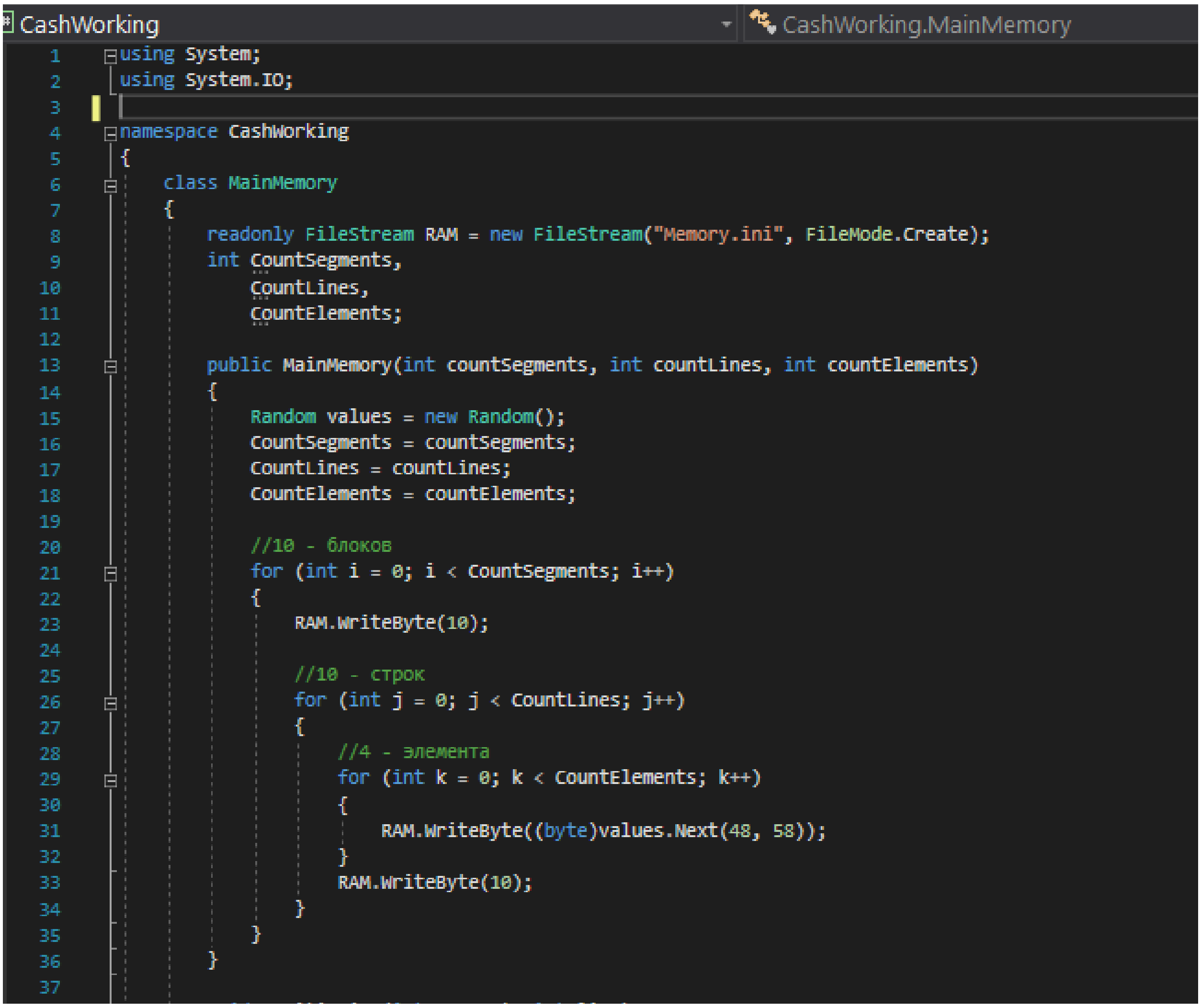
Приложение имитирующее работу КЭШа с прямым отображением

https://github.com/MAXXXYMIRON/ECM/tree/master/%D0%9A%D0%AD%D0%A8%20-%20%D0%A0%D0%B0%D0%B1%D0%BE%D1%82%D0%B0/%D0%9A%D0%AD%D0%A8_%D0%A0%D0%B0%D0%B1%D0%BE%D1%82%D0%B0_%D0%9F%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5

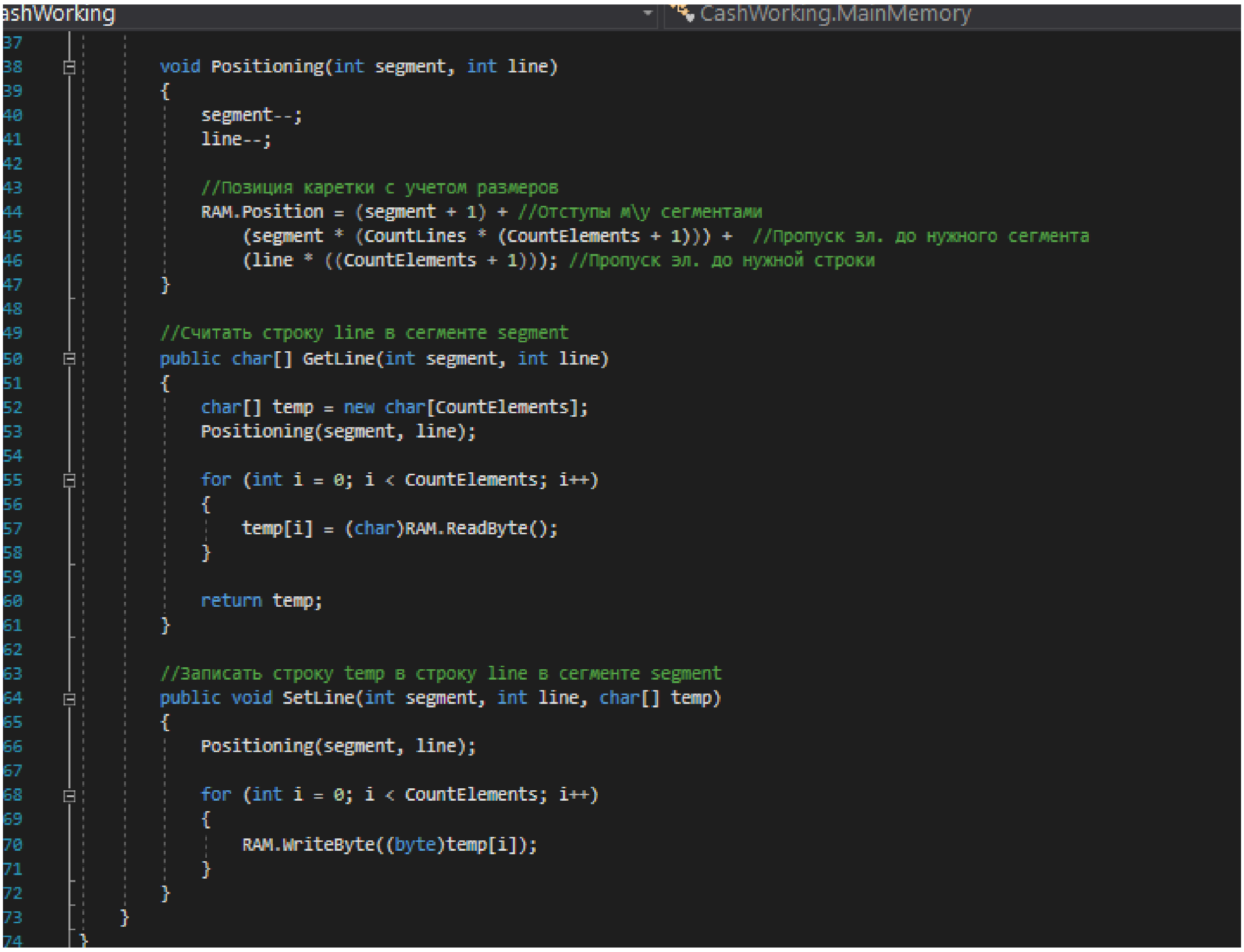
Код реализации

https://github.com/MAXXXYMIRON/ECM/tree/master/%D0%9A%D0%AD%D0%A8%20-%20%D0%A0%D0%B0%D0%B1%D0%BE%D1%82%D0%B0/CashWorking

