IBM Qiskit的学习

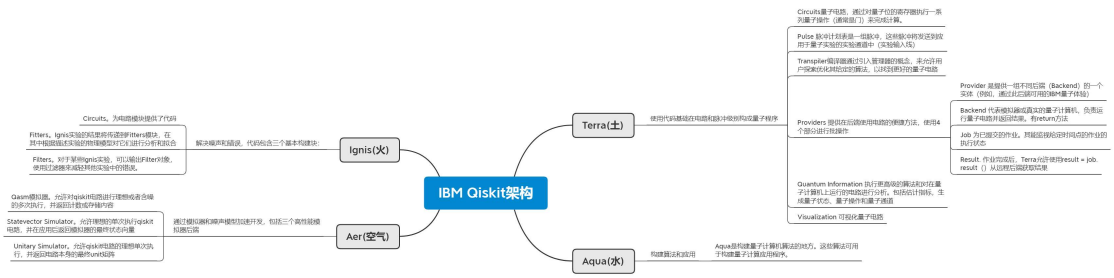
这个平台使用的最为普遍，在一些论文中也有用到，并且可用申请连接IBM的量子计算机运行

一、IBM Qiskit架构

In [164]:

```
%config InlineBackend.figure_format = 'svg' # 让图像更美观
from IPython.display import Image
Image(filename = 'E:/桌面/IBM Qiskit架构.png')
```

Out[164]:



二、基础语法

2.1.量子电路

用QuantumCircuit对象创建

In [165]:

```
from qiskit import *
qc = QuantumCircuit()
```

该电路当前完全为空，没有量子位，也没有输出

2.2量子寄存器

使用QuantumRegister对象创建的。例如，定义一个由两个量子位组成的寄存器，并将其称为qr。

In [166]:

```
qr = QuantumRegister(2,'qreg')
```

使用add_register方法将其添加到电路中，并通过检查qregs电路对象的变量来查看是否已添加它

In [167]:

```
qc.add_register( qr )
qc.qregs
```

Out[167]:

```
[QuantumRegister(2, 'qreg')]
```

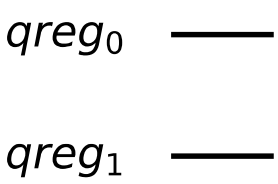
现在电路具有两个量子位，使用电路的另一个方法来查看其外观：draw()

此时两个寄存器中的状态均为 |0⟩

In [168]:

```
qc.draw(output='mpl') # mpl美化图像
```

Out[168]:



2.3添加逻辑门

2.3.1添加H门

该函数需要一个特定的qubit，前面已经定义的寄存器中有两个qubit，对应的寄存器分别为qr[0]和qr[1]。

In [169]:

```
qc.h(qr[0]) # 在寄存器qr[0]上，作用了一个H门
```

Out[169]:

```
<qiskit.circuit.instructionset.InstructionSet at 0x1b959eaca20>
```

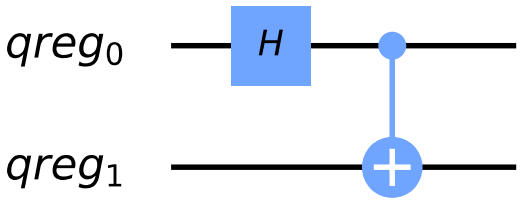
2.3.2添加受控非门 C-Not

需要两个参数：控制量子位以及目标量子位， (控制bit,目标bit)

In [170]:

```
qc.cx(qr[0], qr[1]);
qc.draw(output='mpl')
```

Out[170]:



2.4 状态向量模拟器

例如，使用statevector_simulator

In [171]:

```
vector_sim = Aer.get_backend('statevector_simulator')
```

模拟器后端（backend）指量子程序实际运行时的后台设备（可以是模拟器或真实的量子计算机）。使用Aer.backends()可以查看Aer中所有模拟器的列表

In [172]:

```
Aer.backends()
```

Out[172]:

```
[<QasmSimulator('qasm_simulator') from AerProvider()>,
 <StatevectorSimulator('statevector_simulator') from AerProvider()>,
 <UnitarySimulator('unitary_simulator') from AerProvider()>,
 <PulseSimulator('pulse_simulator') from AerProvider()>]
```

In [173]:

```
job = execute(qc, vector_sim) # 此时产生了一个处理execute的对象称为job
ket = job.result().get_statevector() # 提取运算结果
for amplitude in ket:
    print(amplitude) # 提取最终状态向量
```

```
(0.7071067811865476+0j)
0j
0j
(0.7071067811865476+0j)
```

得到bell态 $(|00\rangle + |11\rangle)/\sqrt{2}$

2.5经典寄存器和QASM模拟器

2.4中得出了运算结果的状态向量。但那不是从真正的量子计算机中得到的。实际中，需要测量来确定结果状态。为了进行测量，在2.4电路的基础上再定义一个两位经典寄存器。

In [174]:

```
cr = ClassicalRegister(2,'creg') # 创建经典寄存器
qc.add_register(cr) # 将经典寄存器加到之前的量子电路中
```

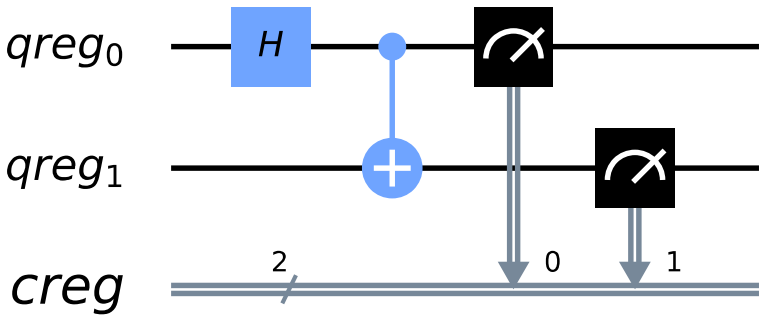
现在再使用measure方法，该方法需要两个参数：需要测量的qubit以及要将要写入结果的经典位。

下面代码，测量了两个量子位，并将它们的结果写入不同的经典位。

In [175]:

```
qc.measure(qr[0],cr[0])
qc.measure(qr[1],cr[1])
qc.draw(output='mpl')
```

Out[175]:



现在，可以在本地运行此模拟器，其效果是模拟真实的量子设备。为此，需要向execute函数中添加另一个输入shots，它确定运行电路的次数。如果不提供任何shots值，则默认值为1024。

In [176]:

```
emulator = Aer.get_backend('qasm_simulator') # 创建模拟器
job = execute( qc, emulator, shots=8192 ) # 以次模拟器运行电路8192次
```

结果本质上是Python字典形式的直方图。可以用print来展示

In [177]:

```
hist = job.result().get_counts()
print(hist)
```

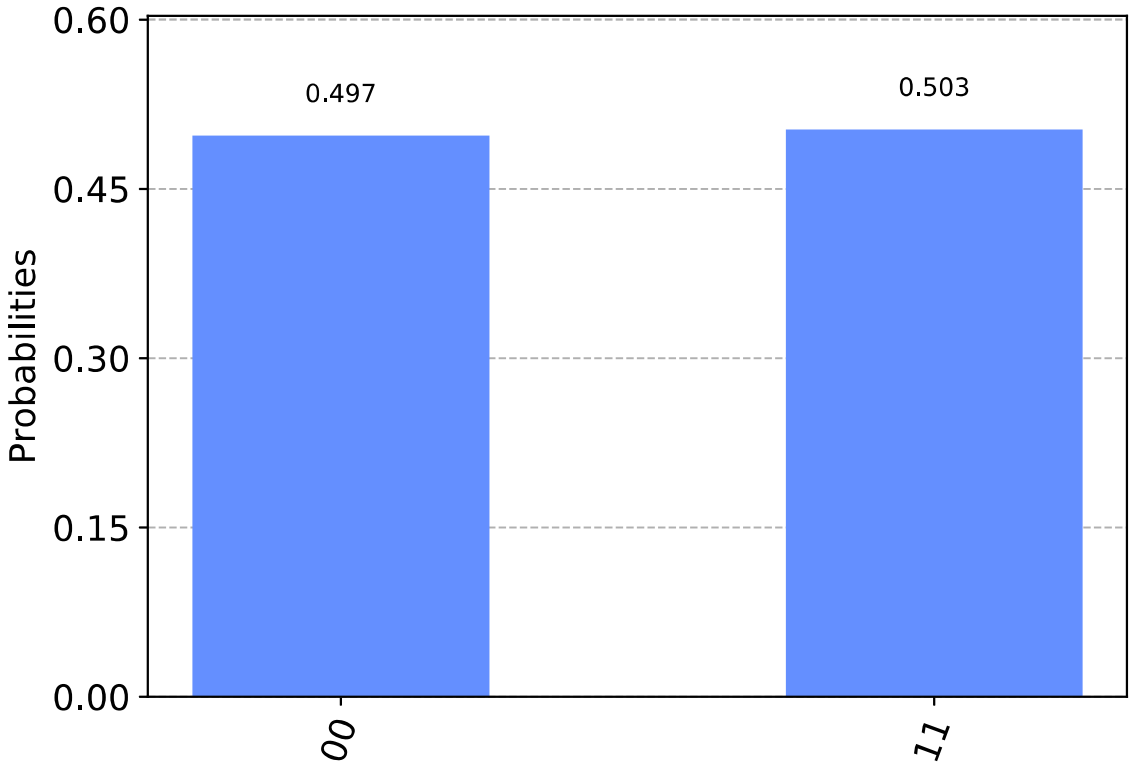
{'00': 4074, '11': 4118}

将其绘制为直方图

In [178]:

```
from qiskit.visualization import plot_histogram
plot_histogram(hist)
```

Out[178]:



对于本地所有兼容的后端，可以以有序列表的形式获取各次测量的结果。

In [179]:

```
job = execute(qc, emulator, shots=10, memory=True)
samples = job.result().get_memory()
print(samples) # 这些位是从右到左标记的，cr[0]是最右边
```

['00', '11', '11', '00', '11', '11', '11', '11', '11', '00']

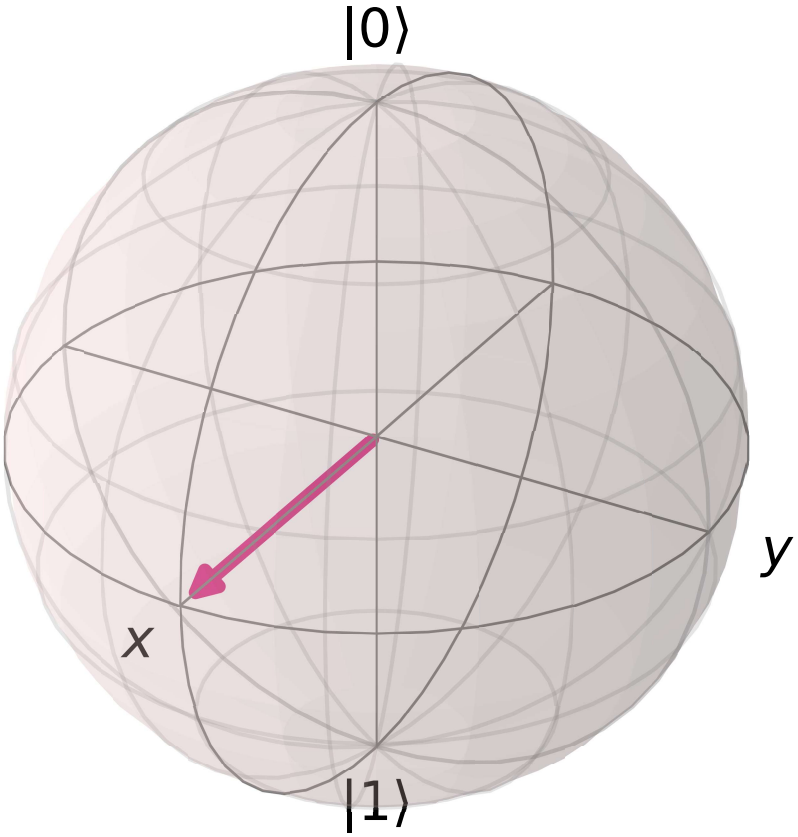
2.6向量空间的可视化

这里利用qiskit，使用Bloch球进行可视化。例如表示$|0\rangle$和$|1\rangle$ 的叠加态

In [180]:

```
from qiskit.visualization import plot_bloch_vector
plot_bloch_vector([1, 0, 0])
```

Out[180]:



2.7 访问真正的量子设备

后端的处理对象也可以使用IBMQ包来设置

In [181]:

```
IBMQ.load_account()
```

```
ibmqfactory.load_account:WARNING:2020-07-26 20:36:57,522: Credentials are already
in use. The existing account in the session will be replaced.
```

Out[181]:

```
<AccountProvider for IBMQ(hub='ibm-q', group='open', project='main')>
```

连接成功，现在可以查看还有哪些其他后端

In [182]:

```
provider = IBMQ.get_provider(hub='ibm-q')
provider.backends()
```

Out[182]:

```
[<IBMQSimulator('ibmq_qasm_simulator') from IBMQ(hub='ibm-q', group='open', projec
t='main')>,
 <IBMQBackend('ibmqx2') from IBMQ(hub='ibm-q', group='open', project='main')>,
 <IBMQBackend('ibmq_16_melbourne') from IBMQ(hub='ibm-q', group='open', project='m
ain')>,
 <IBMQBackend('ibmq_vigo') from IBMQ(hub='ibm-q', group='open', project='main')>,
 <IBMQBackend('ibmq_ourense') from IBMQ(hub='ibm-q', group='open', project='main')
>,
 <IBMQBackend('ibmq_london') from IBMQ(hub='ibm-q', group='open', project='main')
>,
 <IBMQBackend('ibmq_burlington') from IBMQ(hub='ibm-q', group='open', project='mai
n')>,
 <IBMQBackend('ibmq_essex') from IBMQ(hub='ibm-q', group='open', project='main')>,
 <IBMQBackend('ibmq_armonk') from IBMQ(hub='ibm-q', group='open', project='main')
>]
```

上面结果只有第一个是模拟器，其余的是真实的量子设备。

用status()方法查看这些设备的状态。

In [183]:

```
for backend in provider.backends():
    print(backend.status())
```

```
<qiskit.providers.models.backendstatus.BackendStatus object at 0x000001B95994F7B8>
<qiskit.providers.models.backendstatus.BackendStatus object at 0x000001B95994F7B8>
<qiskit.providers.models.backendstatus.BackendStatus object at 0x000001B95994F7B8>
<qiskit.providers.models.backendstatus.BackendStatus object at 0x000001B95994F7B8>
<qiskit.providers.models.backendstatus.BackendStatus object at 0x000001B95994F7B8>
<qiskit.providers.models.backendstatus.BackendStatus object at 0x000001B95994F7B8>
<qiskit.providers.models.backendstatus.BackendStatus object at 0x000001B95994F7B8>
<qiskit.providers.models.backendstatus.BackendStatus object at 0x000001B95994F7B8>
<qiskit.providers.models.backendstatus.BackendStatus object at 0x000001B95994F7B8>
```

获取量子位最大的公共后端对象

In [184]:

```
real_device = provider.get_backend('ibmq_16_melbourne')
```

可以使用它与仿真器完全相同的方式在设备上运行电路。还可以提取其某些属性。

In [185]:

```
properties = real_device.properties()
coupling_map = real_device.configuration().coupling_map
```

此外，可以构造一个噪声模型来模拟设备上的噪声

In [186]:

```
from qiskit.providers.aer.noise import NoiseModel # 导入模拟噪声的库

noise_model = NoiseModel.from_backend(properties)
```

在仿真器上运行该作业，使其再现真实设备的所有这些功能。下面这个电路示例本应在无噪声的情况下输出'10'。然而由于噪声存在，输出结果有偏差

In [187]:

```
qc = QuantumCircuit(2,2)
qc.x(1)
qc.measure(0,0)
qc.measure(1,1)

job = execute(qc, emulator, shots=1024, noise_model=noise_model,
                    coupling_map=coupling_map,
                    basis_gates=noise_model.basis_gates)

job.result().get_counts()
```

Out[187]:

{'00': 61, '10': 963}

结果显示有$\frac{66}{958}$的时间输出的是00，噪声比：$\frac{66}{1024} = 0.064$

三、使用Qiskit的工作流程包括三个主要步骤：

- 构建：设计一个代表您正在考虑的问题的量子电路。
- 执行：在不同的后端（包括系统和模拟器）上运行实验。
- 分析：计算汇总统计数据并可视化实验结果。 ### 分步工作流程 上面的程序可以细分为六个步骤：
- 导入包
- 初始化变量
- 添加门
- 可视化电路
- 模拟实验
- 可视化结果

1.导入包

In [188]:

```
import numpy as np
from qiskit import(
  QuantumCircuit, # 可以看作是量子系统的指令，它拥有所有的量子运算
  execute, # 运行电路/实验
  Aer) # 运行电路/实验。
from qiskit.visualization import plot_histogram # 创建直方图
```

2.初始化变量

In [189]:

```
circuit = QuantumCircuit(2, 2) # 使用处于零状态的2个量子位进行初始化。将2个经典位设置为零；circuit是量子电路
```

3.添加逻辑门来操纵电路的寄存器

In [190]:

```
circuit.h(0) # 添加H门，作用在qubit_0上，使其处于叠加状态
circuit.cx(0, 1) # 添加受控非门在控制量子位qubit_0和目标量子位qubit_1上，使量子位处于纠缠状态
circuit.measure([0,1], [0,1]) # 如果将整个量子和经典寄存器传递给measure，第i位量子比特的测量结果将存储在第i个经典位中。
```

Out[190]:

```
<qiskit.circuit.instructionset.InstructionSet at 0x1b95417e080>
```

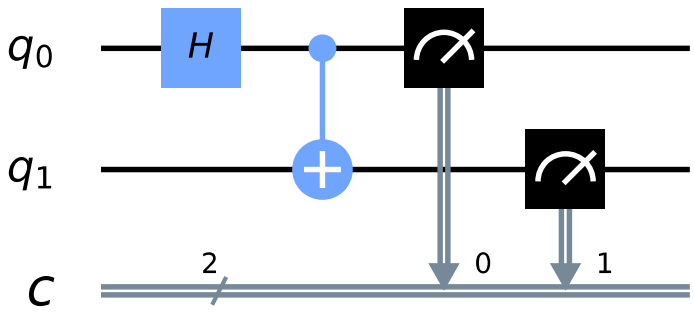
4.可视化量子电路

QuantumCircuit.draw()

In [191]:

```
circuit.draw(output='mpl')
```

Out[191]:



QuantumCircuit所带的draw方法讲刚刚创建的量子电路进行可视化，从左到右依次是H门、受控非门、并将qubit_0和qubit_1存储在c_0和c_1两个寄存器中

5.模拟实验

Qiskit Aer用于量子电路的高性能仿真器框架，用其中的qasm_simulator来模拟电路，该电路每次运行都会产生位串00或11

In [192]:

```
simulator = Aer.get_backend('qasm_simulator')
job = execute(circuit, simulator, shots=1000)
result = job.result()
counts = result.get_counts(circuit)
print("\nTotal count for 00 and 11 are:",counts)
```

```
Total count for 00 and 11 are: {'00': 482, '11': 518}
```

输出位串为00和11的时间比列约为1:1。因为qubit_0和qubit_1纠缠在了一起

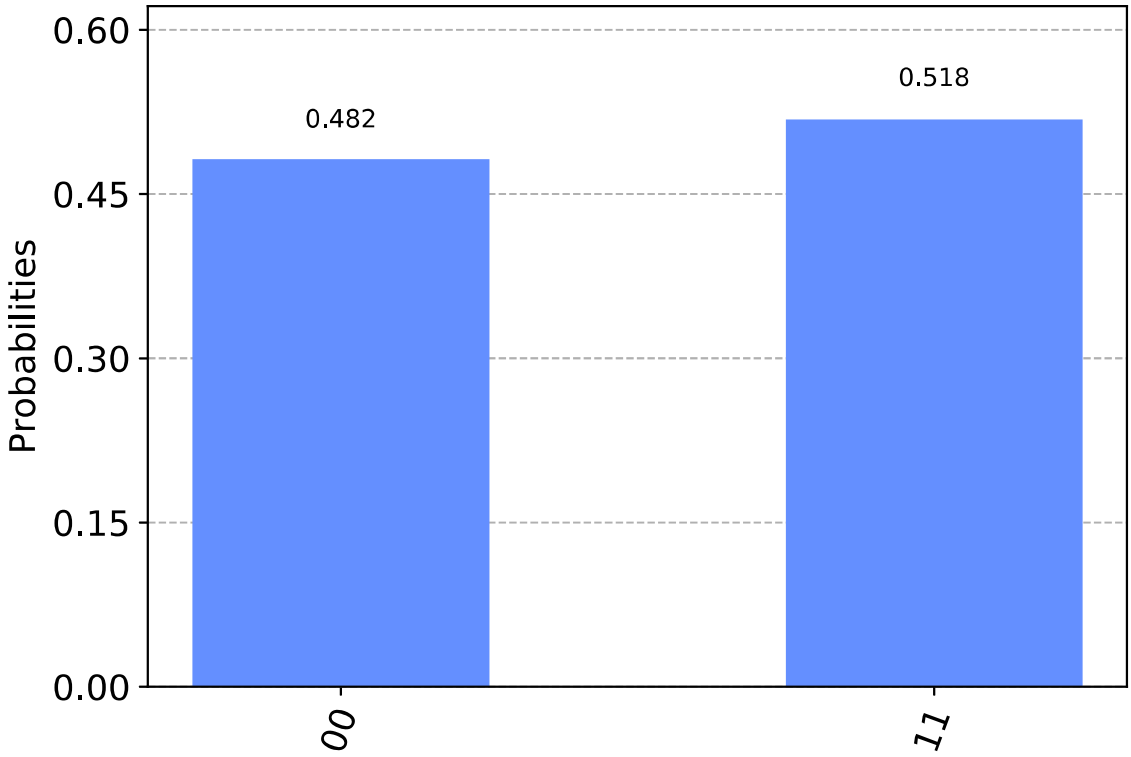
6.可视化结果

这里用直方图可视化输出输出位串00和11和时间

In [193]:

```
plot_histogram(counts)
```

Out[193]:



四、实现量子相位估计算法QPE

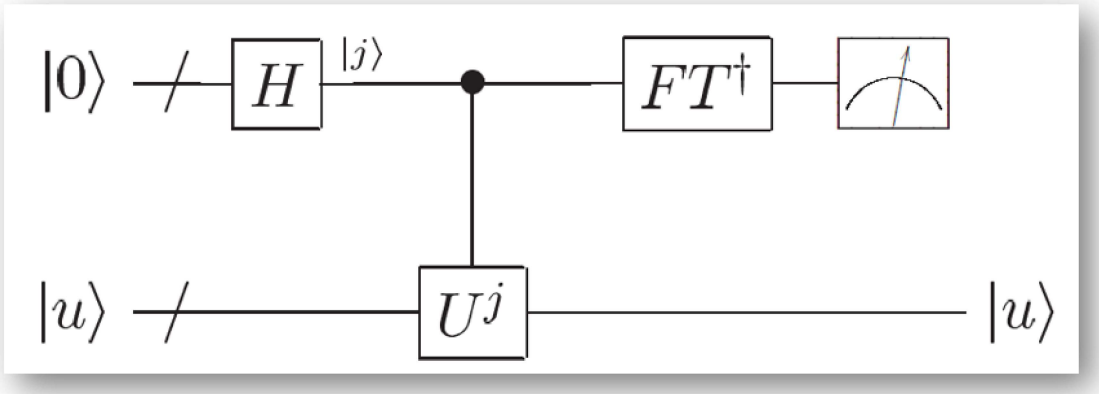
整个过程可以分为几个步骤:

- 1.在第一个寄存器上创建叠加
- 2.应用受控U门
- 3.傅里叶逆变换
- 4.测量

In [194]:

```
Image(filename = 'E:/桌面/图片1.png')
```

Out[194]:



In [195]:

```
%matplotlib inline
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister
from qiskit import execute
from qiskit import BasicAer
import math

# 设置寄存器中的比特数
n = 4  # 第一个寄存器中的有4个比特
m = 1  # 第二个寄存器中有1个比特

# 设置量子寄存器中的比特数 = 两个寄存器中比特数之和
q = QuantumRegister(n+m, 'q')

# 在寄存器上创建量子线路
register = QuantumCircuit(q)
```

4.1在第一个寄存器上创建叠加

4.2应用受控U门

In [196]:

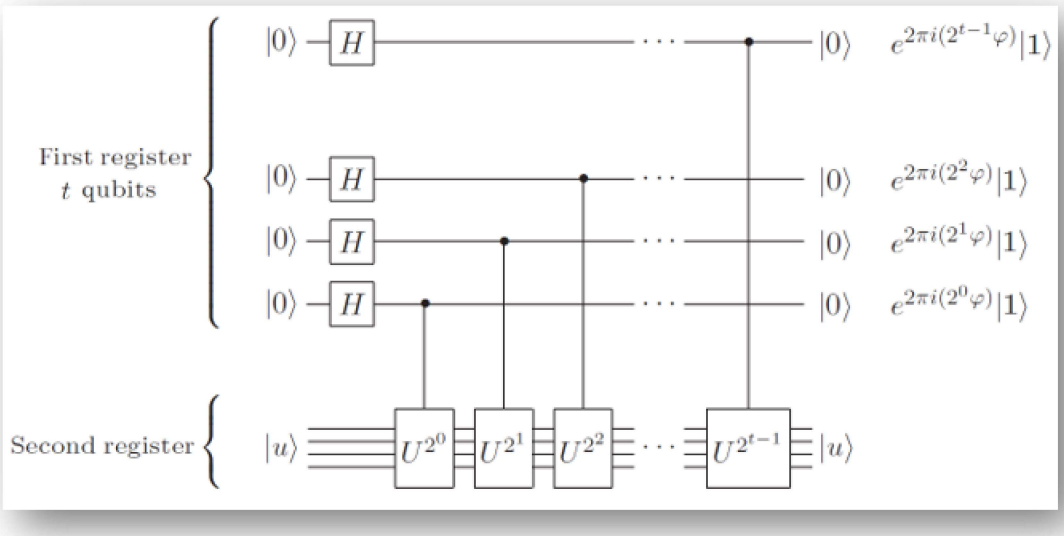
```
# 对第一个寄存器中的4个比特进行操作
for j in range(n-1,-1,-1):
    # 4.1 在第一个寄存中对所有量子位加H门，将这些量子位叠加:
    register.h(q[j])
    # 4.2 当第1个量子位为1时，对第2个量子位使用幺正算子U:
    register.u1(math.pi/float(2**(j)), q[j])
```

对应的线路表示为:

In [197]:

```
Image(filename = 'E:/桌面/图片2.png')
```

Out[197]:



4.3逆傅里叶变换

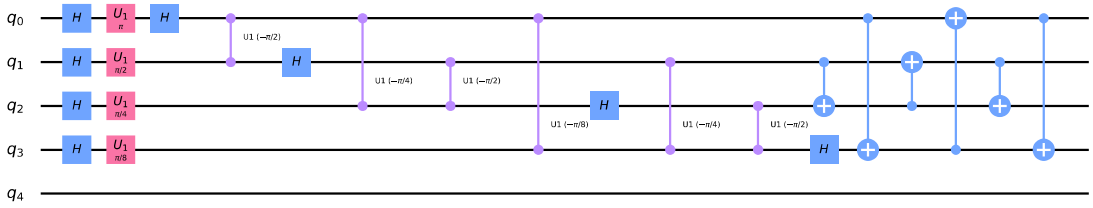
In [198]:

```
# 实现下图的R_k操作，移动小数点，向左移动k位
for j in range(n):
    for k in range(j):
        register.cu1(-math.pi/float(2**(j-k)), q[j], q[k])
    register.h(q[j])

# 利用3个受控非门进行两比特的交换
def swap(qc, q, i, j):
    qc.cx(q[i], q[j])
    qc.cx(q[j], q[i])
    qc.cx(q[i], q[j])

if n%2==1:
    for i in range(int((n-1)/2)):
        swap(register, q, i, n-i-1)
else:
    for i in range(int(n/2)):
        swap(register, q, i, n-i-1)
register.draw(output="mpl")
```

Out[198]:

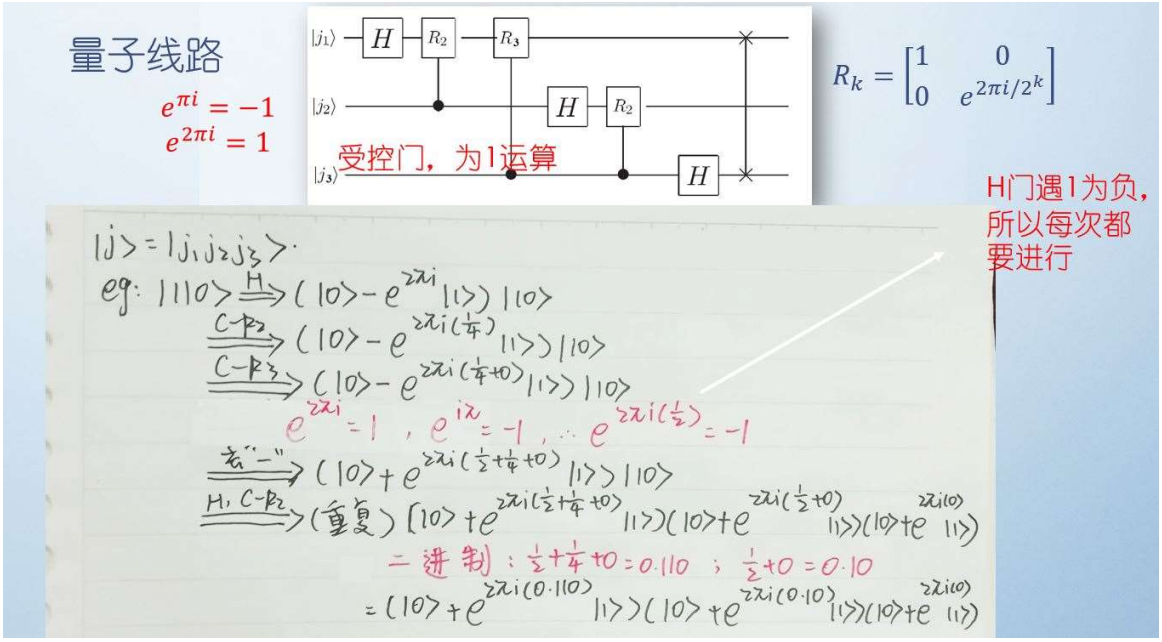


将R_k操作作用于第一个寄存器中的每个比特

In [199]:

```
Image(filename = 'E:/桌面/捕获.jpg')
```

Out[199]:



4.4测量

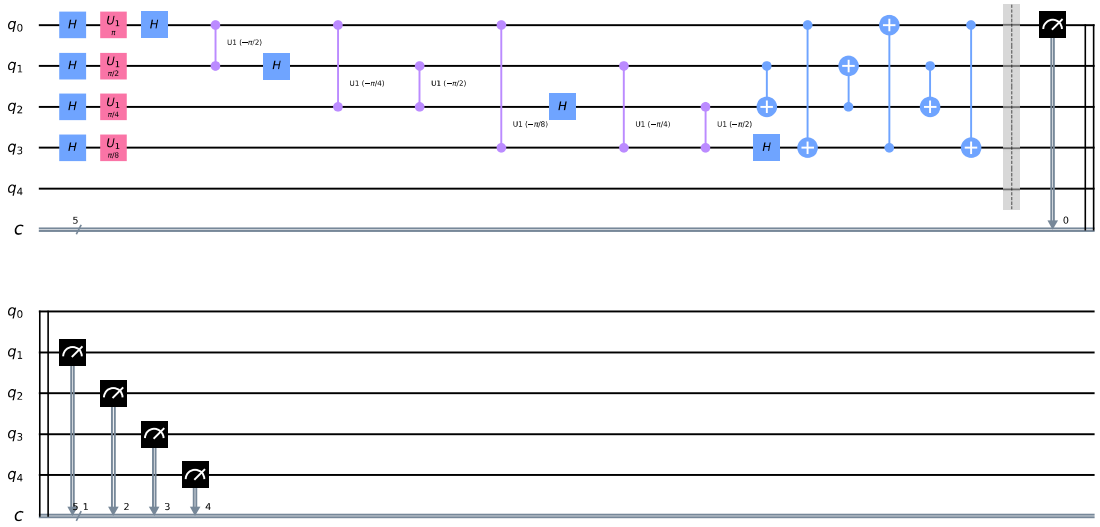
In [200]:

```
# 创建4比特的经典寄存器
c = ClassicalRegister(n+m, 'c')
# 创建一个量子线路
measure = QuantumCircuit(q, c)
measure.barrier(q)
# 将量子测量映射到经典比特
measure.measure(q,c)

# Qiskit电路对象支持组合使用
# 加法操作符
qc = register+measure

# 绘制电路
qc.draw(output="mpl")
```

Out[200]:



In [201]:

```
# 用Aer中的qasm_simulator模拟器进行模拟
backend_sim = BasicAer.get_backend('qasm_simulator')

# 在qasm模拟器上执行电路
# 将电路的重复次数设置为1024次，此为默认值。
job_sim = execute(qc, backend_sim, shots=1024)

# 从作业job中中获取结果。
result_sim = job_sim.result()

counts = result_sim.get_counts(qc)
print(counts)
```

```
{'01000': 1024}
```

结果位按顺序显示qn...q2q1，因此需要将其更改为顺序q1q2...qn，然后除以2^n得到正确的相位。

例如，如果结果是01000（q4q3q2q1q0），应该首先按照正确的顺序00010（q0q1q2q3q4）进行排序。忽略第二个寄存器，因此结果变为0001。除以2^4（这里n=4）后，得到的答案（十进制）1/16，这是相位。

In [202]:

```
from qiskit.tools.visualization import plot_histogram
plot_histogram(result_sim.get_counts(qc))
```

Out[202]: