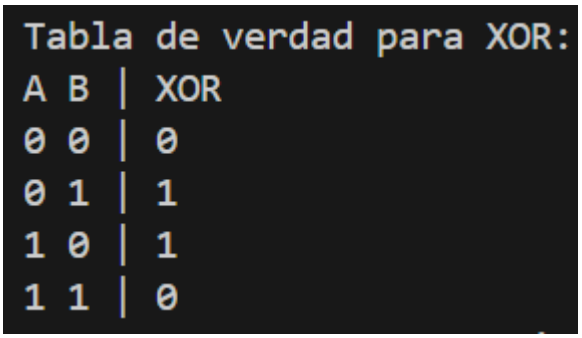
Implementacion

Diseñen y programen la operación XOR (3 puntos) la cual requiere mínimo 3 neuronas de MyP interconectadas.

```python
# Definición de la clase NeuronaMyP que implementa una neurona perceptrón simple
class NeuronaMyP:
    # Método constructor que recibe los pesos y el umbral
    def __init__(self, pesos, umbral):
        # Asigna los pesos a la neurona (lista de valores numéricos)
        self.pesos = pesos
        # Asigna el umbral de activación (valor numérico)
        self.umbral = umbral

    # Método para activar la neurona con un conjunto de entradas
    def activar(self, entradas):
        # Calcula la suma ponderada de las entradas multiplicadas por los pesos
        suma = sum(entrada * peso for entrada, peso in zip(entradas, self.pesos))
        # Retorna 1 si la suma supera o iguala el umbral, 0 en caso contrario
        return 1 if suma >= self.umbral else 0
# Creación de una neurona NAND (NOT AND) con pesos [1,1] y umbral 2
# Esta configuración implementa la función lógica NAND
neurona_nand = NeuronaMyP(pesos=[1, 1], umbral=2)
# Creación de una neurona OR con pesos [1,1] y umbral 1
# Esta configuración implementa la función lógica OR
neurona_or = NeuronaMyP(pesos=[1, 1], umbral=1)
# Creación de una neurona AND con pesos [1,1] y umbral 2
# Esta configuración implementa la función lógica AND
neurona_and = NeuronaMyP(pesos=[1, 1], umbral=2)
```

```python
# Función para implementar la compuerta XOR usando combinación de NAND, OR y AND
def xor(a, b):
    # Calcula el NAND de las entradas (y aplica NOT al resultado)
    nand = 1 if neurona_nand.activar([a, b]) == 0 else 0
    # Calcula el OR de las entradas
    or_result = neurona_or.activar([a, b])
    # Calcula el AND del resultado del NAND y del OR
    return neurona_and.activar([nand, or_result])
# Impresión de la tabla de verdad para XOR
print("Tabla de verdad para XOR:")
# Encabezado de la tabla
print("A B | XOR")
# Prueba con todas las combinaciones posibles de entradas (0 y 1)
print("0 0 |", xor(0, 0))  # 0 XOR 0 = 0
print("0 1 |", xor(0, 1))  # 0 XOR 1 = 1
print("1 0 |", xor(1, 0))  # 1 XOR 0 = 1
print("1 1 |", xor(1, 1))  # 1 XOR 1 = 0
```



📌 También programen las 12 neuronas de la imagen anterior y comprueben sus salidas al ejecutar cada combinación de entradas posibles.

📌 Agreguen **comentarios** en el código para describir su funcionamiento.

