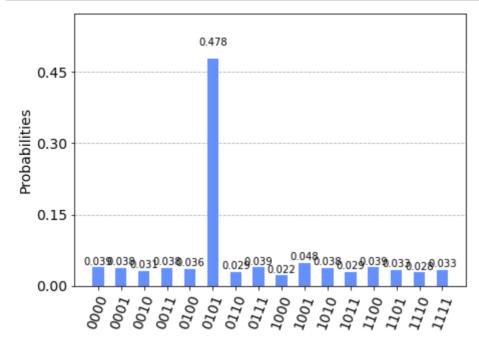
```
In [1]:
        ▶ | from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister
           from qiskit import Aer,execute
           import numpy as np
           import pandas as pd
           import matplotlib.pyplot as plt
           from qiskit.compiler import transpile
           from qiskit.transpiler import PassManager
           from qiskit.visualization import plot histogram
In [2]: ▶ def diffuser(nqubits):
               qc = QuantumCircuit(nqubits)
               # Apply transformation |s> -> |00...0> (H-gates)
               for qubit in range(nqubits):
                   qc.h(qubit)
               #Apply transformation |00...0> -> |11..1> (X-gates)
               for qubit in range(nqubits):
                   qc.x(qubit)
               # Do multicontrolled-Z gate
               qc.h(nqubits-1)
               qc.mct(list(range(nqubits-1)),nqubits-1)
               qc.h(nqubits-1)
               #Apply transformation |11..1> -> |00..0>
               for qubit in range(nqubits):
                   qc.x(qubit)
               # Apply transformation |11...1> -> |s>
               for qubit in range(nqubits):
                   qc.h(qubit)
               # We will return the diffuser as a gate
               U_s = qc.to_gate()
               U s.name = "$U s$"
               return U s
[1,0,3],
                    [2,0,3],
                    [3,1,2]
# compute
               for ce in c_eff:
                   qc.cx(ce,eff_reg[ce[0]])
               # mct
               qc.mcx(eff_reg,output_reg,anci,mode='basic')
               # uncompute
               for ce in c eff:
                   qc.cx(ce,eff_reg[ce[0]])
In [5]:
       def add_iter(qc,cf,c_reg,eff_reg,anci,cls):
               lights_out_oracle(qc,cf,c_reg,eff_reg,anci,cls)
               qc.barrier()
               qc.append(diffuser(n),c_reg)
```

```
In [6]:
        ▶ # Construct the circuit for 2x2 lights out board
            n = 4
            # initial state of board
            lights = [0,1,0,1]
            # cause_reg ---> contains possible solution states. "Causes" the state of board to d
            # effect_reg --> contains the tiles of the board on which the cause_reg acts
            # output_reg --> qubit in the |-> state for "marking" the solution
            # ancilla ----> for using with mct in order to reduce cost.
            # cbits ----> classical bits for storing the solution when read.
            cause_reg = QuantumRegister(n,name='cause')
            effect reg = QuantumRegister(n,name='effect')
            output reg = QuantumRegister(1,name='out')
            ancilla = QuantumRegister(n-2,name='anci')
            cbits = ClassicalRegister(n)
            # circuit
            quc = QuantumCircuit(cause reg,effect reg,output reg,ancilla,cbits)
            # initialize all possible solutions
            quc.h(cause_reg)
            # initialize |->
            quc.x(output reg)
            quc.h(output reg)
            quc.barrier()
            for 1 in range(n):
                if lights[l] is 0:
                    quc.x(effect reg[1])
            # add 1 iteration of oracle and diffuser
            add_iter(quc,c_eff,cause_reg,effect_reg,ancilla,cbits)
            quc.measure(cause_reg,cbits)
            quc.draw('mpl')
```

Out[6]:





/