3. Running on IBM Q

Author : Gwonhak Lee (gwonhak@gmail.com)

다음으로 IBM의 양자컴퓨터를 활용하는 방법에 대해 살펴보겠습니다.

0. 필요한 요소 불러오기

In [14]:
```
from qiskit import IBMQ, QuantumRegister, QuantumCircuit, ClassicalRegister, transpi
from qiskit.tools import backend_overview, backend_monitor
from qiskit.providers.ibmq.job import job_monitor
from qiskit.quantum_info import hellinger_fidelity
from qiskit.providers.ibmq import IBMQAccountCredentialsNotFound
from qiskit.providers.aer import AerProvider
from qiskit.tools.visualization import plot_gate_map, plot_histogram
```

1. QPU Backend 확인

IBMQ의 계정을 활성화합니다.

1. IBMQ 홈페이지(https://quantum-computing.ibm.com/)에 로그인합니다.
2. 좌측 상단의 계정 아이콘 -> Account Details 를 클릭합니다.
3. 좌측 중앙에 Copy Token 버튼을 클릭하여 아래 Block의 token 변수에 string 형태로 입력합니다.

In [15]:
```
token = 'f5876d2c6ea00ce4cefe8fcf6324956e042cd2a7cd57f6efc1e7ba38727964f10cef560830337
try:
    IBMQ.disable_account()
except IBMQAccountCredentialsNotFound:
    pass
IBMQ.enable_account(token)
```

Out[15]:
```
<AccountProvider for IBMQ(hub='ibm-q', group='open', project='main')>
```

활용할 수 있는 backend의 목록을 확인합니다.

In [16]:
```
backend_overview()
```

```
ibmq_manila
-----------
Num. Qubits:  5
Pending Jobs: 4
Least busy:   False
Operational:  True
Avg. T1:      163.6
Avg. T2:      58.3

ibmq_quito
----------
Num. Qubits:  5
Pending Jobs: 95
Least busy:   False
Operational:  True
Avg. T1:      89.8
Avg. T2:      109.8

ibmq_belem
----------
Num. Qubits:  5
Pending Jobs: 76
Least busy:   False
Operational:  True
Avg. T1:      88.7
Avg. T2:      108.8

ibmq_lima
---------
Num. Qubits:  5
Pending Jobs: 0
Least busy:   True

ibmq_bogota
-----------
Num. Qubits:  5
Pending Jobs: 1
Least busy:   False

ibmq_santiago
-------------
Num. Qubits:  5
Pending Jobs: 101
Least busy:   False
```

```
Operational:  True          Operational:  True          Operational:  True
Avg. T1:      95.6          Avg. T1:      88.2          Avg. T2:      94.8
Avg. T2:      104.4         Avg. T2:      113.4         Avg. T2:      113.0
```

```
ibmq_armonk
-----------
Num. Qubits:  1
Pending Jobs: 385
Least busy:   False
Operational:  True
Avg. T1:      244.9
Avg. T2:      255.7
```

특정한 qpu backend의 자세한 사항을 확인할 수 있습니다.

아래 예시에서는 `ibmq_manila` 의 특성을 확인합니다.

In [17]:
```
IBMQ_provider = IBMQ.get_provider()
ibmq_manila_backend = IBMQ_provider.get_backend('ibmq_manila')
backend_monitor(ibmq_manila_backend)
```

```
ibmq_manila
===========
Configuration
-------------
    n_qubits: 5
    operational: True
    status_msg: active
    pending_jobs: 4
    backend_version: 1.0.5
    basis_gates: ['id', 'rz', 'sx', 'x', 'cx', 'reset']
    local: False
    simulator: False
    quantum_volume: 32
    credits_required: True
    sample_name: family: Falcon, revision: 5.11, segment: L
    input_allowed: ['job']
    pulse_num_channels: 9
    memory: True
    meas_levels: [1, 2]
    qubit_lo_range: [[4.46277998307546, 5.46277998307546], [4.338419511638877, 5.33841
9511638877], [4.536938824727899, 5.536938824727899], [4.45130087049468, 5.451300870494
68], [4.566354311434293, 5.566354311434293]]
    u_channel_lo: [[{'q': 1, 'scale': (1+0j)}], [{'q': 0, 'scale': (1+0j)}], [{'q': 2,
'scale': (1+0j)}], [{'q': 1, 'scale': (1+0j)}], [{'q': 3, 'scale': (1+0j)}], [{'q': 2,
'scale': (1+0j)}], [{'q': 4, 'scale': (1+0j)}], [{'q': 3, 'scale': (1+0j)}]]
    description: 5 qubit device
    conditional: False
    url: None
    default_rep_delay: 250.0
    processor_type: {'family': 'Falcon', 'revision': '5.11', 'segment': 'L'}
    max_shots: 8192
    dt: 0.2222222222222222
    acquisition_latency: []
    rep_delay_range: [0.0, 500.0]
    supported_instructions: ['u3', 'cx', 'x', 'acquire', 'setf', 'sx', 'shiftf', 'rese
t', 'u1', 'id', 'delay', 'rz', 'measure', 'u2', 'play']
    meas_lo_range: [[6.663214088, 7.663214088], [6.7833221550000005, 7.783322155000000
5], [6.7189281020000005, 7.7189281020000005], [6.6101423420000005, 7.610142342000000
```

5], [6.846997692, 7.846997692]]
coupling_map: [[0, 1], [1, 0], [1, 2], [2, 1], [2, 3], [3, 2], [3, 4], [4, 3]]
hamiltonian: {'description': 'Qubits are modeled as Duffing oscillators. In this case, the system includes higher energy states, i.e. not just |0> and |1>. The Pauli operators are generalized via the following set of transformations:₩n₩n$(₩₩mathbb{I}-₩₩sigma_{i}^z)/2 ₩₩rightarrow 0_i ₩₩equiv b^₩₩dagger_{i} b_{i}$,₩n₩n$₩₩sigma_{+} ₩₩rightarrow b^₩₩dagger$,₩n₩n$₩₩sigma_{-} ₩₩rightarrow b$,₩n₩n$₩₩sigma_{i}^X ₩₩rightarrow b^₩₩dagger_{i} + b_{i}$.₩n₩nQubits are coupled through resonator buses. The provided Hamiltonian has been projected into the zero excitation subspace of the resonator buses leading to an effective qubit-qubit flip-flop interaction. The qubit resonance frequencies in the Hamiltonian are the cavity dressed frequencies and not exactly what is returned by the backend defaults, which also includes the dressing due to the qubit-qubit interactions.₩n₩nQuantities are returned in angular frequencies, with units 2*pi*GHz.₩n₩nWARNING: Currently not all system Hamiltonian information is available to the public, missing values have been replaced with 0.₩n', 'h_latex': '₩₩begin{align} ₩₩mathcal{H}/₩₩hbar = & ₩₩sum_{i=0}^{4}₩₩left(₩₩frac{₩₩omega_{q,i}}{2}(₩₩mathbb{I}-₩₩sigma_i^{z})+₩₩frac{₩₩Delta_{i}}{2}(0_i^2-0_i)+₩₩Omega_{d,i}D_i(t)₩₩sigma_i^{X}₩₩right) ₩₩₩₩₩ & + J_{0,1}(₩₩sigma_{0}^{+}₩₩sigma_{1}^{-}+₩₩sigma_{0}^{-}₩₩sigma_{1}^{+}) + J_{1,2}(₩₩sigma_{1}^{+}₩₩sigma_{2}^{-}+₩₩sigma_{1}^{-}₩₩sigma_{2}^{+}) + J_{2,3}(₩₩sigma_{2}^{+}₩₩sigma_{3}^{-}+₩₩sigma_{2}^{-}₩₩sigma_{3}^{+}) + J_{3,4}(₩₩sigma_{3}^{+}₩₩sigma_{4}^{-}+₩₩sigma_{3}^{-}₩₩sigma_{4}^{+}) ₩₩₩₩₩ & + ₩₩Omega_{d,0}(U_{0}^{(0,1)}(t))₩₩sigma_{0}^{X} + ₩₩Omega_{d,1}(U_{1}^{(1,0)}(t)+U_{2}^{(1,2)}(t))₩₩sigma_{1}^{X} ₩₩₩₩₩ & + ₩₩Omega_{d,2}(U_{3}^{(2,1)}(t)+U_{4}^{(2,3)}(t))₩₩sigma_{2}^{X} + ₩₩Omega_{d,3}(U_{6}^{(3,4)}(t)+U_{5}^{(3,2)}(t))₩₩sigma_{3}^{X} ₩₩₩₩₩ & + ₩₩Omega_{d,4}(U_{7}^{(4,3)}(t))₩₩sigma_{4}^{X} ₩₩₩₩₩ ₩₩end{align}', 'h_str': ['_SUM[i,0,4,wq{i}/2*(I{i}-Z{i})]', '_SUM[i,0,4,delta{i}/2*0{i}*0{i}]', '_SUM[i,0,4,-delta{i}/2*0{i}]', '_SUM[i,0,4,omegad{i}*X{i}||D{i}]', 'jq0q1*Sp0*Sm1', 'jq0q1*Sm0*Sp1', 'jq1q2*Sp1*Sm2', 'jq1q2*Sm1*Sp2', 'jq2q3*Sp2*Sm3', 'jq2q3*Sm2*Sp3', 'jq3q4*Sp3*Sm4', 'jq3q4*Sm3*Sp4', 'omegad1*X0||U0', 'omegad0*X1||U1', 'omegad2*X1||U2', 'omegad1*X2||U3', 'omegad3*X2||U4', 'omegad4*X3||U6', 'omegad2*X3||U5', 'omegad3*X4||U7'], 'osc': {}, 'qub': {'0': 3, '1': 3, '2': 3, '3': 3, '4': 3}, 'vars': {'delta0': -2.1573187977651487, 'delta1': -2.1753119475601674, 'delta2': -2.159281266514359, 'delta3': -2.158603148482815, 'delta4': -2.1495256907311115, 'jq0q1': 0.011845444218797994, 'jq1q2': 0.01196783968906386, 'jq2q3': 0.012402113956012375, 'jq3q4': 0.01218691037040823, 'omegad0': 0.958459680297847, 'omegad1': 0.9867433019584951, 'omegad2': 0.9541014919519683, 'omegad3': 0.9720864126008828, 'omegad4': 0.9836397693823058, 'wq0': 31.18206627242469, 'wq1': 30.40068638550042, 'wq2': 31.648020016692 75, 'wq3': 31.10994088091767, 'wq4': 31.832842970569903}}
online_date: 2021-04-28 04:00:00+00:00
allow_object_storage: True
open_pulse: False
rep_times: [1000.0]
meas_map: [[0, 1, 2, 3, 4]]
backend_name: ibmq_manila
multi_meas_enabled: True
pulse_num_qubits: 3
dynamic_reprate_enabled: True
channels: {'acquire0': {'operates': {'qubits': [0]}, 'purpose': 'acquire', 'type': 'acquire'}, 'acquire1': {'operates': {'qubits': [1]}, 'purpose': 'acquire', 'type': 'acquire'}, 'acquire2': {'operates': {'qubits': [2]}, 'purpose': 'acquire', 'type': 'acquire'}, 'acquire3': {'operates': {'qubits': [3]}, 'purpose': 'acquire', 'type': 'acquire'}, 'acquire4': {'operates': {'qubits': [4]}, 'purpose': 'acquire', 'type': 'acquire'}, 'd0': {'operates': {'qubits': [0]}, 'purpose': 'drive', 'type': 'drive'}, 'd1': {'operates': {'qubits': [1]}, 'purpose': 'drive', 'type': 'drive'}, 'd2': {'operates': {'qubits': [2]}, 'purpose': 'drive', 'type': 'drive'}, 'd3': {'operates': {'qubits': [3]}, 'purpose': 'drive', 'type': 'drive'}, 'd4': {'operates': {'qubits': [4]}, 'purpose': 'drive', 'type': 'drive'}, 'm0': {'operates': {'qubits': [0]}, 'purpose': 'measure', 'type': 'measure'}, 'm1': {'operates': {'qubits': [1]}, 'purpose': 'measure', 'type': 'measure'}, 'm2': {'operates': {'qubits': [2]}, 'purpose': 'measure', 'type': 'measure'}, 'm3': {'operates': {'qubits': [3]}, 'purpose': 'measure', 'type': 'measure'}, 'm4': {'operates': {'qubits': [4]}, 'purpose': 'measure', 'type': 'measure'}, 'u0': {'operates': {'qubits': [0, 1]}, 'purpose': 'cross-resonance', 'type': 'control'}, 'u1': {'operates': {'qubits': [1, 0]}, 'purpose': 'cross-resonance', 'type': 'control'}, 'u2': {'operates': {'qubits': [1, 2]}, 'purpose': 'cross-resonance', 'type': 'control'}, 'u3': {'operates': {'qubits': [2, 1]}, 'purpose': 'cross-resonance', 'type': 'control'}, 'u4': {'operates': {'qubits': [2, 3]}, 'purpose': 'cross-resonance', 'type':

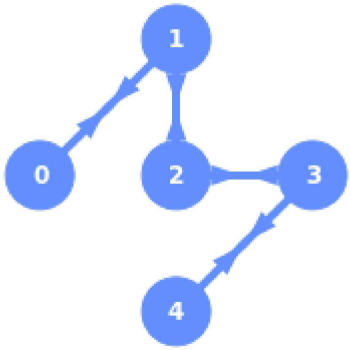
```
'control'}, 'u5': {'operates': {'qubits': [3, 2]}, 'purpose': 'cross-resonance', 'typ
e': 'control'}, 'u6': {'operates': {'qubits': [3, 4]}, 'purpose': 'cross-resonance',
'type': 'control'}, 'u7': {'operates': {'qubits': [4, 3]}, 'purpose': 'cross-resonanc
e', 'type': 'control'}}
    allow_q_object: True
    meas_kernels: ['hw_qmfk']
    dtm: 0.2222222222222222
    discriminators: ['hw_qmfk', 'quadratic_discriminator', 'linear_discriminator']
    qubit_channel_mapping: [['u0', 'd0', 'u1', 'm0'], ['u3', 'u0', 'u1', 'm1', 'd1',
'u2'], ['d2', 'm2', 'u2', 'u4', 'u5', 'u3'], ['m3', 'u7', 'u6', 'u4', 'u5', 'd3'], ['u
6', 'd4', 'u7', 'm4']]
    n_registers: 1
    measure_esp_enabled: False
    n_uchannels: 8
    conditional_latency: []
    parametric_pulses: ['gaussian', 'gaussian_square', 'drag', 'constant']
    uchannels_enabled: True
    max_experiments: 75

Qubits [Name / Freq / T1 / T2 / RZ err / SX err / X err / Readout err]
----------------------------------------------------------------------
    Q0 / 4.96278 GHz / 129.01354 us / 101.74870 us / 0.00000 / 0.00022 / 0.00022 / 0.0
2510
    Q1 / 4.83842 GHz / 212.80826 us / 48.67210 us / 0.00000 / 0.00025 / 0.00025 / 0.03
290
    Q2 / 5.03694 GHz / 164.49714 us / 24.07609 us / 0.00000 / 0.00022 / 0.00022 / 0.02
140
    Q3 / 4.95130 GHz / 157.12573 us / 73.03125 us / 0.00000 / 0.00017 / 0.00017 / 0.01
800
    Q4 / 5.06635 GHz / 154.44051 us / 44.02548 us / 0.00000 / 0.00070 / 0.00070 / 0.04
280

Multi-Qubit Gates [Name / Type / Gate Error]
--------------------------------------------
    cx4_3 / cx / 0.00679
    cx3_4 / cx / 0.00679
    cx2_3 / cx / 0.00619
    cx3_2 / cx / 0.00619
    cx1_2 / cx / 0.00918
    cx2_1 / cx / 0.00918
    cx0_1 / cx / 0.00739
    cx1_0 / cx / 0.00739
```

IBM의 Superconducting QPU는 CNOT 게이트를 적용할 수 있는 qubit pair (Connectivity)가 제한되어 있습니다. `ibmq_manila` 는 다음과 같은 linear connectivity를 가지고 있습니다.

In [18]:
```
plot_gate_map(ibmq_manila_backend, plot_directed=True)
```



2. QPU Backend 사용하기 및 회로 최적화

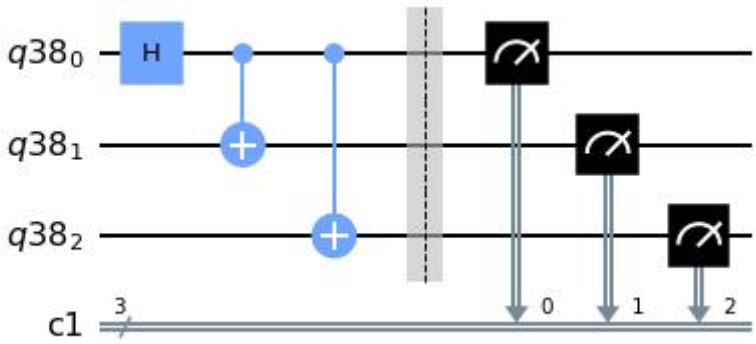
3 qubit ghz상태를 준비하여 측정하는 회로를 구성하여 qpu backend를 사용해보겠습니다.

$$|GHZ_3\rangle = \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$$

먼저 회로를 구현합니다.

In [19]:
```
qr = QuantumRegister(3)
cr = ClassicalRegister(3)
ghz3 = QuantumCircuit(qr, cr)
ghz3.h(qr[0])
ghz3.cx(qr[0], qr[1])
ghz3.cx(qr[0], qr[2])
ghz3.barrier()
ghz3.measure(qr, cr)

ghz3.draw('mpl')
```



주어진 회로를 ibmq_manila_backend 를 통해 실행합니다.

In [20]:
```
job_exp = ibmq_manila_backend.run(ghz3, shots=2048)
job_monitor(job_exp) # An error must be raised here.
```

```
Job Status: ERROR - The Qobj uses gates (['h']) that are not among the basis gates
(['id', 'rz', 'sx', 'x', 'cx', 'reset']). Error code: 1106.
```

여기서, 에러 메세지가 출력될 것입니다. 그 이유는 manila qpu에서 Hadamard gate를 구현할 수

없기 때문입니다. qpu에서 물리적으로 구현 가능한 basis gate는 다음과 같이 확인할 수 있습니다.

In [21]:
```
print(ibmq_manila_backend.configuration().basis_gates)
```

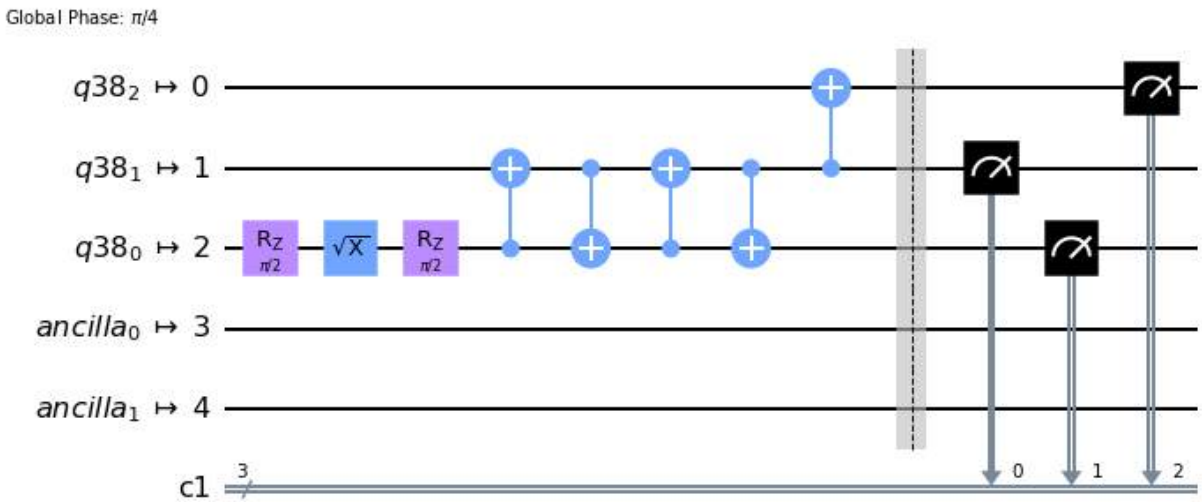
```
['id', 'rz', 'sx', 'x', 'cx', 'reset']
```

주어진 회로를 manila backend의 특성에 맞도록 변환하기 위해 transpile 과정을 거칩니다. hadamard gate가 manila backend에서 제공하는 basis gate들로 변환되었음을 확인할 수 있습니다. 또한 connectivity에 맞게 SWAP operation이 적용되었음을 확인할 수 있습니다.

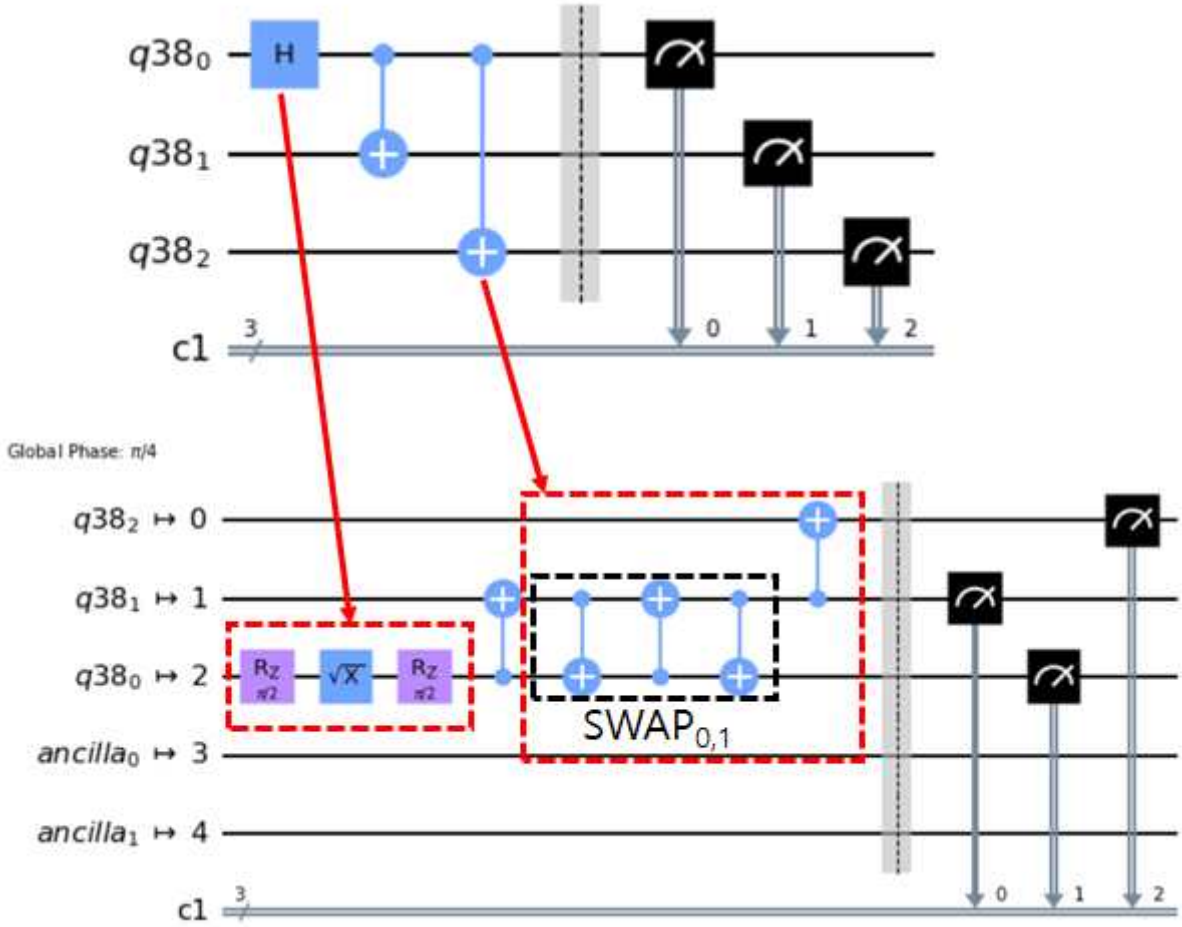
transpile 과정에서 기본적으로 제공되는 과정은 다음과 같습니다.

- Physical Qubit Layout (Connectivity)에 맞게 virtual to physical Qubit mapping
- 회로의 모든 gate들을 Basis gate set으로 구현
- 측정 gate 전에 barrier 삽입
- Qubit mapping에 맞도록 SWAP gate 생성
- SWAP gate를 3개의 CNOT gate로 분해
- CNOT-CNOT 상쇄
- Single qubit gate 최적화

In [22]:
```
transpiled_circuit = transpile(ghz3, backend=ibmq_manila_backend)
transpiled_circuit.draw('mpl')
```

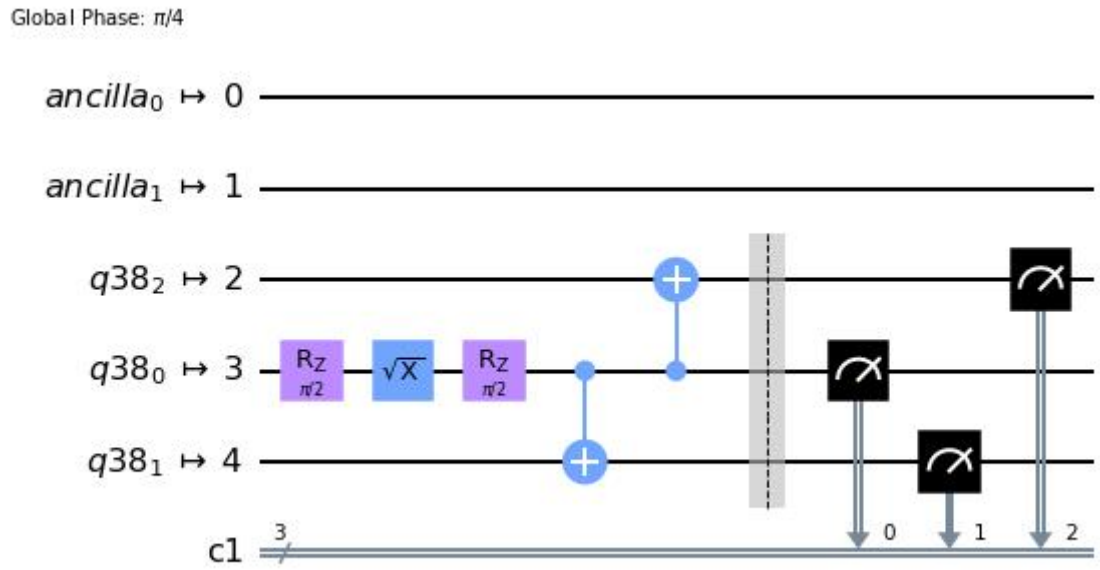


기본 transpile 과정을 거친 회로를 비교하면 다음과 같이 H gate가 basis gate로 decompose 되었음을 확인할 수 있고, qubit connectivity에 맞게 qubit에 SWAP 연산이 적용됨을 확인할 수 있습니다.



변환된 회로에서 여기서 0번째 qubit을 중앙으로 옮긴다면 불필요한 SWAP gate를 줄일 수 있음을 확인할 수 있습니다. 이는 `transpile` 에 `optimization_level` 옵션을 추가하여 자동으로 수행할 수 있습니다.

In [23]:
```
transpiled_circuit_opt = transpile(ghz3, backend=ibmq_manila_backend, optimization_le
transpiled_circuit_opt.draw('mpl')
```



`optimization_level` 은 0 ~ 3의 정수를 입력할 수 있으며, 각 단계마다 다음의 회로 최적화 기능이 추가됩니다.

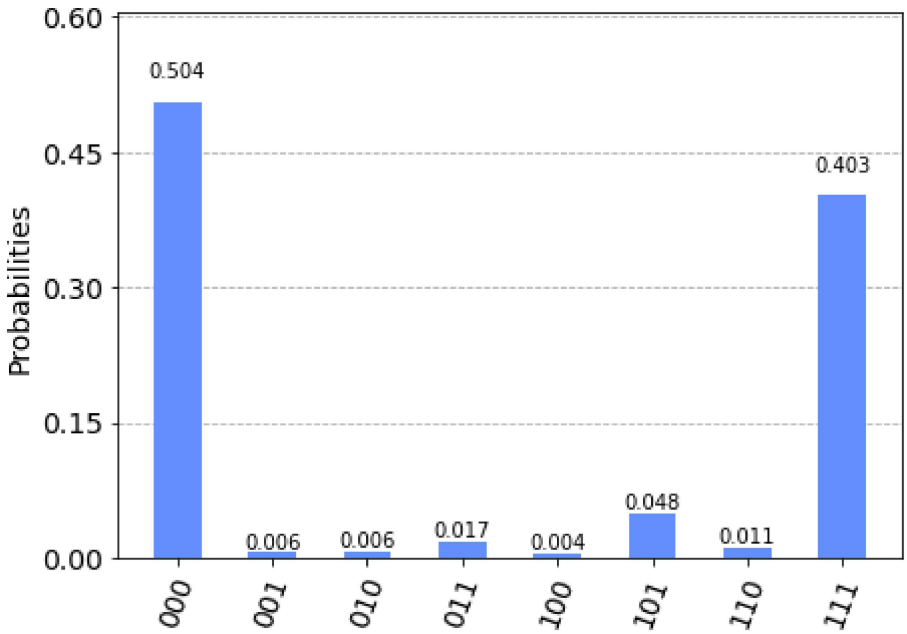
- 0단계 : 최적화 수행하지 않음
- 1단계(기본) : 이웃한 게이트만 소거
- 2단계 : Noise-adaptive qubit mapping, Gate cancellation using commutativity
- 3단계 : Heavier noise-adaptive qubit mapping, Gate cancellation using commutativity and unitary synthesis.

이제 주어진 회로를 QPU에서 실행하고 결과를 확인합니다.

In [24]:
```
job_exp = ibmq_manila_backend.run(transpiled_circuit_opt, shots=2048)
job_monitor(job_exp)
```

```
Job Status: job has successfully run
```

In [25]:
```
noisy_counts = job_exp.result().get_counts()
plot_histogram(noisy_counts)
```



Ideal simulation을 수행하고, fidelity를 확인합니다.

In [26]:
```
qasm_backend = AerProvider().get_backend('qasm_simulator')
job_ideal = qasm_backend.run(ghz3, shots=2048)
ideal_counts = job_ideal.result().get_counts()

print(hellinger_fidelity(noisy_counts, ideal_counts))
```

```
0.9023008480548862
```

3. Fake Backend 사용하기

Fake backend를 활용하면 IBM QPU의 특성 및 노이즈 모델 등에 대한 정보를 불러와 시뮬레이션을 수행 할 수 있습니다. ibmq_rome QPU에 해당하는 fake backend를 불러옵니다.

In [30]:
```
from qiskit.test.mock import FakeRome

fake_rome = FakeRome()
```

ghz3 회로를 fake_rome에서 실행합니다. 이상적인 결과에 비해 노이즈가 있음을 확인할 수 있습

니다.

In [28]:
```
rome_transpiled = transpile(ghz3, backend=fake_rome, optimization_level=3)
job_fake_rome = fake_rome.run(rome_transpiled, shots=2048)
counts_fake_rome = job_fake_rome.result().get_counts()
plot_histogram(counts_fake_rome)
```