Assignment

Name: Heja Bibani
Student Number: 16301173

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

Section 1 A General Proof to the Non-Cloning Theorem

In Lecture 4, we proved the No-Cloning Theorem by using $|0\rangle$ as the ancilla qubit. In this task, you are asked to prove this theorem by using a general qubit $a|0\rangle + b|1\rangle$ as the ancilla qubit. That is, no matter which state the ancilla qubit assumes, cloning is impossible.

There are three equations that we need to consider:

Equation 1:

$G((a|0\rangle + b|1\rangle) \otimes |0\rangle) = |0\rangle \otimes |0\rangle = |00\rangle$

Equation 2:

$G((a|0\rangle + b|1\rangle) \otimes |1\rangle) = |1\rangle \otimes |1\rangle = |11\rangle$

Equation 3:

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

We can also derive equation 3 as follows:

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

Since matrix multiplication satisfies distributivity:

$$G(\frac{1}{\sqrt{2}}((a|0\rangle + b|1\rangle) \otimes |0\rangle) + \frac{1}{\sqrt{2}}((a|0\rangle + b|1\rangle) \otimes |1\rangle)) = \frac{1}{\sqrt{2}}(G((a|0\rangle + b|1\rangle) \otimes |0\rangle)) + \frac{1}{\sqrt{2}}(G((a|0\rangle + b|1\rangle) \otimes |1\rangle))$$

From equation 1 and 2 we know that:

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

The results form a different equation from equation 3 above. Thus demonstrating the contradiction.

Section 2 An alternative Teleportation Protocol

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 $\frac{1}{\sqrt{2}}|01\rangle + \frac{1}{\sqrt{2}}|10\rangle$, and make the Teleportation still work.

Task 2.1

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.

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 the default $|0\rangle$ and q_2 to $|1\rangle$ thus producing the following state:

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

In [2]:
```python
# 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 [3]:
```python
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 [4]:
```python
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:

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

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

$$= (\frac{1}{\sqrt{2}}|01\rangle + \frac{1}{\sqrt{2}}|10\rangle) \otimes (a|0\rangle + b|1\rangle)$$
$$= \frac{a}{\sqrt{2}}|010\rangle + \frac{b}{\sqrt{2}}|011\rangle + \frac{a}{\sqrt{2}}|100\rangle + \frac{b}{\sqrt{2}}|101\rangle$$

Step 2: Reverse Bell

$$= ReverseBell(\frac{a}{\sqrt{2}}|010\rangle + \frac{b}{\sqrt{2}}|011\rangle + \frac{a}{\sqrt{2}}|100\rangle + \frac{b}{\sqrt{2}}|101\rangle)$$
$$= ReverseBell(\frac{a}{\sqrt{2}}|0\rangle \otimes |10\rangle + \frac{b}{\sqrt{2}}|0\rangle \otimes |11\rangle + \frac{a}{\sqrt{2}}|1\rangle \otimes |00\rangle + \frac{b}{\sqrt{2}}|1\rangle \otimes |01\rangle)$$

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

[illegible]

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

$00 \rightarrow b|0\rangle + a|1\rangle \rightarrow$ Apply X gate

$01 \rightarrow -b|0\rangle + a|1\rangle \rightarrow$ Apply X and then Z

$10 \rightarrow a|0\rangle + b|1\rangle \rightarrow$ Do nothing

$11 \rightarrow a|0\rangle - b|1\rangle \rightarrow$ Apply Z gate.

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 [5]:
```python
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.
    [illegible]
```

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 functions are different.

In [7]:
```python
# 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 [8]:
```python
inverse_init_gate = init_gate.gates_to_uncompute()
```

This operation has the property:

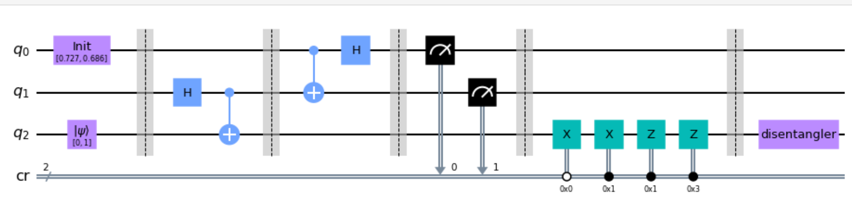
$$|\psi\rangle \xrightarrow{\text{Inverse Initialize gates}} |0\rangle$$

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_2 to be in the state $|1\rangle$.

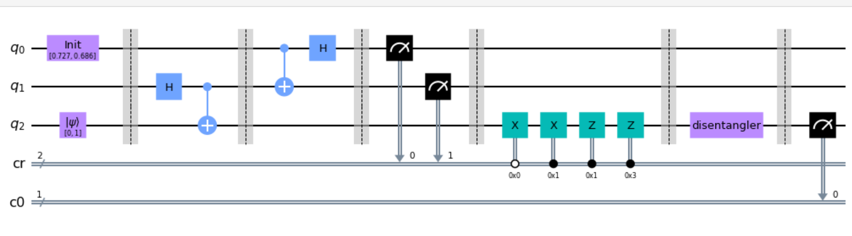
In [9]:
```python
# SETUP
[illegible]

## 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[9]:



In [10]:
```python
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[10]:

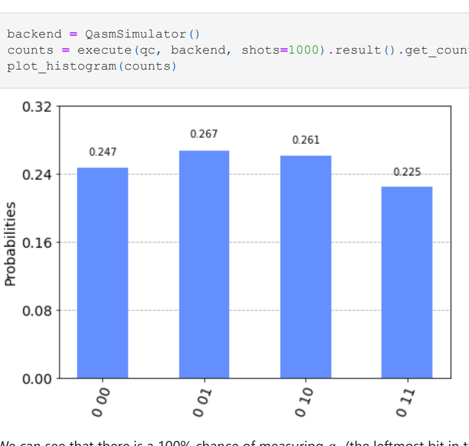


Task 2.2

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 [11]:
```python
backend = QasmSimulator()
counts = execute(qc, backend, shots=1000).result().get_counts()
plot_histogram(counts)
```

Out[11]:



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.

Section 3 Proof of Probability in Ekert Protocol

In the variant of the Ekert Protocol taught in our Lecture 7, if Eve eavesdrops every pair of entangled qubits, then for those entangled pairs where Alice and Bob pick different bases, the probability for them to see the same measurement outcome is $\frac{3}{8}$. In this task, you are asked to prove this fact.

Note: We have added more scenarios where eve measures first and second, the first case is when eve measures second. The second case is after this one where eve measures first.

We are going to pick the two bases as recommended for Alice and Bob:

$\theta = \frac{2\pi}{3}, \theta = \frac{4\pi}{3}$

Whilst Eve picks:

$\theta = 0, \frac{2\pi}{3}, \frac{4\pi}{3}$

This is the case where Eve measures Second:

Case 1: $\theta = 0$

Alice measures first and gets either the following results:

[illegible]

We will test for both, firstly:

[illegible]

We get the dot product of the following:

[illegible]

There are two cases since Eve can get $|0\rangle$ with probability $\frac{1}{4}$ and $|1\rangle$ with probability $\frac{3}{4}$. If She gets state $|0\rangle$ the following would be needed to be calculated in the $\theta = \frac{2\pi}{3}$ basis of Bob.

[illegible]

We get the dot product of the following:

[illegible]

These are two cases similar to the previous, however, if we want both of them to get the same value, then Bob must get the value of $|\searrow\rangle$ with probability $\frac{1}{4}$. Thus, there is a $\frac{1}{4} * \frac{1}{4}$ to get the same value correct. If She gets state $|1\rangle$, it will be with the probability $\frac{3}{4}$ and the following would be needed to be calculated in the $\theta = \frac{4\pi}{3}$ basis of Bob.

[illegible]

We get the dot product of the following:

[illegible]

To get zero just as also this would be with the probability of $\frac{3}{4}$, thus there would be a $\frac{3}{4} * \frac{3}{4}$ probability to get 0. This is the same case for the other state $|\nwarrow\rangle$. The total probability would be [illegible]

Case 2: $\theta = \frac{2\pi}{3}$

Alice measures first and gets either the following results:

[illegible]

We will test for both, firstly:

[illegible]

In both cases it will be a 100% chance of getting any state since the base is the same. Thus the focus of the calculation would be on the third measurement made by Bob.

[illegible]

We get the dot products for the case:

[illegible]

There is thus a $\frac{1}{4}$ chance of it being the same.

If on the other case, the Alice chooses the other measurement:

[illegible]

Since there is a 100% chance of getting the same direction because they have picked the same base, then for Bob's measurement we do the following:

[illegible]

We get the dot products for the case:

[illegible]

Thus there is a $\frac{1}{4}$ chance of getting it correct

Case 3: $\theta = \frac{4\pi}{3}$

Alice measures first and gets either the following results:

[illegible]

This means there is a $\frac{1}{4}$ of getting the same bit.

Similarly if on the other hand eve gets the following state, and Alice is to measure, the other case would be:

$|\nearrow\rangle = c|\searrow\rangle + d|\nwarrow\rangle$

We get the dot products for the case:

$$c = \langle\searrow|\nearrow\rangle = \begin{bmatrix} \frac{1}{2} \\ \frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} -\frac{\sqrt{3}}{2} \\ -\frac{1}{2} \end{bmatrix} = -\frac{\sqrt{3}}{2}$$

$$d = \langle\nwarrow|\nearrow\rangle = \begin{bmatrix} -\frac{\sqrt{3}}{2} \\ \frac{1}{2} \end{bmatrix} \begin{bmatrix} -\frac{\sqrt{3}}{2} \\ -\frac{1}{2} \end{bmatrix} = \frac{1}{2}$$

The chance of getting this correct is $\frac{1}{4}$, since the next base is the same then the probability would be 100% of the current result. Since the measurement that will be calcualted by Bob would be in the following state, this is because Bob's qubit is in the same state as Eve as the entanglement is broken.:

$|\nearrow\rangle = c|\swarrow\rangle + d|\nearrow\rangle$

Doing the calculation manually we get the following result:

$$c = \langle\swarrow|\nearrow\rangle = \begin{bmatrix} -\frac{1}{2} \\ \frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} -\frac{\sqrt{3}}{2} \\ -\frac{1}{2} \end{bmatrix} = 0$$

$$d = \langle\nearrow|\nearrow\rangle = \begin{bmatrix} -\frac{\sqrt{3}}{2} \\ -\frac{1}{2} \end{bmatrix} \begin{bmatrix} -\frac{\sqrt{3}}{2} \\ -\frac{1}{2} \end{bmatrix} = 1$$

To calculate the total percentage of the values we need to look at all configurations:

V-S-N $\rightarrow \frac{5}{8} * \frac{1}{3} = \frac{5}{24}$

S-S-N $\rightarrow \frac{1}{4} * \frac{1}{3} = \frac{2}{24}$

N-S-N $\rightarrow \frac{1}{4} * \frac{1}{3} = \frac{2}{24}$

Thus, the probability equals $\frac{3}{8}$.

The calculation for these sections are a bit obscure, and it takes a lot of investigation to confirm that there indeed exists a variation in the probabilities for each of the circuit configurations. Below is the demonstration of this, we will recreate each circuit and show that the probabilities are the same as those demonstrated in the previous section.

Below is the recreation of these circuits. Instead of choosing them randomly, each of the circuits are of the same type as the configurations for each of the cases. So for all three cases each of the cirucits contained in the arrays should be of the same configuration.

Case 1: V-S-N $\rightarrow \frac{5}{8}$

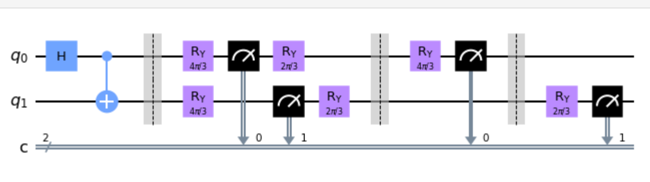
In [12]:
```python
from qiskit import QuantumCircuit, execute, Aer
from qiskit.visualization import plot_histogram
from numpy.random import randint
import numpy as np
import math
%matplotlib inline

n = 300
# Obtain a simulator instance
backend = Aer.get_backend('qasm_simulator')

""" Setting up """
n = 300

# For storing the n circuits generated
circuit_list = []

for i in range(n):
    qc = QuantumCircuit(2,2)
    # Entangle qubits with Bell Circuit
    qc.h(0)
    qc.cnot(0,1)
    qc.barrier()
    qc.measure(1,1)
    qc.barrier()
    qc.ry(math.pi*4/3, 0)
    qc.measure(0, 0)
    qc.barrier()
    qc.ry(math.pi*2/3, 1)
    qc.measure(1, 1)
    qc.draw()
    circuit_list.append(qc)
```

```python
count = 0
for i in range(length):
    if alice_badbits[i] == bob_badbits[i]:
        count += 1
print("Same bit proportion:", count/length)
```

Same bit proportion: 0.2347417840375587

In [17]:
```python
circuit_list[2].draw()
```

Out[17]:



Case 3: N-S-N $\rightarrow \frac{1}{4}$

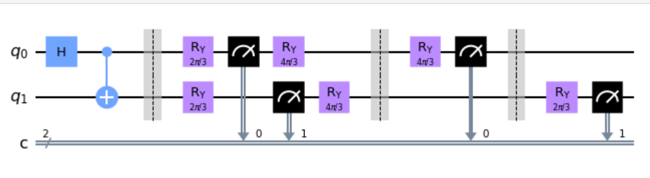
In [18]:
```python
""" Setting up """
n = 300

# For storing the n circuits generated
circuit_list = []

for i in range(n):
    qc = QuantumCircuit(2,2)
    # Entangle qubits with Bell Circuit
    qc.h(0)
    qc.cnot(0,1)
    qc.barrier()
    qc.ry(math.pi*2/3, [0, 1])
    qc.measure([0, 1], [0, 1])
    # Restoration
    qc.ry(math.pi*4/3, [0, 1])
    qc.barrier()
    qc.ry(math.pi*4/3, 0)
    qc.measure(0, 0)
    qc.barrier()
    qc.ry(math.pi*2/3, 1)
    qc.measure(1, 1)
    qc.draw()
    circuit_list.append(qc)
```

In [19]:
```python
""" Step 2 """
# Eve generates random bases in [0, 1, 2]
eve_bases = randint(3, size=n)
# Alice generates random bases in [0, 1, 2]
alice_bases = randint(3, size=n)
# Bob generates random bases in [0, 1, 2]
bob_bases = randint(3, size=n)

# For storing the measurement result
alice_bits = []
bob_bits = []

# Measure each qubit pair
for i in range(n):
    result = execute(circuit_list[i], backend, shots=1, memory=True).result()
    # Alice and Bob extract their measurement results
    bit0 = int(result.get_memory()[0][0])
    alice_bits.append(bit0)
    bit1 = int(result.get_memory()[0][1])
    bob_bits.append(bit1)

""" Step 3 """
# Extracting good bits and bad bits
alice_goodbits = []
alice_badbits = []
bob_goodbits = []
bob_badbits = []
for q in range(n):
    if alice_bases[q] == bob_bases[q]:
        # Add to the list of 'good' bits
        alice_goodbits.append(alice_bits[q])
        bob_goodbits.append(bob_bits[q])
    else:
        alice_badbits.append(alice_bits[q])
        bob_badbits.append(bob_bits[q])

""" Step 4 """
# Calculating the same bit proportion of bad bits
length = len(alice_badbits)
count = 0
for i in range(length):
    if alice_badbits[i] == bob_badbits[i]:
        count += 1
print("Same bit proportion:", count/length)
```

Same bit proportion: 0.23232323232323232

In [20]:
```python
circuit_list[2].draw()
```

Out[20]:



We have shown in the above that all the probabilities are expected and there is indeed a variation between each circuit and this demonstrates that the calculations are accurate and according to the expected values:

The total percentage of the values of all configurations have been demonstrated by experiment to be:

V-S-N $\rightarrow \frac{5}{8} * \frac{1}{3} = \frac{5}{24}$

S-S-N $\rightarrow \frac{1}{4} * \frac{1}{3} = \frac{2}{24}$

N-S-N $\rightarrow \frac{1}{4} * \frac{1}{3} = \frac{2}{24}$

Thus, the probability equals $\frac{3}{8}$

Section 4 A Circuit for Grover's Algorithm

Suppose $f(x_3, x_2, x_1, x_0)$ is 4-Boolean-variable function. It outputs 1 only when the input is $x_3 = 1, x_2 = 1, x_1 = 0, and x_0 = 0$, otherwise it outputs 0. In this task, you should implement a Grover's circuit that reveals which input will cause $f(x_3, x_2, x_1, x_0)$ to output 1.

Task 4.1

Roughly follow the notebook accompanying Lecture 10 to implement this circuit and conduct measurement. Run this circuit for at least 1000 shots and plot the result.

This time we'll use the above function as the oracle. As derived in the Lecture 10, the oracle can be achieved by an X-gate on, an MCT gate, and an X-gate again. The $2A - I$ can be implemented by $H^{\otimes 4} \times X^{\otimes 4} \times CCZ \times X^{\otimes 4} \times H^{\otimes 4}$.

Moreover, we need to do the amplitude amplification twice to obtain a high-probability outcome when $n = 4$. The number of amplitude amplifications to perform (denoted by k) is calculated by the following formula:

$$k = \text{floor}\left(\frac{\pi}{4}\sqrt{2^n}\right)$$

Thus k =3, we can construct the circuit as follows.

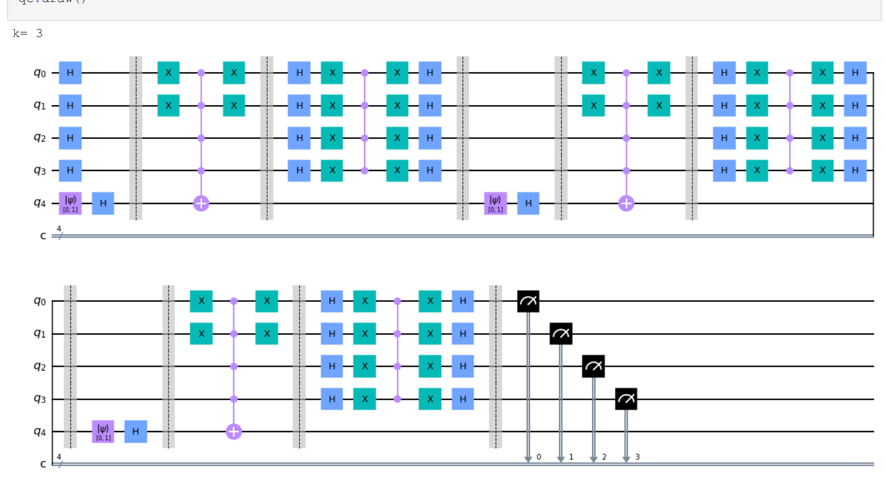
In the bottom to get the desired oracle, we must implement an X-gate on qubits 0 and 1. And then another X-gate to restore its previous values. This should allow for the value to come out on '1100' as expected.

In [27]:
```python
from qiskit import *
from qiskit.providers.aer import QasmSimulator
from qiskit.visualization import plot_histogram
%matplotlib inline
```

In [41]:
```python
# Needed by the construction of the CCZ gate
from qiskit.circuit.library import MCMT
import math

# Construct a circuit with a 5-qubit input, and 4-bit measurement storage.
qc = QuantumCircuit(5, 4)

# Apply H-gates to q0, q1, q2 and q3
qc.h([0, 1, 2, 3])

# Calculate the number of amplitude amplifications
k = math.floor(math.pi*math.sqrt(2**4)/4)
print("k=", k)

# Repeat k times
for i in range(k):
    # Initialise q3 with |1>
    initial_state = [0, 1]
    qc.initialize(initial_state, 4)
    # Apply H-gate to q4
    qc.h(4)

    # Insert the oracle
    # And add two barriers to enclose this oracle.
    qc.barrier()
    qc.x(0)
    qc.x(1)
    qc.mct([0, 1, 2, 3], 4)
    qc.x(0)
    qc.x(1)
    qc.barrier()

    # Apply the gates for 2A-I to q0, q1, q2 and q3
    # Apply H and X gates first
    qc.h([0, 1, 2, 3])
    qc.x([0, 1, 2, 3])
    # Realize the CCZ gate via MCMT circuit
    qc = qc.compose(MCMT('z', 3, 1))
    # Apply X and H gates again
    qc.x([0, 1, 2, 3])
    qc.h([0, 1, 2, 3])

    qc.barrier()

# Measure q0, q1, q2, q3
qc.measure([0,1,2,3], [0,1,2,3])
# Draw the circuit
qc.draw()
```

k= 3

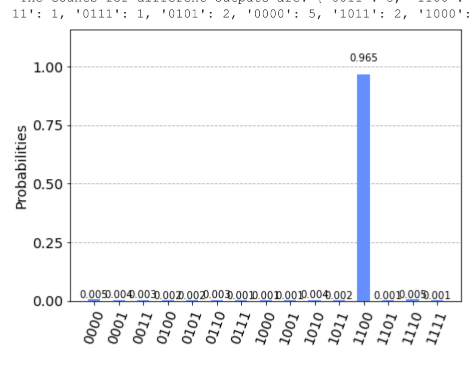
Out[41]:



