SYMULATOR intel 8086

Dominik Możdżeń 13757

WSEI Architektura Systemów Komputerowych

Wymagania

Projekt zaliczeniowy wymagał napisania programu który na ocenę:

I) 3.0
   1) Pozwala wprowadzić dane do rejestrów AH, AL, BH, BL, CH, CL, DH i DL
   2) Umożliwia operacje MOV oraz XCHG na rejestrach
   3) Wyświetla zawartość rejestrów po wykonaniu operacji

II) 3.5 Aplikacja posiada interfejs graficzny

III) 4.0 Aplikacja posiada rozkazy na jednym rejestrze:
   1) INC
   2) DEC
   3) NOT
   4) NEG

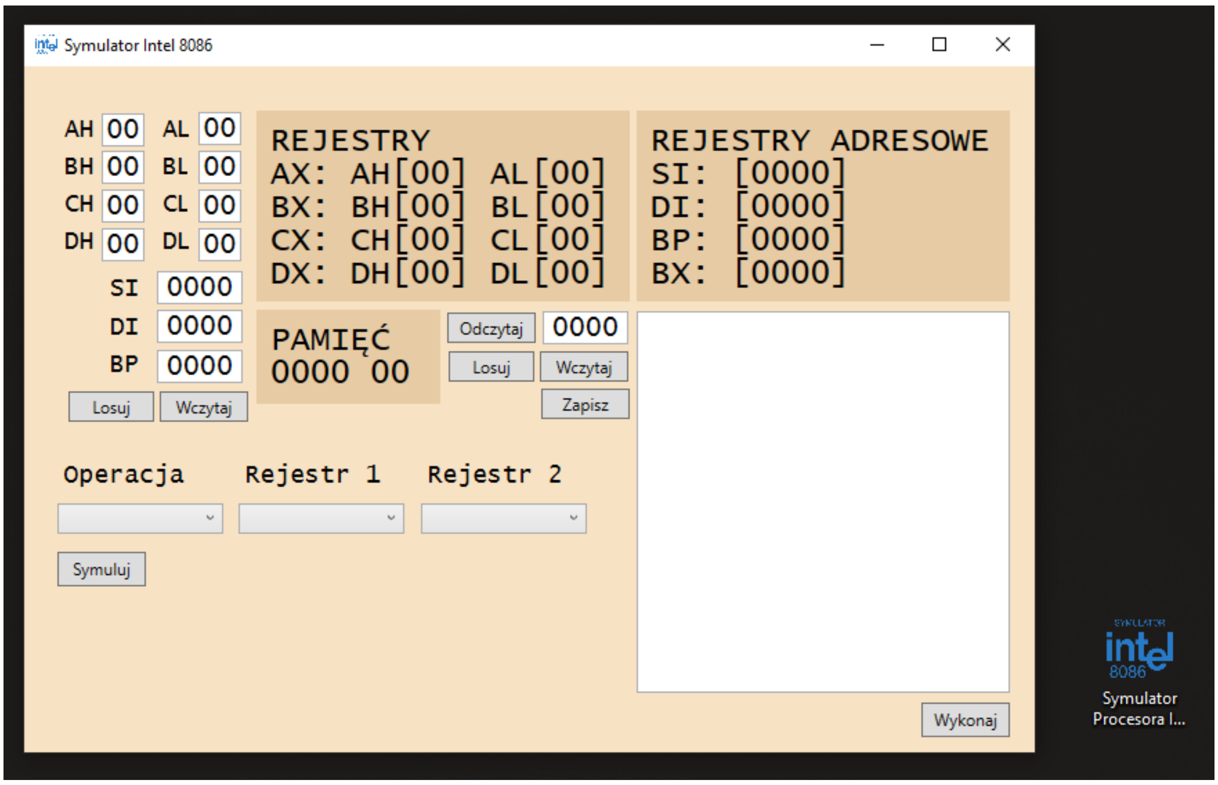
IV) 4.5 Aplikacja posiada rozkazy na dwóch rejestrach
   1) AND
   2) OR
   3) XOR
   4) ADD
   5) SUB

V) 5.0
   1) Posiada rejestry adresowe SI, DI, BP i BX
   2) Aplikacja umożliwia operacje na pamięci 64kB
      a) Za pomocą adresowania bezpośredniego
      b) Za pomocą adresowania pośredniego
         - Indeksowego
         - Bazowego
         - Indeksowo bazowego

Repozytorium na githubie:

https://github.com/Estremo102/8086

Interfejs



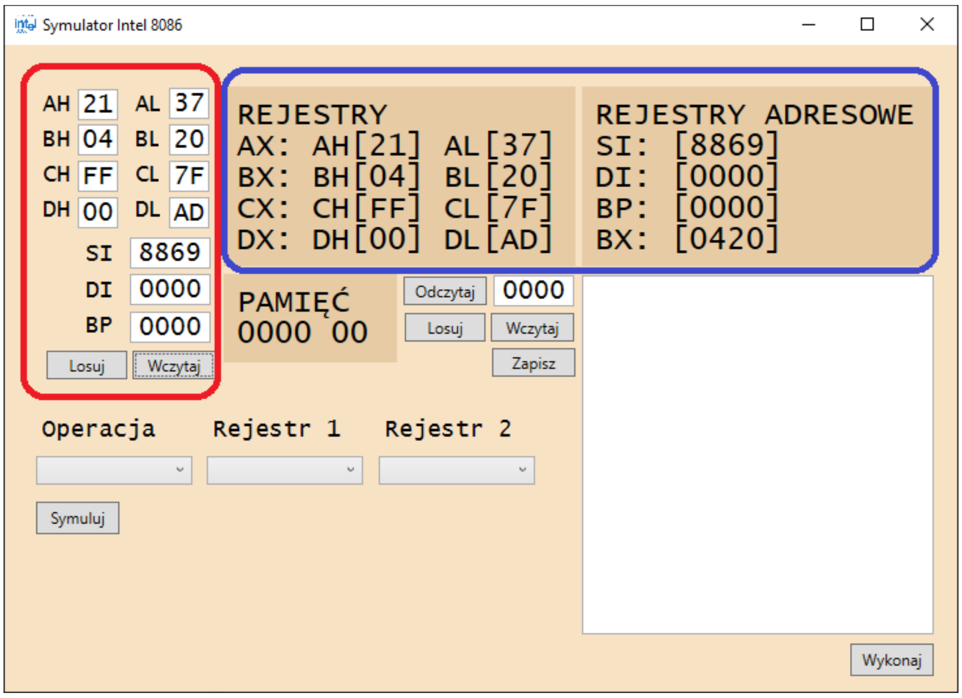
```
REJESTRY
AX:  AH[00]  AL[00]
BX:  BH[00]  BL[00]
CX:  CH[00]  CL[00]
DX:  DH[00]  DL[00]
```

```
REJESTRY ADRESOWE
SI:  [0000]
DI:  [0000]
BP:  [0000]
BX:  [0000]
```

```
PAMIĘĆ
0000 00
```

Aplikacja po uruchomieniu

Dane do rejestrów procesora można wprowadzić za pomocą podpisanych pól lub wylosować



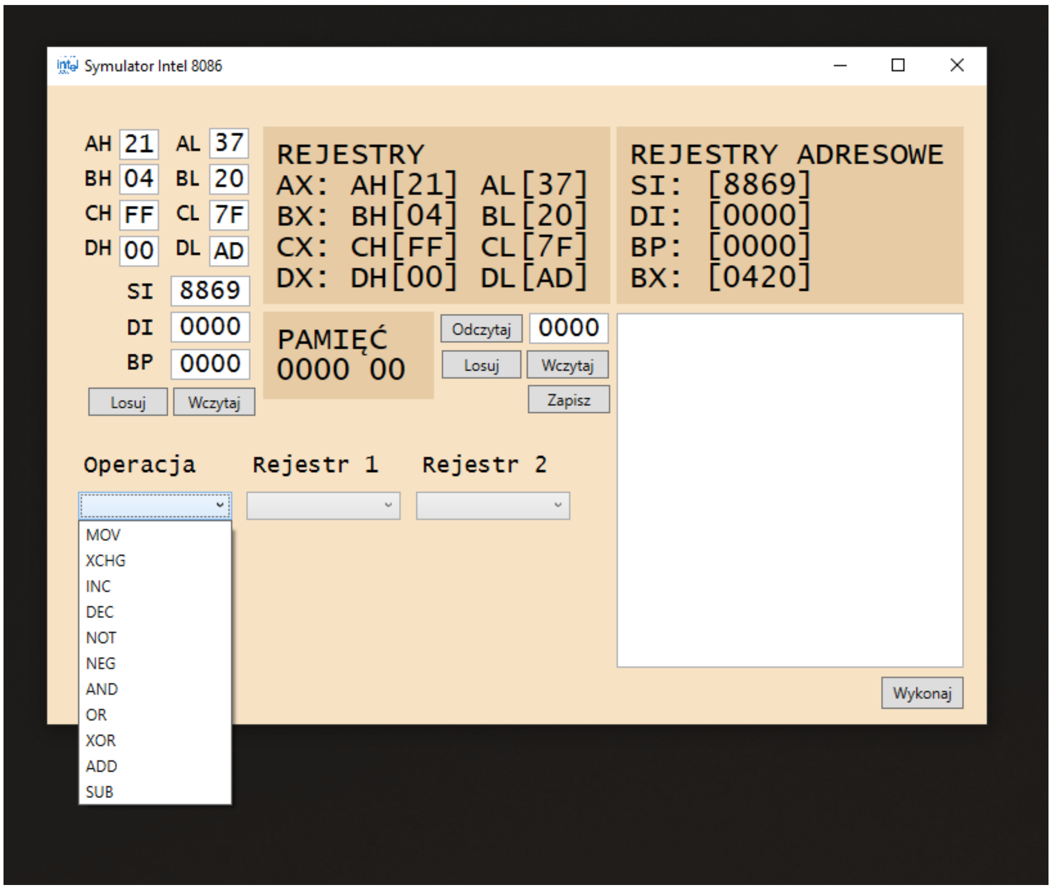
```
REJESTRY
AX:  AH[21]  AL[37]
BX:  BH[04]  BL[20]
CX:  CH[FF]  CL[7F]
DX:  DH[00]  DL[AD]
```

```
REJESTRY ADRESOWE
SI:  [8869]
DI:  [0000]
BP:  [0000]
BX:  [0420]
```

```
PAMIĘĆ
0000 00
```

Obszar zaznaczony na czerwono jest odpowiedzialny za wprowadzanie danych, a obszar zaznaczony na niebiesko wyświetla zawartość Rejestrów

Operacje na rejestrach można wykonywać za pomocą formularza


Symulator Intel 8086

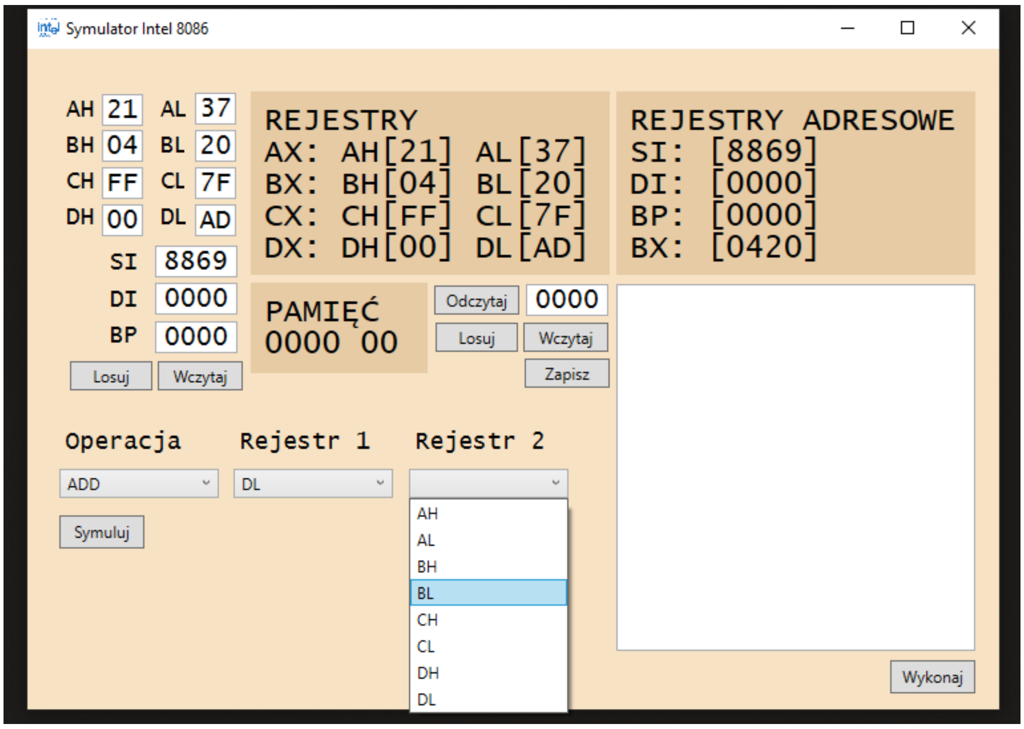
AH	21	AL	37
BH	04	BL	20
CH	FF	CL	7F
DH	00	DL	AD

SI 8869
DI 0000
BP 0000

```
REJESTRY
AX: AH[21] AL[37]
BX: BH[04] BL[20]
CX: CH[FF] CL[7F]
DX: DH[00] DL[AD]
```

```
REJESTRY ADRESOWE
SI: [8869]
DI: [0000]
BP: [0000]
BX: [0420]
```

```
PAMIĘĆ
0000 00
```


Symulator Intel 8086

AH	21	AL	37
BH	04	BL	20
CH	FF	CL	7F
DH	00	DL	AD

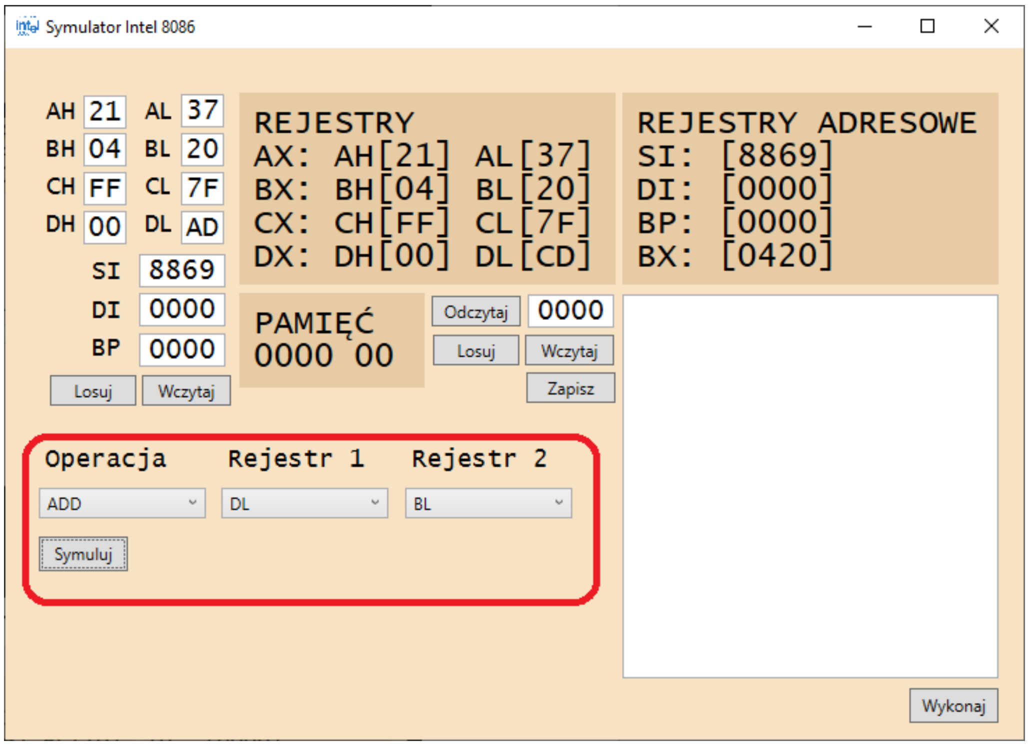
SI 8869
DI 0000
BP 0000

```
REJESTRY
AX: AH[21] AL[37]
BX: BH[04] BL[20]
CX: CH[FF] CL[7F]
DX: DH[00] DL[AD]
```

```
REJESTRY ADRESOWE
SI: [8869]
DI: [0000]
BP: [0000]
BX: [0420]
```

```
PAMIĘĆ
0000 00
```

Operacja: ADD
Rejestr 1: DL



```
AH 21   AL 37
BH 04   BL 20
CH FF   CL 7F
DH 00   DL AD
    SI 8869
    DI 0000
    BP 0000

REJESTRY
AX: AH[21] AL[37]
BX: BH[04] BL[20]
CX: CH[FF] CL[7F]
DX: DH[00] DL[CD]

REJESTRY ADRESOWE
SI: [8869]
DI: [0000]
BP: [0000]
BX: [0420]

PAMIĘĆ
0000 00          0000

Operacja    Rejestr 1    Rejestr 2
ADD         DL           BL
```

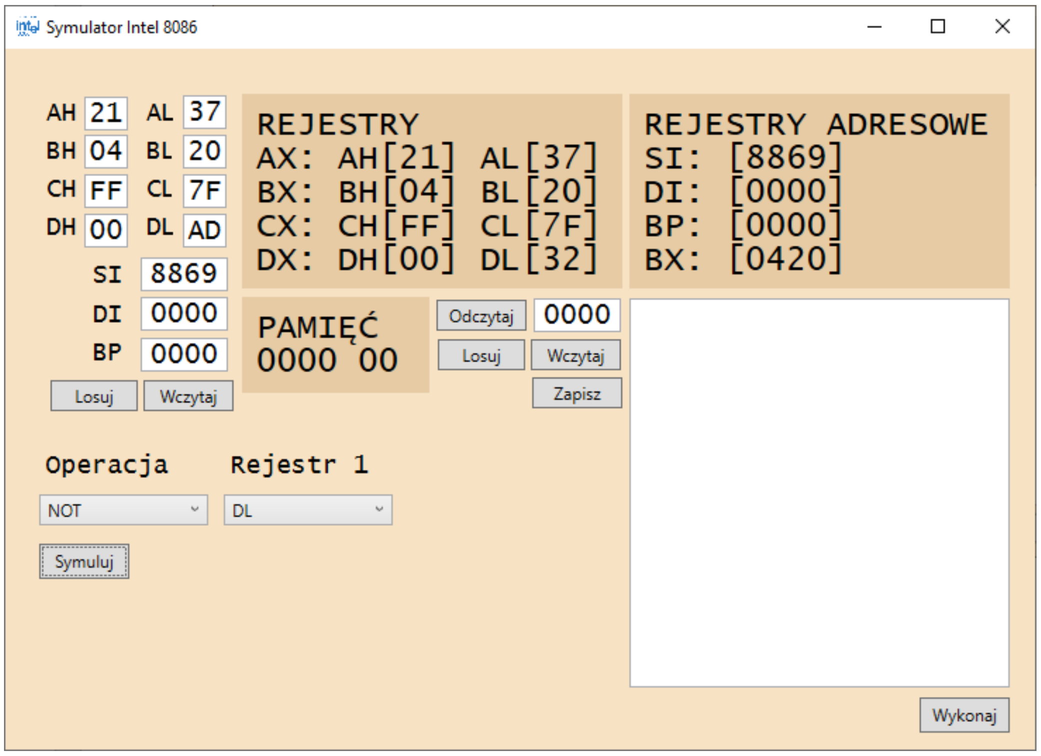
Zaznaczony na czerwono obszar jest odpowiedzialny za operacje na rejestrach, w tym przypadku wykonana została operacja

ADD DL,BL

Co spowodowało zwiększenie wartości rejestru DL o wartość rejestru BL zmieniając AD w CD

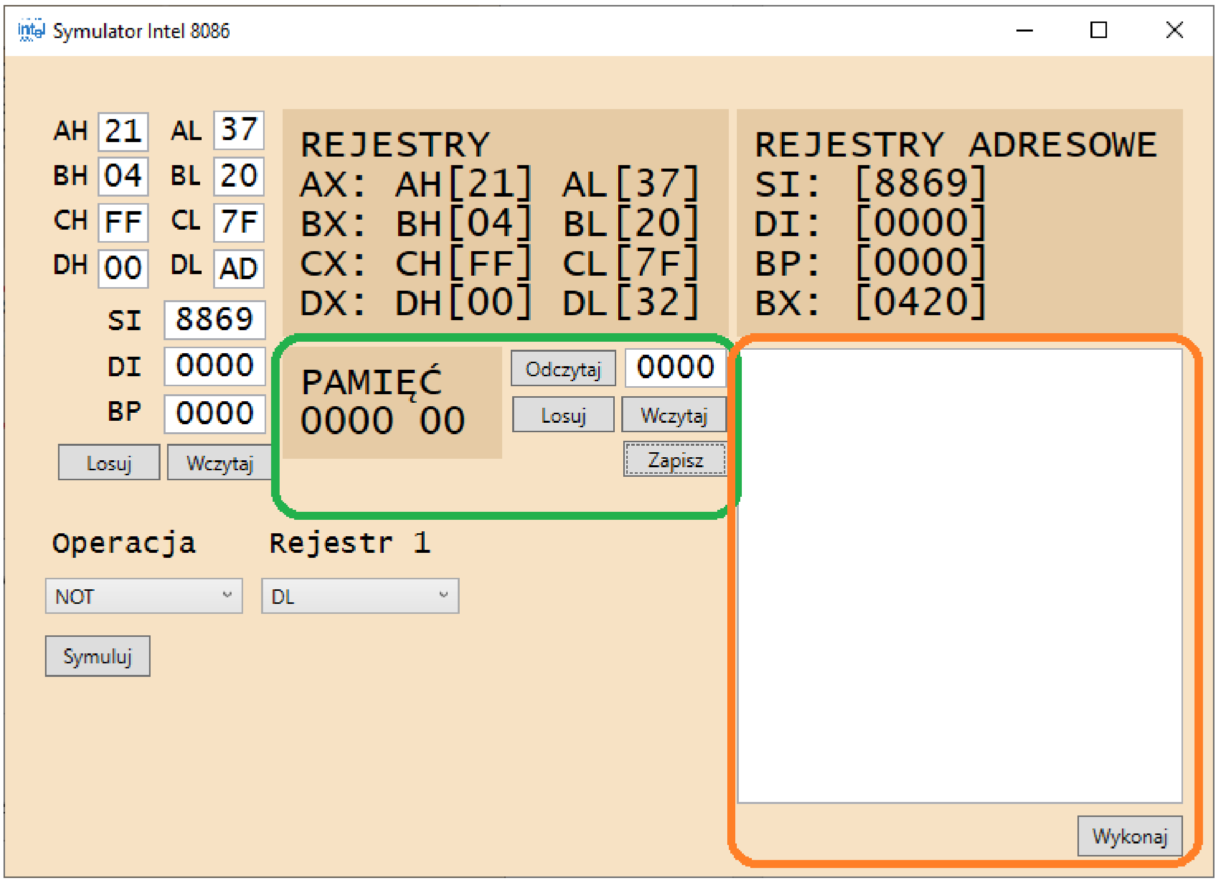
AD+20=CD

W przypadku operacji na pojedynczym rejestrze opcja wyboru drugiego rejestru jest ukrywana



```
AH 21   AL 37
BH 04   BL 20
CH FF   CL 7F
DH 00   DL AD
    SI 8869
    DI 0000
    BP 0000

REJESTRY
AX: AH[21] AL[37]
BX: BH[04] BL[20]
CX: CH[FF] CL[7F]
DX: DH[00] DL[32]

REJESTRY ADRESOWE
SI: [8869]
DI: [0000]
BP: [0000]
BX: [0420]

PAMIĘĆ
0000 00          0000

Operacja    Rejestr 1
NOT         DL
```

Ostatnie 2 elementy interfejsu to:
zaznaczony na zielono obszar do wczytywania/odczytywania/zapisu pamięci
zaznaczony na pomarańczowo obszar to interpreter assemblera umożliwiający zarówno operacje na rejestrach jak i pamięci



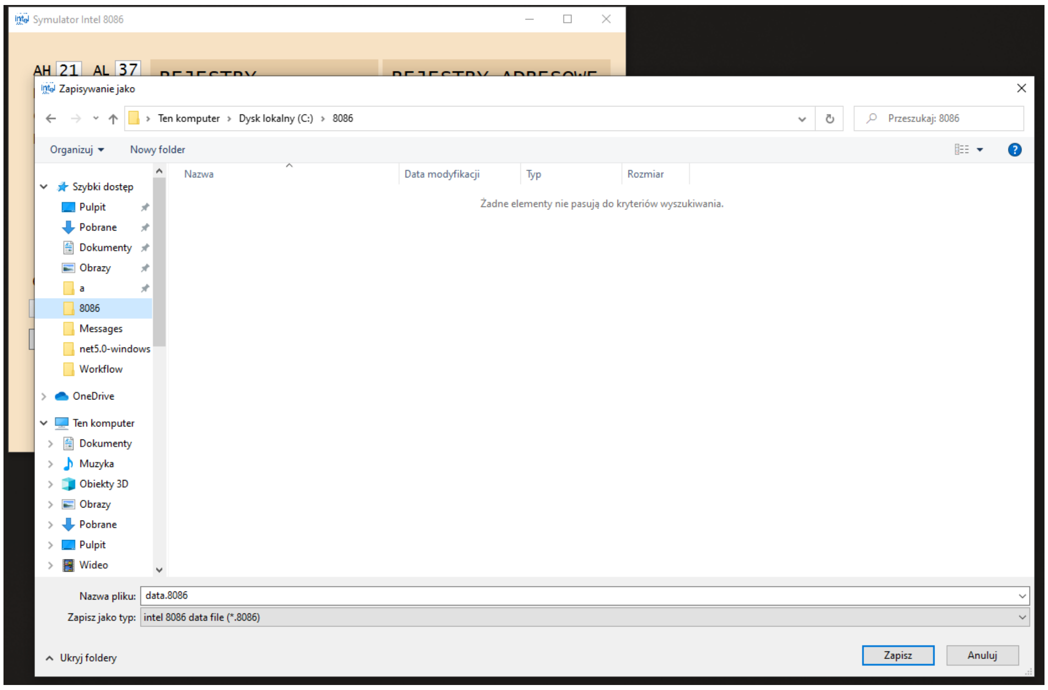
Przycisk odczytaj pozwala na szybkie sprawdzenie zawartości konkretnej komórki

Przycisk losuj wypełnia pamięć losowymi wartościami

Przycisk wczytaj pozwala załadować do pamięci dane z pliku o rozszerzeniu .8086

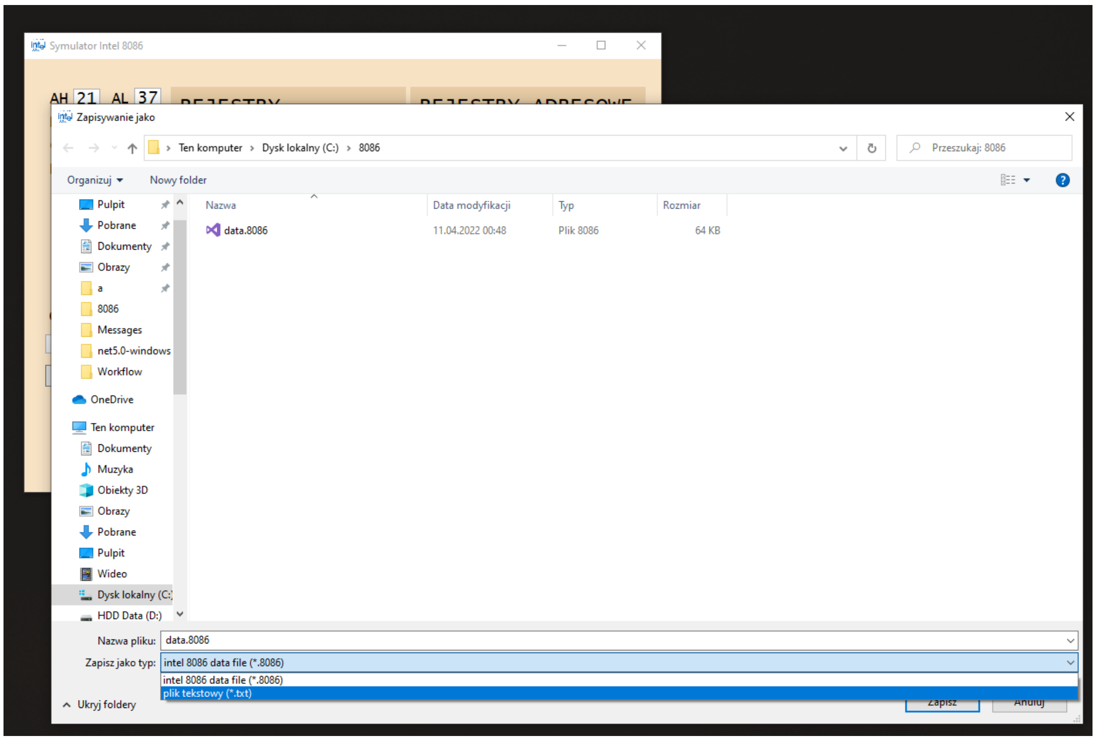
Przycisk zapisz pozwala przenieść pamięć do pliku o rozszerzeniu .8086 umożliwiając późniejsze wczytanie danych oraz do pliku .txt

Zapis pamięci do pliku



Żadne elementy nie pasują do kryteriów wyszukiwania.

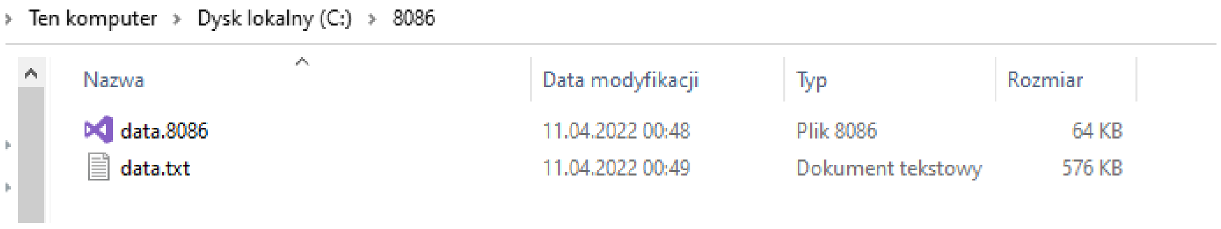
Nazwa pliku: data.8086
Zapisz jako typ: intel 8086 data file (*.8086)



Nazwa	Data modyfikacji	Typ	Rozmiar
data.8086	11.04.2022 00:48	Plik 8086	64 KB

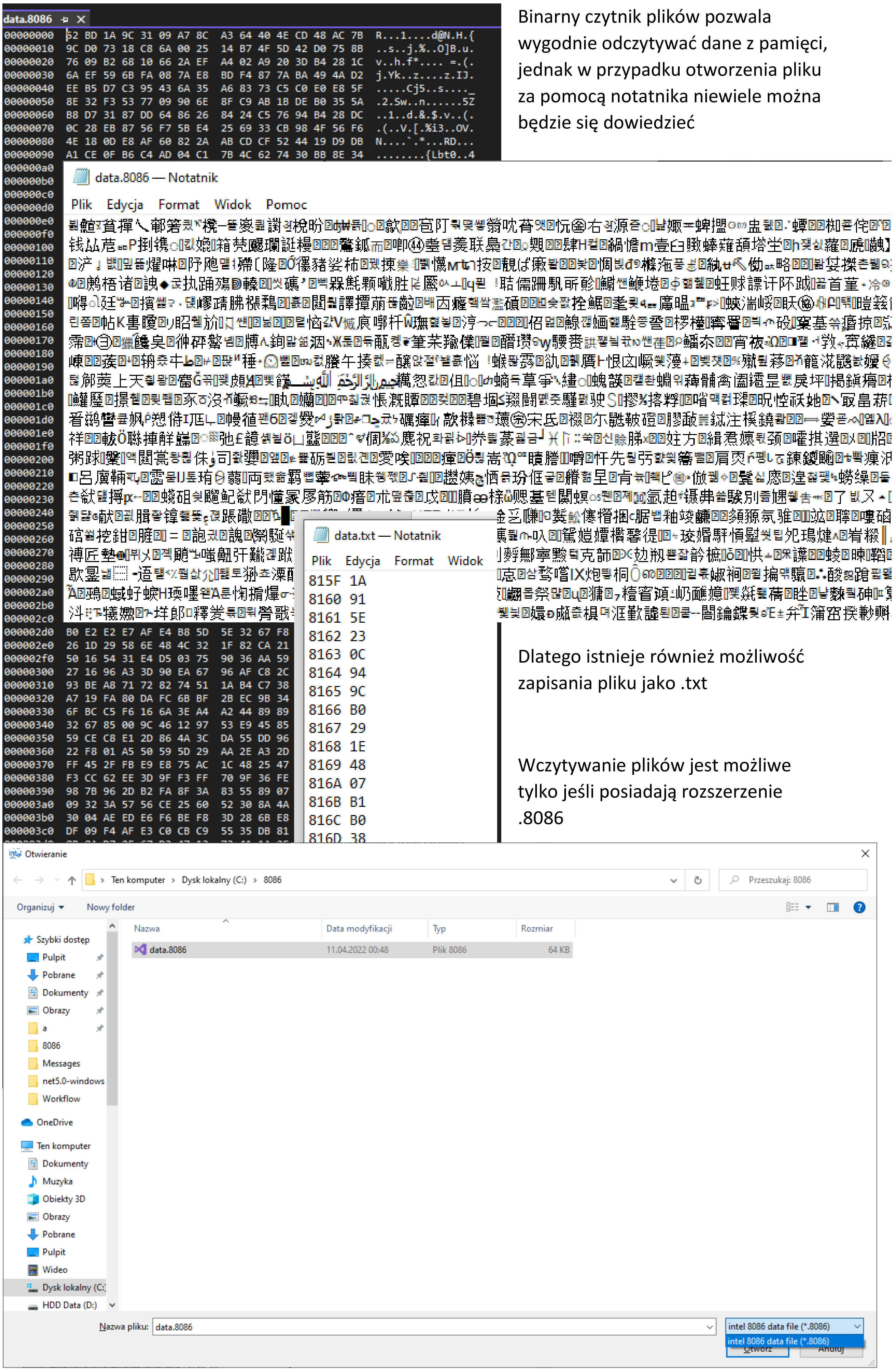
Nazwa pliku: data.8086
Zapisz jako typ: intel 8086 data file (*.8086)
intel 8086 data file (*.8086)
plik tekstowy (*.txt)

Plik o rozszerzeniu .8086 to binarny zapis pamięci jednak da się odczytać jego dane odpowiednimi programami, plik w rozszerzeniu txt ma wypisane w kolumnie wszystkie komórki pamięci wraz z ich zawartością


› Ten komputer › Dysk lokalny (C:) › 8086

Nazwa	Data modyfikacji	Typ	Rozmiar
data.8086	11.04.2022 00:48	Plik 8086	64 KB
data.txt	11.04.2022 00:49	Dokument tekstowy	576 KB

Binarny czytnik plików pozwala wygodnie odczytywać dane z pamięci, jednak w przypadku otworzenia pliku za pomocą notatnika niewiele można będzie się dowiedzieć



```
data.8086
00000000  52 BD 1A 9C 31 09 A7 8C  A3 64 40 4E CD 48 AC 7B  R...1....d@N.H.{
00000010  9C D0 73 18 C8 6A 00 25  14 B7 4F 5D 42 D0 75 8B  ..s..j.%..O]B.u.
00000020  76 09 B2 68 10 66 2A EF  A4 02 A9 20 3D B4 28 1C  v..h.f*.... =.(.
00000030  6A EF 59 6B FA 08 7A E8  BD F4 87 7A BA 49 4A D2  j.Yk..z....z.IJ.
00000040  EE B5 D7 C3 95 43 6A 35  A6 83 73 C5 C0 E0 E8 5F  .....Cj5..s...._
00000050  8E 32 F3 53 77 09 90 6E  8F C9 AB 1B DE B0 35 5A  .2.Sw..n......5Z
00000060  B8 D7 31 87 DD 64 86 26  84 24 C5 76 94 B4 28 DC  ..1..d.&.$.v..(.
00000070  0C 28 EB 87 56 F7 5B E4  25 69 33 CB 98 4F 56 F6  .(..V.[.%i3..OV.
00000080  4E 18 0D E8 AF 60 82 2A  AB CD CF 52 44 19 D9 DB  N....`.*...RD...
00000090  A1 CE 0F B6 C4 AD 04 C1  7B 4C 62 74 30 BB 8E 34  ........{Lbt0..4
...
000002d0  B0 E2 E2 E7 AF E4 B8 5D  5E 32 67 F8
000002e0  26 1D 29 58 6E 48 4C 32  1F 82 CA 21
000002f0  50 16 54 31 E4 D5 03 75  90 36 AA 59
00000300  27 16 96 A3 3D 90 EA 67  96 AF C8 2C
00000310  93 BE A8 71 72 82 74 51  1A B4 C7 38
00000320  A7 19 FA 80 DA FC 6B BF  2B EC 9B 34
00000330  6F BC C5 F6 16 6A 3E A4  A2 44 89 89
00000340  32 67 85 00 9C 46 12 97  53 E9 45 85
00000350  59 CE C8 E1 2D 86 4A 3C  DA 55 DD 96
00000360  22 F8 01 A5 50 59 5D 29  AA 2E A3 2D
00000370  FF 45 2F FB E9 E8 75 AC  1C 48 25 47
00000380  F3 CC 62 EE 3D 9F F3 FF  70 9F 36 FE
00000390  98 7B 96 2D B2 FA 8F 3A  83 55 89 07
000003a0  09 32 3A 57 56 CE 25 60  52 30 8A 4A
000003b0  30 04 AE ED E6 F6 BE F8  3D 28 6B E8
000003c0  DF 09 F4 AF E3 C0 CB C9  55 35 DB 81
```

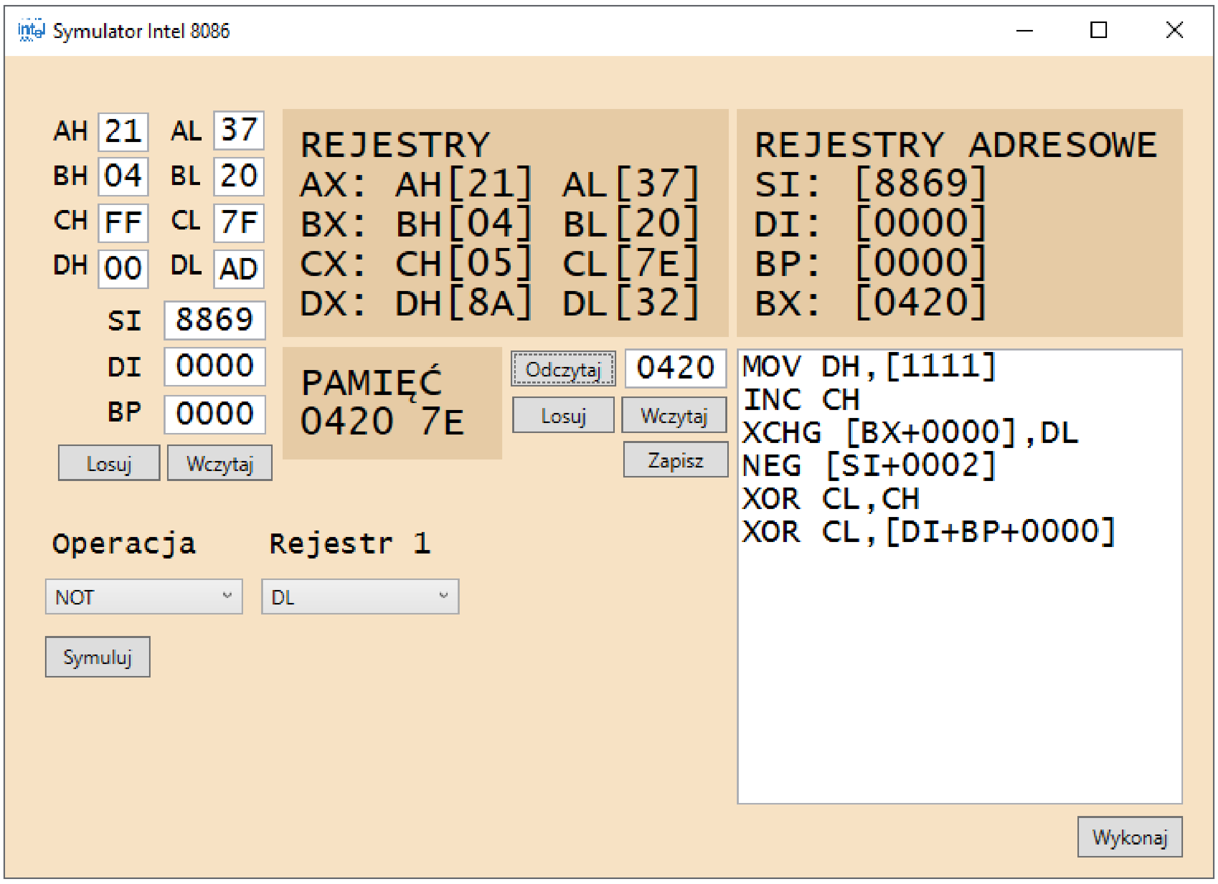
```
815F 1A
8160 91
8161 5E
8162 23
8163 0C
8164 94
8165 9C
8166 B0
8167 29
8168 1E
8169 48
816A 07
816B B1
816C B0
816D 38
```

Dlatego istnieje również możliwość zapisania pliku jako .txt

Wczytywanie plików jest możliwe tylko jeśli posiadają rozszerzenie .8086

Nazwa	Data modyfikacji	Typ	Rozmiar
data.8086	11.04.2022 00:48	Plik 8086	64 KB

Nazwa pliku: data.8086
intel 8086 data file (*.8086)
intel 8086 data file (*.8086)



Symulator Intel 8086

```
AH 21  AL 37
BH 04  BL 20
CH FF  CL 7F
DH 00  DL AD
   SI 8869
   DI 0000
   BP 0000
```

```
REJESTRY
AX: AH[21] AL[37]
BX: BH[04] BL[20]
CX: CH[05] CL[7E]
DX: DH[8A] DL[32]
```

```
REJESTRY ADRESOWE
SI: [8869]
DI: [0000]
BP: [0000]
BX: [0420]
```

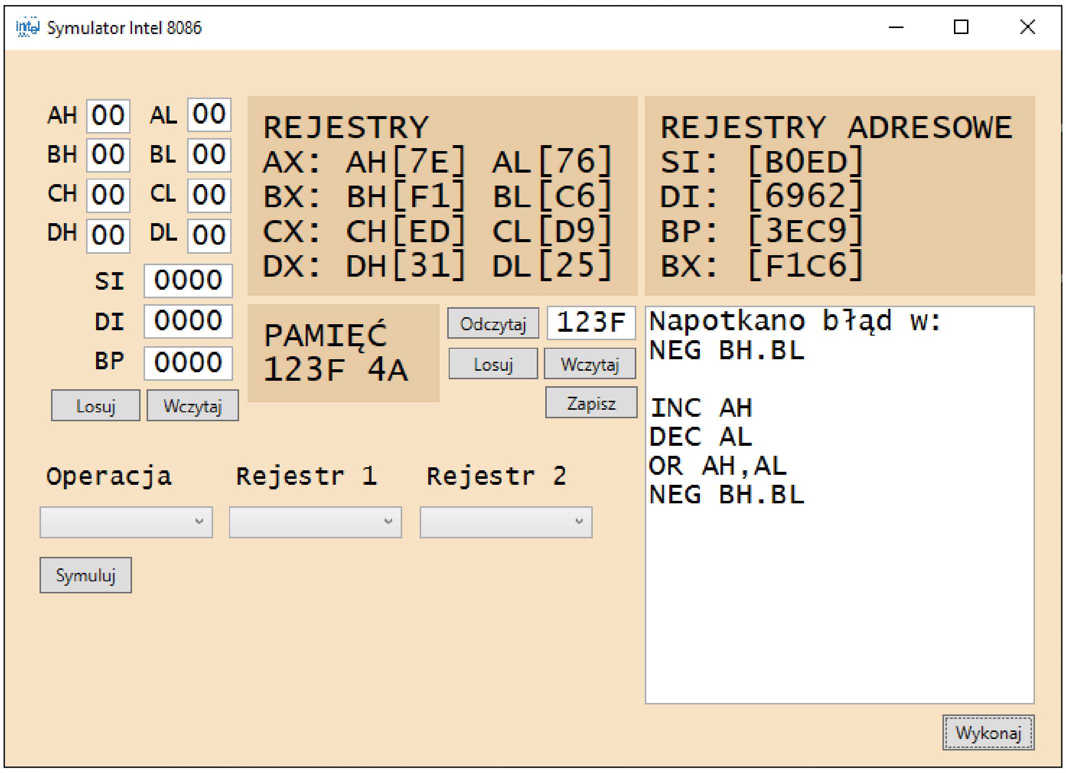
```
PAMIĘĆ
0420 7E
```

Odczytaj: 0420

Operacja: NOT
Rejestr 1: DL

```
MOV DH,[1111]
INC CH
XCHG [BX+0000],DL
NEG [SI+0002]
XOR CL,CH
XOR CL,[DI+BP+0000]
```

Interpreter pozwala na wykonywanie wielu operacji jedna po drugiej a w przypadku błędnej składni informuje gdzie napotkał błąd



Symulator Intel 8086

```
AH 00  AL 00
BH 00  BL 00
CH 00  CL 00
DH 00  DL 00
   SI 0000
   DI 0000
   BP 0000
```

```
REJESTRY
AX: AH[7E] AL[76]
BX: BH[F1] BL[C6]
CX: CH[ED] CL[D9]
DX: DH[31] DL[25]
```

```
REJESTRY ADRESOWE
SI: [B0ED]
DI: [6962]
BP: [3EC9]
BX: [F1C6]
```

```
PAMIĘĆ
123F 4A
```

Odczytaj: 123F

Operacja / Rejestr 1 / Rejestr 2

```
Napotkano błąd w:
NEG BH.BL

INC AH
DEC AL
OR AH,AL
NEG BH.BL
```

Kod Źródłowy

Całość rozwiązania można znaleźć na repozytorium na githubie
https://github.com/Estremo102/8086

Aplikację można pobrać pod linkiem:
https://github.com/Estremo102/8086/blob/master/SymulatorIntel8086/SymulatorIntel8086.rar?raw=true

W rozwiązaniu można znaleźć 3 projekty:
8086 – pierwszy prototyp aplikacji konsolowej
Intel8086 – Biblioteka posiadająca dwie klasy Procesor i Memory odpowiedzialne za Logikę programu
SymulatorIntel8086 – Aplikacja WPF korzystająca z Biblioteki Intel8086

Klasa Procesor:

using System;

```
using System;

namespace Intel8086
{
    public class Procesor
    {
        private byte[] register = new byte[8];
        private ushort[] addressRegister = new ushort[3];
        public string AH { get => ToHex(register[0]); private set => register[0]
= (byte)ToDecimal(value); }
        public string AL { get => ToHex(register[1]); private set => register[1]
= (byte)ToDecimal(value); }
        public string BH { get => ToHex(register[2]); private set => register[2]
= (byte)ToDecimal(value); }
        public string BL { get => ToHex(register[3]); private set => register[3]
= (byte)ToDecimal(value); }
        public string CH { get => ToHex(register[4]); private set => register[4]
= (byte)ToDecimal(value); }
        public string CL { get => ToHex(register[5]); private set => register[5]
= (byte)ToDecimal(value); }
        public string DH { get => ToHex(register[6]); private set => register[6]
= (byte)ToDecimal(value); }
        public string DL { get => ToHex(register[7]); private set => register[7]
= (byte)ToDecimal(value); }
        public string SI { get => ToHex(addressRegister[0]); private set =>
addressRegister[0] = (ushort)ToDecimal(value); }
        public string DI { get => ToHex(addressRegister[1]); private set =>
addressRegister[1] = (ushort)ToDecimal(value); }
        public string BP { get => ToHex(addressRegister[2]); private set =>
addressRegister[2] = (ushort)ToDecimal(value); }
        public string BX { get => ToHex(register[2]) + ToHex(register[3]); }

        public Memory memory { get; set; } = new Memory();
        delegate void Operation(int a, int b);
        delegate void OperationDR(string a, int b);
        delegate void OperationRD(int a, string b);
        delegate void OperationSR(int a);
        delegate void OperationSRD(string a);

        public Procesor() { }

        public Procesor(int seed)
        {
            Random random = new Random(seed);
            for (int i = 0; i < register.Length; i++)
                register[i] = (byte)random.Next(256);
            for (int i = 0; i < addressRegister.Length; i++)
                addressRegister[i] = (ushort)random.Next(65536);
        }

         public Procesor(params string[] registers)
        {
            if (registers.Length != 8 && registers.Length != 11) throw new
ArgumentException();
            foreach (var register in registers) if (!CheckData(register)) throw
new ArgumentException();
            AH = registers[0];
            AL = registers[1];
            BH = registers[2];
            BL = registers[3];
            CH = registers[4];
            CL = registers[5];
            DH = registers[6];
            DL = registers[7];
            if (registers.Length == 11)
            {
                SI = registers[8];
                DI = registers[9];
                BP = registers[10];
            }
        }

        public static string ToHex(byte x) => x.ToString("x2").ToUpper();
        public static string ToHex(ushort x) => x.ToString("x4").ToUpper();

        public static int ToDecimal(string x) => Convert.ToInt32(x, 16);

        public static bool CheckData(string data)
        {
            try
            {
                Convert.ToInt32(data, 16);
                return true;
            }
            catch
            {
                return false;
            }
        }

        public static bool CheckData(string data, int length)
        {
            if (data.Length != length) return false;
            return CheckData(data);
        }
        static bool CheckRegister(string check)
        {
            if (check.Length != 2) return false;
            if (check[0] >= 65 && check[0] <= 68 && (check[1] == 'H' || check[1]
== 'L'))
                return true;
            return false;
        }
        public bool ExecuteOperation(string input)
        {
            string[] a;
            a = input.ToUpper().Split(' ');
            Operation o = null;
            OperationDR odr = null;
            OperationRD ord = null;
            OperationSR osr = null;
            OperationSRD osrD = null;
            switch (a[0])
            {
                case "MOV":
                    o = MOV;
                    odr = MOV;
                    ord = MOV;
                    break;
                case "XCHG":
                    o = XCHG;
                    ord = XCHG;
                    odr = XCHG;
                    break;
                case "INC":
                    osr = INC;
                    osrD = INC;
                    break;
                case "DEC":
                    osr = DEC;
                    osrD = DEC;
                    break;
                case "NOT":
                    osr = NOT;
                    osrD = NOT;
                    break;
                case "NEG":
                    osr = NOT;
                    osr += INC;
                    osrD = NOT;
                    osrD += INC;
                    break;
                case "AND":
                    o = AND;
                    ord = AND;
                    odr = AND;
                    break;
                case "OR":
                    o = OR;
                    ord = OR;
                    odr = OR;
                    break;
                case "XOR":
                    o = XOR;
                    ord = XOR;
                    odr = XOR;
                    break;
                case "ADD":
                    o = ADD;
                    ord = ADD;
                    odr = ADD;
                    break;
                case "SUB":
                    o = SUB;
                    ord = SUB;
                    odr = SUB;
                    break;
                default:
                    return false;
            }
            if (o == null)
            {
                a = a[1].Split(',');
                if (!CheckRegister(a[0]))
                {
                    if (!Memory.CheckAddress(a[0])) return false;
                    osrD(a[0]);
                    return true;
                }
                osr(RegisterToInt(a[0]));
                return true;
            }
            else
            {
                a = a[1].Split(',');
                if(a.Length != 2) return false;
                if (!CheckRegister(a[0]) || !CheckRegister(a[1]))
                {
                    if (CheckRegister(a[0]) && Memory.CheckAddress(a[1]))
                    {
                        ord(RegisterToInt(a[0]), a[1]);
                        return true;
                    }
                    if (Memory.CheckAddress(a[0]) && CheckRegister(a[1]))
                    {
                        odr(a[0], RegisterToInt(a[1]));
                        return true;
                    }
                    return false;
                }
                o(RegisterToInt(a[0]), RegisterToInt(a[1]));
                return true;
            }
        }

        void MOV(int a, int b) => register[a] = register[b];
        void XCHG(int a, int b)
        {
            byte temp = register[a];
            register[a] = register[b];
            register[b] = temp;
        }
        void INC(int a) => register[a]++;
        void DEC(int a) => register[a]--;
        void NOT(int a) => register[a] = (byte)~register[a];
        void AND(int a, int b) => register[a] = (byte)(register[a] &
register[b]);
        void OR(int a, int b) => register[a] = (byte)(register[a] | register[b]);
        void XOR(int a, int b) => register[a] = (byte)(register[a] ^
register[b]);
```

```
        void ADD(int a, int b) => register[a] += register[b];
        void SUB(int a, int b) => register[a] -= register[b];


        void INC(string a) => memory.data[Memory.StringToAddress(a, this)]++;
        void DEC(string a) => memory.data[Memory.StringToAddress(a, this)]--;
        void NOT(string a) => memory.data[Memory.StringToAddress(a, this)] =
(byte)~memory.data[Memory.StringToAddress(a, this)];
        void MOV(int a, string b) => register[a] =
memory.data[Memory.StringToAddress(b, this)];
        void XCHG(int a, string b)
        {
            byte temp = register[a];
            register[a] = memory.data[Memory.StringToAddress(b, this)];
            memory.data[Memory.StringToAddress(b, this)] = temp;
        }
        void AND(int a, string b) => register[a] = (byte)(register[a] &
memory.data[Memory.StringToAddress(b, this)]);
        void OR(int a, string b) => register[a] = (byte)(register[a] |
memory.data[Memory.StringToAddress(b, this)]);
        void XOR(int a, string b) => register[a] = (byte)(register[a] ^
memory.data[Memory.StringToAddress(b, this)]);
        void ADD(int a, string b) => register[a] +=
memory.data[Memory.StringToAddress(b, this)];
        void SUB(int a, string b) => register[a] -=
memory.data[Memory.StringToAddress(b, this)];
        void MOV(string a, int b) => memory.data[Memory.StringToAddress(a, this)]
= register[b];
        void XCHG(string a, int b)
        {
            byte temp = memory.data[Memory.StringToAddress(a, this)];
            memory.data[Memory.StringToAddress(a, this)] = register[b];
            register[b] = temp;
        }
        void AND(string a, int b) => memory.data[Memory.StringToAddress(a, this)]
= Convert.ToByte(memory.data[Memory.StringToAddress(a, this)] & register[b]);
        void OR(string a, int b) => memory.data[Memory.StringToAddress(a, this)]
= Convert.ToByte(memory.data[Memory.StringToAddress(a, this)] | register[b]);
        void XOR(string a, int b) => memory.data[Memory.StringToAddress(a, this)]
= Convert.ToByte(memory.data[Memory.StringToAddress(a, this)] ^ register[b]);
        void ADD(string a, int b) => memory.data[Memory.StringToAddress(a, this)]
+= register[b];
        void SUB(string a, int b) => memory.data[Memory.StringToAddress(a, this)]
-= register[b];

        static int RegisterToInt(string r) =>
            r switch
            {
                "AH" => 0,
                "AL" => 1,
                "BH" => 2,
                "BL" => 3,
                "CH" => 4,
                "CL" => 5,
                "DH" => 6,
                "DL" => 7,
                _ => -1,
            };

        public string AddressRegisters() => $"SI: [{SI}]\n" +
                                            $"DI: [{DI}]\n" +
                                            $"BP: [{BP}]\n" +
                                            $"BX: [{BX}]";
        public override string ToString() => $"AX: AH[{AH,2}] AL[{AL,2}]\n" +
                                             $"BX: BH[{BH,2}] BL[{BL,2}]\n" +
                                             $"CX: CH[{CH,2}] CL[{CL,2}]\n" +
                                             $"DX: DH[{DH,2}] DL[{DL,2}]";
    }
}
```

KLASA MEMORY

```
using System;
using System.Text;
using System.IO;

namespace Intel8086
{
    public class Memory
    {
        public byte[] data = new byte[65536];
        public Memory(int seed)
        {
            Random random = new Random(seed);
            for (int i = 0; i < data.Length; i++)
                data[i] = (byte)random.Next(256);
        }
        public Memory() { }

        public static bool CheckAddress(string address)
        {
            if (address[0] != '[' || address[address.Length - 1] != ']') return
false;
            address = address.Substring(1, address.Length - 2);
            string[] tmp = address.Split('+');
            if (tmp.Length > 3) return false;
            bool a = false;
            bool b = false;
            bool c = false;
            foreach (string s in tmp)
            {
                try
                {
                    if (s.Length == 4 && !a)
                    {
                        Convert.ToInt32(s, 16);
                        a = true;
                    }
                    else throw new Exception();
                }
                catch
                {
                    if ((s == "SI" || s == "DI") && !c)
                    {
                        c = true;
                        continue;
                    }
                    if ((s == "BP" || s == "BX") && !b)
                    {
                        b = true;
                        continue;
                    }
                    return false;
                }
            }
            return true;
        }

        public static int StringToAddress(string address, Procesor p)
        {
            address = address.Substring(1, address.Length - 2);
            string[] tmp = address.Split('+');
            ushort adr = 0;
            foreach (string s in tmp)
            {
                switch (s)
                {
                    case "SI":
                        adr += (ushort)Procesor.ToDecimal(p.SI);
                        break;
                    case "DI":
                        adr += (ushort)Procesor.ToDecimal(p.DI);
                        break;
                    case "BP":
                        adr += (ushort)Procesor.ToDecimal(p.BP);
                        break;
                    case "BX":
                        adr += (ushort)Procesor.ToDecimal(p.BX);
                        break;
                    default:
                        adr += (ushort)Procesor.ToDecimal(s);
                        break;
                }
            }
            return adr;
        }

        public void Save()
        {
            Save("data.8086");
        }
        public void Save(string fileName)
        {
            string[] tmp = fileName.Split('.');
            if(tmp[tmp.Length - 1] == "txt")
            {
                using (StreamWriter sw = new StreamWriter(fileName))
                {
                    sw.Write(this.ToString());
                }
                return;
            }
            using (BinaryWriter w = new BinaryWriter(File.Create(fileName)))
            {
                w.Write(data);
            }
        }

        public void Load()
        {
            Load("data.8086");
        }
        public void Load(string fileName)
        {
            using (BinaryReader r = new BinaryReader(File.OpenRead(fileName)))
            {
                data = r.ReadBytes(data.Length);
```

```
            }
        }

        public string DisplayData(string s) => (Convert.ToInt32(s,
16).ToString("x4") + " " + data[Convert.ToInt32(s,
16)].ToString("x2")).ToUpper();

        public override string ToString()
        {
            StringBuilder sb = new StringBuilder();
            for (int i = 0; i < data.Length; i++)
                sb.AppendLine($"{i.ToString("x4").ToUpper()}
{data[i].ToString("x2").ToUpper()}");
            return sb.ToString();
        }
    }
}
```

MainWindow.xaml

Czyli kod odpowiadający za wygląd interfejsu

```
<Window x:Class="SymulatorIntel8086.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:SymulatorIntel8086"
        mc:Ignorable="d"
        Title="Symulator Intel 8086" Height="535" Width="747"
Background="#FFF7E2C4" MinWidth="740" MinHeight="530">
    <Grid>
        <Label Content="AH" HorizontalAlignment="Left" Margin="24,32,0,0"
VerticalAlignment="Top" Height="27" Width="33" FontFamily="Lucida Console"
FontSize="18"/>
        <Label Content="AL" HorizontalAlignment="Left" Margin="95,32,0,0"
VerticalAlignment="Top" Height="27" Width="29" FontFamily="Lucida Console"
FontSize="18"/>
        <Label Content="BH" HorizontalAlignment="Left" Margin="24,59,0,0"
VerticalAlignment="Top" Height="27" Width="31" FontFamily="Lucida Console"
FontSize="18"/>
        <Label Content="BL" HorizontalAlignment="Left" Margin="95,59,0,0"
VerticalAlignment="Top" Height="27" Width="28" FontFamily="Lucida Console"
FontSize="18"/>
        <Label Content="CH" HorizontalAlignment="Left" Margin="24,86,0,0"
VerticalAlignment="Top" Height="27" Width="32" FontFamily="Lucida Console"
FontSize="18"/>
        <Label Content="CL" HorizontalAlignment="Left" Margin="95,86,0,0"
VerticalAlignment="Top" Height="27" Width="28" FontFamily="Lucida Console"
FontSize="18"/>
        <Label Content="DH" HorizontalAlignment="Left" Margin="24,113,0,0"
VerticalAlignment="Top" Height="27" Width="33" FontFamily="Lucida Console"
FontSize="18"/>
        <Label Content="DL" HorizontalAlignment="Left" Margin="95,113,0,0"
VerticalAlignment="Top" Height="27" Width="29" FontFamily="Lucida Console"
FontSize="18"/>
        <TextBox x:Name="AH" HorizontalAlignment="Left" Margin="56,34,0,0"
Text="00" TextWrapping="Wrap" VerticalAlignment="Top" Width="31" Height="24"
FontFamily="Lucida Console" FontSize="20" FontWeight="Normal"
HorizontalContentAlignment="Center" VerticalContentAlignment="Center"
MaxLength="2"/>
```

```
        <TextBox x:Name="AL" HorizontalAlignment="Left" Margin="126,33,0,0"
Text="00" TextWrapping="Wrap" VerticalAlignment="Top" Width="31" Height="24"
FontFamily="Lucida Console" FontSize="20" FontWeight="Normal"
HorizontalContentAlignment="Center" VerticalContentAlignment="Center"
MaxLength="2"/>
        <TextBox x:Name="BH" HorizontalAlignment="Left" Margin="56,61,0,0"
Text="00" TextWrapping="Wrap" VerticalAlignment="Top" Width="31" Height="24"
FontFamily="Lucida Console" FontSize="20" FontWeight="Normal"
HorizontalContentAlignment="Center" VerticalContentAlignment="Center"
MaxLength="2"/>
        <TextBox x:Name="BL" HorizontalAlignment="Left" Margin="126,61,0,0"
Text="00" TextWrapping="Wrap" VerticalAlignment="Top" Width="31" Height="24"
FontFamily="Lucida Console" FontSize="20" FontWeight="Normal"
HorizontalContentAlignment="Center" VerticalContentAlignment="Center"
MaxLength="2"/>
        <TextBox x:Name="CH" HorizontalAlignment="Left" Margin="56,89,0,0"
Text="00" TextWrapping="Wrap" VerticalAlignment="Top" Width="31" Height="24"
FontFamily="Lucida Console" FontSize="20" FontWeight="Normal"
HorizontalContentAlignment="Center" VerticalContentAlignment="Center"
MaxLength="2"/>
        <TextBox x:Name="CL" HorizontalAlignment="Left" Margin="126,89,0,0"
Text="00" TextWrapping="Wrap" VerticalAlignment="Top" Width="31" Height="24"
FontFamily="Lucida Console" FontSize="20" FontWeight="Normal"
HorizontalContentAlignment="Center" VerticalContentAlignment="Center"
MaxLength="2"/>
        <TextBox x:Name="DH" HorizontalAlignment="Left" Margin="56,117,0,0"
Text="00" TextWrapping="Wrap" VerticalAlignment="Top" Width="31" Height="24"
FontFamily="Lucida Console" FontSize="20" FontWeight="Normal"
HorizontalContentAlignment="Center" VerticalContentAlignment="Center"
MaxLength="2"/>
        <TextBox x:Name="DL" HorizontalAlignment="Left" Margin="126,117,0,0"
Text="00" TextWrapping="Wrap" VerticalAlignment="Top" Width="31" Height="24"
FontFamily="Lucida Console" FontSize="20" FontWeight="Normal"
HorizontalContentAlignment="Center" VerticalContentAlignment="Center"
MaxLength="2"/>
        <TextBlock x:Name="RegistersView" HorizontalAlignment="Left"
Margin="168,32,0,0" TextWrapping="Wrap" VerticalAlignment="Top" Height="138"
Width="270" FontSize="24" FontFamily="Lucida Console" Background="#FFE6CBA5"
Padding="10,10,10,10"><Run Text="REJESTRY"/><LineBreak/><Run Language="pl-pl"
Text="AX: "/><Run Text="AH[00] AL[00]"/><LineBreak/><Run Language="pl-pl"
Text="BX: "/><Run Text="BH[00] BL[00]"/><LineBreak/><Run Language="pl-pl"
Text="CX: "/><Run Text="CH[00] CL[00]"/><LineBreak/><Run Language="pl-pl"
Text="DX: "/><Run Text="DH[00] DL[00]"/></TextBlock>
        <TextBlock x:Name="AddressRegistersView" HorizontalAlignment="Left"
Margin="443,32,0,0" TextWrapping="Wrap" VerticalAlignment="Top" Height="138"
Width="270" FontSize="24" FontFamily="Lucida Console" Background="#FFE6CBA5"
Padding="10,10,10,10"><Run Text="REJESTRY"/><Run Language="pl-pl" Text="
ADRESOWE"/><LineBreak/><Run Language="pl-pl" Text="SI"/><Run Text=":
[0000]"/><LineBreak/><Run Language="pl-pl" Text="DI"/><Run Text=":
[0000]"/><LineBreak/><Run Language="pl-pl" Text="BP"/><Run Text=":
[0000]"/><LineBreak/><Run Language="pl-pl" Text="BX"/><Run Text=":
[0000]"/></TextBlock>
        <Button x:Name="Insert" Content="Wczytaj" HorizontalAlignment="Left"
Margin="98,235,0,0" VerticalAlignment="Top" Width="64" Click="Insert_Click"
Height="22"/>
        <Label Content="Operacja" HorizontalAlignment="Left" Margin="24,282,0,0"
VerticalAlignment="Top" Height="36" Width="111" FontFamily="Lucida Console"
FontSize="18"/>
        <ComboBox x:Name="ChooseOperation" HorizontalAlignment="Left"
Margin="24,316,0,0" VerticalAlignment="Top" Width="120"
SelectionChanged="ChooseOperation_SelectionChanged"/>
```

```
        <Label Content="Rejestr 1" HorizontalAlignment="Left"
Margin="155,282,0,0" VerticalAlignment="Top" Height="36" Width="111"
FontFamily="Lucida Console" FontSize="18"/>
        <ComboBox x:Name="Register1" HorizontalAlignment="Left"
Margin="155,316,0,0" VerticalAlignment="Top" Width="120"/>
        <Grid x:Name="Reg2" Margin="287,279,0,0" HorizontalAlignment="Left"
Width="126" Height="68" VerticalAlignment="Top">
            <Label Content="Rejestr 2" HorizontalAlignment="Left"
Margin="0,3,0,0" VerticalAlignment="Top" Height="36" Width="111"
FontFamily="Lucida Console" FontSize="18"/>
            <ComboBox x:Name="Register2" HorizontalAlignment="Left"
Margin="0,37,0,0" VerticalAlignment="Top" Width="120"/>
        </Grid>
        <Button x:Name="Execute" Content="Symuluj" HorizontalAlignment="Left"
Margin="24,351,0,0" VerticalAlignment="Top" Height="25" Width="64"
Click="Execute_Click"/>
        <Button x:Name="Random" Content="Losuj" HorizontalAlignment="Left"
Margin="32,235,0,0" VerticalAlignment="Top" Width="62" Click="Random_Click"
Height="22"/>
        <TextBox x:Name="SI" HorizontalAlignment="Left" Margin="96,148,0,0"
Text="0000" TextWrapping="Wrap" VerticalAlignment="Top" Width="62" Height="24"
FontFamily="Lucida Console" FontSize="20" FontWeight="Normal"
HorizontalContentAlignment="Center" VerticalContentAlignment="Center"
MaxLength="4"/>
        <TextBox x:Name="DI" HorizontalAlignment="Left" Margin="96,176,0,0"
Text="0000" TextWrapping="Wrap" VerticalAlignment="Top" Width="62" Height="24"
FontFamily="Lucida Console" FontSize="20" FontWeight="Normal"
HorizontalContentAlignment="Center" VerticalContentAlignment="Center"
MaxLength="4"/>
        <TextBox x:Name="BP" HorizontalAlignment="Left" Margin="96,205,0,0"
Text="0000" TextWrapping="Wrap" Width="62" FontFamily="Lucida Console"
FontSize="20" FontWeight="Normal" HorizontalContentAlignment="Center"
VerticalContentAlignment="Center" MaxLength="4" Height="24"
VerticalAlignment="Top"/>
        <Label Content="SI" HorizontalAlignment="Left" Margin="57,147,0,0"
VerticalAlignment="Top" Height="27" Width="34" FontFamily="Lucida Console"
FontSize="18"/>
        <Label Content="DI" HorizontalAlignment="Left" Margin="57,175,0,0"
VerticalAlignment="Top" Height="26" Width="34" FontFamily="Lucida Console"
FontSize="18"/>
        <Label Content="BP" HorizontalAlignment="Left" Margin="57,202,0,0"
VerticalAlignment="Top" Height="26" Width="34" FontFamily="Lucida Console"
FontSize="18"/>
        <TextBox x:Name="AssemblerBox" HorizontalAlignment="Left"
Margin="443,177,0,0" TextWrapping="Wrap" Width="270" FontFamily="Lucida Console"
FontSize="20" FontWeight="Normal" HorizontalContentAlignment="Left"
VerticalContentAlignment="Top" Height="276" VerticalAlignment="Top"
AcceptsReturn="True"/>
        <Button x:Name="ExecuteAssembler" Content="Wykonaj"
HorizontalAlignment="Left" Margin="649,460,0,0" VerticalAlignment="Top"
Height="25" Width="64" Click="ExecuteAssembler_Click"/>
        <Button x:Name="InsertData" Content="Wczytaj" HorizontalAlignment="Left"
Margin="373,206,0,0" VerticalAlignment="Top" Width="64" Click="InsertData_Click"
Height="22"/>
        <Button x:Name="RandomData" Content="Losuj" HorizontalAlignment="Left"
Margin="307,206,0,0" VerticalAlignment="Top" Width="62" Click="RandomData_Click"
Height="22"/>
        <TextBlock x:Name="DataViewSingle" HorizontalAlignment="Left"
Margin="168,176,0,0" TextWrapping="Wrap" VerticalAlignment="Top" Height="68"
Width="133" FontSize="24" FontFamily="Lucida Console" Background="#FFE6CBA5"
Padding="10,10,10,10"><Run Text="PAMI"/><Run Language="pl-pl"
Text="ĘĆ"/><LineBreak/><Run Language="pl-pl" Text="0000 00"/></TextBlock>
```

```
        <TextBox x:Name="DataAddress" HorizontalAlignment="Left"
Margin="375,177,0,0" Text="0000" TextWrapping="Wrap" Width="62"
FontFamily="Lucida Console" FontSize="20" FontWeight="Normal"
HorizontalContentAlignment="Center" VerticalContentAlignment="Center"
MaxLength="4" Height="24" VerticalAlignment="Top"/>
        <Button x:Name="ShowData" Content="Odczytaj" HorizontalAlignment="Left"
Margin="306,178,0,0" VerticalAlignment="Top" Width="64" Click="ShowData_Click"
Height="22"/>
        <Button x:Name="SaveData" Content="Zapisz" HorizontalAlignment="Left"
Margin="374,233,0,0" VerticalAlignment="Top" Width="64" Click="SaveData_Click"
Height="22"/>
    </Grid>
</Window>
```

MainWindow.xaml.cs

Czyli kod odpowiadający za działanie interfejsu

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Intel8086;
using Microsoft.Win32;

namespace SymulatorIntel8086
{
    public partial class MainWindow : Window
    {
        Procesor proc;
        Memory mem;
        public MainWindow()
        {
            InitializeComponent();
            ChooseOperation.Items.Add("MOV");
            ChooseOperation.Items.Add("XCHG");
            ChooseOperation.Items.Add("INC");
            ChooseOperation.Items.Add("DEC");
            ChooseOperation.Items.Add("NOT");
            ChooseOperation.Items.Add("NEG");
            ChooseOperation.Items.Add("AND");
            ChooseOperation.Items.Add("OR");
            ChooseOperation.Items.Add("XOR");
            ChooseOperation.Items.Add("ADD");
            ChooseOperation.Items.Add("SUB");
            Register1.Items.Add("AH");
            Register1.Items.Add("AL");
            Register1.Items.Add("BH");
            Register1.Items.Add("BL");
            Register1.Items.Add("CH");
            Register1.Items.Add("CL");
            Register1.Items.Add("DH");
            Register1.Items.Add("DL");
            Register2.Items.Add("AH");
            Register2.Items.Add("AL");
            Register2.Items.Add("BH");
            Register2.Items.Add("BL");
            Register2.Items.Add("CH");
            Register2.Items.Add("CL");
            Register2.Items.Add("DH");
            Register2.Items.Add("DL");
            proc = new Procesor();
            mem = new Memory();
            proc.memory = mem;
        }

        private void Insert_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                proc = new Procesor(AH.Text, AL.Text, BH.Text, BL.Text, CH.Text,
CL.Text, DH.Text, DL.Text, SI.Text, DI.Text, BP.Text);
                RefreshRegisters();
                proc.memory = mem;
            }
            catch (ArgumentException)
            {
                proc = new Procesor();
                proc.memory = mem;
                RefreshRegisters(false);
            }
        }

        private void Execute_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                if (proc.ExecuteOperation($"{ChooseOperation.SelectedItem}
{Register1.SelectedItem},{Register2.SelectedItem}"))
                {
                    RefreshRegisters();
                }
                else
                    MessageBox.Show("Proszę wybrać operację oraz sektory");
            }
            catch (Exception ex)
            {
                MessageBox.Show("Proszę wybrać operację oraz sektory");
            }
        }

        private void Random_Click(object sender, RoutedEventArgs e)
        {
            proc = new Procesor(Convert.ToInt32(DateTime.Now.Millisecond));
            proc.memory = mem;
            RefreshRegisters();
        }

        private void ChooseOperation_SelectionChanged(object sender,
SelectionChangedEventArgs e)
        {
            if (ChoosenOperation()) Reg2.Visibility = Visibility.Hidden;
            else Reg2.Visibility = Visibility.Visible;
        }
        bool ChoosenOperation()
        {
            string op = ChooseOperation.SelectedItem.ToString();
            return op == "INC" || op == "DEC" || op == "NOT" || op == "NEG";
        }

        private void ExecuteAssembler_Click(object sender, RoutedEventArgs e)
        {
            string[] commands = AssemblerBox.Text.Split(Environment.NewLine,
StringSplitOptions.RemoveEmptyEntries);
            foreach (string cmd in commands)
                if (!proc.ExecuteOperation(cmd))
                {
                    AssemblerBox.Text = "Napotkano błąd w:\n" + cmd + "\n\n" +
AssemblerBox.Text;
                    break;
                }
            RefreshRegisters();
        }

        private void RefreshRegisters(bool success = true)
        {
            if (success)
            {
                RegistersView.Text = "REJESTRY\n" + proc.ToString();
                AddressRegistersView.Text = "REJESTRY ADRESOWE\n" +
proc.AddressRegisters();
            }
            else
            {
                RegistersView.Text = "BŁĘDNE DANE\n" + proc.ToString();
                AddressRegistersView.Text = "BŁĘDNE DANE\n" +
proc.AddressRegisters();
            }
        }

        private void RandomData_Click(object sender, RoutedEventArgs e)
        {
            mem = new Memory(Convert.ToInt32(DateTime.Now.Millisecond));
            proc.memory = mem;
        }

        private void ShowData_Click(object sender, RoutedEventArgs e)
        {
            if (Procesor.CheckData(DataAddress.Text))
                DataViewSingle.Text = "PAMIĘĆ\n" +
mem.DisplayData(DataAddress.Text);
            else
                DataViewSingle.Text = "ZŁY ADRES";
        }

        private void InsertData_Click(object sender, RoutedEventArgs e)
        {
            OpenFileDialog ofd = new OpenFileDialog();
            ofd.FileName = "data";
            ofd.DefaultExt = ".8086";
            ofd.Filter = "intel 8086 data file|*.8086";
            Nullable<bool> result = ofd.ShowDialog();
            if (result == true)
            {
                string fn = ofd.FileName;
                mem.Load(fn);
            }
        }

        private void SaveData_Click(object sender, RoutedEventArgs e)
        {
            SaveFileDialog sfd = new SaveFileDialog();
            sfd.FileName = "data";
            sfd.DefaultExt = ".8086";
            sfd.Filter = "intel 8086 data file|*.8086|plik tekstowy|*.txt";
            if (sfd.ShowDialog() == true)
                mem.Save(sfd.FileName);
        }
    }
}
```