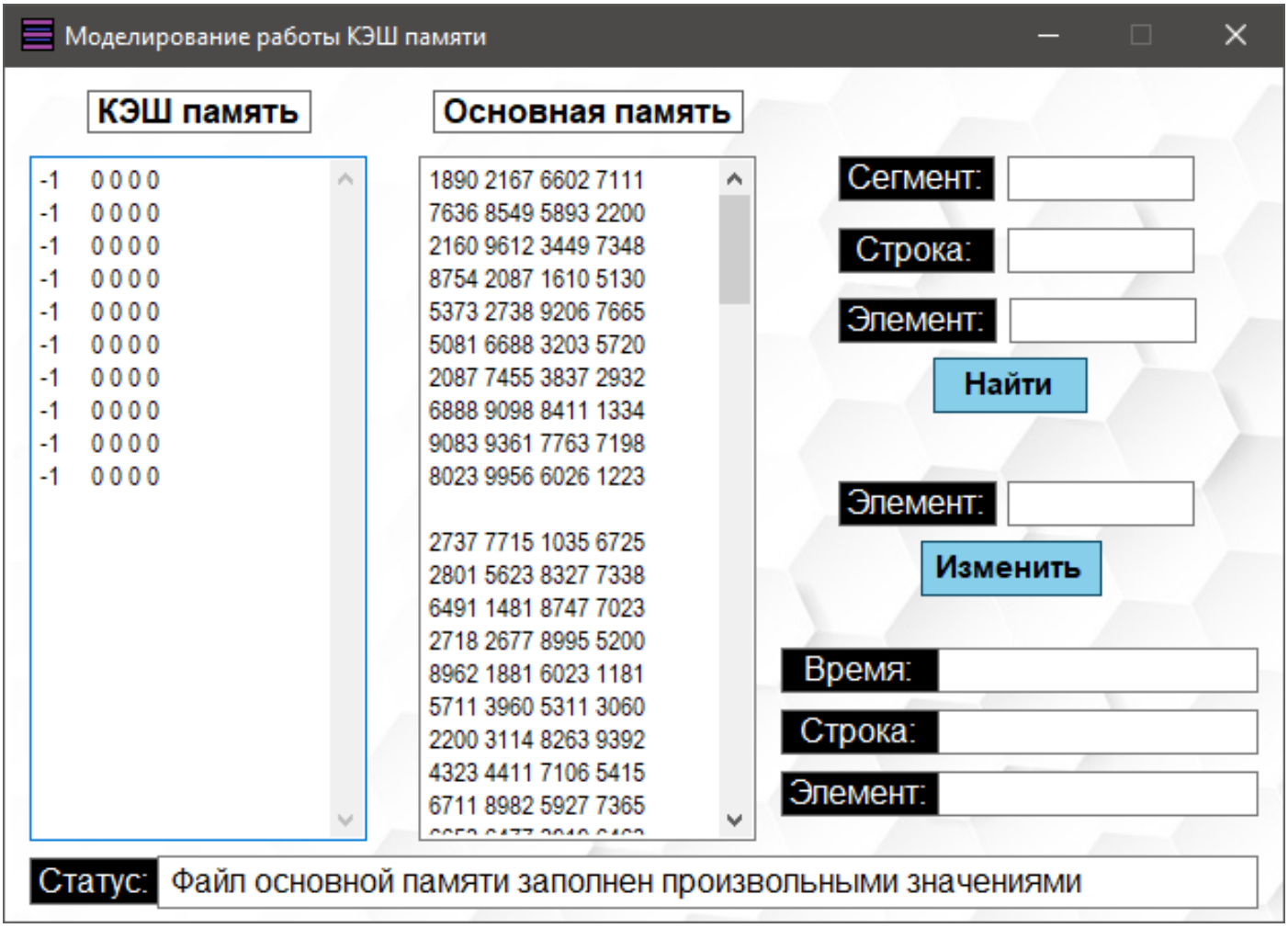
Лабараторная работа на тему

«Моделирование работы КЭШ-памяти»

