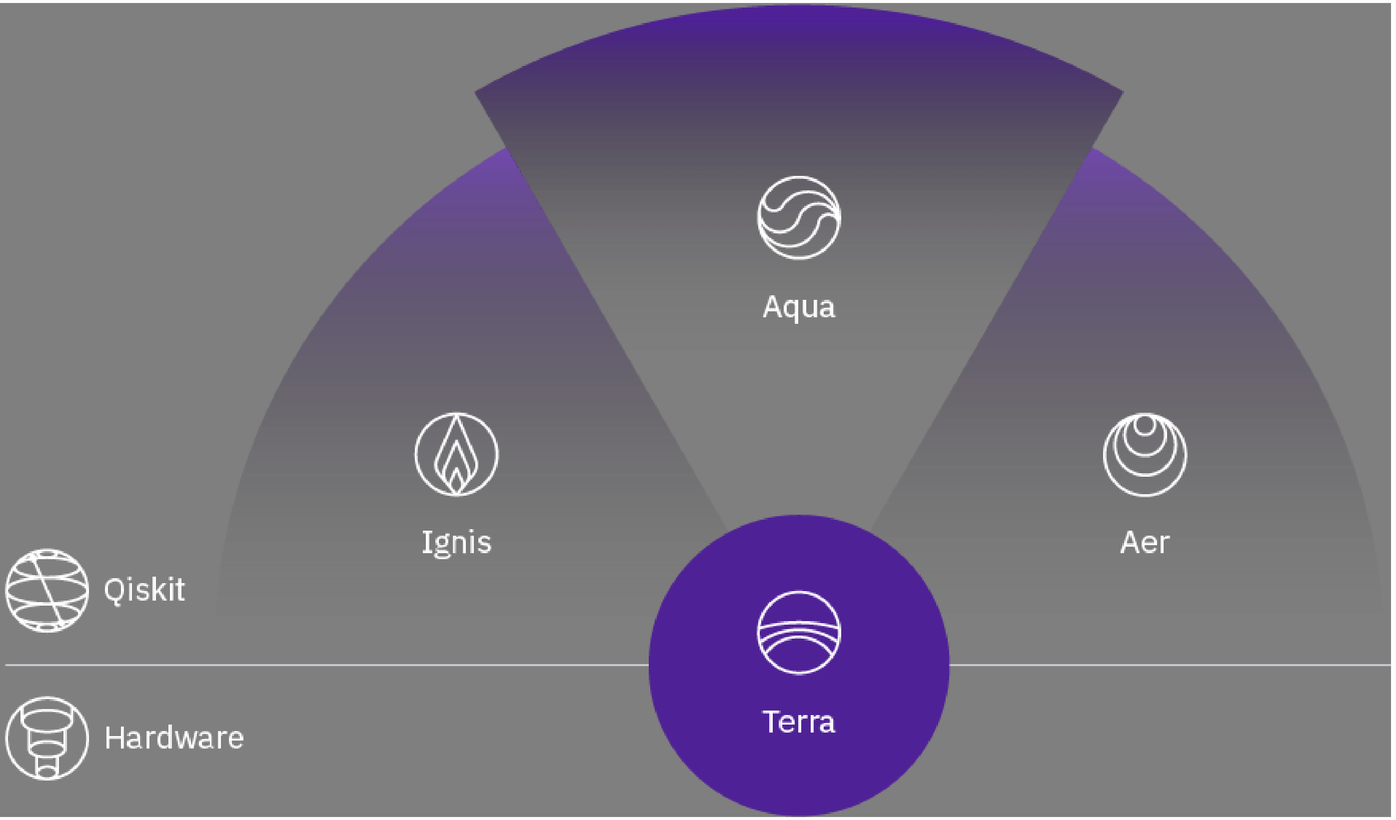
Intro to Qiskit

ECE 592 / CSC 591 - Fall 2019

Qiskit = IBM QC Dev Platform



- **Terra**: Composing programs using circuits and pulses
- **Aqua**: Building algorithms and applications
- **Aer:** Simulators, emulators, and debuggers
- **Ignis:** Addressing errors and noise

Qiskit Terra

- Build
  - Create circuit out of registers, gates
- Compile
  - Translate to QASM, then to backend instructions
- Execute
  - Backends = simulators (Aer), hardware

Building a Circuit

QuantumRegister

- Collection of qubits
- Indexed to reference individual qubit: q[0]

ClassicalRegister

- Collection of bits
- Used as the receiver of measurements on qubits

QuantumCircuit

Starts with set of registers

Add gates specifying registers/qubits as arguments

```python
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit

qreg = QuantumRegister(3)  # a 3-qubit register
creg = ClassicalRegister(3)  # a 3-bit classical register
qc = QuantumCircuit(qreg,creg)   # create a circuit

qc.measure(qreg,creg)  # measure all qubits in qr, put results in cr
```

Basic Gates

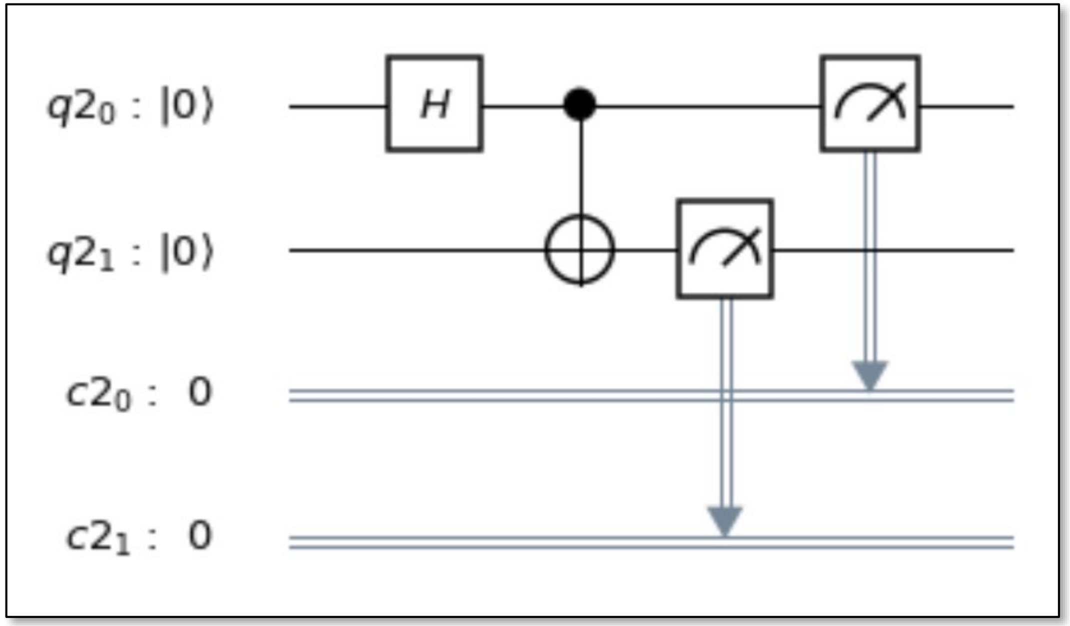
Quantum Gate	...on qubits	...on register
X (NOT)	`qc.x(qreg[0])`	`qc.x(qreg)`
Hadamard	`qc.h(qreg[0])`	`qc.h(qreg)`
CNOT	`qc.cx(qreg[0],qreq[1])`	--
Toffoli	`qc.ccx(qreg[0], qreg[1], qreg[2])`	--
Phase shift	`qc.u1(angle,qreq[0])`	`qc.u1(angle,qreg)`
Swap	`qc.swap(qreg[0],qreg[1])`	--
Measure (not a gate)	`qc.measure(qreg[0],creg[0])`	`qc.measure(qreg)`
Reset (not a gate)	`qc.reset(qreg[0])`	`qc.reset(qreg)`

summary_of_quantum_operations.ipynb

Other Circuit Operations

Operation	Description
`qc.barrier()`	Completes operations before proceeding. Can specify registers, qubits.
`qc.add(`*regs*`)`	Add register(s) to circuit.
`qc.combine(`*circuit*`)`	Appends circuit (if compatible). Creates new circuit (qc + circuit) and returns it.
`qc.extend(`*circuit*`)`	Appends circuit (if compatible). Modifies qc.
`qc.qasm()`	Returns a string containing the QASM representation of circuit.

