In [2]:

```
# Required imports

from qiskit import QuantumCircuit, QuantumRegister, ClassicalReg
#from qiskit_aer import AerSimulator
from qiskit.visualization import plot_histogram
from qiskit.result import marginal_distribution
from qiskit.circuit.library import UGate
from numpy import pi, random
```

In [3]:

```python
qubit = QuantumRegister(1, "Q")
ebit0 = QuantumRegister(1, "A")
ebit1 = QuantumRegister(1, "B")
a = ClassicalRegister(1, "a")
b = ClassicalRegister(1, "b")

protocol = QuantumCircuit(qubit, ebit0, ebit1, a, b)

# Prepare ebit used for teleportation
protocol.h(ebit0)
protocol.cx(ebit0, ebit1)
protocol.barrier()

# Alice's operations
protocol.cx(qubit, ebit0)
protocol.h(qubit)
protocol.barrier()

# Alice measures and sends classical bits to Bob
protocol.measure(ebit0, a)
protocol.measure(qubit, b)
protocol.barrier()

# Bob uses the classical bits to conditionally apply gates
with protocol.if_test((a, 1)):
    protocol.x(ebit1)
with protocol.if_test((b, 1)):
    protocol.z(ebit1)

display(protocol.draw())
```

```
                ░      ┌───┐ ░    ┌─┐ ░                             
  Q: ───────────░───■──┤ H ├─░────┤M├─░─────────────────────────────
     ┌───┐      ░ ┌─┴─┐└───┘ ░ ┌─┐└╥┘ ░                             
  A: ┤ H ├──■───░─┤ X ├──────░─┤M├─╫──░─────────────────────────────
     └───┘┌─┴─┐ ░ └───┘      ░ └╥┘ ║  ░ ┌────── ┌───┐ ───────┐ ┌────
  B: ─────┤ X ├─░────────────░──╫──╫──░─┤ If-0 ─┤ X ├─ End-0 ├─┤ If-
          └───┘ ░            ░  ║  ║  ░ └──╥─── └───┘ ───────┘ └──╥─
                                ║  ║    ┌──╨──┐                   ║ 
a: 1/═══════════════════════════╩══╬════╡ 0x1 ╞═══════════════════╬═
                                0  ║    └─────┘                ┌──╨─
b: 1/══════════════════════════════╩═══════════════════════════╡ 0x1
                                   0                           └────
«
«  Q: ──────────
«
«  A: ──────────
«       ───────┐
«  B:   End-0 ├─
«       ───────┘
«a: 1/══════════
«
«b: 1/══════════
«
```

In [4]:

```
random_gate = UGate(
    theta=random.random() * 2 * pi,
    phi=random.random() * 2 * pi,
    lam=random.random() * 2 * pi,
)

display(random_gate.to_matrix())
```

```
array([[ 0.99981107+0.j        , -0.01084701-0.01612971j],
       [-0.00295527+0.01921176j, -0.90484013+0.42530754j]])
```

In [5]:

```
# Create a new circuit including the same bits and qubits used i
# teleportation protocol.

test = QuantumCircuit(qubit, ebit0, ebit1, a, b)

# Start with the randomly selected gate on Q

test.append(random_gate, qubit)
test.barrier()

# Append the entire teleportation protocol from above.

test = test.compose(protocol)
test.barrier()

# Finally, apply the inverse of the random unitary to B and measu

test.append(random_gate.inverse(), ebit1)

result = ClassicalRegister(1, "Result")
test.add_register(result)
test.measure(ebit1, result)

display(test.draw())
```

```
          ┌────────────────────────────┐ ░                ░      ┌───┐ ░ 
      Q: ─┤ U(0.038878,1.7234,0.97878) ├─░────────────────░───■──┤ H ├─░─
          └────────────────────────────┘ ░ ┌───┐          ░ ┌─┴─┐└───┘ ░ 
      A: ────────────────────────────────░─┤ H ├──■───────░─┤ X ├──────░─
                                         ░ └───┘┌─┴─┐     ░ └───┘      ░ 
      B: ────────────────────────────────░──────┤ X ├─────░────────────░─
                                         ░      └───┘     ░            ░ 
   a: 1/═════════════════════════════════════════════════════════════════
                                                                         
   b: 1/═════════════════════════════════════════════════════════════════
                                                                         
Result: 1/═══════════════════════════════════════════════════════════════
                                                                         
«                                                          ░ »
«      Q: ─────────────────────────────────────────────────░─»
«                                                          ░ »
«      A: ─────────────────────────────────────────────────░─»
«         ┌──────  ┌───┐ ───────┐ ┌──────  ┌───┐ ───────┐ ░ »
«      B: ┤ If-0  ├┤ X ├┤  End-0 ├─┤ If-0  ├┤ Z ├┤  End-0 ├─░─»
«         └──╥───  └───┘ ───────┘ └──╥───  └───┘ ───────┘ ░ »
«         ┌──╨──┐                    ║                       »
«   a: 1/═╡ 0x1 ╞════════════════════╬═══════════════════════»
«         └─────┘                 ┌──╨──┐                    »
«   b: 1/═════════════════════════╡ 0x1 ╞════════════════════»
«                                 └─────┘                    »
«Result: 1/══════════════════════════════════════════════════»
«                                                            »
«                                         
«      Q: ────────────────────────────────
«                                         
«      A: ────────────────────────────────
«         ┌────────────────────────────┐┌─┐
«      B: ┤ U(-0.038878,-0.97878,-1.7234) ├┤M├
«         └────────────────────────────┘└╥┘
«   a: 1/═══════════════════════════════╬═
«                                       ║ 
«   b: 1/═══════════════════════════════╬═
«                                       ║ 
«Result: 1/═════════════════════════════╩═
«                                       0 
```

In [9]:

```python
result = AerSimulator().run(test).result()
statistics = result.get_counts()
display(plot_histogram(statistics))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[9], line 1
----> 1 result = AerSimulator().run(test).result()
      2 statistics = result.get_counts()
      3 display(plot_histogram(statistics))

NameError: name 'AerSimulator' is not defined
```

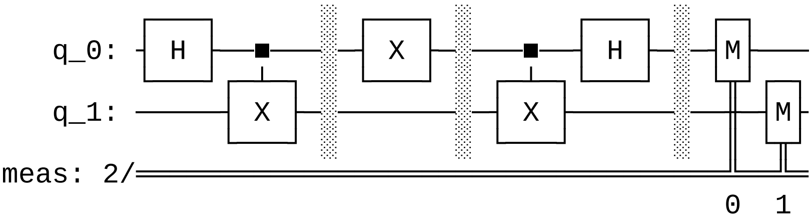
In [10]:
```
filtered_statistics = marginal_distribution(statistics, [2])
display(plot_histogram(filtered_statistics))
```

```
---------------------------------------------------------------------------
--------
NameError                                 Traceback (most recent ca
ll last)
Cell In[10], line 1
----> 1 filtered_statistics = marginal_distribution(statistics,
[2])
      2 display(plot_histogram(filtered_statistics))

NameError: name 'statistics' is not defined
```

In [13]:
```
# Required imports

from qiskit import QuantumCircuit, QuantumRegister, ClassicalReg
#from qiskit_aer.primitives import Sampler
#from qiskit_aer import AerSimulator
from qiskit.visualization import plot_histogram
```

In [14]:
```
c = "1"
d = "0"
```

In [15]:
```
protocol = QuantumCircuit(2)

# Prepare ebit used for superdense coding
protocol.h(0)
protocol.cx(0, 1)
protocol.barrier()

# Alice's operations
if d == "1":
    protocol.z(0)
if c == "1":
    protocol.x(0)
protocol.barrier()

# Bob's actions
protocol.cx(0, 1)
protocol.h(0)
protocol.measure_all()

display(protocol.draw())
```



In [16]:

```

```

```
---------------------------------------------------------------------------
--------
NameError                                 Traceback (most recent ca
ll last)
Cell In[16], line 1
----> 1 result = Sampler().run(protocol).result()
      2 statistics = result.quasi_dists[0].binary_probabilities()
      4 for outcome, frequency in statistics.items():

NameError: name 'Sampler' is not defined
```

In [17]:

```
from qiskit.quantum_info import Statevector, Operator
from numpy import sqrt
```

In [18]:

```
# Required imports

from qiskit import QuantumCircuit
from qiskit_aer.primitives import Sampler
from numpy import pi
from numpy.random import randint
```

```
---------------------------------------------------------------------------
--------
ModuleNotFoundError                       Traceback (most recent ca
ll last)
Cell In[18], line 4
      1 # Required imports
      3 from qiskit import QuantumCircuit
----> 4 from qiskit_aer.primitives import Sampler
      5 from numpy import pi
      6 from numpy.random import randint

ModuleNotFoundError: No module named 'qiskit_aer'
```

In [19]:

```
def chsh_game(strategy):
    """Plays the CHSH game
    Args:
        strategy (callable): A function that takes two bits (as `
            returns two bits (also as `int`s). The strategy must
            rules of the CHSH game.
    Returns:
        int: 1 for a win, 0 for a loss.
    """
    # Referee chooses x and y randomly
    x, y = randint(0, 2), randint(0, 2)

    # Use strategy to choose a and b
    a, b = strategy(x, y)

    # Referee decides if Alice and Bob win or lose
    if (a != b) == (x & y):
        return 1  # Win
    return 0  # Lose
```

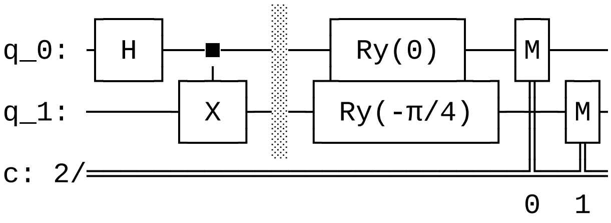
In [20]:

```
def chsh_circuit(x, y):
    """Creates a `QuantumCircuit` that implements the best CHSH s
    Args:
        x (int): Alice's bit (must be 0 or 1)
        y (int): Bob's bit (must be 0 or 1)
    Returns:
        QuantumCircuit: Circuit that, when run, returns Alice and
            answer bits.
    """
    qc = QuantumCircuit(2, 2)
    qc.h(0)
    qc.cx(0, 1)
    qc.barrier()

    # Alice
    if x == 0:
        qc.ry(0, 0)
    else:
        qc.ry(-pi / 2, 0)
    qc.measure(0, 0)

    # Bob
    if y == 0:
        qc.ry(-pi / 4, 1)
    else:
        qc.ry(pi / 4, 1)
    qc.measure(1, 1)

    return qc
```

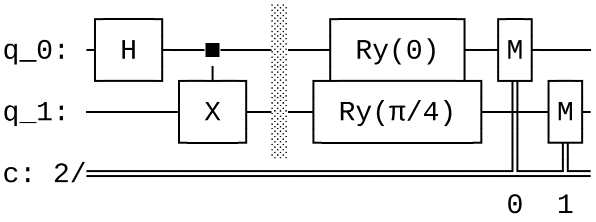
In [21]:

```
# Draw the four possible circuits

print("(x,y) = (0,0)")
display(chsh_circuit(0, 0).draw())

print("(x,y) = (0,1)")
display(chsh_circuit(0, 1).draw())

print("(x,y) = (1,0)")
display(chsh_circuit(1, 0).draw())

print("(x,y) = (1,1)")
display(chsh_circuit(1, 1).draw())
```

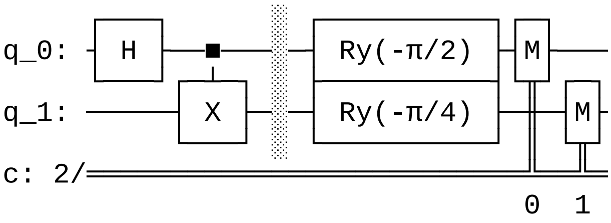
```
(x,y) = (0,0)
```



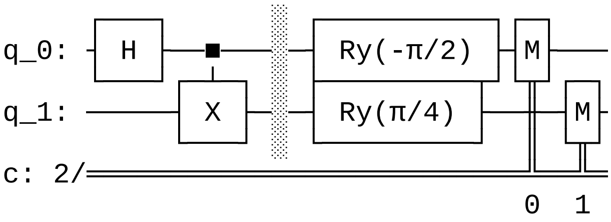
```
(x,y) = (0,1)
```



```
(x,y) = (1,0)
```



```
(x,y) = (1,1)
```



In [22]:

```
sampler = Sampler()


def quantum_strategy(x, y):
    """Carry out the best strategy for the CHSH game.
    Args:
        x (int): Alice's bit (must be 0 or 1)
        y (int): Bob's bit (must be 0 or 1)
    Returns:
        (int, int): Alice and Bob's answer bits (respectively)
    """
    # `shots=1` runs the circuit once
    result = sampler.run(chsh_circuit(x, y), shots=1).result()
    statistics = result.quasi_dists[0].binary_probabilities()
    bits = list(statistics.keys())[0]
    a, b = bits[0], bits[1]
    return a, b
```

```
---------------------------------------------------------------------------
--------
NameError                                 Traceback (most recent ca
ll last)
Cell In[22], line 1
----> 1 sampler = Sampler()
      4 def quantum_strategy(x, y):
      5     """Carry out the best strategy for the CHSH game.
      6     Args:
      7         x (int): Alice's bit (must be 0 or 1)
   (...)
     10         (int, int): Alice and Bob's answer bits (respective
ly)
     11     """

NameError: name 'Sampler' is not defined
```

In [23]:

```
NUM_GAMES = 1000
TOTAL_SCORE = 0

for _ in range(NUM_GAMES):
    TOTAL_SCORE += chsh_game(quantum_strategy)

print("Fraction of games won:", TOTAL_SCORE / NUM_GAMES)
```

```
---------------------------------------------------------------------------
--------
NameError                                 Traceback (most recent ca
ll last)
Cell In[23], line 5
      2 TOTAL_SCORE = 0
      4 for _ in range(NUM_GAMES):
----> 5     TOTAL_SCORE += chsh_game(quantum_strategy)
      7 print("Fraction of games won:", TOTAL_SCORE / NUM_GAMES)

NameError: name 'quantum_strategy' is not defined
```

In [24]:

```
def classical_strategy(x, y):
    """An optimal classical strategy for the CHSH game
    Args:
        x (int): Alice's bit (must be 0 or 1)
        y (int): Bob's bit (must be 0 or 1)
    Returns:
        (int, int): Alice and Bob's answer bits (respectively)
    """
    # Alice's answer
    if x == 0:
        a = 0
    elif x == 1:
        a = 1

    # Bob's answer
    if y == 0:
        b = 1
    elif y == 1:
        b = 0

    return a, b
```

In [25]:

```
NUM_GAMES = 1000
TOTAL_SCORE = 0

for _ in range(NUM_GAMES):
    TOTAL_SCORE += chsh_game(classical_strategy)

print("Fraction of games won:", TOTAL_SCORE / NUM_GAMES)
```

```
-------------------------------------------------------------------
--------
NameError                                 Traceback (most recent ca
ll last)
Cell In[25], line 5
      2 TOTAL_SCORE = 0
      4 for _ in range(NUM_GAMES):
----> 5     TOTAL_SCORE += chsh_game(classical_strategy)
      7 print("Fraction of games won:", TOTAL_SCORE / NUM_GAMES)

Cell In[19], line 11, in chsh_game(strategy)
      2 """Plays the CHSH game
      3 Args:
      4     strategy (callable): A function that takes two bits (as
`int`s) and
   (...)
      8     int: 1 for a win, 0 for a loss.
      9 """
     10 # Referee chooses x and y randomly
---> 11 x, y = randint(0, 2), randint(0, 2)
     13 # Use strategy to choose a and b
     14 a, b = strategy(x, y)

NameError: name 'randint' is not defined
```

In [ ]: