Session 1: How to Program a Quantum Computer

Dominic Moylett
Quantum Engineering Centre for Doctoral Training,
University of Bristol
dominic.moylett@bristol.ac.uk

July 25, 2016

Worksheet Details

This worksheet is for the first session of the Quantum Computing day in the 2016 "Quantum in the Summer" school. This session is focused on introducing the key concepts in quantum computing: The quantum bits that we use to store data and the quantum gates that we use to manipulate that data.

1 What is Quantum Computing?

Quantum physics is a collection of bizarre behaviours that happen to really small particles. The nature of quantum physics makes simulations of these particles really hard, even on today's fastest computers.

Quantum computing came about after a number of proposals by physicists such as Richard Feynman. They suggested that in order to make computers able to simulate quantum systems, we should build computers that take advantage of the same effects that makes quantum physics hard to simulate. These computers run on a very different model of computation to our current laptops and smart phones, and can lead to more powerful machines in turn. In the decades that have followed since then, academics at many institutions – including the University of Bristol – have studied how quantum physics can benefit our technology, and how we can realise these benefits in practice.

In this workshop, we aim to highlight the building blocks of quantum computers: Quantum bits and quantum gates. We will see how these building blocks can give way to the bizarre physical effects of superposition and entanglement, and how we can take advantage of these concepts to perform challenges such as teleportation. These concepts will be used even further in Session 2, when we look at how quantum computers can speed up our computation.

To understand quantum computing, we will use QCEngine, a quantum computing simulator developed by the University of Bristol Centre for Quantum Photonics' own Eric Johnston. All you need to get started is a laptop with a web browser installed. Open up your browser of choice and go to the website `http://machinelevel.com/qc/doc/qcengine_workbench.html`. This will load the engine with a default script. Simply delete this script and you will then be ready to explore the world of quantum computing!

2 Quantum Bits

A classical computer is made up of bits: 1s and 0s that represent data. Quantum computers store data in quantum bits, or *qubits*. We write qubits as $|1\rangle$ and $|0\rangle$. These symbols are called *kets*[1], and are used to describe quantum states.

In QCEngine, we define qubits using the command `qc.reset`, which we write at the top of our code. This specifies the number of qubits we will be using. We then set these qubits to $|0\rangle$ using the command `qc.write(0)` Try running the code below by typing it into the text box and pressing "Run this":

```
qc.reset(1); // generate 1 qubit
qc.write(0); // set state to |0>
```

This code initialises 1 qubit in the $|0\rangle$ state. We can see this in the box on the right, under the section "State Vector". This section shows a blue circle for the $|0\rangle$ state and a white circle for the $|1\rangle$ state, meaning that the state is $|0\rangle$. We can also write `qc.write(1)` to set the state to $|1\rangle$.

We also see another box above, which shows off what our quantum circuit looks like. Clicking on any part of the circuit will show us what the quantum state looks like at that point in the circuit. Finally, it is also worth mentioning that the white square in the corner of the state box will make the box larger or smaller, "Ctrl + Scroll wheel" will make the contents of the state box larger, and "Shift + Scroll wheel" will make the state vectors larger.

3 Measurement

While images like this showing off our complete quantum state are nice, we cannot look at quantum states this way in practice. Instead, we have to *measure* our qubits. We can do this using the `qc.read` command:

[1] This notation was invented by Paul Dirac, one of the best-known quantum physicists and born and raised in Bristol.

```
qc.reset(1); // generate 1 qubit
qc.write(0); // set state to |0>
var result = qc.read(); // measure state
qc.print("The state is |" + result + ">\n"); //
    print measured state
```

The third line measures qubit 0 and puts the result in the `result` variable. The fourth line displays the result.

From this we get the output "The state is $|0\rangle$". So, if we set our qubit to $|0\rangle$ and measure it, we find that the measured qubit is in the $|0\rangle$ state.

4 Quantum Gates

But data is only one half of computation. We also need be able to perform operations. Classical computers do this with logic gates, such as the NOT gate, which flips a bit: $NOT(0) = 1$ and $NOT(1) = 0$.

