# Lab_3

November 8, 2024

## 1 Introduction to Quantum Information and Quantum ML

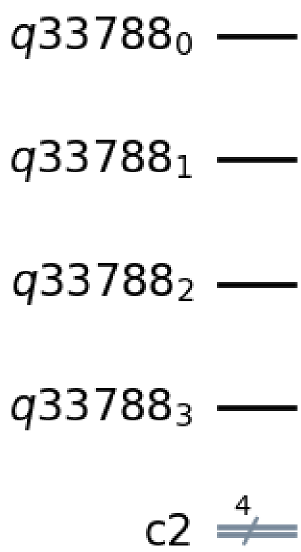
Instructor: Dr Sci. Eng. Przemysław Głowacki

Kacper Dobek 148247

[46]:
```
import math
from numpy import pi
from qiskit import *
from qiskit.tools.jupyter import *
from qiskit.visualization import *
import pandas as pd
```

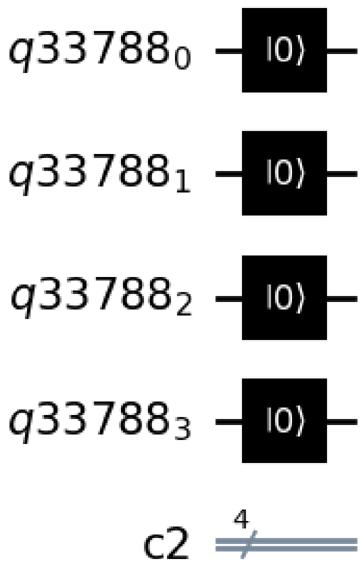
Creating quantum, classical registers and a quantum circuit

[47]:
```
n0 = 4  # Number of qubits and bits
q0 = QuantumRegister(n0)  # Quantum register
c0 = ClassicalRegister(n0)  # Classical register
circuit0 = QuantumCircuit(q0, c0)  # Quantum algorithm - quantum circuit
display(circuit0.draw(output="mpl"))  # Sketch of a quantum circuit
```




Initializing the initial states of individual quantum registers

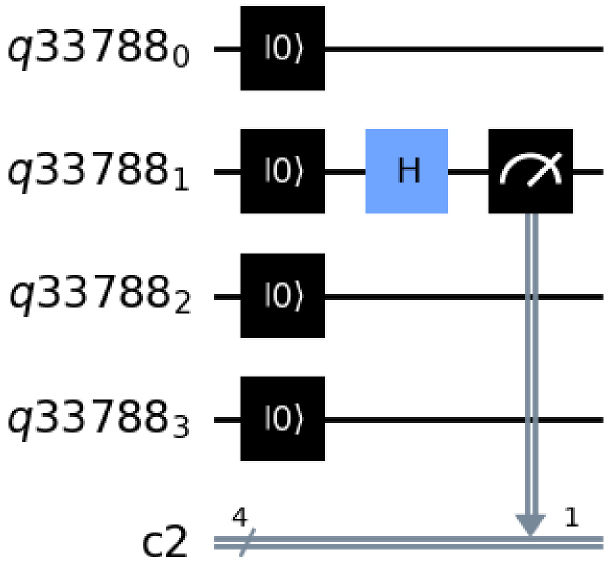
[48]:
```
circuit0.reset([q0[0], q0[1], q0[2], q0[3]])  # Qubit state initialization "0"
display(circuit0.draw(output="mpl"))  # Sketch of a quantum circuit
```



Generation of random number $x_A$

[49]:
```
circuit0.h(q0[1])
circuit0.measure(q0[1], c0[1])
circuit0.draw(output="mpl")  # Sketch of a quantum circuit
```

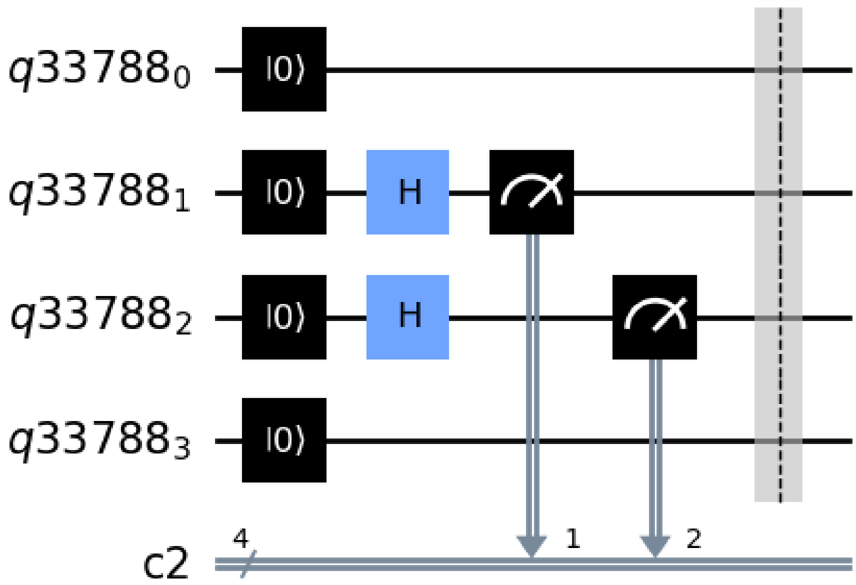
[49]:




Generation of random number $y_A$

[50]:
```
circuit0.h(q0[2])
circuit0.measure(q0[2], c0[2])
circuit0.barrier(q0[0], q0[1], q0[2], q0[3])
circuit0.draw(output="mpl")  # Sketch of a quantum circuit
```

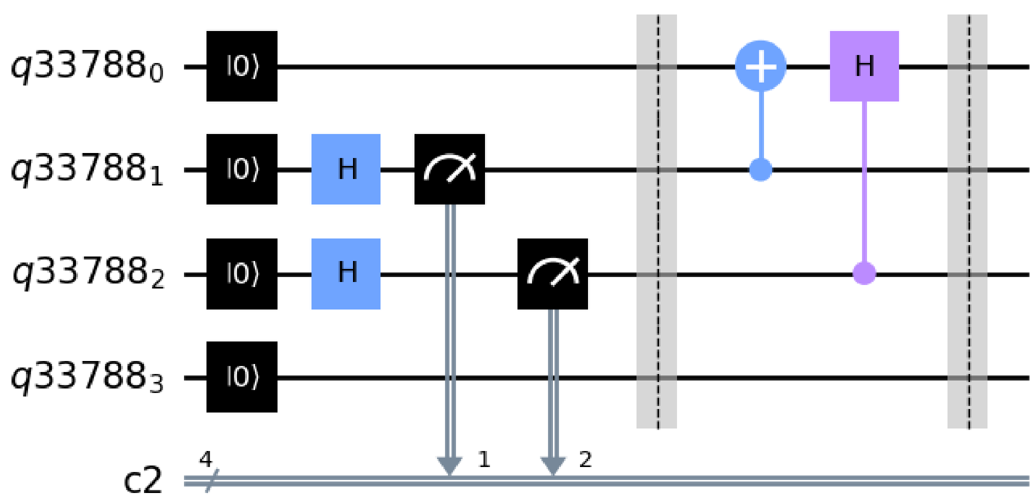
[50]:




Information coding by Alice

[51]:
```
circuit0.cx(q0[1], q0[0])
circuit0.ch(q0[2], q0[0])
circuit0.barrier(q0[0], q0[1], q0[2], q0[3])
circuit0.draw(output="mpl")   # Sketch of a quantum circuit
```

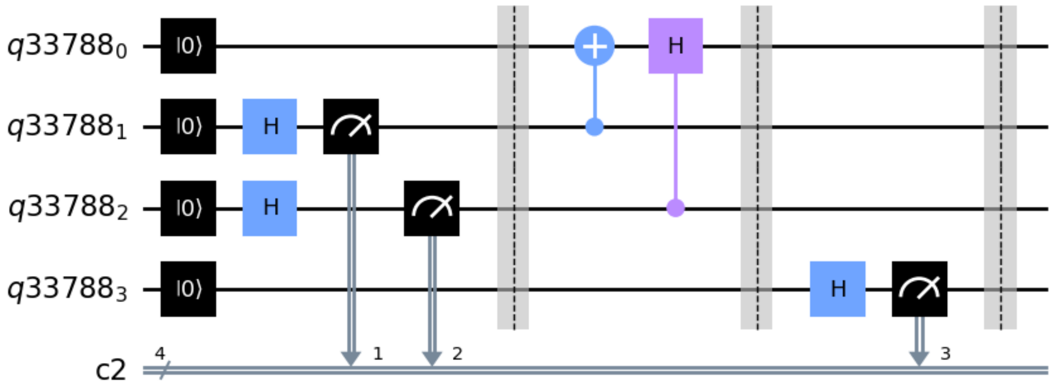
[51]:



Generation of random number $y_B$

[52]:
```
circuit0.h(q0[3])
circuit0.measure(q0[3], c0[3])
circuit0.barrier(q0[0], q0[1], q0[2], q0[3])
circuit0.draw(output="mpl")  # Sketch of a quantum circuit
```

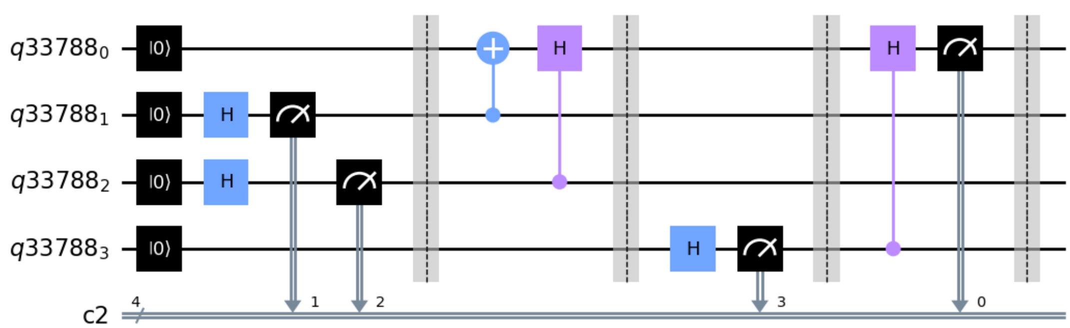
[52]:




Bob decoding information

[53]:
```
circuit0.ch(q0[3], q0[0])
circuit0.measure(
    q0[0], c0[0]
)  # Checking qubit states - quantum measurement on qubit "0„
circuit0.barrier(q0[0], q0[1], q0[2], q0[3])
circuit0.draw(output="mpl")  # Sketch of a quantum circuit
```

[53]:




Selecting a simulator and test run

[54]:
```
# Selecting a quantum simulator
backend = BasicAer.get_backend("qasm_simulator")
# Performing quantum calculations
job_sim0 = execute(circuit0, backend, shots=1)
sim_result0 = job_sim0.result()
wynik = sim_result0.get_counts(circuit0)
# Numerical presentation of measurement results
print(wynik)
```

```
{'1011': 1}
```

Generation of a sequence of numbers $x_B$ (including the remaining numbers)

[55]:
```
sample = 10
bit = []
for kk in range(sample):
    job_sim0 = execute(circuit0, backend, shots=1)
    sim_result0 = job_sim0.result()
    wynik = sim_result0.get_counts(circuit0)
    xA = int(list(wynik.keys())[0][2])
    yA = int(list(wynik.keys())[0][1])
    yB = int(list(wynik.keys())[0][0])
    xB = int(list(wynik.keys())[0][3])
    print(wynik, "->", [xA, yA, yB, xB])
    bit.append([xA, yA, yB, xB])
print(bit)
```

```
{'0011': 1} -> [1, 0, 0, 1]
{'1000': 1} -> [0, 0, 1, 0]
{'0110': 1} -> [1, 1, 0, 0]
{'0011': 1} -> [1, 0, 0, 1]
{'0000': 1} -> [0, 0, 0, 0]
{'0011': 1} -> [1, 0, 0, 1]
{'1100': 1} -> [0, 1, 1, 0]
{'0100': 1} -> [0, 1, 0, 0]
{'0100': 1} -> [0, 1, 0, 0]
{'0111': 1} -> [1, 1, 0, 1]
[[1, 0, 0, 1], [0, 0, 1, 0], [1, 1, 0, 0], [1, 0, 0, 1], [0, 0, 0, 0], [1, 0, 0,
1], [0, 1, 1, 0], [0, 1, 0, 0], [0, 1, 0, 0], [1, 1, 0, 1]]
```

Key sifting

[56]:
```
kluczA = []
kluczB = []
for bb in bit:
    if bb[1] == bb[2]:
        kluczA.append(bb[0])
        kluczB.append(bb[3])
        print("yA=", bb[1], "yB=", bb[2], "->", "xA=", bb[0], ",xB=", bb[3])
```

```
print("kluczA=", kluczA)
print("kluczB=", kluczB)
```

```
yA= 0 yB= 0 -> xA= 1 ,xB= 1
yA= 0 yB= 0 -> xA= 1 ,xB= 1
yA= 0 yB= 0 -> xA= 0 ,xB= 0
yA= 0 yB= 0 -> xA= 1 ,xB= 1
yA= 1 yB= 1 -> xA= 0 ,xB= 0
kluczA= [1, 1, 0, 1, 0]
kluczB= [1, 1, 0, 1, 0]
```

Let's write the code for key sifting as functions

[57]:
```
def generate_bit_sequence(sample_size: int) -> list:
    bit = []
    for kk in range(sample_size):
        job_sim0 = execute(circuit0, backend, shots=1)
        sim_result0 = job_sim0.result()
        wynik = sim_result0.get_counts(circuit0)
        xA = int(list(wynik.keys())[0][2])
        yA = int(list(wynik.keys())[0][1])
        yB = int(list(wynik.keys())[0][0])
        xB = int(list(wynik.keys())[0][3])
        bit.append([xA, yA, yB, xB])
    return bit
```

[58]:
```
def calculate_n_agreements(bit: list) -> int:
    n_agreements = 0
    kluczA = []
    kluczB = []
    for bb in bit:
        if bb[1] == bb[2]:
            if bb[0] == bb[3]:
                n_agreements += 1
    return n_agreements
```

Run the experiment over 3 repetitions and for various sample sizes

[59]:
```
n_repetitions = 3
sample_sizes = [16, 32, 64, 128, 256]

results = [[ss] for ss in sample_sizes]

for _ in range(n_repetitions):
    for idx, sample_size in enumerate(sample_sizes):
        bit = generate_bit_sequence(sample_size)
        n_agreements = calculate_n_agreements(bit)
        results[idx].append(n_agreements)
```

```python
results_df = pd.DataFrame(
    results, columns=["Sample Size", "Test 1", "Test 2", "Test 3"]
)
results_df.set_index("Sample Size", inplace=True)
```

Key sifting results over 3 repetitions

[60]:
```python
display(results_df)
```

```
             Test 1  Test 2  Test 3
Sample Size
16                6       7       7
32               16      13      12
64               28      36      31
128              67      66      61
256             138     130     146
```