```python
# Definición de la clase base Neurona
class Neurona:
    # Método constructor que recibe pesos y umbral
    def __init__(self, pesos, umbral):
        # Asigna los pesos a la neurona
        self.pesos = pesos
        # Asigna el umbral de activación
        self.umbral = umbral
    # Método para activar la neurona con unas entradas dadas
    def activar(self, entradas):
        # Calcula la suma ponderada de entradas por pesos
        suma = sum(e * p for e, p in zip(entradas, self.pesos))
        # Retorna 1 si la suma supera el umbral, 0 en caso contrario
        return 1 if suma >= self.umbral else 0  # Función de activación con umbral
# Neurona que retrasa la señal (no modifica la entrada)
class NeuronaDelay(Neurona):
    def __init__(self):
        pass  # No necesita inicialización
    # Método que simplemente retorna la entrada sin cambios
    def activar(self, entrada):
        return entrada  # Simplemente retrasa la señal
# Neurona que implementa la función lógica AND
class NeuronaAnd(Neurona):
    def __init__(self):
        # Configura pesos [1,1] y umbral 2 para funcionar como AND
        super().__init__([1, 1], 2)
# Neurona que implementa la función lógica OR
class NeuronaOr(Neurona):
    def __init__(self):
        # Configura pesos [1,1] y umbral 1 para funcionar como OR
        super().__init__([1, 1], 1)
```

```python
# Neurona que implementa la función mayoría (devuelve 1 si la mayoría de entradas son 1)
class NeuronaMajority(Neurona):
    def __init__(self, num_entradas):
        # Configura pesos 1 para todas las entradas y umbral en la mitad+1
        super().__init__([1] * num_entradas, num_entradas // 2 + 1)
# Neurona que implementa la función AND NOT (A AND NOT B)
class NeuronaAndNot(Neurona):
    def __init__(self):
        # Configura pesos [1,-1] y umbral 1
        super().__init__([1, -1], 1)
# Neurona con configuración específica (función no estándar)
class NeuronaI(Neurona):
    def __init__(self):
        # Configura pesos [-1,-1] y umbral 0
        super().__init__([-1, -1], 0)
# Neurona AND de 3 entradas
class NeuronaAND(Neurona):
    def __init__(self):
        # Configura pesos [1,1,1] y umbral 3
        super().__init__([1, 1, 1], 3)
# Neurona OR de 3 entradas
class NeuronaOR(Neurona):
    def __init__(self):
        # Configura pesos [1,1,1] y umbral 1
        super().__init__([1, 1, 1], 1)
# Neurona con configuración específica para 4 entradas
class Neurona2I(Neurona):
    def __init__(self):
        # Configura pesos [1,1,-1,-1] y umbral 2
        super().__init__([1, 1, -1, -1], 2)
```

```python
# Neurona NOT (inversor)
class NeuronaNot(Neurona):
    def __init__(self):
        # Configura peso [1] y umbral 0
        super().__init__([1], 0)
# Neurona NOT con peso negativo (otra implementación de NOT)
class NeuronaNotI(Neurona):
    def __init__(self):
        # Configura peso [-1] y umbral 0
        super().__init__([-1], 0)
# Función para probar todas las neuronas
def prueba_neuronas():
    # Pruebas de NeuronaDelay
    print("Neurona Delay:", NeuronaDelay().activar(0))  # Debería retornar 0
    print("Neurona Delay:", NeuronaDelay().activar(1))  # Debería retornar 1
    # Pruebas de NeuronaAnd (AND de 2 entradas)
    print("Neurona AND:", NeuronaAnd().activar([0, 0]))  # 0 AND 0 = 0
    print("Neurona AND:", NeuronaAnd().activar([0, 1]))  # 0 AND 1 = 0
    print("Neurona AND:", NeuronaAnd().activar([1, 0]))  # 1 AND 0 = 0
    print("Neurona AND:", NeuronaAnd().activar([1, 1]))  # 1 AND 1 = 1
    # Pruebas de NeuronaOr (OR de 2 entradas)
    print("Neurona OR:", NeuronaOr().activar([0, 0]))   # 0 OR 0 = 0
    print("Neurona OR:", NeuronaOr().activar([0, 1]))   # 0 OR 1 = 1
    print("Neurona OR:", NeuronaOr().activar([1, 9]))   # 1 OR 1 = 1 (9 se considera como 1)
    print("Neurona OR:", NeuronaOr().activar([1, 1]))   # 1 OR 1 = 1
```

```python
# Pruebas de NeuronaMajority con 3 entradas
print("Neurona Majority:", NeuronaMajority(3).activar([0, 0, 0]))  # 1
print("Neurona Majority:", NeuronaMajority(3).activar([0, 0, 1]))  # 1
print("Neurona Majority:", NeuronaMajority(3).activar([0, 1, 0]))  # 1
print("Neurona Majority:", NeuronaMajority(3).activar([0, 1, 1]))  # 1
print("Neurona Majority:", NeuronaMajority(3).activar([1, 0, 0]))  # 1
print("Neurona Majority:", NeuronaMajority(3).activar([1, 0, 1]))  # 1
print("Neurona Majority:", NeuronaMajority(3).activar([1, 1, 0]))  # 1
print("Neurona Majority:", NeuronaMajority(3).activar([1, 1, 1]))  # 1
# ... (continúan pruebas similares para todas combinaciones)
# Pruebas de NeuronaAndNot
print("Neurona AND NOT:", NeuronaAndNot().activar([0, 0]))  # 0 AND NOT 0 = 0
print("Neurona AND NOT:", NeuronaAndNot().activar([0, 1]))  # 0 AND NOT 1 = 0
print("Neurona AND NOT:", NeuronaAndNot().activar([1, 0]))  # 1 AND NOT 0 = 1
print("Neurona AND NOT:", NeuronaAndNot().activar([1, 1]))  # 1 AND NOT 1 = 0
# Pruebas de NeuronaI (configuración especial)
print("Neurona I:", NeuronaI().activar([0, 0]))  # 1
print("Neurona I:", NeuronaI().activar([0, 1]))  # 1
print("Neurona I:", NeuronaI().activar([1, 0]))  # 1
print("Neurona I:", NeuronaI().activar([1, 1]))  # 1
# ... (continúan pruebas)
# Pruebas de NeuronaAND de 3 entradas
print("Neurona AND:", NeuronaAND().activar([0, 0, 0]))  # 1
print("Neurona AND:", NeuronaAND().activar([0, 0, 1]))  # 1
print("Neurona AND:", NeuronaAND().activar([0, 1, 0]))  # 1
print("Neurona AND:", NeuronaAND().activar([0, 1, 1]))  # 1
print("Neurona AND:", NeuronaAND().activar([1, 0, 0]))  # 1
print("Neurona AND:", NeuronaAND().activar([1, 0, 1]))  # 1
print("Neurona AND:", NeuronaAND().activar([1, 1, 0]))  # 1
print("Neurona AND:", NeuronaAND().activar([1, 1, 1]))  # 1
# ... (continúan pruebas)
```

```python
# Pruebas de NeuronaOR de 3 entradas
print("Neurona OR:", NeuronaOR().activar([0, 0, 0]))  # 1
print("Neurona OR:", NeuronaOR().activar([0, 0, 1]))  # 1
print("Neurona OR:", NeuronaOR().activar([0, 1, 0]))  # 1
print("Neurona OR:", NeuronaOR().activar([0, 1, 1]))  # 1
print("Neurona OR:", NeuronaOR().activar([1, 0, 0]))  # 1
print("Neurona OR:", NeuronaOR().activar([1, 0, 1]))  # 1
print("Neurona OR:", NeuronaOR().activar([1, 1, 0]))  # 1
print("Neurona OR:", NeuronaOR().activar([1, 1, 1]))  # 1
# ... (continúan pruebas)
# Pruebas de NeuronaMajority con 5 entradas
print("Neurona Majority:", NeuronaMajority(5).activar([0, 0, 0, 0, 0]))  # 1
print("Neurona Majority:",  class NeuronaMajority(num_entradas: int)  ) # 1
print("Neurona Majority:", NeuronaMajority(5).activar([0, 0, 0, 1, 0]))  # 1
print("Neurona Majority:", NeuronaMajority(5).activar([0, 0, 0, 1, 1]))  # 1
print("Neurona Majority:", NeuronaMajority(5).activar([0, 0, 1, 0, 0]))  # 1
print("Neurona Majority:", NeuronaMajority(5).activar([0, 0, 1, 0, 1]))  # 1
print("Neurona Majority:", NeuronaMajority(5).activar([0, 0, 1, 1, 0]))  # 1
print("Neurona Majority:", NeuronaMajority(5).activar([0, 0, 1, 1, 1]))  # 1
print("Neurona Majority:", NeuronaMajority(5).activar([0, 1, 0, 0, 0]))  # 1
print("Neurona Majority:", NeuronaMajority(5).activar([0, 1, 0, 0, 1]))  # 1
print("Neurona Majority:", NeuronaMajority(5).activar([0, 1, 0, 1, 0]))  # 1
print("Neurona Majority:", NeuronaMajority(5).activar([0, 1, 0, 1, 1]))  # 1
print("Neurona Majority:", NeuronaMajority(5).activar([0, 1, 1, 0, 0]))  # 1
print("Neurona Majority:", NeuronaMajority(5).activar([0, 1, 1, 0, 1]))  # 1
print("Neurona Majority:", NeuronaMajority(5).activar([0, 1, 1, 1, 0]))  # 1
print("Neurona Majority:", NeuronaMajority(5).activar([0, 1, 1, 1, 1]))  # 1
print("Neurona Majority:", NeuronaMajority(5).activar([1, 0, 0, 0, 0]))  # 1
print("Neurona Majority:", NeuronaMajority(5).activar([1, 0, 0, 0, 1]))  # 1
print("Neurona Majority:", NeuronaMajority(5).activar([1, 0, 0, 1, 0]))  # 1
print("Neurona Majority:", NeuronaMajority(5).activar([1, 0, 0, 1, 1]))  # 1
print("Neurona Majority:", NeuronaMajority(5).activar([1, 0, 1, 0, 0]))  # 1
```

```
print("Neurona Majority:", NeuronaMajority(5).activar([1, 0, 1, 0, 1]))  # 1
print("Neurona Majority:", NeuronaMajority(5).activar([1, 0, 1, 1, 0]))  # 1
print("Neurona Majority:", NeuronaMajority(5).activar([1, 0, 1, 1, 1]))  # 1
print("Neurona Majority:", NeuronaMajority(5).activar([1, 1, 0, 0, 0]))  # 1
print("Neurona Majority:", NeuronaMajority(5).activar([1, 1, 0, 0, 1]))  # 1
print("Neurona Majority:", NeuronaMajority(5).activar([1, 1, 0, 1, 0]))  # 1
print("Neurona Majority:", NeuronaMajority(5).activar([1, 1, 0, 1, 1]))  # 1
print("Neurona Majority:", NeuronaMajority(5).activar([1, 1, 1, 0, 0]))  # 1
print("Neurona Majority:", NeuronaMajority(5).activar([1, 1, 1, 0, 1]))  # 1
print("Neurona Majority:", NeuronaMajority(5).activar([1, 1, 1, 1, 0]))  # 1
print("Neurona Majority:", NeuronaMajority(5).activar([1, 1, 1, 1, 1]))  # 1
# ... (continúan pruebas para varias combinaciones)
# Pruebas de Neurona2I (4 entradas)
print("Neurona 2 I:", Neurona2I().activar([0, 0, 0, 0]))  # 1
print("Neurona 2 I:", Neurona2I().activar([0, 0, 0, 1]))  # 1
print("Neurona 2 I:", Neurona2I().activar([0, 0, 1, 0]))  # 1
print("Neurona 2 I:", Neurona2I().activar([0, 0, 1, 1]))  # 1
print("Neurona 2 I:", Neurona2I().activar([0, 1, 0, 0]))  # 1
print("Neurona 2 I:", Neurona2I().activar([0, 1, 0, 1]))  # 1
print("Neurona 2 I:", Neurona2I().activar([0, 1, 1, 0]))  # 1
print("Neurona 2 I:", Neurona2I().activar([0, 1, 1, 1]))  # 1
print("Neurona 2 I:", Neurona2I().activar([1, 0, 0, 0]))  # 1
print("Neurona 2 I:", Neurona2I().activar([1, 0, 0, 1]))  # 1
print("Neurona 2 I:", Neurona2I().activar([1, 0, 1, 0]))  # 1
print("Neurona 2 I:", Neurona2I().activar([1, 0, 1, 1]))  # 1
print("Neurona 2 I:", Neurona2I().activar([1, 1, 0, 0]))  # 1
print("Neurona 2 I:", Neurona2I().activar([1, 1, 0, 1]))  # 1
print("Neurona 2 I:", Neurona2I().activar([1, 1, 1, 0]))  # 1
print("Neurona 2 I:", Neurona2I().activar([1, 1, 1, 1]))  # 1
# ... (continúan pruebas)
```

```
    # Pruebas de NeuronaNot (inversor)
    print("Neurona Not:", NeuronaNot().activar([0]))  # NOT 0 = 1
    print("Neurona Not:", NeuronaNot().activar([1]))  # NOT 1 = 0
    # Pruebas de NeuronaNotI (otra implementación de NOT)
    print("Neurona NotI:", NeuronaNotI().activar([0]))  # NOT 0 = 1
    print("Neurona NotI:", NeuronaNotI().activar([1]))  # NOT 1 = 0
# Ejecuta las pruebas
prueba_neuronas()
```

```
Neurona Delay: 0
Neurona Delay: 1
Neurona AND: 0
Neurona AND: 0
Neurona AND: 0
Neurona AND: 1
Neurona OR: 0
Neurona OR: 1
Neurona OR: 1
Neurona OR: 1
Neurona Majority: 0
Neurona Majority: 0
Neurona Majority: 0
Neurona Majority: 1
Neurona Majority: 0
Neurona Majority: 1
Neurona Majority: 1
Neurona Majority: 1
Neurona AND NOT: 0
Neurona AND NOT: 0
Neurona AND NOT: 1
Neurona AND NOT: 0
Neurona I: 1
Neurona I: 0
Neurona I: 0
Neurona I: 0
Neurona AND: 0
Neurona AND: 0
Neurona AND: 0
Neurona AND: 0
Neurona AND: 0
Neurona AND: 0
Neurona AND: 0
Neurona AND: 1
Neurona OR: 0
Neurona OR: 1
Neurona OR: 1
Neurona OR: 1
Neurona OR: 1
Neurona OR: 1
Neurona OR: 1
Neurona Majority: 0
Neurona Majority: 0
Neurona Majority: 0
Neurona Majority: 0
Neurona Majority: 0
Neurona Majority: 0
Neurona Majority: 0
Neurona Majority: 1
Neurona Majority: 0
Neurona Majority: 0
Neurona Majority: 1
Neurona Majority: 1
Neurona Majority: 1
Neurona Majority: 0
Neurona Majority: 0
```

```
Neurona Majority: 0
Neurona Majority: 1
Neurona Majority: 0
Neurona Majority: 1
Neurona Majority: 1
Neurona Majority: 1
Neurona Majority: 0  Neurona 2 I: 0
Neurona Majority: 1  Neurona 2 I: 0
Neurona Majority: 1  Neurona 2 I: 0
Neurona Majority: 1  Neurona 2 I: 0
Neurona Majority: 1  Neurona 2 I: 0
Neurona Majority: 1  Neurona 2 I: 0
Neurona Majority: 1  Neurona 2 I: 1
Neurona Majority: 1  Neurona 2 I: 0
Neurona 2 I: 0       Neurona 2 I: 0
Neurona 2 I: 0       Neurona 2 I: 0
Neurona 2 I: 0       Neurona Not: 1
Neurona 2 I: 0       Neurona Not: 1
Neurona 2 I: 0       Neurona NotI: 1
Neurona 2 I: 0       Neurona NotI: 0
```