In quantum computing, we also have logic gates, which are called quantum gates. One example is the X gate, which is the quantum equivalent of a classical NOT gate: $X|0\rangle = |1\rangle$ and $X|1\rangle = |0\rangle$. Let's apply an X gate to qubit 0 in QCEngine:

```
qc.reset(1);
qc.write(0);
qc.not()
```

Now our circles have swapped; the blue circle is now above the $|1\rangle$ state and the white circle is now above the $|0\rangle$ state, just like we saw when we initialised our qubit to $|1\rangle$. So qubit 0 has flipped from $|0\rangle$ to $|1\rangle$. We can also check by measuring:

```
qc.reset(1);
qc.write(0);
qc.not()
var result = qc.read();
qc.print("The state is |" + result + ">\n");
```

Now when we measure our qubit, we find that the result is $|1\rangle$. We can also check the other direction by applying X to a qubit in the $|1\rangle$ state and find that $X|1\rangle = |0\rangle$.

Task 4.1. *What do you think will happen if we apply X twice to one state? Check your answer by running it in the engine.*

5 Superposition

Another example of a single qubit gate is the Hadamard gate, H. Let's see what this gate does to a qubit in the $|0\rangle$ state:

```
qc.reset(1);
qc.write(0);
qc.hadamard(1); // apply a Hadamard gate
```

Now our state vector has partially blue circles for both the $|0\rangle$ and $|1\rangle$ states. Let's measure this state a few times to see what happens:

```
qc.reset(1);
qc.write(0);
qc.hadamard(1);
var result = qc.read();
qc.print("The state is |" + result + ">\n");
```

Running this multiple times, we see that sometimes we get $|0\rangle$ and other times we get $|1\rangle$. This is called a superposition, where measuring the qubit could give $|0\rangle$ or $|1\rangle$. Until we measure the state, we do not know what the result will be.

Now that we've seen what happens when we input $|0\rangle$, what happens when we input $|1\rangle$? Let's first try without measuring:

```
qc.reset(1);
qc.write(1);
qc.hadamard(1);
```

Now we see the same picture as before, but the line in the $|1\rangle$ state is pointing downwards. What about when we measure it?

```
qc.reset(1);
qc.write(1);
qc.hadamard(1);
var result = qc.read();
qc.print("The state is |" + result + ">\n");
```

It seems we get the same measurement results as before: sometimes we get $|0\rangle$ and sometimes we get $|1\rangle$. So does this mean the Hadamard gate produces the same result regardless of what we put into it?

The answer is no; the states produced by the Hadamard gate are as different as the states $|0\rangle$ and $|1\rangle$. The state produced by applying the Hadamard gate to a $|0\rangle$ gate is $|+\rangle = 1/\sqrt{2}(|0\rangle + |1\rangle)$, while the state produced by applying the gate to $|1\rangle$ is $|-\rangle = 1/\sqrt{2}(|0\rangle - |1\rangle)$.

The Bloch sphere is one way of visualising single qubits, by displaying them as points on a sphere. If we click on the "Bloch sphere" link, we can see that the $|+\rangle$ and $|-\rangle$ are on opposite sides of the sphere, just like the $|0\rangle$ and $|1\rangle$ states. You can use the Scroll wheel on your mouse to rotate the Bloch sphere.

But how likely are we to measure $|0\rangle$ or $|1\rangle$? If we measure the state $a|0\rangle + b|1\rangle$, the probability of measuring $|0\rangle$ is $|a|^2$, and the probability of measuring $|1\rangle$ is $|b|^2$. So if we measure the $|+\rangle$ or $|-\rangle$ state, we find the probability of measuring $|0\rangle$ or $|1\rangle$ both $|1/\sqrt{2}|^2 = 1/2$.

You might wonder what would happen if we measured a state in superposition twice. Well, let's find out:

```
qc.reset(1);
qc.write(1);
qc.hadamard(1);
var result = qc.read();
qc.print("The state is |" + result + ">\n");
var result = qc.read();
qc.print("The state is |" + result + ">\n");
```

It appears that while the first measurement is random, the second measurement is the same as the first. This is because

after a measurement, the state is no longer in a superposition. Instead, it has collapsed into the state we measured. If we want our superposition, we need to set the whole experiment up again.

Task 5.1. *Take a quantum state $a|0\rangle + b|1\rangle$. What do you think $|a|^2 + |b|^2$ equals?*

Task 5.2. *What do you think will happen if we apply two Hadamard gates to the qubit $|0\rangle$? Try it in the QCEngine and see what actually happens. Was it what you expected? Try doing the same thing to the qubit $|1\rangle$ too.*

6 Multiple Qubits

Multiple qubits act similarly to multiple classical bits, where we can apply quantum gates to the individual qubits. For example, if the first qubit is $|0\rangle$ and the second is $|1\rangle$, we write this state as $|01\rangle$.

So for example, we can apply an X gate to the first and second qubits as follows:

```
qc.reset(2); // generate 2 qubits
qc.write(0); // write the state |0> = |00>
qc.not(1); // apply an X gate to qubit 1
qc.not(2); // apply an X gate to qubit 2
var result = qc.read(); // measure both qubits
qc.print("The state is |" + result + ">\n"); //
    print measured state
```

And the measured state turns out to be $|3\rangle$. It is worth noting that QCEngine writes this in decimal, which when converted to binary becomes $|11\rangle$. Thus multiple qubits act similarly to when we were acting only on individual qubits. Likewise, if we applied X to only qubit 0, then we would find that they were in the $|2\rangle$ state in decimal, or the $|10\rangle$ state in binary. So applying a quantum gate to one qubit doesn't affect the other.

If we apply a Hadamard gate to one qubit, we create a superposition over that qubit while not producing a superposition over the other:

```
qc.reset(2);
qc.write(0);
qc.hadamard(1);
var result = qc.read();
qc.print("The state is |" + result + ">\n");
```

Like single qubits, we can produce any superposition over two qubits. Thus our state can be any form of $a|00\rangle + b|01\rangle + c|10\rangle + d|11\rangle$ where $|a|^2 + |b|^2 + |c|^2 + |d|^2 = 1$. Some example states include $|01\rangle$, $\frac{|00\rangle+|10\rangle}{\sqrt{2}}$ and $\frac{|00\rangle+|01\rangle+|10\rangle+|11\rangle}{2}$.

It is worth noting that when applying a gate to the i-th qubit in QCEngine, you provide 2^{i-1} as an argument to that gate. This code below, for example, will apply an X gate to the fourth qubit by calling `qc.not(8)`, thus resulting in the state $|8\rangle = |0001\rangle$:

```
qc.reset(4); // generate 3 qubits
qc.write(0); // write the state |0> = |0000>
qc.not(8); // apply an X gate to qubit 4
var result = qc.read();
qc.print("The state is |" + result + ">\n");
```

Task 6.1. *One of the hardest things to do in classical computation is generate random 8-bit numbers such that every number is as likely to be generated as every other number. How might you use a quantum computer to generate 8-bit numbers purely at random? Try programming this in QCEngine.*

7 Entanglement

Not all classical gates are single bit. The classical AND gate for example, takes two classical bits x and y as input and outputs one classical bit which is 1 if $x = y = 1$ and 0 otherwise.

We also have multiple qubit quantum gates. One of particular importance is the Controlled-NOT gate, or $CNOT$, which takes a control qubit as the first argument and a target qubit as the second. In QCEngine, the first argument of the $CNOT$ command is the target qubit, and the second argument is the control qubit. Let's see what happens when the control qubit is $|0\rangle$:

```
qc.reset(2);
qc.write(0);
qc.cnot(2,1); // apply a CNOT with control 1 and
    target 2
var result = qc.read();
qc.print("The state is |" + result + ">\n");
```

It seems that $CNOT|00\rangle = |0\rangle = |00\rangle$. Likewise, we can run again with our target qubit in the $|1\rangle$ state:

```
qc.reset(2);
qc.write(0);
qc.not(2);
qc.cnot(2,1);
var result = qc.read();
qc.print("The state is |" + result + ">\n");
```

Running this shows that $CNOT|01\rangle = |1\rangle = |01\rangle$. So when the control qubit is $|0\rangle$, the $CNOT$ gate does nothing. So what if the control qubit is $|1\rangle$?

```
qc.reset(2);
qc.write(0);
qc.not(1);
qc.cnot(2,1);
var result = qc.read();
qc.print("The state is |" + result + ">\n");
```

Now our target qubit has been flipped, so we can see that $CNOT|10\rangle = |3\rangle = |11\rangle$. Again, if our target qubit is $|1\rangle$:

```
qc.reset(2);
qc.write(0);
qc.not(1);
qc.not(2);
qc.cnot(2,1);
var result = qc.read();
qc.print("The state is |" + result + ">\n");
```

Measuring this shows that $CNOT|11\rangle = |2\rangle = |10\rangle$. So if our control qubit is $|0\rangle$, then our $CNOT$ gate does nothing, but if our control qubit is $|1\rangle$, then $CNOT$ applies an X gate to the target qubit.

But these aren't the only possible states our control qubit can take. What if our control qubit is in a superposition?

```
qc.reset(2);
qc.write(0);
qc.hadamard(1);
qc.cnot(2,1);
var result = qc.read();
qc.print("The state is |" + result + ">\n");
```

This produces a very different result. It now appears that our measurement of qubit 0 is random, but our measurement of qubit 1 is always the same state as qubit 0. If we measure qubit 1 first, then our measurement of qubit 1 will be random, but our measurement of qubit 0 will always be the same state as qubit 1. It seems that our qubits are in a superposition together, and if one collapses then the other collapses with it. We write this state as $\frac{|00\rangle+|11\rangle}{\sqrt{2}}$.

This effect is called *entanglement*, and is one of the strangest results of quantum mechanics. Entangled states collapse at exactly the same time, regardless of distance between the two qubits. One qubit could be in the University of Bristol HH Wills Physics Laboratory and the other could be at the University of Bristol Moon Base[2], and their collapse would still be at the same time. Classically, this simultaneous collapse should be impossible, as it would require one qubit to send a message describing its measurement result to the other faster than the speed of light, breaking the laws of relativity. This led Albert Einstein to describe quantum mechanics as "Spooky action at a distance."

While we haven't been able to perform the above experiment, physicists at a number of institutions – most famously Delft University of Technology – have managed to entangle qubits and witness these effects before light could have travelled from one qubit to the other. So there must be something more to quantum.

8 Teleportation

Now let's talk about something really exciting that we can do with quantum computers: Teleportation!

Teleporting a human is still very hard, even if quantum computing is easy. But we can still send a qubit from one person (called Alice) to another (called Bob) using only an entangled pair of qubits and two classical bits.

But first we need to create the quantum state that we want to teleport. We'll do this by starting with a qubit in the $|0\rangle$ state and applying rotations of various degrees in the Bloch sphere view:

```
qc.reset(3);
qc.write(0);

qc.codeLabel('prep'); // add a label to the circuit
qc.rotx(25, 1); // rotate the state in x direction
qc.roty(25, 1); // rotate the state in y direction
qc.phase(25, 1); // rotate the state in z direction
```

[2]Construction pending.

Now that our qubit is prepared, we need to generate our entangled pair of qubits, one which belongs to Alice and the other to Bob:

```
qc.codeLabel('entangle');
// entangle Alice's qubit with Bob's, like in
    Section 5
qc.hadamard(2);
qc.cnot(4, 2);
```

Next, Alice performs the entanglement operation on her two qubits in revers, takes a measurement of her two qubits and sends the result to Bob as two classical bits:

```
qc.codeLabel('send');
qc.cnot(2, 1); // reverse entanglement of Alice's
    qubits
qc.hadamard(1);
qc.read(1|2); // measurement of Alice's qubits
```

If we look at the Bloch sphere now, our third qubit might be the same as the original, but it probably is not. To be precise, the state will have been correctly teleported if our measured result is $|00\rangle$. So what do we do if that was not our measured result? Well, Bob applies correction operations to his qubit, based on what Alice's measurement result is:

- If Alice's result was $|00\rangle$, he does nothing
- If Alice's result was $|01\rangle$, he performs a Z operation, where $Z|0\rangle = |0\rangle$ and $Z|1\rangle = -|1\rangle$
- If Alice's result was $|10\rangle$, he performs an X operation
- If Alice's result was $|11\rangle$, he performs a Z operation followed by an X operation

```
qc.codeLabel('receive');
qc.cnot(4, 2); // apply corrective X gate
// next three lines apply corrective Z gate
qc.hadamard(4);
qc.cnot(4, 1);
qc.hadamard(4);
```

The second line in the above code applies an X gate to the receiver's qubit, conditioned on the sender's second qubit. The following three lines can be thought of as applying a Z gate, as it can be shown that a Hadamard gate, followed by an X gate, followed by another Hadamard gate, is equivalent to applying a Z gate.

Now if we look at the Bloch sphere image of the third qubit, it matches the first qubit during the state preparation stage. Thus we have successfully managed to teleport a single qubit.

Task 8.1. *Try modifying the state preparation code to get a different qubit. Does the teleportation still work successfully?*

Task 8.2. *Try removing one of the corrections in the receive code, either line 2 or lines 3-5, and run the teleportation protocol again on different states. Can you notice any patterns in the error of the state?*

Task 8.3. *(Harder) How might you teleport two qubits? Try writing a protocol in the QCEngine for it. Does the protocol still work if the qubits Alice wants to send are entangled?*