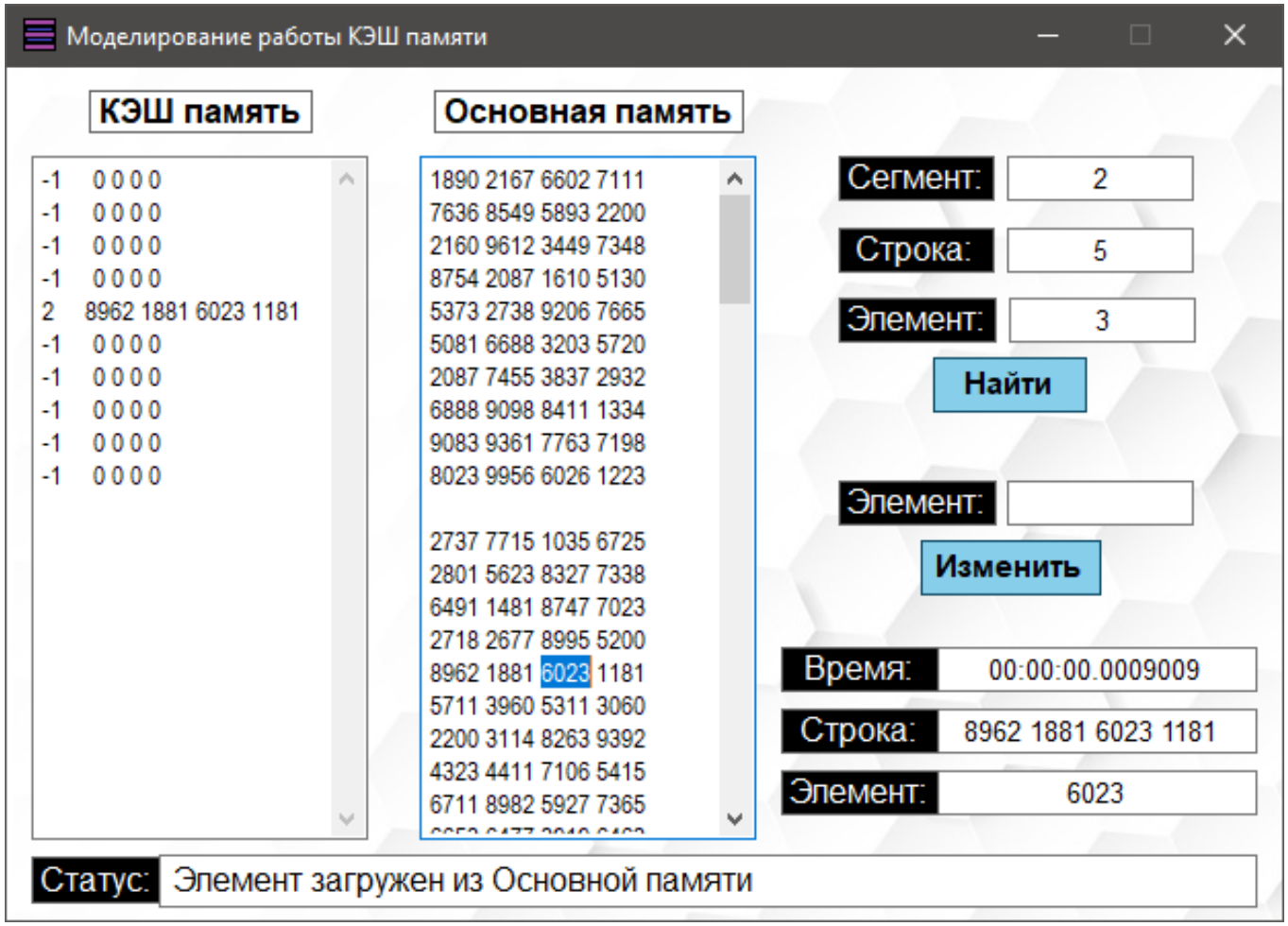
Левасюк Денис, ИВТ-18-2

