Bernstein-Vazirani Algorithm

- Suppose you have a number with n bits , hidden inside somewhere.
- A classical computer can do the guessing of the number in n tries
- Quantum computer can do it in ONE try independent of the size of the umber that you have .
- How?
- Setup->
  - You have a box with the number hidden inside it
  - For example - **110011**
  - You have a function, in which if you feed a number, it returns you a Yes or a No, pertaining to whether the number is same as the one inside the box or not.
  - BOX - 110011
  - Func(000000) -> returns **FALSE**
  - Func(010010) -> returns **FALSE**
  - Func(110011) -> returns **TRUE**

General Doubt people have

- You may say that "Just take the and of the number with 111...111 (n times) and you'll find your number in a single try.
- But hold on, THIS WOULD STILL REQUIRE *N AND OPERATIONS*.
- A Quantum computer can do this in ONE TRY

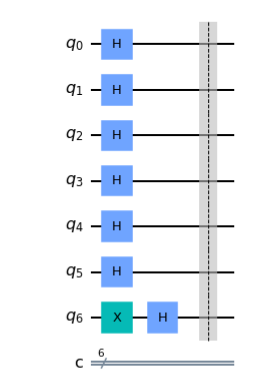
In [1]:
```
# example
from qiskit import *
%matplotlib inline
from qiskit.tools.visualization import plot_histogram
import numpy as np
```

In [2]:
```
secret = '110011'
# we are building the BLACK BOX OURSELVES
# the scenario actually would be we have a black box
# how do we extract the number from it?
# So, we are actually building the black box
# and trying it out for ourselves only...
```

Building the **BLACK BOX**

- This black box will be given to me, my job is to efficiently make a guess
- Which I do after applying the hadamard gates

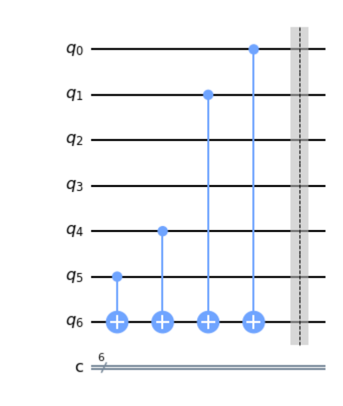
In [3]:
```
# make a six qubit circuit
Q = QuantumCircuit(6+1,6)
Q.h([0,1,2,3,4,5]) # creating equal superposition of six qubits
# a different config for last qubit
# just creates
Q.x(6)
Q.h(6)
Q.barrier()
Q.draw(output='mpl')
```



In [4]:
```
secret
```

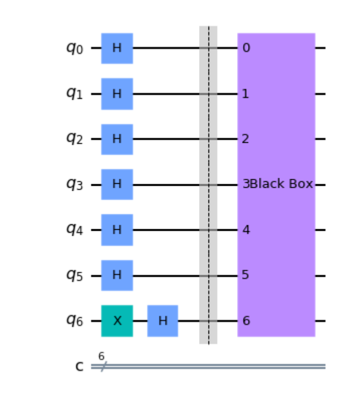
```
'110011'
```

In [5]:
```
# making the circuit corresponding to the secret key
circuit = QuantumCircuit(6+1,6,name="Black Box")
# apply Cx gates from bit i to the last qubit
# to make it one whenever the i bit is  1
circuit.cx(5,6)
circuit.cx(4,6)
circuit.cx(1,6)
circuit.cx(0,6)
circuit.barrier()
circuit.draw(output='mpl')
```

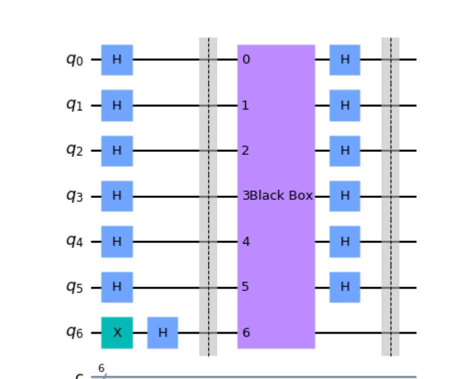


- The essence of the above circuit is **phase kickback**
- The LAST QUBIT is in the |-> state and all the above qubits are in the |+> state
- Since applying Cx over the |+> |-> pair *changes the control and not the target*, we observe the correctness of the algorithm