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit

q = QuantumRegister(2)
c = ClassicalRegister(2)
qc = QuantumCircuit(q, c)

qc.h(q[0])         # Hadamard on first qubit
qc.cx(q[0],q[1])  # CNOT to entangle
# creates a Bell state

qc.measure(q,c)
```



Compiling and Running

- Provider
  - Facilitates access to a selection of backends
  - Aer Provider
    - simulators, running locally on your machine
  - IBM Q Provider
    - hardware, remote simulator
- Backend
  - Runs a compiled program (Qobj) and reports result
- Job
  - The result of an execution
  - Asynchronous -- query job to see status
  - Get result when complete

Backends

- To compile/execute a circuit, must specify a backend.
- Simulators:
  - Local (Aer):
    qasm_simulator -- emulates a machine with/without noise, multi-shot
    statevector_simulator -- single shot, returns state vector
    unitary_simulator -- returns unitary matrix represented by circuit
  - IBMQ: ibm_qasm_simulator
- Hardware:
  - IBMQ provider -- to be discussed later

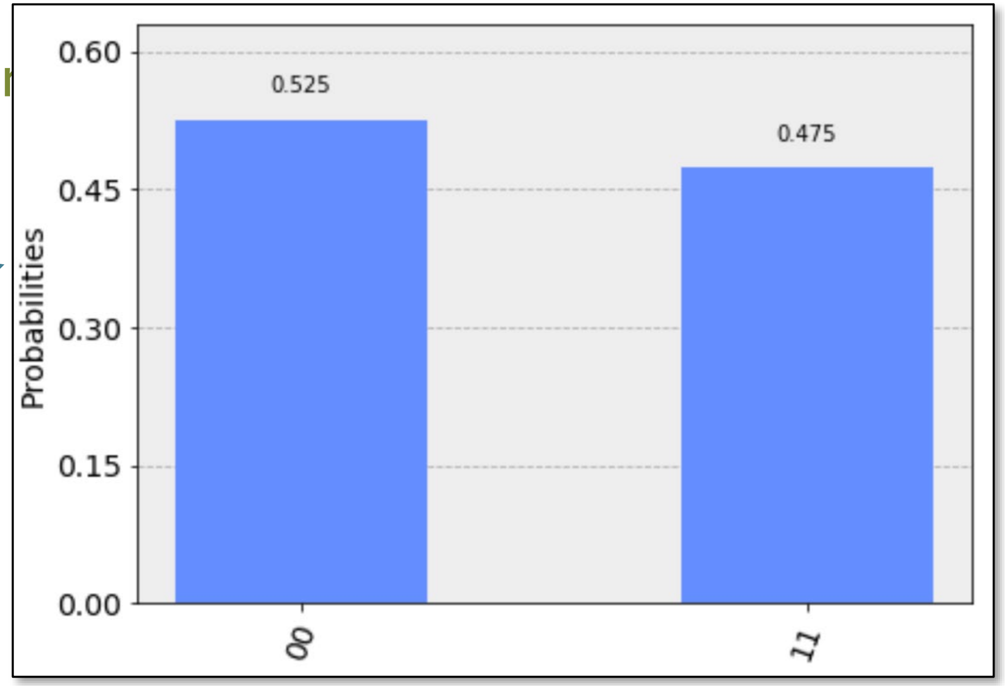
Job Operations

Operation	Description
`job.status()`	Returns current status.
`job.done()`	Returns True if done, False if not.
`job.id()`	Identifier (remote provider only)
`job.result()`	Results from completed job.
`job.result().get_counts()`	Instances of various measured states, e.g. `{'111': 512, '000': 512}`
`job_monitor(job)`	Loop that waits for job to complete, periodically printing the job status.

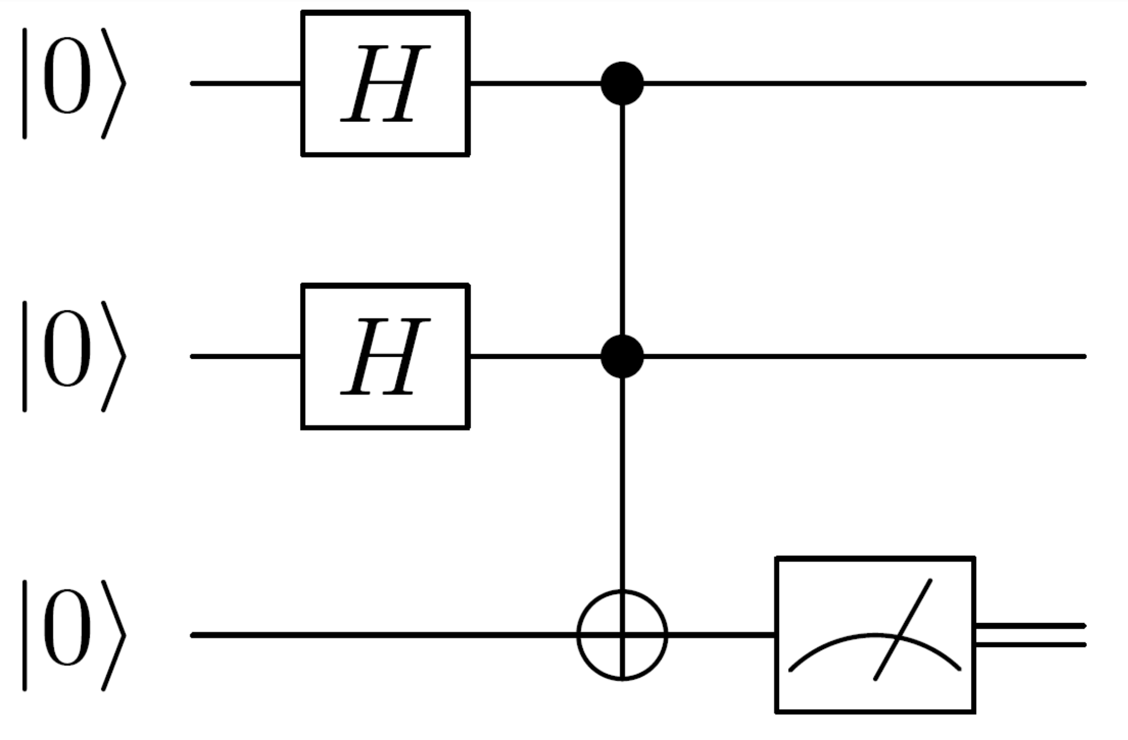
```python
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from qiskit import Aer, execute
from qiskit.tools.visualization import plot_histogram

