ASK-Komunikator - temat 5

Jakub Dobruchowski 188868
Przemysław Piątkiewicz 188823

27.05.2024

1 Zadanie

1.1 Temat

Prosty mikrokontroler

1.2 Cel

Projekt oraz implementacja aplikacji, która będzie symulować zachowanie prostego mikrokontrolera

2 Zagadnienia szczegółowe

2.1 Rejestry

Procesor będzie posiadał cztery 16-bitowe rejestry ogólnego przeznaczenia oznaczone jako AX, BX, CX i DX. Każdy z rejestrów będzie traktowany jako para 8-bitowych rejestrów o oznaczeniach NH dla części starszej i NL dla części młodszej gdzie N oznacza A albo B albo C albo D

2.2 Rozkazy

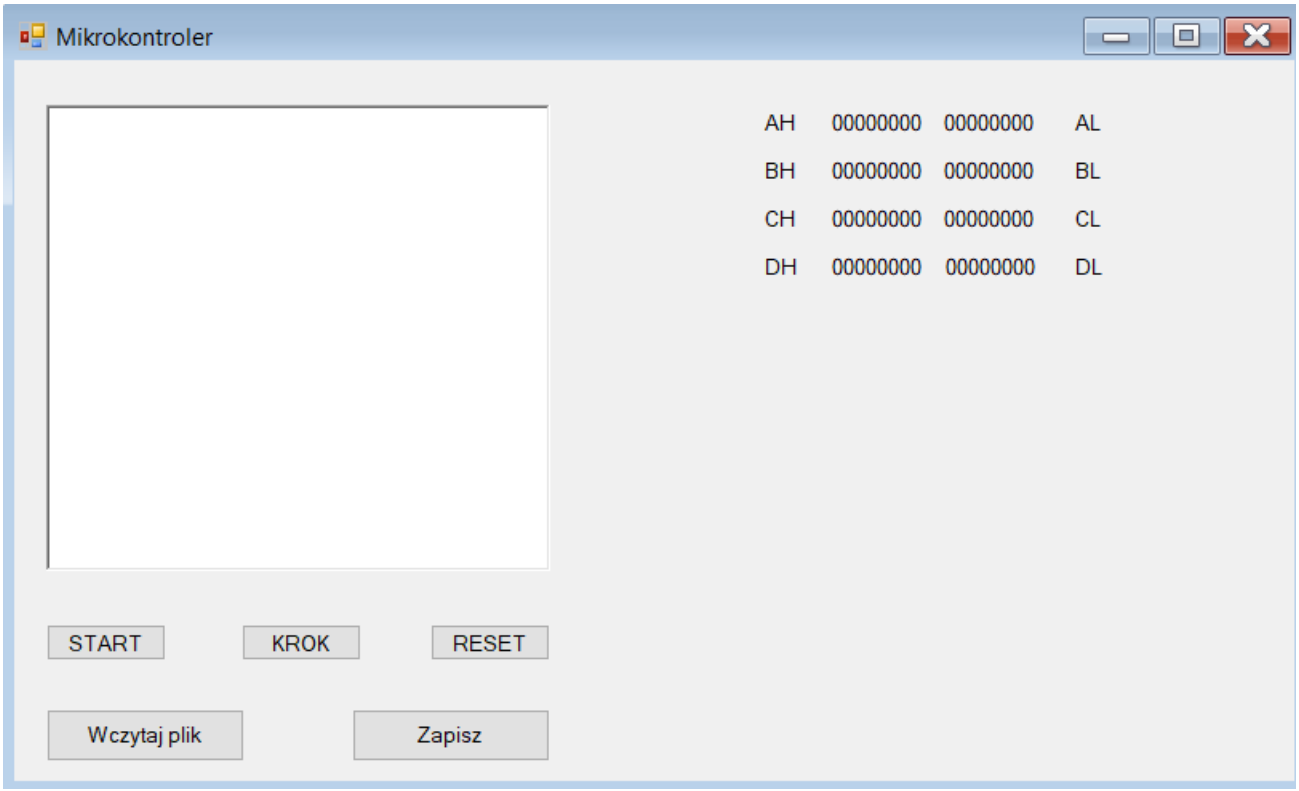
Lista rozkazów procesora obejmje trzy rozkazy MOV – przesłania, ADD – dodawania i SUB – odejmowania. Procesor będzie umożliwiał realizację dwóch trybów adresowania: trybu rejestrowego oraz trybu natychmiastowego.

2.3 Ogólne działanie

Program będzie umożliwiał pisanie krótkich programów z użyciem dostępnych rozkazów i trybów adresowania. Da możliwość wczytania programu z pliku tekstowego. Program umożliwi realizację napisanych programów w trybie całościowego wykonania i w trybie pracy krokowej oraz śledzenie postępu w wykonywaniu programu dla pracy krokowej.

3 Opis przyjętych rozwiązań programowych

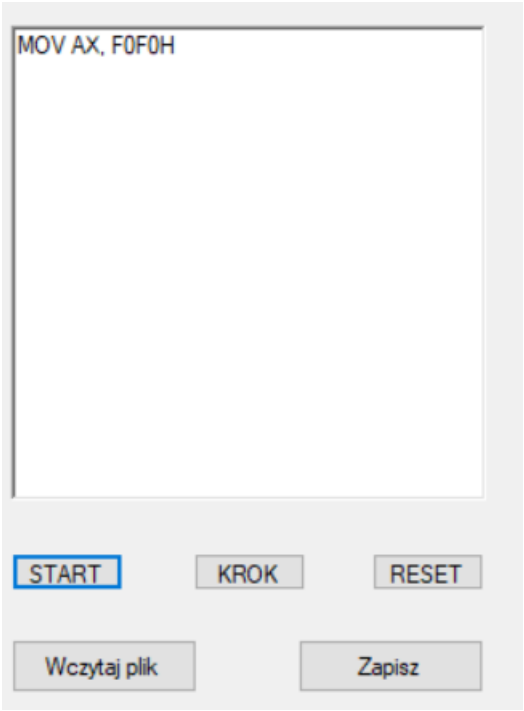
3.1 Interfejs



Rysunek 1: Interfejs użytkownika

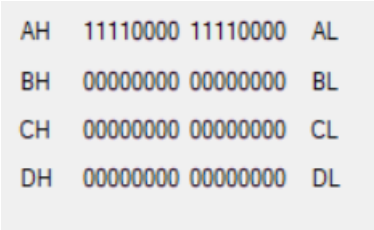
Interfejs użytkownika obejmuje pole wpisywania rozkazów (richTextBox), pięć przycisków umożliwiających:

1. wykonanie programu
2. pracę krokową
3. reset programu
4. wczytanie pliku
5. zapisanie programu do pliku



Rysunek 2: Pole wpisywania rozkazów i przyciski

Po prawej stronie interfejsu umieszczone zostały rejestry 16-bitowe podzielone na 8-bitowe sekcje, zostały do tego użyte kontrolki "label".



Rysunek 3: Rejestry

3.2 Rozkazy:

MOV: Rozkaz MOV przypisuje wartość liczbową do rejestru. Liczba może być wpisywana dziesiętnie lub heksadecymalnnie dodając na koniec znak "H". Prawidłowa składnia rozkazu to: MOV ¡rejestr¿, ¡liczba¿, czyli np. MOV AX, 255 lub MOV AX, FFH dla wpisania samych jedynek w młodszej części rejestru AX (AL).

```
private void MovCommand(string[] args)
{
    if (args.Length != 3)
    {
        MessageBox.Show("Nieprawidłowy format rozkazu MOV. Oczekiwano: MOV <rejestr>, <liczba>");
        return;
    }

    string register = args[1].Trim().ToUpper().TrimEnd(',');
    string valueString = args[2].Trim();

    bool isHex = valueString.EndsWith("H", StringComparison.OrdinalIgnoreCase);

    if (isHex)
    {
        valueString = valueString.Substring(0, valueString.Length - 1);
    }

    int value;
    try
    {
        value = isHex ? int.Parse(valueString, NumberStyles.HexNumber) : int.Parse(valueString);
    }
    catch (FormatException)
    {
        MessageBox.Show("Invalid value format. Value must be a valid number.");
        return;
    }

    SetRegisterValue(register, value);
}
```

Rysunek 4: Kod dla obsługi rozkazu MOV

ADD Rozkaz ADD służy do dodania liczby do liczby zawartej w rejestrze lub do dodania dwóch liczb zawartych w różnych rejestrach. Liczba może być wpisywana dziesiętnie lub heksadecymalnnie dodając na koniec znak ”H”. Prawidłowa składnia rozkazu to: ADD ¡rejestr¿, ¡rejestr¿/¡liczba¿, czyli np. ADD AX, 16 lub ADD AX, 10H dla dodania liczby 16DEC do liczby zapisanej w rejestrze AX. Inną prawidłową składnią jest ADD AX, BX. W tym przypadku w rejestrze AX zostanie zapisana suma liczb zapisanych w rejestrach AX i BX.

```
private void AddCommand(string[] args)
{
    if (args.Length != 3)
    {
        MessageBox.Show("Nieprawidłowy format rozkazu ADD. Oczekiwano: ADD <rejestr>, <rejestr/liczba>");
        return;
    }

    string destinationRegister = args[1].Trim().ToUpper().TrimEnd(',');
    string source = args[2].Trim().ToUpper();

    int sourceValue;
    if (IsRegister(source))
    {
        sourceValue = GetRegisterValue(source);
    }
    else
    {
        bool isHex = source.EndsWith("H", StringComparison.OrdinalIgnoreCase);

        if (isHex)
        {
            source = source.Substring(0, source.Length - 1);
        }

        try
        {
            sourceValue = isHex ? int.Parse(source, NumberStyles.HexNumber) : int.Parse(source);
        }
        catch (FormatException)
        {
            MessageBox.Show("Nieprawidłowy format liczby.");
            return;
        }
    }

    int destinationValue = GetRegisterValue(destinationRegister);

    int result = destinationValue + sourceValue;

    SetRegisterValue(destinationRegister, result);
}
```

Rysunek 5: Kod dla obsługi rozkazu ADD

SUB Rozkaz SUB służy do odjęcia liczby od liczby zawartej w rejestrze lub do odjęcia dwóch liczb zawartych w różnych rejestrach. Liczba może być wpisywana dziesiętnie lub heksadecymalnnie dodając na koniec znak ”H”. Prawidłowa składnia rozkazu to: SUB ¡rejestr¿, ¡rejestr¿/¡liczba¿, czyli np. ADD AX, 16 lub ADD AX, 10H dla odjęcia liczby 16DEC od liczby zapisanej w rejestrze AX. Inną prawidłową składnią jest ADD AX, BX. W tym przypadku w rejestrze AX zostanie zapisana różnica liczb zapisanych w rejestrach AX i BX.

```
private void SubCommand(string[] args)
{
    if (args.Length != 3)
    {
        MessageBox.Show("Nieprawidłowy format rozkazu SUB. Oczekiwano: SUB <rejestr>, <rejestr/liczba>");
        return;
    }

    string destinationRegister = args[1].Trim().ToUpper().TrimEnd(',');
    string source = args[2].Trim().ToUpper();

    int sourceValue;
    if (IsRegister(source))
    {
        sourceValue = GetRegisterValue(source);
    }
    else
    {
        // Czy heksadecymalna liczba?
        bool isHex = source.EndsWith("H", StringComparison.OrdinalIgnoreCase);

        //
        if (isHex)
        {
            source = source.Substring(0, source.Length - 1);
        }

        // Konwersja na int
        try
        {
            sourceValue = isHex ? int.Parse(source, NumberStyles.HexNumber) : int.Parse(source);
        }
        catch (FormatException)
        {
            MessageBox.Show("Invalid source value format. Value must be a valid number.");
            return;
        }
    }

    int destinationValue = GetRegisterValue(destinationRegister);


    int result = destinationValue - sourceValue;


    SetRegisterValue(destinationRegister, result);
}
```

Rysunek 6: Kod dla obsługi rozkazu SUB

3.3 Wykonanie programu:

Natychmiastowe: Po wciśnięciu przycisku "START" program załaduje linie zawarte w richTextBoxie do tablicy stringów "lines" i ustawi indeks linii na 0 używając funkcji "UpdateLinesFromTextBox".

```
private int currentLineIndex = 0;
private string[] lines;
```

Rysunek 7: Inicjallizacja zmiennych

Następnie przetworzy wszystkie linie kodu używając funkcji "ProcessLine"

```
private void ProcessLine(string line)
{
    string[] words = line.Split(new char[] { ' ', ',' }, StringSplitOptions.RemoveEmptyEntries);
    if (words.Length > 0)
    {
        string command = words[0].ToUpper();
        switch (command)
        {
            case "MOV":
                MovCommand(words);
                break;
            case "SUB":
                SubCommand(words);
                break;
            case "ADD":
                AddCommand(words);
                break;
        }
    }
}
```

Rysunek 8: Kod przetwarzania linii proggramu

Krokowe: Jeśli indeks linii jest równy 0, to zostaną załadowane linie programu zawartego w richTextBoxie, a następnie po każdym kliknięciu przycisku "KROK" wykona się linia o obeznym indeksie, a indeks zwiększy się o 1.

```
private void button2_Click(object sender, EventArgs e)
{
    if (lines == null || currentLineIndex >= lines.Length)
    {
        UpdateLinesFromTextBox();
    }

    if (currentLineIndex < lines.Length)
    {
        HighlightCurrentLine(currentLineIndex);
        ProcessLine(lines[currentLineIndex]);

        currentLineIndex++;
    }
    else
    {
        MessageBox.Show("No more lines to process.");
    }
}
```

Rysunek 9: Kod przycisku pracy krokowej

Obecna linia będzie podświetlona na żółto. Przycisk reset ustawia pracę krokową na początek programu.

```
private void HighlightCurrentLine(int lineIndex)
{
    richTextBox1.SelectAll();
    richTextBox1.SelectionBackColor = richTextBox1.BackColor;

    int start = richTextBox1.GetFirstCharIndexFromLine(lineIndex);
    int length = lines[lineIndex].Length;

    richTextBox1.Select(start, length);
    richTextBox1.SelectionBackColor = Color.Yellow;

    richTextBox1.ScrollToCaret();
}
```

Rysunek 10: Funkcja odpowiedzialna za zaznaczenie obecnej lini w pracy krokowej

dodatkowo dodano przycisk "RESET", który zeruje rejestry oraz ustawia wykonanie programu na początek (Pierwsza linia kodu).

```
private void button3_Click(object sender, EventArgs e)
{
    string zero = "00000000";
    UpdateLinesFromTextBox();
    HighlightCurrentLine(currentLineIndex);
    label3.Text = zero; // AH
    label4.Text = zero;
    label5.Text = zero; // BH
    label6.Text = zero;
    label9.Text = zero; // CH
    label10.Text = zero;
    label13.Text = zero; // DH
    label14.Text = zero;
}
```

Rysunek 11: Obsługa przycisku "Reset"

3.4 Wczytywanie pliku i zapis do pliku:

Przyciski "Wczytaj plik" i "zapisz" służą odpowiednio do wczytania pliku tekstowego do pola richTextBox i zapisania zawartości pola richTextBox do pliku tekstowego.

```
private void button4_Click(object sender, EventArgs e)
{
    OpenFileDialog openFileDialog = new OpenFileDialog();
    openFileDialog.Filter = "Text files (*.txt)|*.txt|All files (*.*)|*.*";
    openFileDialog.Title = "Open Text File";

    if (openFileDialog.ShowDialog() == DialogResult.OK)
    {
        string filePath = openFileDialog.FileName;
        try
        {
            string fileContent = File.ReadAllText(filePath);
            richTextBox1.Text = fileContent;
            UpdateLinesFromTextBox();
        }
        catch (Exception ex)
        {
            MessageBox.Show("An error occurred while reading the file: " + ex.Message);
        }
    }
}
```

Rysunek 12: Działanie przycisku "Wczytaj plik"

```
1 odwołanie
private void button5_Click(object sender, EventArgs e)
{
    SaveFileDialog saveFileDialog = new SaveFileDialog();
    saveFileDialog.Filter = "Text files (*.txt)|*.txt|All files (*.*)|*.*";
    saveFileDialog.Title = "Save Text File";

    if (saveFileDialog.ShowDialog() == DialogResult.OK)
    {
        string filePath = saveFileDialog.FileName;
        try
        {
            File.WriteAllText(filePath, richTextBox1.Text);
            MessageBox.Show("File saved successfully.");
        }
        catch (Exception ex)
        {
            MessageBox.Show("An error occurred while saving the file: " + ex.Message);
        }
    }
}
```

Rysunek 13: Działanie przycisku "zapisz"

3.5 Użyte funkcje:

SetRegisterValue Wstawia podaną liczbę do danego rejestru

```
private void SetRegisterValue(string register, int value)
{
    // konwersja na liczbę binarną rozszerzoną na 16 bitów
    string binaryValue = Convert.ToString(value, 2).PadLeft(16, '0');

    try
    {
        switch (register)
        {
            case "AX":
                string axHigh = binaryValue.Substring(0, 8);
                string axLow = binaryValue.Substring(8);
                label3.Text = axHigh; // AH
                label4.Text = axLow; // AL
                break;
            case "AH":
                label3.Text = binaryValue.Substring(binaryValue.Length - 8);
                break;
            case "AL":
                label4.Text = binaryValue.Substring(binaryValue.Length - 8);
                break;
            case "BX":
                string bxHigh = binaryValue.Substring(0, 8);
                string bxLow = binaryValue.Substring(8);
                label5.Text = bxHigh; // BH
                label6.Text = bxLow; // BL
                break;
            case "BH":
                label5.Text = binaryValue.Substring(binaryValue.Length - 8);
                break;
```

Rysunek 14: Funkcja SetRegisterValue

GetRegisterValue Pobiera liczbę z danego rejestru

```
Odwołania: 4
private int GetRegisterValue(string register)
{
    try
    {
        switch (register)
        {
            case "AX":
                string axHigh = label3.Text;
                string axLow = label4.Text;
                return Convert.ToInt32(axHigh + axLow, 2);
            case "AH":
                return Convert.ToInt32(label3.Text, 2);
            case "AL":
                return Convert.ToInt32(label4.Text, 2);
            case "BX":
                string bxHigh = label5.Text;
                string bxLow = label6.Text;
                return Convert.ToInt32(bxHigh + bxLow, 2);
            case "BH":
                return Convert.ToInt32(label5.Text, 2);
            case "BL":
                return Convert.ToInt32(label6.Text, 2);
            case "CX":
                string cxHigh = label9.Text;
                string cxLow = label10.Text;
                return Convert.ToInt32(cxHigh + cxLow, 2);
            case "CH":
                return Convert.ToInt32(label9.Text, 2);
            case "CL":
                return Convert.ToInt32(label10.Text, 2);
            case "DX":
                string dxHigh = label13.Text;
                string dxLow = label14.Text;
                return Convert.ToInt32(dxHigh + dxLow, 2);
            case "DH":
                return Convert.ToInt32(label13.Text, 2);
            case "DL":
                return Convert.ToInt32(label14.Text, 2);
            default:
                throw new ArgumentException($"Unknown register: {register}");
        }
    }
    catch (Exception)
    {
        MessageBox.Show($"Error reading register value for: {register}");
        return 0;
    }
}
```

Rysunek 15: Funkcja GetRegisterValue

IsRegister Sprawdza czy argument jest rejestrem

```
Odwołania: 2
private bool IsRegister(string value)
{
    string[] registers = { "AX", "AH", "AL",
    return registers.Contains(value);
}
```

Rysunek 16: Funkcja IsRegister