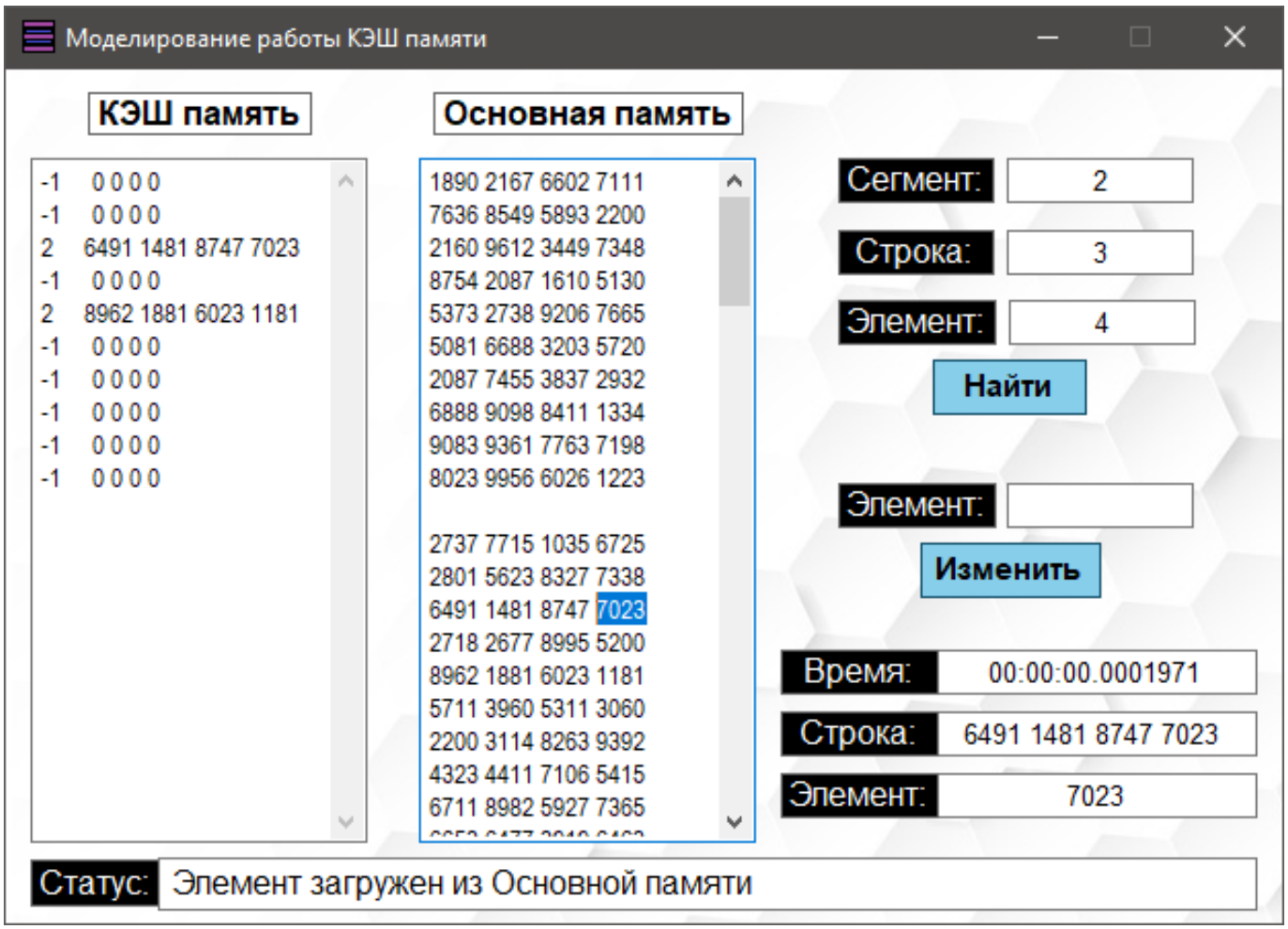
Интерфейс программы:



