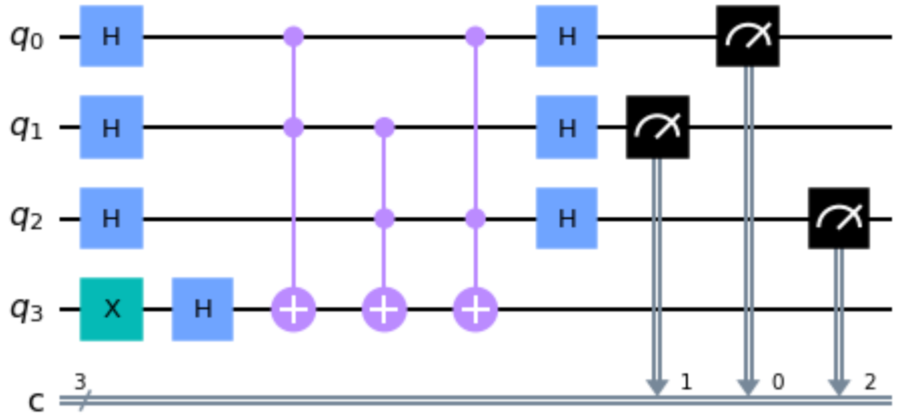
In [58]:
```
#f(x1,x2,x3) = x1x2+x2x3+x3x1, given a oracle for this function
#find whether f is constatnt or balanced.
```

In [59]:
```
import numpy as np

# Importing standard Qiskit libraries
from qiskit import QuantumCircuit, transpile, Aer, IBMQ
from qiskit.tools.jupyter import *
from qiskit.visualization import *
from ibm_quantum_widgets import *
from qiskit.providers.aer import QasmSimulator

# Loading your IBM Quantum account(s)
provider = IBMQ.load_account()
```

```
ibmqfactory.load_account:WARNING:2022-01-01 12:09:25,358: Credentials are already in use. The existing ac
count in the session will be replaced.
```

In [60]:
```
from qiskit import *

q = QuantumRegister(4, 'q')

M = ClassicalRegister(3, 'c')
###########################################################################################
#we are chechiking wheteher f(x1,x2,x3) = x1x2+x2x3+x3x1 is constant or balanced, using        #
#DJ algorithm                                                                             #
###########################################################################################

DJ_f = QuantumCircuit(q, M)
#building the circuit

DJ_f.x(q[3])
DJ_f.h(q)

#now the quantum equivalent of function f oracle

DJ_f.ccx(q[0], q[1], q[3])
DJ_f.ccx(q[1], q[2], q[3])
DJ_f.ccx(q[0], q[2], q[3])

#
for i in range(3):
    DJ_f.h(q[i])
    DJ_f.measure(q[i], M[i])


DJ_f.draw()
```

Out[60]:



In [ ]:

In [61]:

```
backend = Aer.get_backend('qasm_simulator')
qjob = execute(DJ_f, backend, shots=1000)

counts = qjob.result().get_counts()
print(counts)
```

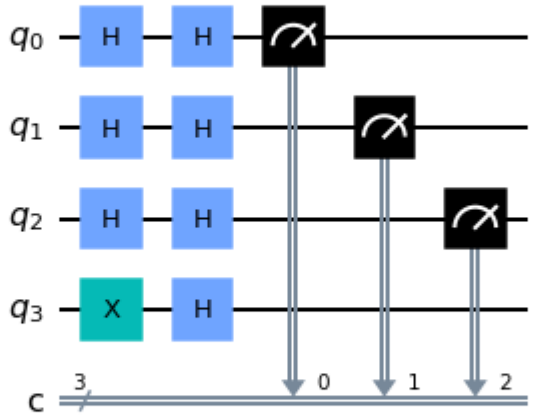
```
{'010': 261, '100': 256, '111': 232, '001': 251}
```

In [62]:

```
#So the function is balanced as the 000 has never been measured in the 1000 shots we have considered here
```

In [63]:
```
from qiskit import *

q = QuantumRegister(4, 'q')

M = ClassicalRegister(3, 'c')
###############################################################################################
#we are chechiking wheteher f(x1,x2,x3) = x1x2+x2x3+x3x1 is constant or balanced, using         #
#DJ algorithm                                                                                  #
###############################################################################################

DJ_f = QuantumCircuit(q, M)
#building the circuit

DJ_f.x(q[3])
DJ_f.h(q)

#now the quantum equivalent of function f oracle

#
for i in range(3):
    DJ_f.h(q[i])
    DJ_f.measure(q[i], M[i])


DJ_f.draw()
```

Out[63]:



In [ ]:

In [64]:

```
backend = Aer.get_backend('qasm_simulator')
qjob = execute(DJ_f, backend, shots=1000)

counts = qjob.result().get_counts()
print(counts)
```

```
{'000': 1000}
```