# ... deleted circuit building commands...
qc.measure(q,c)

backend = Aer.get_backend('qasm_simulator')
job = execute(qc, backend, shots=512)  # shots default = 1024
result = job.result()
print(result.get_counts())
plot_histogram(result.get_counts())
```

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from qiskit import Aer, execute
from qiskit.tools.visualization import plot_histogram

# ... deleted circuit building commands...
qc.measure(q,c)

backend = Aer.get_backend('qasm_simulator
job = execute(qc, backend, shots=512)  #
result = job.result()
print(result.get_counts())
plot_histogram(result.get_counts())
```

{'00': 269, '11': 243}



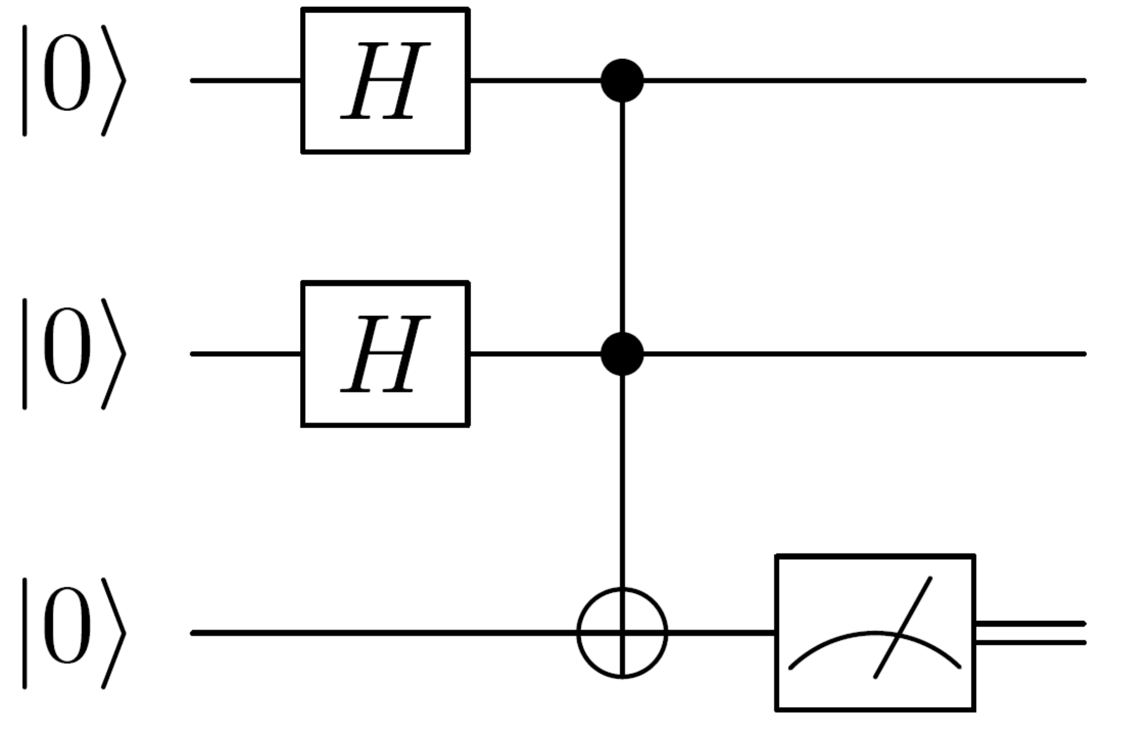
Example 2



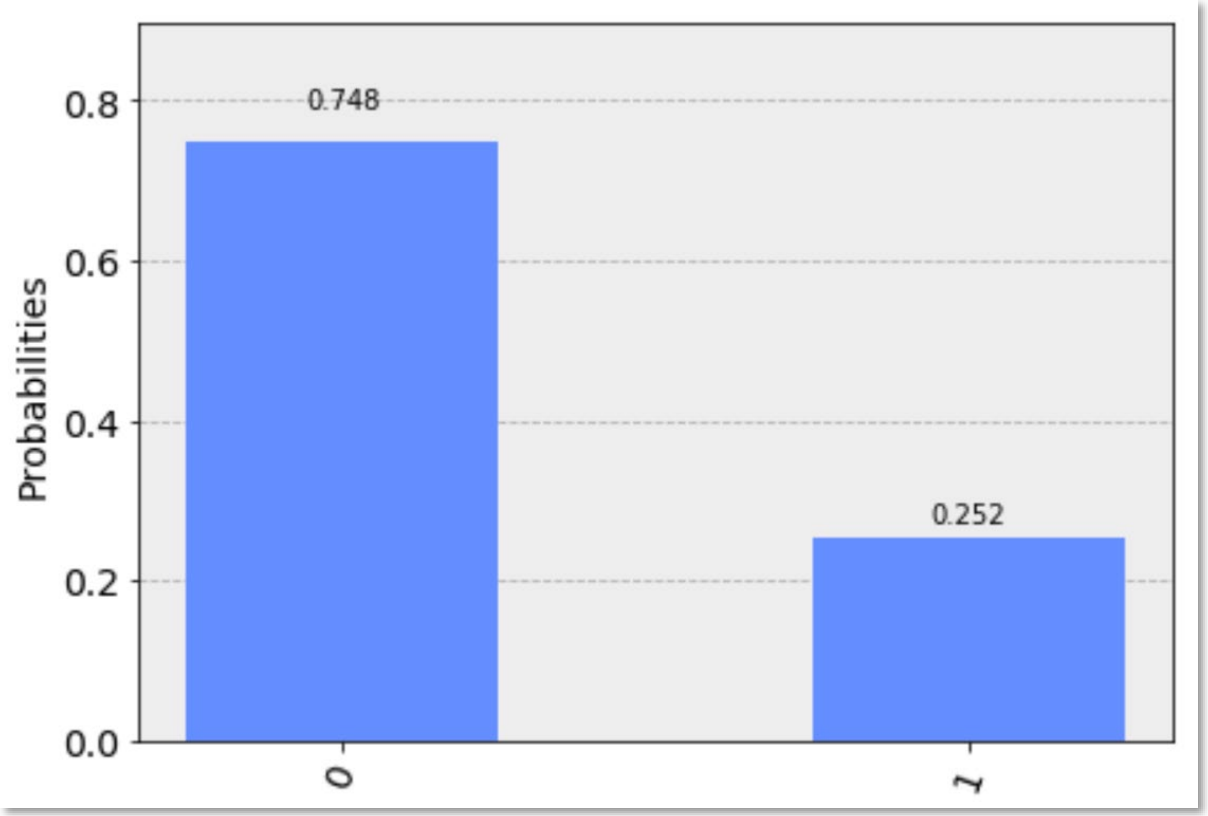
What is the result of this measurement?

After the Toffoli gate, are the qubits entangled?

Example 2



What is the result of this measurement?



Example 3

Implement a quantum circuit that checks whether two qubits are equal (in the computational basis).

Use qiskit to demonstrate that your circuit works.

Qiskit Summary

- Create quantum and classical registers.
- Create quantum circuit, adding registers.
- Add gates and measurements to circuits.

- Choose backend from provider.
- Execute circuit -- compiles circuit to match specifics of backend.
- Get results from job.