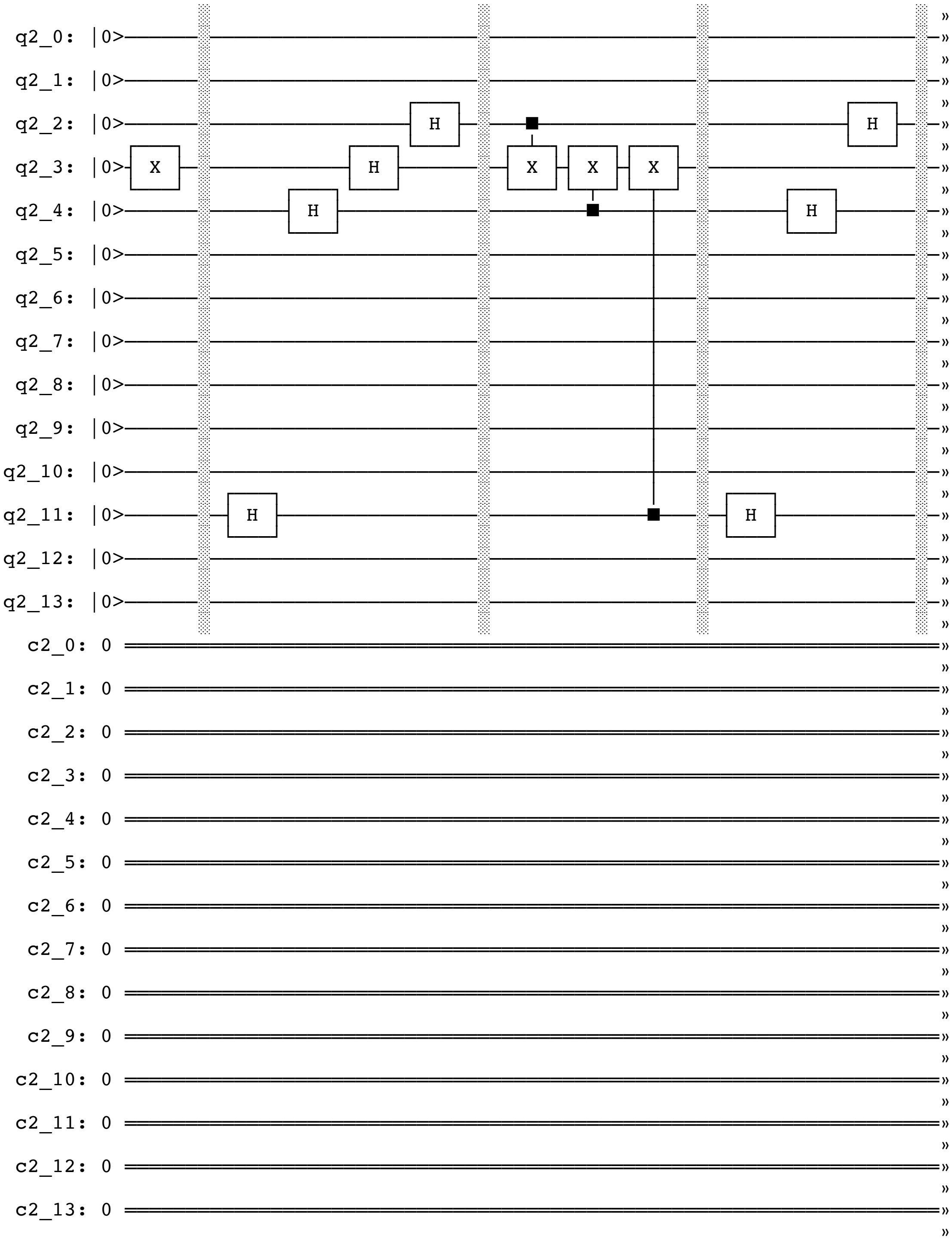
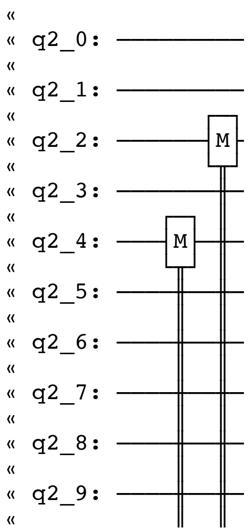
In [3]:
```
#implementation of the DJ algorithm on the simulator and the Melbourne backend

from qiskit import QuantumRegister, ClassicalRegister
from qiskit import QuantumCircuit, Aer, execute
from qiskit import IBMQ

from qiskit.tools.visualization import plot_histogram

from qiskit.tools.monitor import job_monitor

#if you dont have an account you must create one, get an API token, and enable
or save the account.
#once the account is saved you just need to use load below.
#see Qistkit terra for instructions.

#IBMQ.save_account('my API token')

IBMQ.load_accounts()

#currently existing backends
#backend = IBMQ.get_backend('ibmq_16_melbourne')
#backend = IBMQ.get_backend('ibmqx4')
#backend = IBMQ.get_backend('ibmqx2')
#backend = IBMQ.get_backend('ibmq_qasm_simulator')

q = QuantumRegister(14)
c = ClassicalRegister(14)
DJ = QuantumCircuit(q, c)

#quantum circuit using qubits 2,3,4,11
#on melbourne for these qubits the coupling map is CX2-3, CX4-3, CX11-3
#we take a balanced function f(x2,x4,x11)= x2 + x4 + x11
#this is a linear function f = <a, x> and the algo can also be viewed as the Be
rnstein-Vazirani algo to find the vector a

#preparation
DJ.x(q[3])

DJ.barrier(q)

DJ.h(q[2])
DJ.h(q[3])
DJ.h(q[4])
DJ.h(q[11])

DJ.barrier(q)

#function f (oracle)
DJ.cx(q[2],q[3])
DJ.cx(q[4],q[3])
DJ.cx(q[11],q[3])

DJ.barrier(q)

#fourier analysis
DJ.h(q[2])
DJ.h(q[4])
DJ.h(q[11])

#measurement of bits 2,4

DJ.barrier(q)

DJ.measure(q[2], c[2])
DJ.measure(q[4], c[4])
DJ.measure(q[11], c[11])
```

```
Out[3]: <qiskit.circuit.measure.Measure at 0xa1e2704a8>
```

In [4]:
```
#drawing the circuit for verification

DJ.draw()
```

Out[4]:

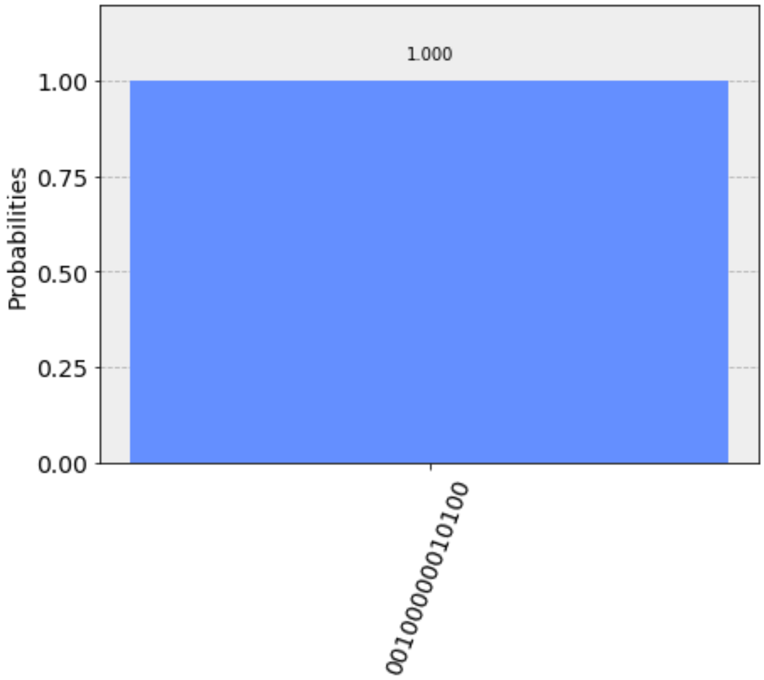

q2_0: |0>
q2_1: |0>
q2_2: |0>
q2_3: |0>
q2_4: |0>
q2_5: |0>
q2_6: |0>
q2_7: |0>
q2_8: |0>
q2_9: |0>
q2_10: |0>
q2_11: |0>
q2_12: |0>
q2_13: |0>
c2_0: 0
c2_1: 0
c2_2: 0
c2_3: 0
c2_4: 0
c2_5: 0
c2_6: 0
c2_7: 0
c2_8: 0
c2_9: 0
c2_10: 0
c2_11: 0
c2_12: 0
c2_13: 0
X
H
X
M


q2_0:
q2_1:
q2_2:
q2_3:
q2_4:
q2_5:
q2_6:
q2_7:
q2_8:
q2_9:
M

In [6]:
```
#simulation - we find the vector a2=1, a4=1, a11=1 and other components 0

backend = Aer.get_backend('qasm_simulator')
job_sim = execute(DJ, backend)
sim_result = job_sim.result()

print(sim_result.get_counts(DJ))
plot_histogram(sim_result.get_counts(DJ))
```

```
{'00100000010100': 1024}
```

Out[6]:



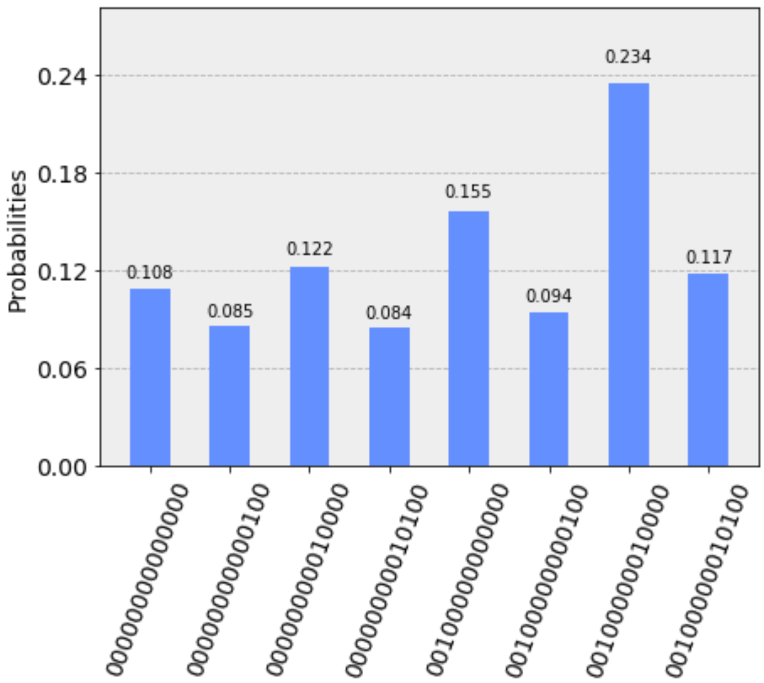
In [7]:
```
#experiment
#you can also check for availability and current parameters of the backend befo
re calling it.
#see instructions in Quiskit terra.

backend = IBMQ.get_backend('ibmq_16_melbourne')  #circuit above respects constr
iants of melbourne device.
shots = 1024         # Number of shots to run the program (experiment); maximum
is 8192 shots.
max_credits = 3        # Maximum number of credits to spend on executions.

job_exp = execute(DJ, backend=backend, shots=shots, max_credits=max_credits)
job_monitor(job_exp)
```

In [8]:

```
#results of experiment

result_exp = job_exp.result()
counts_exp = result_exp.get_counts(DJ)

print(result_exp.get_counts(DJ))
plot_histogram([counts_exp])
```

```
{'00100000000100': 96, '00000000000100': 87, '00100000000000': 159, '000000000
10100': 86, '00000000010000': 125, '00100000010000': 240, '00100000010100': 12
0, '00000000000000': 111}
```

Out[8]:

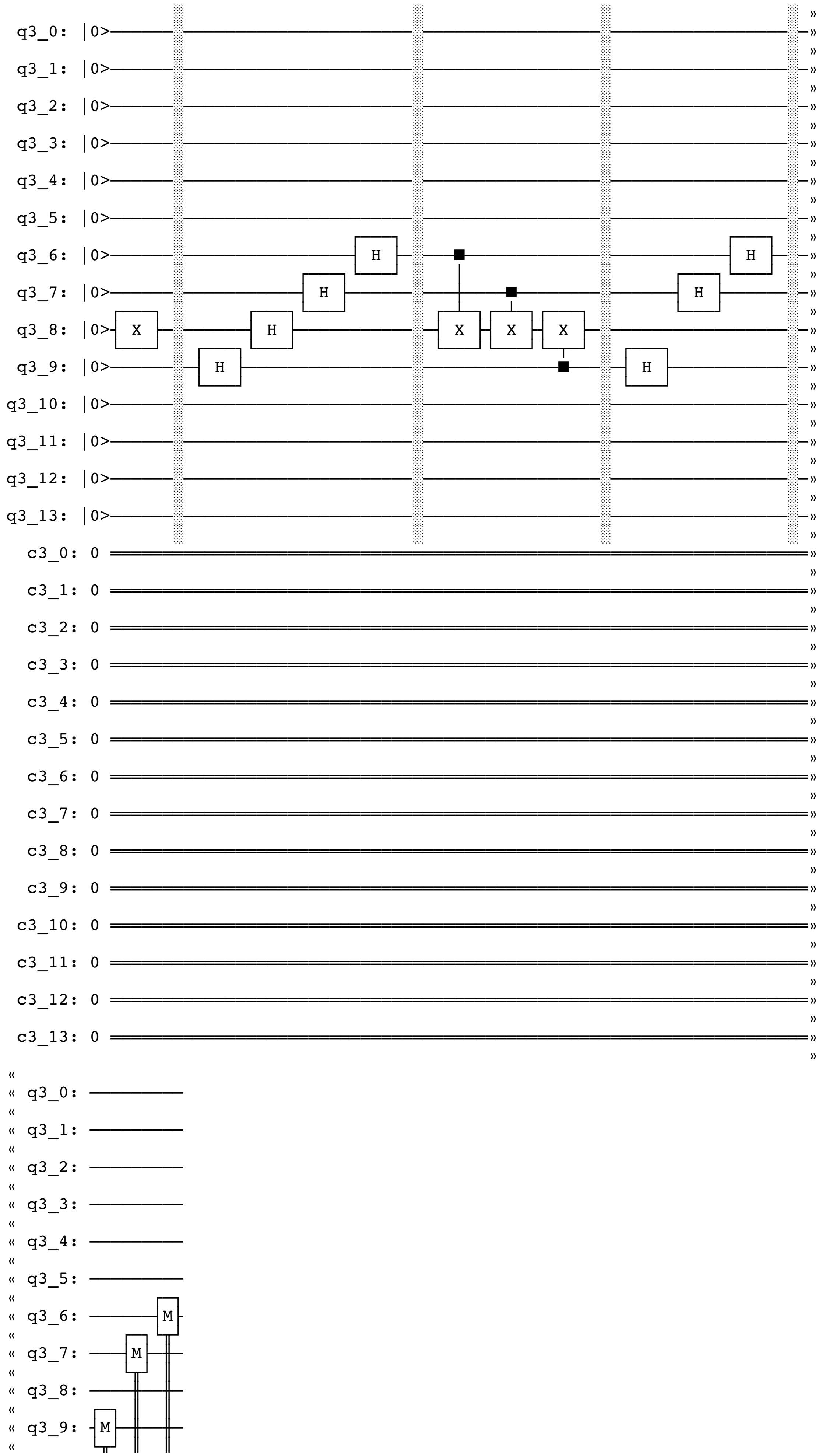


In [9]:
```
#second implementation of the DJ algorithm on the simulator and the Melbourne b
ackend.
#we use other qubits.

from qiskit import QuantumRegister, ClassicalRegister
from qiskit import QuantumCircuit, Aer, execute
from qiskit import IBMQ

from qiskit.tools.visualization import plot_histogram

from qiskit.tools.monitor import job_monitor

#if you dont have an account you must create one, get an API token, and enable
or save the account.
#once the account is saved you just need to use load below.
#see Qistkit terra for instructions.

#IBMQ.save_account('my API token')

IBMQ.load_accounts()

#currently existing backends
#backend = IBMQ.get_backend('ibmq_16_melbourne')
#backend = IBMQ.get_backend('ibmqx4')
#backend = IBMQ.get_backend('ibmqx2')
#backend = IBMQ.get_backend('ibmq_qasm_simulator')

q = QuantumRegister(14)
c = ClassicalRegister(14)
DJ = QuantumCircuit(q, c)

#quantum circuit using qubits 6, 7, 8, 9
#on melbourne for these qubits the coupling map is CX6-8, CX9-8, CX7-8
#we take a balanced function f(x2,x4,x11)= x6 + x7 + x9
#again this is in fact a Bernstein-Vazirani version and the algo even returns '
a' in teh linear function f=<a, x>.

#preparation
DJ.x(q[8])

DJ.barrier(q)

DJ.h(q[6])
DJ.h(q[7])
DJ.h(q[8])
DJ.h(q[9])

DJ.barrier(q)

#function f (oracle)
DJ.cx(q[6],q[8])
DJ.cx(q[7],q[8])
DJ.cx(q[9],q[8])

DJ.barrier(q)

#fourier analysis
DJ.h(q[6])
DJ.h(q[7])
DJ.h(q[9])

#measurement of bits 2,4

DJ.barrier(q)

DJ.measure(q[6], c[6])
DJ.measure(q[7], c[7])
DJ.measure(q[9], c[9])
```

```
Out[9]: <qiskit.circuit.measure.Measure at 0xa1e226c50>
```

In [10]:
```
#drawing the circuit for verification

DJ.draw()
```

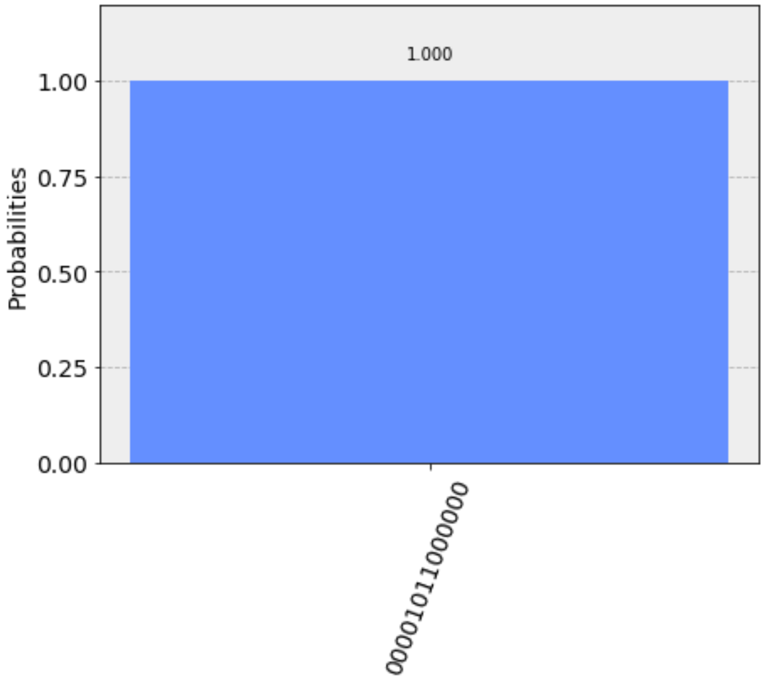
Out[10]:


q3_0: |0>
q3_1: |0>
q3_2: |0>
q3_3: |0>
q3_4: |0>
q3_5: |0>
q3_6: |0>
q3_7: |0>
q3_8: |0>
q3_9: |0>
q3_10: |0>
q3_11: |0>
q3_12: |0>
q3_13: |0>
c3_0: 0
c3_1: 0
c3_2: 0
c3_3: 0
c3_4: 0
c3_5: 0
c3_6: 0
c3_7: 0
c3_8: 0
c3_9: 0
c3_10: 0
c3_11: 0
c3_12: 0
c3_13: 0
X
H
H
H
H
X
X
X
H
H
H
q3_0:
q3_1:
q3_2:
q3_3:
q3_4:
q3_5:
q3_6:
q3_7:
q3_8:
q3_9:
M
M
M

In [11]:
```
#simulation finds vector a

backend = Aer.get_backend('qasm_simulator')
job_sim = execute(DJ, backend)
sim_result = job_sim.result()

print(sim_result.get_counts(DJ))
plot_histogram(sim_result.get_counts(DJ))
```

```
{'00001011000000': 1024}
```

Out[11]:



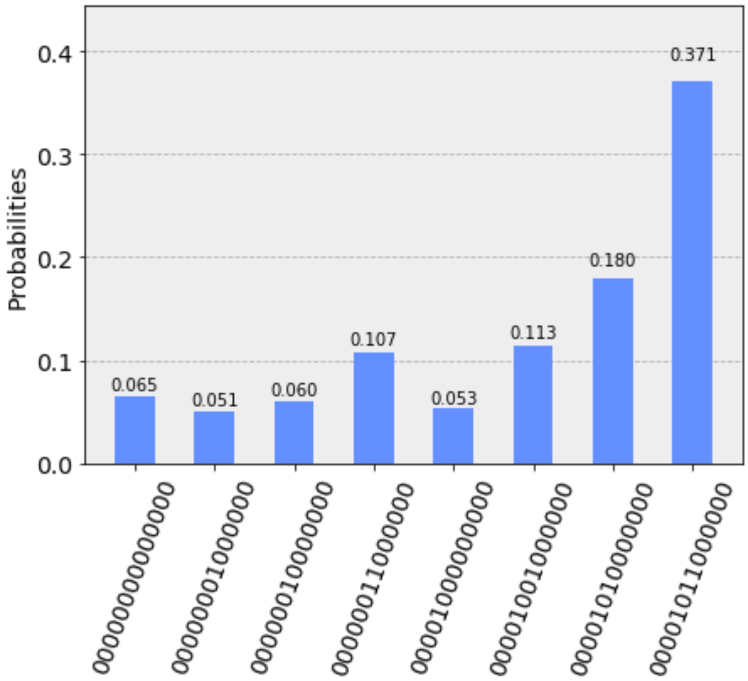
In [12]:
```
#experiment
#you can also check for availability and current parameters of the backend befo
re calling it.
#see instructions in Quiskit terra.

backend = IBMQ.get_backend('ibmq_16_melbourne')  #circuit above respects constr
iants of melbourne device.
shots = 1024         # Number of shots to run the program (experiment); maximum
is 8192 shots.
max_credits = 3        # Maximum number of credits to spend on executions.

job_exp = execute(DJ, backend=backend, shots=shots, max_credits=max_credits)
job_monitor(job_exp)
```

In [13]:

```
#results of experiment

result_exp = job_exp.result()
counts_exp = result_exp.get_counts(DJ)

print(result_exp.get_counts(DJ))
plot_histogram([counts_exp])
```

```
{'00001010000000': 184, '00001011000000': 380, '00001000000000': 54, '00000010
000000': 61, '00001001000000': 116, '00000001000000': 52, '00000011000000': 11
0, '00000000000000': 67}
```

Out[13]:



In [ ]: