Introduction to Quantum Computing

CSCI 490: Independent Research Project

Austin Shouli
Vancouver Island University

Spring 2024

Prerequisite Math

Complex Numbers

Imaginary Numbers

Complex numbers are an important concept in quantum computing. In order to understand complex numbers, we will first examine *imaginary numbers.* Consider the following expression

$$\sqrt{-1}$$

It is not clear that there is an evaluation of this expression. It cannot be 1, and it cannot be negative 1. However, there is a mathematical way of interpreting this expression

$$\sqrt{-1} = i$$

This number *i* is called an imaginary number. Technically, an imaginary number is a number in the form: $b \cdot i$, where $b \in \mathbb{R}$. In the case of the imaginary number i, the real number component *b* is simply 1, so that $1 \cdot i = i$. Note that

$$i^2 = \sqrt{-1}^2 = -1$$

We can also consider imaginary numbers with a real component $\neq 1$, such as $5i$

$$5i^2 = 25\sqrt{-1} = \sqrt{-25}$$

Complex Numbers

Similar to how imaginary numbers are made up of the product of an *imaginary component* and a *real component*, complex numbers are made up of the sum of an imaginary number and a real number. We can represent the set of complex numbers with the notation $\mathbb{C}$, so that a complex number $c \in \mathbb{C}$.

For any complex number c, $c = a + bi$ where a is the real part, and bi is the imaginary part. Note that the real part of a complex number can be zero, such that the complex number is $0 + bi$ or simply bi. Similarly, the imaginary part of a complex number can be zero, such that the complex number is simply some real number *a*.

This implies that the real numbers $\mathbb{R}$ are a subset of the complex numbers, such that $\mathbb{R} \in \mathbb{C}$. Thus, any real number is also a complex number, and any imaginary number is also a complex number.

Linear Algebra

Linear algebra plays a fundamental role in quantum computing, and is a common form of mathematically representing quantum computing states and operations. Vectors are often used to represent the state of classical and quantum bits (qubits), and operations performed on a bit or system are represented by specific matrices. Multiplying a column vector by a matrix equates to performing an operation on a bit or qubit.

Classical States

In classical computing, a bit can be either in state 0 or state 1. Using column vectors to represent a bit, the state of a bit is indicated by the position of the 1 in the vector. The top position in the column vector indicates the bit is in state 0, and the bottom position indicates the bit is in state 1. In the below example, the bit represented by the column vector is in state 0, because there is a 1 in the top position, and a 0 in the bottom position.

$$\begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Note that the bit cannot both be in state 0 and state 1, so the column vector containing 1's in both positions would be invalid. To be exact, the top position represents the probability of the bit being in a 0 state when measured, and the bottom position represents the probability of the bit being in a 1 state when measured. This distinction allows us to represent a probabilistic state, where we don't know for certain if the bit is in state 0 or state 1. For example, consider the scenario where there is a bit, and there is a 75% chance of it being in the 0 state. Since the bit can only be in the 0 state or the 1 state, there must also be a 25% chance the bit is in the 1 state. We can represent the state of this bit as follows

$$\begin{bmatrix} \frac{3}{4} \\ \frac{1}{4} \end{bmatrix}$$

We call this form of representing a bit's state a *state vector.* To be considered a valid classical state vector, note that

1. The sum of the vector's entries must equal 1
2. The entries of the vector must be non-negative, real numbers

If we consider the state vector as representing the probability of a bit being in a particular state when measured, these two conditions make logical sense. If there is a 75% percent change of the bit being in 0-state when measured, there must conversely be a 25% percent chance of it being in the 1-state.

We can also use a column vector to represent a system containing multiple bits. Consider a system consisting of two bits, c_0 and c_1. We can represent the state of the system using a bitstring, so that the bit in the first position represents the state of bit c_0 and the bit in the second position represents the state of c_1:

Bits:	c_0	c_1
State:	0	1
Bitstring:	01	

This bitstring '01' represents a system where the bit c_0 is in state 0, and bit c_1 is in state 1.

Similar to a single bit system, we can use a column vector to represent the state of a system made up of multiple bits. In the example below, the column vector or state vector represents the state of a system consisting of two bits, with the system in the state 01 (one bit is in state 0, and the other is in state 1)

$$\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \; states = \begin{cases} 00 \\ 01 \\ 10 \\ 11 \end{cases}$$

Quantum States

Until now, we have observed how vectors can be used to represent the state of a classical bit, or the state of a system consisting of multiple classical bits. We can use similar notation to represent the state of a quantum bit (or *qubit*), with two key differences.

In the classical state vectors above, we observed that the entries in the vector must be non-negative real numbers, and that the sum of the enteries must equal one. In a *quantum* state vector, the entries in the vector must be *complex* numbers, and the sum of each entry's absolute value *squared* must be equal to one. Recall that the real numbers are a subset of the complex numbers, so the entries of a quantum state vector can be real numbers - or technically complex numbers, with the complex component equal to zero.

Accordingly, the classical state vectors below are also valid quantum state vectors:

$$\begin{bmatrix}1\\0\end{bmatrix}, \begin{bmatrix}0\\1\end{bmatrix}$$

However, note that the classical state vector observed previously, indicating that there is a 75% chance of the bit being in the 0 state *when measured*, is *not* a quantum state vector. This is because the sum or the absolute value squared, of each entry, does not sum to one.

$$\begin{aligned}\begin{bmatrix}\frac{3}{4}\\\frac{1}{4}\end{bmatrix} &\sim \left|\left(\frac{3}{4}\right)\right|^2 + \left|\left(\frac{1}{4}\right)\right|^2\\ &= \frac{9}{16} + \frac{1}{16}\\ &= \frac{10}{16}\\ &= \frac{5}{8}\end{aligned}$$

However, observe that the following vector is a valid quantum state vector, but would not be a valid classical state vector, despite only having complex components equal to zero:

$$\begin{aligned}\begin{bmatrix}\frac{1}{\sqrt{2}}\\\frac{1}{\sqrt{2}}\end{bmatrix} &\sim \left|\left(\frac{1}{\sqrt{2}}\right)\right|^2 + \left|\left(\frac{1}{\sqrt{2}}\right)\right|^2\\ &= \frac{1}{2} + \frac{1}{2}\\ &= \frac{2}{2}\\ &= 1\end{aligned}$$

Recall that the entries in a classical state vector indicated the probability of a system being in each particular state. This is intuitive for a classical bit - if there is a 75% probability of a bit being in the 0-state when measured, conversely, there must be a 25% percent probability of the bit being in the 1-state when measured. However, this intuitive view of probability no longer applies for quantum bits. The mathematics above imply a different sort of probability space for a quantum bit. We will explore this concept in further sections.

Matrix Operations

In order to interact with a bit, we can perform operations on it - such as a bit-flip operation on a classical bit. While a bit-flip can be performed intuitively, there are more complex operations that cannot be. Luckily, any operation we can perform on a bit can can be represented using matrices, and an operation can be performed on a bit by multiplying a matrix by a bit's state vector.

For example, we can represent the classical bit-flip operation (also called a NOT operation), on a single bit, with the matrix

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

We can see that multiplying this matrix by a bit in the 0-state results in a bit in the 1-state

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

Similarly, multiplying this matrix by a bit in the 1-state results in a bit in the 0-state

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

This type of operation is called deterministic and is represented by a matrix with exactly one 1 in each column, and 0 for all other entries. We will take a look at quantum operations and their corresponding matrices in later sections.

Dirac Notation

Dirac notation, also known as bra-ket notation, is a commonly used method to represent state vectors in a concise form. A 'bra' consists of an angle-bracket ont the left, and a straight bracket on the right. In the below example, the bra notation is read as "bra x." Similarly, a ket consists of a straight bracket on the left, and a angle-bracket on the right. The example ket below is read as "ket x."

The naming convention is based on a pun, where a bra and a ket together form a bracket. When a bra and ket are written together this way, we can use a shorthand and eliminate one of the internal straight brackets.

$$bra : \langle x|$$
$$ket : |x\rangle$$

$$bra - ket : \langle x|\,|x\rangle = \langle x\,|\,x\rangle$$

Ket Notation

In quantum computing, this notation is frequently used as a shorthand to describe the state of the system. For example, consider a single bit in the 0 state. We can represent the state of this system using the ket notation, where the value inside the brackets indicates the state the system is currently in

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

Similarly, we can represent a bit in the 1 state using the “ket one” notation:

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

Notably, we can represent a system in a non-deterministic state using a linear combination of the state vectors

$$\begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 0 \end{bmatrix} + \frac{1}{\sqrt{2}}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

The reason for this shorthand may not be clear for a system consisting of a single bit, but consider a system consisting of 3 bits. Representing this system as a column vector would require a 2^3 x 1 column vector (2^3 rows by 1 column). However, we can represent the state of the system as such

$$|010\rangle = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \; states = \begin{matrix} 000 \\ 001 \\ 010 \\ 011 \\ 100 \\ 101 \\ 110 \\ 111 \end{matrix}$$

Here we can see the the $|010\rangle$ notation indicates a system of 3 bits, say b_0, b_1, b_2, where b_0 is in the 0 state, b_1 is in the 1 state, and b_2 is in the 0 state. The dirac notation is a much more concise and human readable way of representing the state of the system.

Bra Notation

Similar to how the *ket* notation is used to represent a column vector, the *bra* notation is used to represent a row vector. While this notation isn't normally used to represent the state of a bit or system, it is equivalent to it. The probability of the system being in a particular state is indicated by the the position of the values in the vector, as shown below.

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

There are mathematical implications for the use of *bra* notation, as we will see later, but both forms can represent the state of a bit or qubit. The row vector is equivalent to the transpose of the column vector, and vice versa.

$$\begin{bmatrix} 1 \\ 0 \end{bmatrix}^T = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

Bra-ket Operations

Inner Product

Since *bras* and *kets* represent vectors, we can perform operations on them in the same way we could apply operations to any matrix. When a bra appears next to a ket, as shown below, this implies a matrix multiplication, or *inner product,* operation. Recall that the second ‘ | ’ can be dropped, simplifying the expression as shown

$$\langle 0||1\rangle \;=\; \langle 0\,|\,1\rangle \;=\; \begin{bmatrix}1 & 0\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix} \;=\; [0] \;=\; 0$$

Since the result of the inner product operation is 0, this indicates that the states $|0\rangle$ and $|1\rangle$ are *orthogonal* to each other. This is equivalent to the concept of at a right-angle to each other, but can be applied across many dimensions. Furthermore, since we can express any possible state in the system as a linear combination of these two states, these states form an *orthonormal basis* for the system, and we can call them *basis states.* In other words, any possible state in the system can be expressed as a linear combination of the $|0\rangle$ state and the $|1\rangle$ state.

Furthermore, observe that

$$\langle 0\,|\,0\rangle \;=\; \langle 1\,|\,1\rangle \;=\; [1] \;=\; 1$$

The resulting 1 indicates that the state vectors $|0\rangle$ and $|1\rangle$ are *normalized*, or *unit vectors*. Since $|0\rangle$ and $|1\rangle$ are orthogonal to each other, and each is normalized, we can say that these vectors are *orthonormal.* This means that they are not only orthogonal to each other, but the ‘length’ of the both state vectors is normalized to 1.

Born’s Rule

Recall from the inner product section above that any possible state vector in the system can be expressed as a linear combination of orthonormal states, or states which form a basis for the system. For a quantum system with basis vectors $|0\rangle$ and $|1\rangle$, this means that for any state in the system, say $|\psi\rangle$, the state $|\psi\rangle$ can be expressed as a linear combination of the basis states, such that:

$$|\psi\rangle \;=\; \alpha|0\rangle \;+\; \beta|1\rangle$$

$$where\ \alpha,\ \beta \,\in\, \mathbb{C}\,,\ and\ |\alpha|^2 \,+\, |\beta|^2 \,=\, 1$$

Note that in this context, α, β are called *amplitudes.* This linear combination of states is what is known as *superposition.* Born's rule states that for a superposition of states, the absolute-value squared of a state is the probability of the system being in that state when measured.

In the above example, this means that the probability if being in the $|0\rangle$ state when measured is $|\alpha|^2$. Furthermore, the sum of the probability for all of the states must equal 1, such that

$$|\alpha|^{\,2} + |\beta|^2 = 1$$

This form of equation may look familiar, and is in fact related to the *Euclidean Norm* of a vector, in which the length of a vector is scaled so that the 'length' of the vector is equal to to 1. This is also called *normalization.* It is also important to note that while the *amplitudes* α, β are complex numbers, the *probabilities* generated by them are not. This can be observed mathematically by the squaring of the complex number amplitudes, in which the imaginary part of the complex number becomes a real number, producing the probability of a qubit being in a particular state.

Outer Product

When a *bra* and *ket* appear in the *ket-bra* order, this represents the *outer product* operation, which is performed as shown

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

Tensor Product

Finally, what happens when we see a *bra-bra* or *ket-ket* notation? According to the rules of matrix multiplication, we cannot multiply these matrices together. The matrix on the left must have the same number of columns as the matrix on the right has rows. This is clearly not the case. However, this notation still has meaning, and is used as a shorthand for the *tensor product* operation.

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

To understand how the tensor product operation is calculated, observe the example below using 2 x 2 matrices *A* and *B*

$$A \otimes B = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \otimes \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} =$$

$$= \begin{bmatrix} a_{11}\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} & a_{12}\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \\ a_{21}\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} & a_{22}\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} & a_{11}b_{12} & a_{12}b_{11} & a_{12}b_{12} \\ a_{11}b_{21} & a_{11}b_{22} & a_{12}b_{21} & a_{12}b_{22} \\ a_{21}b_{11} & a_{21}b_{12} & a_{22}b_{11} & a_{22}b_{12} \\ a_{21}b_{21} & a_{21}b_{22} & a_{22}b_{21} & a_{22}b_{22} \end{bmatrix}$$

Observe that the *tensor product* operation above can be thought of as taking two individual bits and combining them in to one system consisting of two bits, or one register composed of the two original bits.

In the example below, the tensor product takes two individual qubits, both in the $|0\rangle$ state, and combines them into a register containing 2 qubits, in the $|00\rangle$ state. Note that the state of the system, viewed as the state of the individual bits, hasn't changed. Notice how the the tensor product of two individual qubit systems is equivalent to a two qubit system, with the qubits respectively in the same states.

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

Bloch Sphere

We now known the state of a qubit can be represented by a quantum state vector, and any possible vector in the system can be represented as a superposition of two orthogonal vectors in the system. Furthermore, according to Born's Rule, these vectors must be normalized to length one, so the superposition must consist of two orthonormal vectors. This means that any possible state the system can be in can be represented by a vector of length 1.

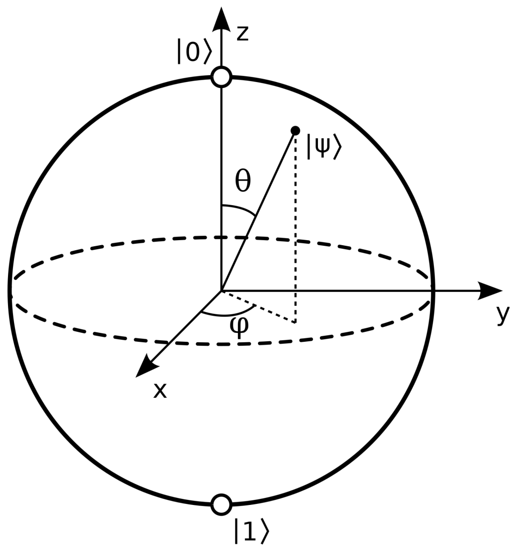
Due to the normalization of the state vector, the complex amplitudes α, β can actually be represented trigonometrically using real angles

$$\alpha = \cos\left(\frac{\theta}{2}\right), \ \beta = e^{i\phi}\sin\left(\frac{\theta}{2}\right), \text{ such that}$$

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle = e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle$$

Since we are now representing the state vector in terms of real values θ and ϕ, we can represent the vector in *three* dimensional space.

While ordinarily vectors are represented in cartesian coordinate systems, we can also represent a vector using a *spherical* coordinate system. Accordingly, every possible state vector for a given qubit can be contained within a *unit sphere*. We use what is called a *Bloch Sphere* to represent a state vector, also called a *Bloch vector,* in three-dimensional space



In a Bloch sphere, $|0\rangle$ is typically taken as the 'north pole' of the sphere, and $|1\rangle$ is taken as the 'south pole.' (Interestingly, while these vectors are orthogonal to each other, their *bloch vectors* do not visually appear orthogonal in a Bloch sphere, but appear inline with, or opposite to, each other).

Note that the Bloch sphere is a unit sphere, so that it's radius $r = 1$. Therefore, any normalized vector should terminate on the surface of the sphere. If a vector does not terminate on the surface of the sphere, it cannot be considered a valid quantum state vector.

To represent a vector in the Bloch sphere, we utilize the two angles, θ and ϕ, such that θ is called the *polar angle* and ϕ is called the *azimuthal* angle. Since the length of any possible vector is already known to be 1, we can represent any possible state vector using a combination of the two angles. The polar angle θindicates the angle of rotation *from* the vertical axis (the axis orientated with $|0\rangle$, and the azimuthal angle ϕrepresenting the rotation *around* the vertical axis.

While $|0\rangle$ is ordinarily taken to be the north pole of the sphere, this is an arbitrary visual choice. Mathematically, $|0\rangle$ is defined as the position where the polar angle θ is 0. Notably, when $\theta = 0$ the azimuthal angle has no impact on the resulting vector.

$$|\psi\rangle = \cos\left(\frac{0}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{0}{2}\right)|1\rangle = 1|0\rangle + 0|1\rangle = |0\rangle$$

Similarly, $|1\rangle$ state is defined mathematically as the state where the polar angle $\theta = \pi$, so that

$$|\psi\rangle = \cos\left(\frac{\pi}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\pi}{2}\right)|1\rangle = 0|0\rangle + 1|1\rangle = |1\rangle$$

Accordingly, the latitude of a vector in the Bloch sphere, determined by θ, represents the amount of contribution to that vector from the $|0\rangle$ and $|1\rangle$ states. The longitude, represented by ϕ, indicates the *phase* of the amplitude β relative to the amplitude α (more on phase later).

By using a Bloch sphere to represent a qubit, or multiple qubits (multiple vectors within the sphere), we can visually interpret information about a quantum state, and even visualize the evolution of the state of a quantum system over time.

Quantum Computing Hardware

Current quantum computers are not stand-alone devices, they in fact require a classical computer(s) to control the operations performed on qubits. The measurement data from the quantum computer is output as classical information, which is then processed by a classical computer. Since quantum computers only provide an advantage over classical architecture in certain problem sets, quantum computers, for the foreseeable future, will be part of a larger classical infrastructure.

In this sense, quantum computers can be thought of as a quantum processing unit (QPU). Similar to a GPU, the QPU is a specialized component in a larger machine, that can be called upon for specific tasks. While a useful framework for thinking about quantum computers role in a computation, this does not mean that PCIe QPUs will soon be on the consumer market, due to current limitations such as size and cooling as we will see.

In practice, today's quantum computers are typically operated like the time-shared mainframe computers, or cloud computing systems. For example, IBM currently operates several quantum computers, located at IBM facilities. Some of these computers are publicly available, with users provided a certain amount of minutes per month based on their subscription. Programs sent to run on a hardware system enter into a queue, based on the current demand for that processor. Due to the supercooling requirements of current quantum computer designs, the near-future of quantum computing will likely use a cloud-service model.

Types of Artificial Qubits

Nuclear Magnetic Resonance (NMR)

One of the oldest types of qubit used for computation, Nuclear Magnetic Resonance works by using the (nucleus) atoms of a molecule as qubits, utilizing their spin property. It is important to note that the term spin is used in this case to describe a property of elementary particles, such as electrons, called angular momentum. This is different from the concept we commonly associated with the term spin, which is called orbital angular momentum, and refers to the motion of a quantum particle in space. Angular spin, however, is an unalienable property of a particle, and even if the particle is at rest, it still has a spin value. Furthermore, particles can have either integer or half-integer spin, and 'substance' particles such as electrons and neutrons have half-integer spin.

In NMR quantum computing, many identical molecules, each containing several atoms with a nuclear spin value of ½ are used. The spin of the atoms interacts with each other, but it can be difficult to control and measure the spin as the number of bits increases. The result is the technology is not scalable for practical computations, although this method was notably used to implement Shor's factoring algorithm in 2001, factoring the number 15, such that 15 = 3 x 5.

Superconducting Qubits

Superconducting qubits are currently the most widely used type of qubit, and are used by IBM and Google, among others. These devices rely on a property of certain metals, called superconductivity. When a metal, or alloy, is cooled to near absolute zero, the electrical resistance of the material becomes negligible. This is due to a phase change in the metal.

A device called a Josephson Junction is built from a piece of non-superconducting material, between two pieces of superconducting material. These devices are used to construct qubits by forming a loop of superconducting material containing one or more josephson junctions, a capacitor, and potentially other electronic devices such as inductors. In addition, control and measurement circuity is also connected to the circuit, and microwave pulses are used to apply operations to the qubit. Notably, this type of qubit is made up of many particles, which act as a single quantum system. A common type of superconducting qubit is the superconducting transmon qubit, used in particular by IBM. Other notable types of superconducting qubits are the Flux, and Phase types.

Spin Qubits

Spin Qubits, sometimes called Charge Qubits or Quantum Dot Qubits, are exciting because they have considerable potential for scalable manufacturing. Due to similar manufacturing processes, charge qubits could be incorporated into the existing CMOS Transistor format, leveraging the extensive manufacturing infrastructure and compatibility with existing electronic systems.

A quantum dot is a minuscule ‘island’ or piece of conductive material placed on a substrate such as silicon. To form a qubit, a pair of quantum dots are placed near each other on the substrate, with one dot representing $|0\rangle$, and the other representing $|1\rangle$. Using microwave pulses to control the flow of electrons between the two quantum dots, operations can be applied to the qubit. If a superconducting material is used to form the quantum dots, then *cooper pairs* move between the quantum dots, instead of single electrons (in superconducting material, free electrons form pairs which move together, known as cooper pairs). The state of the qubit can be determined by measuring which quantum dot has more electrons (or cooper pairs) at a given time.

To leverage existing CMOS manufacturing scale, quantum dots are placed between the source and the drain in a silicon CMOS transistor. This has exciting implications for the scalability of spin qubit technology. Notably, Intel and Harvard have been pursing type of this qubit technology.

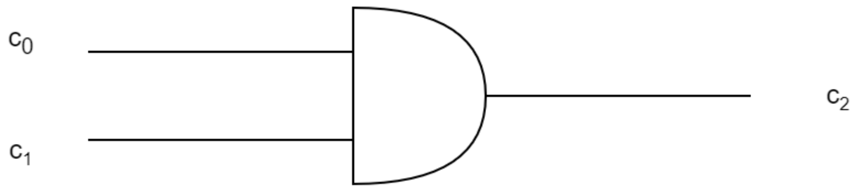
Gates and Circuits

The Quantum Circuit Model

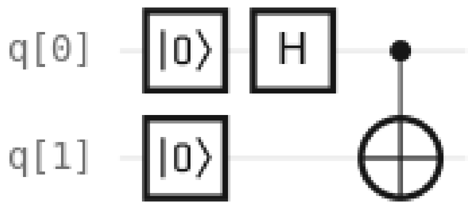
Classical and Quantum Circuits

Similar to how we can create boolean circuits to represent a sequence of boolean logic operations performed on classical bits, we can create quantum circuits to represent operations on performed on qubits. In both classical and quantum computing, the operations used in circuits are based on the fundamental operations we can perform on bits at the hardware level.

Boolean circuits consist of one or more input values or bits, in a state of either 0 or 1. Gates or operators are used to effect the state of a bit, or multiple bits, and the circuit results in one or more outputs, represented as bits in the 0 or 1 state. Lines are used to indicate connections between bits and gates, and different symbols are used to represent different gates. The following circuit diagram shows a classical circuit consisting of two initial states, or bits, c_0 and c_1 which are input to an *AND* gate, resulting in the output state c_2



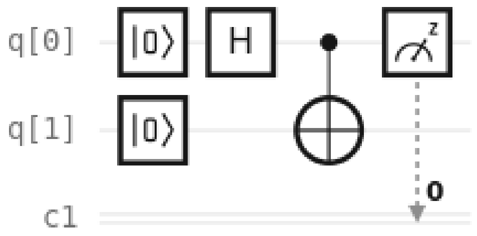
Example of a classical circuit diagram



Example of a quantum circuit diagram from IBM Quantum Composer

However, while classic circuits have the set of operators consisting of *NOT, AND, OR, NAND, NOR,* and *XOR* gates*,* the set of quantum circuit operators consists of a completely different set of gates. In fact, the *X* gate is the only quantum gate with an obvious classical analog (*NOT)*.

Note that in the above circuit digram there is no clear output state. Since we cannot know the actual state of a qubit until it is measured, quantum circuits typically consist of a classical bit or bits, which interact with the quantum bits through the measurement operator. While many quantum circuit tools provide will display the superposition of a qubit at the final stage of the circuit, it is the measurement of the quantum to a classical state that provides the resulting output of the circuit.



Example of a quantum state diagram with measurement from IBM Quantum Composer

Note the addition of the measurement operator in the above diagram. This measures the state of qubit q[0], and sets the classical bit c1 to the resulting classical state of that measurement. The *z* indicates that the measurement is made relative to the standard basis vector, or the z-axis or a Bloch sphere. This circuit above is an important circuit as it implements what is called a Bell state, by entangling the two qubits.

Circuit Depth

Similar to how classical circuits can be reduced to a universal gate set, programs written for classical computers are also reduced to simpler instructions. A classical computer's architecture doesn't know how to handle a switch statement, so a compiler converts the program into a series of operations supported by the architecture, such as adding or multiplying numbers to achieve more complex operations.

The same principles hold true for quantum circuits, and programs The operations in a circuit/program must be reduced into the gates of a universal gate set supported by the architecture of the quantum computer. Since we are often dealing with many qubits at a time, and some operations can be performed on the qubits in parallel, the circuit depth is a measure of the maximal number of sequential operations that take place in the circuit. In other words, circuit depth is a measure of programmatic efficiency, and is specific to the hardware on which the circuit is being run.

Universal Gate Sets

There are many parallels between classical and quantum circuits, including the concept of a universal gate. In classical circuits, any of the gates, and thus any possible circuit, can be built using a universal gate. For example, an AND gate can be built using only NAND gates

Quantum circuits have an equivalent concept, except there is no single universal gate than can be used to compose all possible operations on a qubit. Put another way, there is no single quantum gate that can be used to put a qubit into all it's possible superpositions of states. Instead, quantum circuits have *universal gate sets* - sets of operations which together can create any possible superposition of states, or combination of states of multiple qubits.

Notably, for quantum circuits, there are many universal gate sets. These gate sets are important when considering the physical design of a quantum computer, as they provide a minimum set of operations that the control infrastructure must be able to apply to a bit.

We will earn more about the types of gates used in quantum computing in the next section, but there are some common universal gate sets:

1. The Toffoli gate and the Hadamard gate *{CCNOT, H}*
2. The Clifford Group of operators and the T Gate *{CNOT, S, H, T}*

Below, we will take a look at some of the important quantum gates.

Unitary Operators

Unary operators, or gates, are a set of quantum operators that act upon a single qubit only. Since a qubit is represented by a 2 x 1 state vector, unary quantum operators are represented by a 2 x 2 matrix. The action of a gate on a qubit is equivalent to the matrix multiplication of the gate matrix by the quantum state vector.

Pauli X Operator (NOT Gate)

As was shown in the prerequisite math section, the state of a qubit can be represented by a state vector, and we can apply operations to that qubit by multiplying the state vector with a matrix. Similar to gates in classical computing, such as *AND, OR, and NOT*, in quantum computing these matrices are called gates, or operators. Take for example, the *X* operation, represented by the following matrix

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

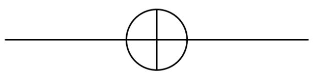
This is also referred to as the *NOT* gate, or *NOT* operator. Similar to the *NOT* operation in boolean algebra or classical computing, this operation will take a transform a qubit in the $|0\rangle$ state to the $|1\rangle$ state, and vice versa

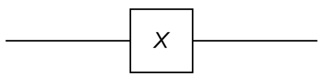
$$X|0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle$$

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

Since this gate is applied to a single qubit, it is referred to as a *Unary Operator*. This gate is also referred to as the *X* gate because the it is part of an important group of operators called the *Pauli Group*. Pauli operators are typically represented by σ, with a subscript indicating the particular gate. Therefore, the *NOT* gate is also referred to as σ_x or simply *X*.

In Quantum circuit diagrams, this gate is shown as one of two symbols, which can be used interchangeably.





Pauli Y Operator

Next we will look at another Pauli operator, the *Y* operator. Recall the Bloch sphere model for representing the state of a qubit. The *X* operator above would flip the state vector from one polar position, along the z-axis, to the other polar position. The *Y* operator works by rotating the state vector around the y-axis of the Bloch sphere, and is given by

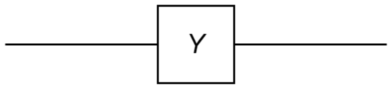
$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$$

Applying the *Y* operator to the basis states results in the following

$$Y|0\rangle = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ i \end{bmatrix} = i|1\rangle$$

$$Y|1\rangle = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -i \\ 0 \end{bmatrix} = i|0\rangle$$

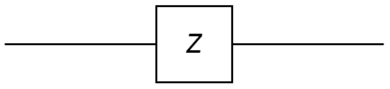
In a quantum circuit diagram, the *Y* operator is represented by



Pauli Z Operator

The third Pauli operator is referred to as σ_z or the *Z* operator. Similar to the *Y* operator, this operation rotates the state vector about the z-axis. Since the z-axis is indicative of the *phase* of the state vector, and the *Z* operator rotates the vector about the z-axis by 180 degrees, this operator is also referred to as the *phase-flip operator*.

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

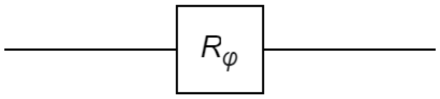


General Phase Shift Operator

Since the *Z* operator can be used to rotate a state vector around the z-axis by 180 degrees (π radians), this raises the question - can we rotate a state vector by other amounts? The answer is yes - this is done with the General Phase Shift operator. Note that this operator, and those following are not part of the Pauli group of operators. The general phase shift operator is represented by a capital *R* and a subscript φ representing the angle of rotation or phase:

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

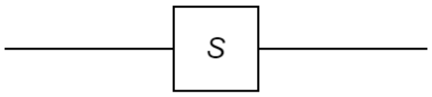
Note that when $\varphi = \pi,\ e^{i\varphi} = -1$, resulting in the Pauli *Z* operation.Therefore, *Z* operator is simply a special case of the R_φ operator, the case where $\varphi = \pi$. We will later look at some other special cases of the R_φ operator. The R_φ operator is represented in a circuit as



S Operator

Another special case of the general phase shift operator is the *S* operator. This is the case of the R_φ operator where $\varphi = \frac{\pi}{2}$. This results in a phase shift of 90 degrees around the z-axis and is represented by

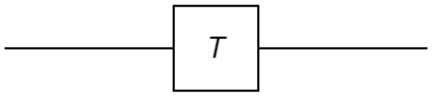
$$S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$$



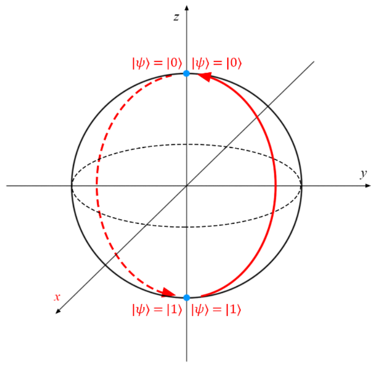
T Operator

Similar to the *S* operator is the *T* operator. Another special case of the general phase shift operator, this is the case of the R_φ operator where $\varphi = \frac{\pi}{4}$. This results in a phase shift of 45 degrees around the z-axis and is represented by

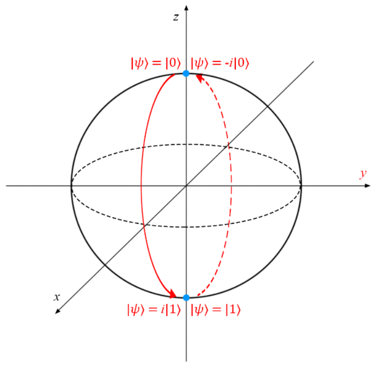
$$T = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}$$



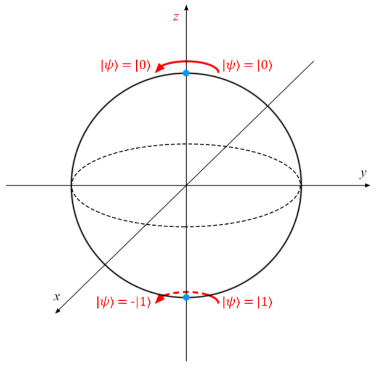
Visually we can represent the Pauli gate set as 180 degree rotations on a Bloch sphere, around the respective *x, y* and *z* axis. The R_φ, *S,* and *T* operators would result in similar rotations by the specific angle used in each gate.



Pauli *X*



Pauli *Y*

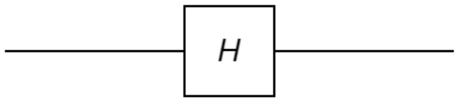


Pauli *Z*

Hadamard Operator

The Hadamard gate is unique among operators, in that it creates a superposition of states. This makes it an important gate in quantum computing.

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



Observe that when the *H* gate is applied to the $|0\rangle$ state, we obtain the following result

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

By applying the *H* gate to the $|0\rangle$ state, we have a resulting superposition of the $|0\rangle$ and $|1\rangle$ states. This is an important state called the $|+\rangle$ state. Similarly, there exists a $|-\rangle$ state

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

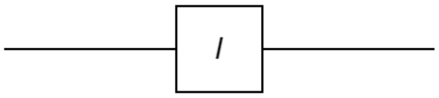
Because of this property, the Hadamard gate is a commonly encountered in quantum computing and is a key step in common methods of entangling qubits.

Identity Operator

The final unary operator we will discuss is the Identity, or *I* gate. Note that this is equivalent to a Identity matrix, and multiplying by this gate has no effect. Essentially, this operator maintains the current state of qubit.

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

$$I\left|0\right\rangle = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \left|0\right\rangle$$



Binary Operators

In addition to gates that can be applied to a single qubit, there are gates that can be applied to multiple qubits. Without these kinds of operators, we would never really have a system of multiple qubits, just many systems of individual qubits, without the ability to interact. The classical equivalent would be constructing a circuit using only NOT gates, which would not result in a very useful computer.

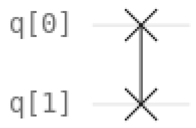
Binary operators are operators than are applied to two qubits, and therefore are represented by a 4 x4 matrix. Recall that a two qubit system can be represented by the tensor product of two individual qubits, resulting in a 4 x 1 state vector

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

SWAP Operator

The swap operator is a simple, symmetric binary operator. Acting on two qubits, it simply swaps the state of the two qubits. This swap operator is represented by the following matrix and circuit symbol

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



Observe the effect of the swap gate on a system of 2 qubits in the $|01\rangle$ state

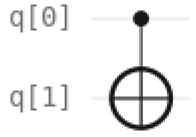
$$SWAP(|01\rangle) \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = |10\rangle$$

CNOT Operator

Short for controlled-NOT gate, the CNOT takes one of the qubits as the *control* qubit, and the other qubit as the *target*. Depending on the state of the control qubit, a NOT gate may or may not be applied to the target. If the control qubit is in the $|0\rangle$ state the target is not effected, and if the control qubit is in the $|1\rangle$ state, then a NOT gate is applied to the target.

The CNOT gate is sometimes referred to as the Controlled-X (CX) or Feynman gate, and is represented by

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

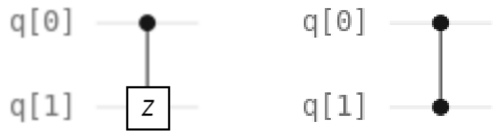


Note that the control qubit, in this case q[0], is indicated by the dot and the target qubit, q[1] is indicated by the symbol for the *X* gate.

CZ Operator

Similar to the CNOT or CX gate, the CZ gate is a controlled-Z gate. If the control qubit is in the 1 state, then a Z operator is applied to the target qubit.

$$CZ = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$



Note that there are two circuit symbols for the CZ diagram, one of which does not indicate which is the control qubit and which is the target. This is because the CZ gate is a symmetric operation, and the result will be the same regardless of which of the qubits is the control, and which is the target. This can also be observed in the matrix representation of the gate, which is also symmetric, because the matrix is equal to it's own transposition, such that $CZ^T = CZ$.

Quantum Computing Software

Quantum Simulators

When working in a quantum development environment, such as IBM's Quantum Platform, there is typically an option to run code on a quantum simulator. This is typically done to debug code and algorithms before sending the code to be run on an actual quantum processor. Since modern quantum computers still have a limited number of qubits, and time on a quantum computer is a timesharing process that may involve subscription costs, it makes sense to run code on a simulator first. However, if we can simulate a quantum computer using a classical computer, it begs the question, why bother to create a physical quantum computer at all? After all, they require advanced manufacturing, cryogenic freezing to near absolute zero, and large amounts of energy.

The answer lies in the memory required to represent the state of a qubit. A classical bit's two possible states can be represented by, well, a single classical bit. However, to usefully model a qubit we need to consider it's complex superposition of states. Each qubit in a system has two complex amplitudes describing it's superposition. If we use a byte of classical memory (8 bits), to represent each complex amplitude, then we require 2 bytes of memory to store the state of a single qubit. Therefore, for of a system of *n* qubits, we require 2^n classical *bytes* of memory. That's exponential growth - the classical memory required grows exponentially as the number of qubits increases.

As of 2024, we are in the NISQ era of quantum computers in the range of 10^2 - 10^3 qubits. However, to simulate a quantum system of 30 qubits requires $2^{30} = 1,073,741,824$ *bytes*, or a gigabyte of memory. To simulate 50 qubits requires a petabyte of memory, which is already pushing the limits of modern supercomputing. Useful, error-corrected quantum computers are in the range of at least 10^6 - 10^7 qubits. Therefore, to simulate a fully error-corrected quantum computer of at least 10^6 qubits requires $2^{10^6} = 2^{1,000,000}$ *bytes* of memory! For scale, when expanded the number $2^{1,000,000}$ is 301,030 digits long, and is over $10^{10,000}$ times greater than the estimated number of atoms in the universe, 10^{82} atoms.

While there are other methods for simulating quantum computers using classical computers, all current methods are limited by a level of exponential-growth memory requirements. Notably, there are existing simulators that can model thousands of qubits, but these simulators either do not support universal operation sets, or only approximate the behavior of a quantum computer.

Quantum Programming Languages

While there are interfaces for constructing quantum circuits using a GUI, this approach can be limited when working with more complex algorithms. That is where quantum programming languages come in. Like classical languages there are many variants, including interpreted/compiled languages and functional/interpreted languages. Note that the languages referred to below are sometimes called software development kits (SDKs), because they are sometimes implemented as libraries, and have associated development tools included.

Quantum Assembly Languages

QASM

QASM, or OpenQASM is short for Open Quantum Assembly Language, and is a low-level open-source quantum programming language. QASM is designed to communicate directly with quantum hardware, and allows for controlling the hardware at the pulse-level. QASM is also designed to allow higher-level languages to compile to QASM code, in order to run on a physical machine. As a result, QASM is used as a foundation for many higher-level quantum languages, such as Qiskit. The current version of QASM is QASM 3, which offers many improvements over QASM 2, including better integration with classical systems interacting with the quantum system.

Quantum Programming Languages

Qiskit

Backed by IBM, Qiskit - short for Quantum Information Science Kit - is a popular, open-source language based on Python. Technically, Qiskit is implemented as a Python library, and is underpinned by the OpenQASM quantum assembly language. Notably, Qiskit includes a robust system for generating circuit diagrams and other imagery.

Cirq

Developed by Google, Cirq is similar to Qiskit in that is is an open-source language implemented as a Python library. However, Cirq's differs from Qiskit in it's syntax, and that it does not make use of OpenQASM. Additionally, Cirq can only output circuit diagrams as ascii style images, although Cirq supports exporting programs to QASM code, allowing Cirq programs to be easily run in IBM's Quantum Platform, taking advantage of their robust development environment and publicly available quantum computers.

Q#

Developed by Microsoft, Q# differs from Qiskit and Cirq considerably in that it is much more closely related to the C family of languages. For example, Q# uses braces to distinguish code segments, and requires explicit type declarations. Additionally, Q# requires multiple files to execute a program, not unlike the header, code and executable files for a C style program.

Quantum Algorithms

Quantum Phenomena

To understand quantum algorithms and the advantages they offer over classical algorithms, there are two quantum principles that must be understood - interference and entanglement.

Interference

Attempts in popular literature to explain the quantum advantage often describe it as "the ability to compute all possible paths simultaneously." This is not strictly true. Interference works because of the phase attribute of a qubit. Consider a qubit in the $|0\rangle$ state with a Hadamard gate applied to it.

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

As you can see above, this results in a qubit in the the $|+\rangle$, an equal distribution of being in either basis state when measured. Let's observe what happens what a second Hadamard gate is applied to that same qubit

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

Interestingly, we have returned to the $|0\rangle$ state. Observe how when the Hadamard operator was applied, the superposition of states was altered. The amplitudes relating to $|0\rangle$ summed together, increasing the likelihood of the qubit being measured to $|0\rangle$. At the same time, the amplitudes relating to $|1\rangle$ have canceled out.

The result is that this operation increased the likelihood of the qubit being found in the $|0\rangle$ state. To illustrate this, let's conduct the same mathematical operation as above, but in a different way.

$$
\begin{aligned}
H\,|+\rangle &= H\left(\frac{1}{\sqrt{2}}\,|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\right) \\
&= \frac{1}{\sqrt{2}}H|0\rangle + \frac{1}{\sqrt{2}}H|1\rangle \\
&= \frac{1}{\sqrt{2}}\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix} + \frac{1}{\sqrt{2}}\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix} \\
&= \frac{1}{2}\begin{bmatrix}1\\1\end{bmatrix} + \frac{1}{2}\begin{bmatrix}1\\-1\end{bmatrix} = \begin{bmatrix}\frac{1}{2}\\ \frac{1}{2}\end{bmatrix} + \begin{bmatrix}\frac{1}{2}\\ \frac{-1}{2}\end{bmatrix} \\
&= \begin{bmatrix}\frac{1}{2} + \frac{1}{2}\\ \frac{1}{2} - \frac{1}{2}\end{bmatrix}^{\star} \\
&= \begin{bmatrix}1\\0\end{bmatrix} = |0\rangle
\end{aligned}
$$

The key step is indicated by the * is when the amplitudes sum or cancel each other out to provide the resulting qubit state. When the amplitudes sum together to increase the probability of a certain outcome, this is called *Constructive Interference*. When the amplitudes subtract, or cancel each other out, reducing the probability of a certain outcome, this is called *Destructive Interference.* Note that constructive and destructive interference must always occur together, to maintain the requirement that the sum of the absolute value of the probability amplitudes squared must be equal to one.

While the above example is trivial, it illustrates the concept of interference. When used in an algorithm, interference can be used to reduce the probability of 'incorrect' outcomes, and increase the probability of the system yielding a desired outcome.

For example, Grover's Search Algorithm makes use of interference to find a specific value in a list with a Big-O of O(root-n), an exponential improvement over the classical equivalent at O(n). Grover's is part of a whole class of quantum algorithms that rely primarily on interference, called Amplitude Amplification Algorithms.

Entanglement

Bell States

Along with the $|0\rangle$, $|1\rangle$, $|+\rangle$ and $|-\rangle$ states, there are other commonly encountered states in quantum computing. Some of the most important are the 4 Bell states, also called EPR states or an EPR pair, which we will look at now to help explain the phenomena of entanglement.

Consider the $|\phi^+\rangle$ state below

$$|\phi^+\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$$

It can be observed that this is a system comprised of two individual qubits, q_0 and q_1. Examining the probability amplitudes indicates that there is a 50% probability of the system being in the $|00\rangle$ state, and a 50% probability of the system being in the $|11\rangle$ state - when measured. Thus, according to the equation above there are only two possible outcomes for this system, the $|00\rangle$ state, and the $|11\rangle$ state. However, what happens if we were to measure only one of the qubits?

We can see from the equation there is a 50/50 chance of the *system* being in either the $|00\rangle$ or $|11\rangle$state. We might expect that measuring either one of the single qubits in the system would also result in a 50/50 chance of that qubit being the $|0\rangle$ or $|1\rangle$ state - and we would be correct.

If we only measure qubit-0 then 50% of the time the measurement will be in the $|0\rangle$ state and 50% of the time the measurement will be in the $|1\rangle$ state - we will see a 50/50 probability distribution. Similarly, if we only measure qubit-1 we will see also a 50/50 probability distribution.

Now, consider we wish to measure both the qubits. First, we will measure q_0, and then q_1. We expect there is a 50% chance of q_0 being in the $|0\rangle$ state, and 50% chance of q_0 being in the $|1\rangle$ state. We then measure q_1 and expect a 50% chance of q_1 being in the $|0\rangle$ state, and a 50% chance of q_1 being in the $|1\rangle$ state. Intuitively, there should be a 25% chance of the resulting measurements being $|00\rangle$, a 25% chance of the measurements being $|10\rangle$, a 25% chance of measuring $|01\rangle$, and a 25% chance of measuring $|11\rangle$.

We can think of this as the odds of flipping two coins.The two flips would result in equal probabilities of resulting heads-heads, heads-tails, tails-heads, and tails-tails.This is not what happens. What is actually observed is what Einstein called “spooky action at a distance.”

First, let's measure q$_0$. As discussed above, we have a 50% chance of measuring a $|0\rangle$ and a 50% chance of measuring a $|1\rangle$. Now let's measure q$_1$. What we find is that the measured state of q$_1$ is the same as the measured state as q$_0$ - every time. Even if we perform some set of operators to one of the entangled qubits but not the other, they will still maintain the same state as each other (or relative states, in the case of $|\phi^+\rangle$ and $|\phi^-\rangle$).

If we measured q$_0$ as being $|0\rangle$, then *whenever* and *wherever* we measure q$_1$, we will also find it in $|0\rangle$. Conversely, if we had measured q$_0$ as $|1\rangle$, then we would also measure q$_1$ to be $|1\rangle$. We call these two qubits *entangled*, and even if they are separated by some arbitrary distance, they will exhibit the same correlation of states. Even if that distance is so great that light could not possibly travel between the two qubits the time between the two measurements took place. Spooky action at a distance, indeed!

Using the coin-flip analogy, if we flipped the first of our *entangled* coins and got a heads, we could state with certainty that the other coin flip would also result in a heads - even if the entangled coins were so far apart from each other, that light itself would not have time to travel from one to the other between the flips. Once we have measure the qubits, however, the entangled state is broken.

It is important to note that there are 4 possible Bell states, each of which corresponds to initial states of each qubit when they are entangled. Here we can see how entanglement strongly correlates the qubit's relative to their initial states when entangled.

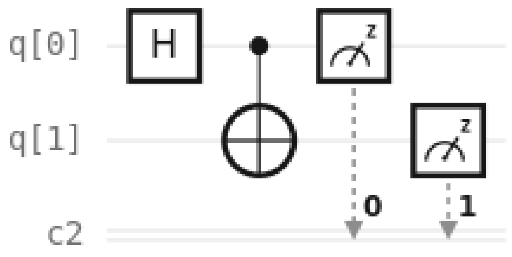
$$|\phi^+\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$$

$$|\phi^-\rangle = \frac{1}{\sqrt{2}}|00\rangle - \frac{1}{\sqrt{2}}|11\rangle$$

$$|\psi^+\rangle = \frac{1}{\sqrt{2}}|01\rangle + \frac{1}{\sqrt{2}}|10\rangle$$

$$|\psi^-\rangle = \frac{1}{\sqrt{2}}|01\rangle + \frac{1}{\sqrt{2}}|10\rangle$$

We can create the Bell $|\phi^+\rangle$ state using the following circuit. A Hadamard gate is applied to one of the qubits. The same qubit is then used as the control-bit in a CNOT operation between the two qubits. These operations create a Bell state. The measurement operators shown are not part of creating the Bell state, but illustrate the measurement process.



GHZ States and W States

Now that we have seen two qubits entangled, the question arises - can we entangle more than two qubits? The answer is yes. We can, in fact, entangle any number of qubits! The GHZ state is often used to refer to a particular entanglement of 3-qubits, although it can be generalized to any positive real number *n* qubits. The 3 qubit GHZ state is shown below.

$$|GHZ\rangle = \frac{1}{\sqrt{2}}|000\rangle + \frac{1}{\sqrt{2}}|111\rangle$$

When the GHZ state is generalized to any number of qubits, the defining factor is that the every qubit in the system will be measured in the same state - either all measured to $|0\rangle$, or all measured to $|1\rangle$.

Similar to the Bell ϕ states, we can have entangled states of *n* qubits where there all not all measured to one particular state. The W state is such a state of entanglement for a 3 qubit system. The W state is also an interesting example as it’s probability distribution is across 3 potential outcomes.

$$|W\rangle = \frac{1}{\sqrt{3}}|001\rangle + \frac{1}{\sqrt{3}}|010\rangle + \frac{1}{\sqrt{3}}|100\rangle$$

Examples of Quantum Algorithms

Deutsch’s Algorithm

Deutsch’s Problem

One of the first quantum algorithms to demonstrate superiority over equivalent classical algorithms, Deutsch’s Algorithm solves what is known as the parity problem. Consider some unknown boolean function, say *f*, that takes one bit of input *x*, and has one bit of output *f(x)*. Without knowing anything about this function, we can describe it as one of two classes - a constant function or a balanced function.

x	$f_0(x)$	$f_1(x)$	$f_2(x)$	$f_3(x)$
0	0	1	0	1
1	0	1	1	0

In the case of one bit of input, and one bit of output, there are only four possible functions, shown in the above table. Two of the functions, $f_0(x)$ and $f_1(x)$ will always return 0, or always return 1 - these are called constant functions. However, $f_2(x)$ and $f_3(x)$ will return a 0 for one input, and a 1 for the other. Because they output an even number of 0’s and 1’s, we call these balanced functions. Put another way, if $f(0) = f(1)$ then the function *f* is constant, and if $f(0) != f(1)$ then the function *f* is balanced.

The problem of determining if our unknown function *f* is constant or balanced is known as Deutsch’s Problem, and while it may seem trivial, it is an excellent illustration of one of the advantages quantum computers can offer. For a classical algorithm to solve Deutsch’s problem, you would need to query the function *f* twice. Once to observe the output with an input of 0, and once to observe the output with an input of 1. This would provide us with enough information to determine if the function is constant or balanced. It might be reasonable, even intuitive, to think that this was the best case solution. However, Deutch’s Algorithm manages to solve this problem with just one query!

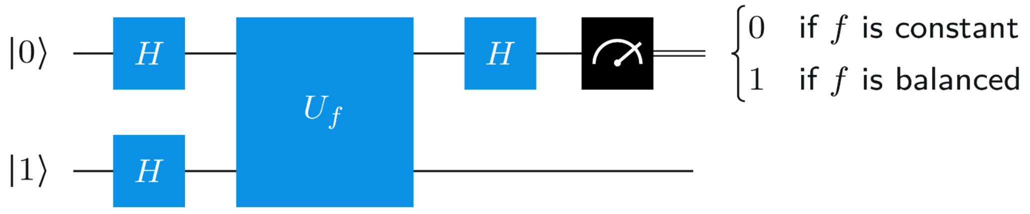
Query Problems

This type of problem is what’s known as a query problem, in which we are tying to determine something about an unknown function *f* - sometimes called an oracle, or a black-box[1]. We have no understanding of how this function works internally, but we can provide input to the function, and observe output. In other words, we can ‘query’ this function, with the goal of determining something about it.

[1] The terms *Oracle* and *Blackbox* are often used interchangeably, but in more advanced problems they may have different meanings.

In a quantum circuit, this unknown function is typically represented by a query gate. In query problems, algorithmic efficiency is measured by the number of queries made to the function.

To illustrate, consider Deutch's Algorithm which uses the query model of computation. The following following circuit implements this algorithm, with the oracle, *f*, represented by the U_f gate. We can refer to this U_f gate as a *query gate*.



$|0\rangle$
$|1\rangle$
H
H
U_f
H
$\begin{cases} 0 & \text{if } f \text{ is constant} \\ 1 & \text{if } f \text{ is balanced} \end{cases}$

You may be wondering what the purpose of the second input to the query gate is, because our function *f* is supposed to take one bit of input, and have one bit of output. Since the efficiency of a query problem is only measured by the number of queries made to the oracle, we can use additional qubits in the circuit without effecting the efficiency. When we perform the operation, we can store the result of the operation in the extra register, leaving the original qubit (or qubits!) unchanged[2].

By utilizing two qubits, we can take advantage of the relative phase difference between them, using interference to solve Deutch's problem. We will call this new operator a query gate and denote it U_f where *f* represents the function *U* is performing.

