MDS572B: QUANTUM MACHINE LEARNING

OBSERVATION NOTEBOOK

Name: Neelanjan Dutta

Roll No: 2448040

Class: 5MDS

LAB No. 3: Create a simple quantum circuit using basic gates

Question:

Implement a circuit for representing an XOR gate using quantum gates

Objective:

The objective of this lab is to design and implement quantum circuits that replicate the functionality of the fundamental classical logic gates: NOT, AND, OR, and XOR. The lab aims to verify the correctness of these quantum implementations by simulating each circuit for all possible input combinations and comparing the outputs against their respective classical truth tables.

Draft Plan:

Program description:

This program uses the Qiskit library to construct four distinct quantum circuits, each corresponding to a classical gate (NOT, AND, OR, and XOR). For each gate, the program systematically tests all possible binary inputs by initializing the qubits, applying the appropriate quantum logic (using Pauli-X, CNOT, and Toffoli gates), and simulating the circuit on the **AerSimulator**. The results are then printed to verify that each quantum circuit correctly reproduces the truth table of its classical counterpart. An example circuit diagram is also generated for each implementation.

Program logic:

1. **Initialization:** The necessary libraries from Qiskit are imported, and the AerSimulator is set up as the backend for running the circuit simulations.
2. **NOT Gate Logic:**
   - ❖ A 1-qubit circuit is used.
   - ❖ The program iterates through inputs 0 and 1.
   - ❖ The input state is prepared on the qubit (an X gate is applied for input 1).
   - ❖ A Pauli-X gate is applied to perform the logical NOT operation.

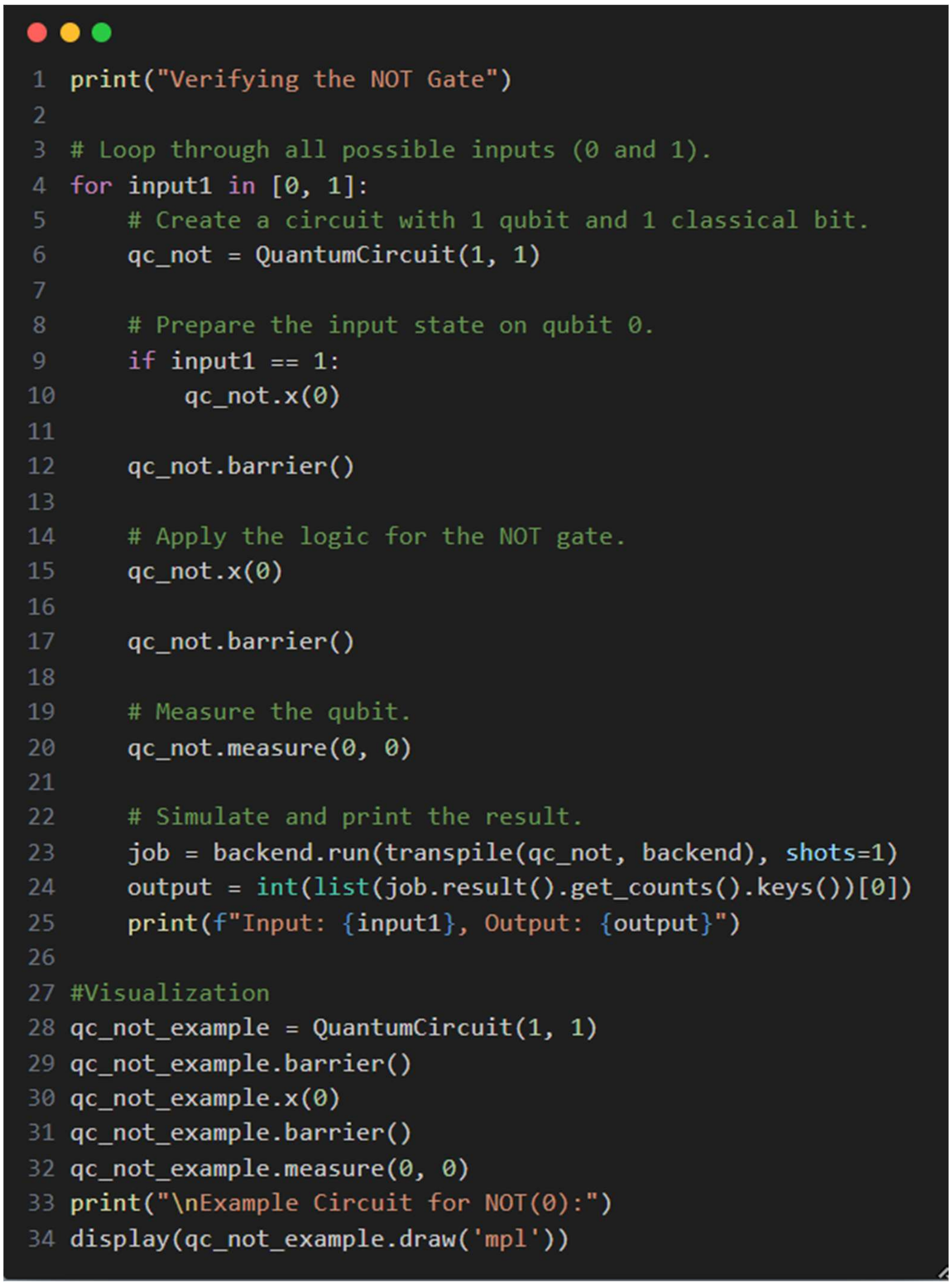
- ❖ The qubit is measured, and the result is recorded to verify the bit-flip.

3. **<u>AND Gate Logic</u>:**
   - ❖ A 3-qubit circuit is used (two for inputs, one for output).
   - ❖ The program iterates through all four input combinations (00, 01, 10, 11).
   - ❖ Input states are prepared on the first two qubits (q_0, q_1).
   - ❖ A Toffoli (CCX) gate is applied, with q_0 and q_1 as the control qubits and q_2 as the target. The target qubit is flipped only if both controls are in the state.
   - ❖ The target qubit q_2 is measured to get the result of the AND operation.
4. **<u>OR Gate Logic</u>:**
   - ❖ A 3-qubit circuit is used.
   - ❖ The program iterates through all four input combinations.
   - ❖ Input states are prepared on q_0 and q_1.
   - ❖ A specific combination of CCX and CNOT gates is applied to the three qubits to construct the OR logic on the target qubit, q_2.
   - ❖ The target qubit q_2 is measured for the final result.
5. **<u>XOR Gate Logic</u>:**
   - ❖ A 2-qubit circuit is used.
   - ❖ The program iterates through all four input combinations.
   - ❖ Input states are prepared on q_0 and q_1.
   - ❖ A CNOT (CX) gate is applied, where q_0 is the control qubit and q_1 is the target. The target qubit is flipped only if the control is in the state.
   - ❖ The target qubit q_1 is measured, as it now holds the result of the XOR operation.

Program:

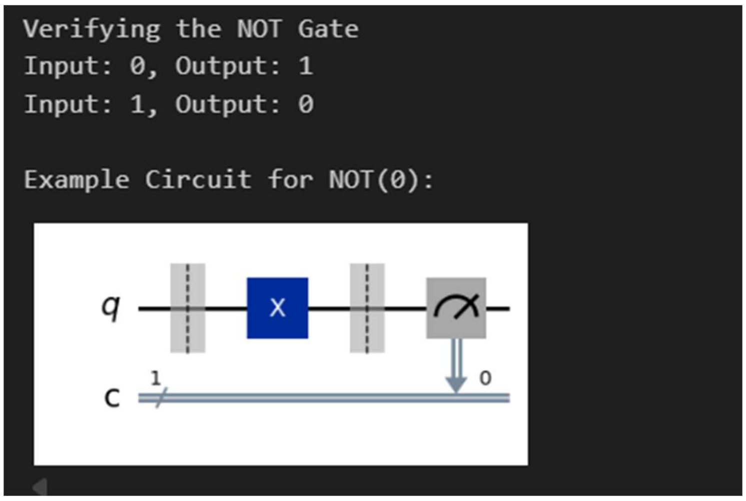
1. **NOT GATE:**

The quantum equivalent of a classical NOT gate is the **Pauli-X gate**. It flips a qubit from |**0**> to |**1**> and |**1**> to |**0**>.



```python
print("Verifying the NOT Gate")

# Loop through all possible inputs (0 and 1).
for input1 in [0, 1]:
    # Create a circuit with 1 qubit and 1 classical bit.
    qc_not = QuantumCircuit(1, 1)

    # Prepare the input state on qubit 0.
    if input1 == 1:
        qc_not.x(0)

    qc_not.barrier()

    # Apply the logic for the NOT gate.
    qc_not.x(0)

    qc_not.barrier()

    # Measure the qubit.
    qc_not.measure(0, 0)

    # Simulate and print the result.
    job = backend.run(transpile(qc_not, backend), shots=1)
    output = int(list(job.result().get_counts().keys())[0])
    print(f"Input: {input1}, Output: {output}")

#Visualization
qc_not_example = QuantumCircuit(1, 1)
qc_not_example.barrier()
qc_not_example.x(0)
qc_not_example.barrier()
qc_not_example.measure(0, 0)
print("\nExample Circuit for NOT(0):")
display(qc_not_example.draw('mpl'))
```

Output:



```
Verifying the NOT Gate
Input: 0, Output: 1
Input: 1, Output: 0

Example Circuit for NOT(0):
```

2. **AND GATE:**

The quantum equivalent of an AND gate is the Toffoli gate (or CCX gate). It acts on three qubits: two "control" qubits and one "target" qubit. It flips the target qubit if and only if both control qubits are in the |1> state.

Truth Table: A AND B = C

0 AND 0 = 0

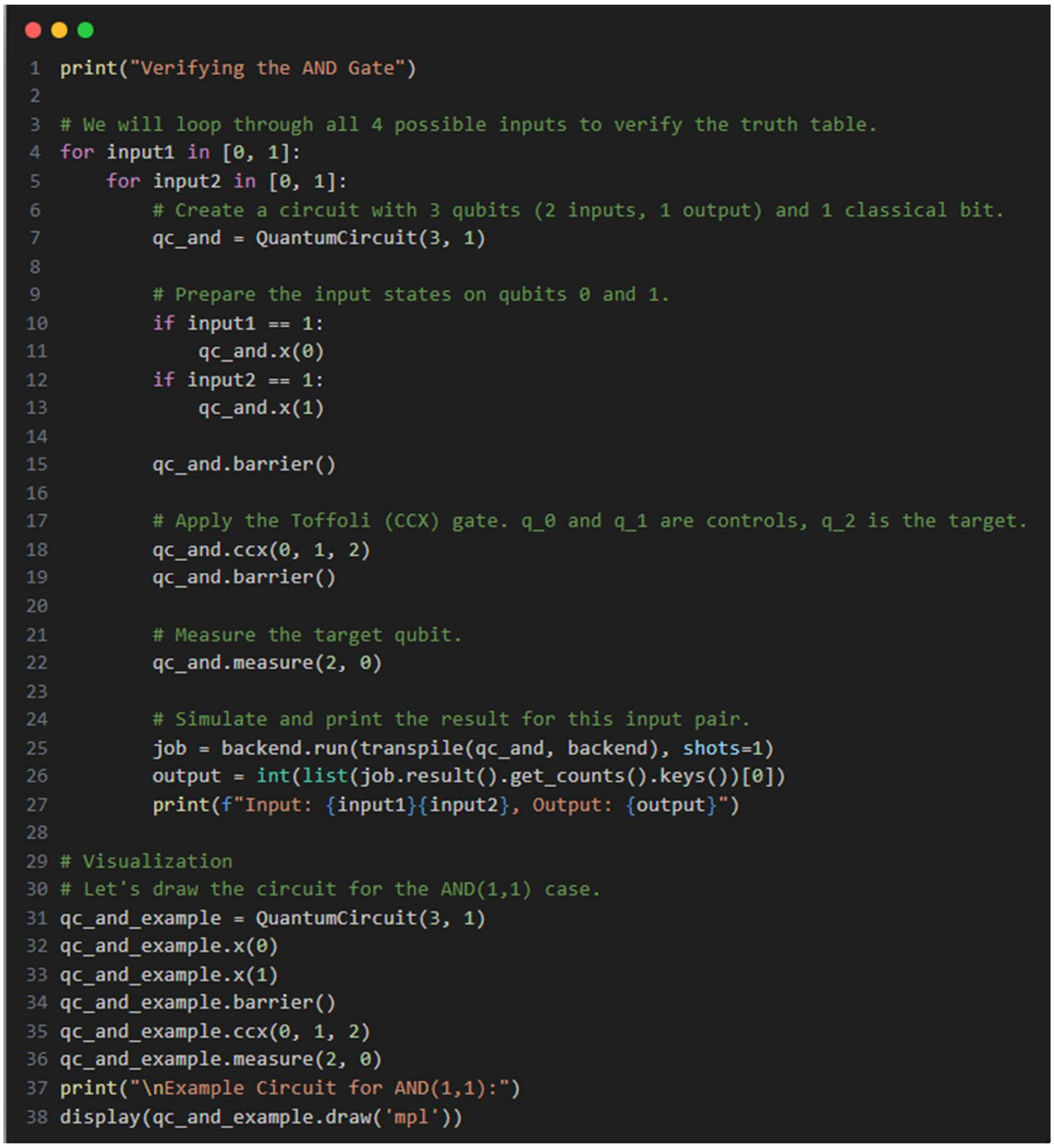
0 AND 1 = 0

1 AND 0 = 0

1 AND 1 = 1

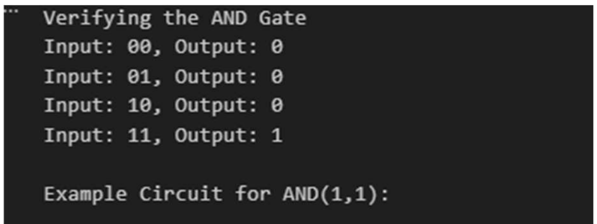
Program:

P.T.O.

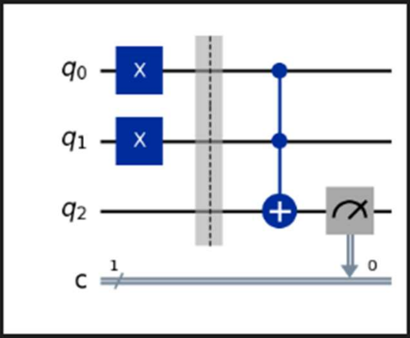


```python
print("Verifying the AND Gate")

# We will loop through all 4 possible inputs to verify the truth table.
for input1 in [0, 1]:
    for input2 in [0, 1]:
        # Create a circuit with 3 qubits (2 inputs, 1 output) and 1 classical bit.
        qc_and = QuantumCircuit(3, 1)

        # Prepare the input states on qubits 0 and 1.
        if input1 == 1:
            qc_and.x(0)
        if input2 == 1:
            qc_and.x(1)

        qc_and.barrier()

        # Apply the Toffoli (CCX) gate. q_0 and q_1 are controls, q_2 is the target.
        qc_and.ccx(0, 1, 2)
        qc_and.barrier()

        # Measure the target qubit.
        qc_and.measure(2, 0)

        # Simulate and print the result for this input pair.
        job = backend.run(transpile(qc_and, backend), shots=1)
        output = int(list(job.result().get_counts().keys())[0])
        print(f"Input: {input1}{input2}, Output: {output}")

# Visualization
# Let's draw the circuit for the AND(1,1) case.
qc_and_example = QuantumCircuit(3, 1)
qc_and_example.x(0)
qc_and_example.x(1)
qc_and_example.barrier()
qc_and_example.ccx(0, 1, 2)
qc_and_example.measure(2, 0)
print("\nExample Circuit for AND(1,1):")
display(qc_and_example.draw('mpl'))
```

Output:



```
Verifying the AND Gate
Input: 00, Output: 0
Input: 01, Output: 0
Input: 10, Output: 0
Input: 11, Output: 1

Example Circuit for AND(1,1):
```



3. OR GATE:

An OR gate can be constructed from a Toffoli gate using De Morgan's laws: A OR B = NOT ((NOT A) AND (NOT B)). We apply X gates before and after the controls and after the target of a **CCX** gate.

Truth Table: A OR B = C

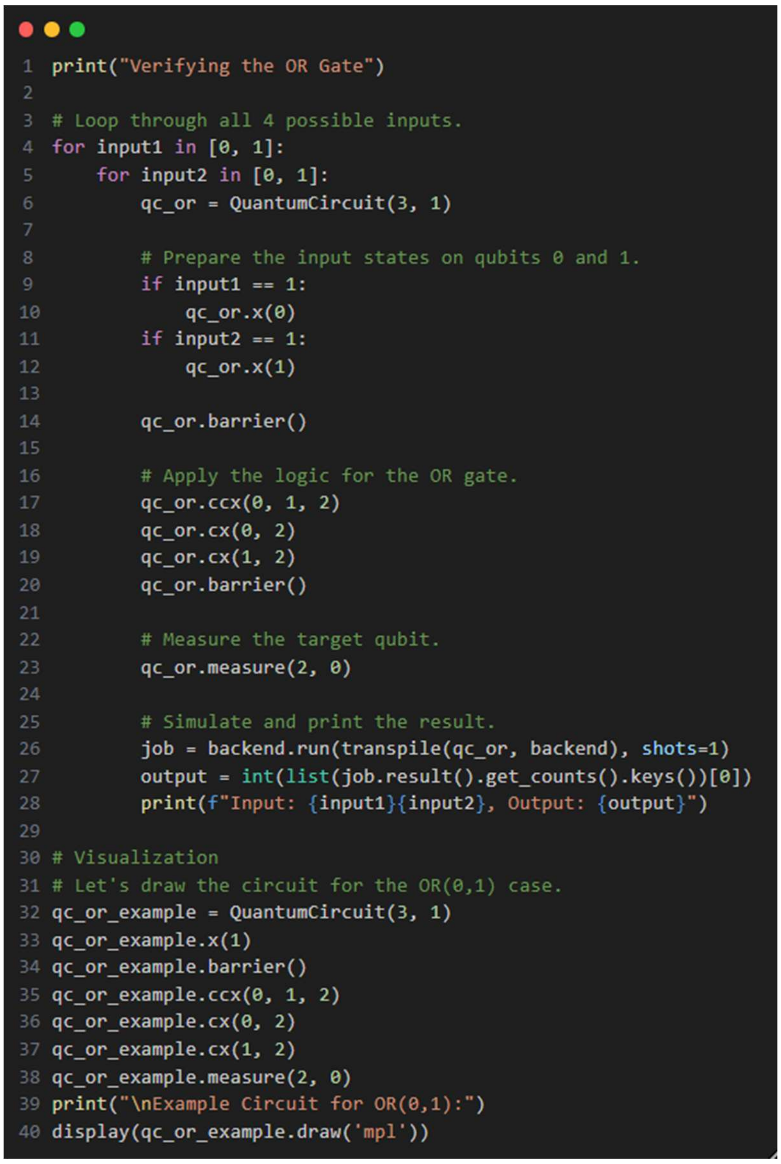
0 OR 0 = 0

0 OR 1 = 1

1 OR 0 = 1

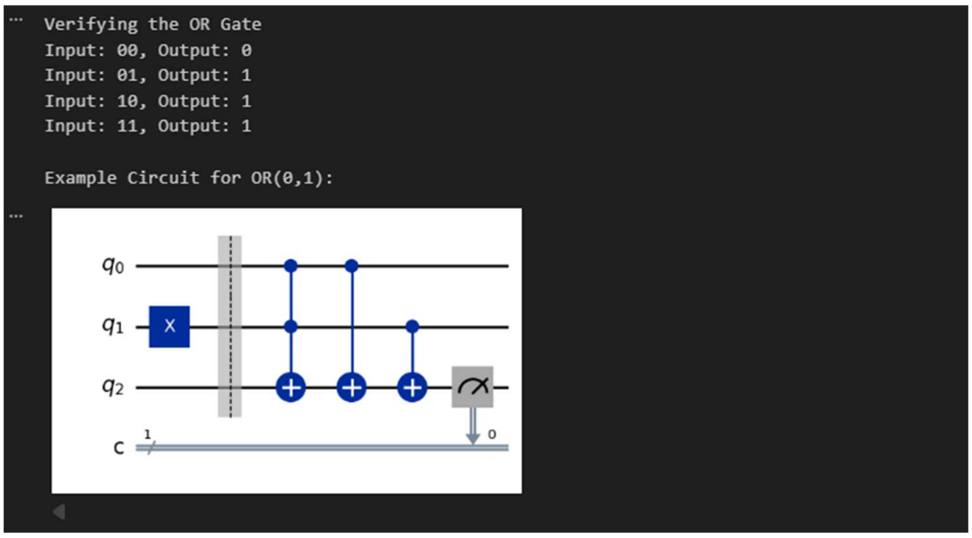
1 OR 1 = 1

Program:



```python
print("Verifying the OR Gate")

# Loop through all 4 possible inputs.
for input1 in [0, 1]:
    for input2 in [0, 1]:
        qc_or = QuantumCircuit(3, 1)

        # Prepare the input states on qubits 0 and 1.
        if input1 == 1:
            qc_or.x(0)
        if input2 == 1:
            qc_or.x(1)

        qc_or.barrier()

        # Apply the logic for the OR gate.
        qc_or.ccx(0, 1, 2)
        qc_or.cx(0, 2)
        qc_or.cx(1, 2)
        qc_or.barrier()

        # Measure the target qubit.
        qc_or.measure(2, 0)

        # Simulate and print the result.
        job = backend.run(transpile(qc_or, backend), shots=1)
        output = int(list(job.result().get_counts().keys())[0])
        print(f"Input: {input1}{input2}, Output: {output}")

# Visualization
# Let's draw the circuit for the OR(0,1) case.
qc_or_example = QuantumCircuit(3, 1)
qc_or_example.x(1)
qc_or_example.barrier()
qc_or_example.ccx(0, 1, 2)
qc_or_example.cx(0, 2)
qc_or_example.cx(1, 2)
qc_or_example.measure(2, 0)
print("\nExample Circuit for OR(0,1):")
display(qc_or_example.draw('mpl'))
```

<u>Output</u>:



```
Verifying the OR Gate
Input: 00, Output: 0
Input: 01, Output: 1
Input: 10, Output: 1
Input: 11, Output: 1

Example Circuit for OR(0,1):
```

4. <u>XOR GATE</u>:

The quantum equivalent of a classical XOR gate is the Controlled-NOT **(or CNOT / CX**) gate. It flips the target qubit if the control qubit is |**1>**

Truth Table: A XOR B = C

0 XOR 0 = 0

0 XOR 1 = 1

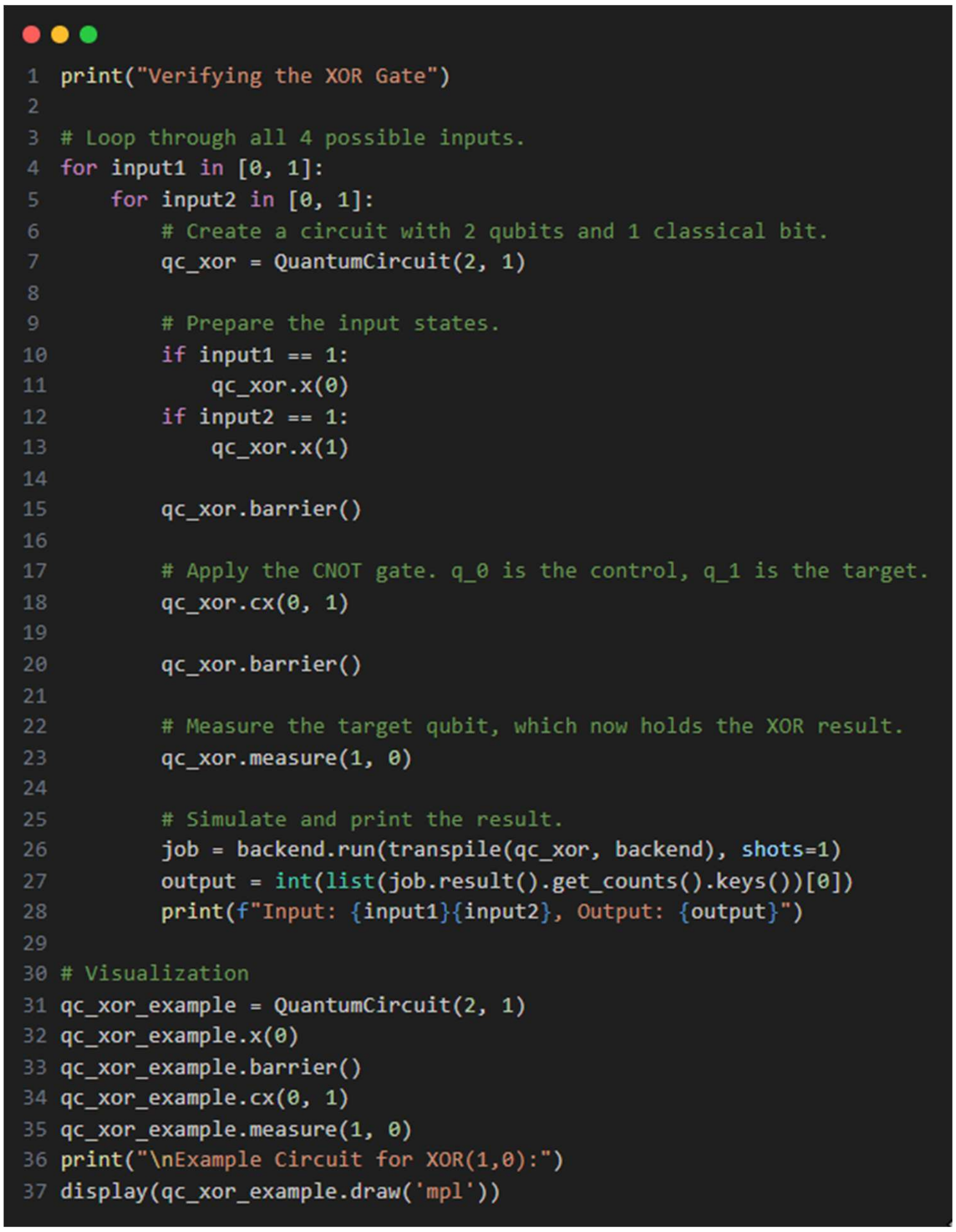
1 XOR 0 = 1

1 XOR 1 = 0

We can implement this by initializing the second qubit to the value of the second input (B) and then applying the CNOT. The final value on the second qubit will be A XOR B.

<u>Program</u>:

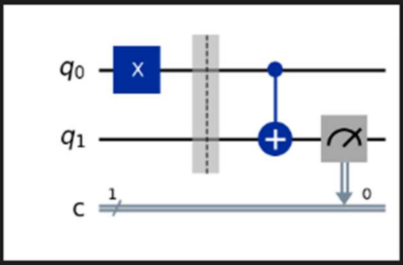
P.T.O.



```python
print("Verifying the XOR Gate")

# Loop through all 4 possible inputs.
for input1 in [0, 1]:
    for input2 in [0, 1]:
        # Create a circuit with 2 qubits and 1 classical bit.
        qc_xor = QuantumCircuit(2, 1)

        # Prepare the input states.
        if input1 == 1:
            qc_xor.x(0)
        if input2 == 1:
            qc_xor.x(1)

        qc_xor.barrier()

        # Apply the CNOT gate. q_0 is the control, q_1 is the target.
        qc_xor.cx(0, 1)

        qc_xor.barrier()

        # Measure the target qubit, which now holds the XOR result.
        qc_xor.measure(1, 0)

        # Simulate and print the result.
        job = backend.run(transpile(qc_xor, backend), shots=1)
        output = int(list(job.result().get_counts().keys())[0])
        print(f"Input: {input1}{input2}, Output: {output}")

# Visualization
qc_xor_example = QuantumCircuit(2, 1)
qc_xor_example.x(0)
qc_xor_example.barrier()
qc_xor_example.cx(0, 1)
qc_xor_example.measure(1, 0)
print("\nExample Circuit for XOR(1,0):")
display(qc_xor_example.draw('mpl'))
```

Output:

```
Verifying the XOR Gate
Input: 00, Output: 0
Input: 01, Output: 1
Input: 10, Output: 1
Input: 11, Output: 0

Example Circuit for XOR(1,0):
```



<u>**Conclusion**</u>**:**

In conclusion, this lab successfully demonstrated how fundamental classical logic gates can be constructed and verified using quantum circuits. The implementations correctly showed that the Pauli-X gate serves as a NOT gate, the CNOT gate as an XOR gate, and the Toffoli gate as an AND gate, which can then be used to construct an OR gate. By simulating each circuit for all possible inputs, it was confirmed that the quantum gates reproduce the exact truth tables of their classical counterparts. This exercise provides a foundational understanding of reversible computation and illustrates how classical logic can be embedded within a quantum framework.

****