In [6]:
```
Q.append(circuit,circuit.qubits,circuit.clbits)
Q.draw(output='mpl')
```



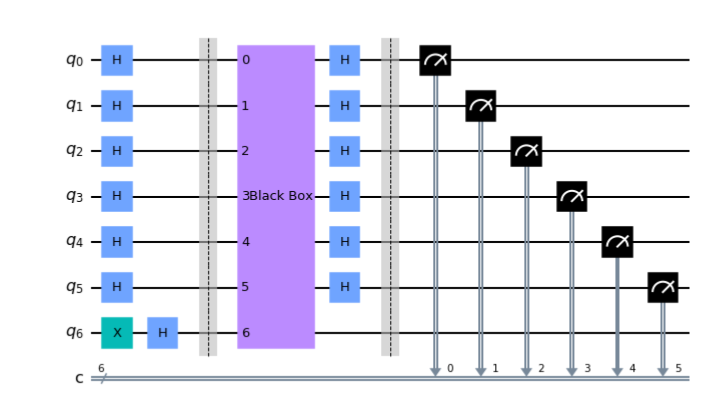
In [7]:
```
# now we need hadamard gates to again get back our qubits from the superposition
Q.h([0,1,2,3,4,5])
Q.barrier()
Q.draw(output='mpl')
```



Final Circuit

- Black box is actually the thing which would produce yes or no for our algorithm.

In [8]:
```
Q.measure([0,1,2,3,4,5],[0,1,2,3,4,5])
Q.draw(output='mpl')
```



Simulating here

In [9]:
```
b = Aer.get_backend('qasm_simulator')
result = execute(Q,backend = b,shots= 1).result()
counts = result.get_counts()
print(counts)
```

```
{'110011': 1}
```

The gist

- Well, we do not have the secret number
- We do not make the black box which gives you YES/No answer
- BUT, we can get the secret number by providing an EQUAL SUPERPOSITION of ALL THE QUBITS to the black box and then recieving the outputs.
- This is what the algorithm is

Generalizing black box for N qubits

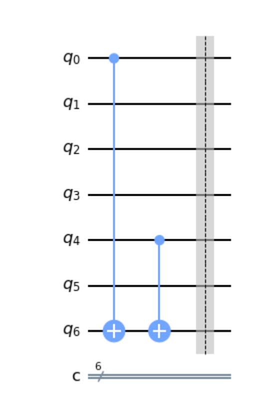
- This is a Black Box which contains a random bit string encoded into a black box.

In [15]:
```
def black_box(n):
    '''Returns an N-bit secret number black box / quantum circuit...'''
    secret = ""
    for i in range(n):
        b = np.random.randint(0,2)
        secret+=str(b)
    Q = QuantumCircuit(n+1,n,name="Black Box")
    for i,s in enumerate(secret[::-1]):
        if(s=='1'):
            # add a controlled x gate to make it 1
            Q.cx(i,n)
    Q.barrier()
    print("Secret key was :",secret,"!")
    display(Q.draw(output='mpl'))
    return Q
```

Testing for n = 6 qubits

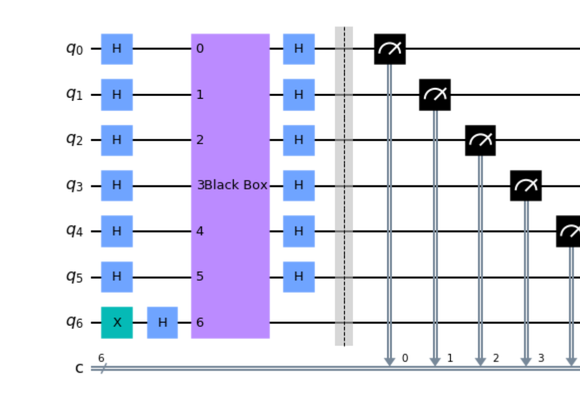
In [16]:
```
box = black_box(6)
Q = QuantumCircuit(6+1,6)
Q.h(range(6))
Q.x(6)
Q.h(6)
Q.append(box,box.qubits,box.clbits)
Q.h(range(6))
Q.barrier()
Q.measure(range(6),range(6))
```

```
Secret key was : 010001 !
```

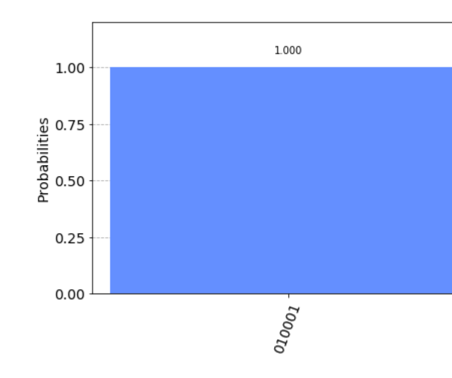


```
<qiskit.circuit.instructionset.InstructionSet at 0x1a6a1c626c8>
```

In [17]:
```
Q.draw(output='mpl')
```



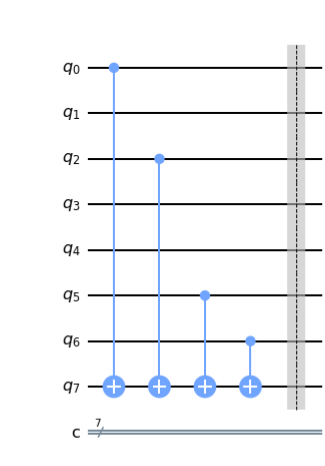
In [18]:
```
b = Aer.get_backend('qasm_simulator')
count = execute(Q,backend=b,shots=1).result().get_counts()
plot_histogram(count)
```



Testing for n = 7 qubits

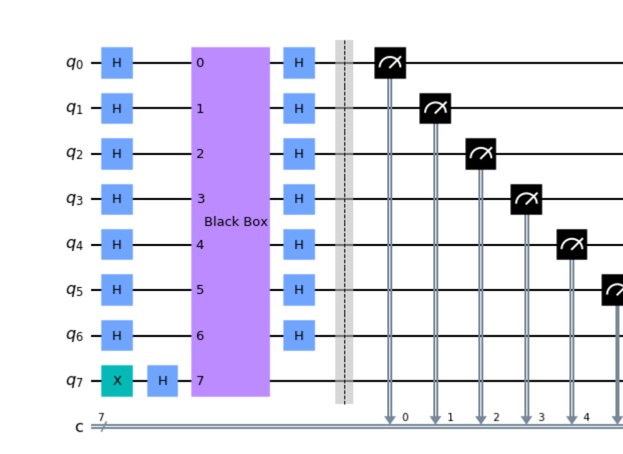
In [19]:
```
box = black_box(7)
Q = QuantumCircuit(7+1,7)
Q.h(range(7))
Q.x(7)
Q.h(7)
Q.append(box,box.qubits,box.clbits)
Q.h(range(7))
Q.barrier()
Q.measure(range(7),range(7))
```

```
Secret key was : 1100101 !
```

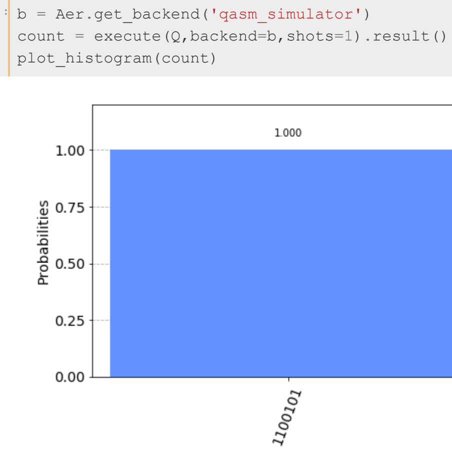


```
<qiskit.circuit.instructionset.InstructionSet at 0x1a6a18aa7c8>
```

In [20]:
```
Q.draw(output='mpl')
```



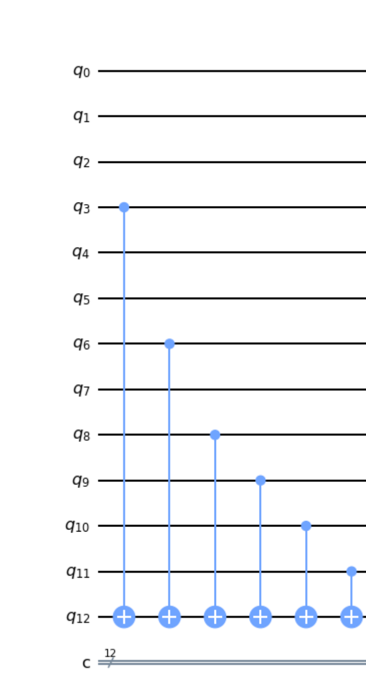
In [21]:
```
b = Aer.get_backend('qasm_simulator')
count = execute(Q,backend=b,shots=1).result().get_counts()
plot_histogram(count)
```



Testing for n = 12 qubits

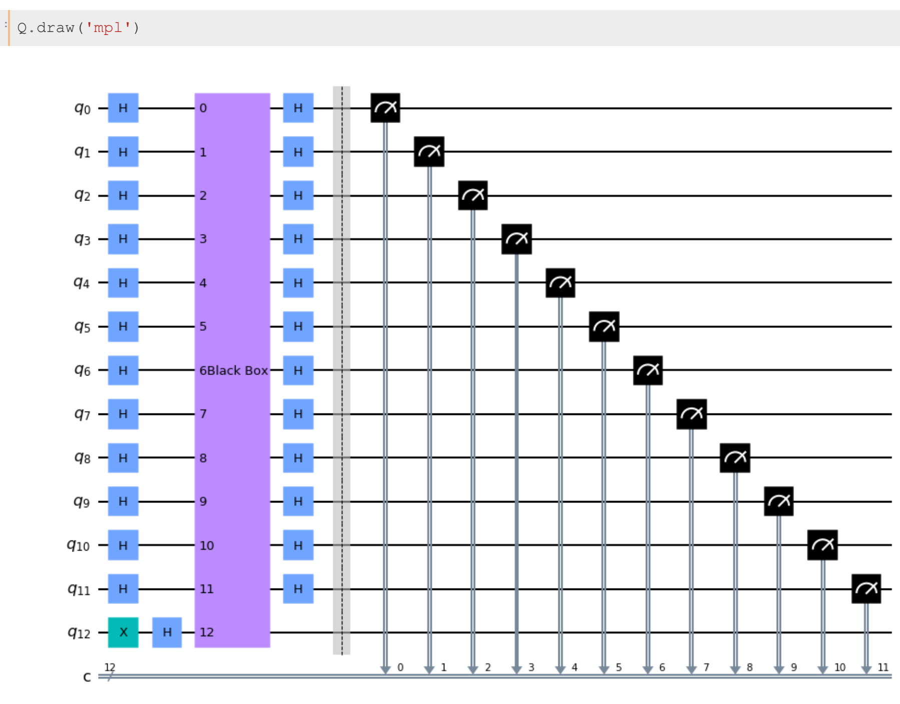
In [22]:
```
box = black_box(12)
Q = QuantumCircuit(12+1,12)
Q.h(range(12))
Q.x(12)
Q.h(12)
Q.append(box,box.qubits,box.clbits)
Q.h(range(12))
Q.barrier()
Q.measure(range(12),range(12))
```

```
Secret key was : 111101001000 !
```

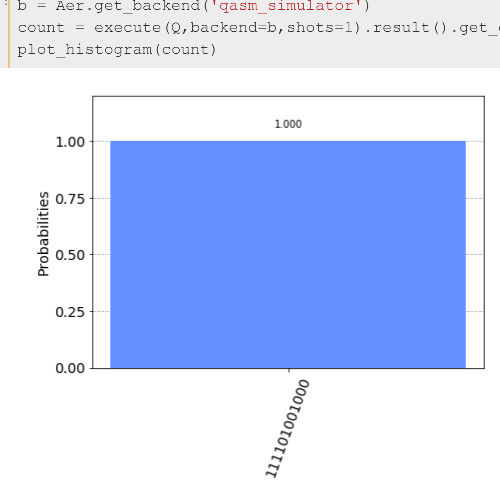


```
<qiskit.circuit.instructionset.InstructionSet at 0x1a6a197c748>
```

In [23]:
```
Q.draw('mpl')
```



In [24]:
```
b = Aer.get_backend('qasm_simulator')
count = execute(Q,backend=b,shots=1).result().get_counts()
plot_histogram(count)
```



- All the above simulations show that this algorithm correctly identifies the black box state correctly!

In [ ]: