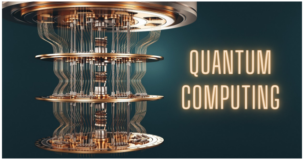
Quantum Computing for Computer Scientist

Quantum Lab
Lab-2

January 19, 2024

Faculty: **Dr. Dhiman Saha**



Recall: Last Experiment

- Single Qubit State
  - $|\psi\rangle = |0\rangle$

Recall: Last Experiment

- Single Qubit State
  - $|\psi\rangle = |0\rangle$

First Problem: 3-Qubit State

Create a State $|\psi\rangle = |000\rangle$ that contains three qubits all in the $|0\rangle$ *state*

Creating Multi-Qubit State

3-Qubit State: $|\psi\rangle = |000\rangle$

```
from qiskit import QuantumRegister, QuantumCircuit, Aer, execute
S_simulator = Aer.backends(name='statevector_simulator')[0]
q = QuantumRegister(3)
three_qubits = QuantumCircuit(q)

three_qubits.id(q[0])
three_qubits.id(q[1])
three_qubits.id(q[2])

job = execute(three_qubits, S_simulator)
result = job.result()
print(result.get_statevector())
```

Explanation for Problem-1

output : array([1.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j, 0.+0.j])

Above output indicates that first entry is in state $|000\rangle$ and has amplitude 1 and remaining are zero.From 3-qubit we can have 8 possible state and since we want each qubit to be in state $|0\rangle$ state, so just applied **Identity gate** (id(q[0]) to each one.

For clarity, the order in which states are represented above are as follows:

$[|000\rangle, |100\rangle, |010\rangle, |110\rangle, |001\rangle, |101\rangle, |011\rangle, |111\rangle]$

3-Qubit State: $|\psi\rangle = |100\rangle$

- Create State $|\psi\rangle = |100\rangle$ that contains first qubit in $|1\rangle$ state and remaining all in the $|0\rangle$ *state*
- Use NOT to first Qubit
  - Instead of "id" use "x"
  - Flipping the Qubit

Flipping a Qubit in Multi-Qubit Setting

$|\psi\rangle = |100\rangle$

```
from qiskit import QuantumRegister, QuantumCircuit, Aer, execute
S_simulator = Aer.backends(name='statevector_simulator')[0]
q = QuantumRegister(3)
three_qubits = QuantumCircuit(q)

three_qubits.x(q[0])  //Applied NOT to 1st qubit
three_qubits.id(q[1])
three_qubits.id(q[2])

job = execute(three_qubits, S_simulator)
result = job.result()
print(result.get_statevector())
```

Inclass Assignment

Problem 1

Write a Qiskit code to generate a 4-qubit quantum state, where the state "n" is determined by the second-to-last digit of your roll number. For instance, if your roll number is 11810290, set n to be 9. In cases where the second-to-last digit is zero, assign n the value of 13.

Important Points

1. Not easy to represent when dealt with larger systems
2. What's alternative we can use?
3. Let's import and use function : **Wavefunction**

Solution

```
from qiskit import QuantumRegister, QuantumCircuit, Aer, execute
import Our_Qiskit_Functions as oq
S_simulator = Aer.backends(name='statevector_simulator')[0]
q = QuantumRegister(3)
three_qubits = QuantumCircuit(q)

three_qubits.x(q[0])
three_qubits.id(q[1])
three_qubits.id(q[2])
oq.Wavefunction(three_qubits)
```

Outputs : $1.0\ |100\rangle$

Error!

No Module Found

Error!

No Module Found

Inclass Assignment: Problem 2

Create Qiskit code named "Wavefunction" in the additional Python file "Our_Qiskit_Functions" to display the qubit state, as demonstrated previously.

Our First Superposition State

H (Hadamard Gate) : H-gate takes a qubit and splits into 50 - 50 probability distribution between state $|0\rangle$ and $|1\rangle$

$$H\ |0\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + |1\rangle\right) \qquad H\ |1\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle - |1\rangle\right)$$

which is aacomplished by the following matrix: $\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$

The Code describing above representation

```
from qiskit import QuantumRegister, QuantumCircuit, Aer, execute
import Our_Qiskit_Functions as oq
q = QuantumRegister(1)
H_circuit = QunatumCircuit(q)
H_circuit.h(q[0])
oq.Wavefunction(H.Circuit)

//Output is:  0.70711 |0>   0.70711 |1>
So, finally our quibit is in superposition state!
```

Problems-3 : Create a Superposition State of 2 Qubits

H (Hadamard 2-Qubit Superposition state)

```
from qiskit import QuantumRegister, QuantumCircuit, Aer, execute
S_simulator = Aer.backends(name='statevector_simulator')[0]
q = QuantumRegister(2)
qubit = QuantumCircuit(q)
qubit.h(q[0])
qubit.h(q[1])
job = execute(qubit, S_simulator)
result = job.result()
print(result.get_statevector)
```

Guess the output.....??

Give mathematical explanation of your output !

Thank You!