LAB 5

Name: Heja Bibani
Student Number: 16301173

In [1]:
```
import numpy as np
import random
from math import sqrt
from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister, execute, IBMQ
from qiskit.providers.aer import QasmSimulator
from qiskit.visualization import plot_histogram
from qiskit.extensions import Initialize
%matplotlib inline
```

Section 1

Suppose $q_1 = |1\rangle$ and $q_0 = |0\rangle$. A Bell Circuit is applied to these two qubits with q_0 as the control qubit in CNOT. What's the resulting state of these two qubits after the Bell Circuit? Please use Markdown cells to detail the calculation steps.

We let, $q_1 = |1\rangle$ and $q_0 = |0\rangle$

The total calculation will be of the following:

$$CNOT(q_1 \otimes H(q_0)) = CNOT\left(\begin{bmatrix}1\\0\end{bmatrix} \otimes \left(\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix}\right)\right)$$

Step 1: Apply a hadamard gate on q_0

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

Step 2: Tensor product of q_1 and q_0

$$= |1\rangle \otimes (\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle)$$
$$= \frac{1}{\sqrt{2}}|10\rangle + \frac{1}{\sqrt{2}}|11\rangle$$

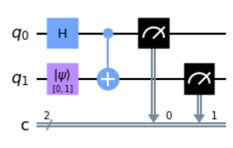
Step 3: CNOT Gate on tensor product in step 2:

$$= CNOT(\frac{1}{\sqrt{2}}|10\rangle + \frac{1}{\sqrt{2}}|11\rangle)$$
$$= \begin{bmatrix}1&0&0&0\\0&0&0&1\\0&0&1&0\\0&1&0&0\end{bmatrix} .\times \begin{bmatrix}0\\0\\\frac{1}{\sqrt{2}}\\\frac{1}{\sqrt{2}}\end{bmatrix}$$
$$= \begin{bmatrix}0\\\frac{1}{\sqrt{2}}\\\frac{1}{\sqrt{2}}\\0\end{bmatrix}$$
$$= \frac{1}{\sqrt{2}}|10\rangle + \frac{1}{\sqrt{2}}|01\rangle$$

In [16]:
```
import numpy as np
import random
from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister, execute, IBMQ
from qiskit.providers.aer import QasmSimulator
from qiskit.visualization import plot_histogram
from qiskit.extensions import Initialize
from math import sqrt
%matplotlib inline
```

In [3]:
```
# If the QuantumCircuit() function only has one parameter,
# it means the number of input qubits.
qc = QuantumCircuit(2,2)
initial_state = [0, 1]
qc.initialize(initial_state, 1)
#Apply h-gate on qubit 0.
qc.h(0)
# Its parameters indicate q0 is the Control Qubit, and q1 is the Target Qubit.
qc.cx(0,1)
# Store the measurement results of qubits 0 and 1 into bits 0 and 1 respectively
qc.measure([0,1], [0,1])
qc.draw()
```

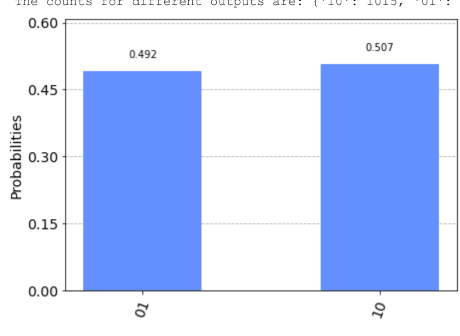
Out[3]:



In [4]:
```
simulator = QasmSimulator()
job = execute(qc, backend=simulator, shots=2000)
# Call result() to obtain the result
result = job.result()
# Call get_counts() to obtain the counts of different outputs
counts = result.get_counts(qc)

# Print the counts of different outputs
print("\n The counts for different outputs are:", counts)
# Plot the histogram
plot_histogram(counts)
```

```
The counts for different outputs are: {'10': 1015, '01': 985}
```

Out[4]:



We notice that the values in the simulator correspond to the output in the markdown cells.

Section 2

Suppose $q_1 = \frac{4}{5}|0\rangle - \frac{3}{5}|1\rangle$ and $q_0 = \frac{1}{3}|0\rangle + \frac{2\sqrt{2}}{3}|1\rangle$. A Reverse Bell Circuit is applied to these two qubits with q_0 as the control qubit in CNOT. What's the resulting state of these two qubits after the Reverse Bell Circuit? Please use Markdown cells to detail the calculation steps.

First the current state is in:

$q_1 = \frac{4}{5}|0\rangle - \frac{3}{5}|1\rangle$ and $q_0 = \frac{1}{3}|0\rangle + \frac{2\sqrt{2}}{3}|1\rangle$

The tensor product of the state:

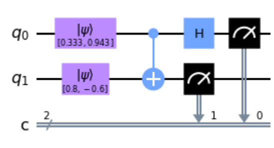
$$q_1 \otimes q_0 = (\frac{4}{5}|0\rangle - \frac{3}{5}|1\rangle) \otimes (\frac{1}{3}|0\rangle + \frac{2\sqrt{2}}{3}|1\rangle)$$
$$= \frac{4}{15}|00\rangle + \frac{8\sqrt{2}}{15}|01\rangle - \frac{3}{15}|10\rangle - \frac{6\sqrt{2}}{15}|11\rangle$$

Applying the reverse bell gate using tables in lecture slides:

$= \frac{4}{15}|00\rangle + \frac{8\sqrt{2}}{15}|01\rangle - \frac{3}{15}|10\rangle - \frac{6\sqrt{2}}{15}|11\rangle$
$= \frac{4}{15} \times (\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|01\rangle) + \frac{8\sqrt{2}}{15} \times (\frac{1}{\sqrt{2}}|10\rangle - \frac{1}{\sqrt{2}}|11\rangle) - \frac{3}{15} \times (\frac{1}{\sqrt{2}}|10\rangle + \frac{1}{\sqrt{2}}|11\rangle) - \frac{6\sqrt{2}}{15} \times (\frac{1}{\sqrt{2}}|00\rangle - \frac{1}{\sqrt{2}}|01\rangle)$
$= \frac{4}{15\sqrt{2}}|00\rangle + \frac{4}{15\sqrt{2}}|01\rangle + \frac{8\sqrt{2}}{15\sqrt{2}}|10\rangle - \frac{8\sqrt{2}}{15\sqrt{2}}|11\rangle - \frac{3}{15\sqrt{2}}|10\rangle - \frac{3}{15\sqrt{2}}|11\rangle - \frac{6\sqrt{2}}{15\sqrt{2}}|00\rangle + \frac{6\sqrt{2}}{15\sqrt{2}}|01\rangle$
$= \frac{4-6\sqrt{2}}{15\sqrt{2}}|00\rangle + \frac{4+6\sqrt{2}}{15\sqrt{2}}|01\rangle + \frac{8\sqrt{2}-3}{15\sqrt{2}}|10\rangle - \frac{8\sqrt{2}+3}{15\sqrt{2}}|11\rangle$

In [5]:
```
# If the QuantumCircuit() function only has one parameter,
# it means the number of input qubits.
qc1 = QuantumCircuit(2,2)
initial_state = [4/5, -3/5]
qc1.initialize(initial_state, 1)
initial_state = [1/3, (2*sqrt(2))/3]
qc1.initialize(initial_state, 0)
#Apply h-gate on qubit 0.
qc1.cx(0,1)
# Its parameters indicate q0 is the Control Qubit, and q1 is the Target Qubit.
qc1.h(0)
# Store the measurement results of qubits 0 and 1 into bits 0 and 1 respectively
qc1.measure([0,1], [0,1])
qc1.draw()
```

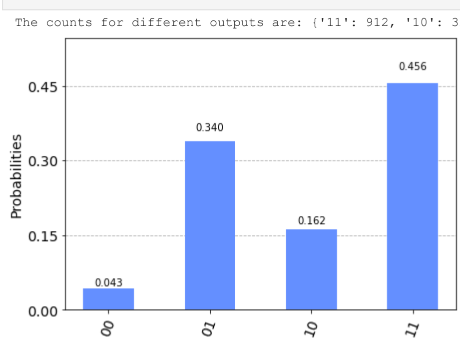
Out[5]:



In [6]:
```
simulator = QasmSimulator()
job = execute(qc1, backend=simulator, shots=2000)
# Call result() to obtain the result
result = job.result()
# Call get_counts() to obtain the counts of different outputs
counts = result.get_counts(qc1)

# Print the counts of different outputs
print("\n The counts for different outputs are:", counts)
# Plot the histogram
plot_histogram(counts)
```

```
The counts for different outputs are: {'11': 912, '10': 324, '01': 679, '00': 85}
```

Out[6]:



We notice that the probabilities are as expected from the calculation above. Since for 00 the expected probability is 0.0447, for 01 is 0.346, for 10 is 0.154, and for 11 is 0.4553.

Section 3

In the original Teleportation Protocol, a third party prepares a pair of entangled qubits with the state $\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$ and then sends Alice one qubit from the pair and sends Bob the other. In this task, you are asked to replace the entangled state $\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$ with $\frac{1}{\sqrt{2}}|00\rangle - \frac{1}{\sqrt{2}}|11\rangle$, and make the Teleportation still work.

Task 3.1

Give the protocol and the circuit for it first. You should detail what operations (e.g., gates or measurements) are taken in each step by both Markdown and Code cells. In the Markdown cells, manual calculation should be given to show what will happen in each step by assuming that the qubit to teleport has a general state $a|0\rangle + b|1\rangle$. In the Code cells, the circuit for this Teleportation Protocol should be constructed step by step with barriers introduced between steps.

Hints: Step 1 needs to be changed to produce $\frac{1}{\sqrt{2}}|00\rangle - \frac{1}{\sqrt{2}}|11\rangle$. Steps 2 and 3 don't need changes. Step 4 needs to apply X and/or Z gate differently from the original Teleportation protocol.

Step 1

A third party, Telamon, creates an entangled pair of qubits and gives one to Bob and another to Alice.

The pair Telamon creates is a special pair called a Bell pair. In quantum circuit, the way to create a Bell pair between two qubits is to apply the Bell Circuit consisting of H gate and CNOT gate. In the case that we are doing we have set the value for q_1 to be $|1\rangle$ and q_2 to default $|0\rangle$ thus producing the following state:

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

In [7]:
```
# For source code reuse, we create a function for this
def create_bell_pair(qc, ctrl, target):
    """
        qc: the circuit to work on
        ctrl: the index of the control qubit in CNOT
        target: the index of the target qubit in CNOT
    """
    qc.h(ctrl) # Put qubit ctrl into state |+>
    qc.cx(ctrl,target) # Apply CNOT gate
```

Let's say Alice owns q_1 and Bob owns q_2.

Step 2

Alice applies Reverse Bell Circuit to q_0 and q_1, with q_0 being the qubit to teleport to Bob.

In [8]:
```
def reverse_bell(qc, ctrl, target):
    """
        qc: the circuit to work on
        ctrl: the index of the control qubit in CNOT
        target: the index of the target qubit in CNOT
    """
    qc.cx(ctrl, target)
    qc.h(ctrl)
```

Step 3

Next, Alice measures the two qubits that she owns: q_0 and q_1, and stores the results into the two classical registers. She then sends these two bits to Bob.

In [9]:
```
def measure_and_send(qc, ctrl, target):
    # Measures qubits ctrl & target
    qc.measure(ctrl, cr[0])
    qc.measure(target, cr[1])
    # In simulation, 'sending' the results to Bob can be omitted.
    # We simply assume that Bob can access cr.
```

Step 4

In this step it is different from the lecture notes, and because of this a new set of steps is required to complete this task proficiently. The reason is because we are using the entangled qubit state:

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

This changes the way in which we need to tackle this problem. Therefore, below we have tried to identify how to determine which gates need to be applied by following the lecture notes:

Step 1: Tensor Product with qubit

$$= (\frac{1}{\sqrt{2}}|00\rangle - \frac{1}{\sqrt{2}}|11\rangle) \otimes (a|0\rangle + b|1\rangle)$$
$$= \frac{a}{\sqrt{2}}|000\rangle + \frac{b}{\sqrt{2}}|001\rangle - \frac{a}{\sqrt{2}}|110\rangle - \frac{b}{\sqrt{2}}|111\rangle$$

Step 2: Reverse Bell

$$= ReverseBell(\frac{a}{\sqrt{2}}|000\rangle + \frac{b}{\sqrt{2}}|001\rangle - \frac{a}{\sqrt{2}}|110\rangle - \frac{b}{\sqrt{2}}|111\rangle)$$
$$= ReverseBell(\frac{a}{\sqrt{2}}|0\rangle \otimes |00\rangle + \frac{b}{\sqrt{2}}|0\rangle \otimes |01\rangle - \frac{a}{\sqrt{2}}|1\rangle \otimes |10\rangle - \frac{b}{\sqrt{2}}|1\rangle \otimes |11\rangle)$$

We apply this on the former two qubits, consequently this equals:

$$= ReverseBell(\frac{a}{\sqrt{2}}|0\rangle \otimes |00\rangle + \frac{b}{\sqrt{2}}|0\rangle \otimes |01\rangle - \frac{a}{\sqrt{2}}|1\rangle \otimes |10\rangle - \frac{b}{\sqrt{2}}|1\rangle \otimes |11\rangle)$$
$$= \frac{a}{\sqrt{2}}|0\rangle \otimes (\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|01\rangle) + \frac{b}{\sqrt{2}}|0\rangle \otimes (\frac{1}{\sqrt{2}}|10\rangle - \frac{1}{\sqrt{2}}|11\rangle) - \frac{a}{\sqrt{2}}|1\rangle \otimes (\frac{1}{\sqrt{2}}|10\rangle + \frac{1}{\sqrt{2}}|11\rangle) - \frac{b}{\sqrt{2}}|1\rangle \otimes (\frac{1}{\sqrt{2}}|00\rangle - \frac{1}{\sqrt{2}}|01\rangle)$$
$$= \frac{a}{2}|0\rangle \otimes |00\rangle + \frac{a}{2}|0\rangle \otimes |01\rangle + \frac{b}{2}|0\rangle \otimes |10\rangle - \frac{b}{2}|0\rangle \otimes |11\rangle - \frac{a}{2}|1\rangle \otimes |10\rangle - \frac{a}{2}|1\rangle \otimes |11\rangle - \frac{b}{2}|1\rangle \otimes |00\rangle + \frac{b}{2}|1\rangle \otimes |01\rangle$$
$$= \frac{1}{2}(a|0\rangle - b|1\rangle) \otimes |00\rangle + \frac{1}{2}(a|0\rangle + b|1\rangle) \otimes |01\rangle + \frac{1}{2}(b|0\rangle - a|1\rangle) \otimes |10\rangle + \frac{1}{2}(-b|0\rangle - a|1\rangle) \otimes |11\rangle$$

Bob, who already has the qubit q_2, applies the following gates depending on the state of the classical bits:

00 $\rightarrow a|0\rangle - b|1\rangle \rightarrow$ Apply Z gate

01 $\rightarrow a|0\rangle + b|1\rangle \rightarrow$ Do Nothing

10 $\rightarrow b|0\rangle - a|1\rangle \rightarrow$Apply Z gate then X gate

11 $\rightarrow -b|0\rangle - a|1\rangle \rightarrow$Apply Z gate first then X gate second and then the Z gate again.

In the bottom below we implemented a sufficient gate which allows to solve this problem. This is more complicated than the initial problem and so required a different approach. Because of this, we applied the gates according to the total value of both the registers.

In [10]:
```
def bob_gates(qc, qubit, cr):
    """
        qc: the circuit to work on
        qubit: the index of the qubit to apply X or Z gate
        crz: the crz register
        crx: the crx register
    """
    # Here we use c_if() to control our gates by the cr register.
    # Apply the gates if cr has that decimal value.
    qc.z(qubit).c_if(cr, 0)
    qc.z(qubit).c_if(cr, 2)
    qc.x(qubit).c_if(cr, 2)
    qc.z(qubit).c_if(cr, 3)
    qc.x(qubit).c_if(cr, 3)
    qc.z(qubit).c_if(cr, 3)
```

In this notebook, we will initialize Alice's q_0 to a random state $|\psi\rangle$ (named as `psi`). This random state is created as follows.

Next, we need to create a gate that initializes a qubit to the state of $|\psi\rangle$. Note that:

Here we need a gate, because we need to get an inverse gate for this gate later. The Initialize used below is a class from the qiskit.extensions module, while the initialize() you saw in previous notebooks is a function of the QuantumCircuit class. They have different names, since the names in Python is case-sensitive.

In [11]:
```
# Create the two random amplitudes of psi
p0 = random.uniform(0,1)
p1 = np.sqrt(1 - p0*p0)
psi = [p0, p1]
# Display psi
psi
# Create a gate that initializes a qubit to the state of psi
init_gate = Initialize(psi)
init_gate.label = "Init"
```

The `Initialize` class first performs a reset, setting our qubit to the state $|0\rangle$. It then applies gates to turn $|0\rangle$ into the state $|\psi\rangle$:

$$|0\rangle \xrightarrow{\text{Initialize gates}} |\psi\rangle$$

Since all quantum gates are reversible, we can find the inverse of these gates using:

In [12]:
```
inverse_init_gate = init_gate.gates_to_uncompute()
```

This operation has the property:

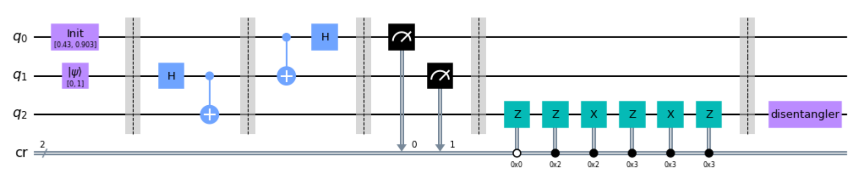
$$|\psi\rangle \xrightarrow{\text{Inverse Initialize gates}} |0\rangle$$

To prove the qubit $|q_0\rangle$ has been teleported to $|q_2\rangle$, we can do this inverse operation on $|q_2\rangle$. If the measurement result is $|0\rangle$ with certainty, then the teleportation protocol is verified.

In the following circuit we have initialized q_1 to be in the state $|1\rangle$.

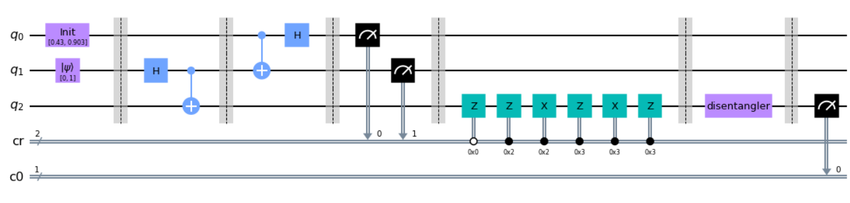
In [13]:
```
## SETUP
qr = QuantumRegister(3, name="q")
cr = ClassicalRegister(2, name="cr")
qc = QuantumCircuit(qr, cr)
initial_state = [0, 1]
qc.initialize(initial_state, 1)
## STEP 0
# First, let's initialize Alice's q0
qc.append(init_gate, [0])
qc.barrier()

## STEP 1
# Now begins the teleportation protocol
create_bell_pair(qc, 1, 2)
qc.barrier()

## STEP 2
# Send q1 to Alice and q2 to Bob
reverse_bell(qc, 0, 1)
qc.barrier()

## STEP 3
# Alice then sends her classical bits to Bob
measure_and_send(qc, 0, 1)
qc.barrier()

## STEP 4
# Bob decodes qubits
bob_gates(qc, 2, cr)
qc.barrier()

## STEP 5
# reverse the initialization process
qc.append(inverse_init_gate, [2])

# Display the circuit
qc.draw()
```

Out[13]:



In [14]:
```
qc.barrier()
# Need to add a new ClassicalRegister to see the result
cr_result = ClassicalRegister(1)
qc.add_register(cr_result)
# Measure q_2 and store it in ClassicalRegister 2, which is cr_result
qc.measure(2,2)
qc.draw()
```

Out[14]:

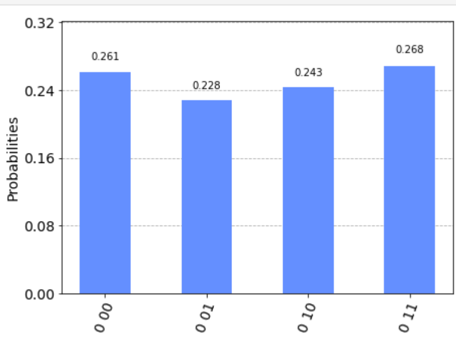


Task 3.2

Then, verify the above circuit by running it under the QasmSimulator, with randomness introduced to the qubit to teleport. The verification technique should be the same as the one used in the notebook NB05

In [15]:
```
backend = QasmSimulator()
counts = execute(qc, backend, shots=1000).result().get_counts()
plot_histogram(counts)
```

Out[15]:



We can see that there is a 100% chance of measuring q_2 (the leftmost bit in the string) in the state $|0\rangle$. This is the expected result, and indicates the teleportation protocol has worked correctly.