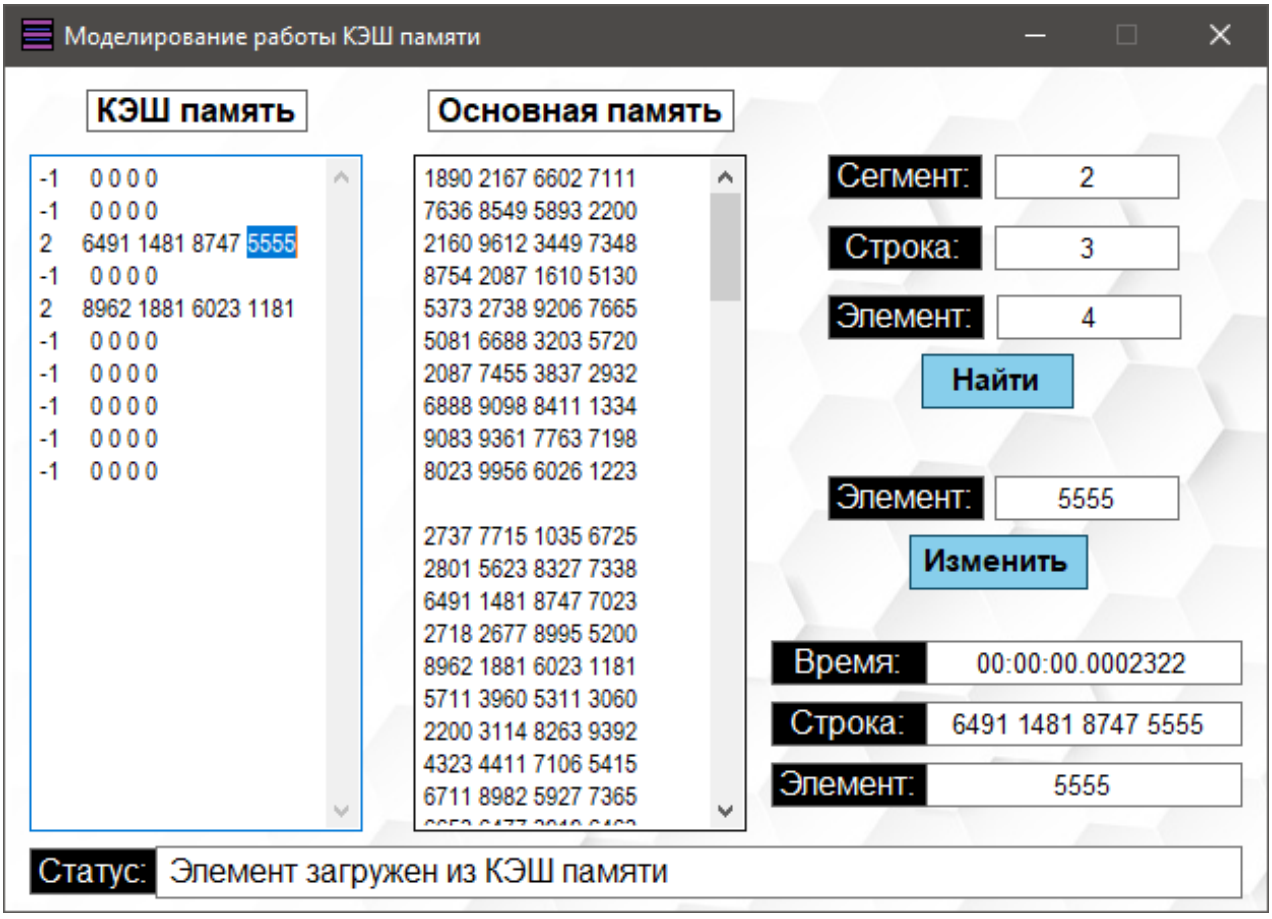
Проект на GitHub:
https://github.com/LevasyukDY/ECM/tree/master/%D0%9A%D0%AD%D0%A8%20-%20%D0%A0%D0%B0%D0%B1%D0%BE%D1%82%D0%B0

Загрузка элемента из Основной памяти:

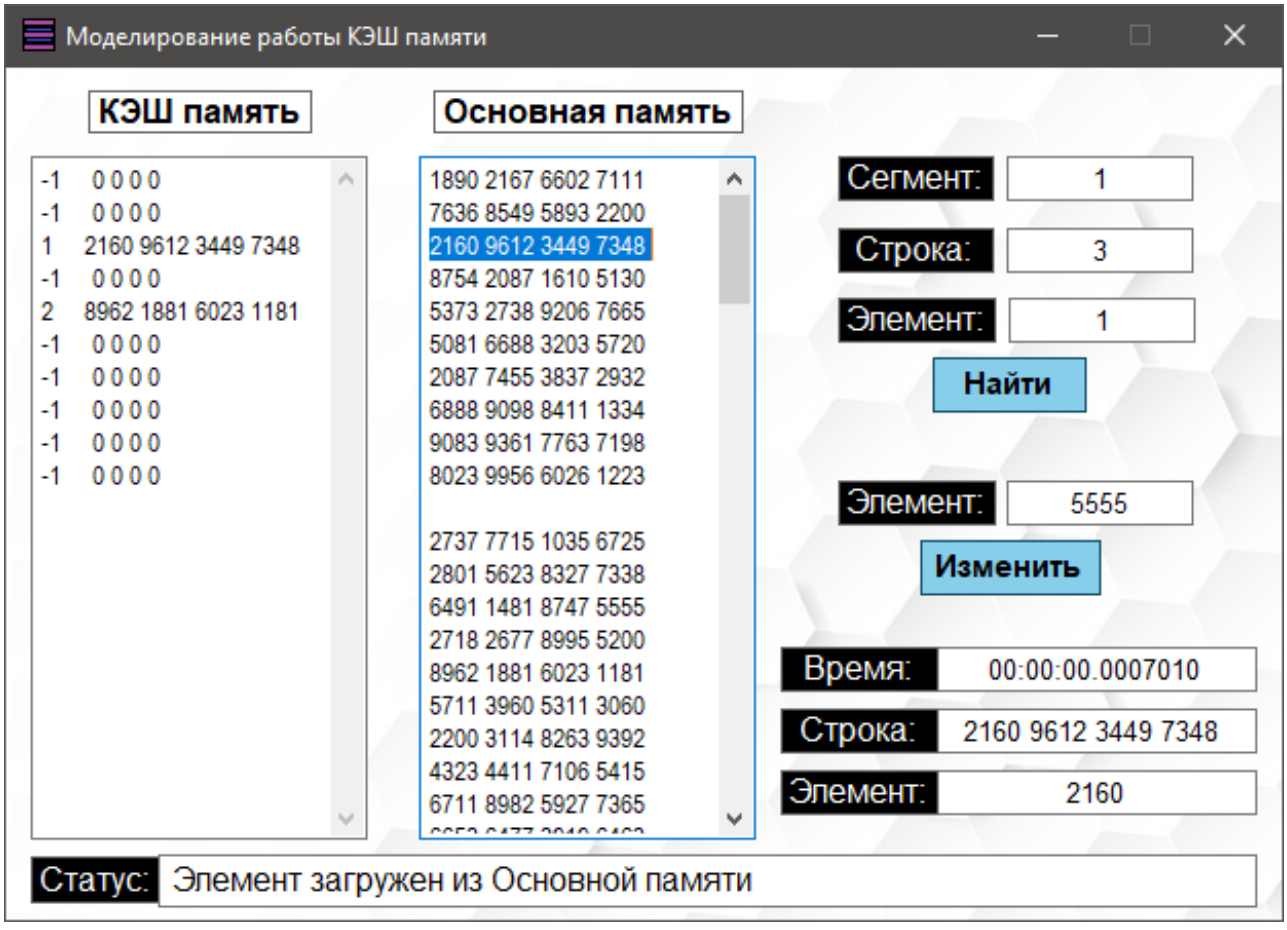




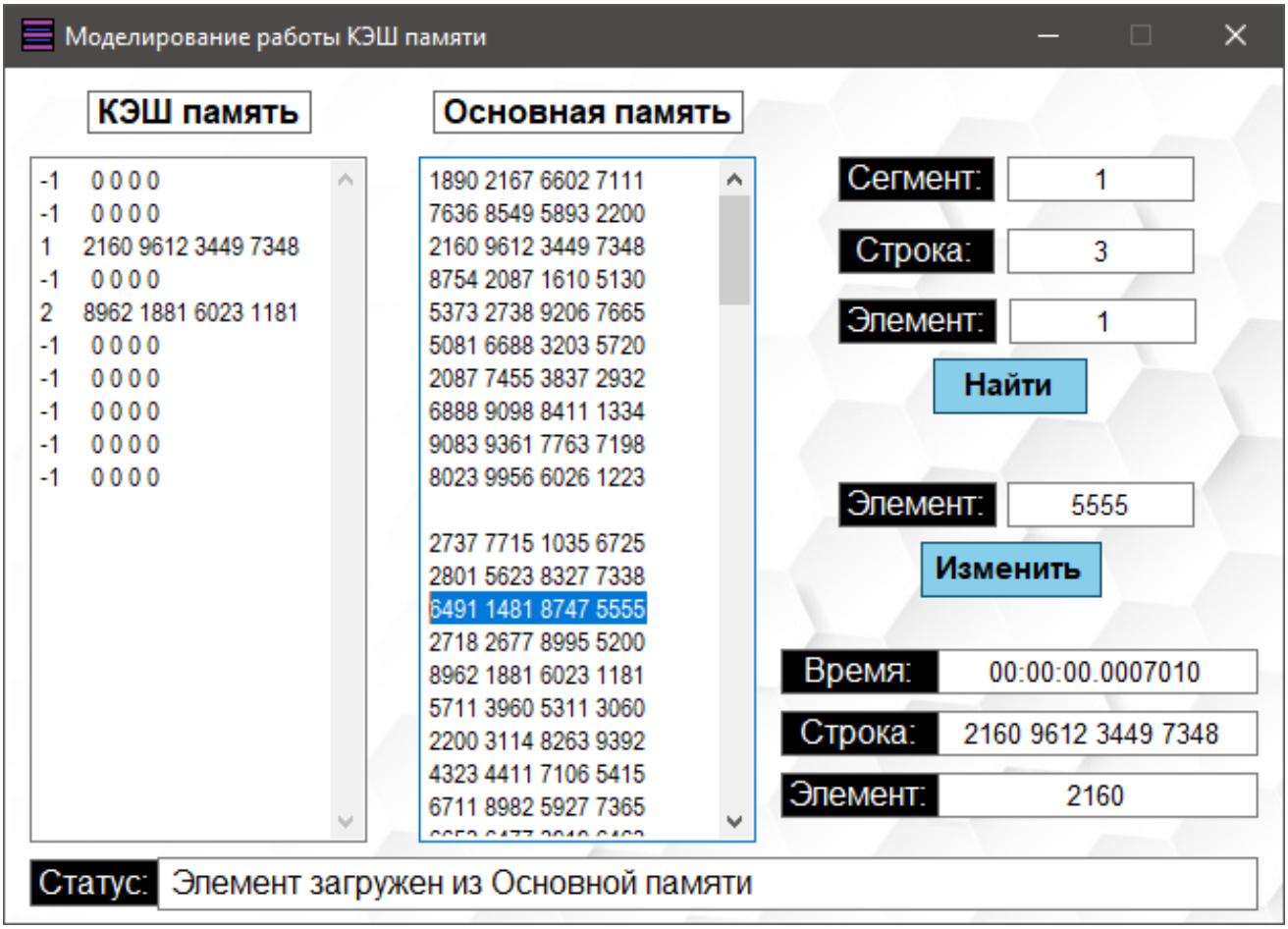
Изменение значения элемента, загруженного из КЭШ-памяти:



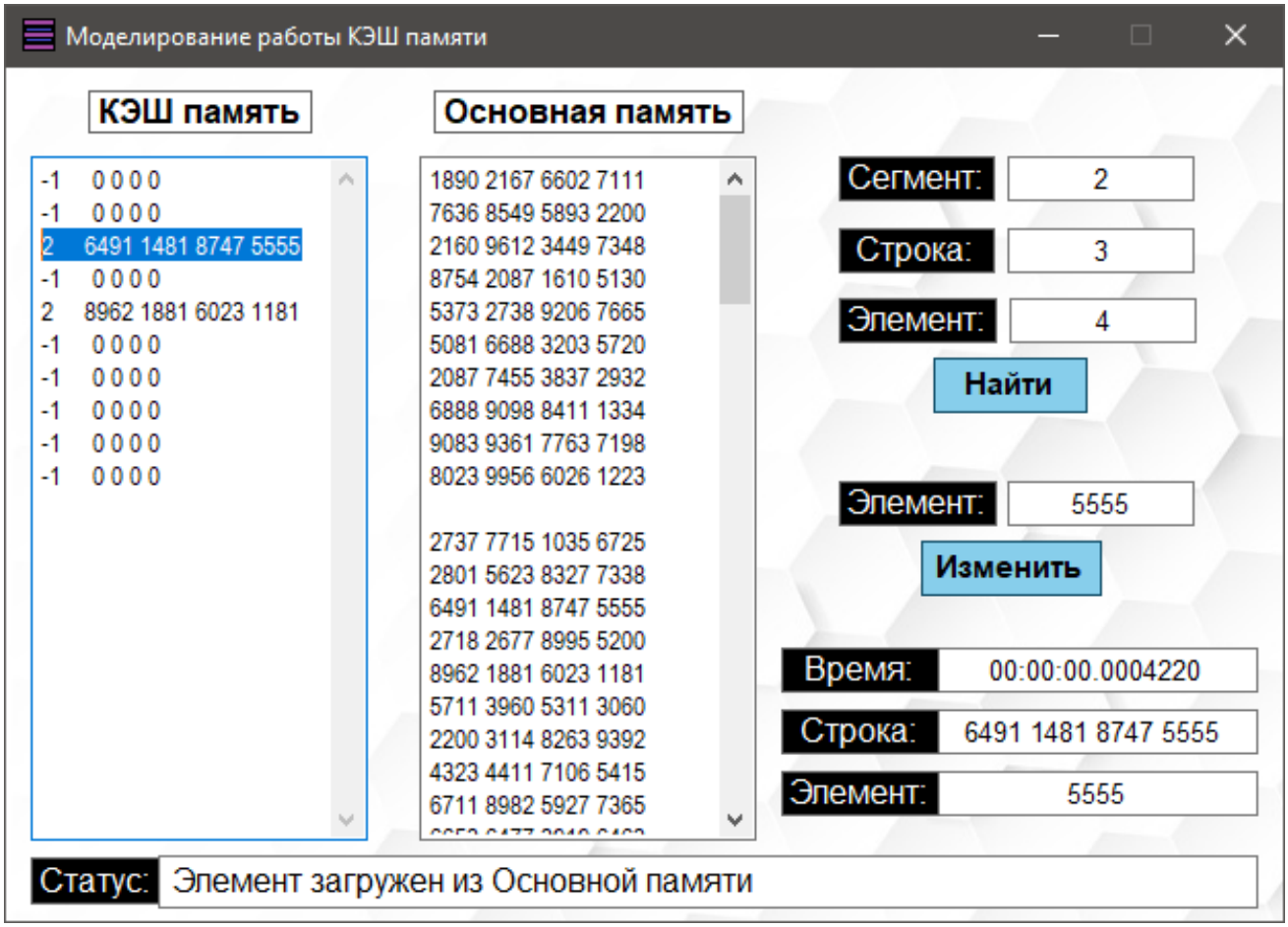
Загрузка строки из Основной памяти при том, что место в КЭШ-памяти, занятое другой строкой, освободилось:



Строка, изменённая ранее в КЭШ-памяти, поместилась обратно в ОП:



Загрузка изменённого ранее элемента из Основной памяти:



КОД ПРОГРАММЫ

- MainForm.cs

```
using System;
using System.Diagnostics;
using System.Windows.Forms;

namespace CacheWork
{
    public partial class MainForm : Form
    {
        static int segmentsCount = 10;
        static int stringsCount = 10;
        static int elementsCount = 4;

        Controller Data = new Controller(segmentsCount, stringsCount, elementsCount);
        Stopwatch Time = new Stopwatch();

        int[] stringData;
        int I_adress, J_adress, K_adress;
        int Values;

        public MainForm()
        {
            InitializeComponent();
            PrintMainMemory();
            PrintCache();
        }

        private void searchButton_Click(object sender, EventArgs e)
        {
            try
            {
                I_adress = Convert.ToInt32(segmentTextBox.Text);
                J_adress = Convert.ToInt32(stringTextBox.Text);
                K_adress = Convert.ToInt32(elementTextBox.Text);

                Time.Start();
                Values = Data[I_adress, J_adress, K_adress];
                Time.Stop();

                UpdateStatus(Values);

                PrintMainMemory();
                PrintCache();
            }
            catch
            {
                MessageBox.Show("Введены неккоректные данные.", "Ошибка!",
                    MessageBoxButtons.OK, MessageBoxIcon.Warning);
            }
        }

        private void changeButton_Click(object sender, EventArgs e)
        {
            try
            {
                I_adress = Convert.ToInt32(segmentTextBox.Text);
                J_adress = Convert.ToInt32(stringTextBox.Text);
                K_adress = Convert.ToInt32(elementTextBox.Text);
                Values = Convert.ToInt32(newElementTextBox.Text);

                Time.Start();
                Data[I_adress, J_adress, K_adress] = Values;
                Time.Stop();

                UpdateStatus(Values);
```

```
            PrintMainMemory();
            PrintCache();
        }
        catch
        {
            MessageBox.Show("Введены неккоректные данные.", "Ошибка!",
                MessageBoxButtons.OK, MessageBoxIcon.Warning);
        }
    }

    void PrintCache()
    {
        cacheTextBox.Text = "";
        for (int i = 1; i <= stringsCount; i++)
        {
            stringData = Controller.CacheController[i];
            cacheTextBox.Text += Controller.CacheController.GetTag(i);
            cacheTextBox.Text += "     ";

            for (int k = 0; k < elementsCount; k++)
            {
                cacheTextBox.Text += stringData[k] + " ";
            }

            cacheTextBox.Text += Environment.NewLine;
        }
    }

    void PrintMainMemory()
    {
        mainMemoryTextBox.Text = "";
        for (int i = 1; i <= segmentsCount; i++)
        {
            for (int j = 1; j <= stringsCount; j++)
            {
                stringData = Controller.MainMemoryController.GetString(i, j);
                for (int k = 0; k < elementsCount; k++)
                {
                    mainMemoryTextBox.Text += stringData[k] + " ";
                }
                mainMemoryTextBox.Text += Environment.NewLine;
            }
            mainMemoryTextBox.Text += Environment.NewLine;
        }
    }

    void UpdateStatus(int el)
    {
        if (Data.FromCache)
        {
            statusTextBox.Text = " Элемент загружен из КЭШ памяти";
        }
        else
        {
            statusTextBox.Text = " Элемент загружен из Основной памяти";
        }

        elementFromTextBox.Text = el.ToString();
        timeTextBox.Text = Time.Elapsed.ToString();
        Time.Reset();

        stringFromTextBox.Text = "";
        for (int i = 1; i <= elementsCount; i++)
        {
            stringFromTextBox.Text += Controller.CacheController[J_adress, i] + " ";
        }
    }
}
```

```
}
```

