In [1]:

```
# useful additional packages
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

# importing Qiskit
import qiskit
from qiskit import BasicAer, IBMQ
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister, execute
from qiskit.compiler import transpile
from qiskit.tools.monitor import job_monitor

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

In [2]:

```
n=13
```

In [15]:

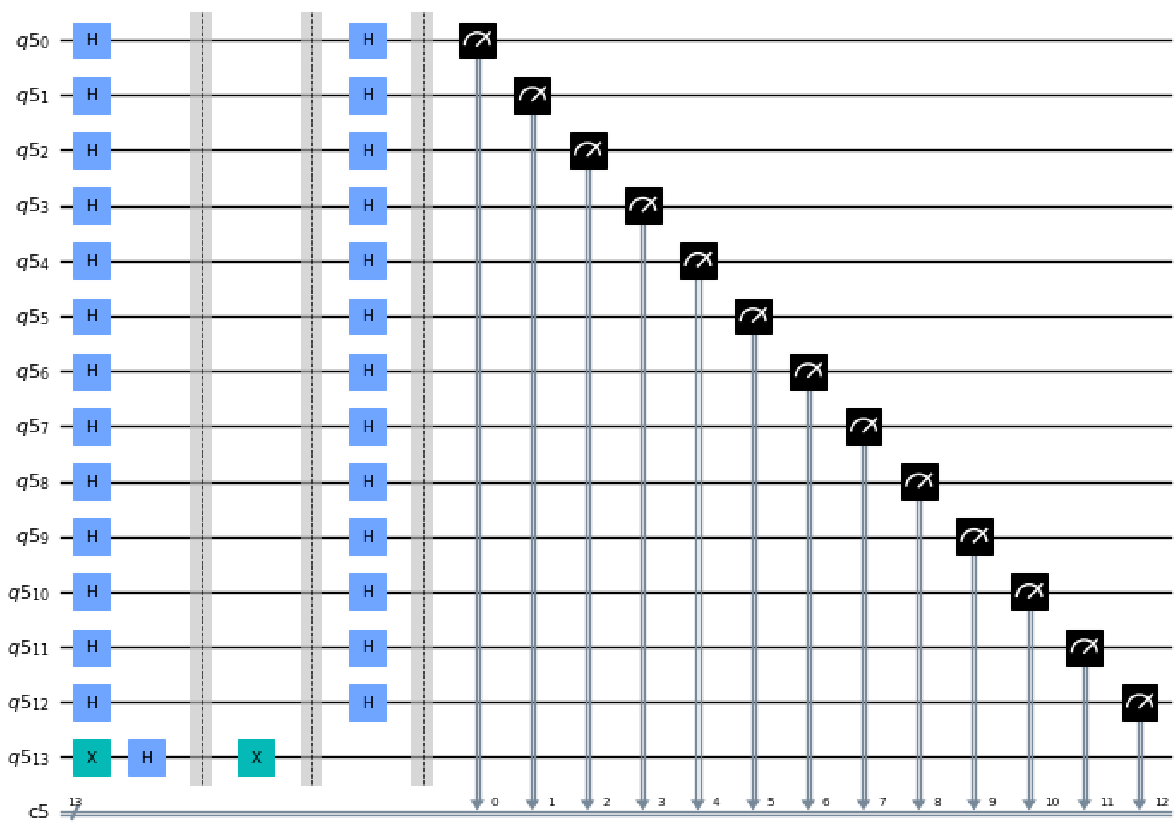
```
# Choose a type of oracle at random. With probability half it is constant,
# and with the same probability it is balanced
oracleType, oracleValue = np.random.randint(2), np.random.randint(2)

if oracleType == 0:
    print("The oracle returns a constant value ", oracleValue)
else:
    print("The oracle returns a balanced function")
    a = np.random.randint(1,2**n) # this is a hidden parameter for balanced ora


# Creating registers
# n qubits for querying the oracle and one qubit for storing the answer
qr = QuantumRegister(n+1) #all qubits are initialized to zero
# for recording the measurement on the first register
cr = ClassicalRegister(n)

circuitName = "DeutschJozsa"
djCircuit = QuantumCircuit(qr, cr)

# Create the superposition of all input queries in the first register by apply
for i in range(n):
    djCircuit.h(qr[i])

# Flip the second register and apply the Hadamard gate.
djCircuit.x(qr[n])
djCircuit.h(qr[n])

# Apply barrier to mark the beginning of the oracle
djCircuit.barrier()

if oracleType == 0:#If the oracleType is "0", the oracle returns oracleValue fo
    if oracleValue == 1:
        djCircuit.x(qr[n])
    else:
        djCircuit.id(qr[n])
else: # Otherwise, it returns the inner product of the input with a (non-zero b
    for i in range(n):
        if (a & (1 << i)):
            djCircuit.cx(qr[i], qr[n])
# Apply barrier to mark the end of the oracle
```

```
djCircuit.barrier()

# Apply Hadamard gates after querying the oracle
for i in range(n):
    djCircuit.h(qr[i])

# Measurement
djCircuit.barrier()
for i in range(n):
    djCircuit.measure(qr[i], cr[i])
```

```
The oracle returns a constant value  0
```

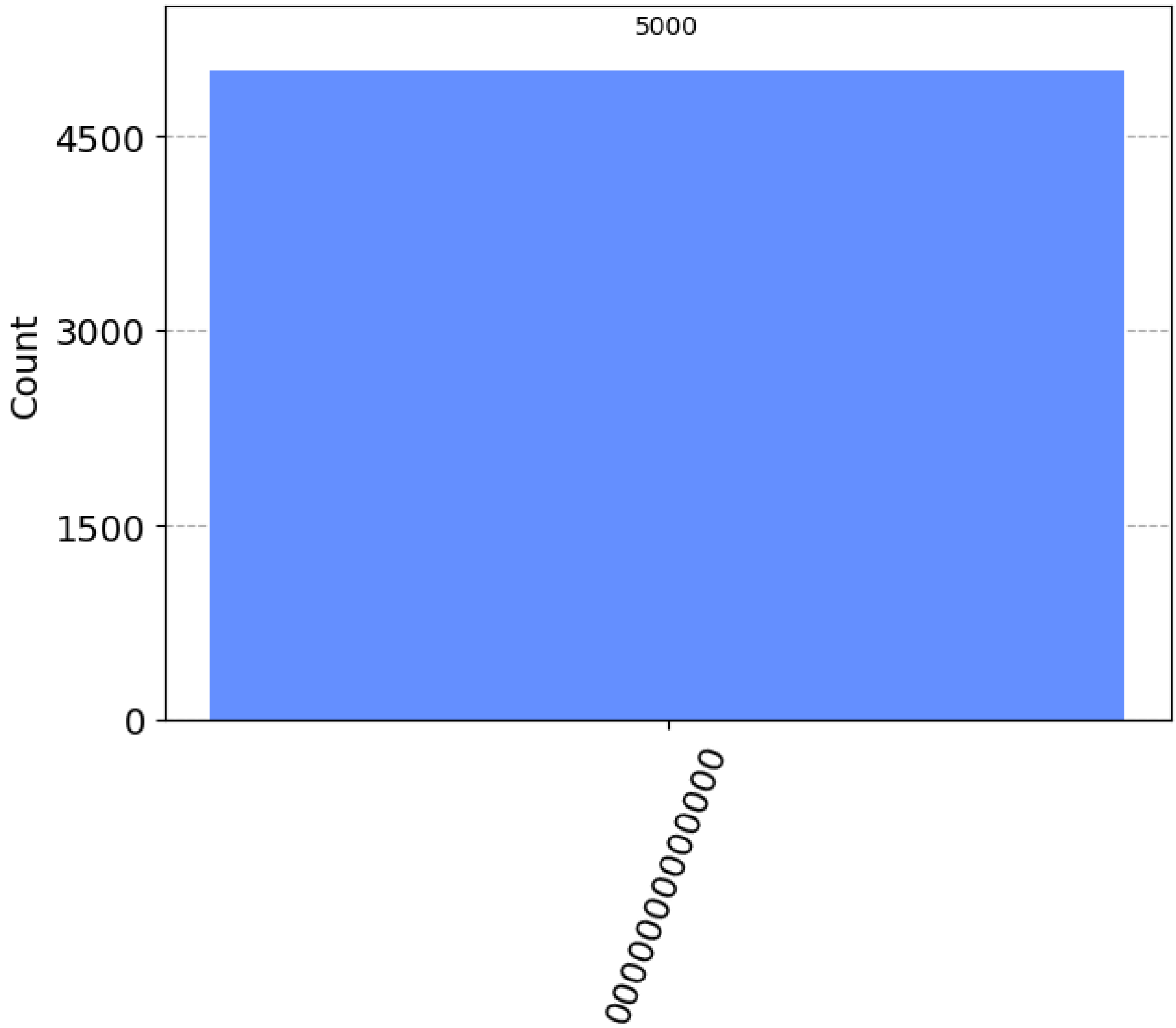
In [10]:
```
#draw the circuit
djCircuit.draw(output='mpl',scale=0.5)
```

Out[10]:




In [14]:
```
backend = BasicAer.get_backend('qasm_simulator')
shots = 5000
job = execute(djCircuit, backend=backend, shots=shots)
results = job.result()
answer = results.get_counts()

plot_histogram(answer)
```

Out[14]:




In [ ]: