In [1]:
```
# !pip install qiskit qiskit_aer qiskit_ibm_runtime
# !pip install jupyter
# !pip install matplotlib pylatexenc
```

In [1]:
```
import qiskit
qiskit.__version__
```

Out[1]: '1.4.2'

In [2]:
```
import qiskit_aer
qiskit_aer.__version__
```

Out[2]: '0.15.1'

In [3]:
```
import qiskit_ibm_runtime
qiskit_ibm_runtime.__version__
```

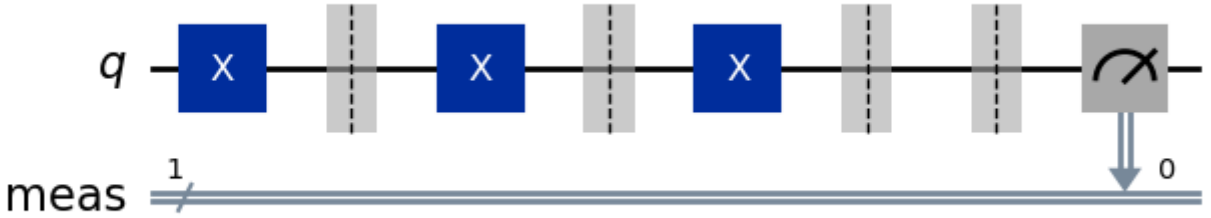
Out[3]: '0.37.0'

In [1]:

```
from qiskit.circuit import QuantumCircuit

MAX_DEPTH = 100
circuits = []
for d in range(MAX_DEPTH+1):
    circ = QuantumCircuit(1)
    for _ in range(d):
        circ.x(0)
        circ.barrier(0)
    circ.measure_all()
    circuits.append(circ)

display(circuits[3].draw(output="mpl"))
```



In [2]:

```
from qiskit.quantum_info import SparsePauliOp
obs = SparsePauliOp.from_list([("Z", 1.0)])
obs
```

Out[2]:

```
SparsePauliOp(['Z'],
              coeffs=[1.+0.j])
```

In [3]:
```
from qiskit_aer.noise.errors import coherent_unitary_error, amplitude_damping_error, ReadoutError
from qiskit.circuit.library import RXGate

# Coherent (unitary) error: Over X-rotation error
# https://qiskit.github.io/qiskit-aer/stubs/qiskit_aer.noise.coherent_unitary_error.html#qiskit_aer.nois
OVER_ROTATION_ANGLE = 0.05
coherent_error = coherent_unitary_error(RXGate(OVER_ROTATION_ANGLE).to_matrix())

# Incoherent error: Amplitude dumping error
# https://qiskit.github.io/qiskit-aer/stubs/qiskit_aer.noise.amplitude_damping_error.html#qiskit_aer.noi
AMPLITUDE_DAMPING_PARAM = 0.02  # in [0, 1] (0: no error)
incoherent_error = amplitude_damping_error(AMPLITUDE_DAMPING_PARAM)

# Readout (measurement) error: Readout error
# https://qiskit.github.io/qiskit-aer/stubs/qiskit_aer.noise.ReadoutError.html#qiskit_aer.noise.ReadoutE
PREP0_MEAS1 = 0.03  # P(1|0): Probability of preparing 0 and measuring 1
PREP1_MEAS0 = 0.08  # P(0|1): Probability of preparing 1 and measuring 0
readout_error = ReadoutError([
    [1 - PREP0_MEAS1, PREP0_MEAS1],
    [PREP1_MEAS0, 1 - PREP1_MEAS0]
])
```

In [4]:
```
from qiskit_aer.noise import NoiseModel
noise_model = NoiseModel()
noise_model.add_quantum_error(coherent_error.compose(incoherent_error), 'x', (0, ))
noise_model.add_readout_error(readout_error, (0, ))
```

In [5]:
```
from qiskit_aer.primitives import SamplerV2 as Sampler
noisy_sampler = Sampler(options={"backend_options": {"noise_model": noise_model}})
```

In [6]:
```
job = noisy_sampler.run(circuits, shots=400)
```

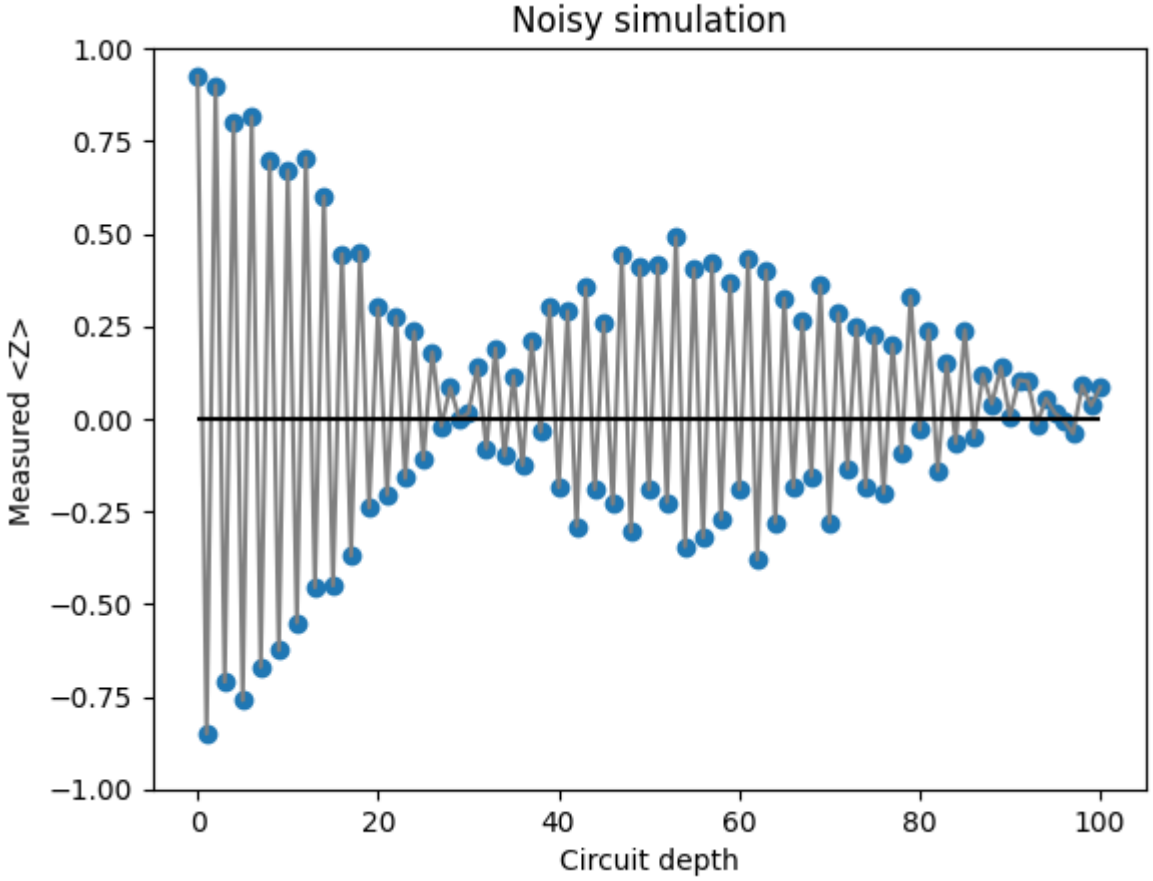
In [7]:
```
result = job.result()
```

In [8]:
```
result[0].data.meas.get_counts()
```

Out[8]: {'0': 385, '1': 15}

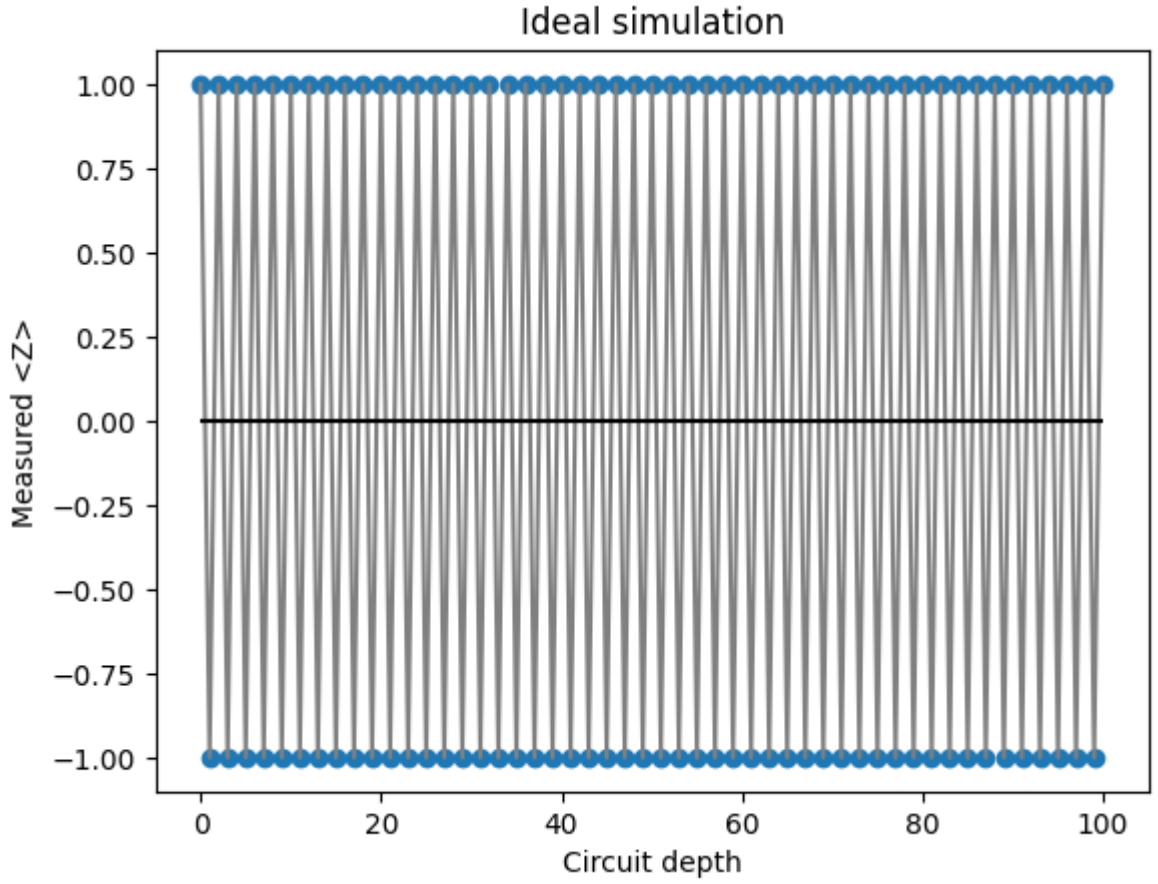
In [9]:

```
 1 import matplotlib.pyplot as plt
 2 plt.title(f"Noisy simulation")
 3 ds = list(range(MAX_DEPTH+1))
 4 plt.plot(ds, [result[d].data.meas.expectation_values(["Z"]) for d in ds], color="gray", linestyle="-",)
 5 plt.scatter(ds, [result[d].data.meas.expectation_values(["Z"]) for d in ds], marker="o")
 6 plt.hlines(0, xmin=0, xmax=MAX_DEPTH, colors="black")
 7 plt.ylim(-1, 1)
 8 plt.xlabel("Circuit depth")
 9 plt.ylabel("Measured <Z>")
10 plt.show()
```



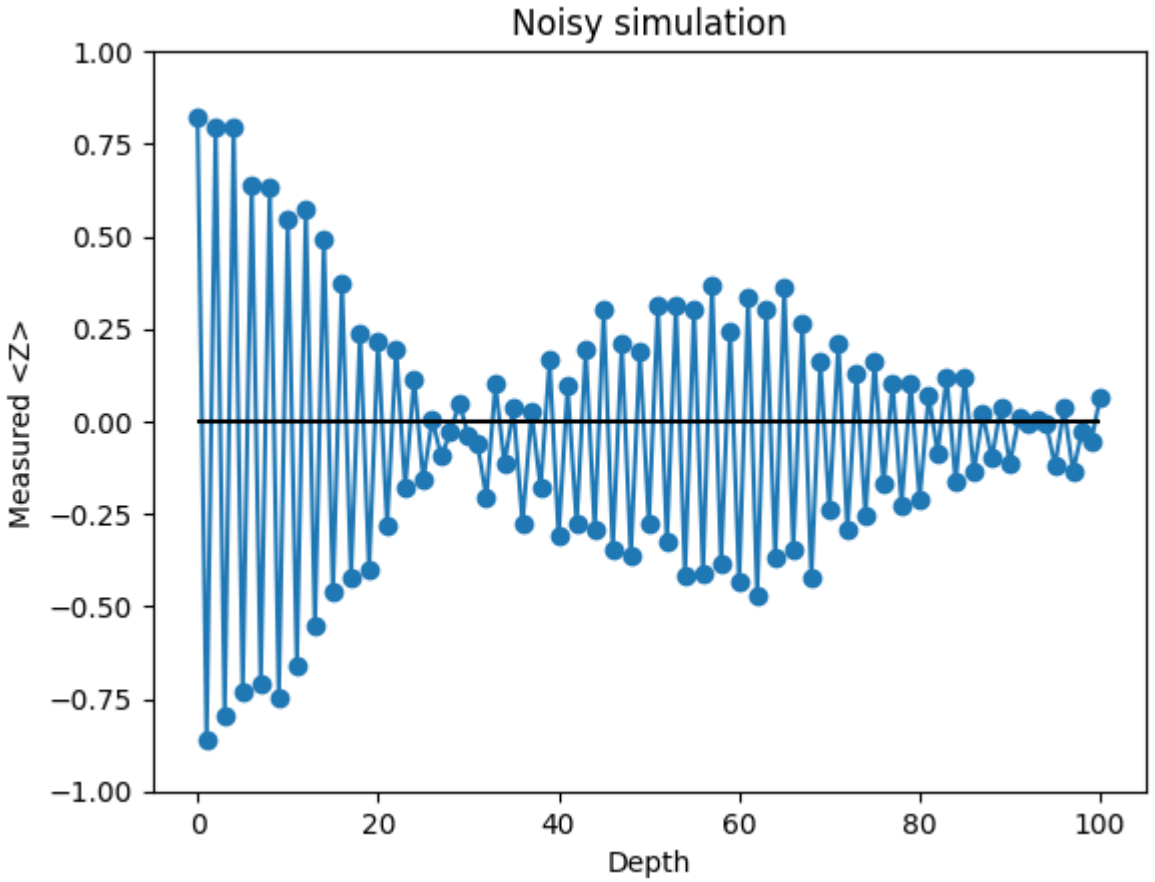
In [10]:

```
ideal_sampler = Sampler()
job_ideal = ideal_sampler.run(circuits)
result_ideal = job_ideal.result()
plt.title(f"Ideal simulation")
ds = list(range(MAX_DEPTH+1))
plt.plot(ds, [result_ideal[d].data.meas.expectation_values(["Z"]) for d in ds], color="gray", linestyle=
plt.scatter(ds, [result_ideal[d].data.meas.expectation_values(["Z"]) for d in ds], marker="o")
plt.hlines(0, xmin=0, xmax=MAX_DEPTH, colors="black")
plt.xlabel("Circuit depth")
plt.ylabel("Measured <Z>")
plt.show()
```



In [11]:

```
OVER_ROTATION_ANGLE = 0.05
coherent_error = coherent_unitary_error(RXGate(OVER_ROTATION_ANGLE).to_matrix())
AMPLITUDE_DAMPING_PARAM = 0.02  # in [0, 1] (0: no error)
incoherent_error = amplitude_damping_error(AMPLITUDE_DAMPING_PARAM)
PREP0_MEAS1 = 0.1  # P(1|0): Probability of preparing 0 and measuring 1
PREP1_MEAS0 = 0.05  # P(0|1): Probability of preparing 1 and measuring 0
readout_error = ReadoutError([
    [1 - PREP0_MEAS1, PREP0_MEAS1],
    [PREP1_MEAS0, 1 - PREP1_MEAS0]
])
noise_model = NoiseModel()
noise_model.add_quantum_error(coherent_error.compose(incoherent_error), 'x', (0, ))
noise_model.add_readout_error(readout_error, (0, ))
options = {
    "backend_options": {"noise_model": noise_model},
}
noisy_sampler = Sampler(options=options)
job = noisy_sampler.run(circuits, shots=400)
result = job.result()
plt.title(f"Noisy simulation")
ds = list(range(MAX_DEPTH+1))
plt.plot(ds, [result[d].data.meas.expectation_values(["Z"]) for d in ds], marker="o", linestyle="-",)
plt.hlines(0, xmin=0, xmax=MAX_DEPTH, colors="black")
plt.ylim(-1, 1)
plt.xlabel("Depth")
plt.ylabel("Measured <Z>")
plt.show()
```

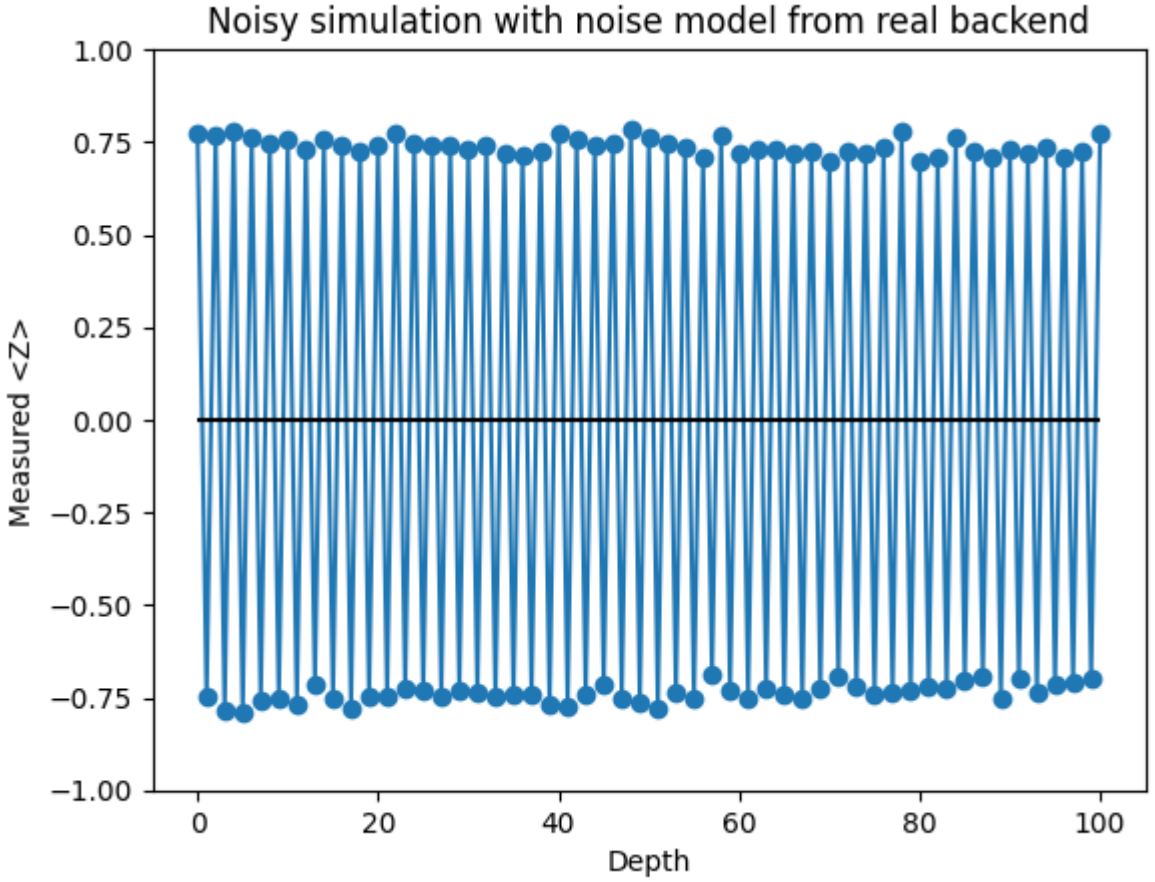


In [12]:

```
1 from qiskit_aer import AerSimulator
2 from qiskit_ibm_runtime import SamplerV2 as Sampler, QiskitRuntimeService
3 service = QiskitRuntimeService()
4 real_backend = service.backend('ibm_torino')
```

In [13]:

```
aer = AerSimulator.from_backend(real_backend)
noisy_sampler = Sampler(mode=aer)
job = noisy_sampler.run(circuits)
result = job.result()
plt.title(f"Noisy simulation with noise model from real backend")
ds = list(range(MAX_DEPTH+1))
plt.plot(ds, [result[d].data.meas.expectation_values(["Z"]) for d in ds], marker="o", linestyle="-",)
plt.hlines(0, xmin=0, xmax=MAX_DEPTH, colors="black")
plt.ylim(-1, 1)
plt.xlabel("Depth")
plt.ylabel("Measured <Z>")
plt.show()
```



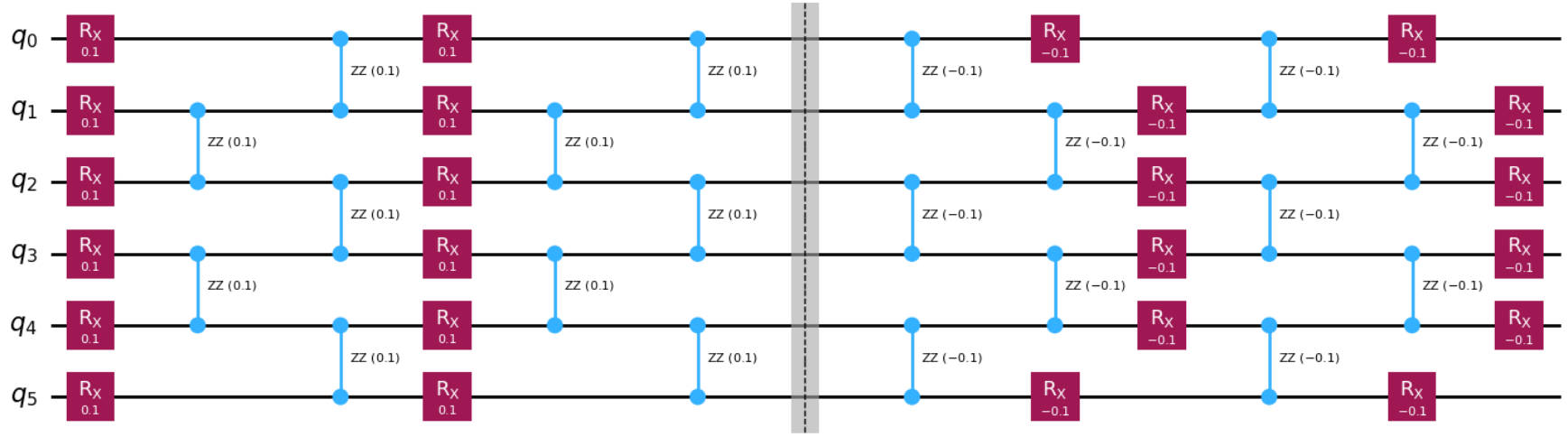
In [14]:
```
backend = service.backend('ibm_torino')
```

In [15]:
```
NUM_QUBITS = 6
NUM_TIME_STEPS = list(range(8))
RX_ANGLE = 0.1
RZZ_ANGLE = 0.1
```

In [16]:
```
# Build circuits with different number of time steps
circuits = []
for n_steps in NUM_TIME_STEPS:
    circ = QuantumCircuit(NUM_QUBITS)
    for i in range(n_steps):
        # rx layer
        for q in range(NUM_QUBITS):
            circ.rx(RX_ANGLE, q)
        # 1st rzz layer
        for q in range(1, NUM_QUBITS-1, 2):
            circ.rzz(RZZ_ANGLE, q, q+1)
        # 2nd rzz layer
        for q in range(0, NUM_QUBITS-1, 2):
            circ.rzz(RZZ_ANGLE, q, q+1)
    circ.barrier()  # need not to optimize the circuit
    # Uncompute stage
    for i in range(n_steps):
        for q in range(0, NUM_QUBITS-1, 2):
            circ.rzz(-RZZ_ANGLE, q, q+1)
        for q in range(1, NUM_QUBITS-1, 2):
            circ.rzz(-RZZ_ANGLE, q, q+1)
        for q in range(NUM_QUBITS):
            circ.rx(-RX_ANGLE, q)
    circuits.append(circ)
```

In [17]:
```
# Print the circuit with 2 time steps
circuits[2].draw(output="mpl")
```

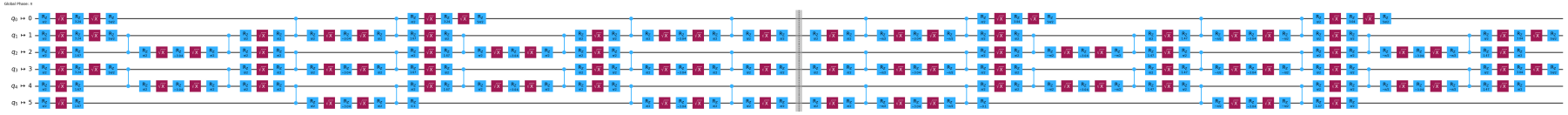
Out[17]:



In [18]:
```
obs = SparsePauliOp.from_sparse_list([("Z", [0], 1.0)], num_qubits=NUM_QUBITS)
obs
```

Out[18]:
```
SparsePauliOp(['IIIIIZ'],
              coeffs=[1.+0.j])
```

In [19]:
```
from qiskit.transpiler.preset_passmanagers import generate_preset_pass_manager
pm = generate_preset_pass_manager(optimization_level=1, backend=backend)
isa_circuits = pm.run(circuits)
display(isa_circuits[2].draw('mpl', idle_wires=False, fold=-1))
```



In [20]:
```
from qiskit_ibm_runtime import Batch
from qiskit_ibm_runtime import EstimatorV2 as Estimator

jobs = []
with Batch(backend=backend):
    for resilience_level in [0, 1, 2]:
        estimator = Estimator()
        estimator.options.resilience_level = resilience_level
        job = estimator.run([(circ, obs.apply_layout(circ.layout)) for circ in isa_circuits])
        print(f"Job ID (rl={resilience_level}): {job.job_id()}")
        jobs.append(job)
```

```
Job ID (rl=0): d13ho7tfetss73eon6lg
Job ID (rl=1): d13ho97anitc73cju460
Job ID (rl=2): d13hoa5fetss73eon6o0
```

In [21]:
```
# REPLACE WITH YOUR OWN JOB IDS
job_ids = ["d13ho7tfetss73eon6lg", "d13ho97anitc73cju460", "d13hoa5fetss73eon6o0"]
jobs = [service.job(job_id) for job_id in job_ids]
```

In [ ]:
```
# Get results
results = [job.result() for job in jobs]
print(results)
```

In [ ]:
```

```