In [65]:

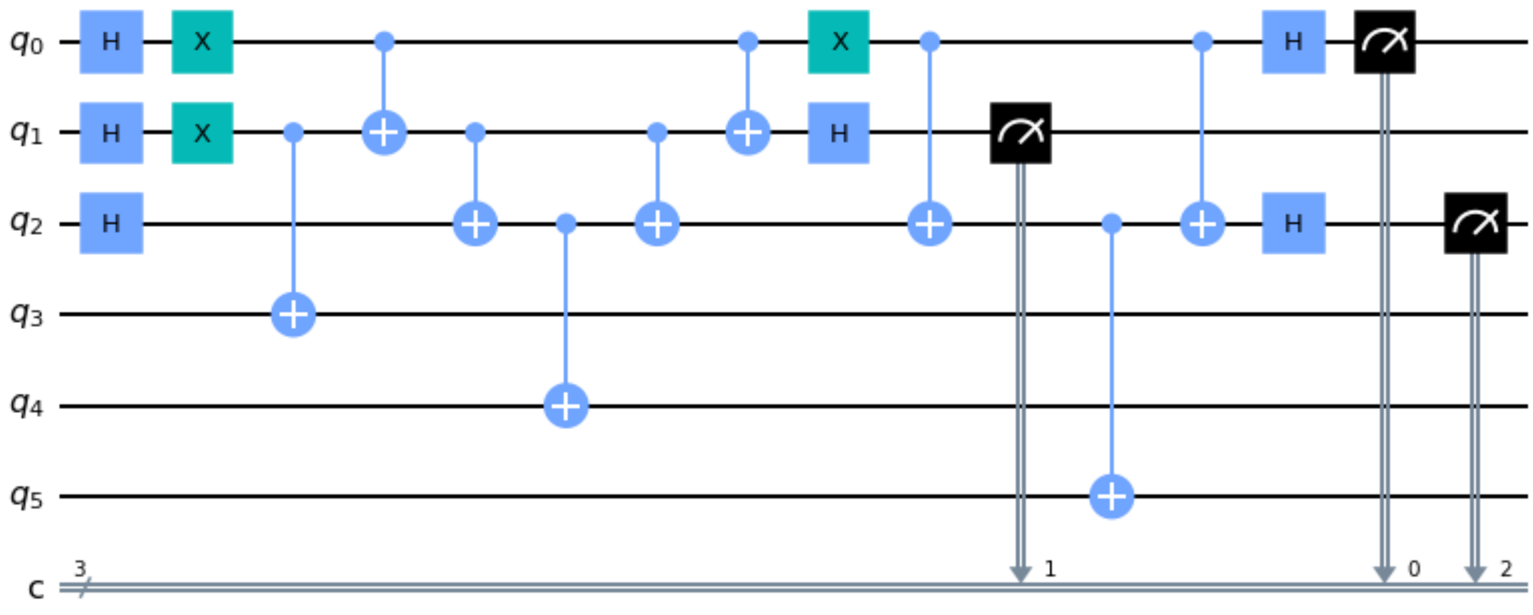
```
#here we can see that only 000 has been detected 1000 times
#so the function is a constatnt, which is indeed true.
```

In [66]:

```
#3 f(x1,x2,x3) = (1+x2, (1+x1)+x2+x3, x1+x3), using simon's algorithm find the hidden shift
```

In [67]:
```
q = QuantumRegister(6, 'q')

M = ClassicalRegister(3, 'c')

S_f=QuantumCircuit(q, M)

for i in range(3):
    S_f.h(q[i])

S_f.x(q[1])
S_f.cx(q[1], q[3])  #firts coordinate q[3] = x2+1

S_f.x(q[0])
S_f.cx(q[0], q[1])
S_f.cx(q[1], q[2])
S_f.cx(q[2], q[4]) #second coordinate q[4] = x1'+x2+x3

S_f.cx(q[1], q[2]) #q[2] = x3

S_f.cx(q[0], q[1]) # q[1] = x2

S_f.x(q[0]) # q[0] = x1

S_f.cx(q[0], q[2])
S_f.cx(q[2], q[5]) #q[5] = x1+x3

S_f.cx(q[0], q[2]) # q[2] = x3

for i in range(3):
    S_f.h(q[i])
    S_f.measure(q[i], M[i])



S_f.draw()
```

Out[67]:



In [68]:
```
backend = Aer.get_backend('qasm_simulator')
qjob = execute(S_f, backend, shots=1)

counts = qjob.result().get_counts()
print(counts)
```

```
{'000': 1}
```

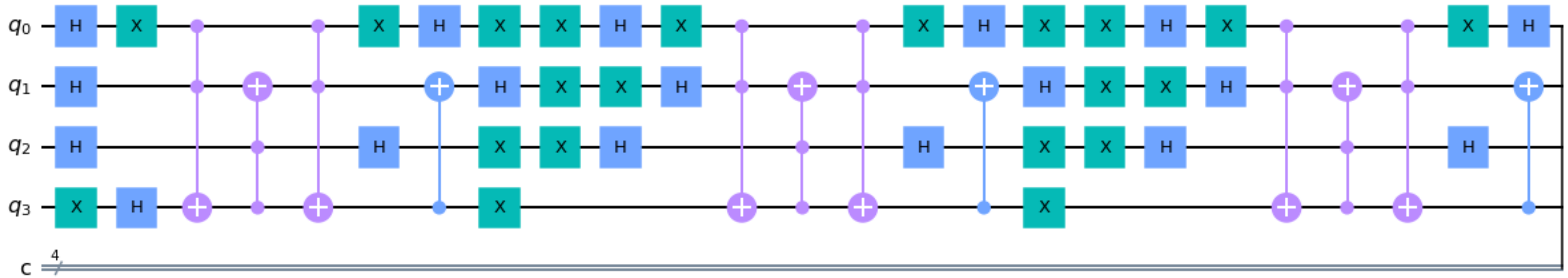
In [ ]:
```

```

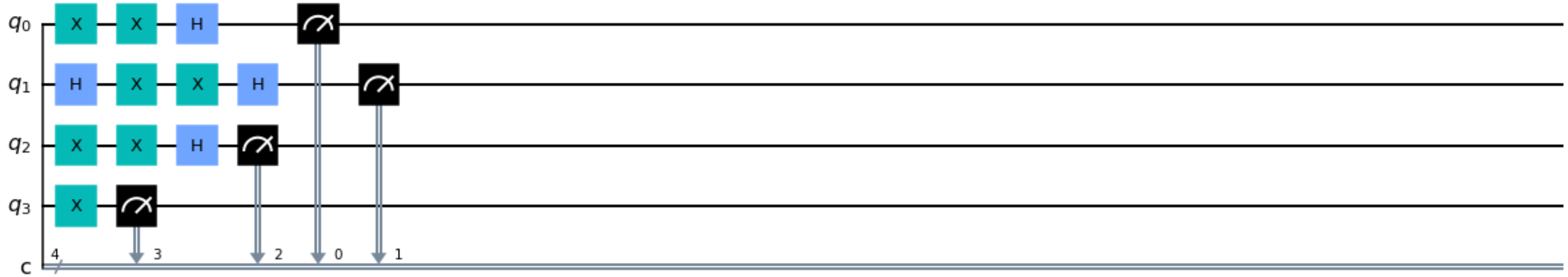
In [ ]:
```
#2 f(x1,x2,x3) = 1+x2x3+x1x2x3, using grover's search algorithm
#find the points where f(x1,x2,x3)=0
```

In [90]:
```
# from qiskit.circuit.library import *

#building an oracle for the function
#f(x1,x2,x3) = 1+x2x3+x1x2x3

def oracle():
    oracl = QuantumCircuit(4)
    oracl.x(0)
    oracl.ccx(0,1,3)
    oracl.ccx(3,2,1)
    oracl.ccx(0,1,3)
    oracl.cx(3,1)
    oracl.x(0)
    oracl.x(3)

q = QuantumRegister(4,'q')
M= ClassicalRegister(4, 'c')

G_f = QuantumCircuit(q, M)

G_f.x(q[3])
G_f.h(q)

for k in range(3):

    G_f.x(q[0])
    G_f.ccx(q[0],q[1],q[3])
    G_f.ccx(q[3],q[2],q[1])
    G_f.ccx(q[0],q[1],q[3])
    G_f.cx(q[3],q[1])
    G_f.x(q[0])
    G_f.x(q[3])


    G_f.h(q[0])
    G_f.h(q[1])
    G_f.h(q[2])
    G_f.x(q[0])
    G_f.x(q[1])
```

```
    G_f.x(q[2])

    #G_f.z(q[0])
    G_f.x(q[0])
    G_f.x(q[1])
    G_f.x(q[2])

    G_f.h(q[0])
    G_f.h(q[1])
    G_f.h(q[2])


for i in range(4):
    G_f.measure(q[i], M[i])


G_f.draw()
```

Out[90]:





In [92]:

```
backend = Aer.get_backend('qasm_simulator')
qjob = execute(G_f, backend, shots=1000)

counts = qjob.result().get_counts()
print(counts)
```

```
{'0001': 64, '1110': 64, '0100': 66, '0110': 68, '1011': 62, '1100': 51, '0101': 70, '1001': 64, '1010':
61, '1111': 63, '1000': 52, '0011': 63, '0111': 61, '0000': 60, '0010': 61, '1101': 70}
```

In [ ]: