САНКТ–ПЕТЕРБУРГСКОЕ ГОСУДАРСТВЕННОЕ
БЮДЖЕТНОЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

«КОЛЛЕДЖ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ»

ОТЧЕТ ПО УЧЕБНОЙ ПРАКТИКЕ

УП01

Специальность 09.02.07
«Информационные системы и программирование»
Квалификация «Программист»

Руководитель учебной практики:
Смирнова И. П.
Другие преподаватели:
Матысик И. А.
Полякова А. Н.
Фомин А. В.
Выполнил студент группа 493:
Сидоров антон дмитриевич

Санкт–Петербург, 2023

СОДЕРЖАНИЕ

1 УП01.01. Разработка программных модулей

2 УП01.02. Поддержка и тестирование программных модулей

2.1 Задание расчёт сырья

2.1.1 Задание

Для того чтобы в производстве могли быстро и одинаково рассчитывать количество необходимого сырья для производства той или иной продукции, необходимо разработать библиотеку классов.

Чтобы система правильно интегрировалась вам необходимо обязательно следовать правилам именования библиотек, классов и методов в них. В случае ошибок в рамках именования ваша работа не может быть проверена и ваш результат не будет зачтен. Классы и методы должны содержать модификатор public (если это реализуемо в рамках платформы), чтобы внешние приложения могли получить к ним доступ.

В качестве названия для библиотеки необходимо использовать: WSUniversalLib. Вам необходимо загрузить исходный код проекта с библиотекой в отдельный репозиторий с названием, совпадающим с названием проекта.

2.1.2 Спецификация метода

Метод должен принимать идентификатор типа продукции, идентификатор типа материала, количество необходимой продукции для производства, ширину продукции и длину продукции, а возвращать целое число – количество необходимого сырья с учетом возможного брака сырья.

Библиотека классов	WSUniversalLib.dll
Название класса	Calculation

Название метода	GetQuantityForProduct()
Входящие обязательные параметры	int productType, int materialType, int count, float width, float length
Возвращаемые параметры	int

2.1.3 Программный код

Программный код находится в репозитории на GitHub по адресу https://github.com/AntonSidorov1/PracticeTesting_sidorov493.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace WSUniversalLib
{
    public class Calculation
    {
        static List<Product> products = new List<Product>(new Product[] { new Product(1, 1.1f), new Product(2, 2.5f), new Product(3, 8.43f) });
        static List<Material> materials = new List<Material>(new Material[] {new Material(1, 0.3f), new Material(2, 0.12f) });

        public static int GetQuantityForProduct(float productType, float materialType, int count, float width, float length)
        {
            float materialCount = productType * width * length;
            materialCount *= count;
            materialCount += (materialCount * materialType);
            float materialCount1 = (float)Math.Round(materialCount);
            //materialCount = (float)Math.Round(materialCount, MidpointRounding.AwayFromZero);
            materialCount = materialCount1 >= materialCount ? materialCount1 : materialCount1 + 1;

            return (int)materialCount;
        }

        public static int GetQuantityForProduct(int productType, int materialType, int count, float width, float length)
        {
            Product product = products.Find(p => p.ProductType == productType);
            Material material = materials.Find(p => p.MaterialType == materialType);
            return GetQuantityForProduct(product.MaterialCount, material.Defect, count, width, length);
        }
    }
}
```

2.2 Unit–tests

Первые 2 теста будут для метода *public static int GetQuantityForProduct(float productType, float materialType, int count, float width, float length)*, поскольку, как видно из кода, который приведён в предыдущем разделе, этот метод вызывается из того метода, который на самом деле нужно тестировать, а значит, от того, насколько правильно работает этот метод, зависит работа второго метода.

Первый unit-тест:

```
        /// <summary>
            ///  Результат расчётов без использования метода совпадает с результатом рассчётов без использования метода
            /// </summary>
            [TestMethod]
            public void TheResultCalculatedWithoutApplyingThisMethodCoincidesWithTheResultCalculatedUsingTheMethod()
            {
                float productType = 8.43f;
                float materialType = 0.003f;
                int count = 15;
                float width = 20;
                float length = 45;
                float materialCount = productType * width * length;
                materialCount *= count;
                materialCount += (materialCount * materialType);

                float materialCount4 = materialCount;
                float materialCount1 = (float)Math.Round(materialCount);
                materialCount = materialCount1 >= materialCount ? materialCount1 : materialCount1 + 1;
                //materialCount     =     (float)Math.Round(materialCount, MidpointRounding.AwayFromZero);

                int materialCount2 = (int)materialCount;
                int                    materialCount3                    = Calculation.GetQuantityForProduct(productType, materialType, count, width, length);
                Assert.AreEqual(materialCount2,   materialCount3,   0.001, materialCount2 + $"({materialCount4})" + "!=" + materialCount3);
            }
```

Второй unit-тест:

```
            /// <summary>
            ///  Для 15 единиц продукции шириной 20 и длиной 15, где количества материала на 1 единицу требуется 8,43 с процентом деффекта 0.03, требуется 114147 единиц материала
            /// </summary>
            [TestMethod]
            public void For15ProductUnits20WidthAnd45LongProductType8And43MaterialType03TheMaterialQuantityIs114147()
            {
```

```
                float productType = 8.43f;
                float materialType = 0.003f;
                int count = 15;
                float width = 20;
                float length = 45;

                int materialCount = Calculation.GetQuantityForProduct(productType, materialType, count, width, length);
                Assert.AreEqual(114147, materialCount, 0.001, materialCount + "!=" + 114147);
            }
```

Оба теста проводятся на правильность результата. Оба теста были успешными, как показано на рисунке 1.

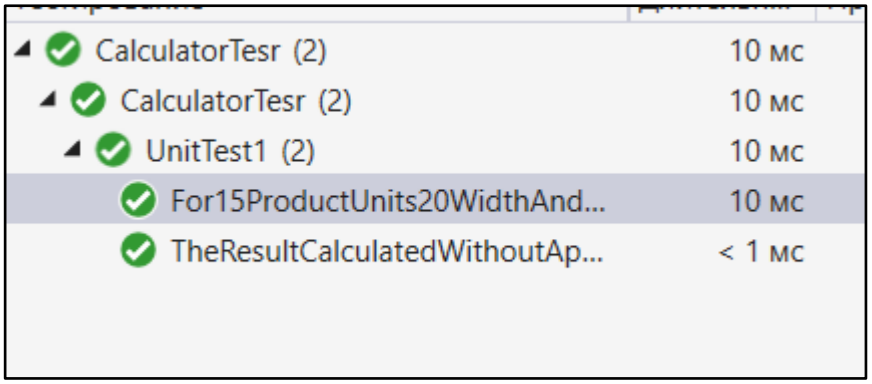

Рисунок 1 – Unit-тесты

Теперь, переходим ко второму методу (*GetQuantityForProduct(int productType, int materialType, int count, float width, float length)*), который вычисляет результат на основе данных, приведённых в коллекциях *products* (вид продукции: тип продукции и количество материала) и *materials* (материалы: тип материала и процент брака)

Первый unit-тест:

```
        /// <summary>
        /// для тестирования случая передачи несуществующего типа продукции.
        /// </summary>
        [TestMethod]
        public void GetQuantityForProduct_NonExistentProductType()
        {
```

```
            try
            {
                int materialCount = Calculation.GetQuantityForProduct(4, 2, 12, 14, 34);
                Assert.AreEqual(114147, materialCount, 0.001, materialCount + "!=" + 114147);
            }
            catch (Exception e)
            {
                NullReferenceException ex = new NullReferenceException();
                Assert.AreEqual(ex.Message, e.Message);
            }
        }
```

Второй unit-тест:

```
        /// <summary>
        /// Для 5 единиц продукции 1 типа шириной 4 и длиной 2 материала 2 типа требуется 45 единиц материала
        /// </summary>
        [TestMethod]
        public void For5UnitsOfProductType1WithAWidthOf4AndALengthOf2MaterialsOfType2Is45UnitsOfMaterialAreRequired()
        {


            try
            {
                int materialCount = Calculation.GetQuantityForProduct(1, 2, 4, 2, 5);
                Assert.AreEqual(45, materialCount, 0.001, materialCount + "!=" + 114147);
            }
            catch (Exception e)
            {
                NullReferenceException ex = new NullReferenceException();
                Assert.AreEqual(ex.Message, e.Message);
            }
        }
```

Третий unit-тест:

```
        /// <summary>
        /// Для 50 единиц продукции 3 типа шириной 40 и длиной 80 материала 1 типа требуется 1352847 единиц материала
        /// </summary>
        [TestMethod]
        public void For50UnitsOfProduct3TypesWithAWidthOf40AndALengthOf80MaterialOfType1Is1352847UnitsOfMaterialAreRequired()
        {


            try
            {
                int materialCount = Calculation.GetQuantityForProduct(3, 1, 40, 80, 50);
                Assert.AreEqual(1352847, materialCount, 0.001, materialCount + "!=" + 1352847);
            }
            catch (Exception e)
            {
                NullReferenceException ex = new NullReferenceException();
                Assert.AreEqual(ex.Message, e.Message);
            }
        }
```

Четвёртый unit-тест:

```
        /// <summary>
        /// Для 150 единиц продукции 2 типа шириной 25 и длиной 75 материала 2 типа требуется 703969 единиц материала
        /// </summary>
        [TestMethod]
        public void For150UnitsOfProductType2WithAWidthOf25AndALengthOf75MaterialOfType2Is703969UnitsOfMaterialAreRequired()
        {


            try
            {
                int materialCount = Calculation.GetQuantityForProduct(2, 2, 25, 75, 150);
```

```
                    Assert.AreEqual(703969,      materialCount,      0.001,
materialCount + "!=" + 703969);
                }
                catch (Exception e)
                {
                    NullReferenceException         ex         =         new
NullReferenceException();
                    Assert.AreEqual(ex.Message, e.Message);
                }
            }
```

Эти все тесты были пройдены успешно, как показано на рисунке 2.

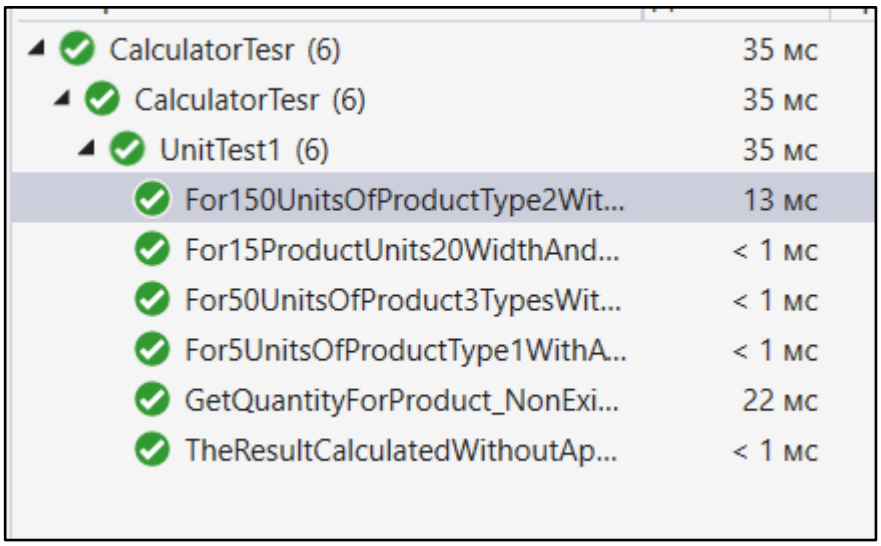

Рисунок 2 – Unit-тесты

Теперь, напишем Unit-тесты, которые не будут выполнены успешно.

```
            /// <summary>
            ///  Получить  количество  материала  при  несуществующем  типе
материала
            /// </summary>
            [TestMethod]
            public void GetQuantityOfMaterialWhenMaterialTypeDoesNotExist()
            {


                    int                    materialCount                    =
Calculation.GetQuantityForProduct(2, 12, 12, 14, 34);
                    Assert.AreEqual(114147,      materialCount,      0.001,
materialCount + "!=" + 114147);
```

```
        }

        /// <summary>
        /// Получить количество материала, если тип продукции - текст
        /// </summary>
        [TestMethod]
        public void GetMaterialQuantityIfproductTypeIsText()
        {



            int                     materialCount                   =
Calculation.GetQuantityForProduct(Convert.ToInt32("abc"), 12, 12, 14, 34);
            Assert.AreEqual(114147, materialCount, 0.001, materialCount
+ "!=" + 114147);

        }

        /// <summary>
        ///  Получить  количество  материала,  если  тип  продукции  -  Не
целое число
        /// </summary>
        [TestMethod]
        public void GetMaterialQuantityIfProductTypeIsNotAnInteger()
        {



            int                     materialCount                   =
Calculation.GetQuantityForProduct(Convert.ToInt32(5.12.ToString()),  12,  12,
14, 34);
            Assert.AreEqual(114147, materialCount, 0.001, materialCount
+ "!=" + 114147);

        }
```

Все 3 теста были провалены, как показано на рисунке 3.

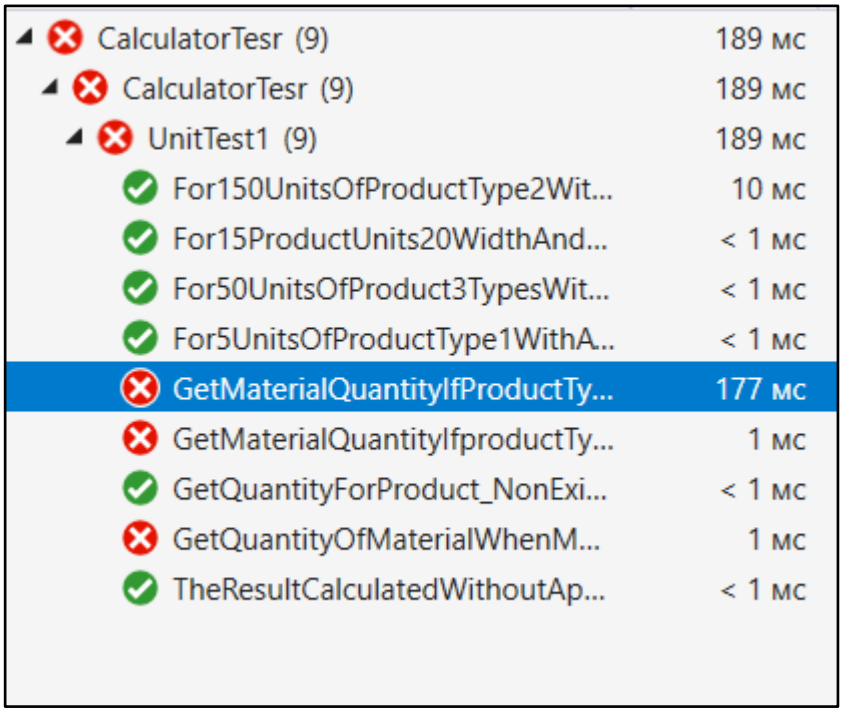

Рисунок 3 – Unit-тесты

Для того, чтобы тесты были успешными изменим код следующим образом:

```
        /// <summary>
        /// Получить количество материала при несуществующем типе материала
        /// </summary>
        [TestMethod]
        public void GetQuantityOfMaterialWhenMaterialTypeDoesNotExist()
        {

            try
            {

                int materialCount = Calculation.GetQuantityForProduct(2, 12, 12, 14, 34);
                Assert.AreEqual(114147, materialCount, 0.001, materialCount + "!=" + 114147);
            }
            catch (Exception e)
            {
                NullReferenceException ex = new NullReferenceException();
                Assert.AreEqual(ex.Message, e.Message);
            }
        }
```

```
        /// <summary>
        /// Получить количество материала, если тип продукции - текст
        /// </summary>
        [TestMethod]
        public void GetMaterialQuantityIfproductTypeIsText()
        {
            try
            {


                int materialCount =
Calculation.GetQuantityForProduct(Convert.ToInt32("abc"), 12, 12, 14, 34);
                Assert.AreEqual(114147, materialCount, 0.001,
materialCount + "!=" + 114147);
                }
                catch (Exception e)
                {
                    FormatException ex = new FormatException("Входная
строка имела неверный формат.");
                    Assert.AreEqual(ex.Message, e.Message);
                }
            }

            /// <summary>
            /// Получить количество материала, если тип продукции - Не
целое число
            /// </summary>
            [TestMethod]
            public void GetMaterialQuantityIfProductTypeIsNotAnInteger()
            {
                    try
                    {


                        int materialCount =
Calculation.GetQuantityForProduct(Convert.ToInt32(5.12.ToString()), 12, 12,
14, 34);
                Assert.AreEqual(114147, materialCount, 0.001, materialCount
+ "!=" + 114147);
                }
                catch (Exception e)
                {
                    FormatException ex = new FormatException("Входная
строка имела неверный формат.");
                    Assert.AreEqual(ex.Message, e.Message);
                }
            }
```

Теперь, все тесты успешны, как показано на рисунке 4.

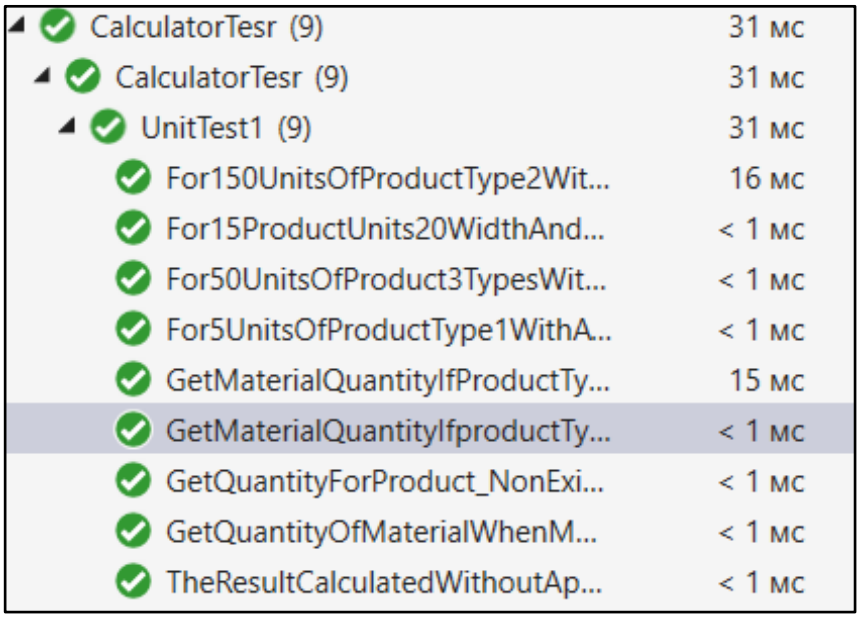

Рисунок 4 – Unit-тесты

Ещё один тест будет на вложенный метод. Его нынешний код:

```
        public static int GetQuantityForProduct(float productType, float
materialType, int count, float width, float length)
        {
            float materialCount = productType * width * length;
            materialCount *= count;
            materialCount += (materialCount * materialType);
            float materialCount1 = (float)Math.Round(materialCount);
            //materialCount      =      (float)Math.Round(materialCount,
MidpointRounding.AwayFromZero);
            materialCount   =   materialCount1   >=   materialCount   ?
materialCount1 : materialCount1 + 1;

            return (int)materialCount;
        }
```

Про результат данного метода, вопрос: Что будет при вводе отрицательного значения количества продукции, длины или ширины? Unit-тест:

```
        /// <summary>
        /// Получить количество материала при отрицательном значении
количества продукции, длины или ширины
        /// </summary>
        [TestMethod]
        public                                                    void
GetMaterialQuantityWhenProductQuantityOrLengthOrWidthIsNegative()
        {
            ArgumentException ex = new ArgumentException();
```

```
            try
            {

                int                    materialCount                    =
Calculation.GetQuantityForProduct(1, 1, -12, -14, -34);
                Assert.AreEqual(ex.Message, materialCount);
            }
            catch (Exception e)
            {
                //ArgumentException ex = new ArgumentException();
                Assert.AreEqual(ex.Message, e.Message);
            }
        }
```

Данный тест не был успешным, как показано на рисунке 5.

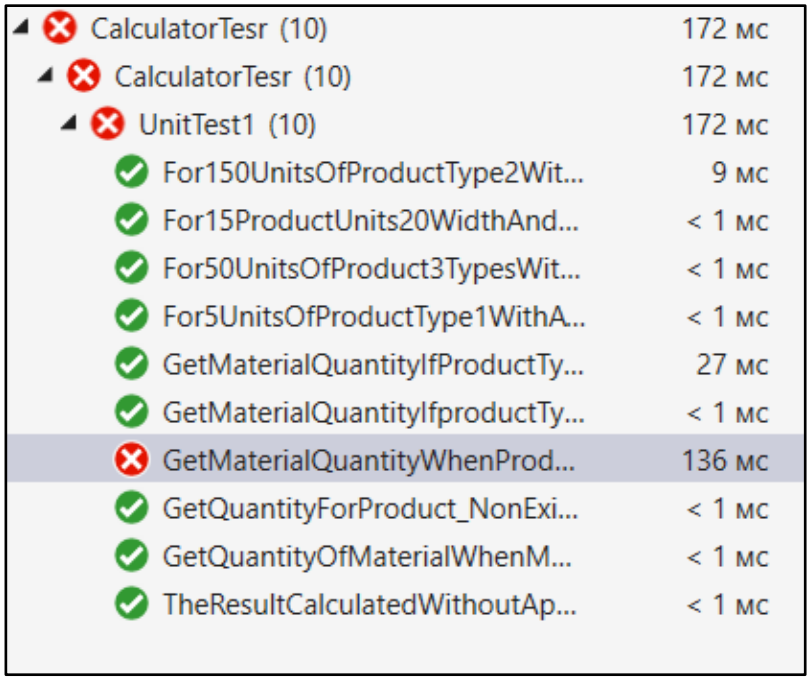

Рисунок 5 – Unit-тесты

Для этого изменим код метода следующим образом:

```
        public static int GetQuantityForProduct(float productType,
float materialType, int count, float width, float length)
        {
            if(productType < 0 || materialType <0 || count < 0 || width
< 0 || length < 0)
            {
                throw new ArgumentException();
            }
            float materialCount = productType * width * length;
            materialCount *= count;
            materialCount += (materialCount * materialType);
            float materialCount1 = (float)Math.Round(materialCount);
```

```
                //materialCount      =      (float)Math.Round(materialCount,
MidpointRounding.AwayFromZero);
                materialCount   =   materialCount1   >=   materialCount   ?
materialCount1 : materialCount1 + 1;

                return (int)materialCount;
            }
```

Теперь тест выполнен успешно, как показано на рисунке 6.

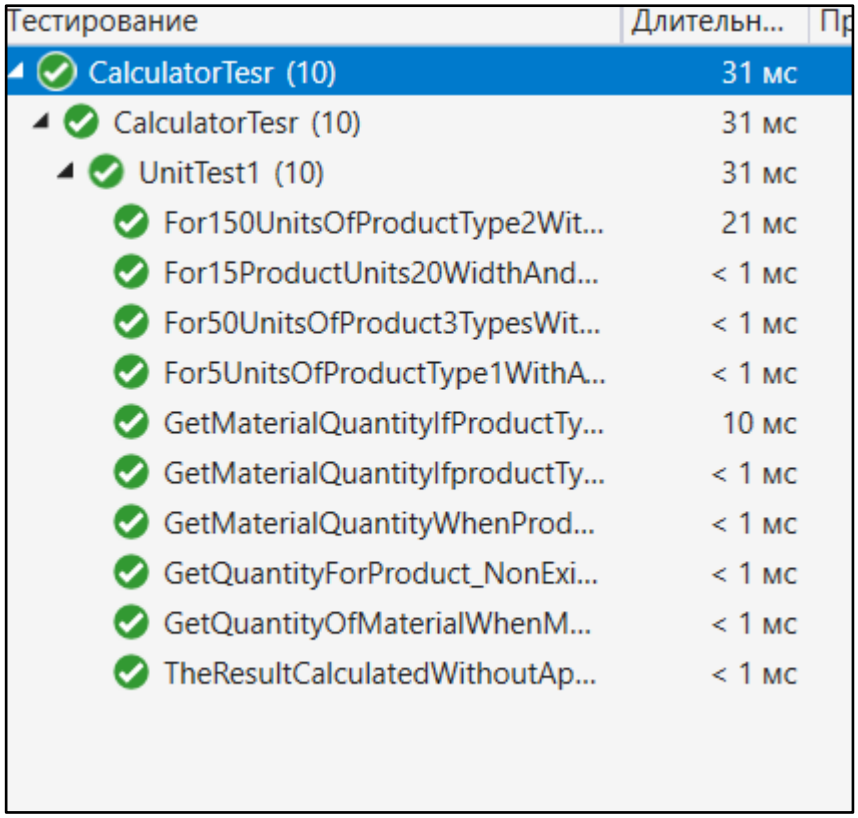

Рисунок 6 – Unit-тесты

2.3 Test case

3 УП01.03. Разработка мобильных приложений

3.1 Цель работы

3.2 Название работы

Разработка чата

3.3 Макеты окон

3.1.1 Примечания

Окна в приложении могут отличаться от своих макетов.

3.1.2 Макеты окон

Макеты окон приложения показаны на рисунке 1.

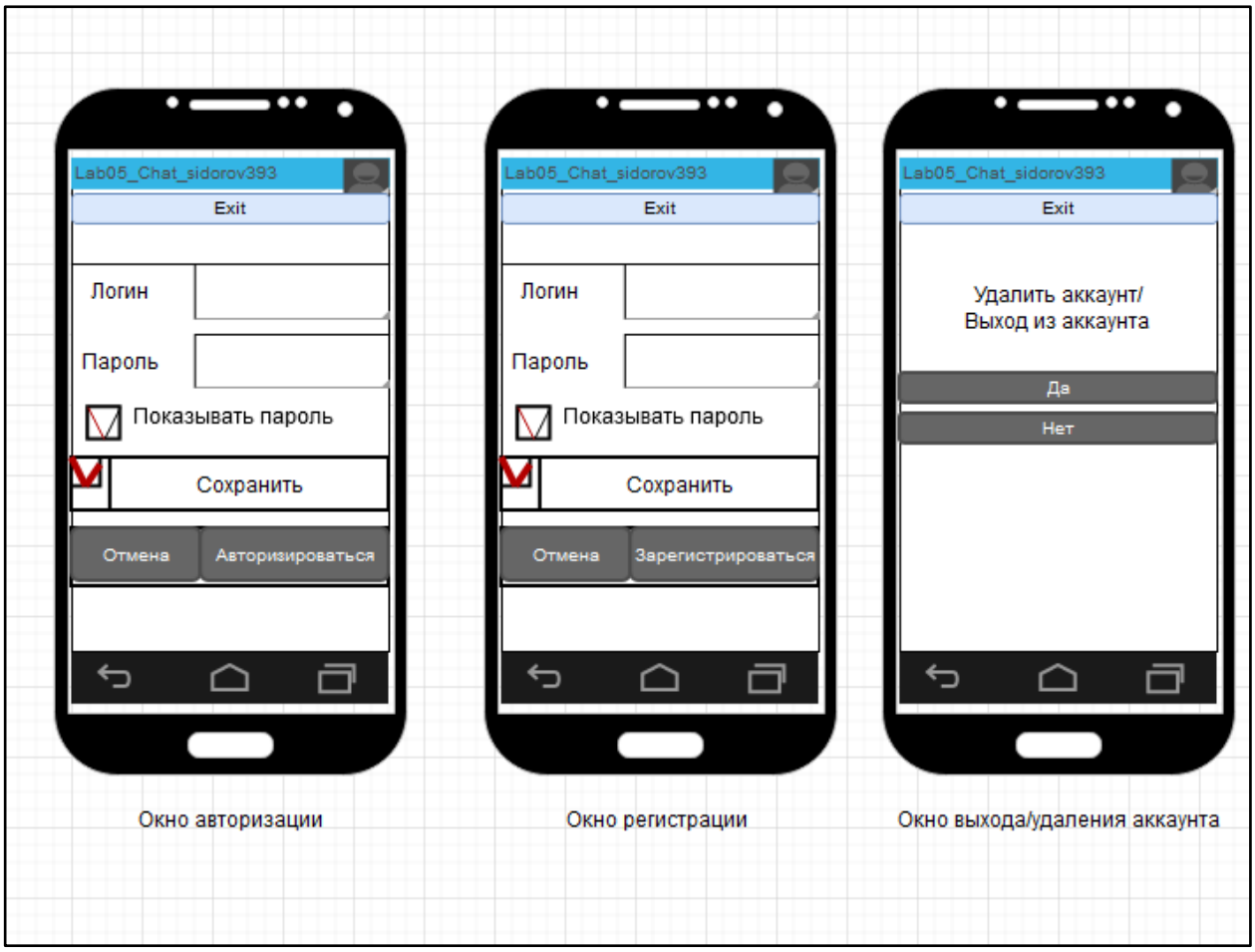

Рисунок 1 – Макеты окон

Также, есть, ещё окна, макеты которых показаны на рисункуе 2.

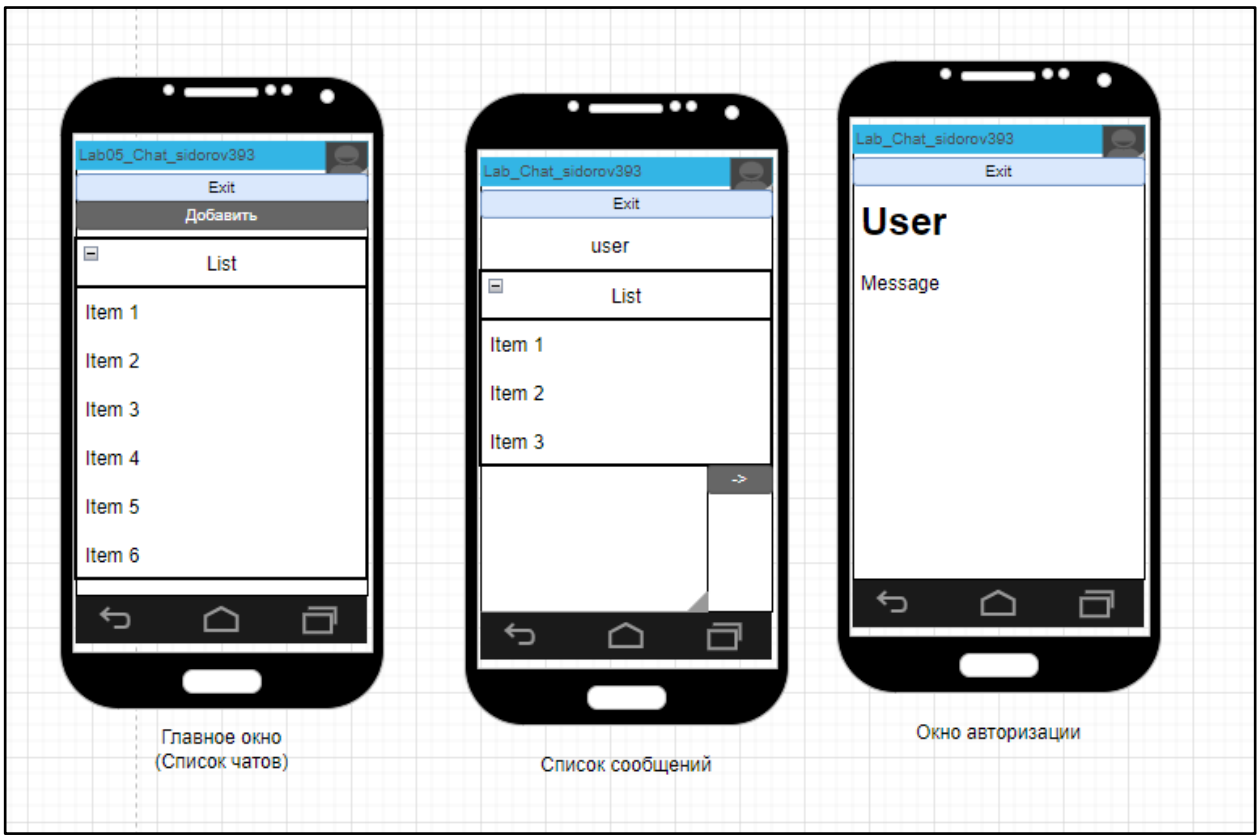

Рисунок 2 – Макеты окон

Окно главного и редактора url–ссылки меню показано на рисунке 3.

Рисунок 3 – Главное меню

3.1.3 Описание окон (функции)

Окно авторизации:

- Поле ввода логина
- Поле ввода пароля
- Флажок для показа/скрытия пароля
- Флажок для сохранения/несохранения данных на устройстве
- Кнопка отмены входа
- Кнопка входа в аккаунт

Окно регистрации:

- Поле ввода логина
- Поле ввода пароля
- Флажок для показа/скрытия пароля
- Флажок для сохранения/несохранения данных на устройстве
- Кнопка отмены входа
- Кнопка создания аккаунта и входа в него

Окно выхода из аккаунта:

- Предупреждающая надпись
- Кнопки подтверждения (да/нет)

Окно удаления аккаунта:

- Предупреждающая надпись
- Кнопки подтверждения (да/нет)

Окно списка чатов:

- Список чатов, с возможностью перейти к чату
- Кнопка добавления чата, перенаправляющая к списку аккаунтов

Окно списка аккаунтов:

- Список аккаунтов, с возможностью перейти к аккаунту и начать с ним чат

Окно редактора url–ссылки:

- Поле ввода url–адреса
- Кнопка сохранения url–адреса

- Кнопка отмены сохранения url–адреса

Главное меню (Первое открывающееся окно):

- Отображаемый url–адрес
- Отображаемый ключ сессии
- Кнопка авторизации, перенаправляющая на окно авторизации
- Кнопка регистрации, перенаправляющая на окно регистрации
- Кнопка выхода из аккаунта
- Кнопка удаления аккаунта
- Кнопка перехода к списку чатов
- Кнопка, перенаправляющая к редактору url–ссылок

Все окна, также, имеют кнопу выхода, перенаправляющую на предыдущее окно.

3.4 Диаграмма базы данных

База данных предназначена для локального хранения данных об аккаунте и о url–ссылок.

Диаграмма базы данных представлена на рисунке 5.

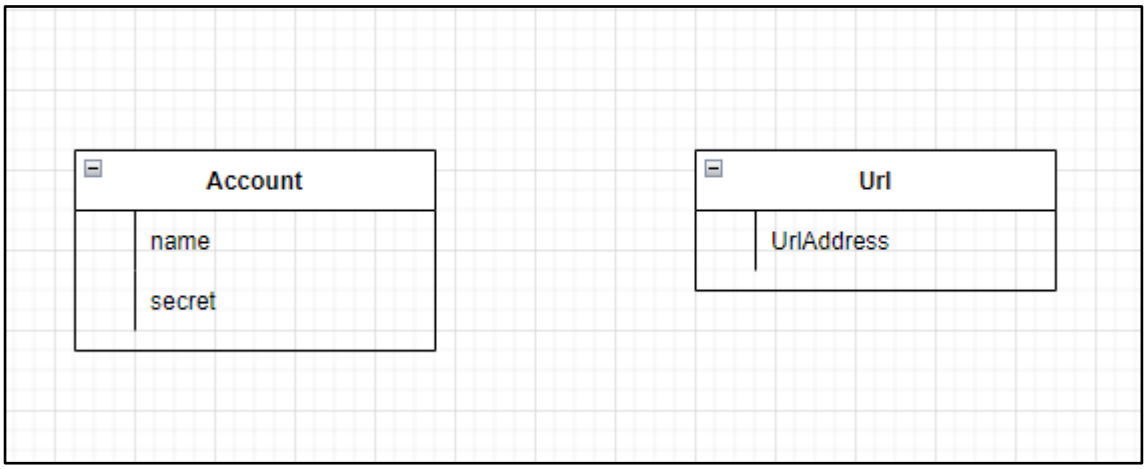

Рисунок 5 – Диаграмма базы данных

Таблица *Account* предназначена для хранения данных о пользователе: логин (*name*) и пароль (*secret*).

Таблица *UrlAddress* предназначена для хранения адреса сервера с API в поле *UrlAddress*.

3.5 Описание API

Первая функция – *sign_in* с адресом Http://spbcoit.ru/lab/chat/api/rpc/sign_in. Эта функция открывает сессию пользователя при входе в аккаунт. На вход подаются имя пользователя (*name*) и пароль (*password*). На выход выдаётся ключ сессии. Выполнение данной функции на Postman показано на рисунке 6.

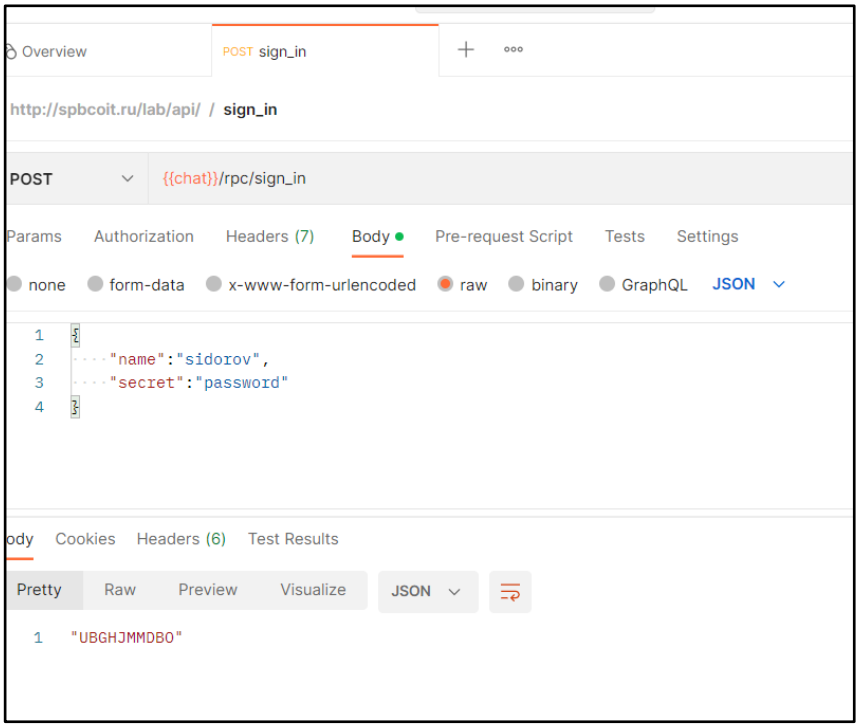

Рисунок 6 – API на Postman

Обратное действие производит *sign_out* с адресом http://spbcoit.ru/lab/chat/api/rpc/sign_out. Это, соответственно, закрытие сессии пользователя при выходе из аккаунта. На вход подаётся ключ сессии (*stoken*). На выход выдаётся значение, указывающее, удачно ли было закрытие сессии.

Для создания аккаунта используется функция *register_account*, расположенная по адресу http://spbcoit.ru/lab/chat/api/rpc/register_account.

4 УП01.04. Системное программирование

4.1 Лабораторная работа №1

4.1.1 Цель работы

Изучение архитектуры микропроцессора КР580.

4.1.2 структура МП

Числовые обозначения структуры представлены на рисунке 1.

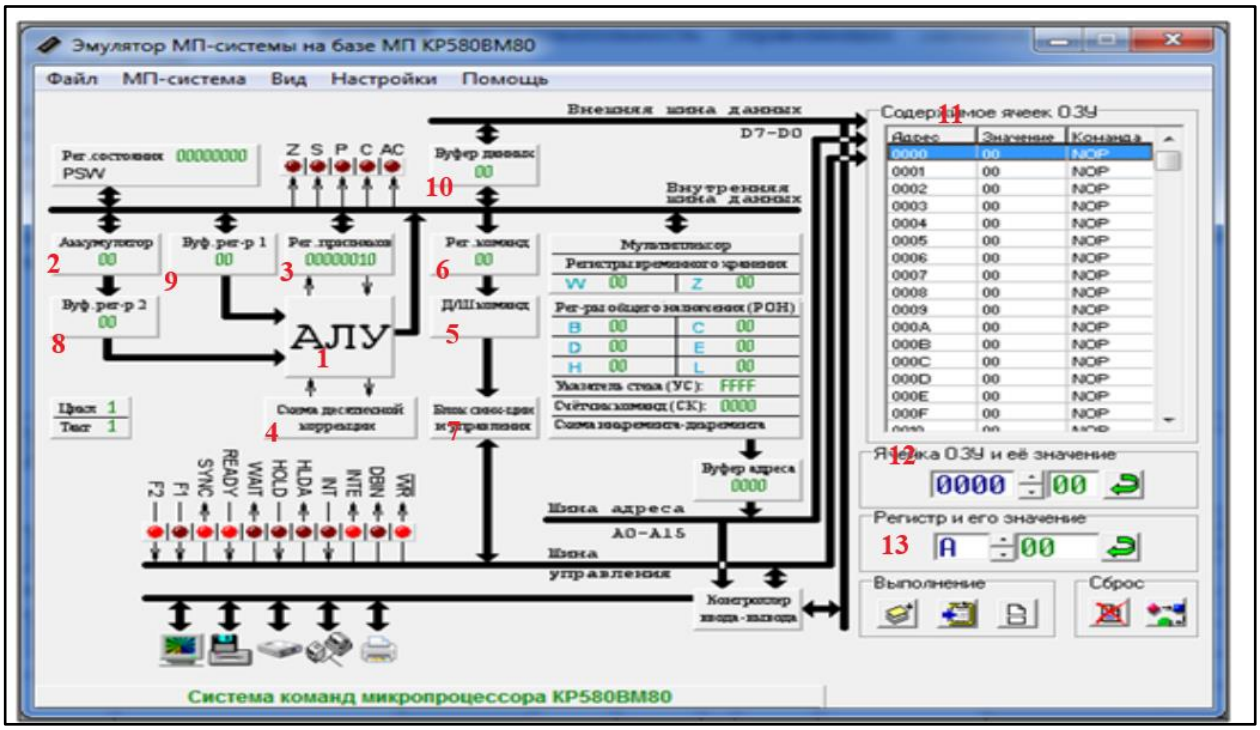

Рисунок 1 – структура МП.

Обозначения (Красным цветом на фотографии):

1 – 8–разрядное арифметико–логическое устройство АЛУ (ALU)

2 – аккумулятор (A)

3 – регистр признаков RS, фиксирующий признаки, вырабатываемые АЛУ в процессе выполнения команды

4 – десятичный корректор (DAA), выполняющий перевод информации из двоичной в двоично–десятичную форму

5 – дешифратор команд (DCU)

6 – регистр команд (IR), предназначенный для хранения первого байта команды, содержащего код операции

7 – схема управления и синхронизации (CU), формирующая последовательности управляющих сигналов для работы ALU и регистров

8 – однонаправленный 16–разрядный буферный регистр адреса (BA)

9 – двунаправленный 8–разрядный буферный регистр данных (BD)

10 – регистр временного хранения операндов (RGb)

11 – Пространство памяти и ввода–вывода

12 – Значения ячеек памяти

13 – Значения регистров

4.1.3 таблица регистров, которые имеет МП

Регистр	Назначение	Разрядность
регистрами общего назначения (РОН)	8–разрядные регистры F, A вместе с 16–разрядными регистрами HL, SP и PC образуют стандартный регистровый набор микропроцессора с аккумулятором. Этот набор расширен четырьмя 8–разрядными регистрами общего назначения (РОН): B, C, D, E, которые в некоторых командах объединяются в 16–разрядные парные регистры BC и DE. Младшими регистрами пары являются соответственно регистры C и E. Введение РОН позволило создать достаточно эффективный микропроцессор с широкими функциональными возможностями.	8 бит
аккумулятор A	используется в подавляющем большинстве команд логической и арифметической отработки. Обычно он адресуется неявно и служит как источником операнда, так и приемником результата. Благодаря этому в командах ВМ80А явно указывается только один операнд.	8 бит
регистр HL	как правило, служит адресным регистром. При косвенной регистровой адресации он хранит 16–разрядный адрес основной памяти. В	16 бит

	этом случае к нему ссылаются с помощью мнемоники M (Memory), например: MOV A, M; содержимое ячейки (HL) заносится в аккумулятор	
Регистры PC и SP	выполняют свою обычную функцию счетчика команд и указателя стека.	

4.1.4 блок–схема функционирования МП во время выполнения команды сложения содержимого аккумулятора и регистра B, имеющая мнемоническое обозначение ADD B.

Данная блок–схема показана на рисунке 1.

Рисунок 2 – Блок–схема

4.1.5 таблица флагов

Обозначение флага	Признак флага	Условие установки флагов
C (Carry)	признак переноса	наличие переноса (при сложении) или заема (при вычитании) из старшего разряда аккумулятора, иначе сбрасывается
M (Minus)	признак отрицательного результата	устанавливается, если знаковый бит результата операции (седьмой разряд аккумулятора) равен 1, иначе сбрасывается
Z (Zero)	признак нуля	устанавливается, если результат операции в аккумуляторе равен нулю, иначе сбрасывается
P (Parity)	признак паритета/четности	устанавливается, если результат операции в аккумуляторе содержит четное число единиц, иначе сбрасывается
AC (Auxiliary Carry)	признак половинного переноса	устанавливается при наличии переноса из третьего разряда аккумулятора в четвертый, иначе сбрасывается

4.2 Лабораторная работа №2. Исследование команд прямой адресации

4.2.1 Цель лабораторной работы

Целью данной работы является ознакомление с командами микропроцессора КР580 для прямой адресации

4.2.2 Программный код

0000	3A	LDA adr	загрузить содержимое в аккумулятор
0001	0A	LDAX B	из ячейки памяти с адресом Ah
0002	00	NOP	
0003	32	STA adr	загрузить содержимое аккумулятора в
0004	0B	DCX B	ячейку памяти с адресом Bh
0005	00	NOP	
0006	00	NOP	
0007	76	HLT	Остановка выполнения программы
0008	00	NOP	
0009	00	NOP	
000A	40	MOV B, B	Значение в ячейке памяти
000B	00	NOP	

000C	00	NOP	
000D	00	NOP	
000E	00	NOP	
000F	00	NOP	
0010	00	NOP	

4.2.3 Ввод операндов при прямой адресации

Скрин ввода операндов и начала работы программы представлен на рисунке 3.

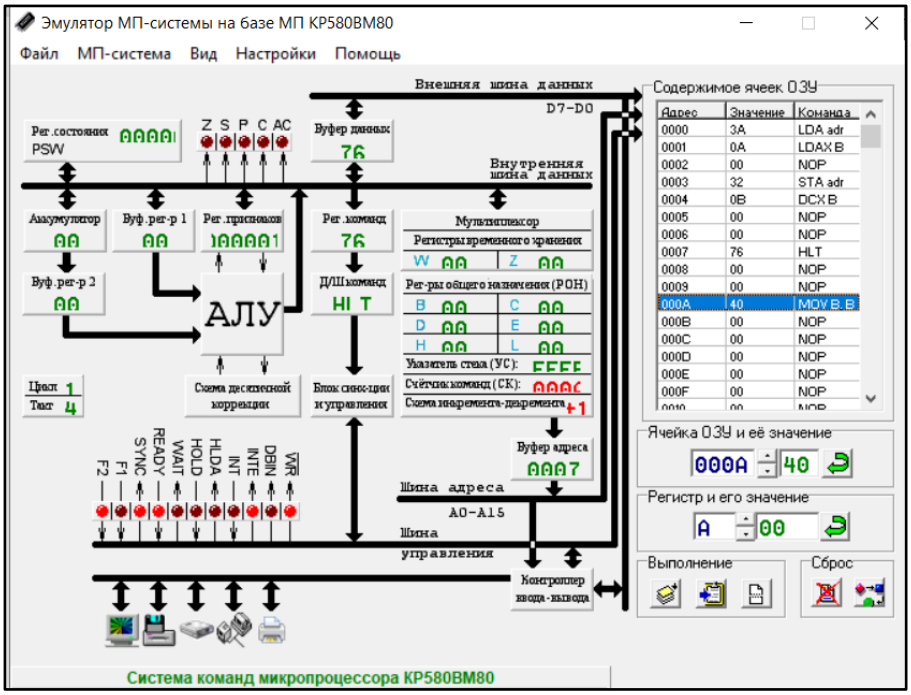

Рисунок 3 – Ввод операндов при прямой адресации

Операнд вводится в ячейку памяти *Ah*.

4.2.4 Результат работы модели при прямой адресации

Скрин результата работы программы представлен на рисунке 4.

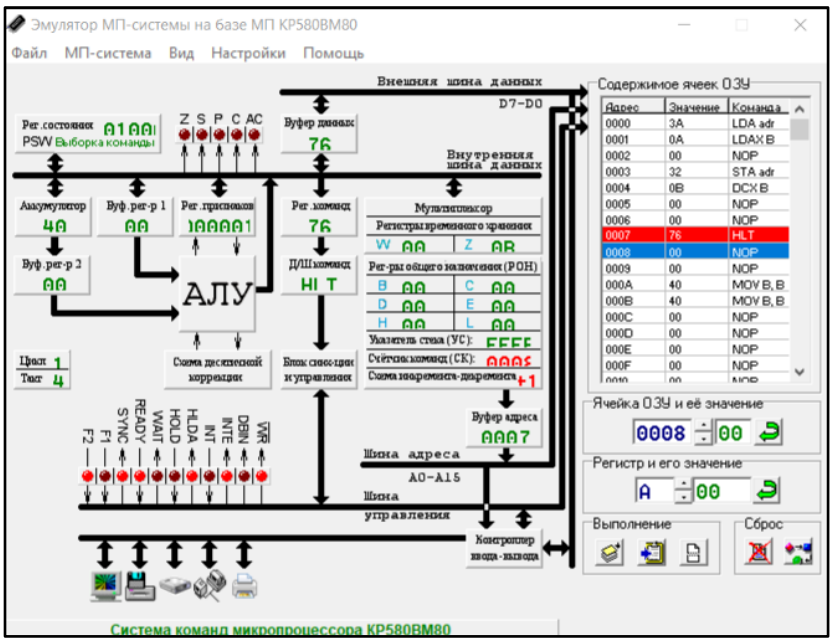

Рисунок 4 – Результат работы модели при прямой адресации

Результат выводится в ячейку памяти *Bh*.

4.2.5 Заключение

Освоено использование команд для прямой адресации.

4.3 Лабораторная работа №3. Исследование команд непосредственной адресации

4.3.1 Цель лабораторной работы

Целью данной работы является ознакомление с командами микропроцессора КР580 для непосредственной адресации

4.3.2 Программный код

№ Ячейки	Значение	Команда	Комментарий
0000	3E	MVI A, d8	Положить в регистр A (аккумулятор)
0001	20	–	число 20h
0002	06	MVI B, d8	Положить в регистр B
0003	05	DCR B	число 5h
0004	26	MVI H, d8	Положить в регистр H

0005	D1	POP D	число D1h
0006	D7	RST 2	Перейти к команде по в ячейке с адресом 10h
0007	00	NOP	
0008	00	NOP	
0009	00	NOP	
000A	00	NOP	
000B	00	NOP	
000C	00	NOP	
000D	00	NOP	
000E	00	NOP	
000F	00	NOP	
0010	00	NOP	
0011	00	NOP	
0012	00	NOP	
0013	00	NOP	
0014	76	HLT	Остановить работу программы
0015	00	NOP	

4.3.3 Ввод операндов при непосредственной адресации

Скрин ввода операндов и начала работы программы представлен на рисунке 5.

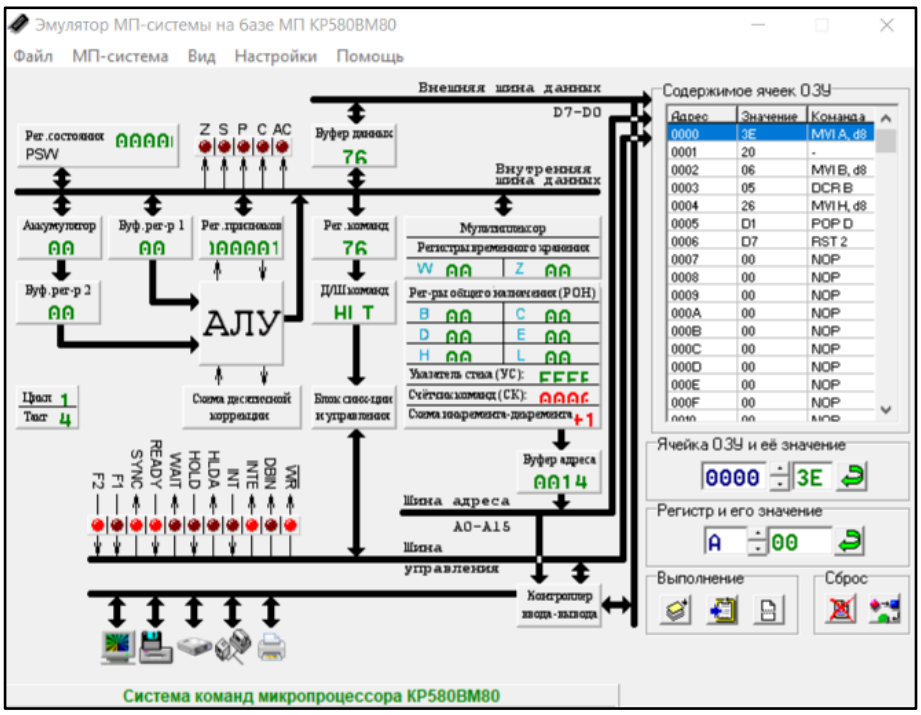

Рисунок 5 – Ввод операндов при непосредственной адресации

В регистр *A* будет вводиться число *20h*, В регистр *B – 5h*, *H – D1h*.

4.3.4 Результат работы модели при непосредственной адресации

Скрин результата работы программы представлен на рисунке 6.

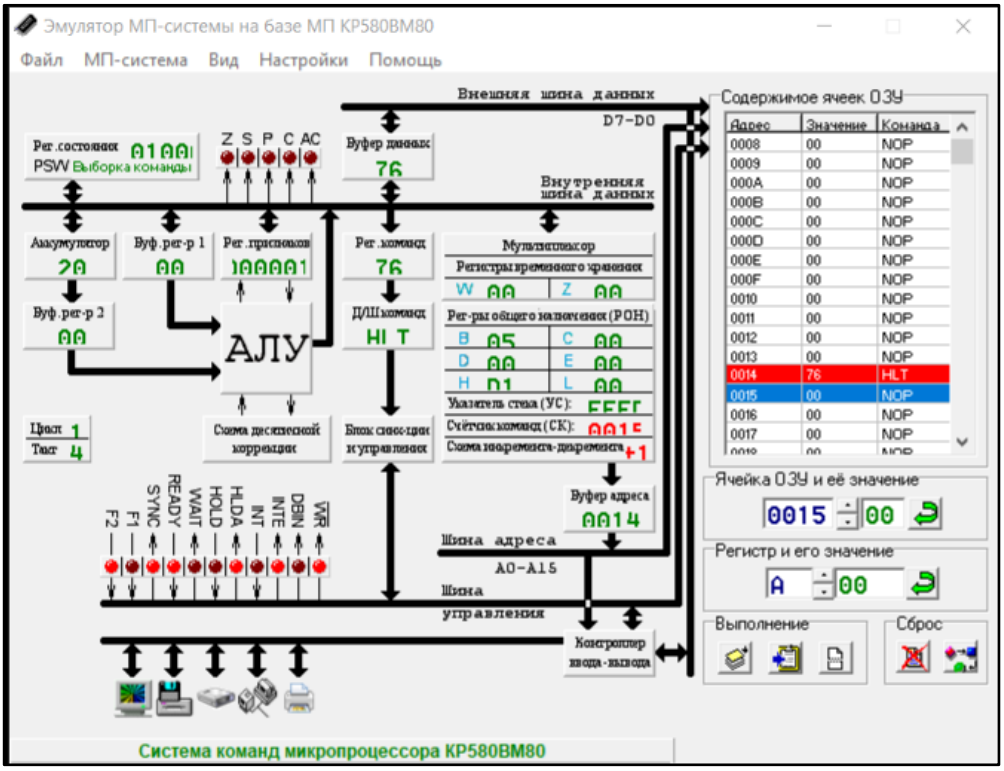

Рисунок 6 – Результат работы модели при непосредственной адресации

Значения регистров приведены в таблице ниже:

	A=	20	
B=	05	C=	00
D=	00	E=	00
H=	D1	L=	00
W=	00	Z=	00

4.3.5 Заключение

Освоено использование команд для непосредственной адресации.

4.4 Лабораторная работа №4. Исследование команд косвенной адресации

4.4.1 Цель лабораторной работы

Целью данной работы является ознакомление с командами микропроцессора КР580 для косвенной адресации

4.4.2 Программный код

№ Ячейки	Значение	Команда	Комментарий
0000	06	MVI B, d8	Переместить адрес ячекйки в регистр B
0001	00	NOP	Адрес ячейки – 00h
0002	0E	MVI C, d8	Переместить адрес ячейки в регистр C
0003	0A	LDAX B	Адрес ячейки – 0Ah
0004	0A	LDAX B	Перенести значение из ячейки с адресом в регистре B, в аккумулятор
0005	00	NOP	
0006	00	NOP	
0007	76	HLT	
0008	00	NOP	
0009	00	NOP	
000A	50	MOV D, B	Значение в ячейке – 50h
000B	00	NOP	
000C	00	NOP	

4.4.3 Ввод операндов при косвенной адресации

Скрин ввода операндов и начала работы программы представлен на рисунке 7.

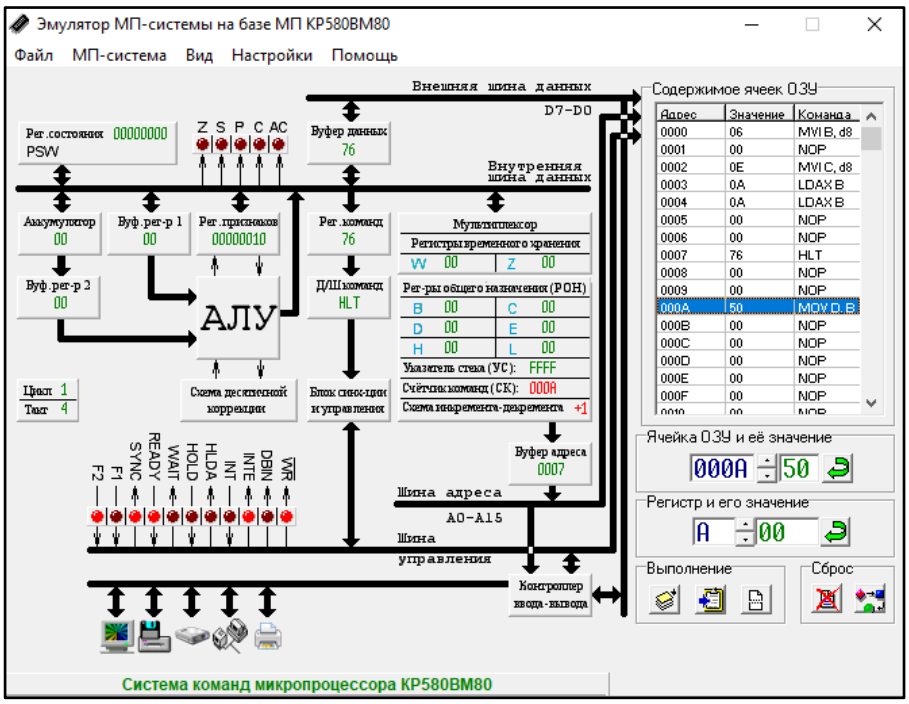

Рисунок 7 – Ввод операндов при косвенной адресации

Вводимое число – 50h.

4.4.4 Результат работы модели при косвенной адресации

Скрин результата работы программы представлен на рисунке 8.

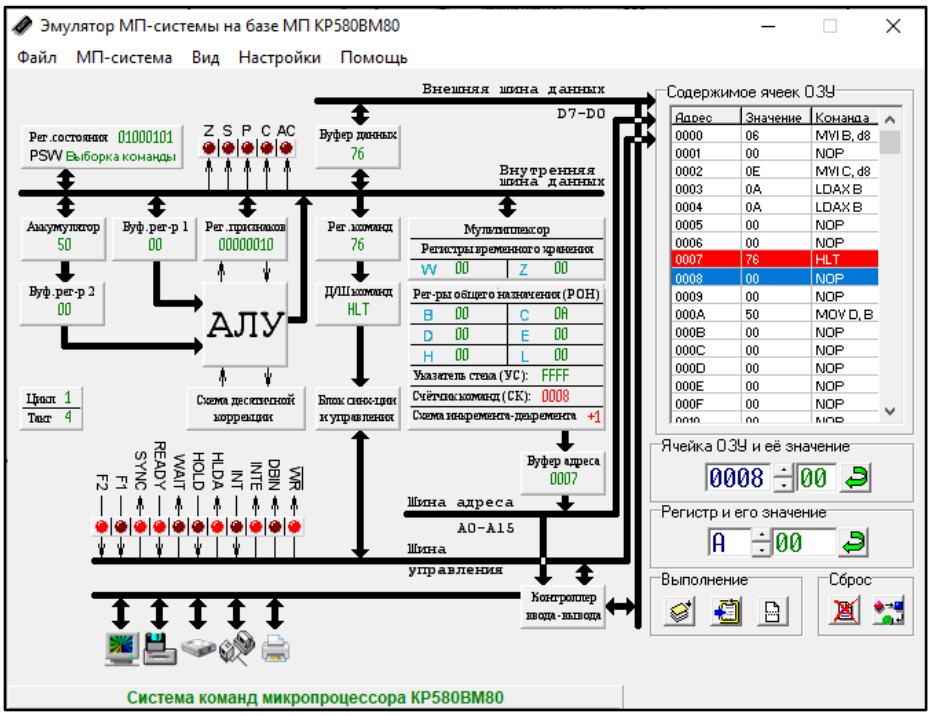

Рисунок 8 – Результат работы модели при косвенной адресации

Значения регистров приведены в таблице ниже:

	A=	50	
B=	00	C=	0A
D=	00	E=	00
H=	00	L=	00
W=	00	Z=	00

4.4.5 Заключение

Освоено использование команд для косвенной адресации.

4.5 Лабораторная работа №5. Исследование команд стековой адресации

4.5.1 Цель лабораторной работы

Целью данной работы является ознакомление с простейшими арифметическими действиями на микропроцессоре КР580

4.5.2 Программный код

№ Ячейки	Значение	Команда	Комментарий
0000	06	MVI B, d8	Переместить адрес ячейки в регистр B
0001	00	NOP	Адрес ячейки – 00h
0002	0E	MVI C, d8	Переместить адрес ячейки в регистр C
0003	0A	LDAX B	Адрес ячейки – 0Ah
0004	0A	LDAX B	Перенести значение из ячейки с адресом в регистре B, в аккумулятор
0005	00	NOP	
0006	00	NOP	
0007	76	HLT	
0008	00	NOP	
0009	00	NOP	
000A	50	MOV D, B	Значение в ячейке – 50h

4.5.3 Ввод операндов при стековой адресации

Скрин ввода операндов и начала работы программы представлен на рисунке 9.

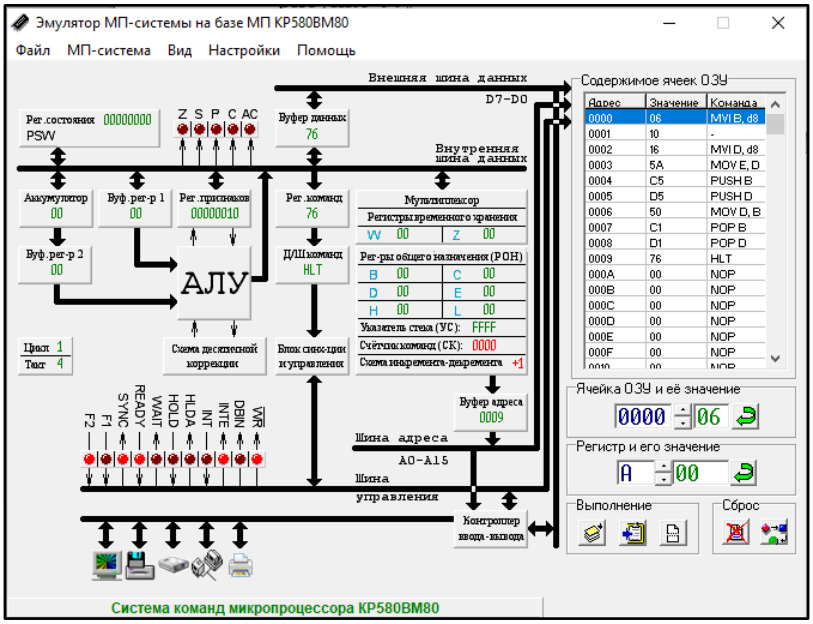

Рисунок 9 – Ввод операндов при стековой адресации

Данные помещаются в регистры B и D, после чего их содержимое посылается в стек. После этого производится обмен значениями между регистрами и значения регистров возвращаются из стека

4.5.4 Результат работы модели при стековой адресации

Скрин результата работы программы представлен на рисунке 10.

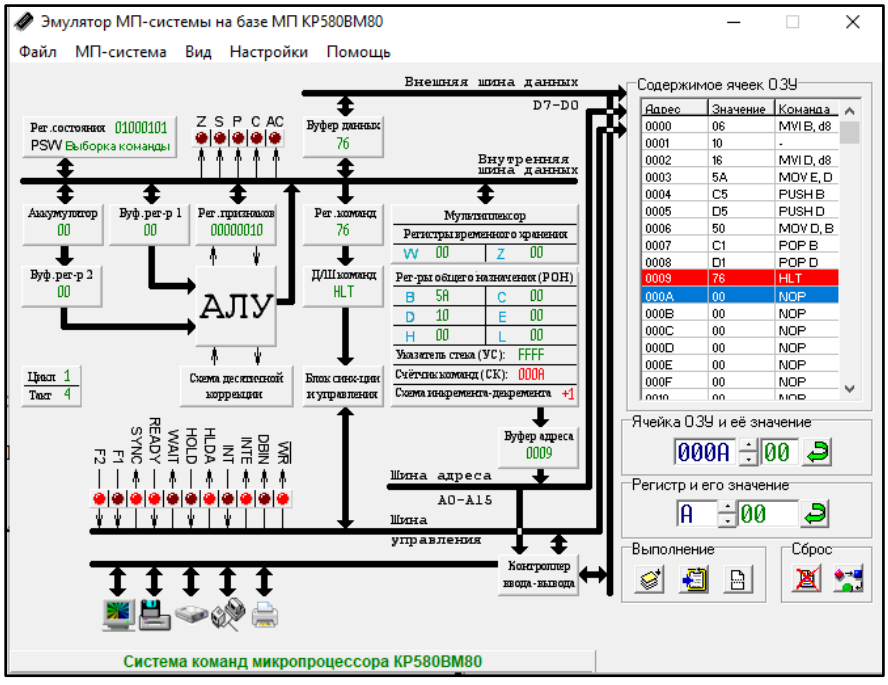

Рисунок 10 – Результат работы модели при стековой адресации

Значения регистров приведены в таблице ниже:

	A=	00	
B=	5A	C=	00
D=	10	E=	00
H=	00	L=	00
W=	00	Z=	00
	SP=	FFFF	

4.5.5 Заключение

Освоено использование команд для стековой адресации.

4.6 Лабораторная работа №6. Пример программы для микропроцессора

4.6.1 Цель лабораторной работы

Целью данной работы является ознакомление с простейшими арифметическими действиями на микропроцессоре КР580

4.6.2 Программный код

Для сложения:

№ Ячейки	Значение	Команда	Комментарий
0000	AF	XRA A	Очистить аккумулятор
0001	3E	MVI A, d8	Записать в аккумулятор
0002	38	-	число $56_{10} = 38h_{16}$
0003	06	MVI B, d8	Записать в регистр B
0004	A3	ANA E	число $163_{10} = a3h_{16}$
0005	80	ADD B	Сложить $38h_{16}$ и $a3h_{16}$ ($56_{10} + 163_{10} = 219_{10} = db_{16}$)
0006	E7	RST 4	Прервать выполнение программы
0007	00	NOP	
0008	00	NOP	
0009	00	NOP	

000A	00	NOP	
000B	00	NOP	
000C	00	NOP	
000D	00	NOP	
000E	00	NOP	
000F	00	NOP	
0010	00	NOP	
0011	00	NOP	
0012	00	NOP	
0013	00	NOP	
0014	00	NOP	
0015	00	NOP	
0016	00	NOP	
0017	00	NOP	
0018	00	NOP	
0019	00	NOP	
001A	00	NOP	
001B	00	NOP	
001C	00	NOP	
001D	00	NOP	
001E	00	NOP	
001F	00	NOP	
0020	00	NOP	
0021	76	HLT	Остановить выполнение программы
0022	00	NOP	

Для вычитания:

№ Ячейки	**Значение**	**Команда**	**Комментарий**
0000	AF	XRA A	Очистить аккумулятор
0001	3E	MVI A, d8	Записать в аккумулятор
0002	E8	-	число $248_{10} = E8h_{16}$
0003	06	MVI B, d8	Записать в регистр B
0004	A3	ANA E	число $163_{10} = a3h_{16}$
0005	90	SUB B	Сложить $E8h_{16}$ и $a3h_{16}$ ($248_{10} – 163_{10} = 65_{10} = 41_{16}$)
0006	E7	RST 4	Прервать выполнение программы
0007	00	NOP	
0008	00	NOP	

0009	00	NOP	
000A	00	NOP	
000B	00	NOP	
000C	00	NOP	
000D	00	NOP	
000E	00	NOP	
000F	00	NOP	
0010	00	NOP	
0011	00	NOP	
0012	00	NOP	
0013	00	NOP	
0014	00	NOP	
0015	00	NOP	
0016	00	NOP	
0017	00	NOP	
0018	00	NOP	
0019	00	NOP	
001A	00	NOP	
001B	00	NOP	
001C	00	NOP	
001D	00	NOP	
001E	00	NOP	
001F	00	NOP	
0020	00	NOP	
0021	76	HLT	Остановить выполнение программы
0022	00	NOP	

4.6.3 Простое сложение двух однобайтных чисел

Скрин программы показан на рисунке 11.

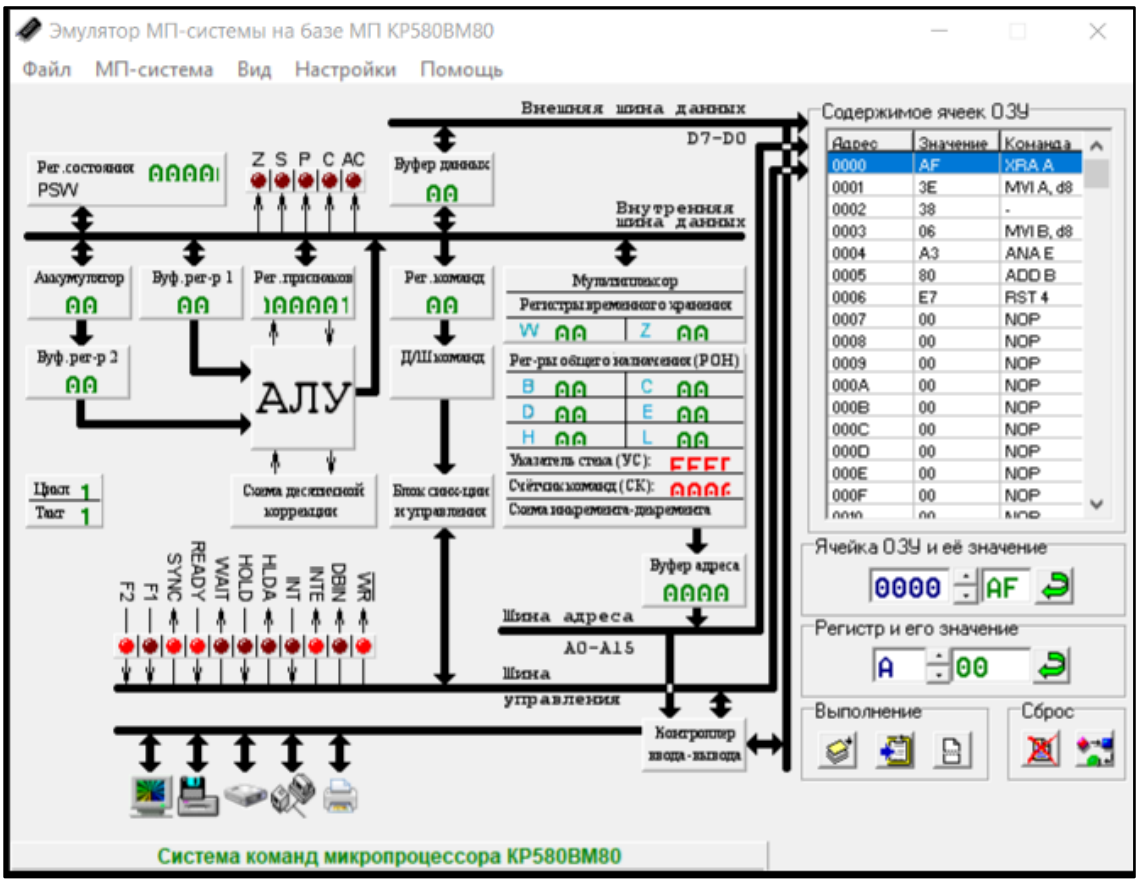

Рисунок 11 – Простое сложение двух однобайтных чисел

Значения регистров, после выполнения программы представлены в таблице ниже:

	A=	DB	
B=	A3	C=	00
D=	00	E=	00
H=	00	L=	00
W=	00	Z=	00

4.6.4 Простое вычитание двух однобайтных чисел

Скрин программы показан на рисунке 12.

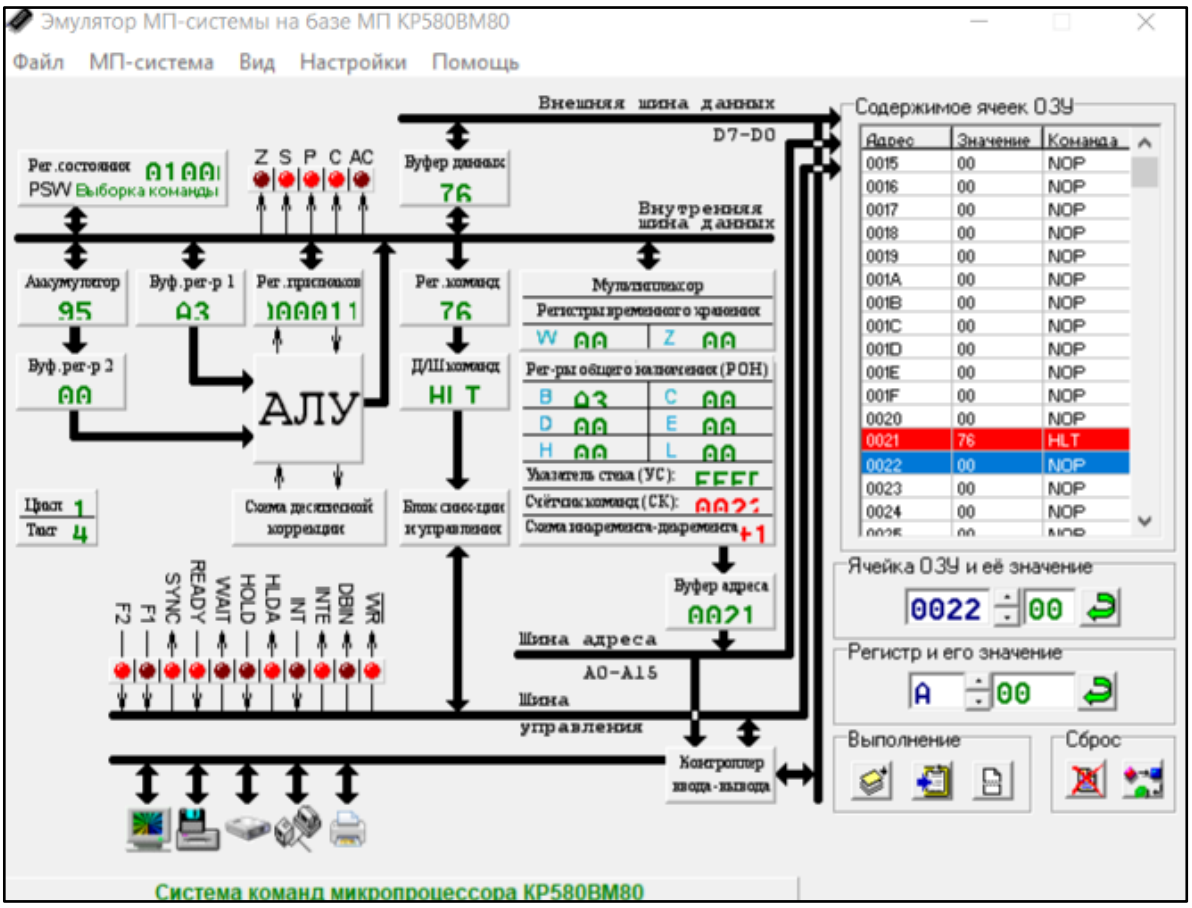

Рисунок 12 – Простое вычитание двух однобайтных чисел

	A=	95	
B=	A3	C=	00
D=	00	E=	00
H=	00	L=	00
W=	00	Z=	00

4.6.5 Заключение

Освоено использование команд для простейших арифметических действий.

4.7 Лабораторная работа №7. Программа сложения двух однобайтных чисел X и У

4.7.1 Цель лабораторной работы

Рассмотреть особенности выполнения простейших арифметических операций над целыми числами без знака на эмуляторе МП К580, познакомиться с программированием в машинных кодах и мнемокодах,

научиться пользоваться средствами управления и кнопками эмулятора.

4.7.2 Вариант задания

$X = 39_{10} = 27_{16}$ и $Y = 198_{10} = C6_{16}$. $X+Y = 27_{16} + C6_{16} = 39_{10} + 198_{10} = 237_{10} = ED_{16}$.

4.7.3 Программный код

№ Ячейки	Значение	Команда	Комментарий
0000	AF	XRA A	Очистить аккумулятор