Query Gates

To understand how a query gate works, let's construct a reversible query gate U_f for some unknown function *f(x)*. Let *x* be the input to our function[3]. Let *y* be the additional bit we are using to make *f(x)* reversible. Therefore, we can represent the initial state of our quantum system as

$$|x\rangle|y\rangle$$

Although we don't know how *f(x)* works, we are able to *call* it. So without knowing what *f(x)* does, we can construct a query gate to act on our system, such that

$$U_f|x\rangle|y\rangle = |x\rangle|y \oplus f(x)\rangle$$

[2] Alternatively, we can store the input value in the extra register, and perform the operation on the original qubit

[3] The input *x* can be made up of any number of bits *n*, so this construction can be used for a query gate with any number of inputs

Note that in this context, ⊕ is used to represent the XOR or exclusive-OR operation. We can perform the XOR operation on a state vector as follows

$$|0 \oplus f(x)\rangle = |0\rangle \oplus f(x) = \begin{bmatrix}1\\0\end{bmatrix} \oplus f(x) = \begin{bmatrix}1 \oplus f(x)\\0 \oplus f(x)\end{bmatrix}$$

By *XORing* the output of the function with *y* we have made *U_f* a reversible implementation of *f(x).* To see why this works, observe what happens when y = 0

$$U_f|x\rangle|0\rangle = |x\rangle|0 \oplus f(x)\rangle$$

There are two possible cases, where *f(x)* = 1, and *f(x)* = 0

$$\text{Case 1.} \quad \text{If f(x)} = 0\text{:} \quad |0 \oplus f(x)\rangle = |0 \oplus 0\rangle = |0\rangle = |f(x)\rangle$$

$$|0 \oplus f(x)\rangle = |f(x)\rangle$$

$$\text{Case 2.} \quad \text{If f(x)} = 1\text{:} \quad |0 \oplus f(x)\rangle = |0 \oplus 1\rangle = |1\rangle = |f(x)\rangle$$

$$|0 \oplus f(x)\rangle = |f(x)\rangle$$

In both cases, the the value of the *y* qubit ends up equal to the output of the function *f(x),* and our original input remains unchanged. Knowing both the input and output states means that the function U_f is completely reversible, and hence a valid quantum operation. Since the result is the same in both cases, when y = 0

$$U_f|x\rangle|0\rangle = |x\rangle|f(x)\rangle$$

It is also worth observing that when *y* = *1*, in both cases

$$|1 \oplus f(x)\rangle = |-f(x)\rangle = \left|\overline{f(x)}\right\rangle$$

Phase Query Gate

Before exploring Deutch's algorithm, there is a special case of query gate we should consider, the Phase Query Gate or Phase Oracle. We already considered the case where *y = 0*, but let's consider the case where the *y* qubit is in the $|-\rangle$ state, such that our system begins in the state

$$|x\rangle|-\rangle$$

Before applying the query gate, let's expand the $|-\rangle$ state, which will help to clarify the operations.

$$|x\rangle|-\rangle \;=\; |x\rangle\left(\frac{1}{\sqrt{2}}|0\rangle \;-\; \frac{1}{\sqrt{2}}|1\rangle\right)$$

$$= \frac{1}{\sqrt{2}}|x\rangle(|0\rangle \;-\; |1\rangle)$$

Now we will apply the query gate U_f to the system

$$U_f|x\rangle|-\rangle \;=\; \frac{1}{\sqrt{2}}U_f|x\rangle(|0\rangle \;-\; |1\rangle)$$

$$= \frac{1}{\sqrt{2}}(U_f|x\rangle|0\rangle \;-\; U_f|x\rangle|1\rangle)$$

$$= \frac{1}{\sqrt{2}}(|x\rangle|0 \,\oplus\, f(x)\rangle \;-\; |x\rangle|1 \,\oplus\, f(x)\rangle)$$

Recall that the XOR of 0 and *f(x)* is always 0, and the XOR of 1 and *f(x)* is always *-f(x)*. This results in the following

$$= \frac{1}{\sqrt{2}}\left(|x\rangle|f(x)\rangle \;-\; |x\rangle\left|\overline{f(x)}\right\rangle\right)$$

This again results in two possible cases, when *f(x) = 0, and f(x) = 1*

$$\text{Case 1.}\quad \text{If f(x)} = 0: \qquad \frac{1}{\sqrt{2}}(|x\rangle|0\rangle \;-\; |x\rangle|1\rangle)$$

$$= \frac{1}{\sqrt{2}}|x\rangle(|0\rangle \;-\; |1\rangle)$$

$$= |x\rangle\left(\frac{1}{\sqrt{2}}|0\rangle \;-\; \frac{1}{\sqrt{2}}|1\rangle\right)$$

$$= |x\rangle|-\rangle$$

$$\text{Case 2.}\quad \text{If f(x)} = 1: \qquad \frac{1}{\sqrt{2}}(|x\rangle|1\rangle \;-\; |x\rangle|0\rangle)$$

$$= \frac{1}{\sqrt{2}}|x\rangle(|1\rangle \;-\; |0\rangle)$$

$$= -\frac{1}{\sqrt{2}}|x\rangle(|0\rangle \;-\; |1\rangle)$$

$$= -|x\rangle\left(\frac{1}{\sqrt{2}}|0\rangle \;-\; \frac{1}{\sqrt{2}}|1\rangle\right)$$

$$= -|x\rangle|-\rangle$$

The two cases can be represented in one equation, as such

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

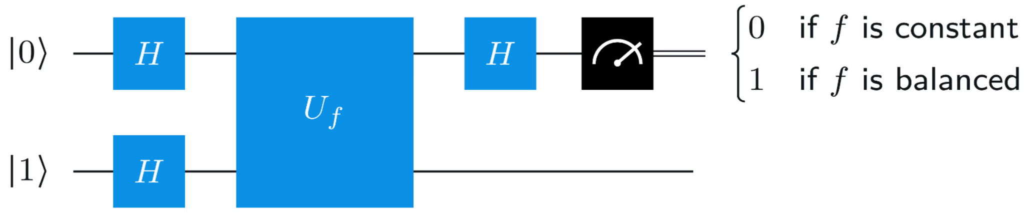
Technically, the query gate has not changed, only the state of the *y* qubit when the gate is applied. None the less, this can be thought of as a phase query gate. This gate can be used to employ the phase kickback technique in quantum algorithms, and it's application will become apparent as we explore Deutch's algorithm.

Deutch's Algorithm

Deutch's Algorithm is a quantum algorithm that can be used to solve Deutch's problem, and is notable for being this first quantum algorithm to demonstrate quantum supremacy over the best known corresponding classical algorithms.

The goal here is to query a one-bit function and determine if it is constant or balanced. A classical algorithm requires, at best, two queries to make a determination, but Deutch's Algorithm can do this in only 1 query.

While it may not have many practical applications, it is one of the simplest quantum algorithms and therefore useful to illustrate some essential quantum algorithmic techniques. Consider the circuit diagram for Deutch's algorithm shown below.


$|0\rangle$
$|1\rangle$
H
H
U_f
H
$\begin{cases} 0 & \text{if } f \text{ is constant} \\ 1 & \text{if } f \text{ is balanced} \end{cases}$

To perform Deutch's Algorithm, we begin with two qubits. The first qubit[4] is the input *x* to the function, and we initialize this qubit to the $|0\rangle$ state. The other qubit present is the *y* qubit, and it is initialized to the $|1\rangle$ state. This results in a system in the following state

$$|x\rangle|y\rangle \;=\; |0\rangle|1\rangle$$

Next, we perform Hadamard operations on both the *x* and the *y* qubits. Regarding the *x* qubit, this can be thought of as inputting a superposition of 0 and 1 into the function, or both 0 and 1. Since the *x* qubit was in the $|0\rangle$ state, this results in the $|+\rangle$ state. And since the *y* qubit was in the $|1\rangle$ state, this results in a qubit in the $|-\rangle$ state.

$$H(|x\rangle|y\rangle) \;=\; |+\rangle|-\rangle$$

[4] The qubits in a circuit diagram are described from top (first) to bottom (last).

To better illustrate what is happening, we will expand the $|+\rangle$ state

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

$$= \frac{1}{\sqrt{2}}(|0\rangle \;+\; |1\rangle)\,|-\rangle$$

$$= \frac{1}{\sqrt{2}}(|0\rangle|-\rangle \;+\; |1\rangle|-\rangle)$$

Next, we will apply the query gate U_f gate. Since the y qubit is in the $|-\rangle$ state, the query gate will function as a phase query gate. Applying the query gate U_f to our system results in

$$U_f|+\rangle|-\rangle \;=\; \frac{1}{\sqrt{2}}U_f(|0\rangle|-\rangle \;+\; |1\rangle|-\rangle)$$

$$= \frac{1}{\sqrt{2}}(U_f|0\rangle|-\rangle \;+\; U_f|1\rangle|-\rangle)$$

Recall the equation for a phase query gate:

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

We can observe that the system contains two such structures. Substituting these into our equation provides us with the following

$$U_f|+\rangle|-\rangle \;=\; \frac{1}{\sqrt{2}} U_f(|0\rangle|-\rangle \;+\; |1\rangle|-\rangle)$$

$$= \frac{1}{\sqrt{2}}\Big((-1)^{f(0)}|0\rangle|-\rangle \;+\; (-1)^{f(1)}|1\rangle|-\rangle\Big)$$

$$= \frac{1}{\sqrt{2}}\Big((-1)^{f(0)}|0\rangle \;+\; (-1)^{f(1)}|1\rangle\Big)\,|-\rangle$$

At this point, the *y* qubit is no longer needed. Since we are only measuring the *x* qubit, we can discard the *y* qubit from the equation, providing

$$\frac{1}{\sqrt{2}}\Big((-1)^{f(0)}|0\rangle \;+\; (-1)^{f(1)}|1\rangle\Big)$$

Recall that in Deutch's Problem, if *f(0) = f(1)* then *f* is constant, and if *f(0)* $\neq$ *f(1)* then *f* is balanced. Therefore, there are two possible cases to consider. Note that case 1 considers what occurs when the function *f* is a constant function, and case 2 considers what occurs when the function is balanced.

Case 1. If $f(0) = f(1)$:

Here, there are two subcases. Either *f(0) = f(1) = 0, or f(0) = f(1) = 1.*

Subcase 1.1 If $f(0) = f(1) = 0$: $\frac{1}{\sqrt{2}}\Big((-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle\Big)$

$$= \frac{1}{\sqrt{2}}\Big((-1)^{0}|0\rangle + (-1)^{0}|1\rangle\Big)$$

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

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

$$= |+\rangle$$

Subcase 1.2 If $f(0) = f(1) = 1$: $\frac{1}{\sqrt{2}}\Big((-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle\Big)$

$$= \frac{1}{\sqrt{2}}\Big((-1)^{1}|0\rangle + (-1)^{1}|1\rangle\Big)$$

$$= \frac{1}{\sqrt{2}}(-1|0\rangle + -1|1\rangle)$$

$$= -\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

$$= -|+\rangle$$

Since subcase 1.1 results in $|+\rangle$ and subcase 1.2 results in $-|+\rangle$, we will generalize case 1 so that the resulting state of the *x* qubit is

$$\pm|+\rangle$$

Case 2. If $f(0) \neq f(1)$:

Again there are two subcases to consider, either *f(0) = 0* and *f(1) = 1,* or *f(0) = 1* and *f(1) = 0*.

Subcase 2.1 If $f(0) = 0$ and $f(1) = 1$: $\frac{1}{\sqrt{2}}\Big((-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle\Big)$

$$= \frac{1}{\sqrt{2}}\Big((-1)^{0}|0\rangle + (-1)^{1}|1\rangle\Big)$$

$$= \frac{1}{\sqrt{2}}(1|0\rangle + (-1)|1\rangle)$$

$$= \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

$$= |-\rangle$$

Subcase 2.2 If $f(0) = 1$ and $f(1) = 0$: $\frac{1}{\sqrt{2}}\Big((-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle\Big)$

$$= \frac{1}{\sqrt{2}}\Big((-1)^{1}|0\rangle + (-1)^{0}|1\rangle\Big)$$

$$= \frac{1}{\sqrt{2}}((-1)|0\rangle + 1|1\rangle)$$

$$= -\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

$$= -|-\rangle$$

Since subcase 2.1 results in $|-\rangle$ and subcase 2.2 results in $-|-\rangle$, we will generalize case 2 so that the resulting state of the *x* qubit is

$$\pm|-\rangle$$

The result is that after the U_f gate, the *x* is in two possible states[5], either $\pm|+\rangle$ or $\pm|-\rangle$.

The final step of the algorithm, before measuring the result, is to apply a final Hadamard gate to the *x* qubit. We will do this for both cases

$$\text{Case 1.}\quad \text{If } f(0) = f(1): \quad H(\pm|+\rangle)$$

$$= \pm H|+\rangle$$

$$= \pm|0\rangle$$

$$\text{Case 2.}\quad \text{If } f(0) \neq f(1): \quad H(\pm|-\rangle)$$

$$= \pm H|-\rangle$$

$$= \pm|1\rangle$$

Finally, we are able to measure the *x* qubit. In case 1, when the function *f* was constant, the *x* qubit is in the $\pm|0\rangle$ state, and measuring it will result in a 0, with certainty.[6] In case 2, when the function *f* was balanced, the *x* qubit is in the $\pm|1\rangle$ state, and measuring it will result in a 1 with certainty.

Therefore, regardless of what function *f* is used, if the function is constant the output of Deutch's algorithm will be a 0, and if *f* is balanced the output will be a 1. Therefore, Deutch's algorithm can make the determination using only one query.

This algorithm employs a common interference technique in quantum algorithms, known as the Phase Kickback Phenomenon. This results from the use of the phase-query gate and can be described as 'encoding the result in the global phase.' This technique illustrates the advantages quantum algorithms have over classical, and is used in many quantum algorithms such as the Deutsch-Jozsa Algorithm and Grover's Algorithm.

[5] Technically there are 4 possible states if we consider the +/- variations, but these will not impact the measured outcome of the algorithm.
[6] We do not need to consider the +/- as it has no bearing on the outcome of the measurement. Observe than the squaring of the amplitudes eliminates any relative phase when a measurement is performed, so that the resulting outcome is the same.

Other Quantum Algorithms

While Deutch's algorithm solves a somewhat trivial problem, it is a demonstration of quantum supremacy - an increased algorithmic efficiency over the best known (in this case best possible) classical solution. However, there are many quantum algorithms that solve problems with real-world applications.

The Deutch-Jozsa algorithm is a generalization of Deutch's algorithm, solving Deutch's problem for an input of *n* bits. Doing so still requires only one additional bit to be used (*n + 1* bits total*)*, and regardless of the input size, can still solve the problem in only one query! In comparison, a classical algorithm requires at best *n* times.

Other important quantum algorithms include Grover's search algorithm, which can find an element in a list in $O(\sqrt{n})$, which is a quadratic speedup over classical search algorithms. In the case of Grover's algorithm, this is also the optimal solution, so that $O(\sqrt{n}) = \Omega(\sqrt{n})$. Grover's algorithm does this by utilizing a query gate and the phase-kickback phenomena in multiple stages, not unlike Deutch's algorithm. However, not all quantum algorithms solve query problems.

Shor's algorithm is used to find the prime factorization of a number, an important operation in cryptography. The best known classical algorithms can achieve sub-exponential time, but Shor's algorithm achieves polynomial time efficiency, a considerable advantage that means quantum algorithms could be used to break current encryption standards.

As it currently stands, many quantum algorithms already exist, and it is an area of active research. Quantum algorithms have the potential to impact various fields, including machine learning, chemistry, physics, and cryptography. There are different classes of quantum algorithms, some of which utilize interference, and others which take advantage of entanglement or other quantum phenomena.

Summary

While the algorithmic speedup offered by quantum computers may not seem substantial, consider that there are existing problems that may take existing classical machines years, or even decades, to compute! In this case, even a quadratic speedup might make the difference in a problem being *impractically* computable to be *practically* computable.

Already researchers are anticipating when quantum computers can brute-force decrypt existing encryption schemes, and are anticipating a post-quantum era of cryptography. As quantum computing nears the era of useful, error-corrected quantum hardware, we have the possibility to solve problems in various fields that would not be practical on classical hardware.

Quantum hardware will, for the foreseeable future, require ancillary hardware such as supercooling and noise isolation. However, if the last century of industrial evolution has taught us anything, it is that the miniaturization and optimization is inevitable. In the later half of this decade, IBM is planning to develop quantum processors with 10,000 to 100,000 qubits, and each year more and more companies enter the race for quantum development - experimenting with various types of qubit, software, and hardware designs as well as strategies for monetization.

As computer science experiences a boom in global popularity, so does quantum computing, with increasing numbers of post-secondary institutions offering, or expanding on, quantum-related graduate programs (and even undergraduate classes), introducing a new generation of students to quantum computing. As quantum systems scale in size, more and more quantum algorithms developed theoretically will become practical to run on hardware devices. Despite the exceptional growth in computing power over the last few decades, there are still many computations that cannot be run in a reasonable amount of time. Quantum computing has the potential to bring those problems within reach, opening new avenues of science and discovery.

References

https://learning.quantum.ibm.com

https://learn.microsoft.com/en-us/azure/quantum/concepts-dirac-notation

https://learning.quantum.ibm.com/course/basics-of-quantum-information/multiple-systems

https://math.libretexts.org/Bookshelves/Precalculus/Precalculus_1e_(OpenStax)/03%3A_Polynomial_and_Rational_Functions/3.01%3A_Complex_Numbers

https://www.scientificamerican.com/article/what-are-josephson-juncti/

https://ebookcentral.proquest.com/lib/viu/reader.action?docID=6737930

https://openqasm.com/intro.html#scope

https://www.youtube.com/watch?v=7MdEHsRZxvo

https://quantumcomputing.stackexchange.com/questions/50/how-much-memory-is-required-to-simulate-a-48-qubit-circuit

https://leimao.github.io/images/blog/2020-06-14-Qubit-Bloch-Sphere/Bloch_sphere.svg.png

https://www.youtube.com/watch?v=jfJckA7Amik

https://scitechdaily.com/first-experimental-proof-that-quantum-entanglement-is-real/

https://quantumcomputing.stackexchange.com/questions/1474/what-programming-languages-are-available-for-quantum-computers

https://en.wikipedia.org/wiki/Quil_(instruction_set_architecture)

https://learning-api.quantum.ibm.com/assets/98e3e1ff-a50e-443c-9927-5221659366ae?format=auto&quality=80

https://www.youtube.com/watch?v=jfJckA7Amik

https://learning-api.quantum.ibm.com/assets/98e3e1ff-a50e-443c-9927-5221659366ae?format=auto&quality=80