Experiment 2

Aim

Implementation of Single qubit gates on Qiskit.

Theory

Quantum computing uses quantum mechanical phenomena, similar as superposition and entanglement, to perform operations on data. Unlike classical computing, where bits can only be in one state(0 or 1) at a time, qubits can live in both states simultaneously due to the phenomenon of superposition. This allows for the representation and manipulation of complex data sets that are difficult to reuse using classical computing methods.

The Hadamard gate is a single- qubit gate that puts a qubit into a superposition state, allowing the qubit to exist in both the 0 and 1 states simultaneously. The Hadamard gate is represented by the matrix:

$$H = 1/\sqrt{2}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$$

The Pauli gates, including the X, Y, and Z gates, are also single-qubit gates that perform rotations around the X, Y, and Z axes of the Bloch sphere, respectively. The X, Y, and Z gates are represented by the matrices:

$$X = \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix},$$

$$Y = \begin{bmatrix}0 & -i\\ i & 0\end{bmatrix},$$

$$Z = \begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}$$

These gates can be used to manipulate qubits and perform operations on quantum data. In particular, the Hadamard gate is often used to create superposition states, which are important for many quantum algorithms, and the Pauli gates are often used to perform operations on qubits, such as flipping their states or measuring their values.

Qiskit provides a simple and intuitive way to implement these gates using its QuantumCircuit class and a variety of built-in gate methods, such as h, x, y, and z. These gates can be used to construct more complex quantum circuits and execute quantum algorithms on a variety of quantum devices and simulators.

Quantum computers are subject to several types of noise, which can lead to errors in quantum computations. One type of noise is decoherence, which occurs when a quantum system interacts with its environment and loses coherence. Decoherence limits the amount of time that a quantum system can maintain its superposition state, making it difficult to perform complex computations. To mitigate the effects of noise, quantum error correction techniques, such as the surface code, are used to detect and correct errors in quantum computations.

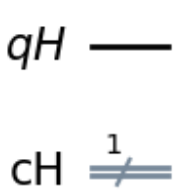
Lab Exercise

Code:

In [1]:
```
from qiskit import *
from qiskit.quantum_info import Statevector
import numpy as np

qH = QuantumRegister(1, name='qH')
cH = ClassicalRegister(1, name='cH')
circuit_H = QuantumCircuit(qH, cH)

qX = QuantumRegister(1, name='qX')
cX = ClassicalRegister(1, name='cX')
circuit_X = QuantumCircuit(qX, cX)

qY = QuantumRegister(1, name='qY')
cY = ClassicalRegister(1, name='cY')
circuit_Y = QuantumCircuit(qY, cY)

qZ = QuantumRegister(1, name='qZ')
cZ = ClassicalRegister(1, name='cZ')
circuit_Z = QuantumCircuit(qZ, cZ)

%matplotlib inline
```

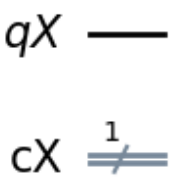
In [2]:
```
circuit_H.draw(output='mpl')
```

Out[2]:



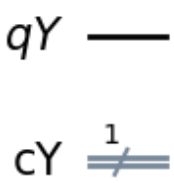
In [3]:
```
circuit_X.draw(output='mpl')
```

Out[3]:



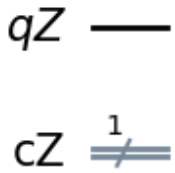
In [4]:
```
circuit_Y.draw(output='mpl')
```

Out[4]:



In [5]:
```
circuit_Z.draw(output='mpl')
```

Out[5]:



In [ ]:
```

```

Here, We prepare all qubits in State: |0⟩

In [6]:
```
circuit_H.barrier(qH, label='|0⟩')
circuit_H.reset(qH)

circuit_X.barrier(qX, label='|0⟩')
circuit_X.reset(qX)

circuit_Y.barrier(qY, label='|0⟩')
circuit_Y.reset(qY)

circuit_Z.barrier(qZ, label='|0⟩')
circuit_Z.reset(qZ)
```

Out[6]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x25e8b7fc5e0>
```

In [7]:
```
state_H_1 = Statevector.from_int(0, 2**1)
state_H_1 = state_H_1.evolve(circuit_H)
state_H_1.draw('latex')
```

Out[7]:

$$|0\rangle$$

In [8]:
```
state_X_1 = Statevector.from_int(0, 2**1)
state_X_1 = state_X_1.evolve(circuit_X)
state_X_1.draw('latex')
```

Out[8]:

$$|0\rangle$$

In [9]:
```
state_Y_1 = Statevector.from_int(0, 2**1)
state_Y_1 = state_Y_1.evolve(circuit_Y)
state_Y_1.draw('latex')
```

Out[9]:

$$|0\rangle$$

In [10]:
```
state_Z_1 = Statevector.from_int(0, 2**1)
state_Z_1 = state_Z_1.evolve(circuit_Z)
state_Z_1.draw('latex')
```

Out[10]:

$$|0\rangle$$

In [ ]:
```
```

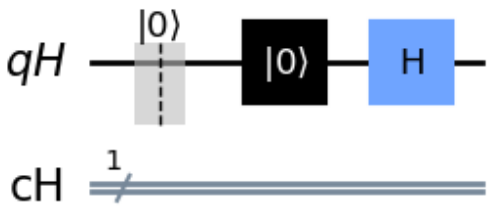
We now apply quantum logic gates on |0⟩

In [11]:
```
circuit_H.h(qH)
circuit_X.x(qX)
circuit_Y.y(qY)
circuit_Z.z(qZ)
```

Out[11]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x25e8b882310>
```

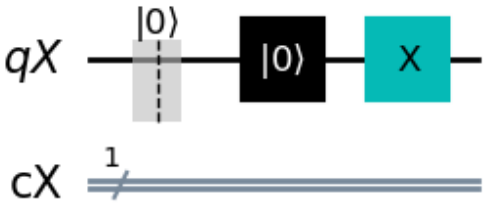
In [12]:
```
circuit_H.draw(output='mpl')
```

Out[12]:



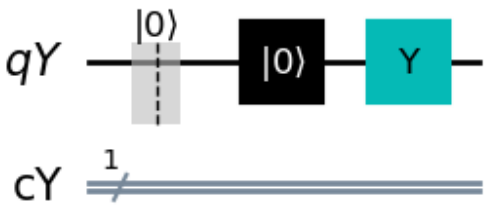
In [13]:
```
circuit_X.draw(output='mpl')
```

Out[13]:



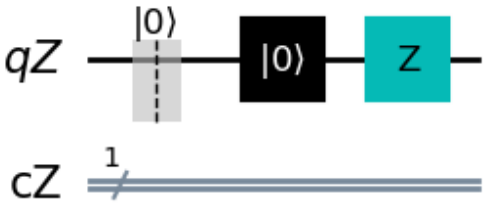
In [14]:
```
circuit_Y.draw(output='mpl')
```

Out[14]:



In [15]:
```
circuit_Z.draw(output='mpl')
```

Out[15]:



In [16]:
```
state_H_2 = Statevector.from_int(0, 2**1)
state_H_2 = state_H_2.evolve(circuit_H)
state_H_2.draw('latex')
```

Out[16]:

$$\frac{\sqrt{2}}{2}|0\rangle + \frac{\sqrt{2}}{2}|1\rangle$$

In [17]:
```
state_X_2 = Statevector.from_int(0, 2**1)
state_X_2 = state_X_2.evolve(circuit_X)
state_X_2.draw('latex')
```

Out[17]:

$$|1\rangle$$

In [18]:
```
state_Y_2 = Statevector.from_int(0, 2**1)
state_Y_2 = state_Y_2.evolve(circuit_Y)
state_Y_2.draw('latex')
```

Out[18]:

$$i|1\rangle$$

In [19]:
```
state_Z_2 = Statevector.from_int(0, 2**1)
state_Z_2 = state_Z_2.evolve(circuit_Z)
state_Z_2.draw('latex')
```

Out[19]:

$$|0\rangle$$

In [ ]:

Here, We prepare all qubits in State: |1⟩

In [20]:
```
circuit_H.barrier(qH, label='|1⟩')
circuit_H.reset(qH)
circuit_H.x(qH)

circuit_X.barrier(qX, label='|1⟩')
circuit_X.reset(qX)
circuit_X.x(qX)

circuit_Y.barrier(qY, label='|1⟩')
circuit_Y.reset(qY)
circuit_Y.x(qY)
circuit_Y.barrier()

circuit_Z.barrier(qZ, label='|1⟩')
circuit_Z.reset(qZ)
circuit_Z.x(qZ)
```

Out[20]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x25e8c0293d0>
```

In [21]:
```
state_H_3 = Statevector.from_int(0, 2**1)
state_H_3 = state_H_3.evolve(circuit_H)
state_H_3.draw('latex')
```

Out[21]:

$$|1\rangle$$

In [22]:
```
state_X_3 = Statevector.from_int(0, 2**1)
state_X_3 = state_X_3.evolve(circuit_X)
state_X_3.draw('latex')
```

Out[22]:

$$|1\rangle$$

In [23]:
```
state_Y_3 = Statevector.from_int(0, 2**1)
state_Y_3 = state_Y_3.evolve(circuit_Y)
state_Y_3.draw('latex')
```

Out[23]:

$$i|1\rangle$$

In [24]:
```
state_Z_4 = Statevector.from_int(0, 2**1)
state_Z_4 = state_Z_4.evolve(circuit_Z)
state_Z_4.draw('latex')
```

Out[24]:

$$|1\rangle$$

In [ ]:

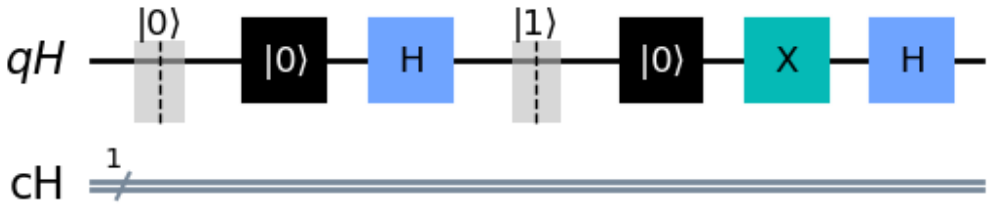
We now apply quantum logic gates on $|1\rangle$

In [25]:
```
circuit_H.h(qH)
circuit_X.x(qX)
circuit_Y.y(qY)
circuit_Z.z(qZ)
```

Out[25]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x25e8c034370>
```

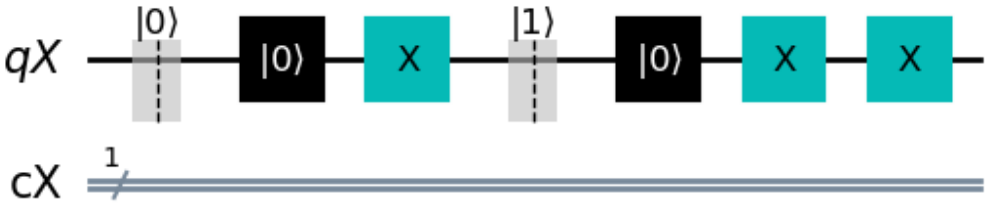
In [26]:
```
circuit_H.draw(output='mpl')
```

Out[26]:



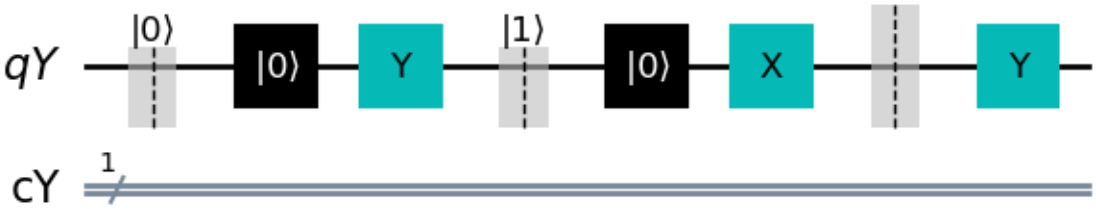
In [27]:
```
circuit_X.draw(output='mpl')
```

Out[27]:



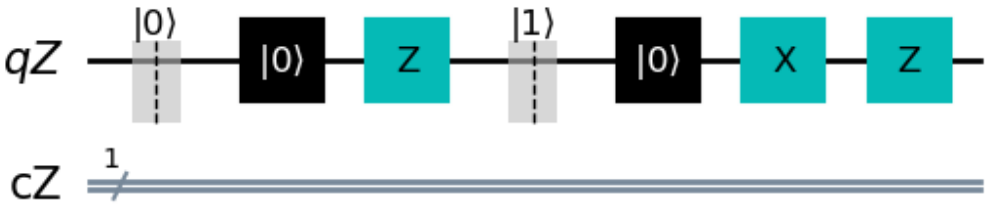
In [28]:
```
circuit_Y.draw(output='mpl')
```

Out[28]:



In [29]:
```
circuit_Z.draw(output='mpl')
```

Out[29]:



In [30]:
```
state_H_4 = Statevector.from_int(0, 2**1)
state_H_4 = state_H_4.evolve(circuit_H)
state_H_4.draw('latex')
```

Out[30]:

$$\frac{\sqrt{2}}{2}|0\rangle - \frac{\sqrt{2}}{2}|1\rangle$$

In [31]:
```
state_X_4 = Statevector.from_int(0, 2**1)
state_X_4 = state_X_4.evolve(circuit_X)
state_X_4.draw('latex')
```

Out[31]:

$$|0\rangle$$

In [32]:
```
state_Y_4 = Statevector.from_int(0, 2**1)
state_Y_4 = state_Y_4.evolve(circuit_Y)
state_Y_4.draw('latex')
```

Out[32]:

$$|0\rangle$$

In [33]:
```
state_Z_4 = Statevector.from_int(0, 2**1)
state_Z_4 = state_Z_4.evolve(circuit_Z)
state_Z_4.draw('latex')
```

Out[33]:

$$-|1\rangle$$

In [ ]:
```

```

Here, We prepare all qubits in State: |+⟩

In [34]:
```
circuit_H.barrier(qH, label='|+⟩')
circuit_H.reset(qH)
circuit_H.h(qH)

circuit_X.barrier(qX, label='|+⟩')
circuit_X.reset(qX)
circuit_X.h(qX)

circuit_Y.barrier(qY, label='|+⟩')
```

```
circuit_Y.reset(qY)
circuit_Y.h(qY)

circuit_Z.barrier(qZ, label='|+⟩')
circuit_Z.reset(qZ)
circuit_Z.h(qZ)
```

Out[34]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x25e8c2aa6d0>
```

In [35]:
```
state_H_5 = Statevector.from_int(0, 2**1)
state_H_5 = state_H_5.evolve(circuit_H)
state_H_5.draw('latex')
```

Out[35]:

$$-\frac{\sqrt{2}}{2}|0\rangle - \frac{\sqrt{2}}{2}|1\rangle$$

In [36]:
```
state_X_5 = Statevector.from_int(0, 2**1)
state_X_5 = state_X_5.evolve(circuit_X)
state_X_5.draw('latex')
```

Out[36]:

$$\frac{\sqrt{2}}{2}|0\rangle + \frac{\sqrt{2}}{2}|1\rangle$$

In [37]:
```
state_Y_5 = Statevector.from_int(0, 2**1)
state_Y_5 = state_Y_5.evolve(circuit_Y)
state_Y_5.draw('latex')
```

Out[37]:

$$\frac{\sqrt{2}}{2}|0\rangle + \frac{\sqrt{2}}{2}|1\rangle$$

In [38]:
```
state_Z_5 = Statevector.from_int(0, 2**1)
state_Z_5 = state_Z_5.evolve(circuit_Z)
state_Z_5.draw('latex')
```

Out[38]:

$$-\frac{\sqrt{2}}{2}|0\rangle - \frac{\sqrt{2}}{2}|1\rangle$$

In [ ]:
```

```

We now apply quantum logic gates on $|+\rangle$

In [39]:
```
circuit_H.h(qH)
circuit_X.x(qX)
circuit_Y.y(qY)
circuit_Z.z(qZ)
```

Out[39]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x25e8c2bb670>
```

In [40]:
```
circuit_H.draw(output='mpl')
```

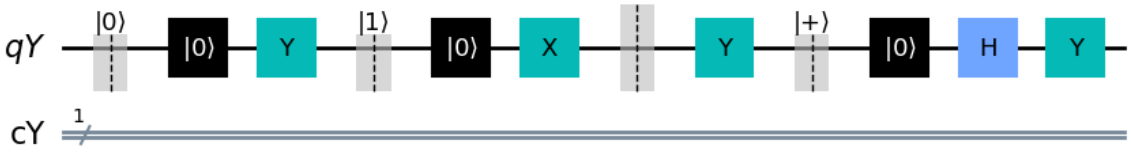
Out[40]:

In [41]:
```
circuit_X.draw(output='mpl')
```

Out[41]:

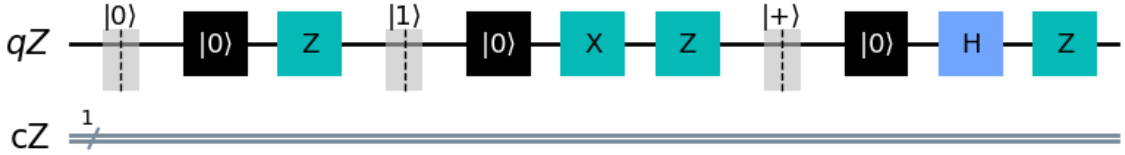
In [42]:
```
circuit_Y.draw(output='mpl')
```

Out[42]:



In [43]:
```
circuit_Z.draw(output='mpl')
```

Out[43]:



In [44]:
```
state_H_6 = Statevector.from_int(0, 2**1)
state_H_6 = state_H_6.evolve(circuit_H)
state_H_6.draw('latex')
```

Out[44]:

$$-|0\rangle$$

In [45]:
```
state_X_6 = Statevector.from_int(0, 2**1)
state_X_6 = state_X_6.evolve(circuit_X)
state_X_6.draw('latex')
```

Out[45]:

$$\frac{\sqrt{2}}{2}|0\rangle + \frac{\sqrt{2}}{2}|1\rangle$$

In [46]:
```
state_Y_6 = Statevector.from_int(0, 2**1)
state_Y_6 = state_Y_6.evolve(circuit_Y)
state_Y_6.draw('latex')
```

Out[46]:

$$-\frac{\sqrt{2}i}{2}|0\rangle + \frac{\sqrt{2}i}{2}|1\rangle$$

In [47]:
```
state_Z_6 = Statevector.from_int(0, 2**1)
state_Z_6 = state_Z_6.evolve(circuit_Z)
state_Z_6.draw('latex')
```

Out[47]:

$$-\frac{\sqrt{2}}{2}|0\rangle + \frac{\sqrt{2}}{2}|1\rangle$$

In [ ]:
```

```

Here, We prepare all qubits in State: |-⟩

In [48]:
```
circuit_H.barrier(qH, label='|-⟩')
circuit_H.reset(qH)
circuit_H.x(qH)
circuit_H.h(qH)

circuit_X.barrier(qX, label='|-⟩')
circuit_X.reset(qX)
circuit_X.x(qX)
circuit_X.h(qX)

circuit_Y.barrier(qY, label='|-⟩')
circuit_Y.reset(qY)
circuit_Y.x(qY)
circuit_Y.h(qY)

circuit_Z.barrier(qZ, label='|-⟩')
circuit_Z.reset(qZ)
circuit_Z.x(qZ)
circuit_Z.h(qZ)
```

Out[48]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x25e8d4fec10>
```

In [49]:
```
state_H_7 = Statevector.from_int(0, 2**1)
state_H_7 = state_H_7.evolve(circuit_H)
state_H_7.draw('latex')
```

Out[49]:

$$-\frac{\sqrt{2}}{2}|0\rangle + \frac{\sqrt{2}}{2}|1\rangle$$

In [50]:
```
state_X_7 = Statevector.from_int(0, 2**1)
state_X_7 = state_X_7.evolve(circuit_X)
state_X_7.draw('latex')
```

Out[50]:

$$\frac{\sqrt{2}}{2}|0\rangle - \frac{\sqrt{2}}{2}|1\rangle$$

In [51]:
```
state_Y_7 = Statevector.from_int(0, 2**1)
state_Y_7 = state_Y_7.evolve(circuit_Y)
state_Y_7.draw('latex')
```

Out[51]:

$$-\frac{\sqrt{2}i}{2}|0\rangle + \frac{\sqrt{2}i}{2}|1\rangle$$

In [52]:
```
state_Z_7 = Statevector.from_int(0, 2**1)
state_Z_7 = state_Z_7.evolve(circuit_Z)
state_Z_7.draw('latex')
```

Out[52]:

$$-\frac{\sqrt{2}}{2}|0\rangle + \frac{\sqrt{2}}{2}|1\rangle$$

In [ ]:
```

```

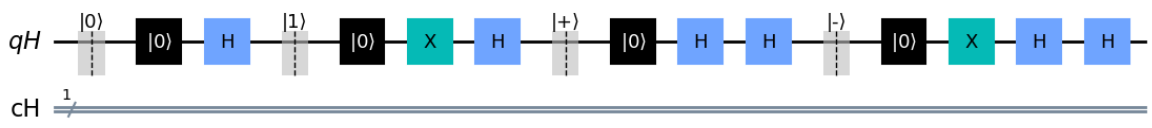
We now apply quantum logic gates on |-⟩

In [53]:
```
circuit_H.h(qH)
circuit_X.x(qX)
circuit_Y.y(qY)
circuit_Z.z(qZ)
```

Out[53]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x25e8d4f72b0>
```

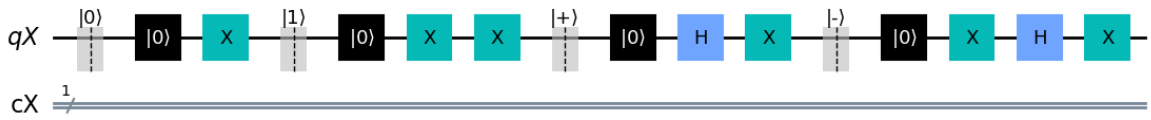
In [54]:
```
circuit_H.draw(output='mpl')
```

Out[54]:



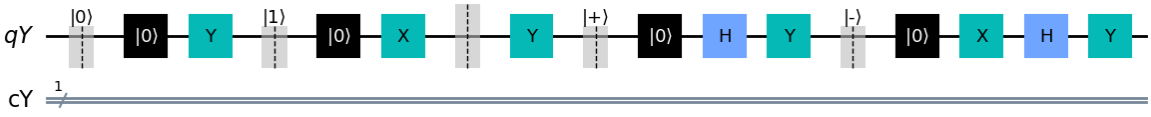
In [55]:
```
circuit_X.draw(output='mpl')
```

Out[55]:



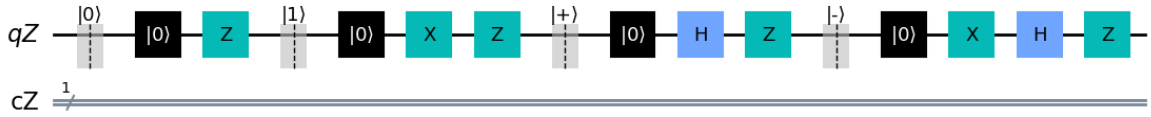
In [56]:
```
circuit_Y.draw(output='mpl')
```

Out[56]:



In [57]:
```
circuit_Z.draw(output='mpl')
```

Out[57]:



In [58]:
```
state_H_8 = Statevector.from_int(0, 2**1)
state_H_8 = state_H_8.evolve(circuit_H)
state_H_8.draw('latex')
```

Out[58]:

$$-|1\rangle$$

In [59]:
```
state_X_8 = Statevector.from_int(0, 2**1)
state_X_8 = state_X_8.evolve(circuit_X)
state_X_8.draw('latex')
```

Out[59]:

$$-\frac{\sqrt{2}}{2}|0\rangle + \frac{\sqrt{2}}{2}|1\rangle$$

In [60]:
```
state_Y_8 = Statevector.from_int(0, 2**1)
state_Y_8 = state_Y_8.evolve(circuit_Y)
state_Y_8.draw('latex')
```

Out[60]:

$$-\frac{\sqrt{2}}{2}|0\rangle - \frac{\sqrt{2}}{2}|1\rangle$$

In [61]:
```
state_Z_8 = Statevector.from_int(0, 2**1)
state_Z_8 = state_Z_8.evolve(circuit_Z)
state_Z_8.draw('latex')
```

Out[61]:

$$\frac{\sqrt{2}}{2}|0\rangle + \frac{\sqrt{2}}{2}|1\rangle$$

Conclusion

In conclusion, the use of Qiskit to create single qubit gates has given users a practical introduction to quantum computing. To produce superposition states and alter the foundation of a qubit state, respectively, the Hadamard and Pauli gates were used. These gates are key building blocks for creating more complex quantum circuits and are utilised in numerous quantum algorithms.

The usage of Qiskit enabled the construction of quantum circuits, their visualisation, and their execution on a simulator or a genuine quantum device via IBMQ. The results of the circuit simulation on the simulator were in line with what was anticipated, proving that the gates were implemented correctly.

Overall, this experiment has provided a better understanding of quantum computing and its potential applications. The use of Qiskit and IBMQ has made quantum computing more accessible and has enabled the exploration of quantum algorithms, quantum error correction techniques, and other areas of quantum research.