0001	3E	MVI A, d8	Записать в аккумулятор
0002	27	DAA	число X ($39_{10} = 27_{16}$)
0003	06	MVI B, d8	Записать в регистр B
0004	C6	ADI d8	число У ($198_{10} = C6_{16}$)
0005	80	ADD B	Сложить X и Y Сложить 27_{16} и $C6_{16}$ ($39_{10} + 198_{10} = 237_{10} = ED_{16}$)
0006	32	STA adr	Записать содержимое аккумулятора в
0007	0D	DCR C	ячейку 000D
0008	00	NOP	
0009	76	HLT	Остановить программу

4.7.4 Выполнение программы

Скрин программы до её выполнения представлен на рисунке 13.

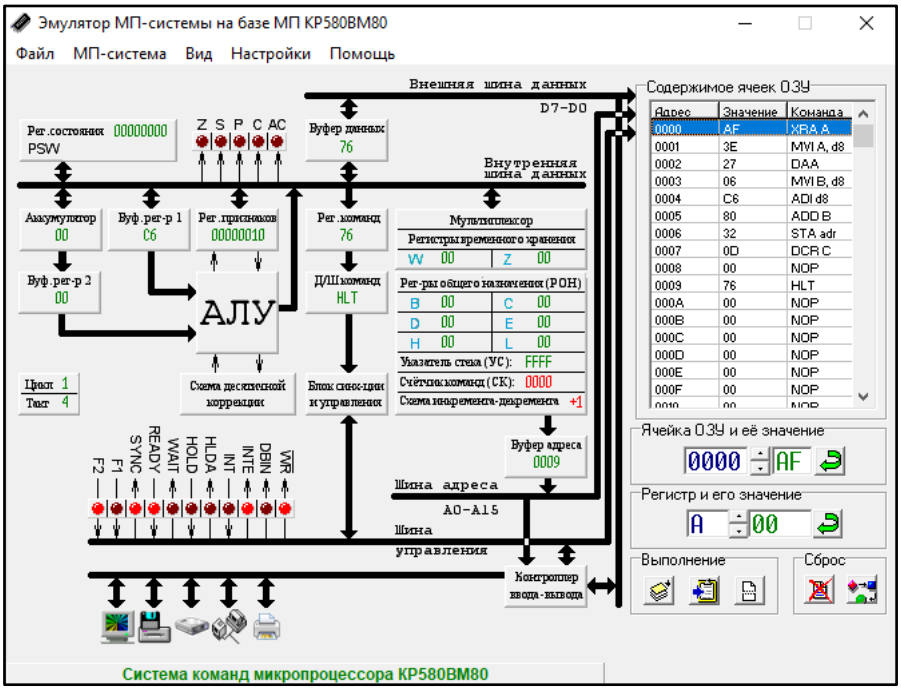

Рисунок 13 – Скрин программы

Скрин программы после её выполнения представлен на рисунке 14.

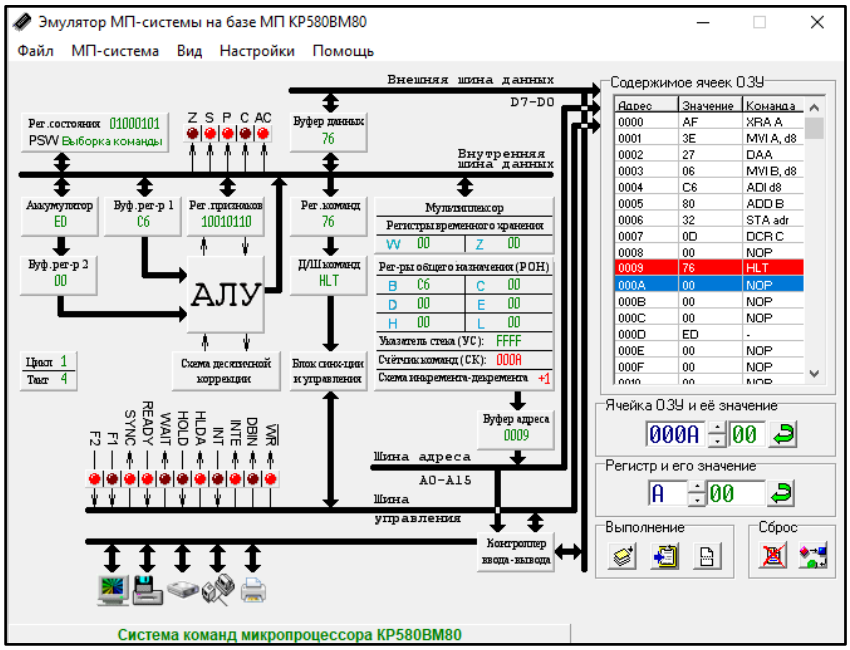

Рисунок 14 – Скрин программы

Результат вычислений был выведен в ячейку *000D*. Этот результат равен ED_{16} (237_{10}).

4.7.5 Заключение

Освоено выполнение сложения двух чисел.

4.8 Лабораторная работа №8. Программа вычитания двух однобайтных чисел X и У

4.8.1 Цель лабораторной работы

Рассмотреть особенности выполнения простейших арифметических операций над целыми числами без знака на эмуляторе МП К580, познакомиться с программированием в машинных кодах и мнемокодах, научиться пользоваться средствами управления и кнопками эмулятора.

4.8.2 Вариант задания

X = 200_{10} = $C8_{16}$ и Y = 45_{10} = $2D_{16}$. X–Y = $C8_{16}$ – $2D_{16}$ = 200_{10} – 45_{10} = 155_{10} = $9B_{16}$.

4.8.3 Программный код

№ Ячейки	Значение	Команда	Комментарий
0000	AF	XRA A	Очистить аккумулятор
0001	3E	MVI A, d8	Записать в аккумулятор
0002	C8	RZ	число X (200_{10} = $C8_{16}$)
0003	06	MVI B, d8	Записать в регистр B
0004	2D	DCR L	число У (45_{10} = $2D_{16}$)
0005	90	SUB B	Вычесть X и Y Сложить $C8_{16}$ и $2D_{16}$ (200_{10} – 45_{10} = 144_{10} = $9B_{16}$)
0006	32	STA adr	
0007	0D	DCR C	
0008	00	NOP	
0009	76	HLT	
000A	00	NOP	

4.8.4 Выполнение программы

Скрин программы до её выполнения представлен на рисунке 15.

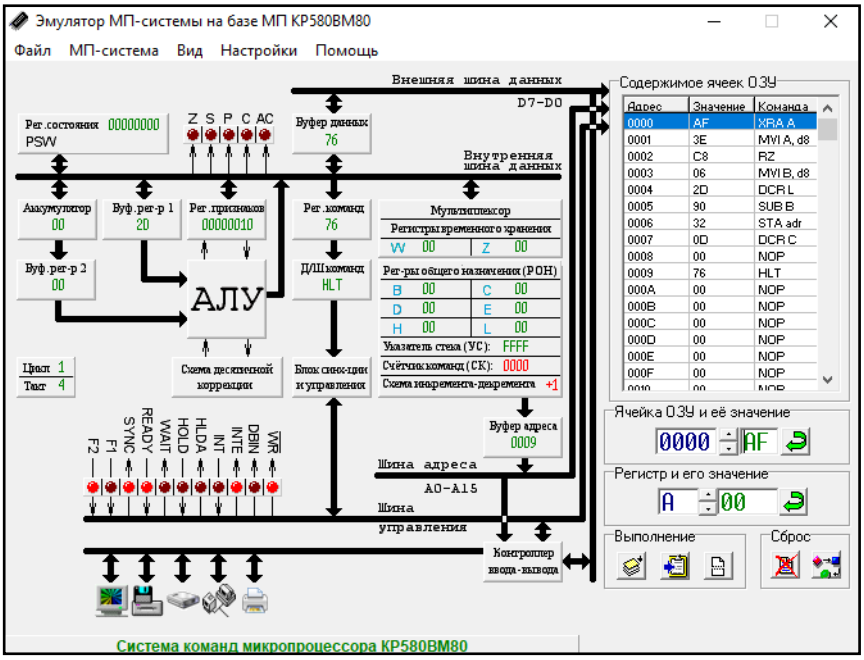

Рисунок 15 – Скрин программы

Скрин программы после её выполнения представлен на рисунке 16.

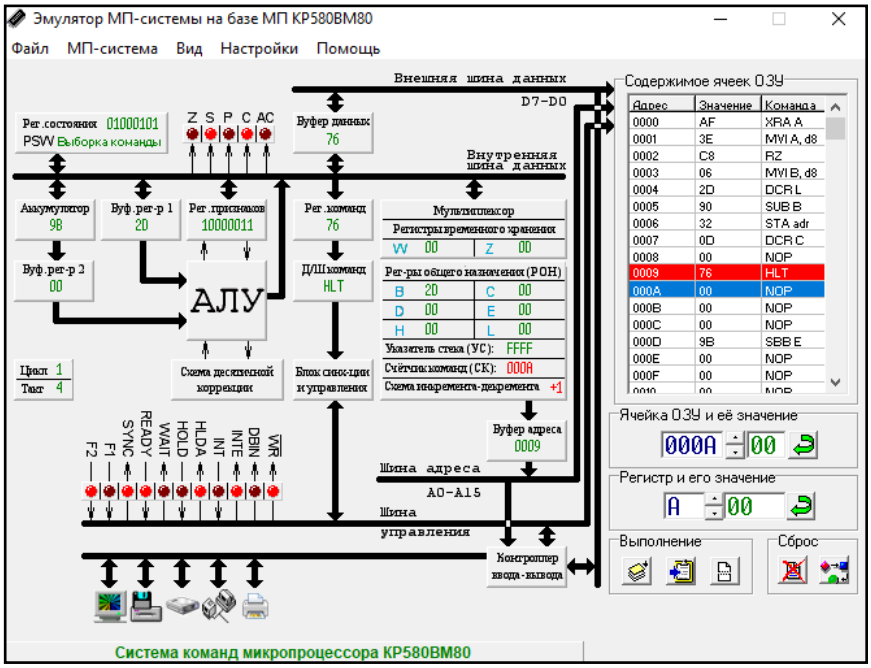

Рисунок 16 – Скрин программы

Результат вычислений был выведен в ячейку *000D*. Этот результат равен $9B_{16}$ *(*155_{10}*)*.

4.7.5 Заключение

Освоено выполнение вычитание двух чисел.

4.9 Лабораторная работа №9. Сложение массива однобайтных чисел

4.9.1 Цель лабораторной работы

Рассмотреть особенности выполнения простейших арифметических операций над целыми числами без знака на эмуляторе МП К580, познакомиться с программированием в машинных кодах и мнемокодах, научиться пользоваться средствами управления и кнопками эмулятора.

4.9.2 Вариант задания

$123_{10} + 56_{10} + 178_{10} + 89_{10} + 90_{10} + 99_{10} = 7B_{16} + 38_{16} + B2_{16} + 59_{16} + 5A_{16} + 63_{16} = 635_{10} = 27B_{16}$.

4.9.3 Программный код

№ Ячейки	Значение	Команда	Комментарий
0000	21	LXI H, d16	Загрузить в регистры HL, адрес первого слагаемого
0001	20	-	Первое слагаемое дано в ячейке 0020
0002	00	NOP	
0003	0E	MVI C, d8	Загрузить в регистр C количество слагаемых
0004	06	MVI B, d8	Количество слагаемых = 6
0005	AF	XRA A	Очистить аккумулятор
0006	47	MOV B, A	Очистить регистр B
0007	86	ADD M	Прибавить к содержимому аккумулятора число из массива слагаемых
0008	D2	JNC adr	Если переноса нет, то идти на M2
0009	0D	DCR C	M2=000D
000A	00	NOP	
000B	04	INR B	Увеличить содержимое регистра B на 1
000C	B7	ORA A	Очистить флаг переноса
000D	23	INX H	Указать на следующий адрес слагаемого
000E	0D	DCR C	Уменьшить содержимое регистра C на 1
000F	C2	JNZ adr	Если не все слагаемые, то идти на M1
0010	07	RLC	M1=0007
0011	00	NOP	
0012	32	STA adr	Записать значение аккумулятора
0013	2A	LHLD adr	в ячейку 002A
0014	00	NOP	
0015	78	MOV A, B	Перенести значение из регистра B в регистр A
0016	32	STA adr	Записать значение аккумулятора
0017	29	DAD H	в ячейку 0029
0018	00	NOP	
0019	76	HLT	Остановить программу
001A	00	NOP	
001B	00	NOP	
001C	00	NOP	
001D	00	NOP	
001E	00	NOP	
001F	00	NOP	
0020	7B	MOV A, E	Массив чисел

0021	38	-	
0022	B2	ORA D	
0023	59	MOV E, C	
0024	5A	MOV E, D	
0025	63	MOV H, E	
0026	00	NOP	
0027	00	NOP	
0028	00	NOP	

4.9.4 Выполнение программы

Скрин программы до её выполнения представлен на рисунке 17.

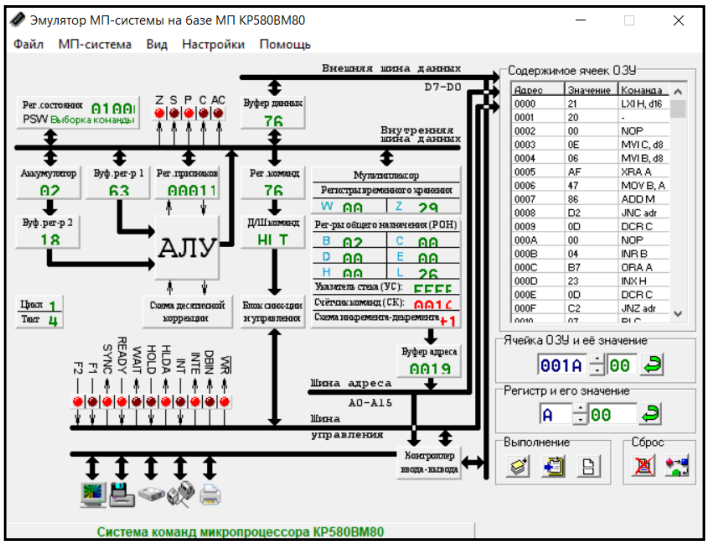

Рисунок 17 – Скрин программы

Скрин программы после её выполнения представлен на рисунке 18.

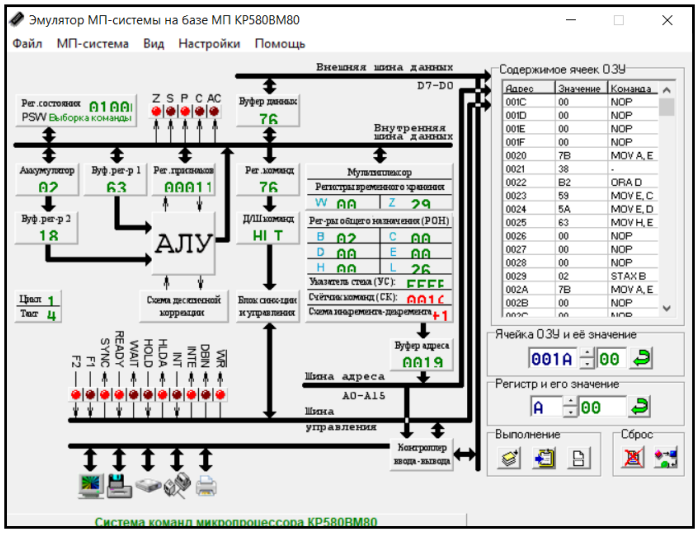

Рисунок 18 – Скрин программы

Результат вычислений был выведен в ячейки *0029* и *002A*. Этот результат равен $27B_{16}$ *(*635_{10}*) – 02* выведено в ячейку *0029*, а *7B* – в *002A*.

4.7.5 Заключение

Освоена работа с массивами.

4.10 Лабораторная работа №10. Сложение двухбайтовых десятичных чисел

4.10.1 Цель лабораторной работы

Выполнить операцию сложения двух двухбайтных чисел

4.10.2 Вариант задания

$6701_{10} + 4670_{10} = 1A2D_{16} + 123E_{16} = 11371_{10} = 2C6B_{16}$.

4.10.3 Программный код

№ Ячейки	Значение	Команда	Комментарий
0000	3E	MVI A, d8	Младший байт первого слагаемого заносится в аккумулятор $2D_{16}$ –> A
0001	2D	DCR L	
0002	06	MVI B, d8	Младший байт второго слагаемого заносится в регистр B: $3E_{16}$ –> B
0003	3E	MVI A, d8	
0004	80	ADD B	Сложить ($2D_{16} + 3E_{16} = 6B_{16}$)
0005	32	STA adr	Записать значение аккумулятора
0006	14	INR D	В ячейку *0014*
0007	00	NOP	
0008	3E	MVI A, d8	Старший байт первого слагаемого заносится в аккумулятор $1A_{16}$ –> A
0009	1A	LDAX D	
000A	06	MVI B, d8	Старший байт второго слагаемого заносится в регистр B: 12_{16} –> B

000B	12	STAX D	
000C	88	ADC B	Сложить ($1A_{16} + 12_{16} = 2C_{16}$)
000D	32	STA adr	Записать значение аккумулятора
000E	15	DCR D	В ячейку *0014*
000F	00	NOP	
0010	76	HLT	
0011	00	NOP	

4.10.4 Выполнение программы

Скрин программы до её выполнения представлен на рисунке 19.

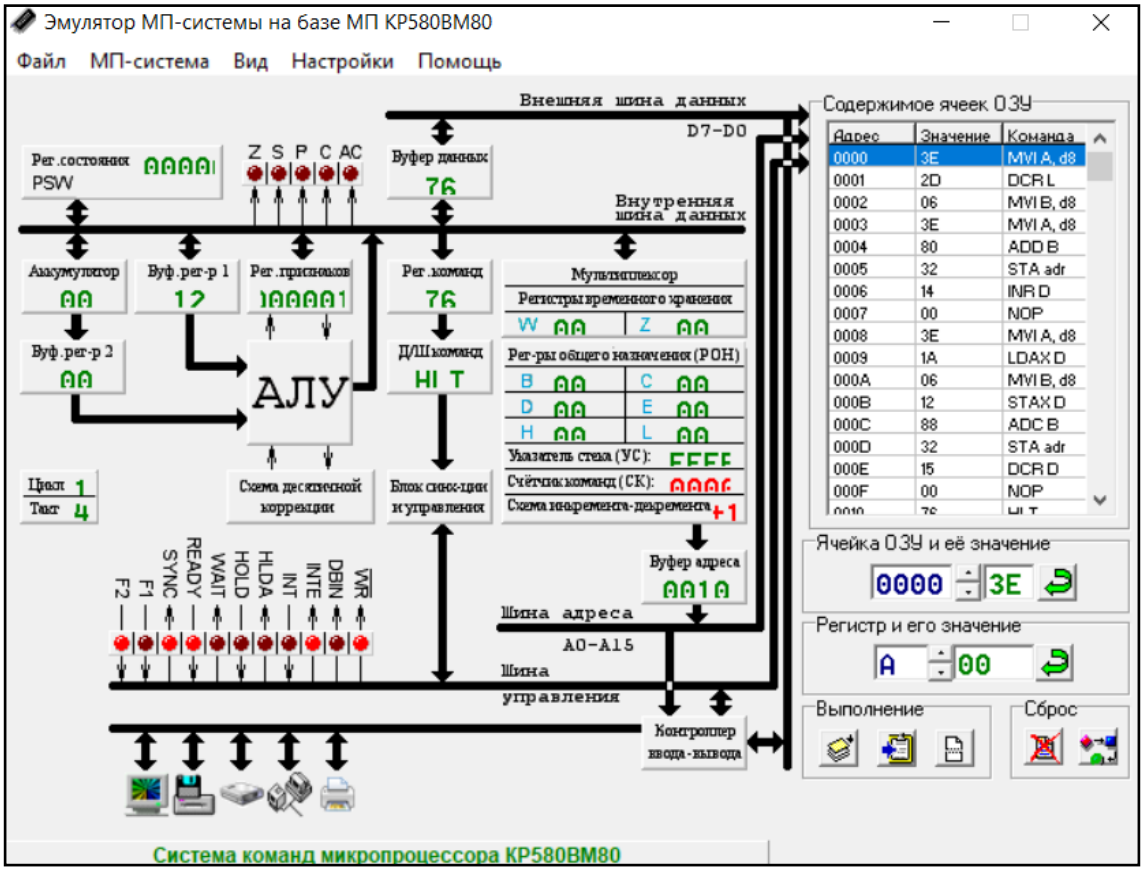

Рисунок 19 – Скрин программы

Скрин программы после её выполнения представлен на рисунке 20.

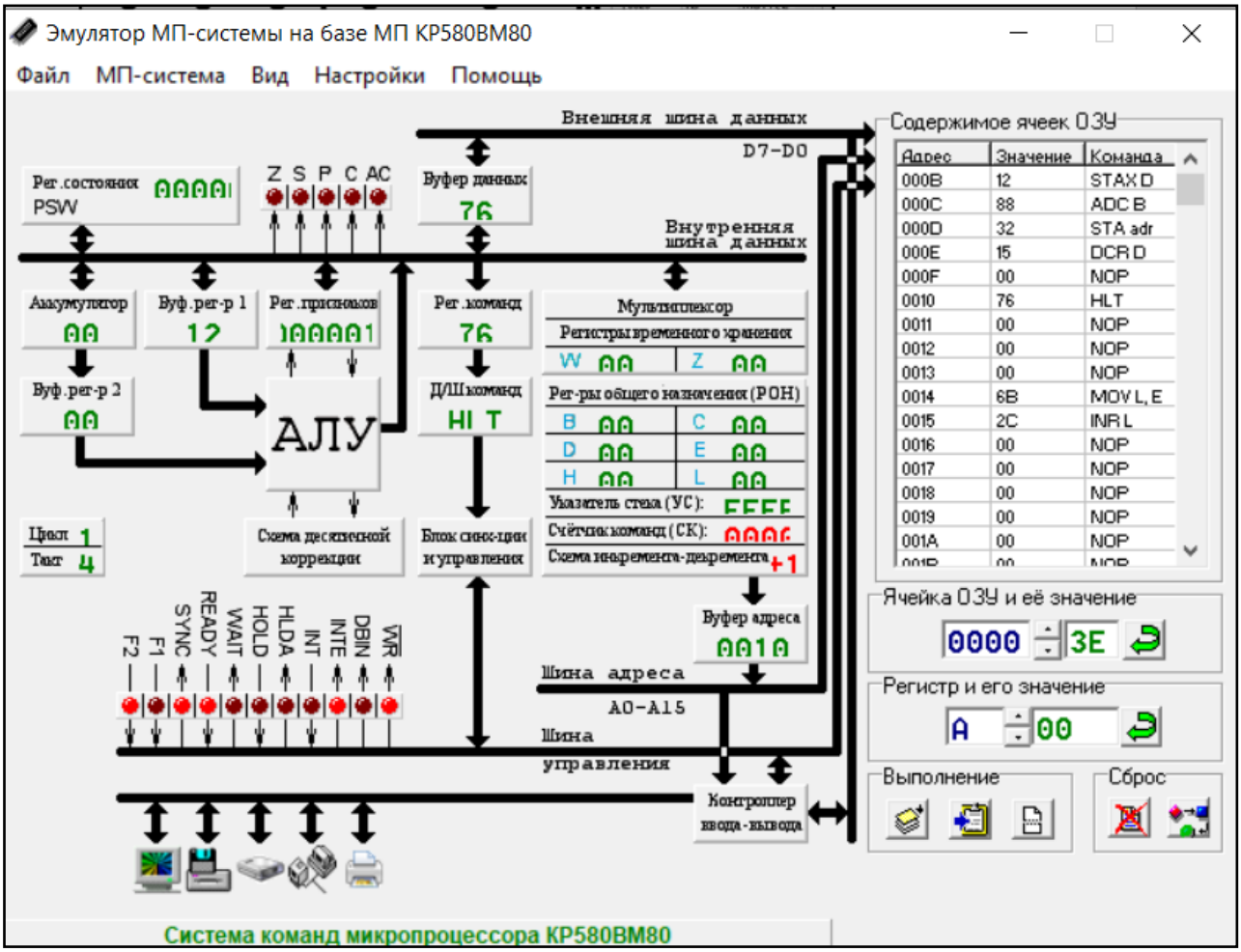

Рисунок 20 – Скрин программы

Результат вычислений был выведен в ячейки *0014* и *0015*. Этот результат равен $2C6B_{16}$ *(*11371_{10}*) – 6B (Младший байт)* выведено в ячейку *0014*, а *2C (Старший байт)* – в *0015*.

4.7.5 Заключение

Освоено сложение двухбайтовых чисел.

4.11 Лабораторная работа №11. Вычитание одинаковых по длине чисел

4.11.1 Цель лабораторной работы

Выполнить операцию вычитание двух двухбайтных чисел

4.11.2 Вариант задания

$63670_{10} – 17954_{10} = F8B6_{16} – 4622_{16} = 45716_{10} = B294_{16}$.

4.11.3 Программный код

№ Ячейки	Значение	Команда	Комментарий
0000	3E	MVI A, d8	Младший байт первого слагаемого заносится в аккумулятор $B6_{16}$ –> A
0001	B6	ORA M	
0002	D6	SUI d8	Вычесть ($B6_{16} + 22_{16} = 94_{16}$)
0003	22	SHLD adr	
0004	32	STA adr	Записать значение аккумулятора в ячейку *0010*
0005	10	-	
0006	00	NOP	
0007	3E	MVI A, d8	Старший байт первого слагаемого заносится в аккумулятор $F8_{16}$ –> A
0008	F8	RM	
0009	DE	SBI d8	Вычесть ($F8_{16} + 46_{16} = B2_{16}$)
000A	46	MOV B, M	
000B	32	STA adr	Записать значение аккумулятора в ячейку *0011*
000C	11	LXI D, d16	
000D	00	NOP	
000E	76	HLT	Остановить программу
000F	00	NOP	

4.11.4 Выполнение программы

Скрин программы до её выполнения представлен на рисунке 21.

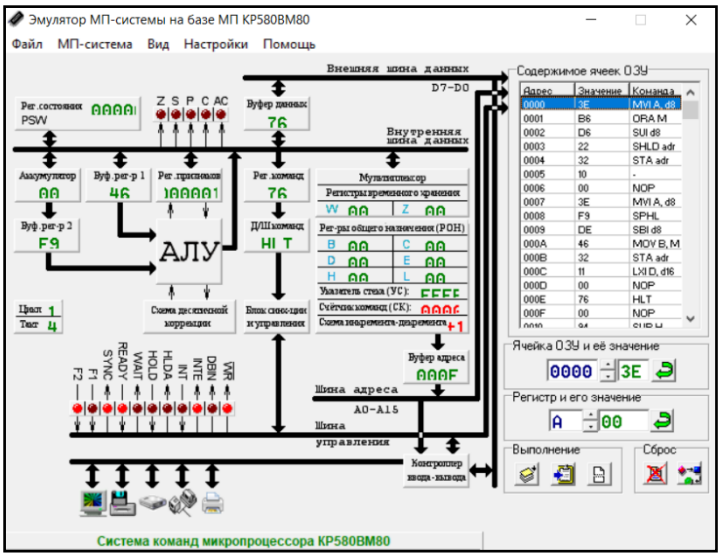

Рисунок 21 – Скрин программы

Скрин программы после её выполнения представлен на рисунке 22.

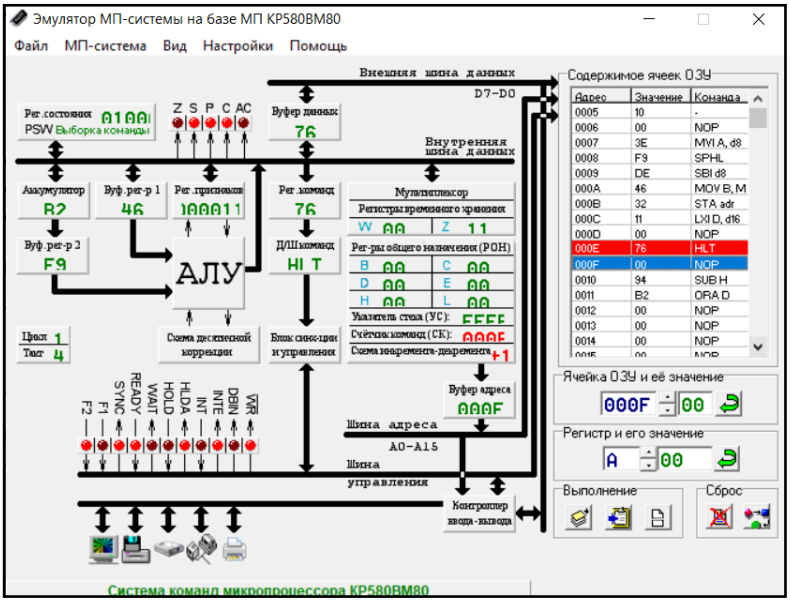

Рисунок 22 – Скрин программы

Результат вычислений был выведен в ячейки *0014* и *0015*. Этот результат равен $B294_{16}$ *(*155_{10}*) – 94 (Младший байт)* выведено в ячейку *0010*, а *B2 (Старший байт)* – в *0010*.

4.7.5 Заключение

Освоено вычитание двухбайтовых чисел.

4.12 Лабораторная работа №12