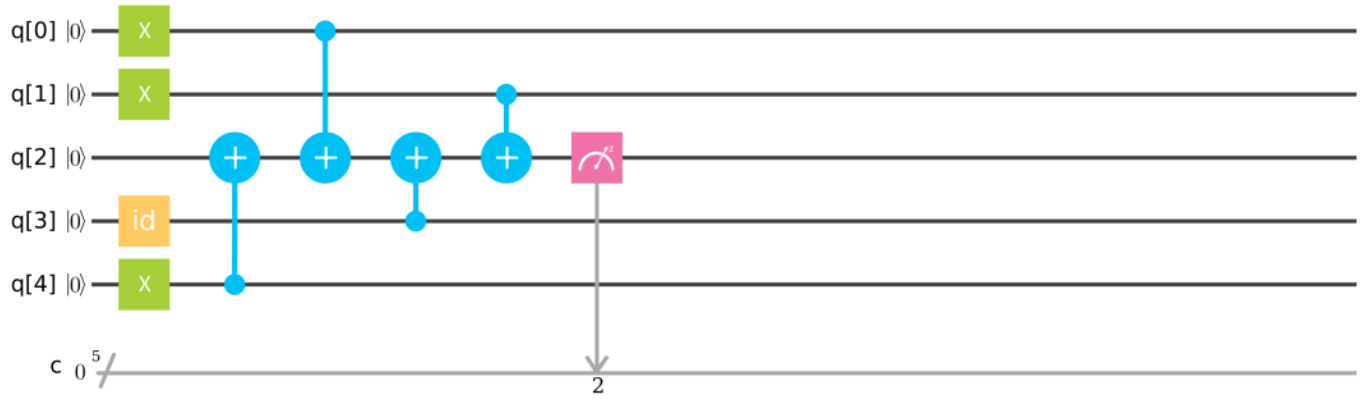
Plaquette Z 1100



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=9d8a5f01839f380c33855ba11789cc51&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=9d8a5f01839f380c33855ba11789cc51&sharedCode=true)

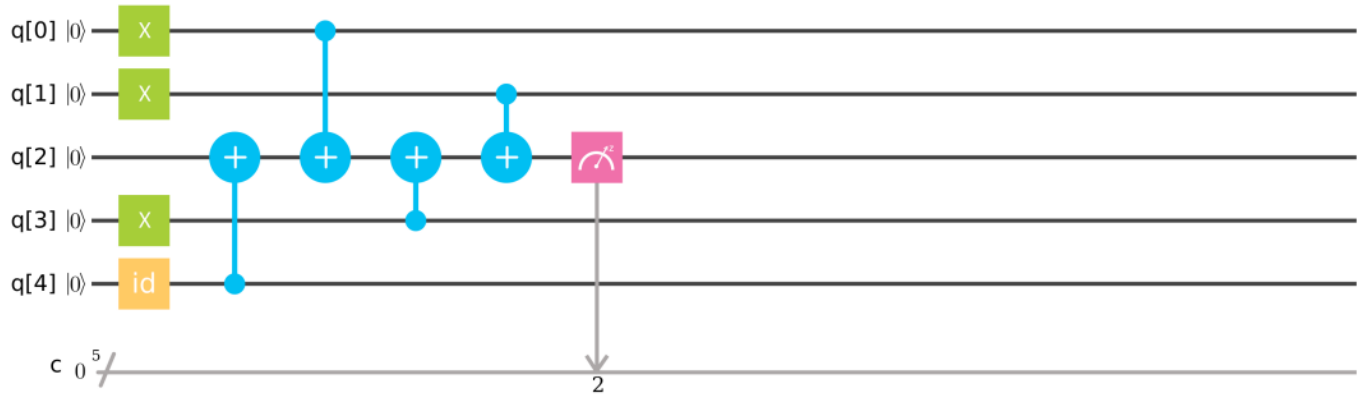
Plaquette Z 1101



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=5cba20db95acffdf97bf95af4fc5c23c&sharedCode=true)

Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=5cba20db95acffdf97bf95af4fc5c23c&sharedCode=true)

