In [1]:

```
#initialization
import matplotlib.pyplot as plt
import numpy as np
import math

# importing Qiskit
from qiskit import IBMQ, Aer, transpile, assemble
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister

# import basic plot tools
from qiskit.visualization import plot_histogram
```

1. T Gate

Let's take a gate we know well, the T-gate, and use Quantum Phase Estimation to estimate its phase. You will remember that the T-gate adds a phase of $e^{\frac{i\pi}{4}}$ to the state $|1\rangle$:

$$T|1\rangle = \begin{bmatrix} 1 & 0 \\ 0 & e^{\frac{i\pi}{4}} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = e^{\frac{i\pi}{4}}|1\rangle$$

Since QPE will give us θ where:

$$T|1\rangle = e^{2i\pi\theta}|1\rangle$$

We expect to find:

$$\theta = \frac{1}{8}$$

In this example we will use three qubits and obtain an *exact* result (not an estimation!)

1.1. Circuit

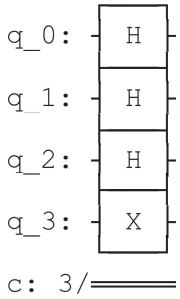
Now, set up the quantum circuit. We will use four qubits -- qubits 0 to 2 as counting qubits, and qubit 3 as the eigenstate of the unitary operator (T)

We initialize |ψ⟩ = |1⟩ by applying an X gate:

In [2]:

```
qpe = QuantumCircuit(4, 3)
qpe.x(3)

#Apply Hadamard gate
for qubit in range(3):
    qpe.h(qubit)

qpe.draw()
```

Out[2]:



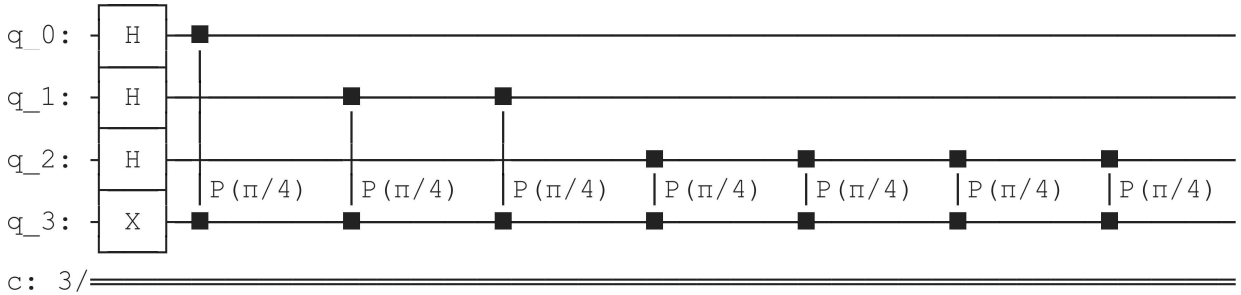
Next we perform the controlled unitary operations.

Remember: Qiskit orders its qubits the opposite way

In [3]:

```
repetitions = 1
for counting_qubit in range(3):
    for i in range(repetitions):
        qpe.cp(math.pi/4, counting_qubit, 3); # This is CU  # we use 2*pi*(1/theta)

    repetitions *= 2
qpe.draw()
```

Out[3]:



We apply the inverse quantum Fourier transformation to convert the state of the counting register.

In [4]:

```
def qft_dagger(qc, n):
    """n-qubit QFTdagger the first n qubits in circ"""
    # Don't forget the Swaps!
    for qubit in range(n//2):
        qc.swap(qubit, n-qubit-1)
```

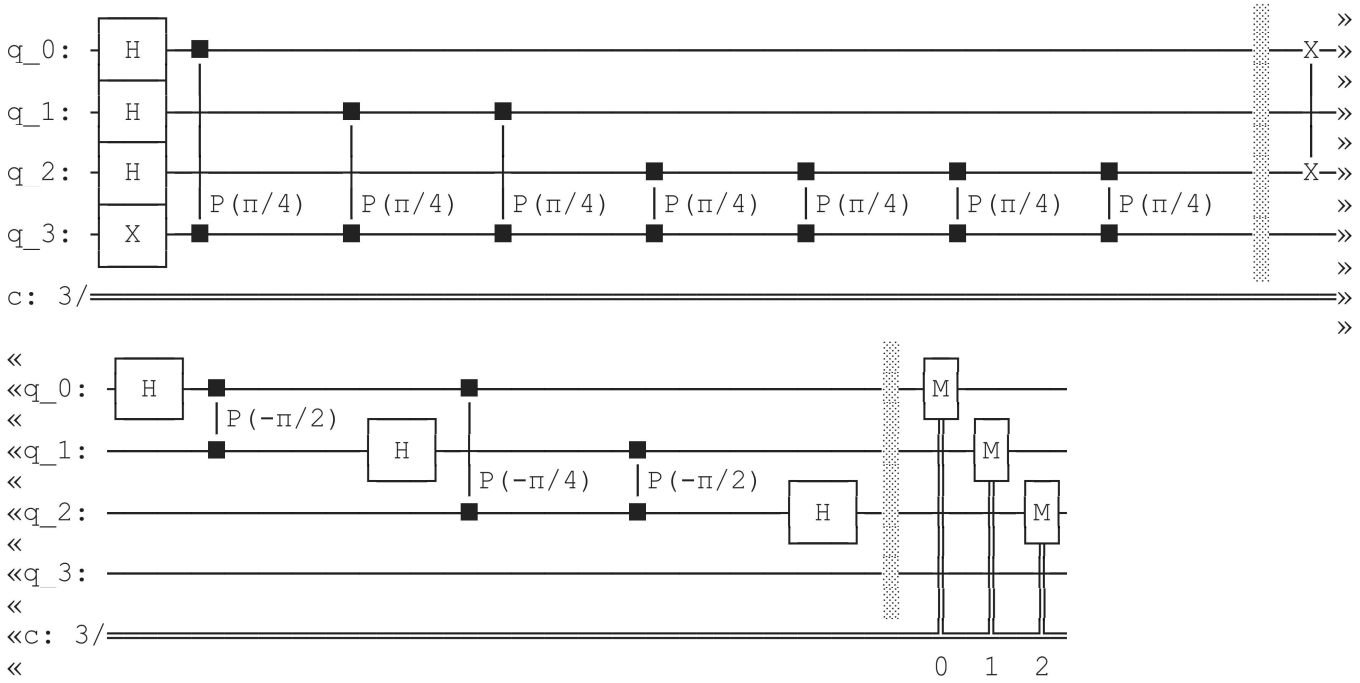
```
    for j in range(n):
        for m in range(j):
            qc.cp(-math.pi/float(2**(j-m)), m, j)
        qc.h(j)
```

We then measure the counting register:

In [5]:
```
qpe.barrier()
# Apply inverse QFT
qft_dagger(qpe, 3)
# Measure
qpe.barrier()
for n in range(3):
    qpe.measure(n,n)
```

In [6]:
```
qpe.draw()
```

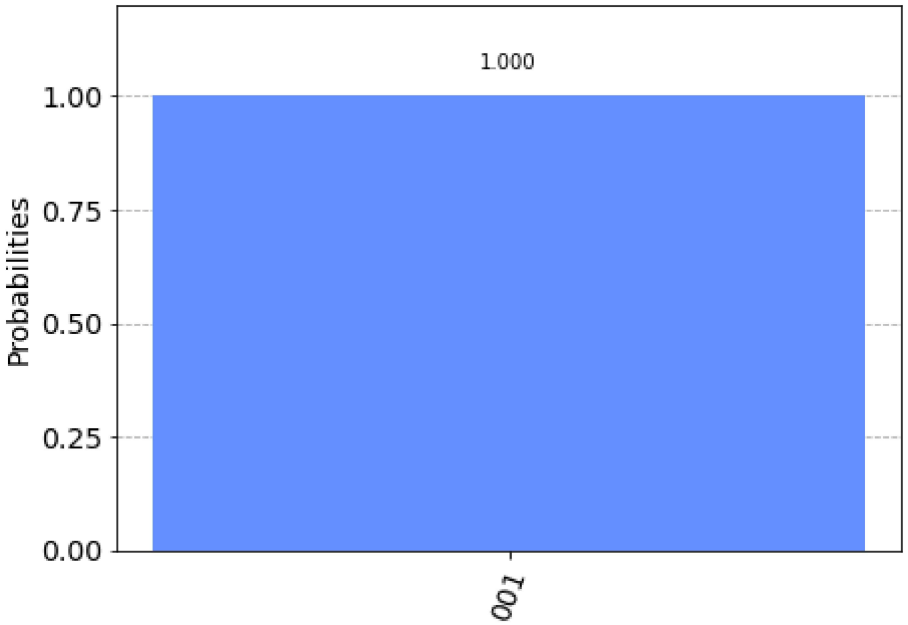
Out[6]:



1.2 . Results(Simulator)

In [7]:
```
aer_sim = Aer.get_backend('aer_simulator')
shots = 2048
t_qpe = transpile(qpe, aer_sim)
qobj = assemble(t_qpe, shots=shots)
results = aer_sim.run(qobj).result()
answer = results.get_counts()

plot_histogram(answer)
```

Out[7]:



We see we get one result (`001`) with certainty, which translates to the decimal: `1`. We now need to divide our result (`1`) by 2^n to get θ:

$$\theta = \frac{1}{2^3} = \frac{1}{8}$$

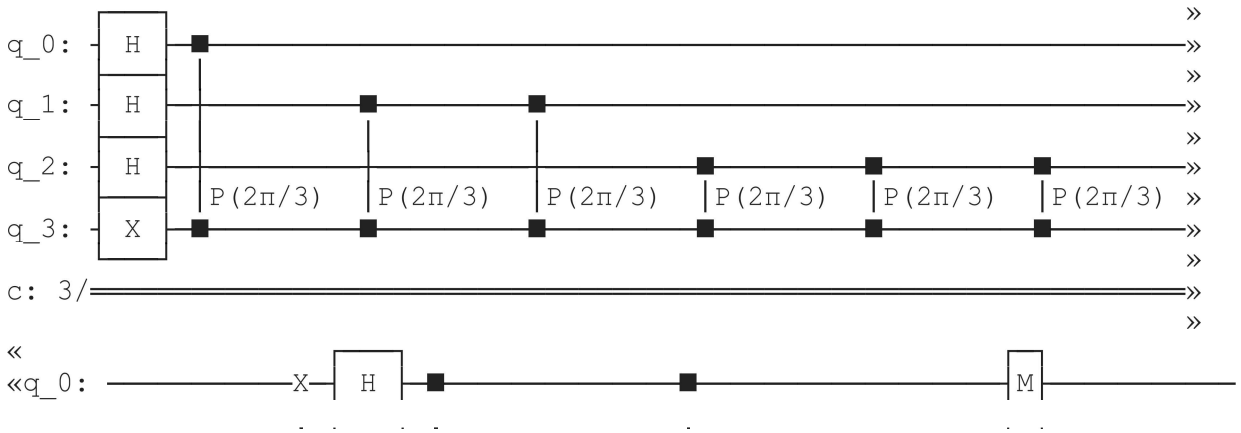
This is exactly the result we expected!

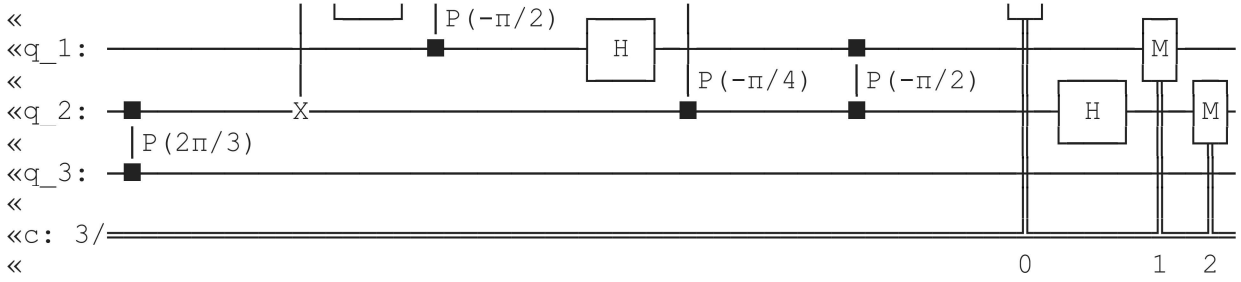
2. Using Theta = 1/3

In [8]:

```
# Create and set up circuit
qpe2 = QuantumCircuit(4, 3)

# Apply H-Gates to counting qubits:
for qubit in range(3):
    qpe2.h(qubit)

# Prepare our eigenstate |psi>:
qpe2.x(3)

# Do the controlled-U operations:
angle = 2*math.pi/3
repetitions = 1
for counting_qubit in range(3):
    for i in range(repetitions):
        qpe2.cp(angle, counting_qubit, 3);
    repetitions *= 2

# Do the inverse QFT:
qft_dagger(qpe2, 3)

# Measure of course!
for n in range(3):
    qpe2.measure(n,n)

qpe2.draw()
```

Out[8]:

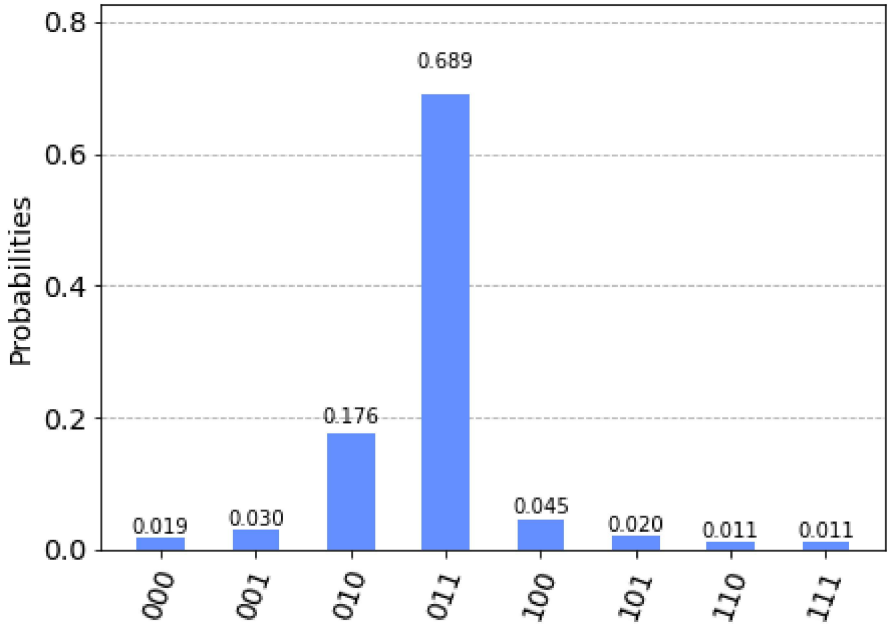




In [9]:

```
# Let's see the results!
aer_sim = Aer.get_backend('aer_simulator')
shots = 4096
t_qpe2 = transpile(qpe2, aer_sim)
qobj = assemble(t_qpe2, shots=shots)
results = aer_sim.run(qobj).result()
answer = results.get_counts()

plot_histogram(answer)
```

Out[9]:

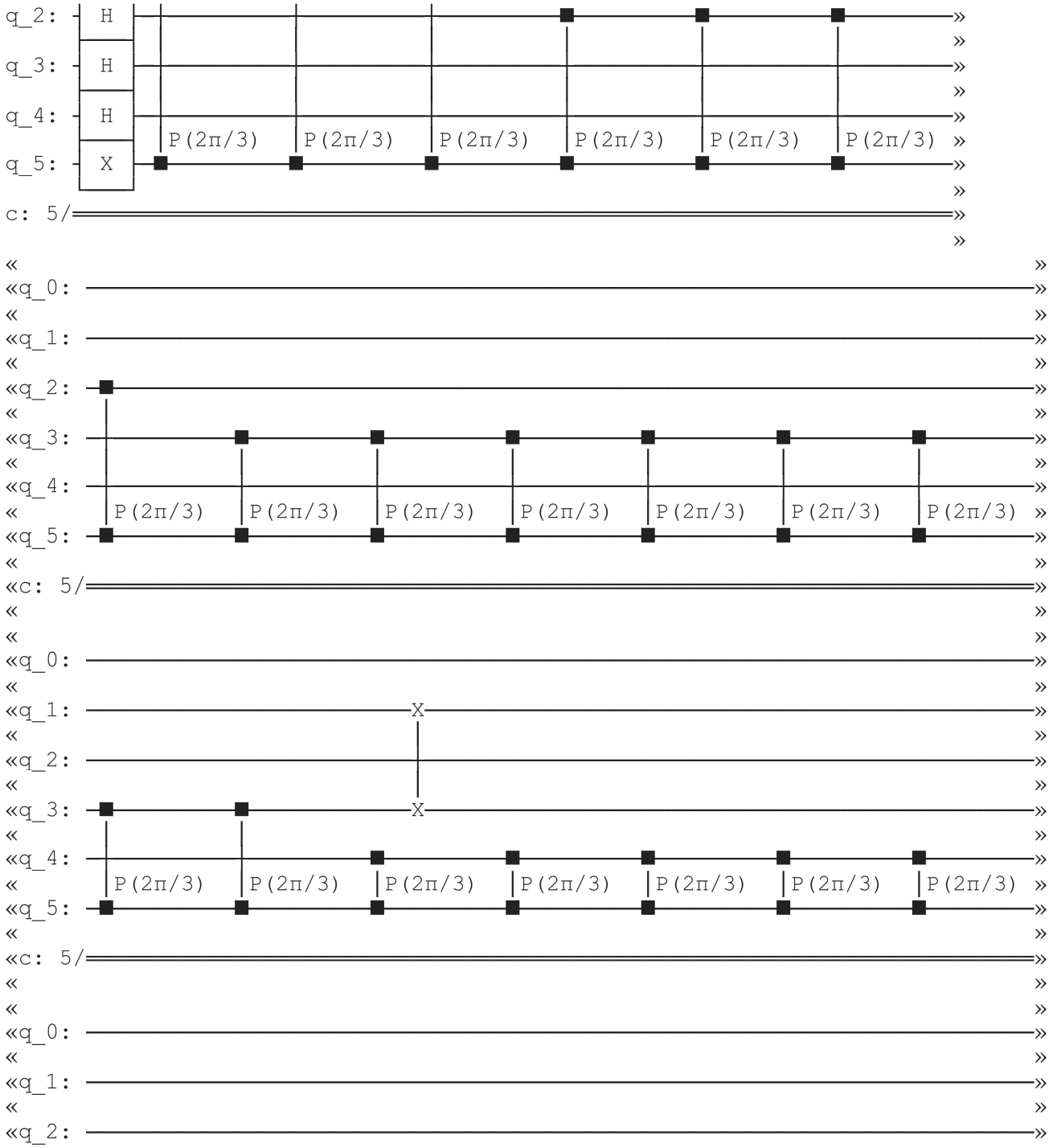


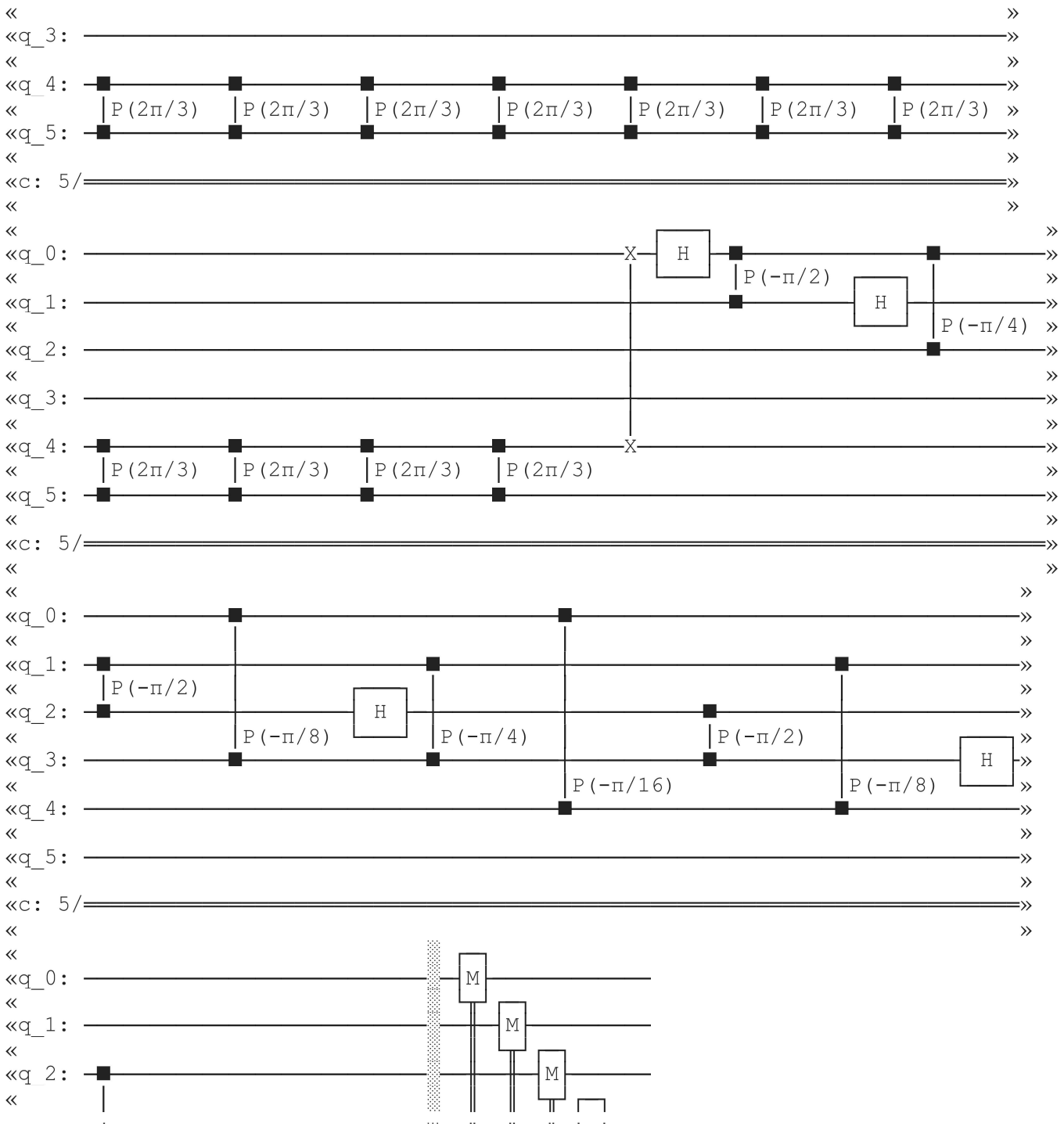
We are expecting the result $\theta = 0.3333\ldots$, and we see our most likely results are `010(bin) = 2(dec)` and `011(bin) = 3(dec)`. These two results would tell us that $\theta = 0.25$ (off by 25%) and $\theta = 0.375$ (off by 13%) respectively. The true value of θ lies between the values we can get from our counting bits, and this gives us uncertainty and imprecision.

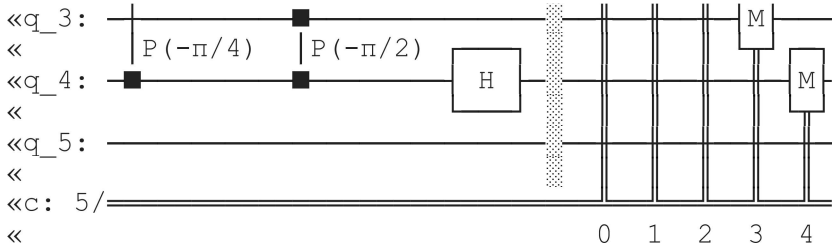
To get more precision we simply add more counting qubits. We are going to add two more counting qubits

In [10]:
```
# Create and set up circuit
qpe3 = QuantumCircuit(6, 5)

# Apply H-Gates to counting qubits:
for qubit in range(5):
    qpe3.h(qubit)

# Prepare our eigenstate |psi>:
qpe3.x(5)

# Do the controlled-U operations:
angle = 2*math.pi/3
repetitions = 1
for counting_qubit in range(5):
    for i in range(repetitions):
        qpe3.cp(angle, counting_qubit, 5);
    repetitions *= 2

# Do the inverse QFT:
qft_dagger(qpe3, 5)

# Measure of course!
qpe3.barrier()
for n in range(5):
    qpe3.measure(n,n)

qpe3.draw()
```

Out[10]:


q_2:
q_3:
q_4:
q_5:
c: 5/
H
H
H
X
P(2π/3)
q_0:
q_1:
q_2:
q_3:
q_4:
q_5:
c: 5/
X

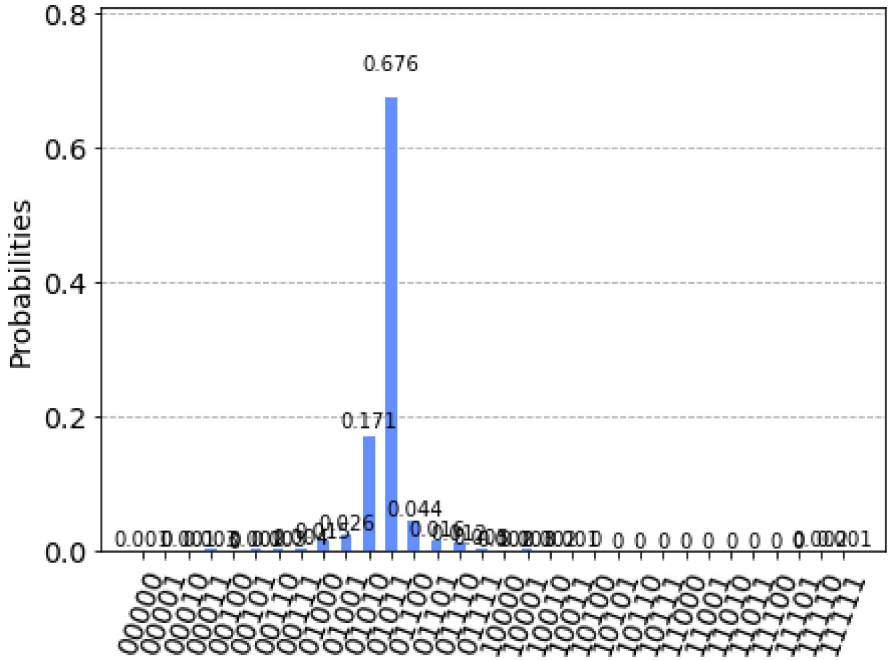

q_3:
q_4:
P(2π/3)
q_5:
c: 5/
q_0:
X
H
P(-π/2)
q_1:
H
P(-π/4)
q_2:
q_3:
q_4:
X
P(2π/3)
q_5:
c: 5/
q_0:
q_1:
P(-π/2)
q_2:
H
P(-π/8)
P(-π/4)
P(-π/2)
q_3:
H
P(-π/16)
P(-π/8)
q_4:
q_5:
c: 5/
q_0:
M
q_1:
M
q_2:
M



In [11]:

```
# Let's see the results!
aer_sim = Aer.get_backend('aer_simulator')
shots = 4096
t_qpe3 = transpile(qpe3, aer_sim)
qobj = assemble(t_qpe3, shots=shots)
results = aer_sim.run(qobj).result()
answer = results.get_counts()

plot_histogram(answer)
```

Out[11]:



The two most likely measurements are now `01011` (decimal 11) and `01010` (decimal 10). Measuring these results would tell us θ is:

$$\theta = \frac{11}{2^5} = 0.344, \quad \text{or} \quad \theta = \frac{10}{2^5} = 0.313$$

These two results differ from $\frac{1}{3}$ by 3% and 6% respectively. A much better precision!

In [ ]: