Experiment 3

Aim

Implementation of Classical gates using multi qubit quantum gates in Qiskit.

Theory

The implementation of classical gates using multi-qubit quantum gates in Qiskit involves using quantum gates to simulate classical operations. In this experiment, we will use the controlled-NOT (CNOT) gate and the Toffoli gate, which are two multi-qubit gates commonly used for implementing classical gates.

The CNOT gate is a two-qubit gate that flips the target qubit if the control qubit is in the state —1⟩. The CNOT gate can be used to simulate classical gates, such as the AND, OR, and NOT gates. The truth table for the CNOT gate is given by:

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

Here, the first two rows correspond to the input states $|00\rangle$ and $|01\rangle$, and the last two rows correspond to the input states $|10\rangle$ and $|11\rangle$. The first column corresponds to the output state $|00\rangle$, the second column corresponds to the output state $|01\rangle$, and so on.

To implement classical gates using the CNOT gate, we can use one qubit as the control qubit and the other qubit as the target qubit. If the control qubit is in the state $|1\rangle$, the target qubit is flipped, simulating the classical operation.

The Toffoli gate, also known as the controlled-controlled-NOT (CCNOT) gate, is a three-qubit gate that flips the target qubit if both control qubits are in the state $|1\rangle$. The Toffoli gate can be used to simulate classical gates, such as the NAND gate. The truth table for the Toffoli gate is given by:

$$CCNOT = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Here, the first four rows correspond to the input states $|000\rangle$, $|001\rangle$, $|010\rangle$, and $|011\rangle$, and the last four rows correspond to the input states $|100\rangle$, $|101\rangle$, $|110\rangle$, and $|111\rangle$. The first column corresponds to the output state —000⟩, the second column corresponds to the output state $|001\rangle$, and so on.

To implement classical gates using the Toffoli gate, we can use two qubits as the control qubits and the third qubit as the target qubit. If both control qubits are in the state —1⟩, the target qubit is flipped, simulating the classical operation.

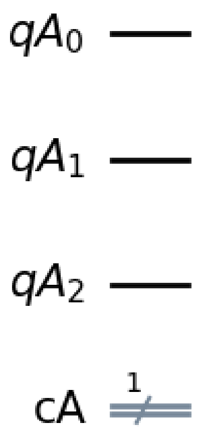
Lab Exercise

Code:

In [1]:
```
from qiskit import *
from qiskit.quantum_info import Statevector
import numpy as np

qA = QuantumRegister(3, name='qA')
cA = ClassicalRegister(1, name='cA')
circuit_A = QuantumCircuit(qA, cA)

qO = QuantumRegister(3, name='qO')
cO = ClassicalRegister(1, name='cO')
circuit_O = QuantumCircuit(qO, cO)

qE = QuantumRegister(2, name='qE')
cE = ClassicalRegister(1, name='cE')
circuit_E = QuantumCircuit(qE, cE)

%matplotlib inline
```

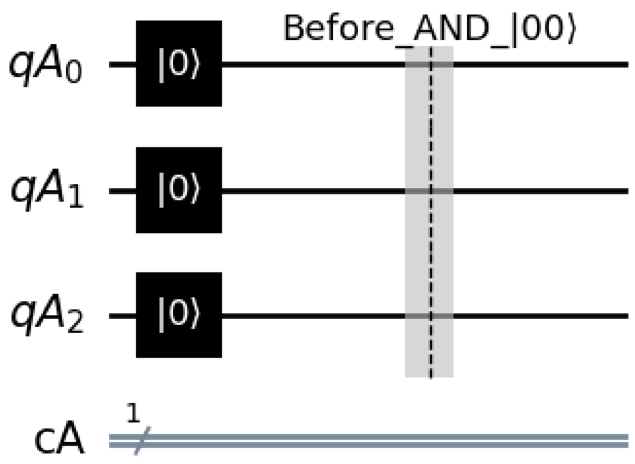
In [2]:
```
circuit_A.draw(output='mpl')
```

Out[2]:



In [3]:
```
circuit_A.reset(qA)
circuit_A.barrier(label='Before_AND_|00⟩')
circuit_A.draw(output='mpl')
```

Out[3]:



In [4]:

```
state_A_1 = Statevector.from_int(0, 2**3)
state_A_1 = state_A_1.evolve(circuit_A)
state_A_1.draw('latex')
```

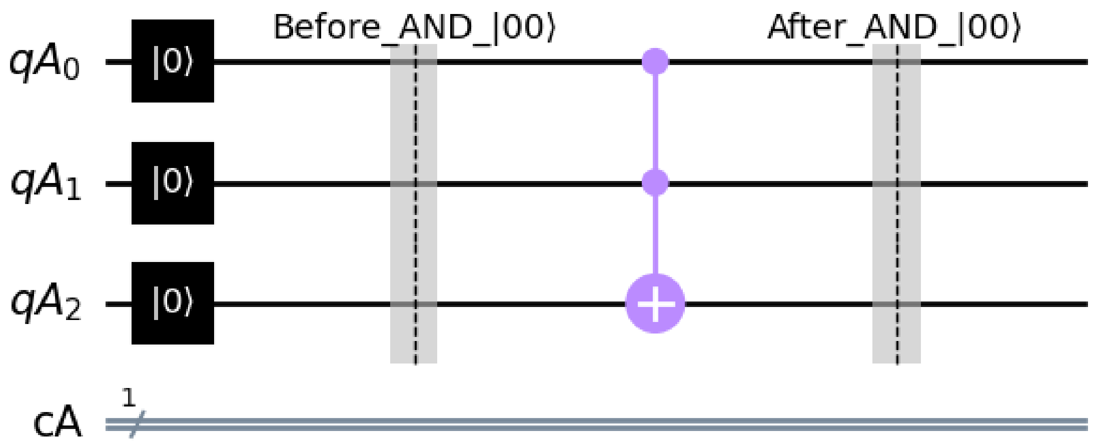
Out[4]:

$$|000\rangle$$

In [5]:

```
circuit_A.ccx(qA[0], qA[1], qA[2])
circuit_A.barrier(label='After_AND_|00⟩')
circuit_A.draw(output='mpl')
```

Out[5]:



In [6]:

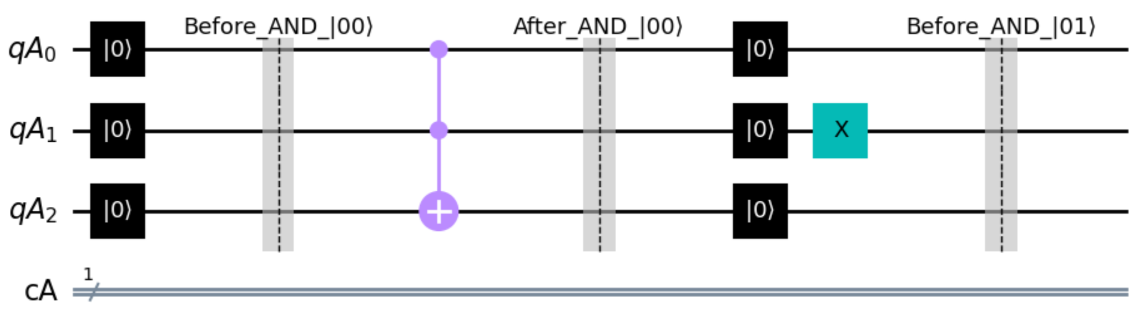
```
state_A_2 = Statevector.from_int(0, 2**3)
state_A_2 = state_A_2.evolve(circuit_A)
state_A_2.draw('latex')
#output(MSB) = 0
```

Out[6]:

$$|000\rangle$$

In [7]:
```
circuit_A.reset(qA)
circuit_A.x(qA[1])
circuit_A.barrier(label='Before_AND_|01⟩')
circuit_A.draw(output='mpl')
```

Out[7]:



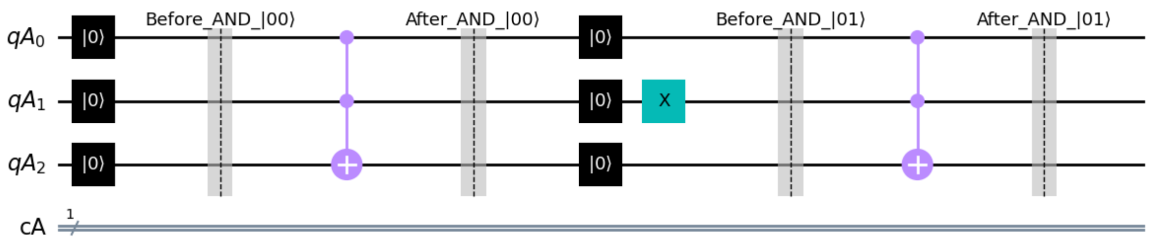
In [8]:
```
state_A_3 = Statevector.from_int(0, 2**3)
state_A_3 = state_A_3.evolve(circuit_A)
state_A_3.draw('latex')
```

Out[8]:

$$|010\rangle$$

In [9]:
```
circuit_A.ccx(qA[0], qA[1], qA[2])
circuit_A.barrier(label='After_AND_|01⟩')
circuit_A.draw(output='mpl')
```

Out[9]:



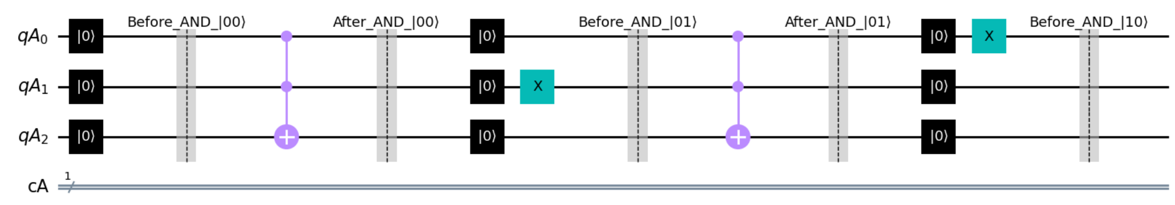
In [10]:
```
state_A_4 = Statevector.from_int(0, 2**3)
state_A_4 = state_A_4.evolve(circuit_A)
state_A_4.draw('latex')
#output(MSB) = 0
```

Out[10]:

$$|010\rangle$$

In [11]:
```
circuit_A.reset(qA)
circuit_A.x(qA[0])
circuit_A.barrier(label='Before_AND_|10⟩')
circuit_A.draw(output='mpl')
```

Out[11]:



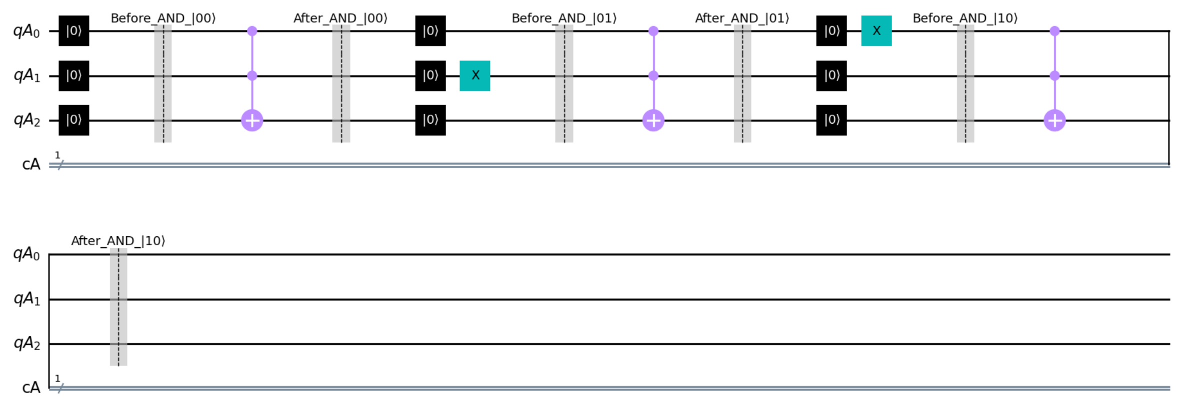
In [12]:
```
state_A_5 = Statevector.from_int(0, 2**3)
state_A_5 = state_A_5.evolve(circuit_A)
state_A_5.draw('latex')
```

Out[12]:

$$|001\rangle$$

In [13]:
```
circuit_A.ccx(qA[0], qA[1], qA[2])
circuit_A.barrier(label='After_AND_|10⟩')
circuit_A.draw(output='mpl')
```

Out[13]:



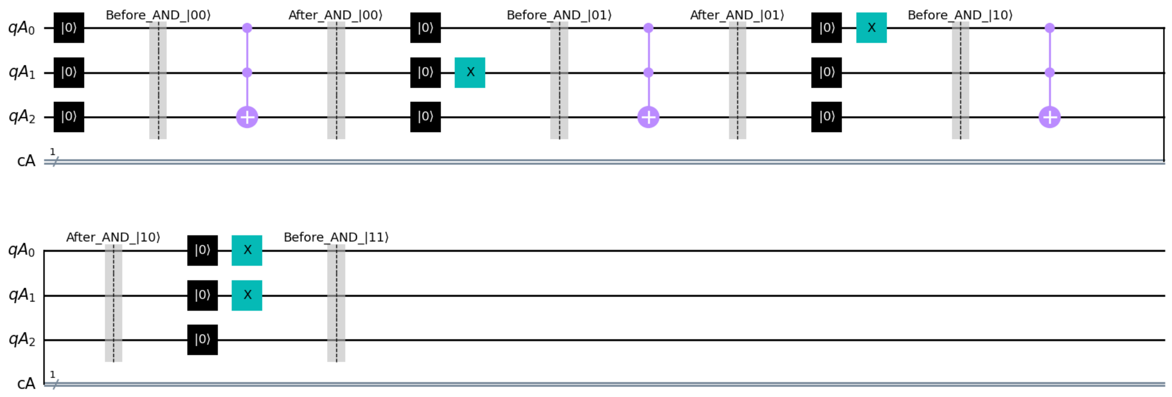
In [14]:
```
state_A_6 = Statevector.from_int(0, 2**3)
state_A_6 = state_A_6.evolve(circuit_A)
state_A_6.draw('latex')
#output(MSB) = 0
```

Out[14]:

$$|001\rangle$$

In [15]:
```
circuit_A.reset(qA)
circuit_A.x(qA[0])
circuit_A.x(qA[1])
circuit_A.barrier(label='Before_AND_|11⟩')
circuit_A.draw(output='mpl')
```

Out[15]:



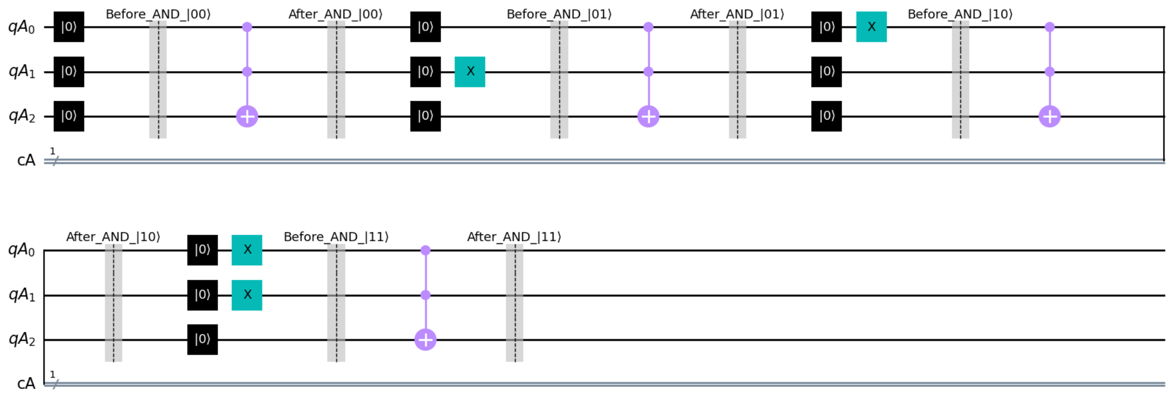
In [16]:
```
state_A_7 = Statevector.from_int(0, 2**3)
state_A_7 = state_A_7.evolve(circuit_A)
state_A_7.draw('latex')
```

Out[16]:

$$|011\rangle$$

In [17]:
```
circuit_A.ccx(qA[0], qA[1], qA[2])
circuit_A.barrier(label='After_AND_|11⟩')
circuit_A.draw(output='mpl')
```

Out[17]:



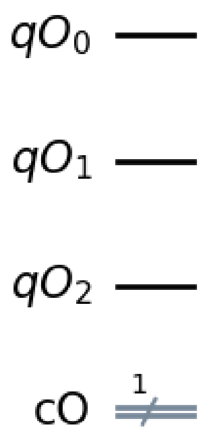
In [18]:
```
state_A_8 = Statevector.from_int(0, 2**3)
state_A_8 = state_A_8.evolve(circuit_A)
state_A_8.draw('latex')
#output(MSB) = 1
```

Out[18]:

$$|111\rangle$$

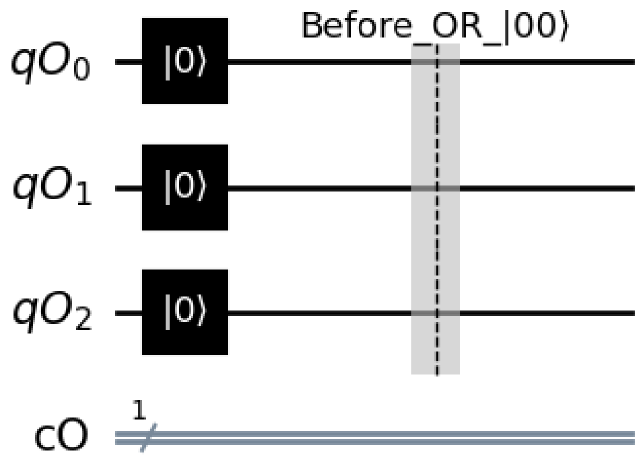
In [19]:
```
circuit_O.draw(output='mpl')
```

Out[19]:



In [20]:
```
circuit_O.reset(qO)
circuit_O.barrier(label='Before_OR_|00⟩')
circuit_O.draw(output='mpl')
```

Out[20]:



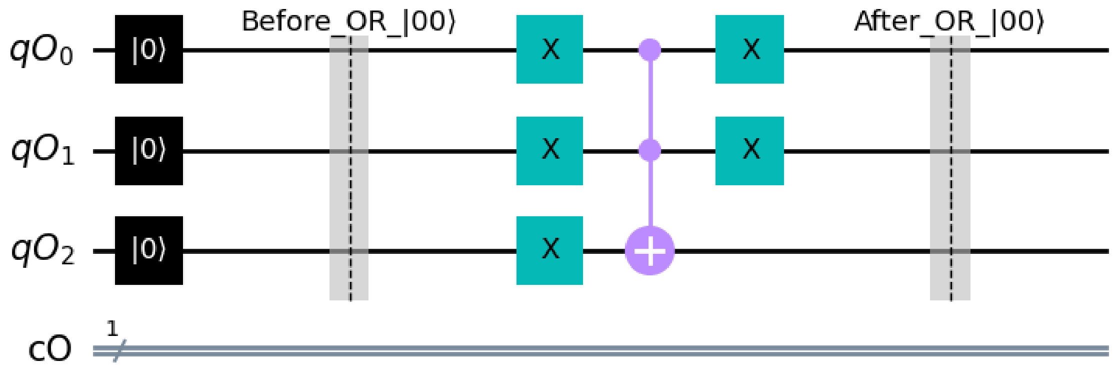
In [21]:
```
state_O_1 = Statevector.from_int(0, 2**3)
state_O_1 = state_O_1.evolve(circuit_O)
state_O_1.draw('latex')
```

Out[21]:

$$|000\rangle$$

In [22]:
```
circuit_O.x(qO)
circuit_O.ccx(qO[0], qO[1], qO[2])
circuit_O.x(qO[0])
circuit_O.x(qO[1])
circuit_O.barrier(label='After_OR_|00⟩')
circuit_O.draw(output='mpl')
```

Out[22]:



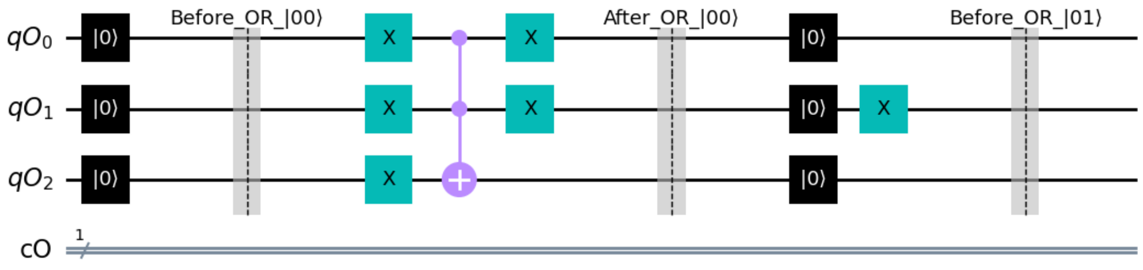
In [23]:
```
state_O_2 = Statevector.from_int(0, 2**3)
state_O_2 = state_O_2.evolve(circuit_O)
state_O_2.draw('latex')
#output(MSB) = 0
```

Out[23]:

$$|000\rangle$$

In [24]:
```
circuit_O.reset(qO)
circuit_O.x(qO[1])
circuit_O.barrier(label='Before_OR_|01⟩')
circuit_O.draw(output='mpl')
```

Out[24]:



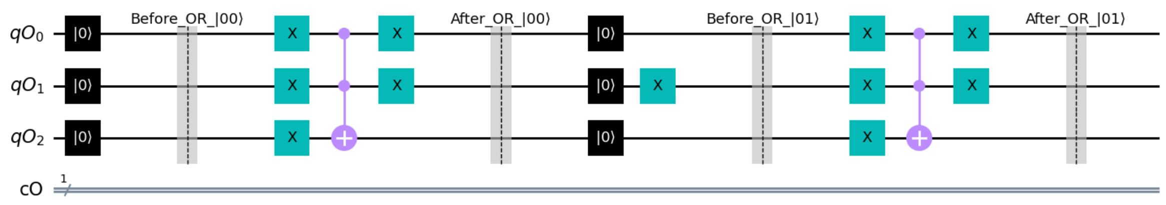
In [25]:
```
state_O_3 = Statevector.from_int(0, 2**3)
state_O_3 = state_O_3.evolve(circuit_O)
state_O_3.draw('latex')
```

Out[25]:

$$|010\rangle$$

In [26]:
```
circuit_O.x(qO)
circuit_O.ccx(qO[0], qO[1], qO[2])
circuit_O.x(qO[0])
circuit_O.x(qO[1])
circuit_O.barrier(label='After_OR_|01⟩')
circuit_O.draw(output='mpl')
```

Out[26]:



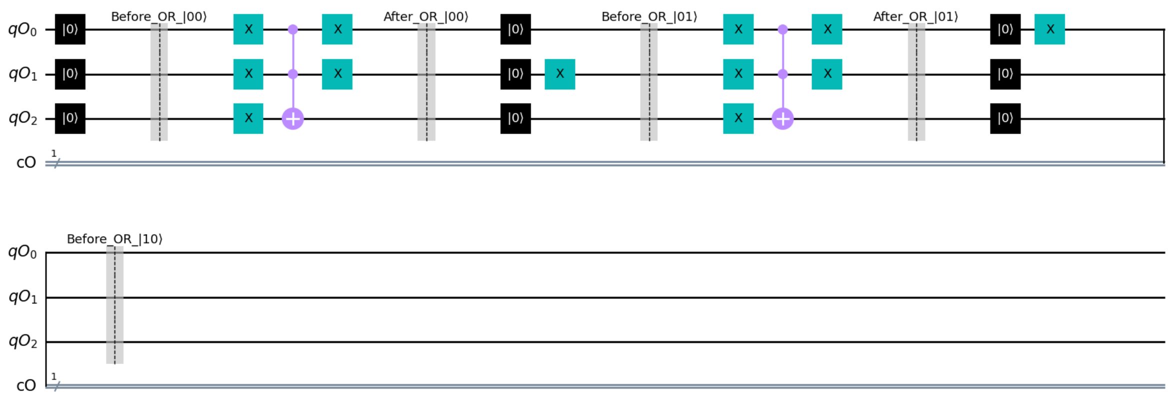
In [27]:
```
state_O_4 = Statevector.from_int(0, 2**3)
state_O_4 = state_O_4.evolve(circuit_O)
state_O_4.draw('latex')
#output(MSB) = 1
```

Out[27]:

$$|110\rangle$$

In [28]:
```
circuit_O.reset(qO)
circuit_O.x(qO[0])
circuit_O.barrier(label='Before_OR_|10⟩')
circuit_O.draw(output='mpl')
```

Out[28]:



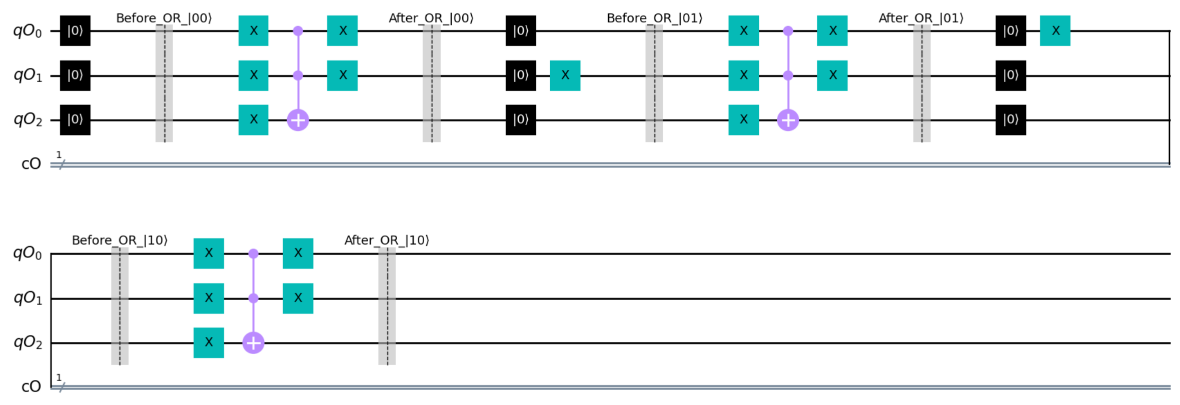
In [29]:
```
state_O_5 = Statevector.from_int(0, 2**3)
state_O_5 = state_O_5.evolve(circuit_O)
state_O_5.draw('latex')
```

Out[29]:

$$|001\rangle$$

In [30]:
```
circuit_O.x(qO)
circuit_O.ccx(qO[0], qO[1], qO[2])
circuit_O.x(qO[0])
circuit_O.x(qO[1])
circuit_O.barrier(label='After_OR_|10⟩')
circuit_O.draw(output='mpl')
```

Out[30]:



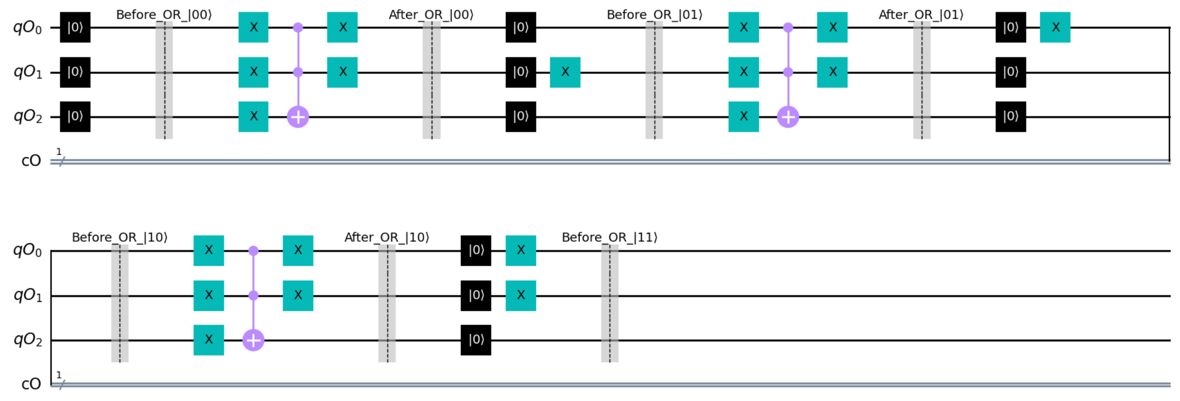
In [31]:
```
state_O_6 = Statevector.from_int(0, 2**3)
state_O_6 = state_O_6.evolve(circuit_O)
state_O_6.draw('latex')
#output(MSB) = 1
```

Out[31]:

$$|101\rangle$$

In [32]:
```
circuit_O.reset(qO)
circuit_O.x(qO[0])
circuit_O.x(qO[1])
circuit_O.barrier(label='Before_OR_|11⟩')
circuit_O.draw(output='mpl')
```

Out[32]:



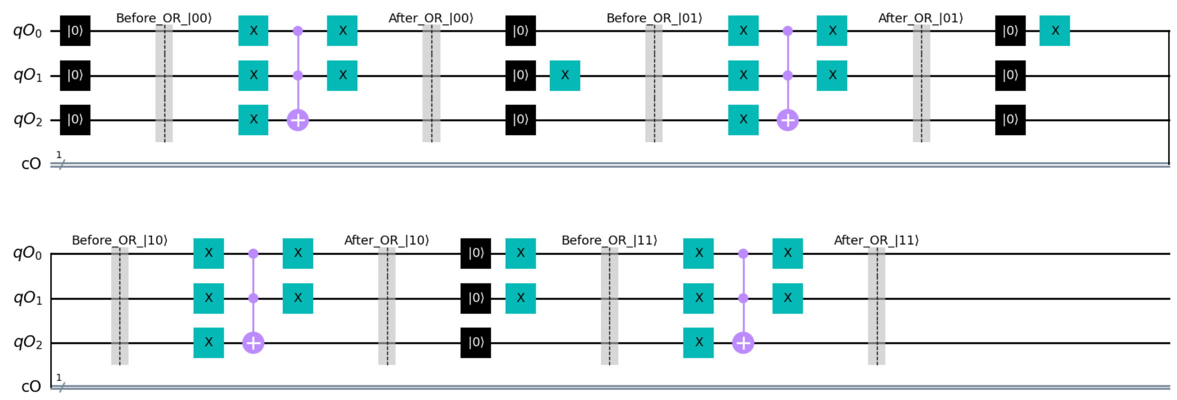
In [33]:
```
state_O_7 = Statevector.from_int(0, 2**3)
state_O_7 = state_O_7.evolve(circuit_O)
state_O_7.draw('latex')
```

Out[33]:

$$|011\rangle$$

In [34]:
```
circuit_O.x(qO)
circuit_O.ccx(qO[0], qO[1], qO[2])
circuit_O.x(qO[0])
circuit_O.x(qO[1])
circuit_O.barrier(label='After_OR_|11⟩')
circuit_O.draw(output='mpl')
```

Out[34]:



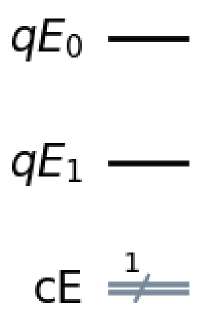
In [35]:
```
state_O_8 = Statevector.from_int(0, 2**3)
state_O_8 = state_O_8.evolve(circuit_O)
state_O_8.draw('latex')
#output(MSB) = 1
```

Out[35]:

$$|111\rangle$$

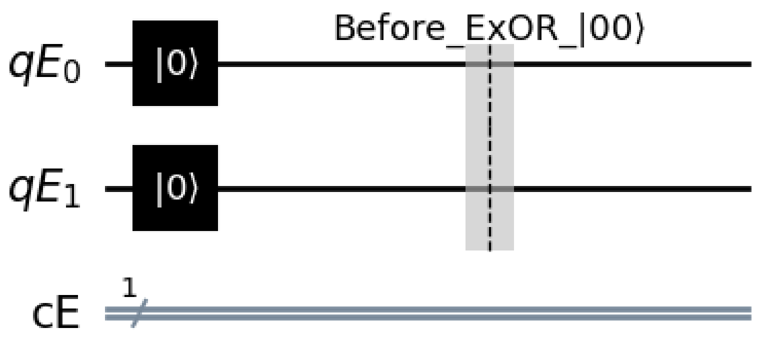
In [36]:
```
circuit_E.draw(output='mpl')
```

Out[36]:



In [37]:
```
circuit_E.reset(qE)
circuit_E.barrier(label='Before_ExOR_|00⟩')
circuit_E.draw(output='mpl')
```

Out[37]:



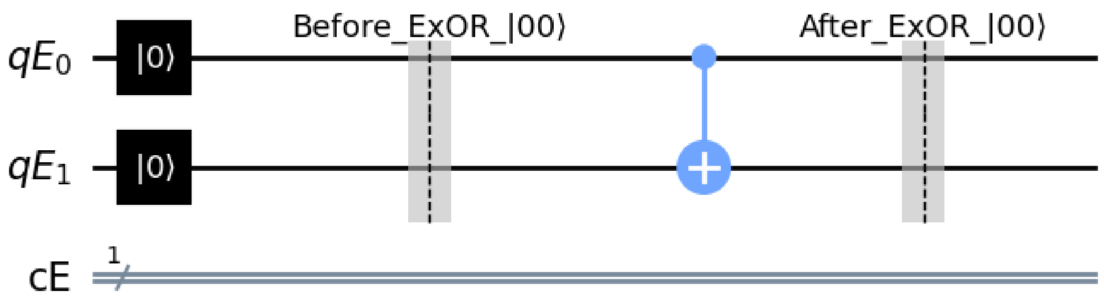
In [38]:
```
state_E_1 = Statevector.from_int(0, 2**2)
state_E_1 = state_E_1.evolve(circuit_E)
state_E_1.draw('latex')
```

Out[38]:

$$|00\rangle$$

In [39]:
```
circuit_E.cx(qE[0], qE[1])
circuit_E.barrier(label='After_ExOR_|00⟩')
circuit_E.draw(output='mpl')
```

Out[39]:



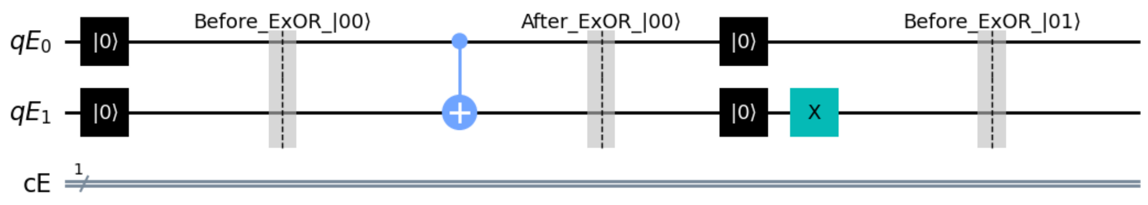
In [40]:
```
state_E_2 = Statevector.from_int(0, 2**2)
state_E_2 = state_E_2.evolve(circuit_E)
state_E_2.draw('latex')
#output(MSB) = 0
```

Out[40]:

$$|00\rangle$$

In [41]:
```
circuit_E.reset(qE)
circuit_E.x(qE[1])
circuit_E.barrier(label='Before_ExOR_|01⟩')
circuit_E.draw(output='mpl')
```

Out[41]:



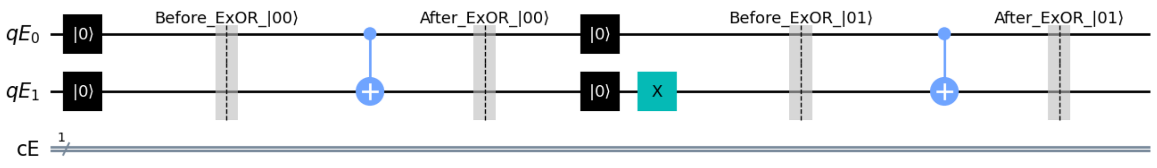
In [42]:
```
state_E_3 = Statevector.from_int(0, 2**2)
state_E_3 = state_E_3.evolve(circuit_E)
state_E_3.draw('latex')
```

Out[42]:

$$|10\rangle$$

In [43]:
```
circuit_E.cx(qE[0], qE[1])
circuit_E.barrier(label='After_ExOR_|01⟩')
circuit_E.draw(output='mpl')
```

Out[43]:



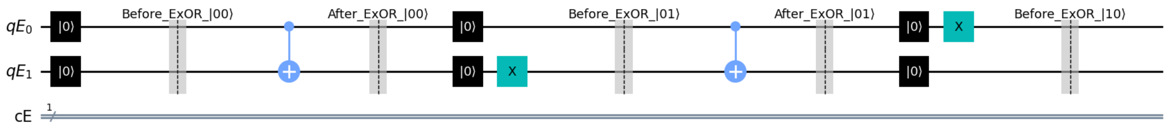
In [44]:
```
state_E_4 = Statevector.from_int(0, 2**2)
state_E_4 = state_E_4.evolve(circuit_E)
state_E_4.draw('latex')
#output(MSB) = 1
```

Out[44]:

$$|10\rangle$$

In [45]:
```
circuit_E.reset(qE)
circuit_E.x(qE[0])
circuit_E.barrier(label='Before_ExOR_|10⟩')
circuit_E.draw(output='mpl')
```

Out[45]:



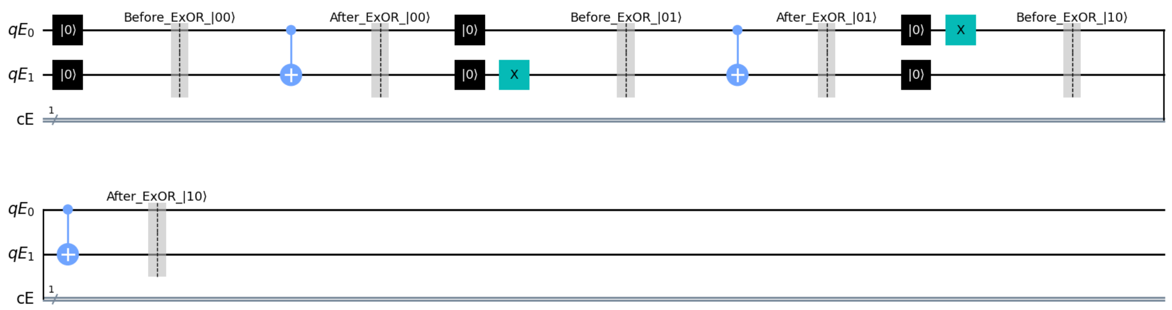
In [46]:
```
state_E_5 = Statevector.from_int(0, 2**2)
state_E_5 = state_E_5.evolve(circuit_E)
state_E_5.draw('latex')
```

Out[46]:

$$|01\rangle$$

In [47]:
```
circuit_E.cx(qE[0], qE[1])
circuit_E.barrier(label='After_ExOR_|10⟩')
circuit_E.draw(output='mpl')
```

Out[47]:



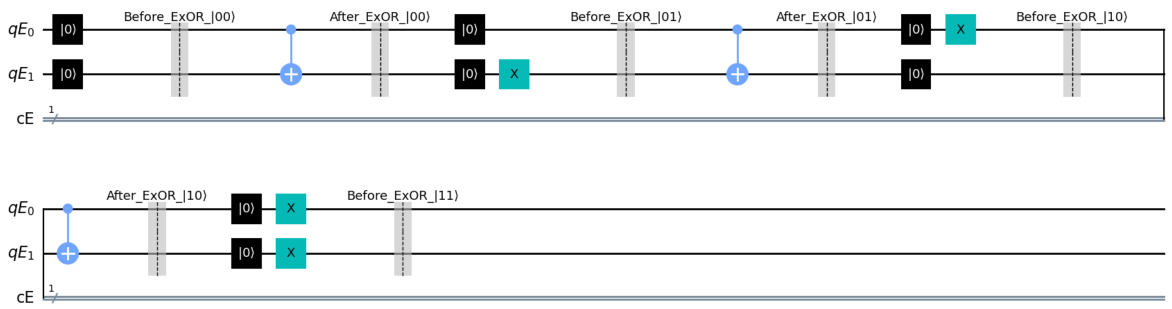
In [48]:
```
state_E_6 = Statevector.from_int(0, 2**2)
state_E_6 = state_E_6.evolve(circuit_E)
state_E_6.draw('latex')
#output(MSB) = 1
```

Out[48]:

$$|11\rangle$$

In [49]:
```
circuit_E.reset(qE)
circuit_E.x(qE[0])
circuit_E.x(qE[1])
circuit_E.barrier(label='Before_ExOR_|11⟩')
circuit_E.draw(output='mpl')
```

Out[49]:



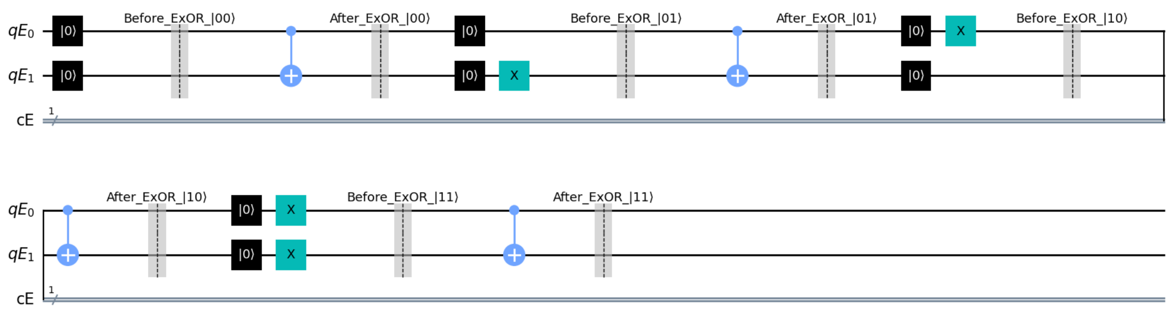
In [50]:
```
state_E_7 = Statevector.from_int(0, 2**2)
state_E_7 = state_E_7.evolve(circuit_E)
state_E_7.draw('latex')
```

Out[50]:

$$|11\rangle$$

In [51]:
```
circuit_E.cx(qE[0], qE[1])
circuit_E.barrier(label='After_ExOR_|11⟩')
circuit_E.draw(output='mpl')
```

Out[51]:



In [52]:

```
state_E_8 = Statevector.from_int(0, 2**2)
state_E_8 = state_E_8.evolve(circuit_E)
state_E_8.draw('latex')
#output(MSB) = 0
```

Out[52]:

$$|01\rangle$$

In [ ]:

```
```

Conclusion

In conclusion, it has been successfully shown that the multi-qubit quantum gates Controlled Not and Toffoli can be used to build classical gates in Qiskit. We have demonstrated how these multi-qubit gates can be used to create conventional gates like AND, OR and XOR. The outcomes of the simulations performed with the Qiskit simulator were in line with the anticipated classical logic operations.

We have also shown that these gates can be utilised to develop quantum error correcting codes and to generate entangled states. This demonstrates how quantum computers have the ability to execute calculations that are not possible with classical computers.

The use of multi-qubit gates in Qiskit has practical applications in quantum computing, particularly in the areas of quantum error correction and quantum-classical computation. As quantum computing technology continues to advance, the use of these gates will become increasingly important for realizing the potential of quantum computing in solving real-world problems.