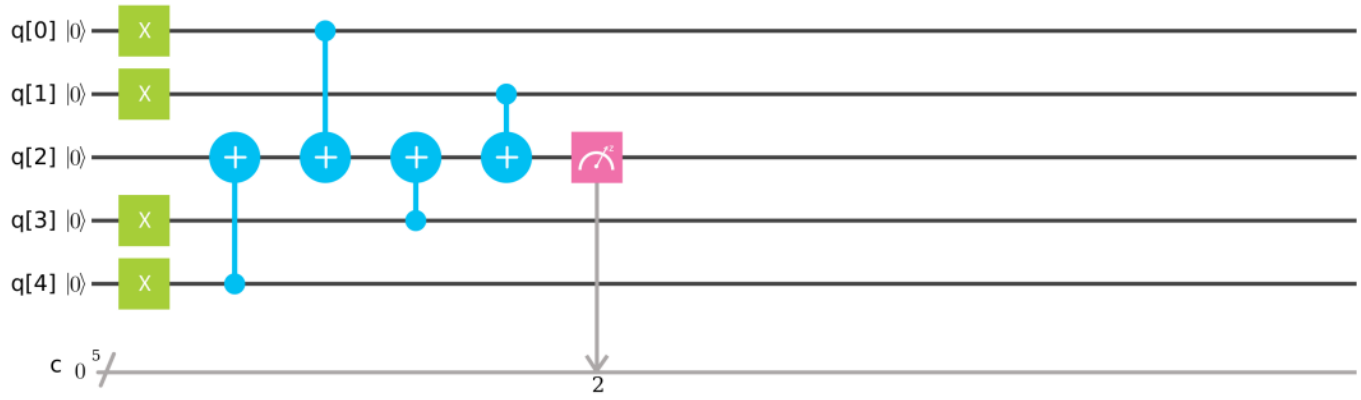
Plaquette Z 1110



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=b2d815e741f0b820bd5b93a583b3bf93&sharedCode=true)

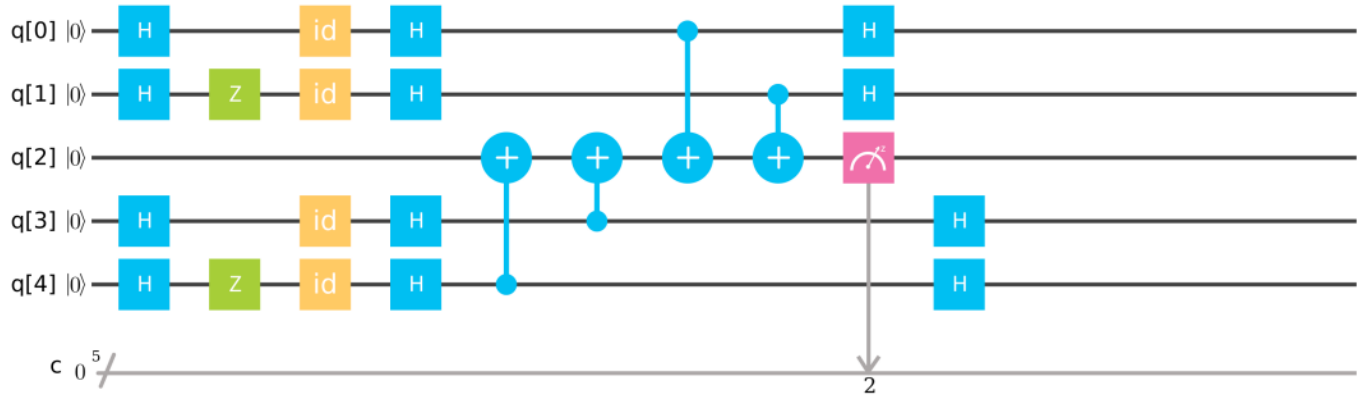
Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=b2d815e741f0b820bd5b93a583b3bf93&sharedCode=true)

Plaquette Z 1111



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=84becc47be891676e7ddc04e8760069c&sharedCode=true)
Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=84becc47be891676e7ddc04e8760069c&sharedCode=true)

Plaquette X +-+-



(https://quantumexperience.ng.bluemix.net/qx/editor?codeId=b5a0e7376ded40cd7dc1022e777c3e7a&sharedCode=true)
Open in composer (https://quantumexperience.ng.bluemix.net/qx/editor?codeId=b5a0e7376ded40cd7dc1022e777c3e7a&sharedCode=true)
Here we will walk through basic examples of quantum error correction and the measurements necessary for more sophisticated quantum codes.

- Quantum Repetition Code
- Stabilizer Measurements

Contributing

We appreciate all kinds of help, so thank you!

You can contribute in many ways to this project.

Issue reporting

This is a good point to start, when you find a problem please add it to the issue traker (https://github.com/QISKit/qiskit-qx-user-guides/issues). The ideal report should include the steps to reproduce it.

Doubts solving

To help less advanced users is another wonderful way to start. You can help us to close some opened issues. This kind of tickets should be labeled as `question` .

Improvement proposal

If you have an idea for a new feature please open a ticket labeled as `enhancement` . If you could also add a piece of code with the idea or a partial implementation it would be awesome.

Documentation

Review the parts of the documentation regarding the new changes and update it if it's needed.

Pull requests

We use GitHub pull requests (https://help.github.com/articles/about-pull-requests) to accept the contributions. You can explain yourself as much as you want. Please follow the next rules for the commit messages:

- It should be formed by a one-line subject, followed by one line of white space. Followed by one or more descriptive paragraphs, each separated by one line of white space. All of them finished by a dot.
- If it fixes an issue, it should include a reference to the issue ID in the first line of the commit.
- It should provide enough information for a reviewer to understand the changes and their relation to the rest of the code.
