Practical 12

Name: Heja Bibani
Student Number: 16301173

In [2]:
```
from qiskit import *
from qiskit.providers.aer import QasmSimulator
from qiskit.visualization import plot_histogram
from math import sqrt
%matplotlib inline
```

Section 1

Alice needs to send a qubit to Bob. Suppose the quantum channel between Alice and Bob only suffers from the quantum bit-flip error (i.e., $a|0\rangle + b|1\rangle$ becoming $a|1\rangle + b|0\rangle$), and the probability for this error is 0.1 for any qubit. Alice and Bob use the Quantum Bit-Flip Code as the quantum error correction technique. Then, what is the probability that Bob will obtain the qubit from Alice correctly?

Bob will obtain the qubit correctly when there is only one error and if there is no error. This is the combination of the following probabilities.

$$(\binom{3}{0}(0.1)^0 \times (0.9)^3) + (\binom{3}{1}(0.1)^1 \times (0.9)^2) = 0.729 + 0.243 = 0.972$$

Section 2

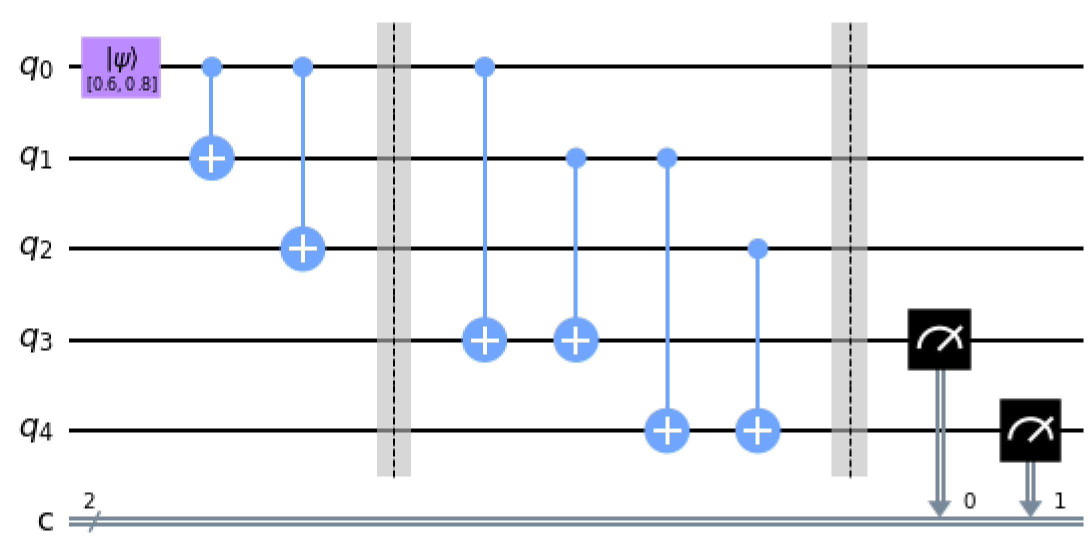
Implement a variant of the Quantum Bit-Flip Code. This variant differs from the one presented in the lecture only in the following way: checking $q_1 = q_0$ and $q_1 = q_2$ instead of $q_0 = q_1$ and $q_0 = q_2$.

2.1

2.1 Suppose the qubit to send is $\frac{3}{5}|0\rangle + \frac{4}{5}|1\rangle$. Give the circuit implementations for this variant at the both the Alice and Bob sides. The circuits should roughly follow the structures given in the notebook accompanying Lecture 12.

In [3]:
```
# Construct a circuit with a 5-qubit input and a 2-bit measurement storage.
qc = QuantumCircuit(5, 2)

""" Generate the three qubits sent from Alice """
# Initialise q0 with 3/5|0> + 4/5|1>
initial_state = [3/5, 4/5]
qc.initialize(initial_state, 0)

# Apply CNOT gate to q0, q1 with q0 as control
qc.cx(0, 1)
# Apply CNOT gate to q0, q2 with q0 as control
qc.cx(0, 2)

# Introduce a barrier
qc.barrier()

""" Test if q0 equals q1; Store result in q3 """
# Apply CNOT gate to q0, q3 with q0 as control
qc.cx(0, 3)
# Apply CNOT gate to q1, q3 with q1 as control
qc.cx(1, 3)

""" Test if q0 equals q2; Store result in q4 """
# Apply CNOT gate to q0, q4 with q0 as control
qc.cx(1, 4)
# Apply CNOT gate to q2, q4 with q2 as control
qc.cx(2, 4)

# Introduce a barrier
qc.barrier()

# Measure q3, q4
qc.measure([3, 4], [0, 1])

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

Out[3]:



2.2

Given the above circuit implementations, describe how Bob should recover the error based on the measurement results of q_4 and q_3.

In this scenario we are changing the values which need to be tested. Since the xor function is altered we must utilize a different mechanism to correct the error. In this variant we are checking $q_1 = q_0$ and $q_1 = q_2$.

if q_4 = 0 and q_3 = 0 then we do nothing since nothing has changed.

if q_4 = 0 and q_3 = 1 then q_0 must be flipped.

if q_4 = 1 and q_3 = 0 then q_2 must be flipped.

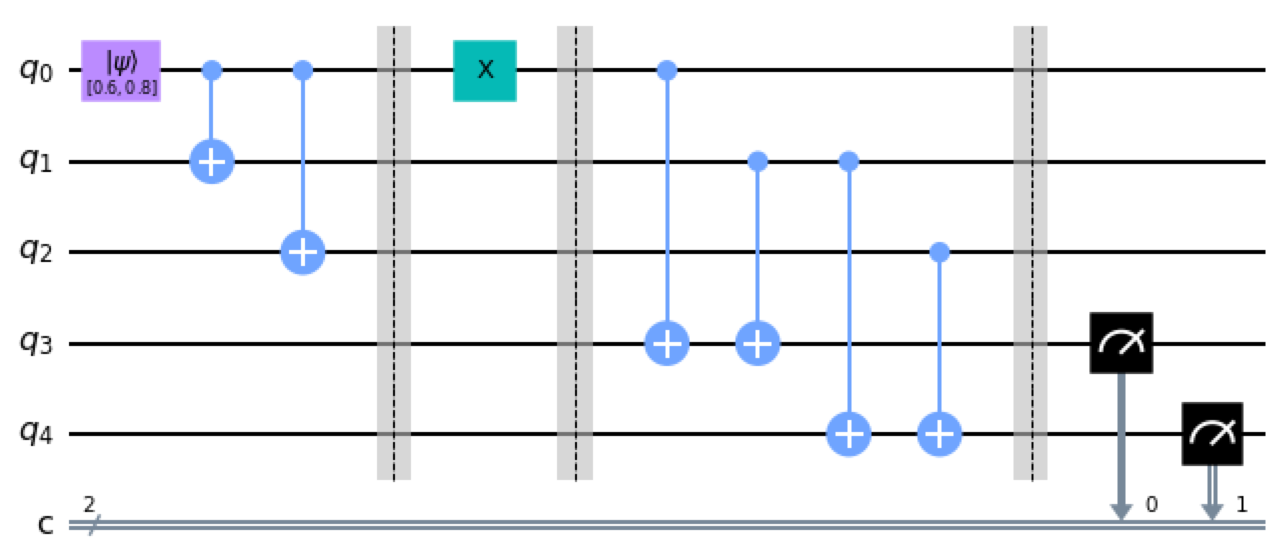
if q_4 = 1 and q_3 = 1 then q_1 must be flipped.

2.3

Suppose a bit-flip error happens to q_0 during the transmission, and Bob successfully recovers this error using the variant implemented above. Construct a circuit to simulate this situation.

In [4]:
```
# Construct a circuit with a 5-qubit input and a 2-bit measurement storage.
qc = QuantumCircuit(5, 2)

""" Generate the three qubits sent from Alice """
# Initialise q0 with 3/5|0> + 4/5|1>
initial_state = [3/5, 4/5]
qc.initialize(initial_state, 0)

# Apply CNOT gate to q0, q1 with q0 as control
qc.cx(0, 1)
# Apply CNOT gate to q0, q2 with q0 as control
qc.cx(0, 2)

# Introduce a barrier
qc.barrier()

""" Simulate a transmission error """
from numpy.random import randint
qc.x(0)
# Introduce a barrier
qc.barrier()

""" Test if q0 equals q1; Store result in q3 """
# Apply CNOT gate to q0, q3 with q0 as control
qc.cx(0, 3)
# Apply CNOT gate to q1, q3 with q1 as control
qc.cx(1, 3)

""" Test if q0 equals q2; Store result in q4 """
# Apply CNOT gate to q0, q4 with q0 as control
qc.cx(1, 4)
# Apply CNOT gate to q2, q4 with q2 as control
qc.cx(2, 4)

# Introduce a barrier
qc.barrier()

# Measure q3, q4
qc.measure([3, 4], [0, 1])

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

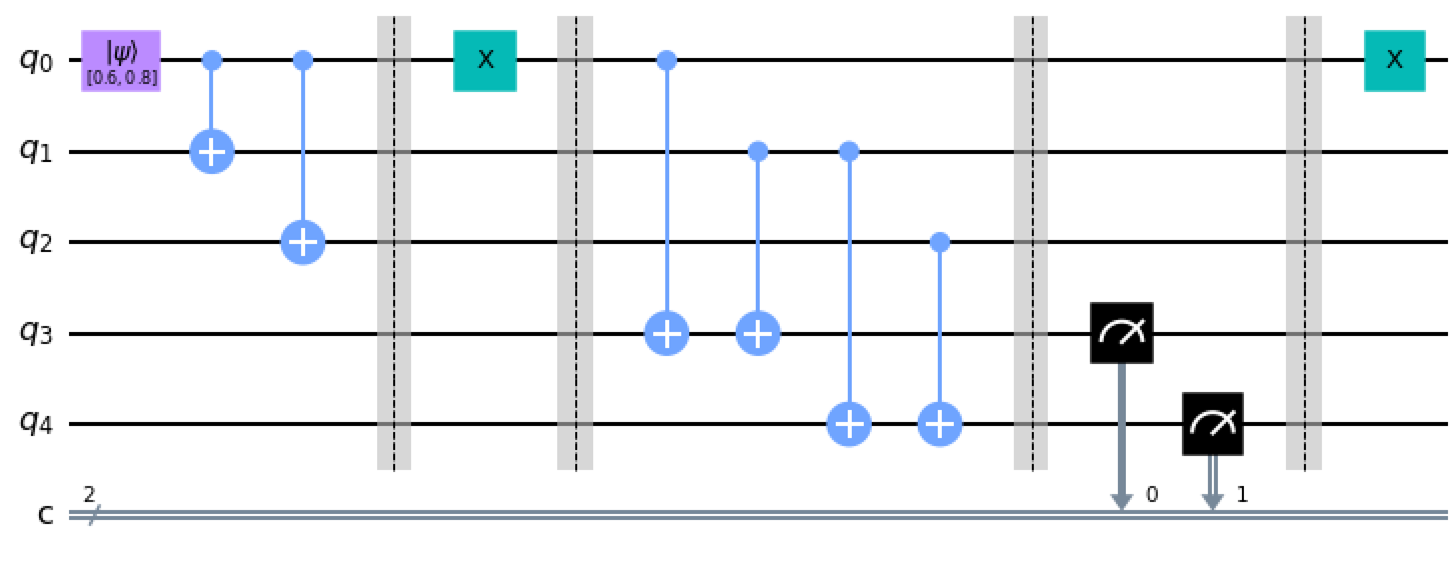
Out[4]:



In [5]:
```
simulator = QasmSimulator()

# Run the circuit with 1 shot only.
# set 'memory=True' so we can retrieve the measurement outcome
job = execute(qc, backend=simulator, shots=1, memory=True)
# The results is a list, with each list item being the result from one shot
results = job.result()
# Since we run the circuit with only 1 shot,
# we only need to retrieve the first item from the memory.
outcome = results.get_memory()[0]

# The outcome should detect the error
# The outcome should be of type 'string'
print("Measured Outcome:", outcome, "  Type:", type(outcome))

# Introduce a barrier
qc.barrier()

# Add an X-gate as per measurement outcome to correct the error
if outcome == '01':
    qc.x(0)
elif outcome == '10':
    qc.x(2)
elif outcome == '11':
    qc.x(1)

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

```
Measured Outcome: 01   Type: <class 'str'>
```

Out[5]:



As we can see when we run the simulator the values for q_4 and q_3 are 01 as specificed in the measured outcome. And thus the value which needed to be flipped was q_0.