In [1]:
```
import numpy as np
```

In [2]:
```
# Aggregation function
def g(X):
    return np.sum(X,axis=1)
```

In [3]:
```
# Decision function
def f(X,operation):
    y=[]
    if operation.upper() == "NOT":
        b=0
        for i in range(X.shape[0]):
            if X[i] == b:
                y.append([1])
            else:
                y.append([0])
    else:
        g_val = g(X)
        if operation.upper() == 'NOR':
            b = 0
            for i in range(X.shape[0]):
                if g_val[i] == b: # neuron fires
                    y.append([1])
                else: # neuron does not fire
                    y.append([0])
        else:
            if operation.upper() == "AND":
                b = X.shape[1]
            elif operation.upper() == "OR":
                b = 1
            for i in range(X.shape[0]):
                if g_val[i] >= b: # neuron fires
                    y.append([1])
                else: # neuron does not fire
                    y.append([0])
    return np.array(y)
```

In [4]:
```
def mcp():
    inp = "y"
    while inp.lower() == "y":
        operation = input("Operation to be performed: ")
        n = int(input("Enter number of instances: "))
        if operation.upper() in ["AND","OR", "NOR"]:
            m = int(input("Enter number of features: "))
        elif operation.upper() == "NOT":
            m = 1
        else:
            print("Error: Invalid operation!")
            continue
        X = np.random.randint(2,size=(n,m))
        print("\nInput table:\n",X)
        output = f(X,operation)
        print(f"\nOutput of {operation} operation is:\n{output}")
        inp = input("\nContinue y/n? ")
```

In [5]:
```
mcp()
```

```
Operation to be performed: AND
Enter number of instances: 4
Enter number of features: 3

Input table:
 [[1 1 1]
 [1 1 1]
 [0 1 0]
 [0 1 1]]

Output of AND operation is:
[[1]
 [1]
 [0]
 [0]]

Continue y/n? y
Operation to be performed: OR
Enter number of instances: 3
Enter number of features: 3

Input table:
 [[0 1 1]
 [0 1 0]
 [0 0 1]]

Output of OR operation is:
[[1]
 [1]
 [1]]

Continue y/n? n
```

In [ ]: