In [2]:

```python
from qiskit_ibm_provider import IBMProvider

# Load your account
provider = IBMProvider()

# List available backends
print("Available Backends:")
print(provider.backends())
```

```
Available Backends:
[<IBMBackend('ibm_kyiv')>, <IBMBackend('ibm_brisbane')>, <IBMBackend('ibm_sherbrooke')>]
```

In [5]:

```python
from qiskit_ibm_provider import IBMProvider

# Step 1: Load IBM Quantum account
provider = IBMProvider()

# Step 2: List all available backends
backends = provider.backends()

# Step 3: Check the queue length for each backend
print("Available backends and their queue lengths:")
for backend in backends:
    status = backend.status()
    print(f"{backend.name}: Queue Length = {status.pending_jobs}")
```

```
Available backends and their queue lengths:
ibm_brisbane: Queue Length = 0
ibm_sherbrooke: Queue Length = 0
ibm_kyiv: Queue Length = 0
```

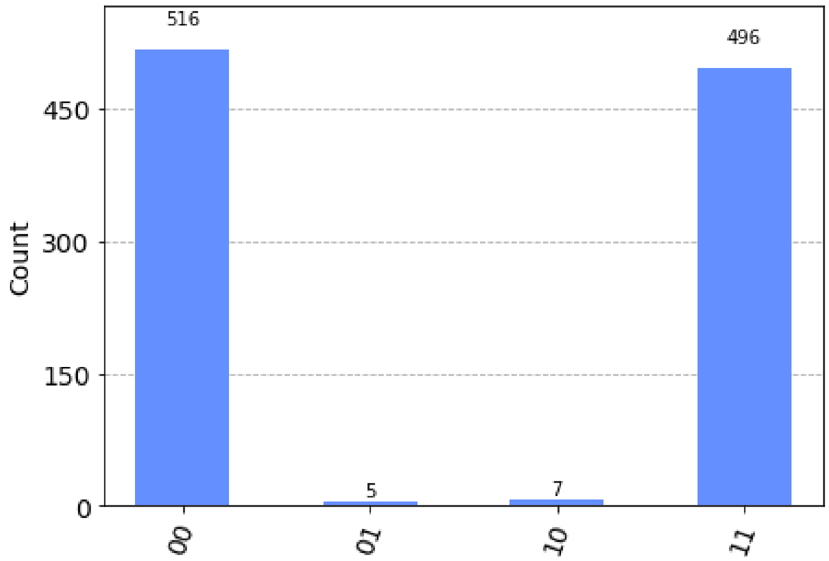
In [6]:

```python
from qiskit_ibm_provider import IBMProvider
from qiskit import QuantumCircuit, transpile
from qiskit.visualization import plot_histogram

# Step 1: Create a Bell State circuit
qc = QuantumCircuit(2, 2)
qc.h(0)  # Apply Hadamard gate
qc.cx(0, 1)  # Apply CNOT gate
qc.measure([0, 1], [0, 1])  # Measure both qubits

# Step 2: Load IBM Quantum account
provider = IBMProvider()

# Step 3: Use the 'ibm_kyiv' backend
backend = provider.get_backend('ibm_kyiv')

# Step 4: Transpile the circuit for the selected backend
compiled_circuit = transpile(qc, backend)

# Step 5: Run the circuit on the selected backend
job = backend.run(compiled_circuit, shots=1024)

# Step 6: Retrieve results
result = job.result()
counts = result.get_counts()

# Step 7: Visualize the results
print("Results from IBM Quantum:", counts)
plot_histogram(counts)
```

```
Results from IBM Quantum: {'10': 7, '11': 496, '01': 5, '00': 516}
```

Out[6]:

Out[6]:



In [ ]:

```
from qiskit import QuantumCircuit, Aer, transpile
from qiskit.visualization import plot_histogram
from qiskit_ibm_provider import IBMProvider

# Step 1: Create a Quantum Circuit with 3 qubits and 2 classical bits
qc = QuantumCircuit(3, 2)

# Step 2: Prepare the state to teleport on qubit 0 (e.g., a superposition state)
qc.h(0)

# Step 3: Create entanglement between qubits 1 and 2
qc.h(1)  # Hadamard on qubit 1
qc.cx(1, 2)   # CNOT gate with qubit 1 as control and qubit 2 as target

# Step 4: Bell measurement on qubits 0 and 1
qc.cx(0, 1)   # CNOT gate with qubit 0 as control and qubit 1 as target
qc.h(0)  # Hadamard gate on qubit 0
qc.measure(0, 0)  # Measure qubit 0
qc.measure(1, 1)  # Measure qubit 1

# Step 5: Conditional operations on qubit 2 (Bob's corrections)
qc.x(2).c_if(qc.clbits[1], 1)  # Apply X gate if classical bit 1 is 1
qc.z(2).c_if(qc.clbits[0], 1)  # Apply Z gate if classical bit 0 is 1

# Step 6: Visualize the circuit
print(qc)
qc.draw('mpl')

# Step 7: Simulate the circuit
simulator = Aer.get_backend('aer_simulator')
compiled_circuit = transpile(qc, simulator)
result = simulator.run(compiled_circuit, shots=1024).result()

# Step 8: Visualize the results
counts = result.get_counts()
print("Simulation Results:", counts)
plot_histogram(counts)

# Optional: Run on IBM Quantum hardware
provider = IBMProvider()
backend = provider.get_backend('ibm_sherbrooke')  # Use an available backend
compiled_circuit = transpile(qc, backend)
job = backend.run(compiled_circuit, shots=1024)
result = job.result()
counts = result.get_counts()
print("Results from IBM Quantum:", counts)
plot_histogram(counts)
```

```
q_0: ┤ H ├───────────■──┤ H ├┤M├──────────────────────────────
     ├───┤           │  └───┘└╥┘
q_1: ┤ H ├─────■───┤ X ├─┤M├───╫──────────────────────────────
     └───┘   ┌─┴─┐ └───┘ └╥┘   ║    ┌───┐         ┌───┐
q_2: ────────┤ X ├────────╫────╫────┤ X ├─────────┤ Z ├───────
             └───┘        ║    ║    └─╥─┘         └─╥─┘
                          ║    ║ ┌────╨────┐   ┌────╨────┐
c: 2/═════════════════════╩════╩═╡ c_1=0x1 ╞═══╡ c_0=0x1 ╞═
                          1    0 └─────────┘   └─────────┘
Simulation Results: {'01': 287, '00': 232, '10': 254, '11': 251}
```

In [ ]:

```
```