通信實驗 Lab 1

B07901103 電機三 陳孟宏

<Prob 1> Installing the Qiskit

(a)(b)(c) Install the qiskit

```
In [1]: from qiskit import IBMQ
        apitoken = 'cdad9ef01fbad88f5ee99fa7fc4cfb7b5eb8755c52fd1697690c6b680797a57384a80d5a6bfd5e99332b8ee8a58e74e3903034a7f4c19eb9d29b9
        IBMQ.enable_account(apitoken)

        C:\Users\user\Downloads\anaconda\envs\IBMQ\lib\site-packages\qiskit\providers\ibmq\ibmqfactory.py:109: UserWarning: Timestamps
        in IBMQ backend properties, jobs, and job results are all now in local time instead of UTC.
          warnings.warn('Timestamps in IBMQ backend properties, jobs, and job results '

Out[1]: <AccountProvider for IBMQ(hub='ibm-q', group='open', project='main')>
```

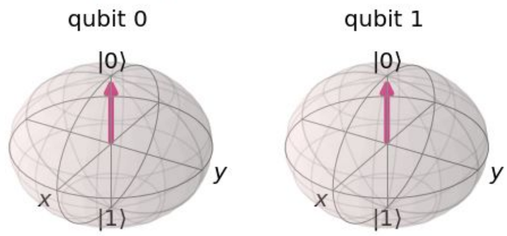
(d) Run the example code (circuit.draw())

```
In [2]: import qiskit
        import numpy as np
        from qiskit import QuantumCircuit, execute, Aer
        from qiskit.visualization import plot_histogram

        # Create a quantum circuit acting on the q register
        circuit = QuantumCircuit(2, 2)

        # Add a H gate on qubit 0
        circuit.h(0)

        # Add a CX (CNOT) gate on control qubit 0 and target qubit 1
        circuit.cx(0, 1)

        # Map the quantum measurement to the classical bits
        circuit.measure([0, 1], [0, 1])

        # Use Aer's qasm_simulator
        simulator = Aer.get_backend('qasm_simulator')

        # Execute the circuit on the qasm simulator
        job = execute(circuit, simulator, shots=10000)

        # Grab results from the job
        result = job.result()

        # Returns counts
        counts = result.get_counts(circuit)
        print("\nTotal count for 00 and 11 are: ", counts)

        # Draw the circuit
        circuit.draw()

        Total count for 00 and 11 are:  {'00': 4945, '11': 5055}

Out[2]:      ┌───┐     ┌─┐   
        q_0: ┤ H ├──■──┤M├───
             └───┘┌─┴─┐└╥┘┌─┐
        q_1: ─────┤ X ├─╫─┤M├
                  └───┘ ║ └╥┘
        c: 2/═══════════╩══╩═
                        0  1 
```

<Prob 2> Manipulating a single qubit state

(a)(b)(c) Run the example code (plot_bloch_multivector())

```
In [3]: # Firstly, we import some libraries
        from qiskit import *
        import numpy as np
        from qiskit import QuantumCircuit, execute, Aer
        from qiskit.visualization import plot_histogram, plot_bloch_vector, plot_bloch_multivector
        from math import sqrt, pi

        # Secondly, we create two qubits and one classical bits
        q = QuantumRegister(2)
        c = ClassicalRegister(2)

        # Thirdly, using the registers created above to create a quantum circuit
        circuit = QuantumCircuit(q, c)

        # optional, verify qubit state by using the following commands to get the vector form
        simulator = Aer.get_backend('statevector_simulator')
        job = execute(circuit, simulator)
        result = job.result()
        statevector = result.get_statevector()
        print(statevector)
        plot_bloch_multivector(statevector)

        [1.+0.j 0.+0.j 0.+0.j 0.+0.j]
Out[3]:
```



(d) Run the example code (simulator())

- Default 的 qubit 為 state |0>, 所以沒輸入兩個 qubits 沒經過任何閘, 輸出依然全為 |00>

```
In [4]: # Firstly, we import some libraries
        from qiskit import *
        import numpy as np
        from qiskit import QuantumCircuit, execute, Aer
        from qiskit.visualization import plot_histogram, plot_bloch_vector, plot_bloch_multivector
        from math import sqrt, pi

        # Secondly, we create two qubits and one classical bits
        q = QuantumRegister(2)
        c = ClassicalRegister(2)

        # Thirdly, using the registers created above to create a quantum circuit
        circuit = QuantumCircuit(q, c)

        # optional, measuring all the states with respect to the computational basis { |0⟩, |1⟩ }
        circuit.measure(q, c)
        simulator = Aer.get_backend('qasm_simulator')
        job = execute(circuit, simulator, shots=1024)
        result = job.result()
        counts = result.get_counts()
        print(counts)

        {'00': 1024}
```

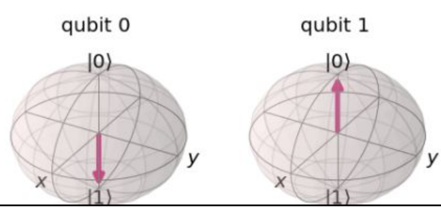
(e) X-gate

- circuit.draw()

```
In [7]: # Problem2-1, 4 %
        # Import some libraries
        from qiskit import *
        import numpy as np
        from qiskit import QuantumCircuit, execute, Aer
        from qiskit.visualization import plot_histogram, plot_bloch_vector, plot_bloch_multivector
        from math import sqrt, pi

        # Create one qubits and one classical bits
        q = QuantumRegister(2)
        c = ClassicalRegister(2)

        # Create a quantum circuit
        circuit = QuantumCircuit(q, c)
        circuit.x(q[0])
        #circuit.y(q[0])
        #circuit.z(q[0])

        # Draw
        circuit.draw()

        # Simulate > [0.+0.j 1.+0.j 0.+0.j 0.+0.j]
        #simulator = Aer.get_backend('statevector_simulator')
        #job = execute(circuit, simulator)
        #result = job.result()
        #statevector = result.get_statevector()
        #print(statevector)
        #plot_bloch_multivector(statevector)

        # Measure > {'01': 1024}
        #circuit.measure(q, c)
        #simulator = Aer.get_backend('qasm_simulator')
        #job = execute(circuit, simulator, shots=1024)
        #result = job.result()
        #counts = result.get_counts()
        #print(counts)

Out[7]:
        q9_0: ┤ X ├
        q9_1: ─────
        c4: 2/═════
```

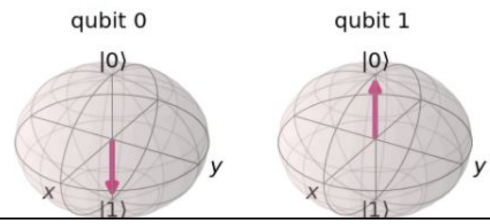
- print state / plot_bloch_multivector(), |00> (default) → |01>

```
In [8]: # Problem2-1, 4 %
        # Import some libraries
        from qiskit import *
        import numpy as np
        from qiskit import QuantumCircuit, execute, Aer
        from qiskit.visualization import plot_histogram, plot_bloch_vector, plot_bloch_multivector
        from math import sqrt, pi

        # Create one qubits and one classical bits
        q = QuantumRegister(2)
        c = ClassicalRegister(2)

        # Create a quantum circuit
        circuit = QuantumCircuit(q, c)
        circuit.x(q[0])
        #circuit.y(q[0])
        #circuit.z(q[0])

        # Draw
        #circuit.draw()

        # Simulate > [0.+0.j 1.+0.j 0.+0.j 0.+0.j]
        simulator = Aer.get_backend('statevector_simulator')
        job = execute(circuit, simulator)
        result = job.result()
        statevector = result.get_statevector()
        print(statevector)
        plot_bloch_multivector(statevector)

        # Measure > {'01': 1024}
        #circuit.measure(q, c)
        #simulator = Aer.get_backend('qasm_simulator')
        #job = execute(circuit, simulator, shots=1024)
        #result = job.result()
        #counts = result.get_counts()
        #print(counts)

        [0.+0.j 1.+0.j 0.+0.j 0.+0.j]

Out[8]:
```



- The result of “circuit.i(q[1])” is the same as “default q[1]”

```
In [9]: # Problem2-1, 4 %
        # Import some libraries
        from qiskit import *
        import numpy as np
        from qiskit import QuantumCircuit, execute, Aer
        from qiskit.visualization import plot_histogram, plot_bloch_vector, plot_bloch_multivector
        from math import sqrt, pi

        # Create one qubits and one classical bits
        q = QuantumRegister(2)
        c = ClassicalRegister(2)

        # Create a quantum circuit
        circuit = QuantumCircuit(q, c)
        circuit.x(q[0])
        #circuit.y(q[0])
        #circuit.z(q[0])
        circuit.i(q[1])

        # Draw
        #circuit.draw()

        # Simulate > [0.+0.j 1.+0.j 0.+0.j 0.+0.j]
        simulator = Aer.get_backend('statevector_simulator')
        job = execute(circuit, simulator)
        result = job.result()
        statevector = result.get_statevector()
        print(statevector)
        plot_bloch_multivector(statevector)

        # Measure > {'01': 1024}
        #circuit.measure(q, c)
        #simulator = Aer.get_backend('qasm_simulator')
        #job = execute(circuit, simulator, shots=1024)
        #result = job.result()
        #counts = result.get_counts()
        #print(counts)

        [0.+0.j 1.+0.j 0.+0.j 0.+0.j]
Out[9]:
```



- Measure |00> → |01>

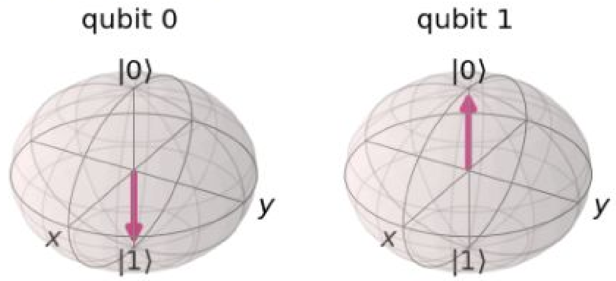
```
In [10]: # Problem2-1, 4 %
         # Import some libraries
         from qiskit import *
         import numpy as np
         from qiskit import QuantumCircuit, execute, Aer
         from qiskit.visualization import plot_histogram, plot_bloch_vector, plot_bloch_multivector
         from math import sqrt, pi

         # Create one qubits and one classical bits
         q = QuantumRegister(2)
         c = ClassicalRegister(2)

         # Create a quantum circuit
         circuit = QuantumCircuit(q, c)
         circuit.x(q[0])
         #circuit.y(q[0])
         #circuit.z(q[0])
         circuit.i(q[1])

         # Draw
         #circuit.draw()

         # Simulate > [0.+0.j 1.+0.j 0.+0.j 0.+0.j]
         #simulator = Aer.get_backend('statevector_simulator')
         #job = execute(circuit, simulator)
         #result = job.result()
         #statevector = result.get_statevector()
         #print(statevector)
         #plot_bloch_multivector(statevector)

         # Measure > {'01': 1024}
         circuit.measure(q, c)
         simulator = Aer.get_backend('qasm_simulator')
         job = execute(circuit, simulator, shots=1024)
         result = job.result()
         counts = result.get_counts()
         print(counts)

         {'01': 1024}
```

- The result of “Y-gate”
  - circuit.draw()

```
Out[11]:
          ┌───┐
  q19_0: ─┤ Y ├─
          └───┘
  q19_1: ───────

   c8: 2/═══════
```

  - simulate(),

```
[0.-0.j 0.+1.j 0.-0.j 0.+0.j]
Out[12]:
```



  - measure()

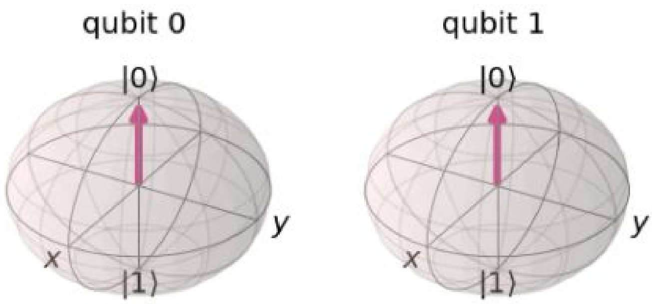
```
{'01': 1024}
```

- The result of “Z-gate”
  - circuit.draw()

```
Out[14]:
          ┌───┐
  q25_0: ─┤ Z ├─
          └───┘
  q25_1: ───────

  c11: 2/═══════
```

  - simulate()

```
[ 1.+0.j -0.+0.j  0.+0.j -0.+0.j]
Out[15]:
```



  - measure()

```
{'00': 1024}
```

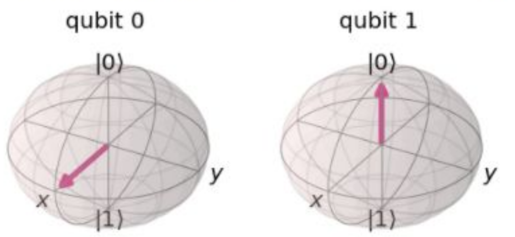
(f) H-gate

- plot_histogram / measure, we will get the superposition state of |0> and |1>, |00> : |01> = 1 : 1 (only apply H-gate on q[0])

In [20]:
```
# Problem2-2, 4 %
# Import some libraries
from qiskit import *
import numpy as np
from qiskit import QuantumCircuit, execute, Aer
from qiskit.visualization import plot_histogram, plot_bloch_vector, plot_bloch_multivector
from math import sqrt, pi

# Create one qubits and one classical bits
q = QuantumRegister(2)
c = ClassicalRegister(2)

# Create a quantum circuit
circuit = QuantumCircuit(q, c)
circuit.h(q[0])
#circuit.h(q[0])

# Draw
#circuit.draw()

# Simulate > [0.+0.j 1.+0.j 0.+0.j 0.+0.j]
simulator = Aer.get_backend('statevector_simulator')
job = execute(circuit, simulator)
result = job.result()
statevector = result.get_statevector()
print(statevector)
plot_bloch_multivector(statevector)

# Measure > {'01': 1024}
#circuit.measure(q, c)
#simulator = Aer.get_backend('qasm_simulator')
#job = execute(circuit, simulator, shots=1024)
#result = job.result()
#counts = result.get_counts()
#print(counts)
#plot_histogram(counts)
```

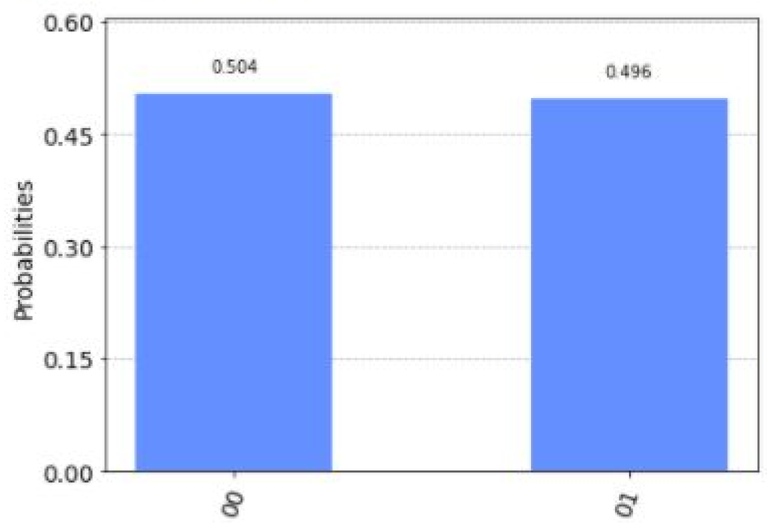
```
[0.70710678+0.j 0.70710678+0.j 0.        +0.j 0.        +0.j]
```

Out[20]:




```
{'00': 516, '01': 508}
```

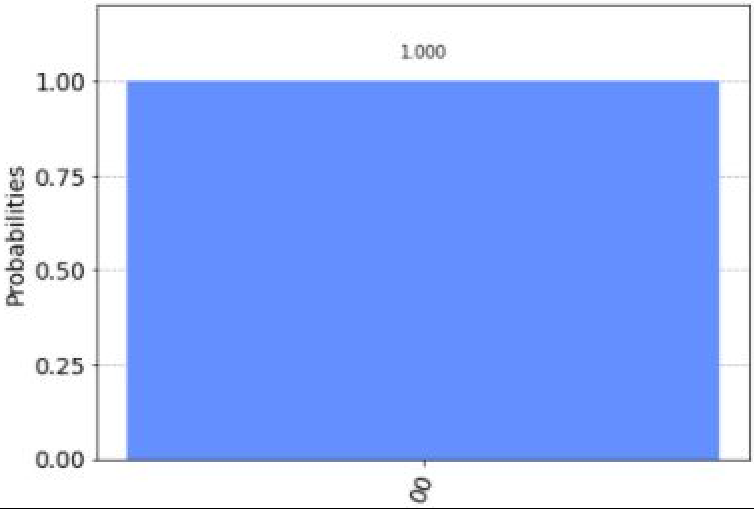
Out[21]:



- H*H = I, |00> → |0>*|+> (H-gate on q[0]) → |00> (H-gate on q[0] again)

```
{'00': 1024}
```

Out[22]:



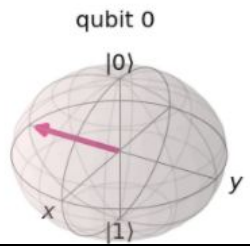
(g) Rx-gate (rotate along x-axis)

- plot on the bloch sphere

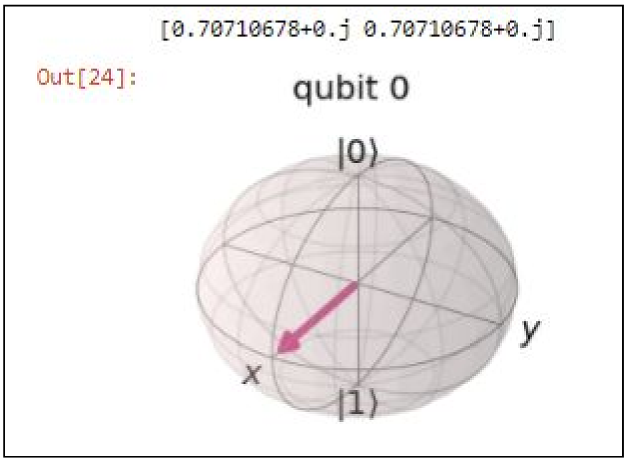
In [23]:
```
# Problem2-3, 4 %
# Import some libraries
from qiskit import *
import numpy as np
from qiskit import QuantumCircuit, execute, Aer
from qiskit.visualization import plot_histogram, plot_bloch_vector, plot_bloch_multivector
from math import sqrt, pi

# Create one qubits and one classical bits
q = QuantumRegister(1)
c = ClassicalRegister(1)

# Create a quantum circuit
circuit = QuantumCircuit(q, c)
circuit.rx(pi/2, q[0])
#circuit.ry(pi/2, q[0])
#circuit.rz(pi/2, q[0])
#circuit.u3(pi/4, pi/2, 1, q[0])

# Draw
#circuit.draw()

# Simulate > [0.+0.j 1.+0.j 0.+0.j 0.+0.j]
simulator = Aer.get_backend('statevector_simulator')
job = execute(circuit, simulator)
result = job.result()
statevector = result.get_statevector()
print(statevector)
plot_bloch_multivector(statevector)

# Measure > {'01': 1024}
#circuit.measure(q, c)
#simulator = Aer.get_backend('qasm_simulator')
#job = execute(circuit, simulator, shots=1024)
#result = job.result()
#counts = result.get_counts()
#print(counts)
```

```
[7.07106781e-01+0.j         4.32978028e-17-0.70710678j]
```

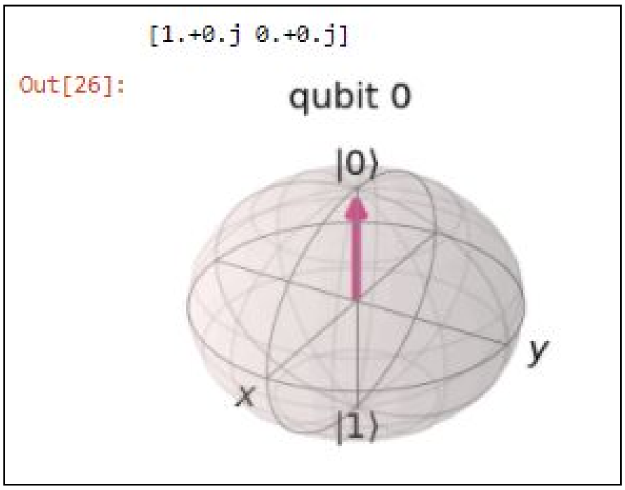
Out[23]:



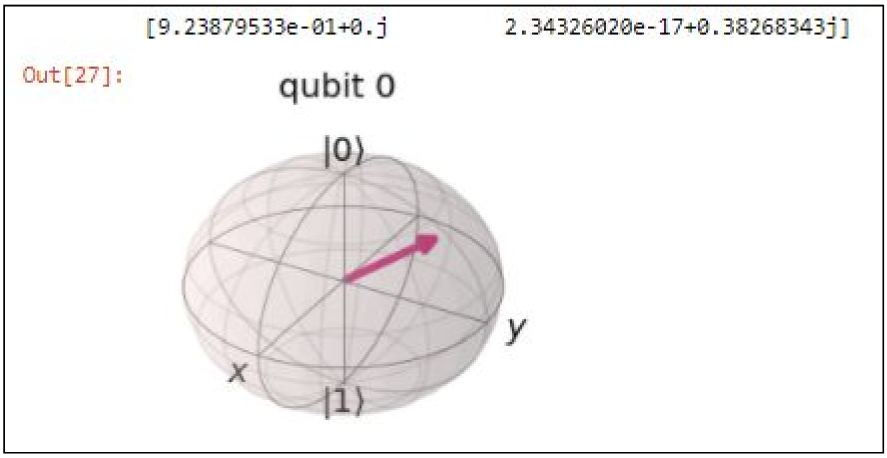
- “Ry-gate”



- “Rz-gate”



- “U3-gate” (arbitrary state) **(circuit.u3(pi/4, pi/2, 1, q[0]) here)**



(h) Probability state

- First, rotate with ‘x’ or ‘y’ axis to make |0> : |1> = 2 : 1

- The quantum R^Z_φ gate rotates φ around the z-axis $R^Z_\varphi := \begin{pmatrix} 1 & 0 \\ 0 & e^{i\varphi} \end{pmatrix}$
- The quantum R^X_φ gate rotates φ around the x-axis $R^X_\varphi := \begin{pmatrix} \cos(\frac{\varphi}{2}) & -i\sin(\frac{\varphi}{2}) \\ -i\sin(\frac{\varphi}{2}) & \cos(\frac{\varphi}{2}) \end{pmatrix}$
- The quantum R^Y_φ gate rotates φ around the y-axis $R^Y_\varphi := \begin{pmatrix} \cos(\frac{\varphi}{2}) & -\sin(\frac{\varphi}{2}) \\ \sin(\frac{\varphi}{2}) & \cos(\frac{\varphi}{2}) \end{pmatrix}$

- Using “Born Rule”, the probability is ⅔

- The Born rule: $|\psi\rangle = a|0\rangle + b|1\rangle$

Probability amplitude

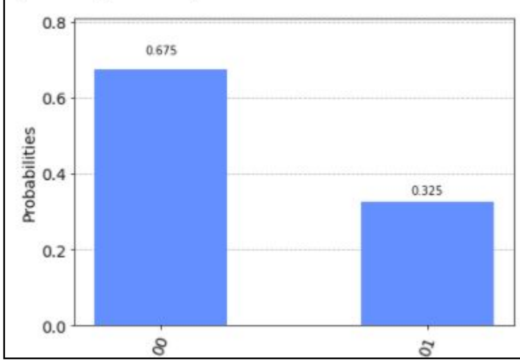
$$\Pr(0) = |\langle 0|\psi\rangle|^2 = \left| \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} \right|^2 = |a|^2$$

$$\Pr(1) = |\langle 0|\psi\rangle|^2 = \left| \begin{pmatrix} 0 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} \right|^2 = |b|^2$$

- We can get if angle = acos(1/3), we can make |0> appear with the probability of ⅔, then concatenate them with H-gate.

```
# Problem2-4, 8 %
# Import some libraries
from qiskit import *
import numpy as np
from qiskit import QuantumCircuit, execute, Aer
from qiskit.visualization import plot_histogram, plot_bloch_vector, plot_bloch_multivector
from math import sqrt, pi, acos

# Create one qubits and one classical bits
q = QuantumRegister(2)
c = ClassicalRegister(2)

# Create a quantum circuit
circuit = QuantumCircuit(q, c)
circuit.rx(acos(1/3), q[0])
circuit.i(q[1])

# Draw
#circuit.draw()

# Simulate > [0.+0.j 1.+0.j 0.+0.j 0.+0.j]
#simulator = Aer.get_backend('statevector_simulator')
#job = execute(circuit, simulator)
#result = job.result()
#statevector = result.get_statevector()
#print(statevector)
#plot_bloch_multivector(statevector)

# Measure > {'01': 1024}
circuit.measure(q, c)
simulator = Aer.get_backend('qasm_simulator')
job = execute(circuit, simulator, shots=1024)
result = job.result()
counts = result.get_counts()
print(counts)
plot_histogram(counts)
```

```
{'00': 691, '01': 333}
```

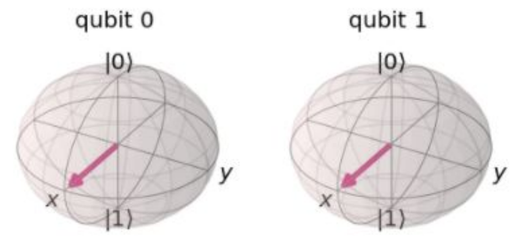


<Prob 3> Manipulating Multi-Qubit gates

(a) Prepare a superposition state

```
In [28]: # Problem3-1, 4 %
         # Import some libraries
         from qiskit import *
         import numpy as np
         from qiskit import QuantumCircuit, execute, Aer
         from qiskit.visualization import plot_histogram, plot_bloch_vector, plot_bloch_multivector
         from math import sqrt, pi

         # Create one qubits and one classical bits
         q = QuantumRegister(2)
         c = ClassicalRegister(2)

         # Create a quantum circuit
         circuit = QuantumCircuit(q, c)
         for qubit in range(2):
             circuit.h(qubit)

         # Draw
         #circuit.draw()

         # Simulate > [0.+0.j 1.+0.j 0.+0.j 0.+0.j]
         simulator = Aer.get_backend('statevector_simulator')
         job = execute(circuit, simulator)
         result = job.result()
         statevector = result.get_statevector()
         print(statevector)
         plot_bloch_multivector(statevector)

         # Measure > {'01': 1024}
         #circuit.measure(q, c)
         #simulator = Aer.get_backend('qasm_simulator')
         #job = execute(circuit, simulator, shots=1024)
         #result = job.result()
         #counts = result.get_counts()
         #print(counts)

         [0.5+0.j 0.5+0.j 0.5+0.j 0.5+0.j]
```

Out[28]:




```
{'00': 249, '01': 249, '10': 251, '11': 275}
```

```
Out[31]:
         q61_0: ┤ H ├
         q61_1: ┤ H ├
         c28: 2/═════
```

(b) Entangled state

In [ ]:
```python
# Problem3-2, 7 %
# Import some libraries
from qiskit import *
import numpy as np
from qiskit import QuantumCircuit, execute, Aer
from qiskit.visualization import plot_histogram, plot_bloch_vector, plot_bloch_multivector
from math import sqrt, pi

# Create one qubits and one classical bits
q = QuantumRegister(2)
c = ClassicalRegister(2)

# Create a quantum circuit
circuit = QuantumCircuit(q, c)

# |Φ+⟩ = CNOT· H1(|00⟩)
circuit.h(q[0])
circuit.cx(q[0], q[1])

# |Φ-⟩ = X1· CNOT· H1(|00⟩)
#circuit.h(q[0])
#circuit.cx(q[0], q[1])
#circuit.z(q[0])

# |Ψ+⟩ = X2· CNOT· H1(|00⟩)
#circuit.h(q[0])
#circuit.cx(q[0], q[1])
#circuit.x(q[1])

# |Ψ-⟩ = Z1(|Ψ+⟩) = Z1· X2· CNOT· H1(|00⟩)
#circuit.h(q[0])
#circuit.cx(q[0], q[1])
#circuit.z(q[0])
#circuit.x(q[1])

# Draw
#circuit.draw()

# Simulate > [0.+0.j 1.+0.j 0.+0.j 0.+0.j]
#simulator = Aer.get_backend('statevector_simulator')
#job = execute(circuit, simulator)
#result = job.result()
#statevector = result.get_statevector()
#print(statevector)
#plot_bloch_multivector(statevector)

# Measure > {'01': 1024}
circuit.measure(q, c)
simulator = Aer.get_backend('qasm_simulator')
job = execute(circuit, simulator, shots=1024)
result = job.result()
counts = result.get_counts()
print(counts)
```

- **|Φ+⟩ = CNOT · H1|00⟩, if Alice = ‘1’, Bob = ‘1’**

```
Out[32]:
         ┌───┐
q62_0: ──┤ H ├──■──
         └───┘┌─┴─┐
q62_1: ───────┤ X ├
              └───┘
c29: 2/════════════
```

```
[0.70710678+0.j 0.         +0.j 0.         +0.j 0.70710678+0.j]
```

```
{'00': 516, '11': 508}
```

- **|Φ−⟩ = Z1|Φ+⟩ = Z1 · CNOT · H1|00⟩, if Alice = ‘1’, Bob = ‘1’**

```
Out[35]:
        ┌───┐     ┌───┐
q70_0: ─┤ H ├──■──┤ Z ├
        └───┘┌─┴─┐└───┘
q70_1: ──────┤ X ├─────
             └───┘
c32: 2/════════════════
```

```
[ 0.70710678+0.j -0.         +0.j  0.         +0.j -0.70710678+0.j]
```

```
{'00': 511, '11': 513}
```

- $|\Psi+\rangle = X2|\Phi+\rangle = X2 \cdot CNOT \cdot H1|00\rangle$, if Alice = ‘1’, Bob = ‘0’

```
Out[36]:
         q71_0: ┤ H ├──■───────
                     ┌─┴─┐┌───┐
         q71_1: ─────┤ X ├┤ X ├
                     └───┘└───┘
         c33: 2/═══════════════
```

```
[0.        +0.j 0.70710678+0.j 0.70710678+0.j 0.        +0.j]
```

```
{'01': 518, '10': 506}
```

- $|\Psi-\rangle = Z1|\Psi+\rangle = Z1 \cdot X2 \cdot CNOT \cdot H1|00\rangle$, if Alice = ‘1’, Bob = ‘0’

```
Out[37]:
         q72_0: ┤ H ├──■──┤ Z ├
                     ┌─┴─┐┌───┐
         q72_1: ─────┤ X ├┤ X ├
                     └───┘└───┘
         c34: 2/═══════════════
```

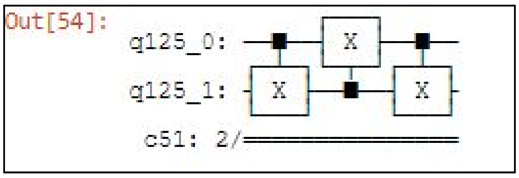
```
[ 0.        +0.j -0.70710678+0.j  0.70710678+0.j -0.        +0.j]
```

```
{'01': 529, '10': 495}
```

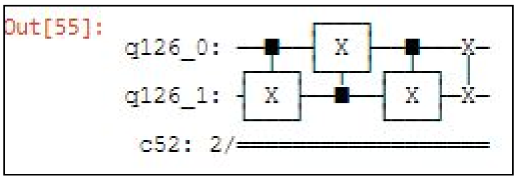
(c) Swap gate

```
# Problem3-3, 4 %
# Import some libraries
from qiskit import *
import numpy as np
from qiskit import QuantumCircuit, execute, Aer
from qiskit.visualization import plot_histogram, plot_bloch_vector, plot_bloch_multivector
from math import sqrt, pi

# Create one qubits and one classical bits
q = QuantumRegister(2)
c = ClassicalRegister(2)

# Create a quantum circuit
circuit = QuantumCircuit(q, c)

# |00> --> |00>
circuit.swap(q[0], q[1])

# |01> --> |10>
#circuit.x(q[0])
#circuit.swap(q[0], q[1])

# |10> --> |01>
#circuit.x(q[1])
#circuit.swap(q[0], q[1])

# |11> --> |11>
#circuit.x(q[0])
#circuit.x(q[1])
#circuit.swap(q[0], q[1])

# check (q=1, CNOT rotate the other bit ; q=0, CNOT does not work)
#circuit.cx(q[0], q[1])
#circuit.cx(q[1], q[0])
#circuit.cx(q[0], q[1])

# Draw
#circuit.draw()

# Simulate > [0.+0.j 1.+0.j 0.+0.j 0.+0.j]
#simulator = Aer.get_backend('statevector_simulator')
#job = execute(circuit, simulator)
#result = job.result()
#statevector = result.get_statevector()
#print(statevector)
#plot_bloch_multivector(statevector)

# Measure > {'01': 1024}
circuit.measure(q, c)
simulator = Aer.get_backend('qasm_simulator')
job = execute(circuit, simulator, shots=1024)
result = job.result()
counts = result.get_counts()
print(counts)
```

- Equivalent Circuit, check by concatenating with swap gate, every state will not change.



(equivalent circuit)



(check circuit)

  - |00> → |00>

    {'00': 1024}

  - |01> → |01>

    {'01': 1024}

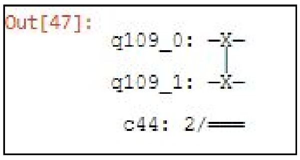
  - |10> → |10>

    {'10': 1024}

  - |11> → |11>

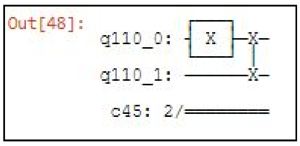
    {'11': 1024}

- |00> → |00>



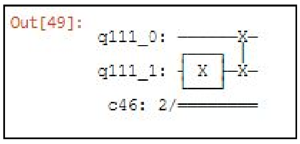
{'00': 1024}

- |01> → |10>



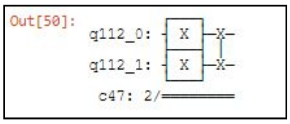
{'10': 1024}

- |10> → |01>



```
{'01': 1024}
```

- |11> → |11>




```
{'11': 1024}
```

<Prob 4> The swap test

```
# Problem4, 20 %
# Import some libraries
from qiskit import *
import numpy as np
from qiskit import QuantumCircuit, execute, Aer
from qiskit.visualization import plot_histogram, plot_bloch_vector, plot_bloch_multivector
from math import sqrt, pi

# Create one qubits and one classical bits
q = QuantumRegister(3)
c = ClassicalRegister(3)

# Create a quantum circuit
circuit = QuantumCircuit(q, c)

# states are the same --> output: '100'
circuit.h(q[2])
circuit.cswap(q[2], q[1], q[0])
circuit.h(q[2])
circuit.x(q[2])

# states are different (not orthogonal) --> output: '000' : '100' = 1 : 3
#circuit.h(q[1])
#circuit.h(q[2])
#circuit.cswap(q[2], q[1], q[0])
#circuit.h(q[2])
#circuit.x(q[2])

# states are different (orthogonal) --> output: '000' : '100' = 1 : 1
#circuit.x(q[1])
#circuit.h(q[2])
#circuit.cswap(q[2], q[1], q[0])
#circuit.h(q[2])
#circuit.x(q[2])

# Draw
#circuit.draw()

# Simulate > [0.+0.j 1.+0.j 0.+0.j 0.+0.j]
#simulator = Aer.get_backend('statevector_simulator')
#job = execute(circuit, simulator)
#result = job.result()
#statevector = result.get_statevector()
#print(statevector)
#plot_bloch_multivector(statevector)

# Measure > {'01': 1024}
circuit.measure(q[2], c[2])
simulator = Aer.get_backend('qasm_simulator')
job = execute(circuit, simulator, shots=1024)
result = job.result()
counts = result.get_counts()
print(counts)
```

- **If states are the same, (q[2], q[1], q[0]) = (1, 0, 0), q[2] will always output ‘1’, so we cannot tell what qubit state it is when they are the same.**

```
{'100': 1024}
```

  - (q[0], q[1]) = (|0>, |0>)

```
# states are the same --> output: '100'
circuit.h(q[2])
circuit.cswap(q[2], q[1], q[0])
circuit.h(q[2])
circuit.x(q[2])
```

  - (q[0], q[1]) = (|1>, |1>)

```
# states are the same --> output: '100'
circuit.x(q[0])
circuit.x(q[1])
circuit.h(q[2])
circuit.cswap(q[2], q[1], q[0])
circuit.h(q[2])
circuit.x(q[2])
```

  - (q[0], q[1]) = (|+>, |+>)

```
# states are the same --> output: '100'
circuit.h(q[0])
circuit.h(q[1])
circuit.h(q[2])
circuit.cswap(q[2], q[1], q[0])
circuit.h(q[2])
circuit.x(q[2])
```

  - (q[0], q[1]) = (|->, |->)

```
# states are the same --> output: '100'
circuit.x(q[0])
circuit.h(q[0])
circuit.x(q[1])
circuit.h(q[1])
circuit.h(q[2])
circuit.cswap(q[2], q[1], q[0])
circuit.h(q[2])
circuit.x(q[2])
```

- **If states are orthogonal, (q[2], q[1], q[0]) → (0, 0, 0) : (1, 0, 0) = 1 : 3, q[2]’s state → ‘0’ : ‘1’ = 1 : 3**
  - (q[0], q[1]) = (|0>, |1>)

```
# states are different (orthogonal) --> output: '000' : '100' = 1 : 1
circuit.x(q[1])
circuit.h(q[2])
circuit.cswap(q[2], q[1], q[0])
circuit.h(q[2])
circuit.x(q[2])
```

```
{'000': 496, '100': 528}
```

  - (q[0], q[1]) = (|1>, |0>)

```
# states are different (orthogonal) --> output: '000' : '100' = 1 : 1
circuit.x(q[0])
circuit.h(q[2])
circuit.cswap(q[2], q[1], q[0])
circuit.h(q[2])
circuit.x(q[2])
```

```
{'000': 531, '100': 493}
```

- **If states are different and not orthogonal, (q[2], q[1], q[0]) → (0, 0, 0) : (1, 0, 0) = 1 : 1, q[2]'s state → '0' : '1' = 1 : 1**
    - (q[0], q[1]) = (|0>, |+>)

```
# states are different (not orthogonal) --> output: '000' : '100' = 1 : 3
circuit.h(q[1])
circuit.h(q[2])
circuit.cswap(q[2], q[1], q[0])
circuit.h(q[2])
circuit.x(q[2])
```

```
{'000': 253, '100': 771}
```

    - (q[0], q[1]) = (|0>, |->)

```
# states are different (not orthogonal) --> output: '000' : '100' = 1 : 3
circuit.x(q[1])
circuit.h(q[1])
circuit.h(q[2])
circuit.cswap(q[2], q[1], q[0])
circuit.h(q[2])
circuit.x(q[2])
```

```
{'000': 268, '100': 756}
```

    - (q[0], q[1]) = (|1>, |+>)

```
# states are different (not orthogonal) --> output: '000' : '100' = 1 : 3
circuit.x(q[0])
circuit.h(q[1])
circuit.h(q[2])
circuit.cswap(q[2], q[1], q[0])
circuit.h(q[2])
circuit.x(q[2])
```

```
{'000': 266, '100': 758}
```

    - (q[0], q[1]) = (|1>, |->)

```
# states are different (not orthogonal) --> output: '000' : '100' = 1 : 3
circuit.x(q[0])
circuit.x(q[1])
circuit.h(q[1])
circuit.h(q[2])
circuit.cswap(q[2], q[1], q[0])
circuit.h(q[2])
circuit.x(q[2])
```

```
{'000': 266, '100': 758}
```

- (q[0], q[1]) = (|+>, |0>)

```
# states are different (not orthogonal) --> output: '000' : '100' = 1 : 3
circuit.h(q[0])
circuit.h(q[2])
circuit.cswap(q[2], q[1], q[0])
circuit.h(q[2])
circuit.x(q[2])
```

```
{'000': 251, '100': 773}
```

- (q[0], q[1]) = (|->, |0>)

```
# states are different (not orthogonal) --> output: '000' : '100' = 1 : 3
circuit.x(q[0])
circuit.h(q[0])
circuit.h(q[2])
circuit.cswap(q[2], q[1], q[0])
circuit.h(q[2])
circuit.x(q[2])
```

```
{'000': 259, '100': 765}
```

- (q[0], q[1]) = (|+>, |1>)

```
# states are different (not orthogonal) --> output: '000' : '100' = 1 : 3
circuit.h(q[0])
circuit.x(q[1])
circuit.h(q[2])
circuit.cswap(q[2], q[1], q[0])
circuit.h(q[2])
circuit.x(q[2])
```

```
{'000': 268, '100': 756}
```

- (q[0], q[1]) = (|->, |1>)

```
# states are different (not orthogonal) --> output: '000' : '100' = 1 : 3
circuit.x(q[0])
circuit.h(q[0])
circuit.x(q[1])
circuit.h(q[2])
circuit.cswap(q[2], q[1], q[0])
circuit.h(q[2])
circuit.x(q[2])
```

```
{'000': 236, '100': 788}
```

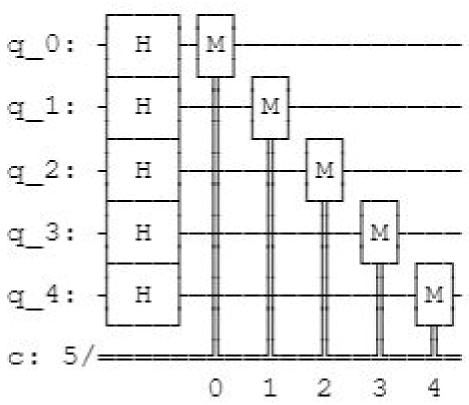
<Prob 5> QRNG

(a) For every qubit, concatenated with a H-gate, each of them will be at the superposition state of ‘0’ and ‘1’, with the probability of 50% respectively.

```python
# Problem5, 30 %, quantum random generator
# Import some libraries
from qiskit import *
import numpy as np
from qiskit import QuantumCircuit, execute, Aer
from qiskit.visualization import plot_histogram, plot_bloch_vector, plot_bloch_multivector
from math import sqrt, pi
import time

# Create one qubits and one classical bits
q = QuantumRegister(2**5, 'q')
c = ClassicalRegister(2**5, 'c')

# Create a quantum circuit
circuit = QuantumCircuit(q, c)
circuit.h(q)
circuit.measure(q, c)

# Draw
circuit.draw()

# Simulate
simulator = Aer.get_backend('qasm_simulator')

start = time.time() # start time

job = execute(circuit, simulator, shots=1)

# Print
print("Executing Job...\n")

result = job.result()
counts = result.get_counts(circuit)

print("Result (binary): ", counts, '\n')

end = time.time() # end time

print(end-start) # execute time
```

```
Executing Job...

Result (binary):  {'10110101100111000111110100111011': 1}

0.00598454475402832
```

Out[28]:



(e.g. 5-qubit circuit)