- Controller.cs

```
namespace CacheWork
{
    class Controller
    {
        public static MainMemory MainMemoryController;
        public static Cache CacheController;
        public bool FromCache;
        int StringsCount;

        public Controller(int segmentsCount, int stringsCount, int elementsCount)
        {
            StringsCount = stringsCount;
            MainMemoryController = new MainMemory(segmentsCount, stringsCount, elementsCount);
            CacheController = new Cache(stringsCount, elementsCount);
        }

        public int this[int i, int j, int k]
        {
            get
            {
                TagValidation(i, j);
                return CacheController[j, k];
            }
            set
            {
                TagValidation(i, j);
                CacheController[j, k] = value;
            }
        }

        // Проверяет совпадают ли требуемый и текущий теги строки
        void TagValidation(int i, int j)
        {
            FromCache = true;

            if (i != CacheController.GetTag(j))
            {
                FromCache = false;

                if (CacheController.GetTag(j) != -1)
                    MainMemoryController.SetString(CacheController.GetTag(j), j,
CacheController[j]);

                CacheController[j] = MainMemoryController.GetString(i, j);
                CacheController.SetTag(j, i);
            }
        }
    }
}
```

- MainMemory.cs

```
using System;
using System.IO;

namespace CacheWork
{
    class MainMemory
    {
        BinaryWriter BinaryWrite;
        BinaryReader BinaryRead;
```

```
        int SegmentsCount,
            StringsCount,
            ElementsCount;

        public MainMemory(int segmentsCount, int stringsCount, int elementsCount)
        {
            Random values = new Random();
            SegmentsCount = segmentsCount;
            StringsCount = stringsCount;
            ElementsCount = elementsCount;

            using (BinaryWrite = new BinaryWriter(new FileStream("Memory.ini", FileMode.Create)))
            {
                for (int i = 0; i < SegmentsCount; i++)
                {
                    BinaryWrite.Write((char)10);

                    for (int j = 0; j < StringsCount; j++)
                    {
                        for (int k = 0; k < ElementsCount; k++)
                        {
                            BinaryWrite.Write(values.Next(1000, 9999));
                            BinaryWrite.Write(' ');
                        }
                        BinaryWrite.Write((char)10);
                    }
                }
            }
        }

        void Positioning(int segment, int str, IDisposable WriteRead)
        {
            segment--;
            str--;

            int position = (segment + 1) + // Отступы между блоками
                (segment * (StringsCount * ((ElementsCount * 4) + 5))) +  // Пропуск эл-та до
нужного сегмента
                (str * ((ElementsCount * 4) + 5)); // Пропуск эл-та до нужной строки

            // Позиция каретки с учетом размеров
            switch (WriteRead)
            {
                case BinaryWriter writer:
                    writer.BaseStream.Position = position;
                    break;
                case BinaryReader reader:
                    reader.BaseStream.Position = position;
                    break;
            }
        }

        // Получить строку в сегменте
        public int[] GetString(int segment, int str)
        {
            int[] temp = new int[ElementsCount];

            using (BinaryRead = new BinaryReader(new FileStream("Memory.ini", FileMode.Open)))
            {
                Positioning(segment, str, BinaryRead);

                for (int i = 0; i < ElementsCount; i++)
                {
                    temp[i] = BinaryRead.ReadInt32();
                    BinaryRead.BaseStream.Position++;
                }
            }
```

```
            return temp;
        }

        // Записать строку temp в строку str в сегменте segment
        public void SetString(int segment, int str, int[] temp)
        {
            using (BinaryWrite = new BinaryWriter(new FileStream("Memory.ini", FileMode.Open)))
            {
                Positioning(segment, str, BinaryWrite);

                for (int i = 0; i < ElementsCount; i++)
                {
                    BinaryWrite.Write(temp[i]);
                    BinaryWrite.Write(' ');
                }
            }
        }
    }
}
```

- Cache.cs

```
namespace CacheWork
{
    class Cache
    {
        (int Tag, int[] Line)[] Page;

        public Cache(int stringsCount, int elementsCount)
        {
            Page = new (int Tag, int[] Line)[stringsCount];
            for (int i = 0; i < Page.Length; i++)
            {
                Page[i].Line = new int[elementsCount];
                Page[i].Tag = -1;
            }
        }

        // Получить или изменить элемент
        public int this[int i, int j]
        {
            get => Page[i - 1].Line[j - 1];
            set => Page[i - 1].Line[j - 1] = value;
        }

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

        // Получить или изменить тэг
        public int GetTag(int i) => Page[i - 1].Tag;
        public void SetTag(int i, int newTag) => Page[i - 1].Tag = newTag;
    }
}
```