Quantum computing **Lab_4** by Vivek Vittal Biragoni(**211Ai041**)

```
from qiskit import *
from qiskit.visualization import plot_histogram
import numpy as np
```

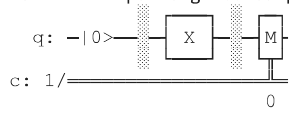
NOT GATE

```
def NOT(inp):

    qc = QuantumCircuit(1, 1) # A quantum circuit with a single qubit and a single classical bit
    qc.reset(0)

    # We encode '0' as the qubit state |0⟩, and '1' as |1⟩
    # Since the qubit is initially |0⟩, we don't need to do anything for an input of '0'
    # For an input of '1', we do an x to rotate the |0⟩ to |1⟩
    if inp=='1':
        qc.x(0)

    # barrier between input state and gate operation
    qc.barrier()

    # Now we've encoded the input, we can do a NOT on it using x
    qc.x(0)

    #barrier between gate operation and measurement
    qc.barrier()

    # Finally, we extract the |0⟩/|1⟩ output of the qubit and encode it in the bit c[0]
    qc.measure(0,0)
    qc.draw('mpl')

    # We'll run the program on a simulator
    backend = Aer.get_backend('aer_simulator')
    # Since the output will be deterministic, we can use just a single shot to get it
    job = backend.run(qc, shots=1, memory=True)
    output = job.result().get_memory()[0]

    return qc, output
```

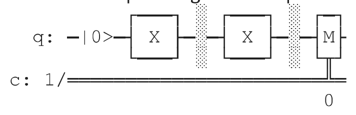
code defines a quantum circuit that performs a NOT operation on a single qubit based on an input bit. If the input bit is '1', the circuit applies an X gate to rotate the qubit from state |0⟩ to state |1⟩ before performing the NOT operation. The output of the circuit, which is either |0⟩ or |1⟩, is then measured and returned as a classical bit.

```
## Test the function
for inp in ['0', '1']:
    qc, out = NOT(inp)
    print('NOT with input',inp,'gives output',out)
    display(qc.draw())
    print('\n')
```

```
NOT with input 0 gives output 1
```



```
NOT with input 1 gives output 0
```



The code tests the NOT gate function by giving it inputs of 0 and 1. It then prints the output of the gate for each input and displays the corresponding quantum circuit diagram. The NOT gate function takes a single qubit input and returns the output qubit that is the negation of the input qubit.

XOR GATE

```
def XOR(inp1,inp2):

    qc = QuantumCircuit(2, 1)
    qc.reset(range(2))

    if inp1=='1':
        qc.x(0)
    if inp2=='1':
        qc.x(1)

    # barrier between input state and gate operation
    qc.barrier()

    # barrier between input state and gate operation
    qc.barrier()

    qc.measure(1,0) # output from qubit 1 is measured

    #We'll run the program on a simulator
    backend = Aer.get_backend('aer_simulator')
    #Since the output will be deterministic, we can use just a single shot to get it
    job = backend.run(qc, shots=1, memory=True)
    output = job.result().get_memory()[0]

    return qc, output
```

code defines a quantum circuit that performs an XOR operation on two input bits using two qubits. If the input bits are '1', the circuit applies an X gate to the corresponding qubit to rotate it from state |0⟩ to state |1⟩. The circuit then applies a CNOT gate to the two qubits, with the first qubit as the control and the second qubit as the target. Finally, the output of the second qubit is measured and returned as a classical bit.

```
## Test the function
for inp1 in ['0', '1']:
    for inp2 in ['0', '1']:
        qc, output = XOR(inp1, inp2)
        print('XOR with inputs',inp1,inp2,'gives output',output)
        display(qc.draw())
        print('\n')
```

```
XOR with inputs 0 0 gives output 0
             ░  ░    
q_0: ─|0>────░──░────
             ░  ░ ┌─┐
q_1: ─|0>────░──░─┤M├
             ░  ░ └╥┘
c: 1/═════════════╩═
                   0


XOR with inputs 0 1 gives output 1
                  ░  ░    
q_0: ─|0>─────────░──░────
         ┌───┐    ░  ░ ┌─┐
q_1: ─|0>┤ X ├────░──░─┤M├
         └───┘    ░  ░ └╥┘
c: 1/══════════════════╩═
                        0


XOR with inputs 1 0 gives output 0
         ┌───┐    ░  ░
```

```
q_1: ─|0>─────────░──░─┤M├
```

AND GATE

```
def AND(inp1,inp2):
    """An AND gate.

    Parameters:
        inpt1 (str): Input 1, encoded in qubit 0.
        inpt2 (str): Input 2, encoded in qubit 1.

    Returns:
        QuantumCircuit: Output XOR circuit.
        str: Output value measured from qubit 2.
    """
    qc = QuantumCircuit(3, 1)
    qc.reset(range(2))

    if inp1=='1':
        qc.x(0)
    if inp2=='1':
        qc.x(1)

    qc.barrier()

    # this is where your program for quantum AND gate goes






    qc.barrier()
    qc.measure(2, 0) # output from qubit 2 is measured

    # We'll run the program on a simulator
    backend = Aer.get_backend('aer_simulator')
    # Since the output will be deterministic, we can use just a single shot to get it
    job = backend.run(qc, shots=1, memory=True)
    output = job.result().get_memory()[0]

    return qc, output
```

code defines a quantum circuit that is intended to perform an AND operation on two input bits using three qubits. If the input bits are '1', the circuit applies an X gate to the corresponding qubit to rotate it from state |0⟩ to state |1⟩.

```
## Test the function
for inp1 in ['0', '1']:
    for inp2 in ['0', '1']:
        qc, output = AND(inp1, inp2)
        print('AND with inputs',inp1,inp2,'gives output',output)
        display(qc.draw())
        print('\n')
```

```
AND with inputs 0 0 gives output 0
                ░  ░
q_0: ─|0>───────░──░──────
                ░  ░
q_1: ─|0>───────░──░──────
                ░  ░ ┌─┐
q_2: ───────────░──░─┤M├─
                ░  ░ └╥┘
c: 1/════════════════╩══
                     0
```

```
AND with inputs 0 1 gives output 0
```

NAND GATE

```
q_1: ─|0>─┤ X ├─░──░──────
```

```
def NAND(inp1,inp2):
    """An NAND gate.

    Parameters:
        inpt1 (str): Input 1, encoded in qubit 0.
        inpt2 (str): Input 2, encoded in qubit 1.

    Returns:
        QuantumCircuit: Output NAND circuit.
        str: Output value measured from qubit 2.
    """
    qc = QuantumCircuit(3, 1)
    qc.reset(range(3))

    if inp1=='1':
        qc.x(0)
    if inp2=='1':
        qc.x(1)

    qc.barrier()

    # this is where your program for quantum NAND gate goes








    qc.barrier()
    qc.measure(2, 0) # output from qubit 2 is measured

    # We'll run the program on a simulator
    backend = Aer.get_backend('aer_simulator')
    # Since the output will be deterministic, we can use just a single shot to get it
    job = backend.run(qc,shots=1,memory=True)
    output = job.result().get_memory()[0]

    return qc, output
```

code defines a quantum circuit that is intended to perform a NAND operation on two input bits using three qubits. If the input bits are '1', the circuit applies an X gate to the corresponding qubit to rotate it from state |0⟩ to state |1⟩.

```
## Test the function
for inp1 in ['0', '1']:
    for inp2 in ['0', '1']:
        qc, output = NAND(inp1, inp2)
        print('NAND with inputs',inp1,inp2,'gives output',output)
        display(qc.draw())
        print('\n')
```

```
NAND with inputs 0 0 gives output 0
            ░  ░    
q_0: ─|0>───░──░────
            ░  ░    
q_1: ─|0>───░──░────
            ░  ░ ┌─┐
q_2: ─|0>───░──░─┤M├
            ░  ░ └╥┘
c: 1/═════════════╩═
                  0 


NAND with inputs 0 1 gives output 0
                    ░  ░    
q_0: ─|0>───────────░──░────
         ┌───┐      ░  ░    
q_1: ─|0>┤ X ├──────░──░────
         └───┘      ░  ░ ┌─┐
q_2: ─|0>───────────░──░─┤M├
                    ░  ░ └╥┘
c: 1/═════════════════════╩═
                          0 


NAND with inputs 1 0 gives output 0
         ┌───┐      ░  ░    
q_0: ─|0>┤ X ├──────░──░────
         └───┘      ░  ░    
q_1: ─|0>───────────░──░────
                    ░  ░ ┌─┐
q_2: ─|0>───────────░──░─┤M├
                    ░  ░ └╥┘
c: 1/═════════════════════╩═
                          0 


NAND with inputs 1 1 gives output 0
         ┌───┐      ░  ░    
q_0: ─|0>┤ X ├──────░──░────
         ├───┤      ░  ░    
q_1: ─|0>┤ X ├──────░──░────
         └───┘      ░  ░ ┌─┐
q_2: ─|0>───────────░──░─┤M├
                    ░  ░ └╥┘
c: 1/═════════════════════╩═
                          0 
```

OR GATE

```
def OR(inp1,inp2):
    """An OR gate.

    Parameters:
        inpt1 (str): Input 1, encoded in qubit 0.
        inpt2 (str): Input 2, encoded in qubit 1.

    Returns:
        QuantumCircuit: Output XOR circuit.
        str: Output value measured from qubit 2.
    """

    qc = QuantumCircuit(3, 1)
    qc.reset(range(3))

    if inp1=='1':
        qc.x(0)
    if inp2=='1':
        qc.x(1)

    qc.barrier()

    # this is where your program for quantum OR gate goes






    qc.barrier()
    qc.measure(2, 0) # output from qubit 2 is measured

    # We'll run the program on a simulator
    backend = Aer.get_backend('aer_simulator')
    # Since the output will be deterministic, we can use just a single shot to get it
    job = backend.run(qc,shots=1,memory=True)
    output = job.result().get_memory()[0]

    return qc, output
```

The OR function defines a quantum circuit for an OR gate that takes in two inputs and outputs a single bit. The function takes in two parameters, inp1 and inp2, which are the binary representations of the two input bits. The circuit consists of three qubits and one classical bit, with inp1 and inp2 encoded on qubits 0 and 1, respectively. The function then applies quantum gates to these qubits to implement the OR operation, which is where the program for the quantum OR gate goes. Finally, the function measures the output on qubit 2 and returns the quantum circuit and the output value as a string.

```
## Test the function
for inp1 in ['0', '1']:
    for inp2 in ['0', '1']:
        qc, output = OR(inp1, inp2)
        print('OR with inputs',inp1,inp2,'gives output',output)
        display(qc.draw())
        print('\n')
```

```
OR with inputs 0 0 gives output 0
            ░  ░    
q_0: ─|0>───░──░────
            ░  ░    
q_1: ─|0>───░──░────
            ░  ░ ┌─┐
q_2: ─|0>───░──░─┤M├
            ░  ░ └╥┘
c: 1/═════════════╩═
                  0 


OR with inputs 0 1 gives output 0
                      ░  ░    
q_0: ─|0>─────────────░──░────
              ┌───┐   ░  ░    
q_1: ─|0>─────┤ X ├───░──░────
              └───┘   ░  ░ ┌─┐
q_2: ─|0>─────────────░──░─┤M├
                      ░  ░ └╥┘
c: 1/═══════════════════════╩═
                            0 


OR with inputs 1 0 gives output 0
              ┌───┐   ░  ░    
q_0: ─|0>─────┤ X ├───░──░────
              └───┘   ░  ░    
q_1: ─|0>─────────────░──░────
                      ░  ░ ┌─┐
q_2: ─|0>─────────────░──░─┤M├
                      ░  ░ └╥┘
c: 1/═══════════════════════╩═
                            0 


OR with inputs 1 1 gives output 0
              ┌───┐   ░  ░    
q_0: ─|0>─────┤ X ├───░──░────
              ├───┤   ░  ░    
q_1: ─|0>─────┤ X ├───░──░────
              └───┘   ░  ░ ┌─┐
```