In [40]:
```python
simulator = QasmSimulator()
# Run the circuit with certain number of shots.
# The 'shots' below means the number of times to run this circuit.
# Its default value is 1024, so if it is omitted, you'll run the circuit 1024 times.
job = execute(qc, backend=simulator, shots=1000)

# 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: {'0011': 3, '1100': 965, '1001': 1, '0001': 4, '1010': 4, '0100': 2, '1111': 1, '0111': 1, '0101': 2, '0000': 5, '1011': 2, '1000': 1, '1110': 5, '0110': 3, '1101': 1}
```

Out[40]:



Task 4.2

Manually calculate the probability for the measurement outcome being '1100' given the above circuit. The probability obtained should roughly match the result from 4.1.

It can be derived that, given the function described in the question, after flipping the sign for the first time, the joint state vector of $|x_3\rangle$, $|x_2\rangle$, $|x_1\rangle$, $|x_0\rangle$ is:

$\frac{1}{4}(|0000\rangle + |0001\rangle + |0010\rangle + |0011\rangle + |0100\rangle + |0101\rangle + |0110\rangle + |0111\rangle + |1000\rangle + |1001\rangle + |1010\rangle + |1011\rangle - |1100\rangle + |1101\rangle + |1110\rangle + |1111\rangle)$

Let's apply flipping about mean to this state vector.

Calculating the mean 'a' first: $a = \frac{1}{16} \times \frac{1}{4} \times (15 - 1) = \frac{7}{32}$

Flipping by the formula $2a - v$: $\frac{1}{4}$ becomes $\frac{3}{16}$; $-\frac{1}{4}$ becomes $\frac{11}{16}$

Measuring now, we'll get $|1100\rangle$ with probability $(\frac{11}{16})^2 = 0.4727$. This is not ideal, so let's go through flipping sign and flipping about mean for a second time.

With flipping the sign, we'll only flip the sign for $|1100\rangle$, thus the state vector becomes:

$\frac{1}{16}(3|0000\rangle + 3|0001\rangle + 3|0010\rangle + 3|0011\rangle + 3|0100\rangle + 3|0101\rangle + 3|0110\rangle + 3|0111\rangle + 3|1000\rangle + 3|1001\rangle + 3|1010\rangle + 3|1011\rangle - 11|1100\rangle + 3|1101\rangle + 3|1110\rangle + 3|1111\rangle)$

Let's apply flipping about mean to this state vector.

Calculating the mean 'a' first: $a = \frac{1}{16} \times \frac{1}{16} \times (45 - 11) = \frac{17}{128}$

Flipping by the formula $2a - v$: $\frac{3}{16}$ becomes $\frac{5}{64}$; $-\frac{11}{16}$ becomes $\frac{61}{64}$.

$\frac{1}{64}(5|0000\rangle + 5|0001\rangle + 5|0010\rangle + 5|0011\rangle + 5|0100\rangle + 5|0101\rangle + 5|0110\rangle + 5|0111\rangle + 5|1000\rangle + 5|1001\rangle + 5|1010\rangle + 5|1011\rangle - 61|1100\rangle + 5|1101\rangle + 5|1110\rangle + 5|1111\rangle)$

Measuring now, we'll get $|1100\rangle$ with probability $(\frac{61}{64})^2 = 0.908$. This is not adequate for our results since we need to go through the flip one more time according to the value of k calculated earlier.

Calculating the mean 'a' first: $a = \frac{1}{16} \times \frac{1}{64} \times (75 - 61) = \frac{7}{512}$

Flipping by the formula $2a - v$: $\frac{5}{64}$ becomes $-\frac{13}{256}$; $-\frac{61}{64}$ becomes $\frac{251}{256}$.

Calculating the probability for state $|1100\rangle$ we must square this number.

$(\frac{251}{256})^2 = 0.9613$.

This matches the results obtained above.