```
namespace Architektura_Komputerowa
{
using System;
using System.Collections.Generic;

class Program
{
    static Dictionary<string, string> registers = new Dictionary<string, string>
    {
        { "AH", "00000000" },
        { "AL", "00000000" },
        { "BH", "00000000" },
        { "BL", "00000000" },
        { "CH", "00000000" },
        { "CL", "00000000" },
        { "DH", "00000000" },
        { "DL", "00000000" }
    };

    static void Main()
    {
        Console.WriteLine("Wprowadź wartości rejestrów (8 znaków binarnych dla
każdego rejestru):");
        foreach (var reg in registers)
        {
            Console.Write($"Rejestr {reg.Key}: ");
            string value = Console.ReadLine();

            while (value.Length != 8 || !IsBinary(value))
            {
                Console.WriteLine("Nieprawidłowa wartość binarna. Wprowadź 8
znaków 0 lub 1.");
                Console.Write($"Rejestr {reg.Key}: ");
                value = Console.ReadLine();
            }

            registers[reg.Key] = value;
        }

        while (true)
        {
            Console.Write("Podaj operację (AND, OR, XOR, NOT, NEG, MOV, INC, DEC,
ADD, SUB, QUIT): ");
            string operation = Console.ReadLine().ToUpper();

            if (operation == "QUIT")
                break;

            string sourceRegister = "";
            string destinationRegister = "";

            if (operation != "NOT" && operation != "NEG")
            {
                Console.Write("Podaj rejestry (np. AH AL): ");
                string[] registersInput = Console.ReadLine().Split();

                if (registersInput.Length != 2)
                {
                    Console.WriteLine("Nieprawidłowe dane wejściowe.");
                    continue;
                }

                sourceRegister = registersInput[0].ToUpper();
```

```
                destinationRegister = registersInput[1].ToUpper();

                if (!registers.ContainsKey(sourceRegister) ||
!registers.ContainsKey(destinationRegister))
                {
                    Console.WriteLine("Nieprawidłowy rejestr.");
                    continue;
                }
            }
            else
            {
                Console.Write("Podaj rejestr docelowy (np. AH): ");
                destinationRegister = Console.ReadLine().ToUpper();

                if (!registers.ContainsKey(destinationRegister))
                {
                    Console.WriteLine("Nieprawidłowy rejestr.");
                    continue;
                }
            }

            try
            {
                switch (operation)
                {
                    case "MOV":
                        PerformMOV(sourceRegister, destinationRegister);
                        break;
                    case "INC":
                        PerformINC(destinationRegister);
                        break;
                    case "DEC":
                        PerformDEC(destinationRegister);
                        break;
                    case "AND":
                        PerformAND(sourceRegister, destinationRegister);
                        break;
                    case "OR":
                        PerformOR(sourceRegister, destinationRegister);
                        break;
                    case "XOR":
                        PerformXOR(sourceRegister, destinationRegister);
                        break;
                    case "NOT":
                        PerformNOT(destinationRegister);
                        break;
                    case "NEG":
                        PerformNEG(destinationRegister);
                        break;
                    case "ADD":
                        PerformADD(sourceRegister, destinationRegister);
                        break;
                    case "SUB":
                        PerformSUB(sourceRegister, destinationRegister);
                        break;
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Wystąpił błąd: {ex.Message}");
            }

            DisplayRegisters();
```

```
    }
}

static void PerformMOV(string sourceRegister, string destinationRegister)
{
    registers[destinationRegister] = registers[sourceRegister];
}

static void PerformINC(string register)
{
    string value = registers[register];
    int result = Convert.ToInt32(value, 2) + 1;
    registers[register] = Convert.ToString(result % 256, 2).PadLeft(8, '0');
}

static void PerformDEC(string register)
{
    string value = registers[register];
    int result = Convert.ToInt32(value, 2) - 1;
    if (result < 0) result += 256;
    registers[register] = Convert.ToString(result % 256, 2).PadLeft(8, '0');
}

static void PerformAND(string sourceRegister, string destinationRegister)
{
    string srcValue = registers[sourceRegister];
    string dstValue = registers[destinationRegister];
    string result = "";

    for (int i = 0; i < 8; i++)
    {
        result += (srcValue[i] == '1' && dstValue[i] == '1') ? '1' : '0';
    }

    registers[destinationRegister] = result;
}

static void PerformOR(string sourceRegister, string destinationRegister)
{
    string srcValue = registers[sourceRegister];
    string dstValue = registers[destinationRegister];
    string result = "";

    for (int i = 0; i < 8; i++)
    {
        result += (srcValue[i] == '1' || dstValue[i] == '1') ? '1' : '0';
    }

    registers[destinationRegister] = result;
}

static void PerformXOR(string sourceRegister, string destinationRegister)
{
    string srcValue = registers[sourceRegister];
    string dstValue = registers[destinationRegister];
    string result = "";

    for (int i = 0; i < 8; i++)
    {
        result += (srcValue[i] != dstValue[i]) ? '1' : '0';
    }

    registers[destinationRegister] = result;
```

```
    }

    static void PerformNOT(string destinationRegister)
    {
        string dstValue = registers[destinationRegister];
        string result = "";

        for (int i = 0; i < 8; i++)
        {
            result += (dstValue[i] == '1') ? '0' : '1';
        }

        registers[destinationRegister] = result;
    }

    static void PerformNEG(string destinationRegister)
    {
        string dstValue = registers[destinationRegister];
        int value = Convert.ToInt32(dstValue, 2);
        if (value == 0)
        {
            registers[destinationRegister] = "00000000";
        }
        else
        {
            value = 256 - value;
            registers[destinationRegister] = Convert.ToString(value,
2).PadLeft(8, '0');
        }
    }

    static void PerformADD(string sourceRegister, string destinationRegister)
    {
        int srcValue = Convert.ToInt32(registers[sourceRegister], 2);
        int dstValue = Convert.ToInt32(registers[destinationRegister], 2);
        int result = (srcValue + dstValue) % 256;
        registers[destinationRegister] = Convert.ToString(result, 2).PadLeft(8,
'0');
    }

    static void PerformSUB(string sourceRegister, string destinationRegister)
    {
        int srcValue = Convert.ToInt32(registers[sourceRegister], 2);
        int dstValue = Convert.ToInt32(registers[destinationRegister], 2);
        int result = (dstValue - srcValue + 256) % 256;
        registers[destinationRegister] = Convert.ToString(result, 2).PadLeft(8,
'0');
    }

    static bool IsBinary(string input)
    {
        foreach (char c in input)
        {
            if (c != '0' && c != '1')
                return false;
        }
        return true;
    }

    static void DisplayRegisters()
    {
        foreach (var reg in registers)
        {
```

```
                Console.WriteLine($"Rejestr {reg.Key}: {reg.Value}");
            }
        }
    }
}
```