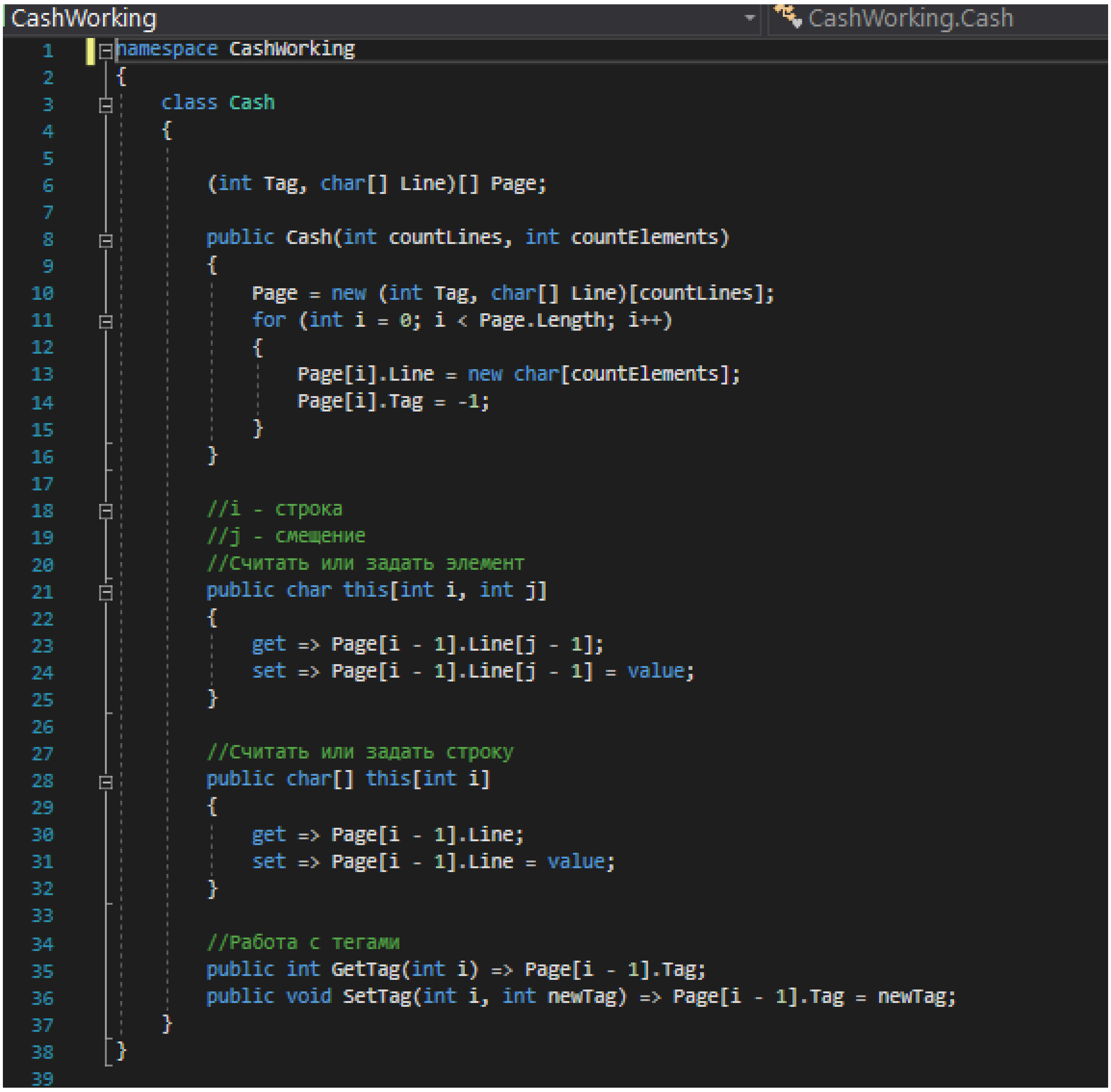
Листинг



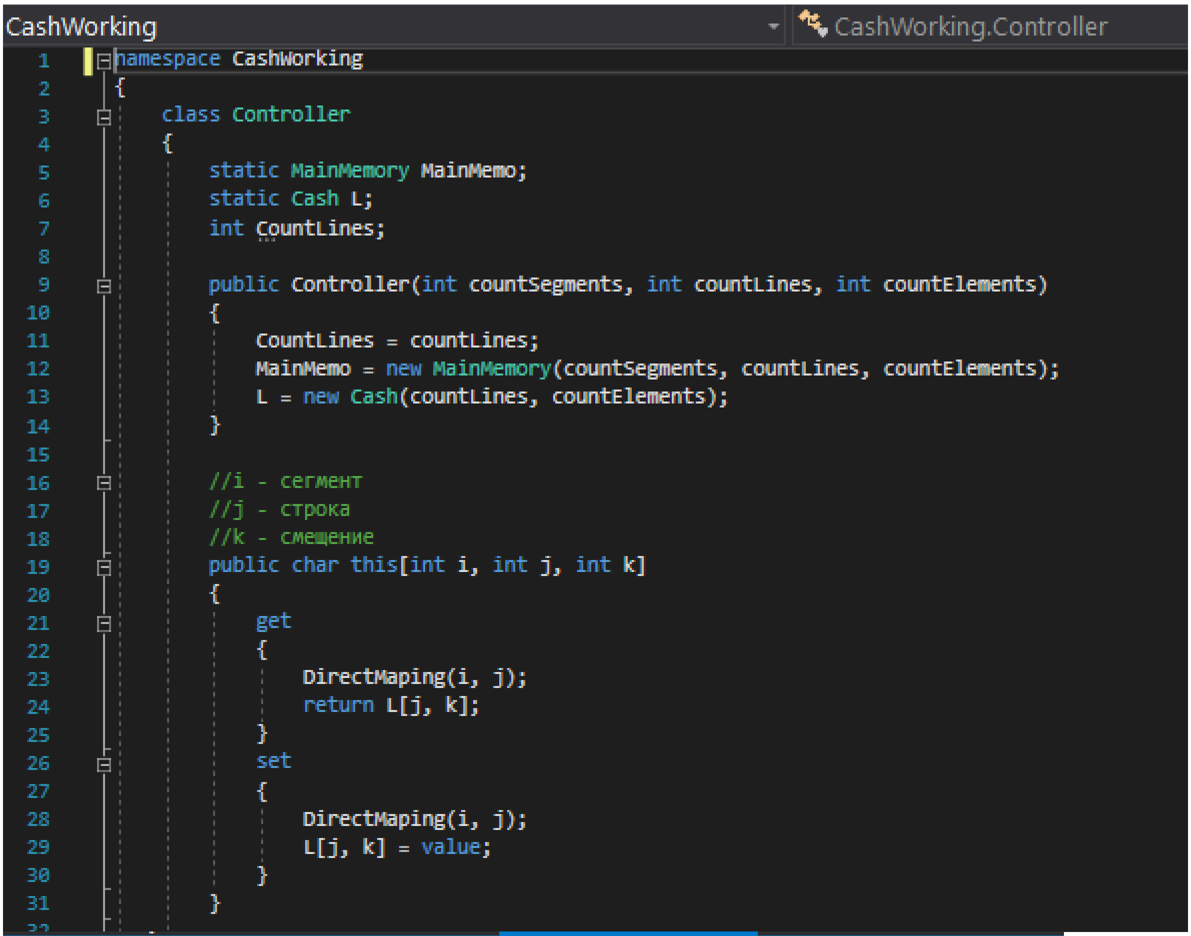
```
using System;
using System.IO;

namespace CashWorking
{
    class MainMemory
    {
        readonly FileStream RAM = new FileStream("Memory.ini", FileMode.Create);
        int CountSegments,
            CountLines,
            CountElements;

        public MainMemory(int countSegments, int countLines, int countElements)
        {
            Random values = new Random();
            CountSegments = countSegments;
            CountLines = countLines;
            CountElements = countElements;

            //10 - блоков
            for (int i = 0; i < CountSegments; i++)
            {
                RAM.WriteByte(10);

                //10 - строк
                for (int j = 0; j < CountLines; j++)
                {
                    //4 - элемента
                    for (int k = 0; k < CountElements; k++)
                    {
                        RAM.WriteByte((byte)values.Next(48, 58));
                    }
                    RAM.WriteByte(10);
                }
            }
        }
```



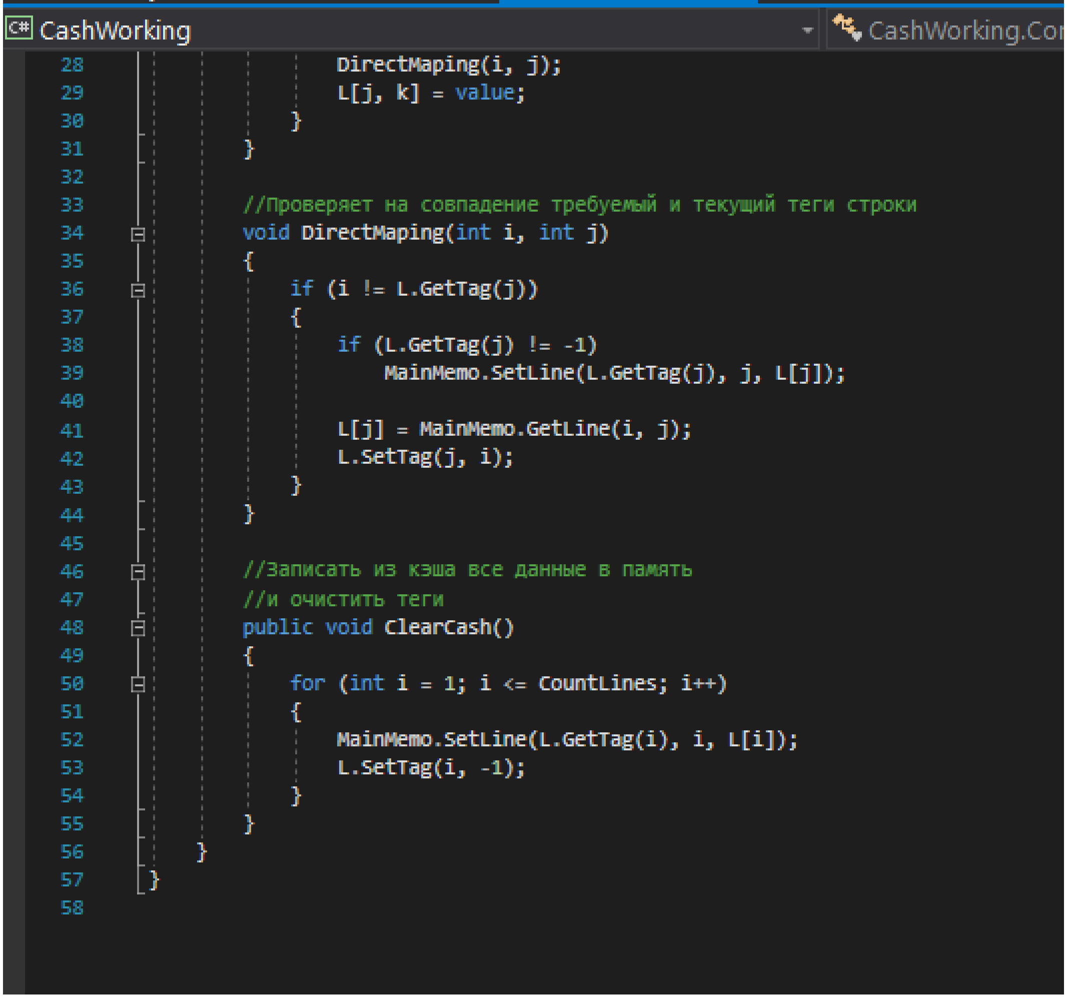
```

        void Positioning(int segment, int line)
        {
            segment--;
            line--;

            //Позиция каретки с учетом размеров
            RAM.Position = (segment + 1) + //Отступы м\у сегментами
                (segment * (CountLines * (CountElements + 1))) +  //Пропуск эл. до нужного сегмента
                (line * ((CountElements + 1))); //Пропуск эл. до нужной строки
        }

        //Считать строку line в сегменте segment
        public char[] GetLine(int segment, int line)
        {
            char[] temp = new char[CountElements];
            Positioning(segment, line);

            for (int i = 0; i < CountElements; i++)
            {
                temp[i] = (char)RAM.ReadByte();
            }

            return temp;
        }

        //Записать строку temp в строку line в сегменте segment
        public void SetLine(int segment, int line, char[] temp)
        {
            Positioning(segment, line);

            for (int i = 0; i < CountElements; i++)
            {
                RAM.WriteByte((byte)temp[i]);
            }
        }
    }
}
```



```
namespace CashWorking
{
    class Cash
    {

        (int Tag, char[] Line)[] Page;

        public Cash(int countLines, int countElements)
        {
            Page = new (int Tag, char[] Line)[countLines];
            for (int i = 0; i < Page.Length; i++)
            {
                Page[i].Line = new char[countElements];
                Page[i].Tag = -1;
            }
        }

        //i - строка
        //j - смещение
        //Считать или задать элемент
        public char this[int i, int j]
        {
            get => Page[i - 1].Line[j - 1];
            set => Page[i - 1].Line[j - 1] = value;
        }

        //Считать или задать строку
        public char[] this[int i]
        {
            get => Page[i - 1].Line;
            set => Page[i - 1].Line = value;
        }

        //Работа с тегами
        public int GetTag(int i) => Page[i - 1].Tag;
        public void SetTag(int i, int newTag) => Page[i - 1].Tag = newTag;
    }
}

```

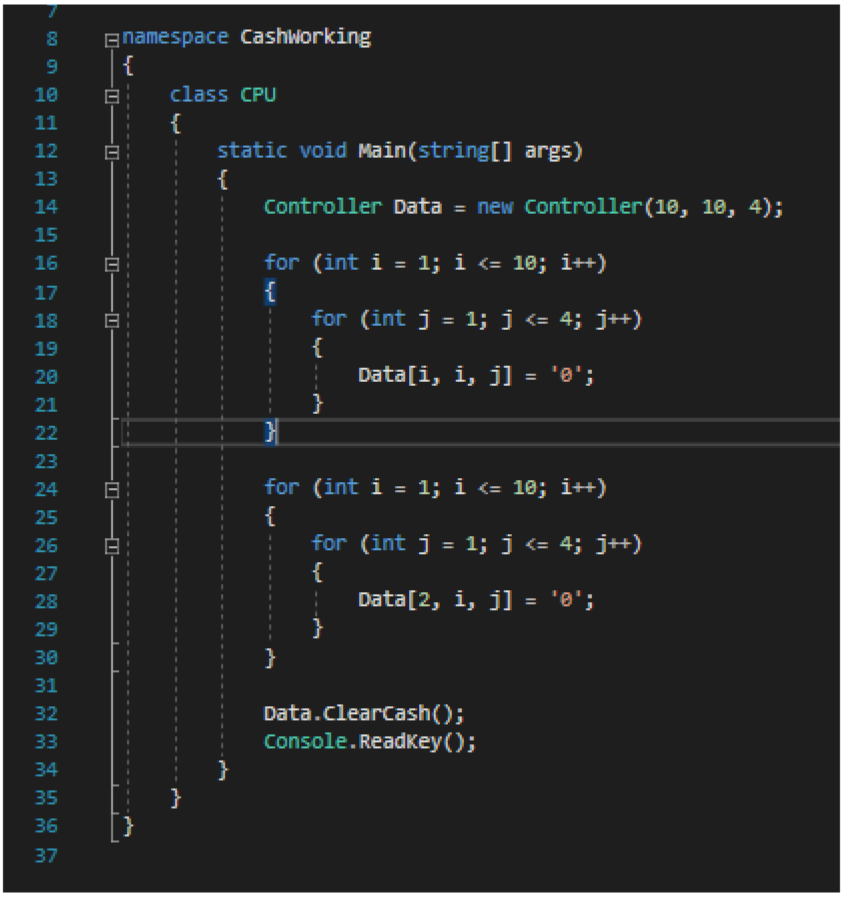


```csharp
namespace CashWorking
{
    class Controller
    {
        static MainMemory MainMemo;
        static Cash L;
        int CountLines;

        public Controller(int countSegments, int countLines, int countElements)
        {
            CountLines = countLines;
            MainMemo = new MainMemory(countSegments, countLines, countElements);
            L = new Cash(countLines, countElements);
        }

        //i - сегмент
        //j - строка
        //k - смещение
        public char this[int i, int j, int k]
        {
            get
            {
                DirectMaping(i, j);
                return L[j, k];
            }
            set
            {
                DirectMaping(i, j);
                L[j, k] = value;
            }
        }
```



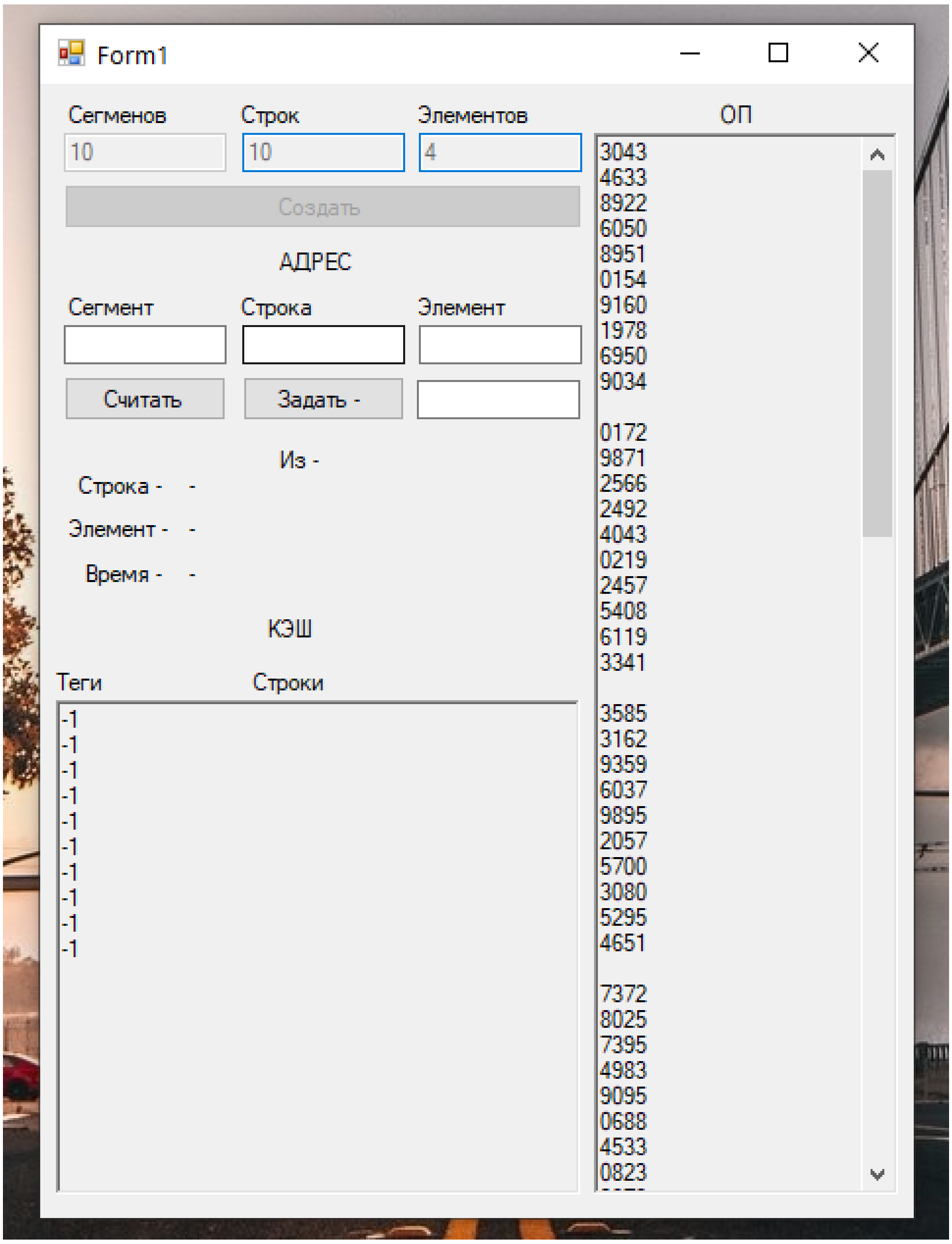
```csharp
                DirectMaping(i, j);
                L[j, k] = value;
            }
        }
```

```csharp

        //Проверяет на совпадение требуемый и текущий теги строки
        void DirectMaping(int i, int j)
        {
            if (i != L.GetTag(j))
            {
                if (L.GetTag(j) != -1)
                    MainMemo.SetLine(L.GetTag(j), j, L[j]);

                L[j] = MainMemo.GetLine(i, j);
                L.SetTag(j, i);
            }
        }

        //Записать из кэша все данные в память
        //и очистить теги
        public void ClearCash()
        {
            for (int i = 1; i <= CountLines; i++)
            {
                MainMemo.SetLine(L.GetTag(i), i, L[i]);
                L.SetTag(i, -1);
            }
        }
    }
}
```



```
namespace CashWorking
{
    class CPU
    {
        static void Main(string[] args)
        {
            Controller Data = new Controller(10, 10, 4);

            for (int i = 1; i <= 10; i++)
            {
                for (int j = 1; j <= 4; j++)
                {
                    Data[i, i, j] = '0';
                }
            }

            for (int i = 1; i <= 10; i++)
            {
                for (int j = 1; j <= 4; j++)
                {
                    Data[2, i, j] = '0';
                }
            }

            Data.ClearCash();
            Console.ReadKey();
        }
    }
}
```

Приложение



Сегменов: 10

Строк: 10

Элементов: 4

Сегмент:

Строка:

Элемент:

Из -

Строка - -

Элемент - -

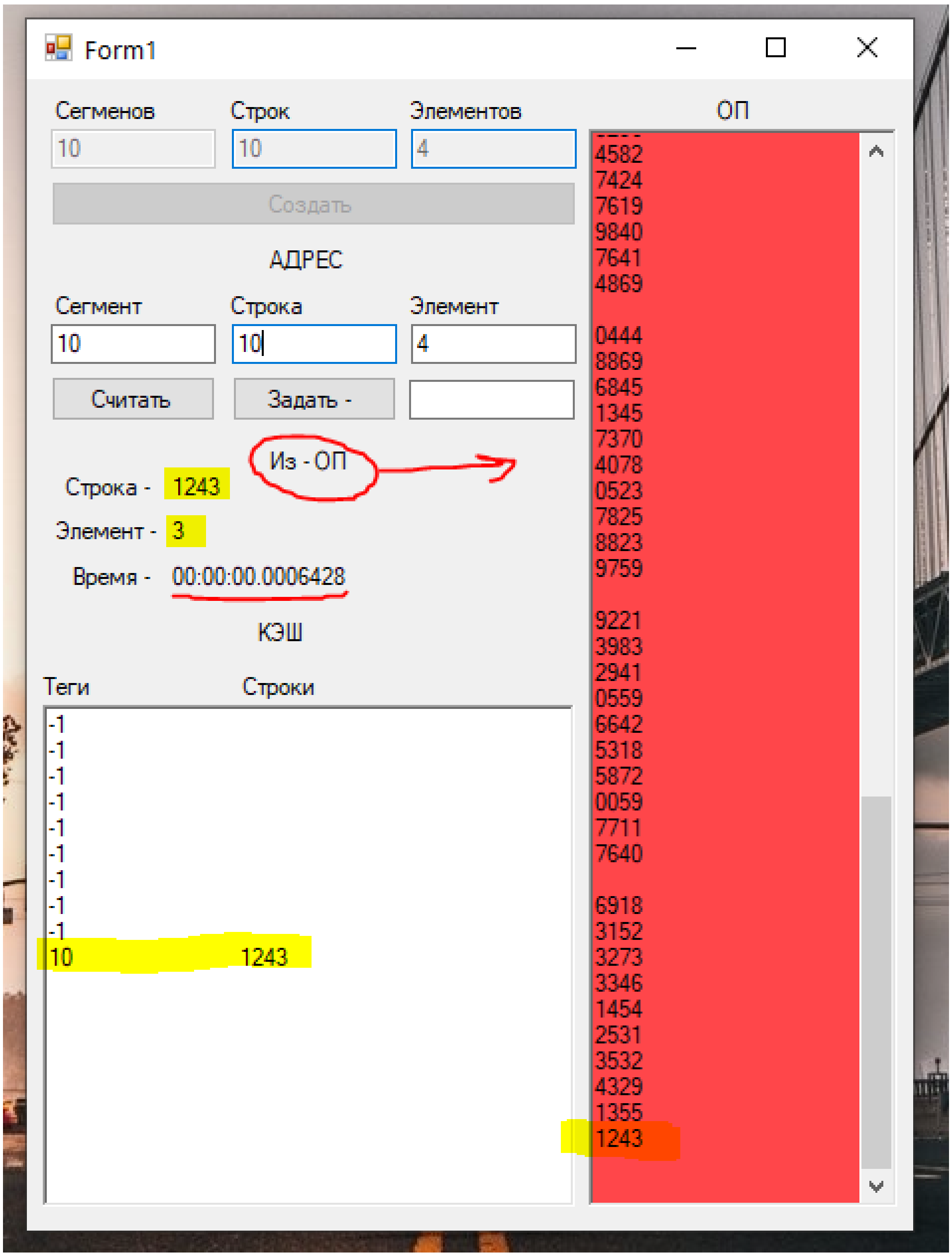
Время - -

Теги Строки

```
-1
-1
-1
-1
-1
-1
-1
-1
-1
-1
```

ОП

```
3043
4633
8922
6050
8951
0154
9160
1978
6950
9034

0172
9871
2566
2492
4043
0219
2457
5408
6119
3341

3585
3162
9359
6037
9895
2057
5700
3080
5295
4651

7372
8025
7395
4983
9095
0688
4533
0823
```



Сегменов 10
Строк 10
Элементов 4

АДРЕС

Сегмент 10
Строка 10
Элемент 4

Из - ОП

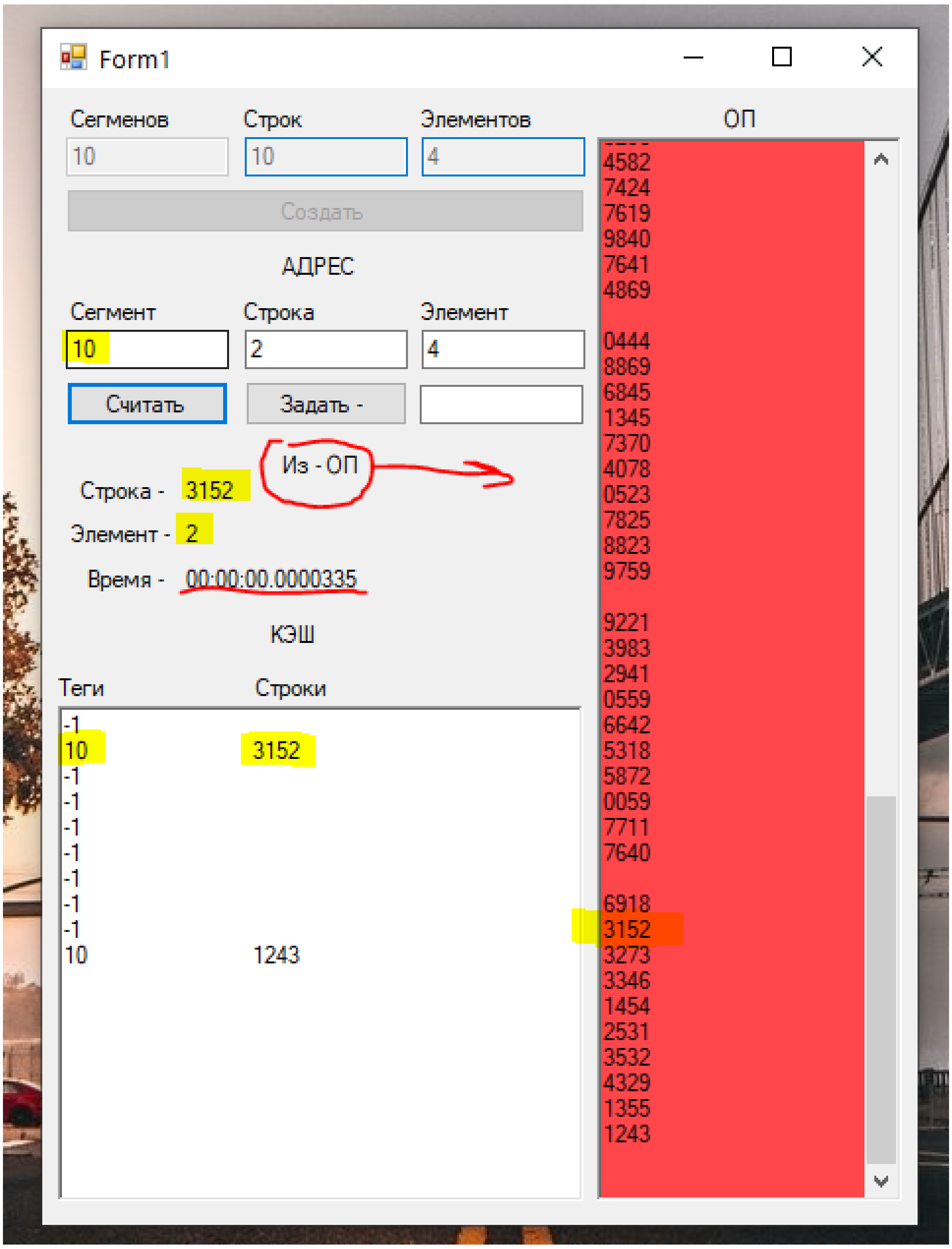
Строка - 1243
Элемент - 3
Время - 00:00:00.0006428

КЭШ

Теги	Строки
-1	
-1	
-1	
-1	
-1	
-1	
-1	
-1	
-1	
10	1243

ОП

```
4582
7424
7619
9840
7641
4869

0444
8869
6845
1345
7370
4078
0523
7825
8823
9759

9221
3983
2941
0559
6642
5318
5872
0059
7711
7640

6918
3152
3273
3346
1454
2531
3532
4329
1355
1243
```



Сегменов 10 Строк 10 Элементов 4

Сегмент 10 Строка 2 Элемент 4

Из - ОП

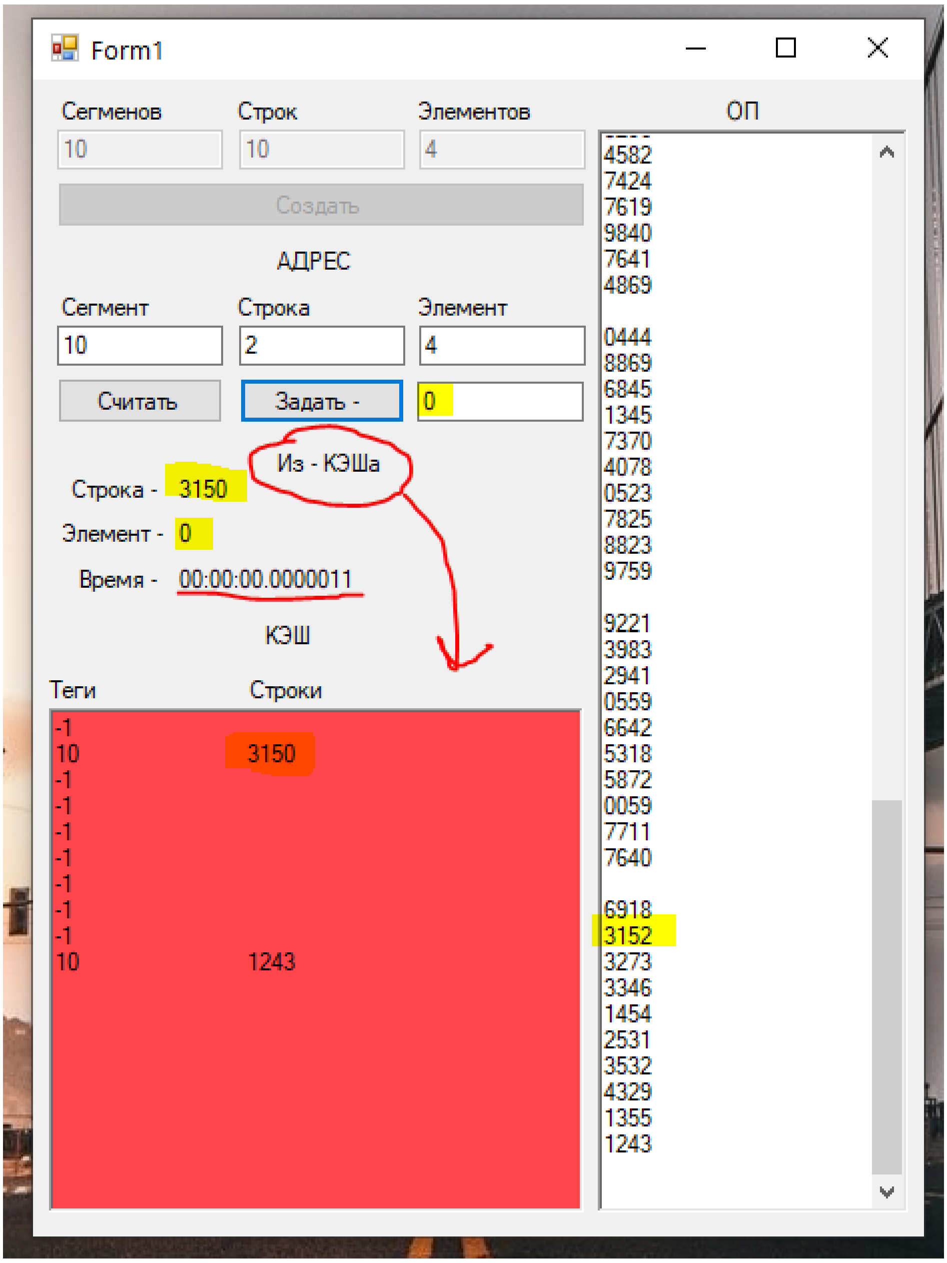
Строка - 3152

Элемент - 2

Время - 00:00:00.0000335

```
Теги        Строки
-1
10          3152
-1
-1
-1
-1
-1
-1
-1
10          1243
```

```
4582
7424
7619
9840
7641
4869

0444
8869
6845
1345
7370
4078
0523
7825
8823
9759

9221
3983
2941
0559
6642
5318
5872
0059
7711
7640

6918
3152
3273
3346
1454
2531
3532
4329
1355
1243
```



Сегменов 10 | Строк 10 | Элементов 4

Сегмент 10 | Строка 2 | Элемент 4

Задать - 0

Из - КЭШа

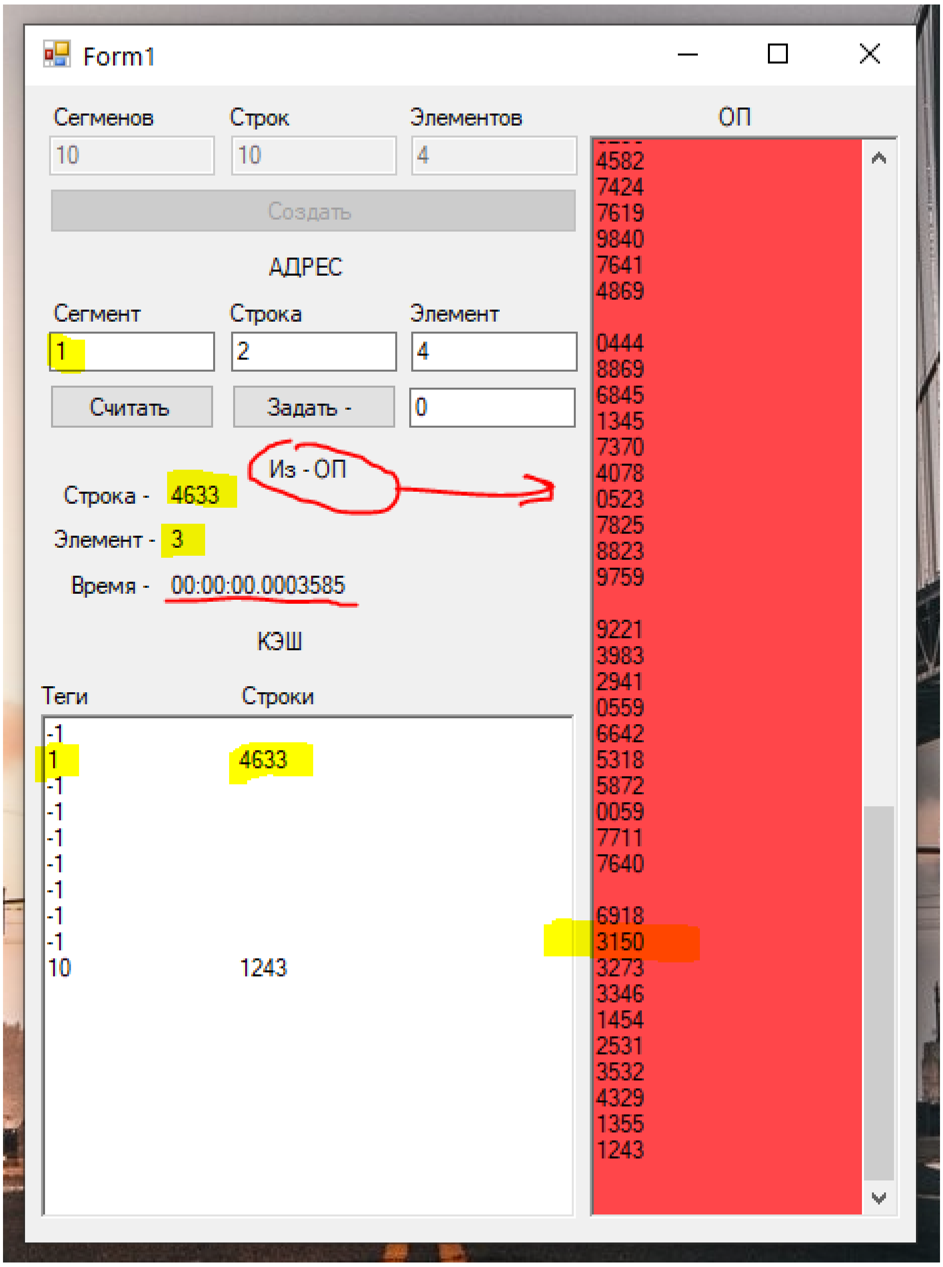
Строка - 3150

Элемент - 0

Время - 00:00:00.0000011

Теги	Строки
-1	
10	3150
-1	
-1	
-1	
-1	
-1	
-1	
-1	
10	1243

```
4582
7424
7619
9840
7641
4869

0444
8869
6845
1345
7370
4078
0523
7825
8823
9759

9221
3983
2941
0559
6642
5318
5872
0059
7711
7640

6918
3152
3273
3346
1454
2531
3532
4329
1355
1243
```



Сегменов 10 Строк 10 Элементов 4

Сегмент 1 Строка 2 Элемент 4

Задать - 0

Из - ОП

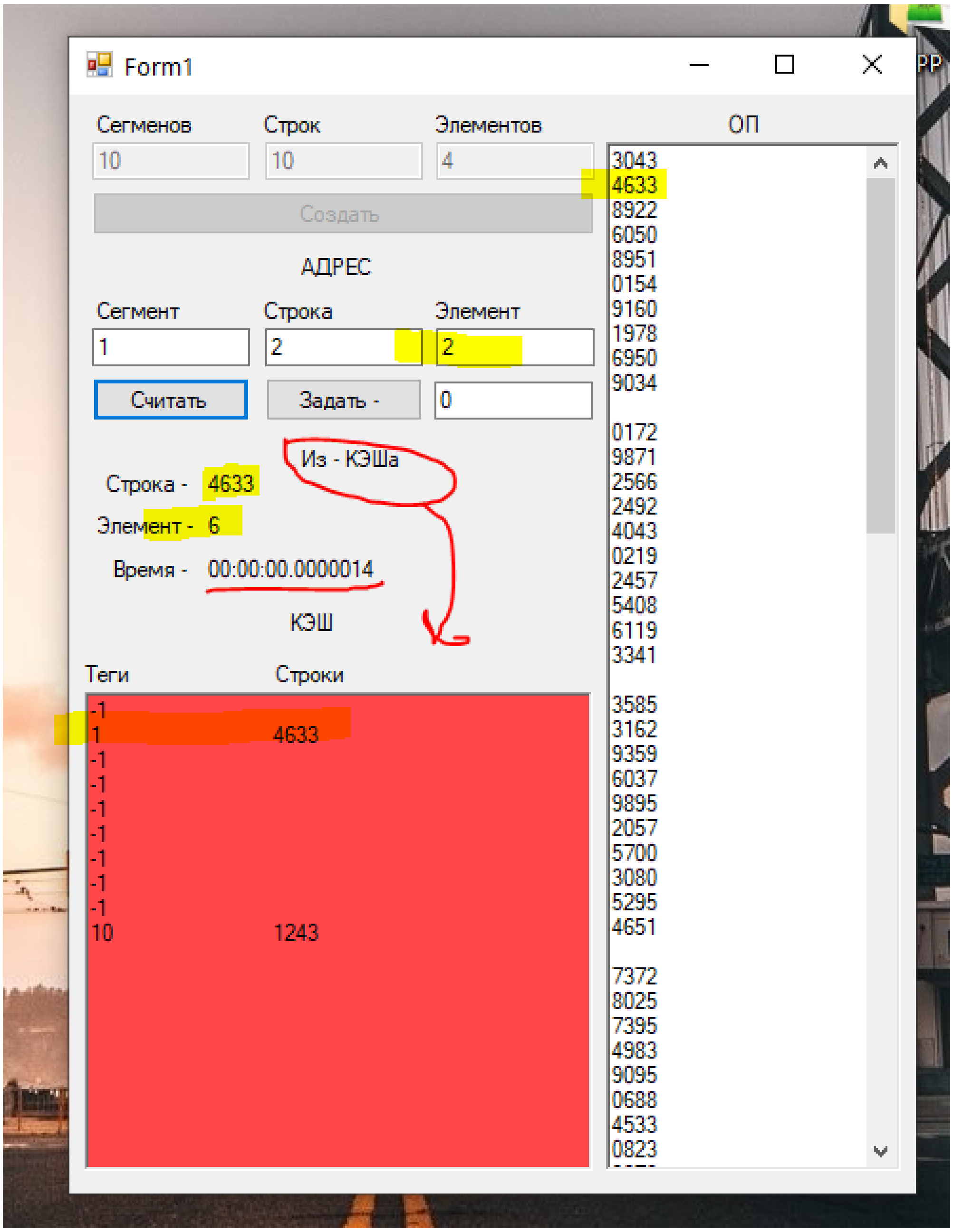
Строка - 4633

Элемент - 3

Время - 00:00:00.0003585

Теги	Строки
-1	
1	4633
-1	
-1	
-1	
-1	
-1	
-1	
-1	
10	1243

```
4582
7424
7619
9840
7641
4869

0444
8869
6845
1345
7370
4078
0523
7825
8823
9759

9221
3983
2941
0559
6642
5318
5872
0059
7711
7640

6918
3150
3273
3346
1454
2531
3532
4329
1355
1243
```



Сегменов: 10
Строк: 10
Элементов: 4

АДРЕС

Сегмент: 1
Строка: 2
Элемент: 2

0

Из - КЭШа

Строка - 4633
Элемент - 6
Время - 00:00:00.0000014

КЭШ

Теги	Строки
-1	
1	4633
-1	
-1	
-1	
-1	
-1	
-1	
-1	
10	1243

ОП

```
3043
4633
8922
6050
8951
0154
9160
1978
6950
9034

0172
9871
2566
2492
4043
0219
2457
5408
6119
3341

3585
3162
9359
6037
9895
2057
5700
3080
5295
4651

7372
8025
7395
4983
9095
0688
4533
0823
```