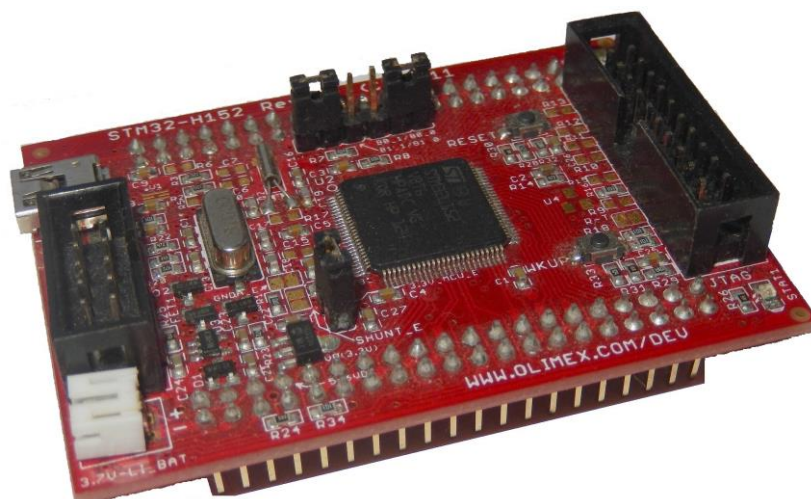


**В.И. Бойков, С.В. Быстров, С.М. Власов,  
Н.А. Николаев**

**МИКРОКОНТРОЛЛЕРНАЯ ТЕХНИКА  
СИСТЕМ УПРАВЛЕНИЯ.  
МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ВЫПОЛНЕНИЮ  
ЛАБОРАТОРНЫХ РАБОТ**



**Санкт-Петербург  
2019**

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ  
ФЕДЕРАЦИИ

УНИВЕРСИТЕТ ИТМО

**В.И. Бойков, С.В. Быстров, С.М. Власов,  
Н.А. Николаев**

**МИКРОКОНТРОЛЛЕРНАЯ ТЕХНИКА  
СИСТЕМ УПРАВЛЕНИЯ.  
МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ВЫПОЛНЕНИЮ  
ЛАБОРАТОРНЫХ РАБОТ**

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

РЕКОМЕНДОВАНО К ИСПОЛЬЗОВАНИЮ В УНИВЕРСИТЕТЕ ИТМО  
по направлению подготовки 15.03.06, 27.03.04  
в качестве практикума для реализации основных профессиональных  
образовательных программ высшего образования бакалавриата

 УНИВЕРСИТЕТ ИТМО

**Санкт-Петербург  
2019**

Бойков В.И., Быстров С.В., Власов С.М., Николаев Н.А.,  
Микроконтроллерная техника систем управления. Методические указания к  
выполнению лабораторных работ– СПб: Университет ИТМО, 2019. – 70 с.

Рецензент:

Герасимов Дмитрий Николаевич, кандидат технических наук, доцент  
(квалификационная категория "ординарный доцент") факультета систем  
управления и робототехники Университета ИТМО.

В пособии приведены методические указания к выполнению  
лабораторных работ по курсу "Микроконтроллерная техника систем  
управления". Приведены основные сведения по программному обеспечению,  
необходимому для выполнения лабораторных работ, варианты заданий и  
методические указания к реализации.

Пособие предназначено для студентов технических университетов,  
обучающихся по направлениям подготовки бакалавров 15.03.06  
–«Мехатроника и робототехника», 27.03.04 –«Управление в технических  
системах». Пособие может быть полезно студентам других технических  
направлений, направленность обучения которых связана с решением вопросов  
реализации цифровых устройств управления и обработки информации на базе  
микроконтроллеров.



**Университет ИТМО** – ведущий вуз России в области информационных и  
фотонных технологий, один из немногих российских вузов, получивших в 2009  
году статус национального исследовательского университета. С 2013 года  
Университет ИТМО – участник программы повышения конкурентоспособности  
российских университетов среди ведущих мировых научно-образовательных  
центров, известной как проект «5 в 100». Цель Университета ИТМО –  
становление исследовательского университета мирового уровня,  
предпринимательского по типу, ориентированного на интернационализацию  
всех направлений деятельности.

© Университет ИТМО, 2019

© Бойков В.И., Быстров С.В., Власов С.М., Николаев Н.А., 2019

## Содержание

Введение .....	4
Моделирование работы микроконтроллера в программной среде MCStudio 1.7 .....	5
Лабораторная работа №1. Выполнение арифметических операций .....	19
Лабораторная работа №2. Управление ходом вычислительного процесса .....	25
Лабораторная работа №3. Использование подпрограмм .....	32
Лабораторная работа №4. Работа с портами ввода-вывода .....	36
Лабораторная работа №5. Отображение информации на индикаторе .....	42
Лабораторная работа №6. Обработка прерываний от внешнего сигнала .....	45
Лабораторная работа №7. Работа с таймером .....	50
Лабораторная работа №8. Измерение интервалов времени .....	56
Список использованных источников .....	61
Приложение 1. Система команд микроконтроллеров семейства MCS51 .....	62

## ВВЕДЕНИЕ

Лабораторный практикум по дисциплине «Микроконтроллерная техника систем управления» направлен на выработку у студентов начальных навыков программирования микроконтроллеров. Наличие таких навыков способствует формированию профессиональных компетенций, связанных со способностью выполнять проектирование блоков и устройств автоматики, разрабатывать программное обеспечение, необходимое для решения задач управления и обработки информации.

Лабораторный практикум содержит 8 работ, для каждой из которых разработаны 20 вариантов задания. Работы построены по принципу последовательного наращивания сложности, причем в последующих работах используются участки программного кода, отлаженного в предыдущих работах. В связи с этим рекомендуется выдавать студентам индивидуальные задания с единым вариантом для всего практикума.

Лабораторный практикум выполняется в программной среде MCStudio. Программная среда MCStudio работает под управлением ОС Windows и позволяет моделировать взаимодействие микроконтроллера с периферийными устройствами. На компьютерах под управлением ОС Linux программная среда MCStudio может работать некорректно. На таких компьютерах для моделирования можно использовать программную среду IDE MCU8051. Однако она имеет меньший набор периферийных устройств, что не позволит выполнить некоторые лабораторные работы.

Лабораторные работы №1 – №3 направлены на знакомство с программной средой моделирования MCStudio, архитектурой микроконтроллера и системой команд.

В лабораторных работах №4 – №5 обучающиеся знакомятся с простейшими приемами работы с периферийными устройствами. При их выполнении следует уделить внимание организации панели управления прибора, отображению информации и правильности программной обработки команд оператора.

Лабораторные работы №6 – №8 направлены на выработку навыков организации работы программы в режиме реального времени. В разрабатываемых программах обязательно использование процедур обработки прерываний как от внешних сигналов, так и от таймеров.

Общая трудоемкость выполнения лабораторного практикума составляет 16 аудиторных часов. Трудоемкость выполнения каждой лабораторной работы равна двум часам.

## МОДЕЛИРОВАНИЕ РАБОТЫ МИКРОКОНТРОЛЛЕРА В ПРОГРАММНОЙ СРЕДЕ MCStudio 1.7

Изложенные ниже сведения о программной среде MCStudio представляют собой сокращенный вариант Руководства пользователя [1].

### 1 Запуск программной среды MCStudio.

Интегрированная программная среда (далее система) MCStudio является инструментальным обеспечением процесса проектирования программ для широкого спектра однокристальных микроконтроллеров семейства MCS-51. Программная среда поддерживает моделирование работы не только самого микроконтроллера, но и некоторого набора аппаратных средств, относящихся к окружению микроконтроллера в физической системе. Это позволяет приблизить процесс тестирования прикладной программы к реальным условиям и повышает наглядность ее работы.

Система MCStudio может быть запущена через ярлык на рабочем столе компьютера либо через пункт главного меню Windows *Все программы* → *MCStudio* → *MCStudio*.

Программная среда MCStudio использует концепцию проектов. Проект – это совокупность файлов, которые являются элементами описания прикладной программы для микроконтроллера. Таким образом, каждый файл проекта может быть сформирован пользователем или создан системой MCStudio.

Система MCStudio поддерживает организацию прикладной программы для микроконтроллера на основе модульного подхода, то есть прикладная программа может состоять из произвольного количества программных модулей (файлов), которые содержат взаимные ссылки. Итоговые файлы проекта создаются системой из исходных файлов в процессе компиляции (build). При компиляции проекта на основе исходных файлов формируются вспомогательные файлы, файлы, которые используются при симуляции, а также исполнимые файлы, которые могут быть загружены в реальный микроконтроллер.

Важным понятием в системе MCStudio является понятие модели микроконтроллера. Модель микроконтроллера определяет совокупность аппаратных ресурсов, которые учитываются компилятором и симулятором в составе системы. Каждая модель микроконтроллера соответствует аппаратной структуре определенной модификации физического микроконтроллера семейства MCS-51 (например, i8051, AT89S51, AT89S8252). С проектом связывается определенная модель микроконтроллера, для которого создается программа в этом проекте. Модель микроконтроллера – это совокупность следующих параметров:

- объемы внутренней и внешней памяти программ и данных;
- номенклатура и количество компонентов периферии, используемых в данной модификации микроконтроллера;

- конфигурационные параметры, используемые при симуляции, например, размер и скорость доступа для модели памяти EEPROM.

Все операции с файлами проекта и описания моделей микроконтроллеров, перечисленными выше, выполняются в системе "MCStudio" менеджерами проектов и моделей, в которых также поддерживается концепция расширения списка моделей микроконтроллеров.

В начале работы отображается окно менеджера проектов. Если во время предыдущего сеанса работы был создан проект и выполнялось редактирование файлов прикладной программы, то сразу будет открыт редактор с этими файлами.

Вид окна менеджера проектов показан на рисунке 1. Непосредственно в этом окне пользователь может выполнить следующие действия:

**Создать...** – создать новый проект, файл программы различными языками (выбирается в появляющемся окне);

**Открыть...** – выбрать существующий проект или отдельный файл и открыть его в соответствующем редакторе (выбор в окне диалога);

**Редактор окружения...** – активизируется подсистема визуального редактирования моделей аппаратуры внешнего окружения микроконтроллера.

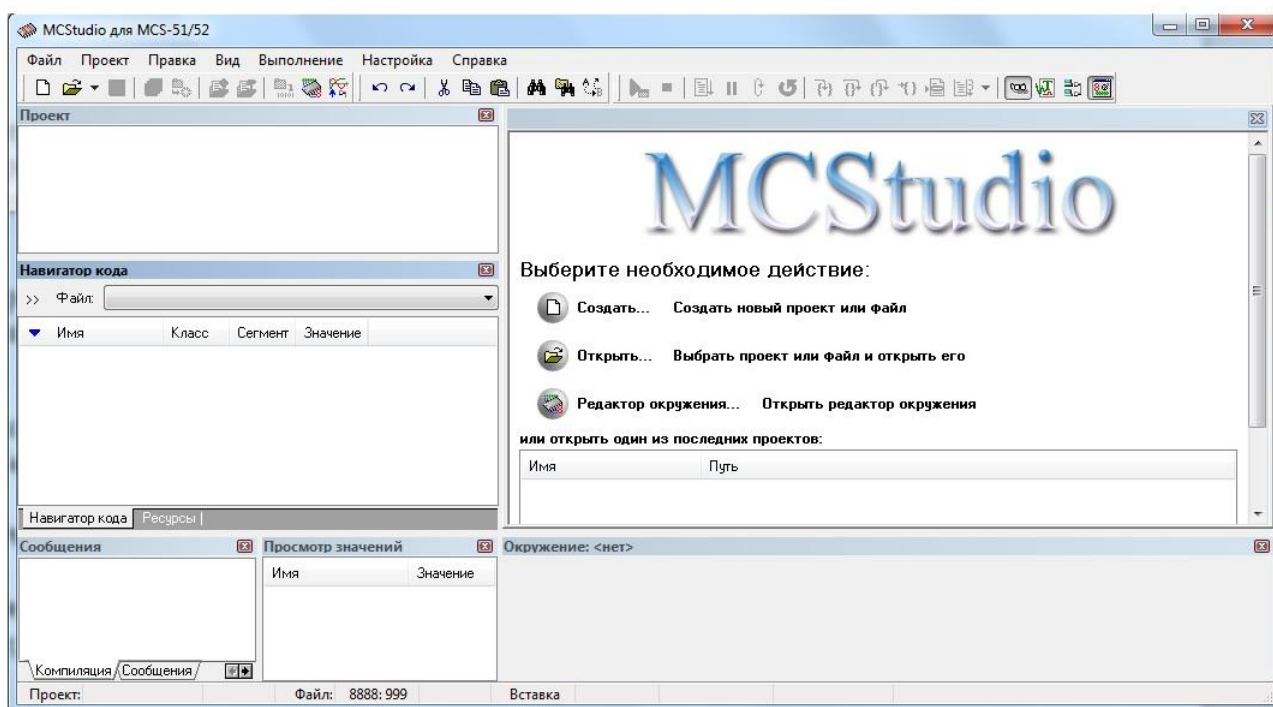


Рисунок 1– Вид окна менеджера проектов

В начале работы следует выбрать действие – «Создать новый проект». Откроется вспомогательное окно (см. рисунок 2), в котором следует выбрать вкладку *Проект*→*ОК*. Откроется вспомогательное окно «Создание проекта»,

показанное на рисунке 3. В открывшемся окне следует задать оригинальное имя проекта и указать путь к проекту. Кроме того, следует выбрать модель микроконтроллера. Приведенные ниже описания лабораторных работ ориентированы на модель Intel 80C51. После задания всех параметров откроется панель интерфейса главного окна.

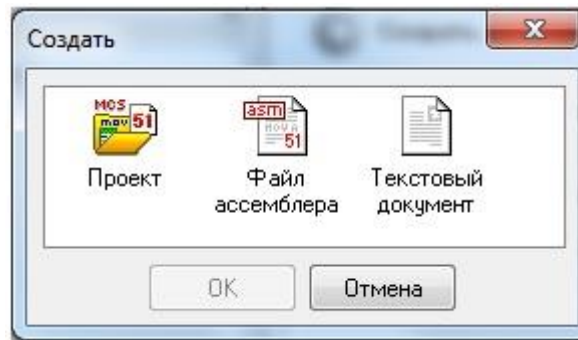


Рисунок 2 – Окно выбора действия в менеджере проектов

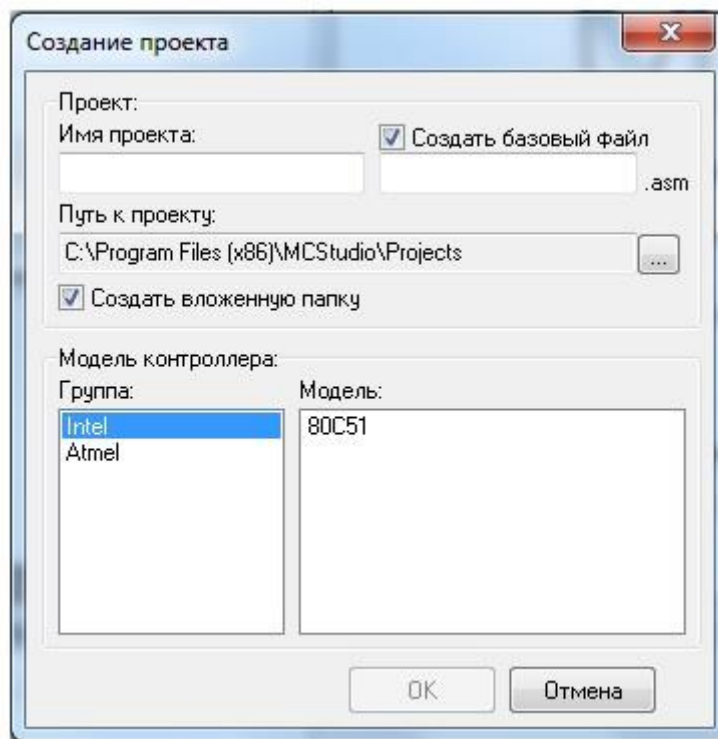


Рисунок 3 – Вид окна «Создание проекта»

## 2. Работа в программной среде MCStudio.

Основные этапы работы с системой MCStudio выполняются через интерфейс главного окна (см. рисунок 4). Главное окно содержит:

- иерархическую систему меню;



- панель инструментов, разделенную на функциональные группы;
- дерево проектов, дерево ресурсов и окно навигатора кода, которые могут отображаться опционально;
- поле редактора в правой части окна, на котором формируются закладки с полями редактирования отдельных файлов;
- поле сообщений внизу окна, на котором представлены закладки **Компиляция, Сообщения, Поиск, Точки останова**;
- строка состояния системы, традиционно размещенная внизу окна.

В главном окне также выполняется автономное тестирование прикладных программ путем симуляции и отображается состояние ресурсов микроконтроллера. Подробно о работе с инструментами главного окна изложено в [1].

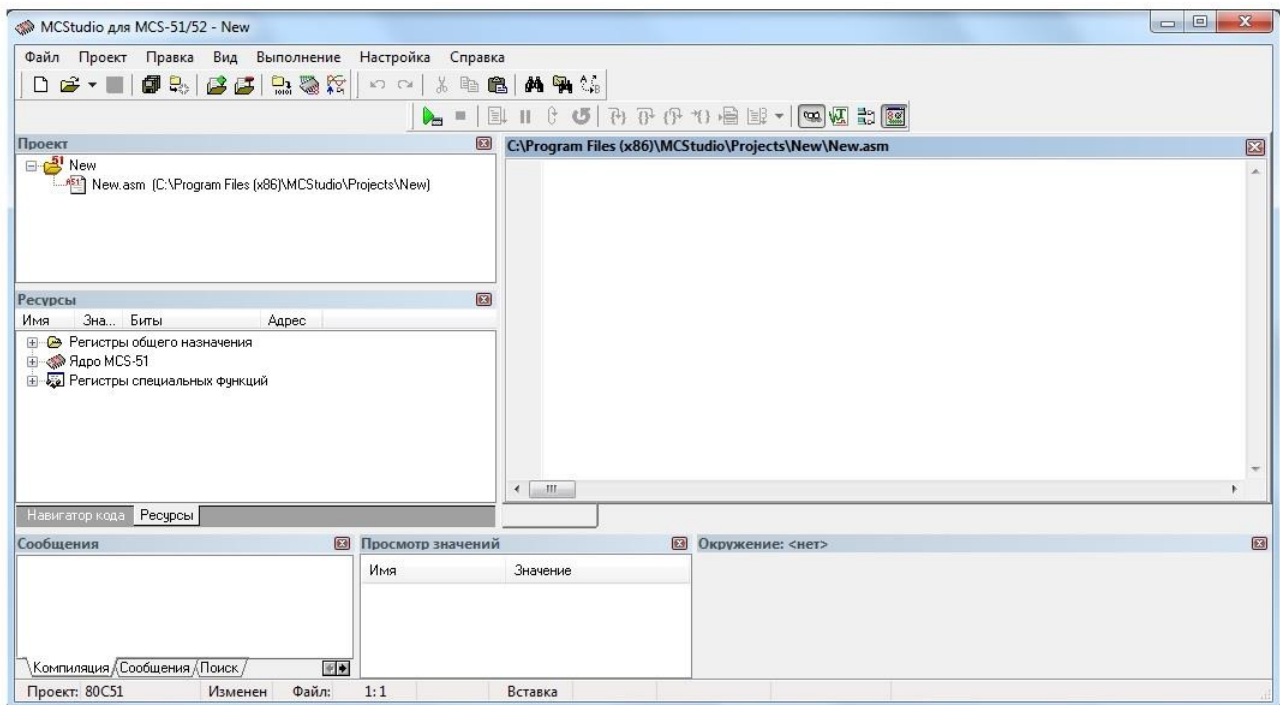


Рисунок 4 - – Вид окна интерфейса главного окна программы

### 3. Формирование структуры проекта.

Система MCStudio ориентирована на формирование пользователем структурированного проекта, который может складываться из нескольких программных модулей, причем подпрограммы, описанные в одних модулях, могут использоваться в других модулях.

Программный код модуля на языке Ассемблер может быть оформлен как абсолютный или как перемещаемый. Тип модуля определяется синтаксическими

конструкциями языка Ассемблер, которые пользователь применил при разработке текста модуля. Однако стартовый код микроконтроллера и таблица векторов прерываний должны быть привязаны к абсолютным адресам. При этом модуль, содержащий абсолютные определения стартового кода и таблицу векторов прерываний, условно считается главным модулем проекта. Способ определения стартового кода и таблицы векторов прерываний являются традиционными:

– например, для стартового кода

```
org 0000h; Задание абсолютного адреса размещения команды jmp  
jmp Main;
```

– для программного перехода в таблице векторов прерывания

```
org 0023h; Задание абсолютного адреса вектора прерывания  
jmp uart_handler ; .
```

#### 4. Ввод кода программы.

При выборе опции **Проект** будет сформирован новый проект с заданным именем и локализацией. На диске будет создана папка с именем проекта и файл <проект>.mcp. Если была выбрана опция **Создать базовый файл**, то одновременно будет создан и открыт в редакторе файл <проект>.asm. При создании текстового документа формируется закладка в редакторе, однако содержимое этого файла не компилируется.

Ввод и редактирование кода программы (в том числе и файла <проект>.asm) выполняется в правом верхнем окне главного окна программы. Операции по редактированию текста совпадают с аналогичными действиями в редакторах систем Borland Delphi или Borland C++ Builder. Также в редакторе действуют функции, описанные в таблице 1. Детально ознакомиться со всеми командами редактирования и установить собственные комбинации клавиш для команд можно через меню *Настройка* → *Опции системы*, выбрав в окне пункт *Редактор* → *Команда*.

При выполнении поиска его результаты представляются на закладке «Поиск». При двойном «клике» левой клавишей мыши (ЛКМ) в списке найденных вхождений слова или выражения курсор будет установлен в позицию редактора с найденным словом или выражением.

На закладке «Точки останова» в панели сообщений отображается перечень установленных точек останова с указанием файла и номера строки. Поле **Адрес** заполняется после компиляции программы. При нажатии правой кнопки мыши (ПКМ) через меню можно установить курсор в редакторе на нужную точку останова программы (также двойным «кликом» ЛКМ) или отменить ее. Точки останова будут обрабатываться симулятором системы MCStudio при автономном тестировании прикладной программы.

Таблица 1 – Основные команды редактирования в текстовом редакторе

Группа	Команда	Клавиши	„Мышь”
Переместить курсор	На начало (конец) строки	Home или Ctrl+↑ (End)	ЛКМ в нужной позиции
	На следующую (предыдущую) строку	↓ (↑)	ЛКМ в нужной позиции
	На начало (конец) страницы	Ctrl+PgUp (PgDn)	ЛКМ в нужной позиции
	На начало (конец) файла	Ctrl+Home (End)	Движок в верхнее (нижнее) положение
Выделить	Слово или фрагмент	Клавиши перемещения + Shift	Перемещать курсор от требуемой позиции, удерживая ЛКМ
	Снять выделение	Переместить курсор	
Удаление	Удалить символ	Delete или BackSpace	–
	Удалить рядок	Ctrl+Y	–
Отмена действия	Отменить предыдущую команду	Ctrl+Z	Нажать кнопку 
Восстановление отмененного действия	Восстановить эффект действия, которое было отменено предыдущей командой	Shift+Ctrl+Z	Нажать кнопку 
Точки останова и трассировки	Установить точку останова на строке	F5, когда курсор стоит на нужной строке	Нажать ЛКМ на сером поле напротив строки
	Снять метку точки останова со строки	Повторное F5, когда курсор стоит на нужной строке	Повторное нажатие ЛКМ на сером поле напротив строки

## 5. Редактор окружения микроконтроллера.


Подсистема формирования описания моделей аппаратных средств, относящихся к окружению микроконтроллера в физической системе, является

средством для приближения процесса тестирования прикладной программы к реальным условиям. Формирование описания окружения не обязательно. Симуляция выполнения программы будет функционировать и без подключения описания окружения.

К аппаратным устройствам, которые могут формировать окружение микроконтроллера, относятся прежде всего средства формирования бинарных сигналов (кнопки, переключатели, генераторы импульсов) и отображения информации (бинарные индикаторы, многосегментные индикаторы). Кроме того, могут быть описаны более сложные устройства, например, формователи и преобразователи аналоговых сигналов.

Результатом работы редактора окружения является специальное окно, в котором размещены визуальные модели аппаратных средств окружения. Модели являются динамическими, то есть в процессе симуляции выполнения прикладной программы с их помощью можно имитировать подачу внешних сигналов на микроконтроллер и выдачу управляющих сигналов на внешние устройства.

Редактор окружения активизируется следующими способами:

- через пункт *Файл* → *Редактор окружения* главного меню системы;
- через кнопку  панели инструментов;
- через файл \*.med с описанием моделей окружения микроконтроллера;
- через пункт **Редактор окружения** в менеджере проектов.

В окне редактора окружения реализована система локального меню и соответствующая панель инструментов, а также три закладки: **Внешнее адресное пространство**, **Окружение контроллера**, **Внешние устройства**.

На закладке **Окружение контроллера** выполняется графическое редактирование свойств физических выводов портов микроконтроллера в системе, для которой создается прикладная программа. Вид окна редактора окружения на данной закладке представлен на рисунке 5. Использование выводов (вход или выход) задается пользователем двойным "кликом" ЛКМ на изображении вывода.

На закладке **Внешние устройства** представлены кнопки для выбора моделей устройств, из которых часто формируется аппаратное окружение микроконтроллера в реальных системах. Пиктограммы виртуальных устройств, реализованных в среде MCStudio, представлены в таблице 2.

При симуляции выполнения программы для корректного функционирования окружения нужно связать каждое виртуальное устройство с определенными линиями портов микроконтроллера. Эти связи имитируют подключение физического устройства к микроконтроллеру. Далее для обозначения связей используется термин "подключение". При подключении моделей устройств автоматически учитывается возможное направление передачи информации. Например, кнопка может быть подключена только к линии порта, для которой установлен режим ввода информации.

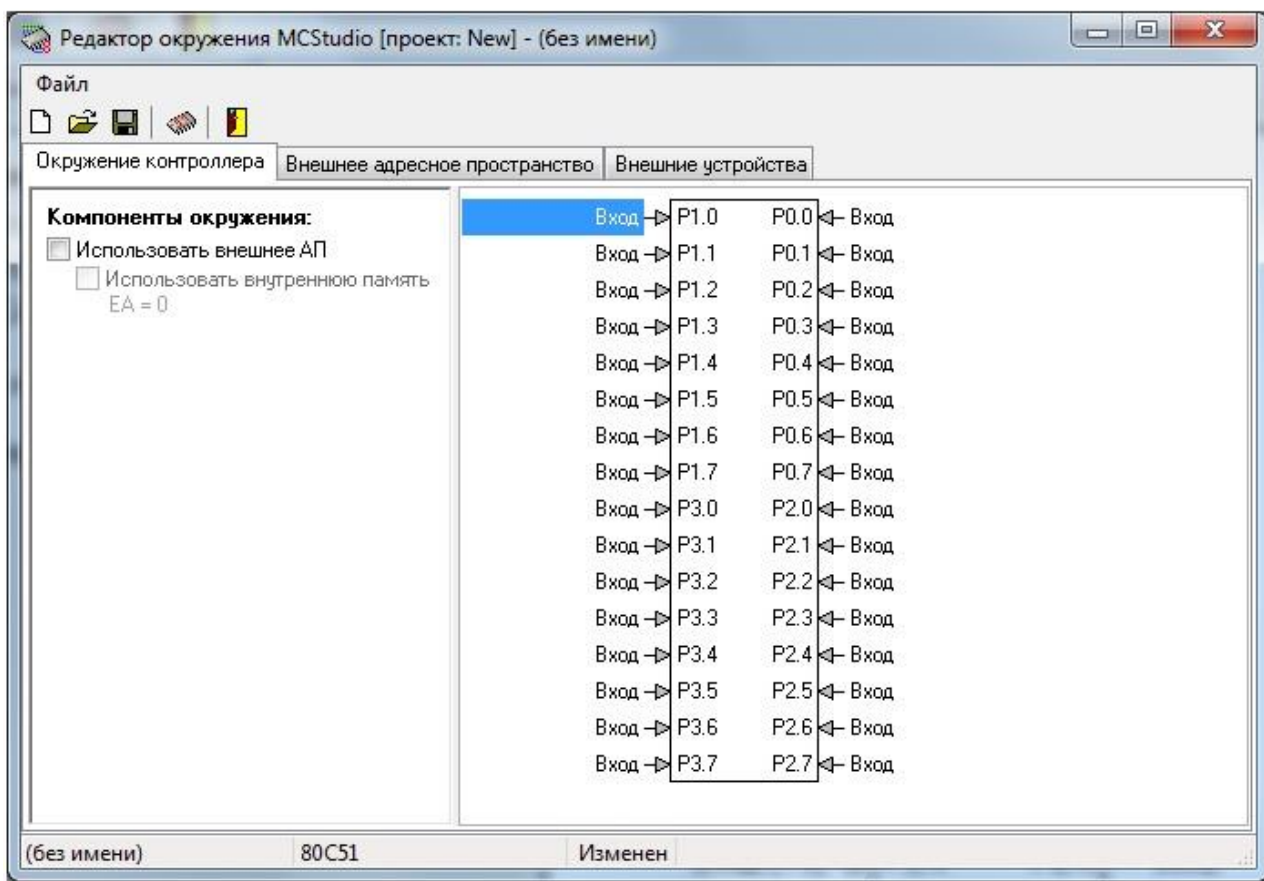


Рисунок 5 – Окно редактора окружения на закладке **Окружение контроллера**

Для всех виртуальных устройств и компонентов редактированию подлежат размеры, размещение на прототипе окна, изображение (для битовых индикаторов и кнопок можно установить произвольный вид).








Для размещения виртуального устройства или компонента на прототипе окна нужно нажать соответствующую кнопку на панели виртуальных устройств, а потом "кликнуть" ЛКМ в нужном месте прототипа окна. Виртуальное устройство или компонент появится на прототипе окна с параметрами, которые установлены как начальные.





Все свойства виртуальных устройств или компонентов, размещенных на прототипе окна, задаются через интерфейс инспектора объектов (размеры и положение также можно изменять непосредственно на прототипе окна).

Инспектор объектов содержит список виртуальных устройств и компонентов, размещенных на прототипе окна, и таблицу с названиями и значениями свойств выбранного устройства или компонента. Компонент, свойства которого нужно редактировать, может быть выбран или в списке в инспекторе объектов, или "кликом" ЛКМ на нужном компоненте в прототипе окна. Для изменения значения

свойства нужно ввести значение в соответствующем правом поле инспектора объектов или выбрать из списка (для порядковых значений).


Таблица 2 – Назначение виртуальных устройств в редакторе окружения

Пиктограмма кнопки	Назначение виртуального устройства или компонента
	Включение режима курсора в редакторе – необходима для прекращения работы с любым компонентом окна
	Битовый индикатор (аналог светодиода). Подключается к отдельной линии порта, назначенной как выход
	Линейка из 8 битовых индикаторов. Подключается к 8-разрядному порту микроконтроллера, линии которого назначены как выход. Каждый из индикаторов может управляться отдельным битом или весь компонент – целым байтом
	Битовая кнопка. Подключается к линии порта, назначенной как вход. Может работать в режимах с фиксацией или без фиксации.
	Линейка из 8 битовых кнопок. Подключается к 8-разрядному порту микроконтроллера, линии которого назначены как входы. Каждая из кнопок формирует отдельный сигнал. Информация от компонента может восприниматься и как целый байт
	<p>Многосегментный индикатор. Реализованы следующие типы:</p> <ol style="list-style-type: none"> <li>1) 7-сегментный индикатор с прямым управлением сегментами</li> <li>2) 7-сегментный индикатор с управлением 4-битовым двоичным кодом и внутренним дешифратором для отображение шестнадцатеричных цифр 0...9, A...F</li> <li>3) 7-сегментный десятичный индикатор с управлением 4-битовым двоичным кодом и внутренним дешифратором для отображение только десятичных цифр 0...9</li> </ol>
	Имитирует аналого-цифровой преобразователь (АЦП), который формирует N-разрядный двоичный код. Входной аналоговый сигнал для АЦП задается моделью потенциометра. Для виртуального устройства задается разрядность АЦП и соответствующее ей количество бит для подключения


Пиктограмма кнопки	Назначение виртуального устройства или компонента
	Виртуальное устройство имитирует работу генератора прямоугольных импульсов с регулируемой частотой и скважностью импульсов. Выход генератора может быть подключен к любому входу микроконтроллера
	Имитация кнопки аппаратного "сброса" микроконтроллера. Не требует подключения к какому-либо выводу на модели МК.
	Комментарий на окне окружения. Редактируется шрифт, цвет, выравнивание. Компонент не связывается с адресом во внешнем АП или линией порта микроконтроллера, то есть не является моделью устройства
	Произвольное изображение, которое может быть установлено в любой части прототипа окна окружения. Компонент не связывается с линией порта микроконтроллера, то есть не является моделью устройства

Самым важным свойством виртуальных устройств является адрес, который связывает модель устройства с линией порта в модели контроллера. При связывании виртуального устройства с адресом используется интерфейс, показанный на рисунке 6. Он активизируется при двойном "клике" ЛКМ на поле **Адрес** в инспекторе объектов. В списке **Байт** выбирается группа выводов микроконтроллера, а в индивидуальных полях битов устанавливаются (в том числе автоматически) распределение разрядов байта по информационным линиям модели устройства.

Размеры прототипа окна могут быть изменены непосредственно в дизайнера интерфейса "перетаскиванием" ЛКМ за правую или нижнюю границы прототипа окна или за его правый нижний угол.

Для завершения редактирования окружения микроконтроллера нужно сохранить описание окружения нажатием на кнопку  или через пункт меню **Сохранить**.

## 6. Компиляция программы.

Компиляция выполняется для всего проекта (по Ctrl+F9 или нажатием кнопки ) с учетом даты изменения отдельных файлов. Для безусловной компиляции всех модулей нужно выполнить построение проекта (меню *Проект* → *Построить* или нажать сочетание клавиш Shift+Ctrl+F9).



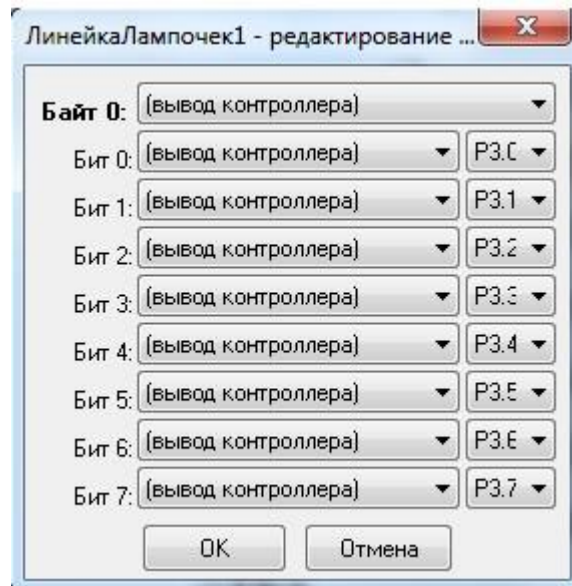



Рисунок 6 – Интерфейс связывания линий виртуального устройства с линиями порта микроконтроллера

Если в процессе компиляции будут обнаружены ошибки в программе, то на закладке окна сообщений **Компиляция** отображаются соответствующие сообщения об ошибке с указанием файла и номера строки программы. Формируются сообщения трех типов: собственно ошибки, предупреждения и подсказки. Наличие ошибки делает невозможной компиляцию определенного модуля.

Предупреждения и подсказки информируют пользователя о неоптимальной или нежелательной организации программы, но процесс компиляции выполняется. Тем не менее, рекомендуется тщательно рассматривать все предупреждения и подсказки, поскольку они часто являются свидетельством наличия логических ошибок в программе.

Если программа была скомпилирована без ошибок, выдается соответствующее сообщение. Кроме того, в текстовом редакторе появляется дополнительное поле с результатом компиляции. В этом поле отображаются физические адреса размещения команд или данных и машинные коды команд и данных в шестнадцатеричном формате. Кнопка  на сером поле редактора позволяет свернуть или развернуть поле с результатами компиляции.

## 7. Симуляция выполнения программы.

Симуляция может выполняться в двух базовых режимах: автоматическом и пошаговом. Автоматическая симуляция предполагает выполнение команд прикладной программы с максимальной скоростью и текущее отображение основных параметров функционирования микроконтроллера (регистр РС,




количество машинных циклов, физическое время) с соответствующим изменением состояния виртуальных устройств в модели окружения микроконтроллера.


При пошаговой симуляции пользователь может "перемещаться" по элементарным фрагментам прикладной программы: по командам ассемблера или по подпрограммам в текстовом редакторе. Одновременно с этим отображается состояние всех аппаратных ресурсов микроконтроллера, которые имеют программно доступные адреса (регистры, память). Также функционируют все виртуальные устройства в модели окружения контроллера.

Симуляция выполнения прикладной программы всегда начинается с адреса 0000h памяти программы, то есть с адреса, на который указывает счетчик команд РС при "сбросе" микроконтроллера. Если функциональная часть программы по определенным причинам должна начинаться с другого адреса, для симуляции в программе нужно предусмотреть "технологический" оператор перехода на начало функциональной части. Обычно это команда `jmp <адрес>`, размещенная в памяти программ по адресу 0000h.

Основные действия пользователя для управления симуляцией определяются кнопками панели инструментов или пунктами меню **Выполнение**.

Перед началом или в процессе симуляции пользователь может установить или отменить точки останова в программе. Самым простым способом является "клик" ЛКМ на сером поле в редакторе напротив нужной команды. Также удобным способом тестирования программы при симуляции является задание точек трассировки.

Независимо от режима симуляции (автоматическая или пошаговая), она всегда начинается с нажатия на кнопку  или клавишу F9. При этом курсор симуляции – синяя полоса в тексте или синий контур на блоках графических программ – устанавливается на первую команду (блок) программы, то есть команду, размещенную по адресу 0000h. Далее симуляция может быть продолжена в любом режиме.

Продолжение симуляции в автоматическом режиме после останова выполняется нажатием на кнопку  или клавишу F9. Поскольку симуляция выполнения команд проводится максимально быстро, то отображение состояния аппаратных ресурсов в дереве ресурсов и в окне просмотра переменных не обновляется в реальном времени после каждой команды, иначе это замедлило бы автоматическую симуляцию. Однако состояние ресурсов программной модели микроконтроллера в симуляторе изменяется в соответствии с выполняемыми командами прикладной программы.

При запуске автоматической симуляции появляется окно контроля за выполнением программы, показанное на рисунке 7. В нем отображены только параметры времени и счетчик команд РС. При отключении опции **Статус программы** единственным динамическим элементом окна остается индикатор

выполнения. Такие ограничения позволяют максимально ускорить симуляцию программы в автоматическом режиме, что важно для больших программ с циклами.

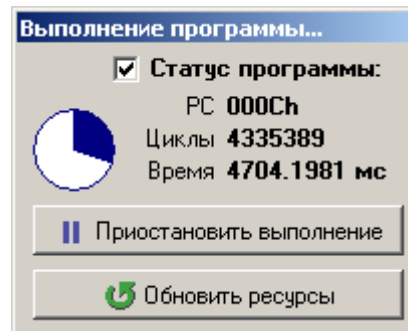




Рисунок 7 – Вид окна контроля за симуляцией выполнения прикладной программы


Нажатие на кнопку **Обновить ресурсы** позволяет обновить значения всех регистров и переменных в дереве ресурсов и окне просмотра в соответствии с программной моделью микроконтроллера на момент нажатия этой кнопки.


Автоматическая симуляция прекращается нажатием на кнопку , или комбинацией клавиш Ctrl-F2, или путем выбора пункта меню **Выполнение–Прервать выполнение**. При этом процесс симуляции полностью завершается, но все ресурсы модели контроллера сохраняют свои текущие значения, то есть пользователь может их просмотреть.

Использование пошаговой симуляции позволяет детально проследить все аспекты выполнения прикладной программы, что обычно нужно для выявления ошибок. Если в процессе выполнения любого из этих действий будет достигнута точка останова, симуляция приостанавливается. Кроме того, в некоторых случаях для перехода через блок программы может использоваться команда меню **Установить точку выполнения**, которая устанавливает значение счетчика команд РС на адрес, который имеет выделенная строка редактора, не выполняя промежуточных команд.

После выполнения одного шага симуляции пользователь может просмотреть или изменить содержимое любых ресурсов контроллера или ячеек памяти.

Во всех вариантах пошаговой симуляции пользователь может использовать интерфейс модели окружения микроконтроллера (активизация окна кнопкой  или пунктом меню **Вид – Окружение**).

В любой момент пошаговой симуляции пользователь может перейти к автоматической симуляции, нажав на кнопку  или клавишу F9. Завершить

симуляцию можно, нажав на кнопку , или комбинацию клавиш Ctrl-F2, или выбором соответствующего пункта меню.

## 8 Средства отображения состояния ресурсов микроконтроллера при симуляции.

Все аппаратные ресурсы микроконтроллера, имеющие программно-доступные адреса, отображаются в дереве ресурсов. Как правило, дерево ресурсов отображается слева от панели редактора. Оно может быть активизировано через пункт меню *Вид* → *Дерево ресурсов*.

Дерево ресурсов имеет три группы параметров:

- **Регистры общего назначения** – ACC, В, PSW, регистры текущего активного банка и регистры всех четырех банков (R0...R7);
- **Внутренний процессор MCS-51** – регистры PC и DPTR, счетчик машинных циклов с начала программы, физическое время, нужное для выполнения программы на микроконтроллере, его тактовая частота;
- **Регистры специальных функций** – арифметические и указательные регистры (повторно отображаются ACC, В, PSW и DPTR), а также все регистры для периферийных блоков: таймеров, параллельных и последовательного портов, контроллера прерываний и др.

Для каждого типа периферийных блоков в одной группе объединены как информационные регистры (например, для таймера T0 – TL0 и TH0), так и средства управления именно этим блоком (для таймера T0 – соответствующие биты регистров TMOD и TCON). Такое объединение облегчает контроль за процессом функционирования блоков во время симуляции.

Для каждого регистра отображаются его программное имя, содержимое регистра, битовый вид содержимого регистра и его физический адрес. При "клике" ПКМ в контекстном меню можно выбрать систему счисления для отображения содержимого регистра.

Текущее содержимое любого регистра может быть изменено динамически, то есть даже без останова симуляции программы. Для этого нужно установить курсор на названии или значении регистра, дважды "кликнуть" ЛКМ и в появившемся окне задать новое значение. Кроме того, можно изменить значение любого бита регистра "кликом" ЛКМ, установив курсор на этом бите в колонке **Биты** или в дереве для регистров с битовой адресацией (например, ACC, В, PSW, TCON).

Такие ресурсы микроконтроллера или системы на его основе, как области памяти, отображаются в отдельных окнах в виде двумерных массивов. Для этого нужно выбрать требуемую область через пункт меню **Вид**. Вид окна области резидентной памяти данных (РПД) показан на рисунке 8.

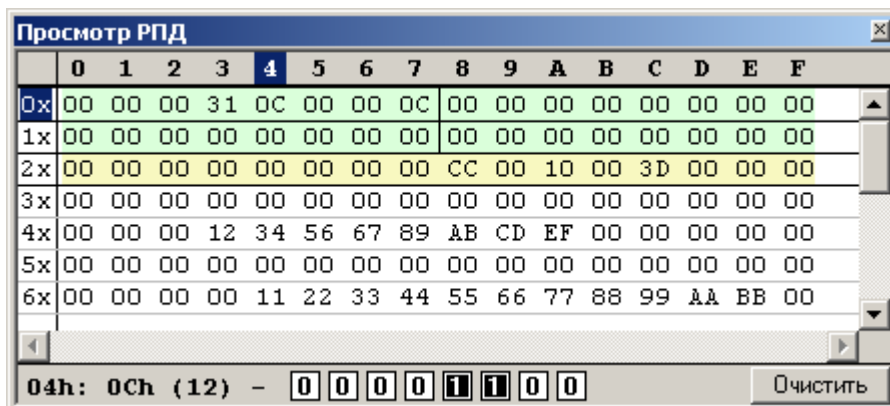


Рисунок 8 – Вид окна отображения содержимого области РПД

В каждой строке представлено 16 байтов, содержимое байта отображается как шестнадцатеричное число. Адрес каждого байта формируется из номера строки, указанного слева и номера колонки, указанного сверху. Для байта, на котором находится курсор, внизу окна побитно отображается содержимое байта. Значение любой ячейки может быть изменено или непосредственно в массиве прямым вводом нового значения (без Enter), или установив курсор на битах внизу окна и нажатием ЛКМ. Кнопка **Очистить** позволяет "обнулить" значения всех байтов данной области, что выполняется только с целью тестирования.

На рисунке 8 зеленым цветом выделена область банков регистров общего назначения, а желтым – область РПД с битовой адресацией. В окнах, которые отображают большие области памяти, можно быстро достичь нужной адресной области, если перевести курсор в левую полосу с адресами и ввести нужный адрес. Альтернативный способ – скролинг.

Более подробно с возможностями системы MCStudio и способами работы с ней можно познакомиться в [1].

## ЛАБОРАТОРНАЯ РАБОТА №1 ВЫПОЛНЕНИЕ АРИФМЕТИЧЕСКИХ ОПЕРАЦИЙ

**ЦЕЛЬ РАБОТЫ:** ознакомление с архитектурой арифметического узла микроконтроллера, основами программирования арифметических операций и приобретение навыков работы с симулятором.

### ОБЩИЕ СВЕДЕНИЯ

Для выполнения лабораторной работы необходимо изучить организацию

оперативного запоминающего устройства (ОЗУ), организацию регистров общего назначения (РОН), способы адресации данных и команды микроконтроллера MCS51 по пересылке и обработке данных. Программа пишется на языке Ассемблер MCS51 [3].

У микроконтроллера MCS51 память программ и ОЗУ (память данных) разделены на две независимые области. Программа пользователя хранится в памяти программ, и первая исполняемая команда должна располагаться в ячейке с адресом 0000h. Указание физического адреса размещения команды выполняется директивой **ORG**.

Объем ОЗУ составляет 128 ячеек с однобайтовой организацией [4]. Младшие 32 ячейки составляют регистры общего назначения (РОН), организованные в 4 банка по 8 ячеек (обозначаются R0 ... R7) в каждом банке. Выбор рабочего банка осуществляется путем задания его номера в регистре PSW – слове состояния процессора (биты PSW.4 – старший и PSW.3 – младший). Регистры R0 и R1 в любом из банков могут использоваться в качестве регистров для косвенной адресации данных.

Арифметические и логические операции выполняются в процессорной части микроконтроллера [2]. Одним из операндов, а также местом хранения результата операции является особый регистр специальных функций (SFR) – аккумулятор. Символическое имя аккумулятора **A** или **ACC** в зависимости от используемой команды. При выполнении арифметических операций изменяются значения разрядов (так называемые флаги) регистра PSW. Чаще всего используются флаги переполнения разрядной сетки (бит PSW.2) и флаг переноса (заема) C (бит PSW.7).

В работе в некоторых операциях требуется изменить знак числа. Напомним, что отрицательные числа при выполнении арифметических операций следует представлять в дополнительном коде. Для преобразования положительного числа (прямой код) в отрицательное число (дополнительный код) следует побитно инвертировать преобразуемое число и к результату прибавить единицу. Другой способ – вычесть преобразуемое число из числа ноль.

Для удобства записи команд допускается использовать числовые значения и адреса переменных в виде вычисляемых выражений. В ходе компиляции программы значения выражений будут вычислены и полученные численные значения поставлены в команды. Кроме того, допускается использование символических обозначений, которые вводятся с помощью директивы **EQU** (равно), например **X EQU 100**. Эта директива позволяет заменять явное указание значений констант и адресов переменных в памяти данных на понятные символьные обозначения.

При написании программы могут пригодиться операторы **HIGH** и **LOW**, которые выделяют соответственно старший и младший байты из двухбайтового числа. Например, результатом оператора **HIGH(07A4h)** будет число 07h, а результатом оператора **LOW(07A4h)** будет число 0A4h.

Команда **mov Rg1,Rg2** помещает значение из ячейки с адресом Rg1 в ячейку с адресом Rg2 оперативной памяти. В качестве адресов могут быть использованы числа или их символические обозначения, а также системные имена регистров специальных функций. Примером такого регистра может служить аккумулятор А. Кроме того, если возникает необходимость записать в ячейку некоторое число, а не значение из другой ячейки, то может быть использован символ #, обозначающий непосредственную адресацию. Например, чтобы записать в аккумулятор число 30, может быть использована команда **mov A,#30**.

Арифметические операции сложения и вычитания **add A,Rg1**, **addc A,Rg1** и **subb A,Rg1** выполняют действие с двумя операндами, один из которых хранится в аккумуляторе, а другой – в ячейке с адресом Rg1. Результат операции помещается в аккумулятор. В случае, если в результате выполнения операции образуется перенос в старший байт или заем, то в регистре слова состояния процессора **PSW** устанавливается в 1 флаг **C**.

Выше были приведены две команды сложения. Команда **ADDC** выполняет то же действие, что и команда **ADD**, но дополнительно прибавляет к результату сложения значение флага **C**. Эта команда позволяет выполнять сложение многобайтовых чисел. Команда **SUBB** всегда использует значение флага **C** в качестве заема.

Для выполнения операций умножения и деления помимо аккумулятора А используется вспомогательный служебный регистр В. Команда **mul AB** выполняет умножение двух однобайтовых чисел, хранящихся в регистрах А и В. Двухбайтовый результат умножения помещается в регистры А (младший байт) и В (старший байт). Команда **div AB** выполняет деление содержимого аккумулятора на содержимое регистра В. Частное от деления помещается в регистр А, а остаток – в регистр В.

Пример программы сложения двух двухбайтовых чисел приведен в листинге 1. Выполняется сложение двух чисел Val1=1200 и Val2=2100. Переменные хранятся в памяти данных побайтно. Результат сложения записывается по адресу переменной Rez.

В листинге 1 первая строка программы – текстовый комментарий. Он служит для повышения читаемости программы и всегда начинается с символа ; (точка с запятой). Текстовые комментарии компилятором не обрабатываются.

Строки 2 – 10 содержат макроопределения. В них задаются имена переменных и адреса их расположения в памяти данных микроконтроллера. Эти макроопределения служат для указания компилятору о принятых обозначениях (именах) переменных или констант. Они не транслируются в команды микроконтроллера и не записываются в память программ.

Программа работы микроконтроллера начинается со строки 11. Директива **org 0** указывает, что располагающиеся далее команды должны последовательно располагаться в памяти программ, начиная с абсолютного адреса 0.

### Листинг 1 – Пример программы сложения двух чисел

```
1 ; Адреса расположения первой переменной
2 VAL1L EQU 21h ; младший байт
3 VAL1H EQU 22h ; старший байт
4 ; Адреса расположения второй переменной
5 VAL2L EQU 23h ; младший байт
6 VAL2H EQU 24h ; старший байт
7 ; Адреса расположения результата вычислений
8 RezL EQU 25h
9 RezH EQU 26h
10 RezE EQU 26h
11 ; Начало программы
12 org 0
13 jmp Main
14 ; Область векторов прерываний
15 ; Основная программа
16 org 30h
17 Main:
18 ; Задаем значения переменных
19 mov Val1L,#Low(1200)
20 mov Val1H,#High(1200)
21 mov Val2L,#Low(2100)
22 mov Val2H,#High(2100)
23 ; Сложение младших байт
24 mov A,Val1L
25 add A,Val2L
26 mov RezL,A ; сохранение результата
27 ; Сложение старших байт с учетом бита
28 ; переноса
29 mov A,Val1H
30 addc A,Val2H
31 mov RezH,A ; сохранение результата
32 ; Учтем возможный перенос при сложении
33 ; старших байт
34 mov A,#0
35 addc A,#0
36 mov RezE,A
37 Stop: jmp Stop ; зацикливание
38 End
```

Первой является команда `jmp Main` (сокращение слова `jump` – прыжок), задающая безусловный переход на метку `Main`. Метке `Main` в этой программе соответствует ячейка памяти `30h`, что определяется директивой `org 30h` в строке 16. Область памяти программ от ячейки 0 до ячейки `30h` в данной программе не используется, но в дальнейшем она будет применена для задания команд обработки прерываний программы. Эта особенность организации программы характерна для микроконтроллеров семейства `MCS51`.

Далее в строках 19 – 22 побайтно командами `mov` задаются значения переменным `Val1` и `Val2`. Команда `mov` является сокращением слова `move` – передвигать. В ячейки памяти с адресами, определенными в начале программы с помощью макроопределений, помещаются значения, задаваемые непосредственно в виде десятичных чисел.

В строках 24 – 26 записаны команды сложения младших байт переменных. Младший байт одной из переменных помещается в аккумулятор `A`, к нему командой `add` (`add` – прибавлять) прибавляется младший байт другой переменной. Результат сложения записывается по адресу `RezL`. При выполнении арифметических операций модифицируются признаки в слове состояния процессора `PSW`. В частности, возможно возникновение единицы переноса в старший байт суммы. Бит переноса заносится в бит `C` слова состояния процессора.

В строках 29 – 31 записаны команды сложения старших байт переменных. Вместо команды `add` использована команда `addc`, которая задает сложение двух байт с прибавлением значения бита переноса `C`. Результат операции записывается по адресу `RezH`.

При сложении старших байт также может получиться единица переноса в следующий байт. Поэтому результат сложения двух двухбайтовых чисел в общем случае требует 3 байта для хранения результата. Вычисление значения третьего байта суммы записано в строках 34 – 36 программы.

Команда в строке 37 закичивает выполнение программы, т.е. выполняет останов хода выполнения программы. Директива `End` в строке 38 указывает компилятору на окончание текста программы.

Система команд микроконтроллеров семейства `MCS51` приведена в Приложении 1.

## ЗАДАНИЕ И ВАРИАНТЫ ВЫПОЛНЕНИЯ РАБОТЫ

В соответствии с вариантом задания необходимо написать программу, реализующую многобайтовую арифметическую операцию сложения или вычитания заданных чисел. Кроме того, в программу нужно включить выполнение однобайтовой операции умножения или деления младших (старших) байт заданных чисел. Полученные промежуточные результаты использовать для вычисления итогового результата в соответствии с заданной формулой.

Варианты выполнения лабораторной работы приведены в таблице 3. В таблице



обозначены: мл(\*) – младший байт числа, ст(\*) – старший байт числа, ост(\*) – остаток от деления чисел.

Таблица 3 – Варианты выполнения лабораторной работы

№ варианта	Операнд X	Операнд Y	Формула
1	20540	8922	$Z=X+Y+x/y$
2	870	7432	$Z=X-Y+x/y$
3	1040	18400	$Z=X+Y+мл(x*y)$
4	1570	6954	$Z=X+Y-мл(x*y)$
5	1830	6711	$Z=X+Y-x/y$
6	2059	5555	$Z=X-Y-x/y$
7	2387	5367	$Z=X-Y+ст(x*y)$
8	2856	4735	$Z=X-Y-ст(x*y)$
9	3489	4357	$Z=X-Y-x/y$
10	3650	4021	$Z=X+Y+ост(x/y)$
11	4026	3658	$Z=X-Y+ост(x/y)$
12	4374	3465	$Z=X+Y+x*y$
13	4745	2378	$Z=X+Y-x*y$
14	5325	2039	$Z=X+Y-ост(x/y)$
15	5555	1835	$Z=X-Y-ост(x/y)$
16	6732	1569	$Z=X-Y+x*y$
17	6943	1063	$Z=X-Y-x*y$
18	7164	872	$Z=X-Y-ост(x/y)$
19	7458	544	$Z=X+Y+ост(x/y)$
20	8916	331	$Z=X-Y+ост(x/y)$

### ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. В соответствии с вариантом задания написать программу работы микроконтроллера на языке Ассемблер MCS51. Запустить программу MCStudio и выбрать *Создать новый проект или файл -> Создать проект->Ок* с выбором личной папки и оригинальным именем проекта. В качестве модели микроконтроллера рекомендуется выбрать Intel 80C51 или Atmel AT89S8252.
2. В открывшемся главном окне проекта набрать текст программы. При разработке программы учесть, что для задания численных значений переменных следует использовать непосредственную адресацию (символ #), при выполнении промежуточных вычислений использовать прямую и косвенную (символ @) адресации. Сохранить проект на диске компьютера.

Детально ознакомится со всеми командами редактирования и установить собственные комбинации клавиш для команд можно через меню *Настройка -> Опции системы*, выбрав в окне пункт *Редактор -> Команда*.

3. Нажать Ctrl-F9 или выбрать *Проект ->Компилировать->Ок* и выполнить компиляцию программы (т.е. перевод текстового файла в двоичный код для микроконтроллера). Исправить обнаруженные компилятором синтаксические ошибки.
4. Проверить работу программы на симуляторе. Для этого запустить симуляцию, выбрав *Выполнение -> Запустить симуляцию* или нажать F9 (дважды). При запущенной симуляции на экране отображается окно **Выполнение программы** с круговой диаграммой хода машинного времени. Открыть окно **РПД – просмотр**, выбрав *Вид -> Резидентная память данных*. В ячейках резидентной памяти данных считать содержимое исходных переменных и результата вычислений. Полученные численные значения должны соответствовать числу, вычисленному на калькуляторе по заданной формуле. Сохранить проект и продемонстрировать работу программы преподавателю.
5. Оформить отчет о работе.

### СОДЕРЖАНИЕ ОТЧЕТА

Отчет должен содержать:

1. Титульный лист
2. Цель работы и конкретный вариант задания
3. Листинг программы работы микроконтроллера
4. Результаты работы программы в виде скриншотов соответствующих окон системы моделирования
5. Выводы по работе

### КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое разрядность представления данных и какие характеристики системы она определяет?
2. Чем отличается оперативная память микроконтроллера от памяти программ?
3. В чем состоит назначение аккумулятора?
4. Чем отличаются результаты выполнения команд `mov A, X` и `mov A, #X`?
5. Что такое банк регистров общего назначения?

## ЛАБОРАТОРНАЯ РАБОТА №2 УПРАВЛЕНИЕ ХОДОМ ВЫЧИСЛИТЕЛЬНОГО ПРОЦЕССА

**ЦЕЛЬ РАБОТЫ:** освоение способов адресации данных и команд управления ходом вычислений.

## ОБЩИЕ СВЕДЕНИЯ

Лабораторная работа направлена на выработку навыков организации и управления ходом вычислительного процесса, использования команд условных и безусловных переходов, способов организации циклических вычислений.

Набор команд микроконтроллеров семейства MCS51 поддерживает следующие типы адресации данных [3].

Прямая адресация (Direct Addressing) – операнд определяется 8-битовым адресом, указанным в команде. Например: `mov A, 45h` – скопировать в аккумулятор содержимое ячейки ОЗУ №45h. Эти команды могут занимать до трех байт в памяти программ.

Косвенная адресация (Indirect Addressing) – в команде указывается адрес РОН, содержащий адрес операнда. Допускается использовать только регистры R0 или R1 выбранного банка регистров и некоторые регистры специальных функций. Например: `mov A, @R1` – скопировать в аккумулятор содержимое ячейки ОЗУ, адрес которой хранится в регистре R1.

Регистровая адресация (Register Instruction) – в команде указываются адреса регистров R0 ... R7 выбранного банка регистров. Выбор одного из четырех доступных регистровых банков осуществляется программированием битов селектора банка (RS1, RS0) в слове состояния процессора PSW. Например: `mov A, R1` – скопировать в аккумулятор содержимое регистра R1. Эти команды являются самыми короткими, они могут занимать всего один байт в памяти программ.

Непосредственная адресация (Immediate Constants) – операнд записан непосредственно в поле команды. Например: `mov A, #45h` – записать в аккумулятор число 45h. Считываемое значение хранится в памяти программ и не может быть изменено в ходе ее выполнения.

Индексная адресация (Indexed Addressing) – число читается из памяти программ с использованием специального базового регистра (регистры DPTR или PC) и индексного регистра (обычно – аккумулятора). Адрес элемента вычисляется сложением содержимых базового и индексного регистров. Обычно используется при работе с таблицами данных, хранящихся в памяти программ. Например: `mov A, @A+DPTR` – записать в аккумулятор число из ячейки памяти программ, адрес которой вычисляется суммированием содержимого аккумулятора и регистра DPTR.

Некоторые из описанных выше типов адресации использовались при выполнении лабораторной работы №1.

В ходе работы по программе осуществляется последовательное выполнение команд, хранящихся в памяти программ. Команда может содержать операнды и в зависимости от типа адресации команда может занимать в памяти программ от 1

до 3 байт. Первый байт любой команды содержит код операции, второй и третий байты содержат либо адреса операндов, либо непосредственно операнд. Адрес текущей ячейки памяти определяется содержимым регистра счетчика команд **PC**, которое постоянно увеличивается на 1 и указывает сначала на код команды, а затем на операнды. Таким образом, содержимое **PC** изменяется автоматически в зависимости от кода последней выполненной команды.

Однако иногда возникает необходимость нарушить естественный ход выполнения команд. Для этого используются команды переходов. Команда `jmp address` - команда безусловного перехода (сокращение слова `jump` – прыжок) к точке программы, указанной в поле `address`. Непосредственное указание адреса при программировании неудобно, поэтому компиляторы допускают использования в поле `address` строковой метки. Метка предшествует исполняемой команде и отделяется от нее символом `:` (двоеточие). При компиляции программы метка будет заменена соответствующим адресом исполняемой команды. Таким образом, последовательности команд, представленные в листинге 2 и листинге 3, эквивалентны.

Листинг 2 – Пример безусловного перехода по адресу

```
1  org 0
2  jmp 30h
3  ; Область векторов прерываний
4
5  ; Основная программа
6  org 30h
7  mov Val1L, #Low(1200)
```

Листинг 3 – Пример безусловного перехода с использованием метки

```
1  org 0
2  jmp Main
3  ; Область векторов прерываний
4
5  ; Основная программа
6  org 30h
7  Main: mov Val1L, #Low(1200)
```

Рассмотренная выше команда `jmp` позволяет выполнить переход независимо от результатов обработки данных. Однако часто требуется выполнить переход только при выполнении некоторого условия.

Команда `jb bit, address` – команда (сокращение `jump bit`) перехода по адресу или метке `address`, если значение бита `bit` равно 1. Например: `jb`

ACC.2, *metka* – переход на метку *metka*, если бит 2 аккумулятора содержит единицу. Команда `jnb bit, address` (сокращение `jump not bit`) осуществляет переход при противоположном условии – если проверяемый бит содержит ноль.

Среди многих бит для организации условных переходов чаще всего используется бит переноса **C**. Для их задания система команд MCS51 имеет специальные команды: `jc address` и `jnc address`. Эти команды задают переход по указанному адресу или метке, если в бите переноса содержится 1 или 0, соответственно.

Система команд MCS51 содержит и более сложные конструкции. Так, имеется команда, позволяющая сравнить содержимое регистра с заданным значением и сделать переход по заданному адресу в случае их неравенства. Например, по команде `cjne R2, #20h, address` (сокращение `compare and jump not equal` – сравнить и перейти, если не равно) выполняется сравнение содержимого регистра **R2** с числом 20h и осуществляется переход по адресу (или к метке) *address*. Достаточно часто используется команда `djnz Rn, address` (сокращение `decrement and jump not zero` – уменьшить на 1 и перейти, если не равно нулю), выполняющая уменьшение на 1 содержимого регистра **Rn** ( $n=0 \dots 7$ ) и переход по указанному адресу, если итог не равен нулю.

В качестве примера рассмотрим программу вычисления суммы 10 однобайтовых чисел, хранящихся в памяти программ в виде массива. Будем считать, что массив содержит числа 100, 59, 25, 64, 17, 201, 148, 15, 92, 60. Результат будем сохранять в переменной *Rez*, прибавляя числа в цикле. Максимально возможное значение суммы 10 чисел равно 2550, поэтому для хранения результата выделим 2 байта. Текст программы приведен в листинге 4.

В приведенной программе регистр **R0** используется для счета количества реализованных циклов сложения. Данные (байты) хранятся в памяти программ в массиве *TAB[i]*. Массив описан в конце программы, и его положение определено меткой *TAB*. Данные в массив занесены простым перечислением через запятую, но перед данными использована директива `db` (байты данных). Регистр **R1** используется для хранения индекса *i* массива данных, который в каждом цикле увеличивается на 1.

В строках 2-3 содержатся макроопределения. В них задаются имена байт переменной результата и их расположение в памяти данных микроконтроллера.

Непосредственно программа работы микроконтроллера начинается со строки 5. Директива `org 0` указывает, что располагающиеся далее команды должны последовательно располагаться в памяти программ, начиная с абсолютного адреса 0.

Первой выполняется команда `jmp Main`, задающая безусловный переход на метку *Main*. Метке *Main* в этой программе соответствует ячейка памяти 30h, что определяется директивой `org 30h` в строке 10. Область памяти программ от

ячейки 0 до ячейки 30h в данной программе не используется, но в дальнейшем она будет применена для задания команд обработки прерываний программы. Эта особенность организации программы характерна для микроконтроллеров семейства MCS51.

#### Листинг 4 – Пример программы сложения чисел в цикле

```
1 ; Адреса расположения результата вычислений
2 RezL EQU 20h
3 RezH EQU 21h
4 ; Начало программы
5 org 0
6 jmp Main
7 ; Область векторов прерываний
8
9 ; Основная программа
10 org 30h
11 Main:
12 ; Задаем начальные значения переменных
13 mov R0,#10; Счетчик числа итераций
14 mov R1,#0; Смещение в массиве данных
15 mov DPTR,#TAB; Задание базового адреса таблицы
16 ;      данных
17 mov RezH,#0; Обнуление результата
18 mov RezL,#0
19 ;Вычислительный цикл
20 Circle:mov A,R1; Задание смещения в таблице данных
21 movc A,@A+DPTR; Чтение байта из памяти программ
22 add A,RezL; Сложение с результатом
23 mov RezL,A ; Сохранение результата
24 ;Коррекция старшего байта с учетом бита переноса
25 mov A,RezH
26 addc A,#0
27 mov RezH,A ; Сохранение результата
28 inc R1; Увеличение на 1 смещения в массиве данных
29 djnz R0,Circle; Счет циклов и заикливание
30 ;      программы
31 Stop: jmp Stop ; Останов вычислительного процесса
32 ; Массив данных
33 TAB: db 100,59,25,64,7,201,148,15,92,60
34 End
```

Далее в строках 13 – 18 задаются начальные значения переменных программы – счетчика числа циклов, смещения в массиве данных, базового адреса массива и нулевое начальное значение результата вычислений.

Вычислительный цикл начинается со строки 20. Задается смещение в таблице данных. Далее в строках 21-27 осуществляется чтение очередного байта из массива данных и сложение его с результатом предыдущих вычислений.

В строке 28 выполняется увеличение индекса массива данных на 1.

В строке 29 декрементируется счетчик циклов и результат сравнивается с нулем. Если число в счетчике циклов отлично от нуля, то осуществляется переход на метку `Circle` для чтения следующего числа и продолжения вычислений. Если число в счетчике циклов равно нулю, то осуществляется переход к следующей команде (строка 29).

Команда в строке 31 выполняет останов хода выполнения программы. После завершения вычислительного процесса в строке 33 записана таблица констант, хранящихся в памяти программ. Начало таблицы отмечено меткой `Tab`. Директива `db` означает начало последовательности однобайтовых чисел, разделенных запятыми. Числа располагаются в последовательно расположенных ячейках в памяти программ.

Директива `End` в строке 34 указывает компилятору на окончание текста программы.

Система команд микроконтроллеров семейства `MCS51` приведена в Приложении 1.

## ЗАДАНИЕ И ВАРИАНТЫ ВЫПОЛНЕНИЯ РАБОТЫ

Написать программу, реализующую циклическое увеличение (уменьшение) двухбайтового числа, хранящегося в ОЗУ. Увеличение (уменьшение) осуществляется на однобайтовую величину, последовательно считываемую из массива данных, хранящихся в памяти программ.

Считываемые из памяти программ данные используются для вычисления результата, если выполнено условие, указанное в задании. Критерий прекращения вычислений также задается заданием.

Рекомендуемые значения для массива данных: 209, 78, 203, 251, 146, 225, 170, 91, 15, 92, 58, 55, 217, 39, 162, 23, 112, 8, 227, 200, 17, 116, 200, 64, 105.

Варианты выполнения лабораторной работы представлены в таблице 4.

## ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. В соответствии с вариантом задания написать программу работы микроконтроллера на языке Ассемблер `MCS51`. Для этого следует запустить программу `MCStudio` и выбрать *Создать новый проект или файл -> Создать проект->Ок* с выбором личной папки и оригинальным именем проекта. В качестве модели микроконтроллера рекомендуется выбрать `Intel 80C51` или `Atmel`

AT89S8252.

В открывшемся главном окне проекта набрать текст программы. Сохранить текст на диске компьютера.

Таблица 4 – Варианты выполнения лабораторной работы

№ варианта	Операция	Условие использования байта данных	Критерий прекращения вычислений
1	увеличение	>20	Операция выполнена 10 раз
2	увеличение	>40	Операция выполнена 12 раз
3	увеличение	>50	Сумма >1024
4	увеличение	<250	Операция выполнена 7 раз
5	увеличение	<150	Сумма >500
6	увеличение	четное число	Операция выполнена 5 раз
7	увеличение	четное число	Сумма >2000
8	увеличение	четное число	Операция выполнена 7 раз
9	увеличение	четное число	Сумма >800
10	увеличение	четное число	Операция выполнена 10 раз
11	уменьшение	нечетное число	Сумма <-1024
12	уменьшение	нечетное число	Операция выполнена 11 раз
13	уменьшение	нечетное число	Сумма <-512
14	уменьшение	нечетное число	Операция выполнена 9 раз
15	уменьшение	нечетное число	Сумма <-2048
16	уменьшение	нечетное число	Операция выполнена 6 раз
17	уменьшение	>45	Сумма <-400
18	уменьшение	>100	Сумма <-850
19	уменьшение	>80	Операция выполнена 7 раз
20	уменьшение	<200	Сумма <-1500

2. Нажать Ctrl-F9 или выбрать *Проект ->Компилировать->Ok* и выполнить компиляцию программы (т.е. перевод текстового файла в двоичный код для микроконтроллера). Исправить обнаруженные компилятором синтаксические ошибки.

3. Проверить работу программы на симуляторе. Для этого запустить симуляцию, выбрав *Выполнение -> Запустить симуляцию* и нажать F9 (дважды). При запущенной симуляции на экране отображается окно **Выполнение программы** с круговой диаграммой хода машинного времени. Открыть окно **РПД – просмотр**, выбрав *Вид -> Резидентная память данных*. В ячейках резидентной памяти данных считать содержимое результата вычислений. Полученные численные значения должны соответствовать числу, вычисленному на калькуляторе по заданному алгоритму. Сохранить проект и



продемонстрировать работу программы преподавателю.

4. Оформить отчет о работе.

### СОДЕРЖАНИЕ ОТЧЕТА

Отчет должен содержать :

1. Титульный лист
2. Цель работы и конкретный вариант задания
3. Листинг программы работы микроконтроллера
4. Результаты работы программы в виде скриншотов соответствующих окон системы моделирования
5. Выводы по работе

### КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какое число разрядов имеет регистр **DPTR**?
2. Какое число разрядов имеет регистр **PC**?
3. В чем различие а применении имен аккумулятора **A** и **ACC**?
4. Что обозначает команда `cjne @R0, #40, Stop` ?
5. Что обозначает команда `jmp @A+DPTR` ?

## ЛАБОРАТОРНАЯ РАБОТА №3 ИСПОЛЬЗОВАНИЕ ПОДПРОГРАММ

**ЦЕЛЬ РАБОТЫ:** выработать навыки использования подпрограмм при разработке программы работы микроконтроллера.

### ОБЩИЕ СВЕДЕНИЯ

При разработке программы работы микроконтроллера широко используется механизм работы с подпрограммами – автономными программными модулями, которые решают некоторую частную задачу и могут вызываться из различных участков основной программы [3].

Для вызова подпрограммы используется команда `call address` (`call` – звать), которая вызывает подпрограмму, размещенную по указанному адресу или метке. Вызванная подпрограмма выполняется однократно, и после ее завершения контроллер продолжает выполнение основной программы. Поэтому при вызове подпрограммы контроллер запоминает адрес точки возврата, т.е. номер команды, которая будет выполняться после завершения подпрограммы.

Вызов подпрограммы предполагает, что в памяти данных выделена специальная область, называемая стеком. Для работы со стеком используется

особый регистр специальных функций **SP** (сокращение слов Stack Pointer – указатель стека). Этот регистр содержит адрес свободной ячейки стека. При записи в стек сначала байт записывается в указанную ячейку, а затем содержимое **SP** увеличивается на 1. При чтении сначала содержимое **SP** уменьшается на 1, а затем читается байт из указанной ячейки. Таким образом, при работе со стеком используется правило «последним вошел – первым вышел» [4].

При вызове подпрограммы устройство управления микроконтроллера автоматически помещает в стек двухбайтовый адрес точки возврата. При этом содержимое **SP** автоматически увеличивается на 2. Микроконтроллер переходит к выполнению команд подпрограммы. Подпрограмма заканчивается командой `ret` (сокращение слова Return – возвращение). Исполняя эту команду, устройство управления микроконтроллера читает из стека адрес точки возврата и записывает его в счетчик команд **PC**. Микроконтроллер возвращается к выполнению основной программы, при этом содержимое **SP** уменьшается на 2.

При вызове подпрограммы и ее завершении работа со стеком выполняется автоматически без участия программиста. Однако имеется возможность прямой работы со стеком для записи и чтения данных. Для этих целей служат команды `push address` (`push` – толкать), которая записывает в стек содержимое указанной ячейки памяти с автоматическим увеличением указателя стека, и команда `pop address` (`pop` – закладывать), по которой происходит чтение байта из стека и запись по указанному адресу с автоматическим уменьшением указателя стека.

Работа подпрограммы обычно сопровождается модификацией данных в используемых регистрах. При этом важно не «испортить» данные, которые, возможно, используют другие участки программы. Обычно это содержимое аккумулятора, **PSW** и регистров общего назначения. Для этого часто используют закрепление различных банков **POH** за основной программой и подпрограммами, сохранение значений аккумулятора и **PSW** в стеке при вызове подпрограммы. В качестве примера листинг 5 содержит фрагмент кода программы, содержащий типовые команды вызова подпрограммы и подготовки обработки данных.

В приведенном фрагменте программы показан пример использования подпрограммы `Subr`. В начале основной программы в строке 3 задана область расположения стека. В указатель стека записывается число 120, т.е. под стек выделяется область оперативной памяти с 120 по 127 ячейки. Размер требуемой области стека можно подсчитать следующим образом. При вызове подпрограммы в стек будет записан двухбайтовый адрес точки возврата, что потребует 2 ячейки для его хранения. В подпрограмме мы сохраним в стеке содержимое аккумулятора и **PSW**, что потребует еще 2 ячейки. Таким образом, требуемый объем стека в данном примере составляет 4 ячейки. Следовательно, размер стека задан даже с

некоторым запасом.

Листинг 5 – Фрагмент кода программы, содержащий вызов подпрограммы

```
1      . . .
2 Main:      ; Задаем начальные значения переменных
3 mov SP,#120; В указатель пишем адрес вершины стека
4      . . .
5 ;Основной вычислительный цикл
6 Circl:
7      . . .
8 call Subr; Вызов подпрограммы Subr
9      . . .
10 jmp Circle; Зацикливание основной программы
11
12 ; Подпрограмма Subr
13 Subr: push ACC;Сохранение аккумулятора в стеке
14 push PSW; Сохранение PSW в стеке
15 setb RS1; Задание рабочего банка PОН
16 clr RS0 ;      задаем банк №2
17 ; Текст подпрограммы
18      . . .
19 ; Завершение подпрограммы
20 pop PSW; Восстановление PSW из стека
21 pop ACC; Восстановление аккумулятора из стека
22 ret; Выход из подпрограммы
23 End
```

В строках 6-10 условно показан основной цикл программы микроконтроллера. Он содержит вызов подпрограммы Subr в строке 8.

Непосредственно подпрограмма начинается со строки 13, которая содержит метку Subr. Строки 13 и 14 содержат команды сохранения содержимого аккумулятора и **PSW** в стеке. Далее в строках 15 и 16 задается рабочий банк регистров общего назначения (банк №2). На этом подготовительная часть подпрограммы завершается.

Далее в строках 17 – 19 размещаются рабочие команды подпрограммы. В этой части нельзя использовать команды условных и безусловных переходов на участки программы вне текста подпрограммы. Исключение составляют команды вызова других подпрограмм, но при этом следует учитывать, что размер области стека должен быть увеличен.

В строках 20 – 22 показана завершающая часть подпрограммы. Содержимое **PSW** и аккумулятора восстанавливаются из стека в обратном порядке командами

por. После восстановления данных команда `ret` завершает подпрограмму. По этой команде микроконтроллер вернется к исполнению команд основного цикла программы.

### ЗАДАНИЕ И ВАРИАНТЫ ВЫПОЛНЕНИЯ РАБОТЫ

Написать программу, реализующую циклическое изменение числа  $Z$ , хранящегося в ОЗУ, в соответствии с заданной формулой. Однобайтовые входные данные  $X$  для вычислений должны последовательно считываться из массива данных, хранящихся в памяти программ.

Считываемые из памяти программ данные используются для вычисления результата, если выполнено условие, указанное в задании. Критерий прекращения вычислений также задается заданием.

Вычисление по заданной формуле должно выполняться в подпрограмме. Подпрограмма должна вызываться в основном цикле, в котором выполняются считывание данных из памяти программ и проверка соответствия данных заданным условиям.

Рекомендуемые значения для массива данных: 209, 78, 203, 251, 146, 225, 170, 91, 15, 92, 58, 55, 217, 39, 162, 23, 112, 8, 227, 200, 17, 116, 200, 64, 105.

Варианты выполнения лабораторной работы представлены в таблице 5.

Таблица 5 – Варианты выполнения лабораторной работы

№	Банк РОН	Формула для вычисления	Условие использования байта данных $X$	Критерий прекращения вычислений
1	2	$Z = 2X + Z/2$	$X > 20$	Цикл выполнен 10 раз
2	1	$Z = X - 8Z$	$X > 40$	Цикл выполнен 12 раз
3	2	$Z = 3X + 100 + Z$	$X > 50$	$Z > 1024$
4	3	$Z = -X - 2Z$	$X < 0$	Цикл выполнен 7 раз
5	1	$Z = 2X + Z$	$X < 150$	$Z > 500$
6	1	$Z = 4X + Z$	$X$ - четное число	Цикл выполнен 5 раз
7	2	$Z = X - 2Z$	$X$ - четное число	Цикл выполнен 9 раз
8	3	$Z = 4X - 2Z$	$X$ - четное число	Цикл выполнен 7 раз
9	3	$Z = X/2 + Z$	$X$ - четное число	$Z > 800$
10	1	$Z = X/4 - Z$	$X$ - четное число	Цикл выполнен 10 раз
11	2	$Z = X/2 - 2Z$	$X$ - нечетное число	$Z < -1024$
12	3	$Z = X - Z/4$	$X$ - нечетное число	Цикл выполнен 11 раз

№	Банк РОН	Формула для вычисления	Условие использования байта данных X	Критерий прекращения вычислений
13	1	$Z = X - 4Z$	X - нечетное число	$Z < -512$
14	1	$Z = X \text{ or } Z$	X - нечетное число	Цикл выполнен 9 раз
15	2	$Z = X \text{ and } Z$	X - нечетное число	$Z=1$
16	3	$Z = X \text{ or } (Z/2)$	X - нечетное число	Цикл выполнен 6 раз
17	2	$Z = X \text{ and } (-Z)$	$X > 45$	Цикл выполнен 10 раз
18	1	$Z = (-X) \text{ or } Z$	$X < 0$	$Z = 127$
19	2	$Z = 4(X - Z)$	$X > 80$	Цикл выполнен 7 раз
20	3	$Z = (X + Z)/4$	$X < 200$	$Z > 1200$

### ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

Порядок выполнения работы и содержание отчета изложены в описании лабораторной работы №2.

### КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Чем отличаются команды `acall Subr` и `lcall Subr`?
2. Сколько байт нужно предусмотреть в стеке для сохранения данных регистра **PC**?
3. Почему в команде `push ACC` следует писать **ACC**, а не **A**?
4. Сколько байт займут в стеке данные регистра **PSW**?
5. Какие действия выполняет микроконтроллер по команде `ret`?

## ЛАБОРАТОРНАЯ РАБОТА №4 РАБОТА С ПОРТАМИ ВВОДА-ВЫВОДА

**ЦЕЛЬ РАБОТЫ:** освоить основные приемы работы с портами ввода-вывода двоичных данных.

## ОБЩИЕ СВЕДЕНИЯ

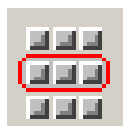
Связь с внешним миром микроконтроллер MCS51 осуществляет через четыре параллельных восьмибитовых порта ввода-вывода. Эти порты являются двунаправленными, т.е. могут как вводить информацию, так и выводить ее. Порты имеют имена **P0**, **P1**, **P2** и **P3**. При этом имеется возможность использовать отдельные линии портов для передачи информации в разных направлениях. При выполнении данной работы требуется знание основных приемов ввода-вывода информации через параллельные порты микроконтроллера [5].

В лабораторной работе к портам ввода-вывода подключаются определенные периферийные устройства, называемые элементами окружения микроконтроллера. Для задания окружения микроконтроллера и конкретизации схем подключения используются возможности редактора окружения симулятора **MCStudio** [1].

В лабораторной работе используются следующие элементы окружения микроконтроллера:



– Битовая кнопка. ”Подключается” к линии порта или разряду внешнего регистра. Может работать в режимах с фиксацией или без фиксации, режим задается в окне свойств кнопки.



– Линейка из 8 битовых кнопок. ”Подключается” к 8-разрядному порту микроконтроллера. Каждая из кнопок формирует отдельный сигнал, при этом информация от компонента может восприниматься побитно или как целый байт.



– Битовый индикатор (аналог светодиода). ”Подключается” к линии порта микроконтроллера. Цвет индикатора задается в окне его свойств.



– Линейка из 8 однобитовых индикаторов. ”Подключается” к 8-разрядному порту микроконтроллера. Каждый из индикаторов может управляться отдельным сигналом, или весь компонент – целым байтом. Цвета индикаторов задаются в окне его свойств.

Используя редактор окружения симулятора **MCStudio**, необходимо собрать одну из двух схем подключения периферийных устройств к микроконтроллеру (в зависимости от варианта лабораторной работы). Примеры «Лицевых панелей» виртуального прибора показаны на рисунках 9 и 10. При этом рекомендуется **Индикатор данных** подключить к порту P1, **Кнопки ввода данных** подключить

к порту P2, дополнительные «лампочки» Л1 и Л2, а также дополнительные Кнопки подключить к порту P3. Режим работы Кнопок ввода данных – с фиксацией нажатия, дополнительных Кнопок – без фиксации нажатия.

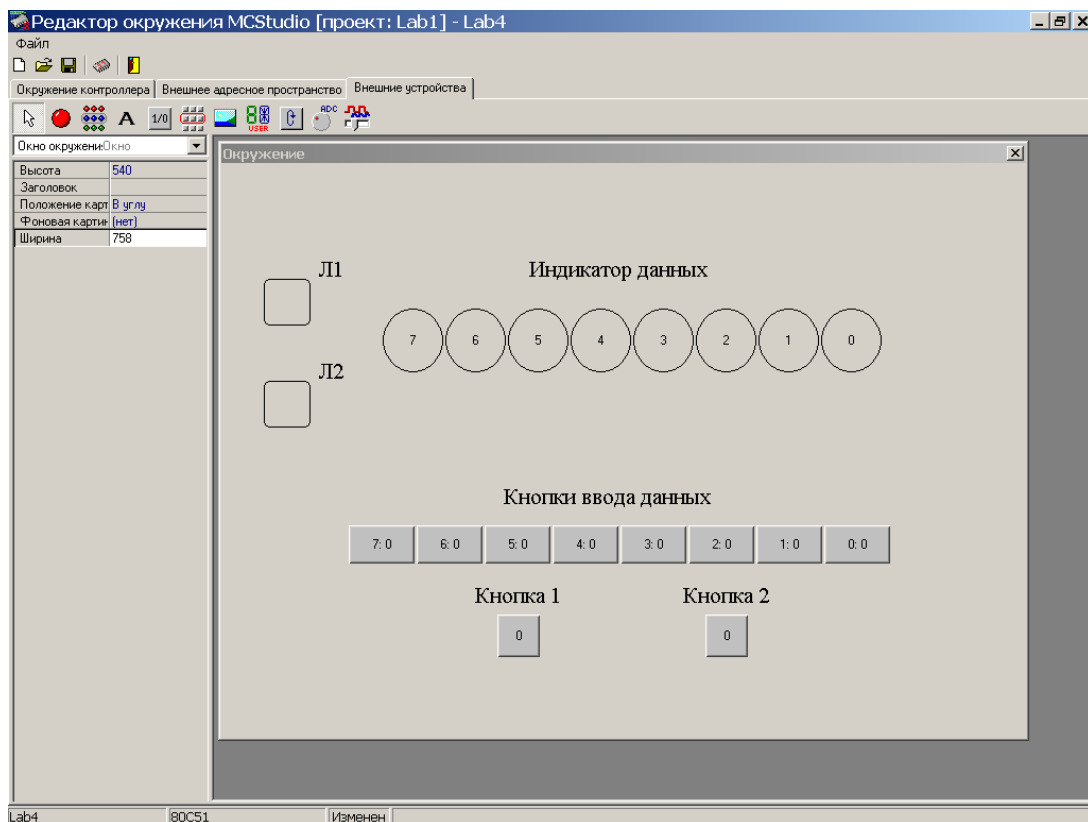


Рисунок 9 – Схема №1 внешнего вида «Лицевой панели» виртуального прибора в редакторе окружения **MCStudio**

Порядок работы с редактором окружения изложен в разделе «Моделирование работы микроконтроллера в программной среде MCStudio 1.7» настоящих методических указаний.

### ЗАДАНИЕ И ВАРИАНТЫ ВЫПОЛНЕНИЯ РАБОТЫ

Написать программу, реализующую циклическое изменение числа  $Y$ , хранящегося в ОЗУ, в соответствии с заданной формулой. Значение  $Y$  должно выводиться на линейку из 8 однобитовых индикаторов. Назначение дополнительных индикаторов Л1 и Л2 определяются вариантом задания.

Однобайтовые входные данные  $X$  для вычислений должны задаваться с помощью 8 внешних кнопок с фиксацией. Значение  $Y$  должно модифицироваться при нажатии на дополнительные кнопки. Режим работы дополнительных кнопок – без фиксации. При написании программы предусмотреть, чтобы реакция на нажатие дополнительных кнопок выполнялась при их отпускании. Сигналы от

кнопок считывать с соответствующих линий порта, к которым подключены кнопки.

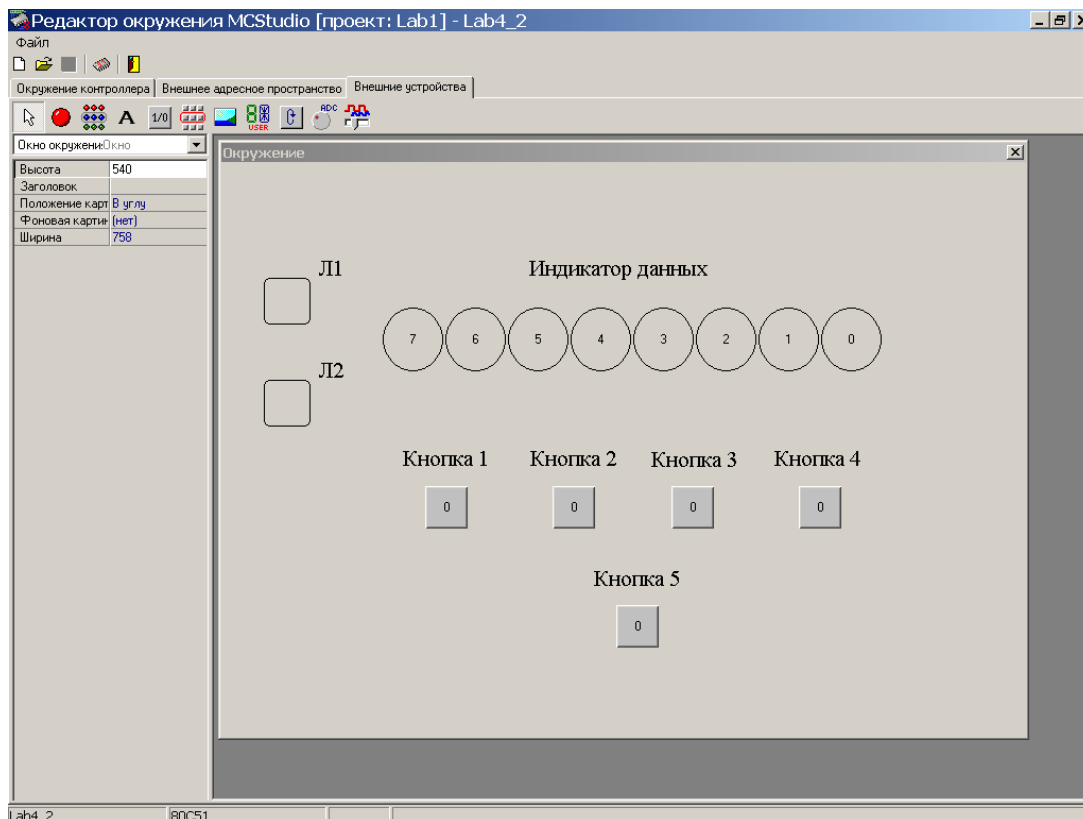


Рисунок 10 – Схема №2 внешнего вида «Лицевой панели» виртуального прибора в редакторе окружения **MCStudio**

Варианты выполнения лабораторной работы представлены в таблице 6. В варианте №20 данные должны храниться в массиве из 4 элементов – 1,2,3 и 4. Нажатием на **Кнопку 2** осуществляется переход к индикации и возможной модификации (вводу данных) следующего элемента массива, при этом номер элемента должен отображаться на индикаторах Л1 и Л2.

### ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. В соответствии с вариантом задания написать программу работы микроконтроллера на языке Ассемблер MCS51. Для этого следует запустить программу MCStudio и выбрать *Создать новый проект или файл -> Создать проект->Ok* с выбором личной папки и оригинальным именем проекта. В качестве модели микроконтроллера рекомендуется выбрать Intel 80C51 или Atmel AT89S8252.



Таблица 6 – Варианты выполнения лабораторной работы

№ вар.	Л1	Л2	дополнительные Кнопки				
			1	2	3	4	5
1	Перенос С	Доп. Перенос	Ввод X, $Y=Y/2+X$	Сброс в 0	-	-	-
2	Перенос С	Доп. Перенос	Ввод X, $Y=Y+X$	Сброс в 0	-	-	-
3	-	-	$Y=Y&X$	Сброс в 0	-	-	-
4	-	-	лог.ИЛИ	Сброс в 0	-	-	-
5	-	-	Искл. ИЛИ	Сброс в 0	-	-	-
6	-	Доп. Перенос	Ввод X	Сдвиг Y влево	Сдвиг Y вправо	Сброс в 0	-
7	Перенос С	Доп. Перенос	Ввод X	+10	-10	Сброс в 0	-
8	Перенос С	Доп. Перенос	Ввод X	+1	-1	Сброс в 0	-
9	Перенос	Доп. Перенос	+1	Инверсия Y	Ввод X	Сброс в 0	-
10	Перенос С	Доп. Перенос	+1	Изменить знак Y	Ввод X	Сброс в 0	-
11	-	-	Обмен тетрад Y	Ввод X	Сброс в 0	-	-
12	Перенос С	Доп. Перенос	$Y=2Y$	Ввод X	Сброс в 0	-	-
13	-	Доп. Перенос	$Y=0,5Y$	Ввод X	Сброс в 0	-	-
14	Перенос С	Доп. Перенос	Сдвиг Y влево через С	Сдвиг Y вправо через С	Ввод X	Сброс в 0	-
15	-	Доп. Перенос	Циклич. Сдвиг Y влево	Циклич. Сдвиг Y вправо	Ввод X	Сброс в 0	-

№ вар.	Л1	Л2	дополнительные Кнопки				
			1	2	3	4	5
16	Бит Y.7	-	Десятичн. Коррекц. Y	Изменить знак Y	Ввод X	Сброс в 0	-
17	Перенос C	Доп. Перенос	$Y=5Y$	Ввод X	Сброс в 0	-	-
18	Бит Y.7	Доп. Перенос	+10	Изменить знак Y	Ввод X	Сброс в 0	-
19	Перенос C	Доп. Перенос	Десятичн. Коррекц. Y	+1	-1	Ввод X	Сброс в 0
20	Ст. бит номера	Мл. бит номера	Ввод X	Просмотр данных	-	-	-

2. В открывшемся главном окне проекта набрать текст программы. Сохранить текст на диске компьютера.
3. Открыть редактор окружения **MCStudio** и в окне **Окружение контроллера** назначить направление передачи данных для используемых линий портов ввода-вывода.
4. В окне **Внешние устройства** создать панель управления в соответствии с заданным вариантом. В свойствах каждого элемента указать подключение выводов устройства к конкретным линиям портов микроконтроллера. Дать названия используемым устройствам в соответствии с выполняемыми функциями.
5. Нажать **Ctrl-F9** или выбрать *Проект ->Компилировать->Ok* и выполнить компиляцию программы (т.е. перевод текстового файла в двоичный код для микроконтроллера). Исправить обнаруженные компилятором синтаксические ошибки.
6. Проверить работу программы на симуляторе. Для этого запустить симуляцию, выбрав *Выполнение -> Запустить симуляцию*, и нажать **F9** (дважды). При запущенной симуляции на экране отображается окно **Выполнение программы** с круговой диаграммой хода машинного времени. Открыть окно **РПД – просмотр**, выбрав *Вид -> Резидентная память данных*. В ячейках резидентной памяти данных контролировать ввод данных и содержимое результата вычислений. Проведя переключения всех кнопок, проверить соответствие работы устройства поставленной задаче. Сохранить проект и продемонстрировать работу устройства преподавателю.
7. Оформить отчет о работе.

## СОДЕРЖАНИЕ ОТЧЕТА

Отчет должен содержать:

1. Титульный лист
2. Цель работы и конкретный вариант задания
3. Листинг программы работы микроконтроллера
4. Результаты работы программы в виде скриншотов соответствующих окон системы моделирования
5. Выводы по работе

## КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Чем отличается порт **P0** от порта **P1**?
2. Для чего при использовании линий порта на ввод данных в регистр порта записывают число 0FFh?
3. Как вывести 1 или 0 на отдельную линию порта?
4. Как проверить состояние (1 или 0) отдельной линии порта?
5. Как заставить микроконтроллер «реагировать» на нажатие (отпускание) кнопки?

## ЛАБОРАТОРНАЯ РАБОТА №5 ОТОБРАЖЕНИЕ ИНФОРМАЦИИ НА ИНДИКАТОРЕ

**ЦЕЛЬ РАБОТЫ:** освоить основные приемы табличного преобразования форматов представления чисел и приемы работы с семисегментными индикаторами.

**ОБЩИЕ СВЕДЕНИЯ.** При выполнении данной работы требуется знание основных команд ввода-вывода информации через параллельные порты микроконтроллера, а также способов представления и преобразования чисел в памяти контроллера [4].

Для задания окружения микроконтроллера и конкретизации схемы подключения периферийных устройств используются возможности редактора окружения симулятора **MCStudio** [1].

В лабораторной работе используются следующие элементы окружения микроконтроллера:



– Битовая кнопка. ”Подключается” к линии порта или разряду внешнего регистра. Может работать в режимах с фиксацией или без фиксации.



– Семисегментный индикатор с прямым управлением сегментами. ”Подключается” к линиям порта микроконтроллера.



– Имитируется поступление N-разрядного параллельного двоичного кода, который формируется аналого-цифровым преобразователем (АЦП). Входной аналоговый сигнал для АЦП задается моделью потенциометра. Для виртуального устройства задается разрядность АЦП и один или два порта для ”подключения” к микроконтроллеру.

Используя редактор окружения симулятора **MCStudio**, необходимо собрать схему подключения периферийных устройств к микроконтроллеру (в зависимости от варианта лабораторной работы). Пример “Лицевой панели” виртуального прибора показан на рисунке 11. При этом рекомендуется **Индикатор данных**, состоящий из трех семисегментных индикаторов, подключить к портам P0, P1 и P2, **Кнопки 1 и 2** подключить к порту P3, выходные линии АЦП также подключить к порту P3. Режим работы **Кнопки** – без фиксации нажатия.

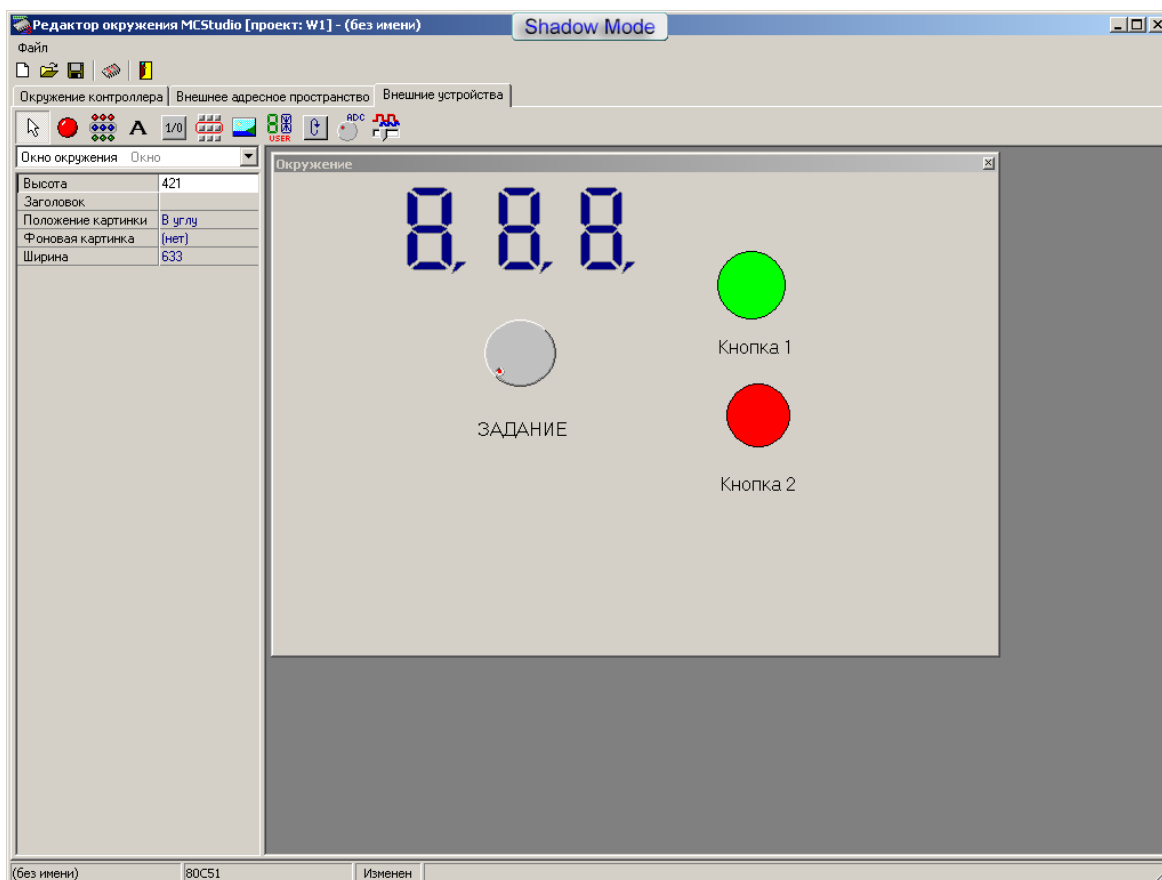


Рисунок 11 – Пример “Лицевой панели” виртуального прибора

Цифровые семисегментные индикаторы предназначены для отображения результата [5]. В лабораторной работе следует использовать цифровые индикаторы без встроенного дешифратора кода. Потенциометр с АЦП предназначен для ввода величины  $X$ , кнопки 1 и 2 – для управления работой устройства.

Для выполнения лабораторной работы необходимо написать программу работы микроконтроллера. Значение величины  $X$  задается поворотом движка «потенциометра», причем должна быть задана разрядность используемого АЦП. Результат вычислений по заданной формуле должен отображаться на **Индикаторах данных**. Отображение данных на индикаторах осуществляется либо в десятичном (10х), либо в шестнадцатиричном (16х) формате. Преобразование числа в код управления семисегментным индикатором должно выполняться табличным способом с хранением таблицы преобразования в памяти программ.

Ввод данных и модификация результата должны осуществляться в соответствии с командами, вводимыми посредством нажатия на **Кнопки**. Вычисление по заданной формуле рекомендуется выполнять в подпрограмме. Подпрограмма должна вызываться в основном цикле, в котором выполняется проверка состояний **Кнопок**. Сигналы от кнопок считывать с соответствующих линий порта, к которым подключены кнопки. Предусмотреть, чтобы реакция на нажатие кнопок выполнялась при их отпускании.

В таблице 7 приведены варианты выполнения лабораторной работы.

Таблица 7 – Варианты выполнения лабораторной работы

№ вар.	Индикация	АЦП, бит	Кнопка 1	Кнопка 2
1	10х	6	$Y = 2X + Y/2$	Ввод $X$
2	16х	4	$Y = X - 8Y$	Ввод $X$
3	10х	6	$Y = 3X + 100 + Y$	Ввод $X$
4	16х	6	$Y = -X - 2Y$	Ввод $X$
5	10х	6	$Y = 2X + Y$	Ввод $X$
6	16х	4	$Y = 4X + Y$	Ввод $X$
7	16х	6	Ввод $X$	$Y = X - 2Y$
8	10х	6	Ввод $X$	$Y = 4X - 2Y$
9	10х	6	Ввод $X$	$Y = X/2 + Y$
10	16х	4	Ввод $X$	$Y = X/4 - Y$
11	10х	6	Ввод $X$	$Y = X/2 - 2Y$
12	16х	4	Ввод $X$	$Y = X - Y/4$
13	10х	6	Ввод $X$	$Y = X - 4Y$

№ вар.	Индикация	АЦП, бит	Кнопка 1	Кнопка 2
14	16x	6	Ввод X	$Y = (X \text{ or } Y) + Y$
15	10x	6	$Y = (X \text{ and } Y) + Y$	Ввод X
16	16x	4	$Y = X \text{ and } (Y/2)$	Ввод X
17	16x	6	$Y = X \text{ and } (-Y)$	Ввод X
18	10x	6	$Y = (-X) \text{ or } Y$	Ввод X
19	10x	6	$Y = 4(X - Y)$	Ввод X
20	16x	4	$Y = (X + Y)/4$	Ввод X

### ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

Порядок выполнения работы и содержание отчета приведены в описании лабораторной работы №4.

### КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое двоично-десятичный код?
2. Зачем нужно преобразовывать число при выводе на индикатор?
3. Как написать код маски для выделения значения отдельного бита в байте?
4. Какие возможны способы проверки состояния (1 или 0) отдельной линии порта?
5. Что такое регистр-защелка?

## ЛАБОРАТОРНАЯ РАБОТА №6 ОБРАБОТКА ПРЕРЫВАНИЙ ОТ ВНЕШНЕГО СИГНАЛА

**ЦЕЛЬ РАБОТЫ:** освоить основные приемы настройки аппаратной части микроконтроллера и разработки подпрограмм обработки прерываний от внешних событий.

**ОБЩИЕ СВЕДЕНИЯ.** Выполнение данной работы требует знания работы устройства обработки прерываний от внешних событий [5].

Для задания окружения микроконтроллера и конкретизации схемы подключения периферийных устройств используются возможности редактора окружения симулятора **MCStudio** [1].

В лабораторной работе используются следующие элементы окружения микроконтроллера:



– Битовая кнопка. ”Подключается” к линии порта. Может работать в режимах с фиксацией или без фиксации.



– Семисегментный индикатор с прямым управлением сегментами. ”Подключается” к линиям порта микроконтроллера.



– Имитируется поступление N-разрядного параллельного двоичного кода, который формируется аналого-цифровым преобразователем (АЦП). Входной аналоговый сигнал для АЦП задается моделью потенциометра. Для виртуального устройства задается разрядность АЦП и один или два порта для ”подключения” к микроконтроллеру.

Лабораторная работа по своему заданию практически повторяет лабораторную работу №5. Отличие заключается в способе обработки сигналов кнопок управления. В данной лабораторной работе сигналы от кнопок управления должны вызывать прерывание программы. Обработчики соответствующих прерываний должны выполнять заданные в задании действия.

Используя редактор окружения симулятора **MCStudio**, необходимо собрать схему подключения периферийных устройств к микроконтроллеру (в зависимости от варианта лабораторной работы). Пример ”Лицевой панели” виртуального прибора показан на рисунке 11. При этом рекомендуется **Индикатор данных**, состоящий из трех цифровых индикаторов, подключить к портам P0, P1 и P2. Цифровые индикаторы предназначены для отображения результата. В лабораторной работе следует использовать цифровые индикаторы без встроенного дешифратора кода.

Потенциометр с АЦП предназначен для ввода величины X, кнопки 1 и 2 – для управления работой устройства. **Кнопки 1 и 2** подключить к порту P3, линии P3.2 и P3.3. Выходные линии АЦП подключить к свободным линиям порта P3. Режим работы **Кнопок** – без фиксации нажатия.

Линии порта P3, помимо выполнения обычных операций ввода-вывода, могут быть использованы для выполнения специальных (альтернативных) функций. Альтернативные функции активизируются только в том случае, если в соответствующие биты порта P3 предварительно записаны единицы. Линии P3.2 и P3.3 в режиме альтернативных функций способны выполнять функции вызова внешнего прерывания 0 (**INT0**) и внешнего прерывания 1 (**INT1**). Внешние прерывания **INT0** и **INT1** (сокращение слова interrupt – прерывать) могут вызываться либо уровнем, либо спадом сигнала на соответствующем входе P3.2 или P3.3. Настройка осуществляется заданием значения управляющих битов **IT0** и **IT1** в регистре специальных функций **TCON**. При возникновении внешнего прерывания в регистре **TCON** устанавливаются флаги **IE0** и **IE1**.

Разрешение прерываний программы выполняется программно путем установки соответствующих битов в регистре специальных функций **IE**. При включении питания все биты регистра **IE** сброшены в ноль и все прерывания запрещены. Поэтому после начала программы следует программно установить в единицу биты разрешения внешних прерываний **EX0** и **EX1**, а также бит снятия блокировки аппаратных прерываний **EA**. Кроме того, в регистре приоритетов прерываний **IP** можно указать приоритеты используемых прерываний, которые могут иметь обычный (0) или высший (1) приоритет. Внешним прерываниям **INT0** и **INT1** соответствуют биты приоритетов **PX0** и **PX1**.

Если прерывания программы разрешены и возникло внешнее прерывание, то микроконтроллер прерывает выполнение текущей части программы, запоминает в стеке адрес точки возврата и переходит к выполнению команды, записанной в векторе прерывания. Внешним прерываниям **INT0** и **INT1** соответствуют вектора **03h** и **13h**. Обычно по этим адресам записывают команды перехода (`jmp address`) к исполняемому коду обработки соответствующего прерывания. Завершается код обработки прерываний командой `reti` (сокращение слов `return from interrupt` – возврат из прерывания). По этой команде микроконтроллер извлекает из стека адрес точки возврата и переходит к продолжению выполнения прерванной программы.

Как отмечалось выше, от внешних прерываний **INT0** и **INT1** в регистре **TCON** устанавливаются флаги **IE0** и **IE1**, которые инициируют вызов соответствующей программы обработки прерывания. Сброс этих флагов выполняется автоматически только в том случае, если прерывания было вызвано спадом сигнала на соответствующей линии порта. Если прерывание вызвано уровнем входного сигнала, то сброс флага произойдет только после снятия запроса от источника прерывания.

Для выполнения лабораторной работы необходимо написать программу работы микроконтроллера. Значение величины **X** задается с помощью «потенциометра», причем должна быть задана разрядность используемого АЦП. Введенное значение **X** должно отображаться на **Индикаторе данных**. Результат вычислений по заданной формуле также должен отображаться на **Индикаторе данных**. Отображение данных на индикаторе осуществляется либо в десятичном (10x), либо в шестнадцатиричном (16x) формате. Преобразование числа в код управления цифровым индикатором должен выполняться табличным способом с хранением таблицы преобразования в памяти программ.

Ввод данных **X** и модификация результата **Y** должны осуществляться в соответствии с командами, вводимыми посредством нажатия на **Кнопки**. Управляющие кнопки должны быть «подключены» к линиям порта P3.2 и P3.3. Рекомендуется использовать настройку запуска процедуры прерывания по спаду входного сигнала. Нажатие (отпускание) кнопки должно вызывать прерывание программы и переход к соответствующей подпрограмме обработки прерывания.



Вычисление по заданной формуле рекомендуется выполнять в подпрограмме обработки прерывания. Результат вычислений рекомендуется сохранять в ячейках оперативной памяти. Желательно предусмотреть, чтобы реакция на нажатие кнопок выполнялась при их отпускании.

Листинг 6 содержит рекомендуемый шаблон программы работы микроконтроллера.

В начале основной программы в строках 1-4 задано расположение используемых переменных в памяти данных.

Непосредственно программа работы микроконтроллера начинается со строки 6. Директива `org 0` указывает, что располагающиеся далее команды должны последовательно располагаться в памяти программ, начиная с абсолютного адреса 0.

Первой является команда `jmp Main`, задающая безусловный переход на метку `Main`. Метке `Main` в этой программе соответствует ячейка памяти `30h`, что определяется директивой `org 30h` в строке 14.

Область памяти программ от ячейки 0 до ячейки `30h` является областью векторов прерываний и служит для задания команд перехода к обработчикам прерываний программы.

В строках 9 и 10 описано задание вектора прерывания **INT0**. В строке 10 в ячейку `03h` помещена команда `jmp Calc` перехода к метке `Calc` начала обработчика прерываний в строке 31. Аналогично в строках 11 и 12 описано задание вектора прерывания **INT1**. В строке 12 в ячейку `13h` помещена команда `jmp Enter` перехода к метке `Enter` в строке 35. Обработка прерываний заканчивается командами `reti` в строках 33 и 37.

В строках 21-23 программы записываются команды настройки аппаратных блоков микроконтроллера. В этой части программы необходимо задать начальные значения переменных, настроить режимы срабатывания внешних прерываний, приоритеты используемых прерываний, снять блокировку прерываний.

Следует задать вершину стека, записав в указатель стека `SP` адрес начальной ячейки памяти. В строке 21 в указатель стека записывается число 120, т.е. под стек выделяется область оперативной памяти с 120 по 127 ячейки. При вызове подпрограммы обработки прерываний в стек будет записан двухбайтовый адрес точки возврата, что потребует 2 ячейки для его хранения. В подпрограмме обычно сохраняют в стеке содержимое аккумулятора и **PSW**, что потребует еще 2 ячейки. Прерывания от управляющих кнопок будут чередоваться, и их одновременное срабатывание маловероятно. Таким образом, минимальный объем стека в данном примере составляет 4 ячейки памяти данных. Следовательно, размер стека задан даже с некоторым запасом.

В строках 25-29 условно показан основной цикл программы микроконтроллера. Он может содержать программный код, осуществляющий вывод на индикатор переменной `Rez`.

## Листинг 6 – Рекомендуемый шаблон программы

```
1 ; Адреса расположения переменных
2 X EQU 20h
3 RezL EQU 21h
4 RezH EQU 21h
5 ; Начало программы
6 org 0
7 jmp Main
8 ; Область векторов прерываний
9 org 03h
10 jmp Calc; Обработчик сигнала INT0
11 org 13h
12 jmp Enter; Обработчик сигнала INT1
13 ; Основная программа
14 org 30h
15 Main:
16 ; Задаем начальные значения переменных
17 ; и настройку в регистрах TCON, IP, IE,
18 ; а также указатель стека SP
19 mov RezH,#0
20 mov RezL,#0
21 mov SP,#120
22 . . .
23 mov IE,#85h; Разрешение прерываний
24 ;Вычислительный цикл
25 Circl:
26 ; Выполняем декодирование Rez и вывод результата
27 ; на индикатор
28 . . .
29 jmp Circle; Зацикливаем
30 ;Обработчик прерывания INT0
31 Calc:
32 . . . ; Вычисление по формуле, записываем в Rez
33 reti ; Завершение обработки прерывания
34 ;Обработчик прерывания INT1
35 Enter:
36 . . . ; Ввод данных и сохранение в X
37 reti ; Завершение обработки прерывания
38 ; Таблица кодов индикатора
39 TAB: db dd1,dd2,dd3,. . .
40 End
```

В строках 31-33 и 35-37 пишутся программные коды обработчиков прерываний. Эти программные коды в соответствии с заданием модифицируют переменную Rez.

В строке 39 показана завершающая часть подпрограммы. Здесь записана таблица кодировки символов цифрового индикатора.

В таблице 7 приведены варианты выполнения лабораторной работы.

### ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

Порядок выполнения работы и содержание отчета приведены в описании лабораторной работы №4.

### КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое внешнее прерывание?
2. Что обозначает термин «вектор прерывания»?
3. Как осуществляется переход к выполнению подпрограммы обработки прерывания?
4. В чем отличие команд `ret` и `reti`?
5. Когда сбрасываются флаги прерываний от внешних событий?

## ЛАБОРАТОРНАЯ РАБОТА №7 РАБОТА С ТАЙМЕРОМ

**ЦЕЛЬ РАБОТЫ:** освоить основные приемы использования таймера для реализации требуемого интервала квантования времени и формирования импульсов заданной длительности.

**ОБЩИЕ СВЕДЕНИЯ.** При выполнении данной работы требуется знание режимов работы таймеров-счетчиков, регистров специальных функций, а также организации подпрограмм обработки прерываний [5,6].

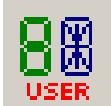
Для задания окружения микроконтроллера и конкретизации схемы подключения периферийных устройств используются возможности редактора окружения симулятора **MCStudio** [1]. В лабораторной работе, в зависимости от варианта выполнения используются следующие элементы окружения микроконтроллера:



– Битовая кнопка. ”Подключается” к линии порта или разряду внешнего регистра. Может работать в режимах с фиксацией или без фиксации.



– Линейка из 8 битовых индикаторов. ”Подключается” к 8-разрядному порту микроконтроллера или ко внешнему регистру. Каждый из индикаторов может управляться отдельным сигналом, или весь компонент – целым байтом.



– Семисегментный индикатор с прямым управлением сегментами. ”Подключается” к линиям порта микроконтроллера.



– Имитирует поступление N-разрядного параллельного двоичного кода, который формируется аналого-цифровым преобразователем (АЦП). Входной аналоговый сигнал для АЦП задается моделью потенциометра. Для виртуального устройства задается разрядность АЦП и порт для ”подключения” АЦП к микроконтроллеру.

Используя редактор окружения симулятора **MCStudio**, необходимо собрать схему подключения периферийных устройств к микроконтроллеру. Рекомендуется **Индикатор данных** подключить к портам P0, P1 и P2, **Кнопки 1 и 2** подключить к свободным линиям портов, выходные линии АЦП подключить к порту P3. Режим работы **Кнопка** – без фиксации нажатия. При написании программы предусмотреть, чтобы реакция на нажатие **Кнопка** выполнялась при их отпускании.

Для выполнения лабораторной работы необходимо написать программу работы микроконтроллера, по которой входные данные задаются при помощи потенциометра и Кнопок ввода данных. Значение величины периода (или частоты) задается «поворотом» движка потенциометра, причем должна быть определена разрядность используемого АЦП. Результат должен отображаться на **Индикаторах**. Ввод данных и модификация должны осуществляться в соответствии с командами, вводимыми посредством нажатия на **Кнопки**. На рисунке 12 приведен пример «лицевой панели» виртуального прибора.

Отсчет интервала квантования времени (такта работы устройства) необходимо выполнить на таймере, запрограммировав его на отсчет заданного временного интервала и обрабатывая в подпрограмме прерывания от таймера. В «настроечной» части программы следует задать режим работы таймера. Для большинства заданий рекомендуется режим работы 1 – шестнадцатитрибитовый таймер. Биты режима выбранного таймера в регистре **TMOD** следует задать таким образом, чтобы таймер подсчитывал импульсы от генератора тактовых импульсов микроконтроллера. Для облегчения расчетов можно установить частоту работы микроконтроллера 12 МГц. Для это во вкладке *Выполнение->Опции симуляции* в

окне *Частота контроллера (в Гц)* следует установить желаемую частоту (см. рисунок 13).

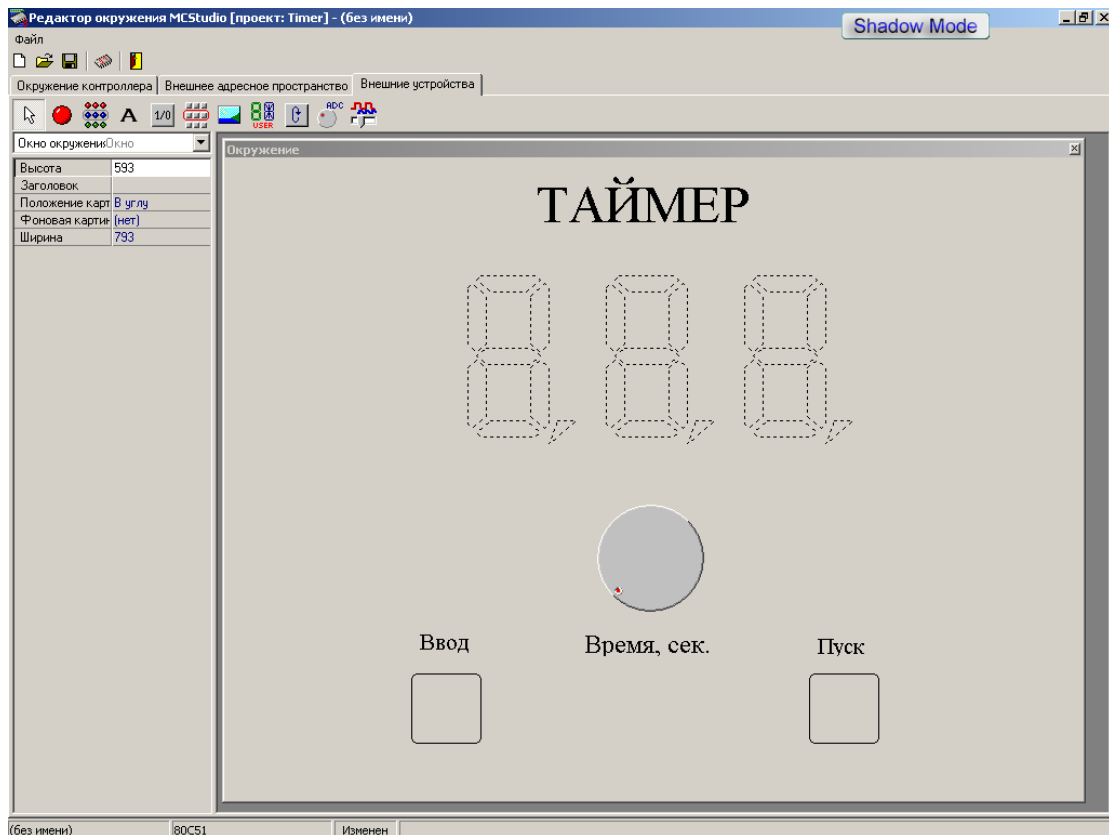


Рисунок 12 – “Лицевая панель” виртуального прибора

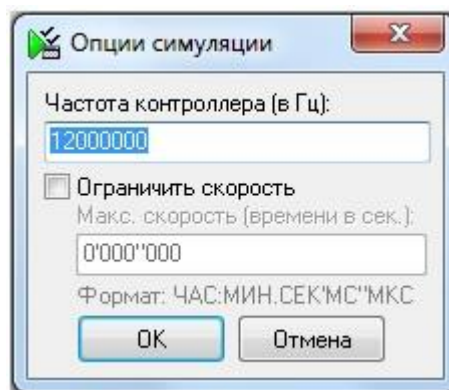


Рисунок 13 – Окно установки тактовой частоты работы микроконтроллера

В регистре **IP** следует разрешить прерывания от таймера, а переход к подпрограмме обработки прерываний следует осуществлять с использованием

вектора прерывания – 0Vh для таймера T/C0 и 1Vh для таймера T/C1. Для пуска работы устройства рекомендуется разрешать работу таймера (бит TRx=1), а при останове – запрещать (бит TRx=0).

В таблице 8 приведены варианты заданий для выполнения лабораторной работы.

Таблица 8 – Варианты выполнения лабораторной работы

№	Задание
1	Реализовать устройство – таймер (обратный отсчет времени). Интервал времени 0-9.9 с., разрешение 0.1 с. Индикация – цифровые индикаторы; задание времени выполнить посредством АЦП; кнопки управления – ввод времени (1), пуск отсчета времени (2)
2	Реализовать секундомер. Интервал времени 0-99.9 с., разрешение 0.1 с. Индикация – цифровые индикаторы; кнопки управления – сброс показания (1), пуск/стоп отсчета времени (2)
3	На линейке из 8 светодиодов отображать в двоичном коде содержимое ячейки ОЗУ. Число в ячейке по тактам наращивать на единицу, т.е. сделать счетчик тактов. Такт работы задавать посредством АЦП от 0 до 9.5 с. с разрешением 0.5 с. Заданный период тактирования отображать на цифровом индикаторе. Кнопки управления: Пуск/Стоп (1) и Ввод периода (2).
4	Управление линейкой светодиодов. На линейке из 16 светодиодов отображать число из ячеек ОЗУ, которое модифицировать периодически по алгоритму двоичного счетчика Джонсона. Период изменения числа задать посредством АЦП от 0 до 5.0 с. с разрешением 0.5 с. Заданный период отображать на цифровом индикаторе. Кнопки управления: Пуск/Стоп (1) и Сброс (2).
5	Управление временем свечения красного светодиода. Частота включения светодиода постоянная, равная 1 Гц, длительность свечения светодиода задавать с помощью АЦП и отображать на цифровом индикаторе. Диапазон регулирования длительности от 0 до 0.8 с, дискретность 0.1 с. Кнопки управления: Пуск/Стоп (1) и Ввод периода (2).
6	«Бегущие огни». На линейке из 8 светодиодов отображать перемещающуюся пару включенных светодиодов – красный и зеленый. Такт работы задавать посредством АЦП от 0.2 до 1.5 с. с разрешением 0.1 с. Заданный период тактирования отображать на цифровом индикаторе. Кнопки управления: Пуск/Стоп (1) и Ввод периода (2).

№	Задание
7	Управление линейкой светодиодов. На линейке из 16 светодиодов отображать число из ячеек ОЗУ, которое модифицировать периодически по алгоритму умножения начального числа (единицы) на 2. Период изменения числа задать посредством АЦП от 0 до 2.0 с. с разрешением 0.2 с. Заданный период отображать на цифровом индикаторе. Кнопки управления: Пуск/Стоп (1) и Начальное состояние (2).
8	Управление цветом свечения светодиода. Наложить изображения двух светодиодов (синий и зеленый) на одно поле. Частота включения синего светодиода постоянная, равная 20 Гц, длительность свечения задавать с помощью АЦП и отображать на цифровом индикаторе. Длительность свечения зеленого светодиода – оставшаяся часть периода. Диапазон регулирования длительности от 0 до 50 мс, дискретность 1 мс.
9	Управление светофором. Предусмотреть задание длительности свечения одного состояния с помощью кнопок: 3 сек. (1), 1 сек. (2) и включение режима «мигающий желтый» (кнопка 3) с периодом 2 с.
10	На цифровом индикаторе отображать перемещающийся сегмент. Такт работы задавать посредством АЦП от 0.1 до 1.0 с. с разрешением 0.1 с. Заданный период тактирования отображать на отдельном цифровом индикаторе. Кнопки управления: Пуск/Стоп (1) и Ввод периода (2).
11	Реализовать устройство – таймер (обратный отсчет времени). Интервал времени 0-99 с., разрешение 1 с. Индикация – цифровые индикаторы; задание времени выполнить посредством АЦП; кнопки управления – ввод времени (1), пуск отсчета времени (2)
12	Реализовать секундомер. Интервал времени 0-9.9 с., разрешение 0.01 с. Индикация – цифровые индикаторы; кнопки управления – сброс показания (1), пуск/стоп отсчета времени (2)
13	На линейке из 8 светодиодов отображать в двоичном коде содержимое ячейки ОЗУ. Число в ячейке по тактам наращивать на единицу, т.е. сделать счетчик тактов. Такт работы задавать посредством АЦП от 0 до 1 с. с разрешением 0.1 с. Заданный период тактирования отображать на цифровом индикаторе. Кнопки управления: Пуск/Стоп (1) и Ввод периода (2).
14	Реализовать секундомер. Интервал времени 0-50.0 с., разрешение 0.1 с. Индикация – цифровые индикаторы; кнопки управления – сброс показания (1), пуск/стоп отсчета времени (2)

15	Реализовать управление временем свечения синего «светодиода». Частота включения светодиода постоянная, равная 0.5 Гц, длительность свечения светодиода задавать с помощью АЦП и отображать на цифровом индикаторе. Диапазон регулирования длительности от 0 до 2 с, дискретность 0.1 с. Кнопки управления: Пуск/Стоп (1) и Ввод периода (2).
16	«Бегущие огни». На линейке из 8 светодиодов отображать перемещающуюся пару включенных светодиодов – зеленый и синий. Период работы задавать посредством АЦП от 0.1 до 2.0 с. с разрешением 0.1 с. Заданный период тактирования отображать на цифровом индикаторе. Кнопки управления: Пуск/Стоп (1) и Ввод периода (2).
17	Управление линейкой светодиодов. На линейке из 16 светодиодов отображать число из ячеек ОЗУ, которое модифицировать периодически по алгоритму умножения начального числа (тройка) на 4. Период изменения числа задать посредством АЦП от 0 до 1.5 с. с разрешением 0.1 с. Заданный период отображать на цифровом индикаторе. Кнопки управления: Пуск/Стоп (1) и Начальное состояние (2).
18	Реализовать устройство – таймер (обратный отсчет времени). Интервал времени 0-50 с., разрешение 0.5 с. Индикация – цифровые индикаторы; задание времени выполнить посредством АЦП; кнопки управления – ввод времени (1), пуск отсчета времени (2)
19	Управление цветом свечения светодиода. Частота включения красного светодиода постоянная, равная 10 Гц, длительность свечения задавать с помощью АЦП и отображать на цифровом индикаторе. Длительность свечения зеленого светодиода – оставшая часть периода. Диапазон регулирования длительности от 10 до 100 мс., дискретность 5 мс. Кнопки управления: Пуск/Стоп (1) и Ввод периода (2).
20	Управление линейкой светодиодов. На линейке из 16 светодиодов отображать число из ячеек ОЗУ, которое модифицировать периодически по алгоритму двоичного счетчика Джонсона. Период изменения числа задать посредством АЦП от 0 до 1.0 с. с разрешением 0.1 с. Заданный период отображать на цифровом индикаторе. Кнопки управления: Пуск/Стоп (1) и Сброс (2).

### ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

Порядок выполнения работы и содержание отчета приведены в описании лабораторной работы №4.

### КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Чем отличается режим работы «таймер» от режима работы «счетчик»?



2. Как рассчитать период переполнения таймера?
3. Как задать нужный период переполнения таймера?
4. Как обнаружить переполнение таймера?
5. Как сбрасываются флаги запросов прерывания от таймеров?

## ЛАБОРАТОРНАЯ РАБОТА №8 ИЗМЕРЕНИЕ ИНТЕРВАЛОВ ВРЕМЕНИ

**ЦЕЛЬ РАБОТЫ:** освоить основные приемы использования таймера для измерения величины периода или частоты внешнего сигнала.

**ОБЩИЕ СВЕДЕНИЯ.** При выполнении данной работы требуется знание режимов работы таймеров–счетчиков, регистров специальных функций, а также организации подпрограмм обработки прерываний.

Для задания окружения микроконтроллера и конкретизации схемы подключения периферийных устройств используются возможности редактора окружения симулятора **MCStudio**. В лабораторной работе в зависимости от варианта выполнения используются следующие элементы окружения микроконтроллера:



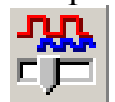
– Битовая кнопка. ”Подключается” к линии порта или разряду внешнего регистра. Может работать в режимах с фиксацией или без фиксации.



– Семисегментный индикатор с прямым управлением сегментами. ”Подключается” к линиям порта микроконтроллера.



– Имитирует поступление N-разрядного параллельного двоичного кода, который формируется аналого-цифровым преобразователем (АЦП). Входной аналоговый сигнал для АЦП задается моделью потенциометра. Для виртуального устройства задается разрядность АЦП и порт для ”подключения” АЦП к микроконтроллеру.



– Виртуальное устройство имитирует работу генератора прямоугольных импульсов (ГПИ) с регулируемой частотой и скважностью импульсов. Выход генератора может быть подключен к любому входу микроконтроллера.

Используя редактор окружения симулятора **MCStudio**, необходимо собрать схему подключения периферийных устройств к микроконтроллеру. Рекомендуется **Индикатор данных** подключить к портам P0, P1 и P2, **Кнопки 1 и 2** подключить к свободным линиям портов, выходные линии ГПИ или АЦП подключить к порту P3. Режим работы **Кнопка** – без фиксации нажатия.

На рисунке 14 приведен пример «лицевой панели» виртуального прибора.

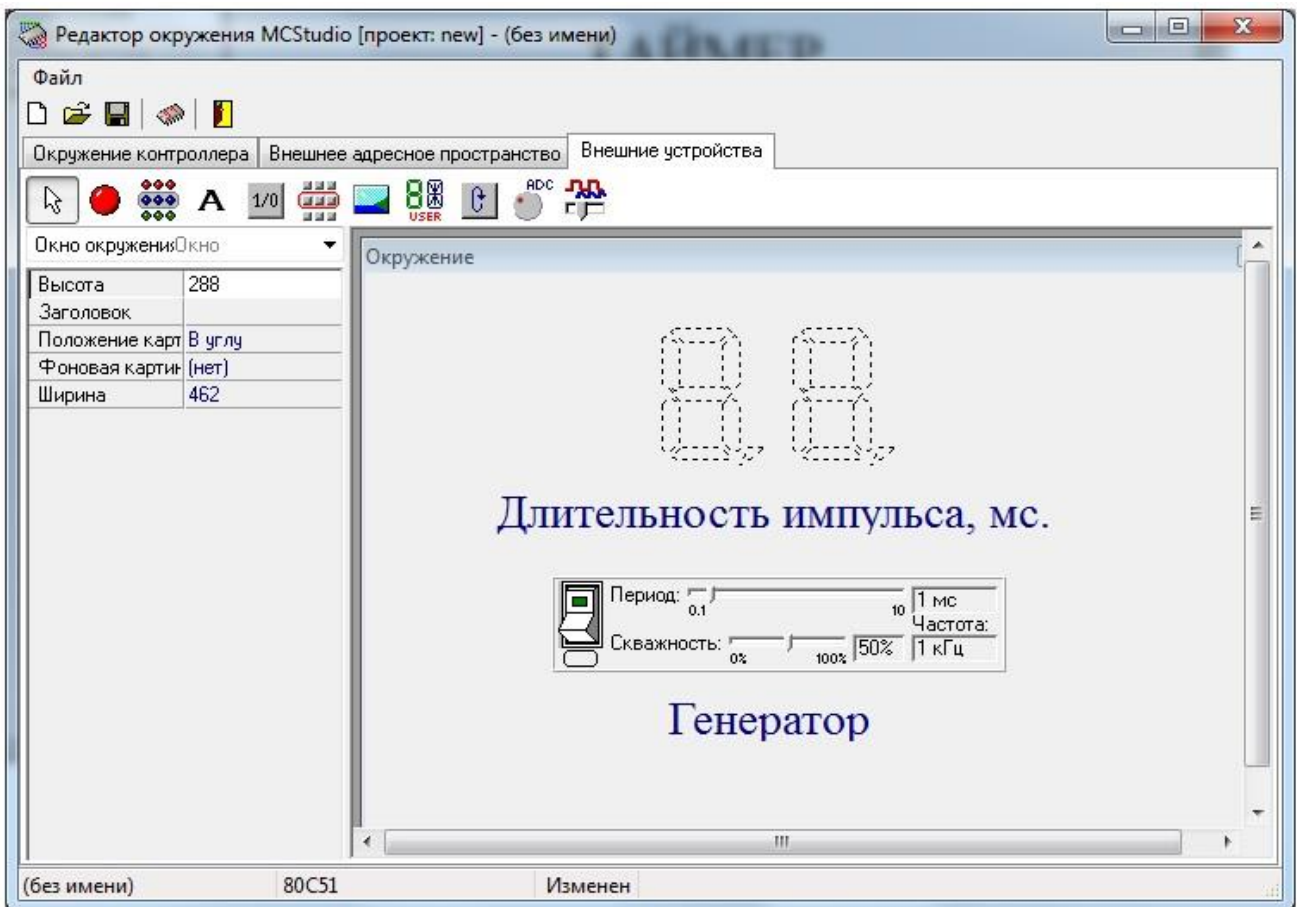


Рисунок 14 – Пример “Лицевой панели” виртуального прибора

Необходимо написать программу работы микроконтроллера, по которой виртуальный прибор реализует задание. Значение величины периода (или частоты) задается внешним ГПИ или другим заданным устройством. Результат должен отображаться на **Индикаторах данных**. Модификация данных должна выполняться со скоростью, удобной для восприятия человеком.

При написании программы необходимо использовать таймеры, запрограммировав их на отсчет заданного временного интервала и обрабатывая в подпрограмме прерывания от таймера. В «настройной» части программы следует задать режим работы таймера. Для большинства заданий рекомендуется режим работы 1 – шестнадцатитрибитовый таймер/счетчик. Биты режима выбранного

таймера устанавливаются в регистре **TMOD** соответствующим образом. Для управления работой таймера используют также биты регистра **TCON**. Для облегчения расчетов можно установить частоту работы микроконтроллера 12 МГц. Для это во вкладке *Выполнение->Опции симуляции* в окне *Частота контроллера (в Гц)* следует установить желаемую частоту (см. рисунок 13).

В регистре **IP** при необходимости следует разрешить прерывания от таймера или внешнего устройства, а переход к подпрограмме обработки прерываний следует осуществлять с использованием векторов прерывания – 0Bh (1Bh) для таймера **T/C0 (T/C1)** и 03h (13h) для внешнего устройства **INT0 (INT1)**.

В таблице 9 приведены варианты заданий для выполнения лабораторной работы.

Таблица 9 – Варианты выполнения лабораторной работы

№	Задание
1	Реализовать измеритель периода сигнала внешнего ГПИ. Результат отобразить на цифровом индикаторе. Разрешение 1 мс., интервал измерения 1-99 мс. Выход генератора подключить к линии порта Int1 и использовать этот сигнал для прерывания основной программы. Период обновления информации на индикаторе 0.5 с.
2	Реализовать измеритель интервала времени между двумя нажатиями на кнопку Пуск/Стоп (1). Результат отобразить на цифровом индикаторе. Разрешение 0.01 с., интервал измерения 1-9.99 с. Для обнуления показаний использовать кнопку Сброс (2).
3	Реализовать измерение частоты периода сигнала внешнего ГПИ. Диапазон измерения 1-99 Гц. Результат отобразить на цифровом индикаторе. Период обновления результата на индикаторе - 1 с.
4	Реализовать измерение длительности импульса на выходе внешнего ГПИ. Диапазон измерения 1-50 мс. Разрешение 1 мс. Частота следования импульсов 20 Гц. Результат отобразить на цифровом индикаторе. Выход генератора подключить к линии порта Int1 и использовать этот сигнал для управления блокировкой таймера/счетчика.
5	Реализовать измеритель интервала времени между двумя нажатиями на кнопку Пуск/Стоп (1). Результат отобразить на цифровом индикаторе. Разрешение 0.01 с., интервал измерения 0-1.0 с. Для обнуления показаний использовать кнопку Сброс (2).
6	Реализовать измеритель периода сигнала внешнего ГПИ. Результат отобразить на цифровом индикаторе. Разрешение 10 мс., интервал измерения 10-500 мс. Выход генератора подключить к линии порта Int0 и использовать этот сигнал для прерывания основной программы.

№	Задание
7	Реализовать измеритель времени реакции оператора на периодически включаемый сигнал светодиода. Период включения светодиода 5 с. Диапазон измерения времени реакции 0.1 – 2.5 с. Разрешение измерения времени реакции 0.1 с. Кнопку ответа оператора подключить к линии порта Int0 и использовать этот сигнал для прерывания основной программы.
8	Реализовать измерение длительности импульса на выходе внешнего ГПИ. Диапазон измерения 1-100 мс. Разрешение 1 мс. Частота следования импульсов 10 Гц. Результат отобразить на цифровом индикаторе. Выход генератора подключить к линии порта Int0 и использовать этот сигнал для управления блокировкой таймера/счетчика.
9	Реализовать измерение частоты периода сигнала внешнего ГПИ. Диапазон измерения 10-999 Гц. Результат отобразить на цифровом индикаторе. Период обновления результата на индикаторе – 0.5 с.
10	Реализовать измеритель периода сигнала внешнего ГПИ. Результат отобразить на цифровом индикаторе. Интервал измерения 0.1-10 с., разрешение 100 мс. Выход генератора подключить к линии порта Int0 и использовать этот сигнал для прерывания основной программы. В качестве генератора опорной частоты использовать второй внешний ГПИ.
11	Реализовать измеритель времени реакции оператора на периодически включаемый сигнал светодиода. Период включения светодиода 5 с. Диапазон измерения времени реакции 0.01 – 0.99 с. Разрешение измерения времени реакции 0.01 с. Кнопку ответа оператора подключить к линии порта Int1 и использовать этот сигнал для прерывания основной программы.
12	Реализовать измеритель периода сигнала внешнего ГПИ. Результат отобразить на цифровом индикаторе. Разрешение 10 мс., интервал измерения 20-999 мс. Выход генератора подключить к линии порта Int1 и использовать этот сигнал для прерывания основной программы.
13	Реализовать измеритель периода сигнала внешнего ГПИ. Результат отобразить на цифровом индикаторе. Интервал измерения 0.01-2.5 с., разрешение 10 мс. Выход генератора подключить к линии порта Int0 и использовать этот сигнал для прерывания основной программы. В качестве генератора опорной частоты использовать второй внешний битовый генератор.

№	Задание
14	Реализовать измерение периода следования импульсов битового сигнала на выходе младшего разряда виртуального потенциометра (или АЦП). Измеренные значения сохранять циклически в восьми ячейках оперативной памяти. Результат измерения выводить из последовательно памяти на цифровой индикатор при нажатии на кнопку «Просмотр данных». Диапазон измерения периода 0.02 - 2.5 с., разрешение 0.02 с.
15	Реализовать измерение частоты периода сигнала внешнего ГПИ. Диапазон измерения 100-9990 Гц. Результат отобразить на цифровом индикаторе. Период обновления результата на индикаторе – 0.5 с.
16	Реализовать измерение периода следования импульсов битового сигнала на выходе младшего разряда виртуального потенциометра (или АЦП). Измеренные значения сохранять циклически в восьми ячейках оперативной памяти. Результат измерения выводить из последовательно памяти на цифровой индикатор при нажатии на кнопку «Просмотр данных». Диапазон измерения периода 0.01 - 0.99 с., разрешение 0.01 с.
17	Реализовать измерение длительности импульса на выходе ГПИ. Диапазон измерения 0.1-20 мс. Разрешение 0.1 мс. Частота следования импульсов 50 Гц. Результат отобразить на цифровом индикаторе. Выход генератора подключить к линии порта Int0 и использовать этот сигнал для управления блокировкой таймера/счетчика.
18	Реализовать измеритель интервала времени между двумя нажатиями на кнопку Пуск/Стоп (1). Результат отобразить на цифровом индикаторе. Разрешение 0.1 с., интервал измерения 0.1-5.0 с. Для обнуления показаний использовать кнопку Сброс (2).
19	Реализовать измеритель времени реакции оператора на периодически включающийся сигнал светодиода. Период включения светодиода 5 с. Диапазон измерения времени реакции 0.1 – 1.5 с. Разрешение измерения времени реакции 0.1 с. Кнопку ответа оператора подключить к линии порта Int1 и использовать этот сигнал для прерывания основной программы.
20	Реализовать измерение периода следования импульсов битового сигнала на выходе младшего разряда виртуального потенциометра (или АЦП). Измеренные значения сохранять циклически в восьми ячейках оперативной памяти. Результат измерения выводить из последовательно памяти на цифровой индикатор при нажатии на кнопку «Просмотр данных». Диапазон измерения периода 0.1 -9.9 с., разрешение 0.1 с.

## ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

Порядок выполнения работы и содержание отчета приведены в описании

лабораторной работы №4.

### КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Для чего нужен генератор опорной частоты?
2. Какие устройства могут быть использованы в качестве генератора опорной частоты?
3. Как задать требуемую тактовую частоту с помощью таймера?
4. Как вычислить разрешающую способность цифрового измерительного прибора?
5. Как вычислить коэффициент передачи цифрового измерительного прибора?

### СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Джулгаков Д.В. MCStudio. Интегрированная среда разработки программного обеспечения однокристальных микроконтроллеров MCS-51/Руководство пользователя. – Харьков: 2006.
2. Злобин В.К., Григорьев В.Л. Программирование арифметических операций в микропроцессорах: Учеб. пособие для технических вузов.-М.: Высш. шк., 1991.-303 с.: ил. ISBN 5-06-00205-5
3. Каспер Э. Программирование на языке ассемблера для микроконтроллеров семейства i8051 / Э. Каспер. Москва : Горячая линия-Телеком, 2003. 192 с.
4. Микушин А.В. Занимательно о микроконтроллерах. – СПб.: БХВ-Петербург, 2006.- 432 с.: ил. ISBN 5-94157-571-8
5. Магда Ю.С. Микроконтроллеры серии 8051: практический подход. – М.: ДМК Пресс. 2008.- 228 с.: ил. ISBN 5-94074-394-3
6. Моисейкин Е.В. Микроконтроллеры семейства MCS-51. Теория и практика: учебно– методическое пособие – Екатеринбург : Изд-во Урал. ун-та, 2017. – 144с. ISBN 978-5-7996-2167-4

## СИСТЕМА КОМАНД МИКРОКОНТРОЛЛЕРОВ СЕМЕЙСТВА MCS51

Система команд MCS-51 содержит 111 базовых инструкций, которые предназначены для выполнения команд пересылки данных, выполнения арифметических и логических операций, а также операций управления ходом вычислений [6]. Формат команд одно-, двух- и трехбайтовый, причем первый байт всегда содержит код операции.

При описании команд используются следующие обозначения:

Rn (n = 0, 1, ..., 7) – регистр общего назначения в выбранном банке регистров;  
@Ri (i= 0, 1) – регистр общего назначения в выбранном банке регистров, используемый в качестве регистра косвенного адреса;

ad – адрес прямо адресуемого байта;

ads – адрес прямо адресуемого байта-источника;

add – адрес прямо адресуемого байта-получателя;

ad11 – 11-разрядный абсолютный адрес перехода;

ad16 – 16-разрядный абсолютный адрес перехода;

rel – относительный адрес перехода;

#d – непосредственный операнд;

#d16 – непосредственный операнд (2 байта);

bit – адрес прямо адресуемого бита

/bit – инверсия прямо адресуемого бита;

A – символическое обозначение аккумулятора;

ACC – аккумулятор (адрес регистра);

PC – счетчик команд;

SP – регистр указатель стека;

DPTR – регистр указатель данных;

( ) – содержимое ячейки памяти или регистра.

### КОМАНДЫ ПЕРЕСЫЛКИ ДАННЫХ

Эта группа содержит 28 команд. Краткое описание команд представлено в таблице П1. Кроме мнемоники, указано количество байт, занимаемое командой в памяти программ.

Таблица П1 – Команды пересылки данных

Мнемокод	Операция	Кол-во байт	Описание
mov A,Rn	$(A) \leftarrow (Rn)$	1	Пересылка байта данных из регистра Rn ( $n=0 \div 7$ ) в аккумулятор
mov A,ad	$(A) \leftarrow (ad)$	2	Пересылка байта данных из ячейки ad в аккумулятор
mov A,@Ri	$(A) \leftarrow ((Ri))$	1	Пересылка байта данных из ячейки ПД ( $i=0,1$ ) в аккумулятор
mov A,#d	$(A) \leftarrow \#d$	2	Пересылка константы в аккумулятор
mov Rn,A	$(Rn) \leftarrow (A)$	1	Пересылка байта данных из аккумулятора в регистр Rn ( $n=0 \div 7$ )
mov Rn,ad	$(Rn) \leftarrow (ad)$	2	Пересылка байта данных из ячейки ad в регистр Rn ( $n=0 \div 7$ )
mov Rn,#d	$(Rn) \leftarrow \#d$	2	Пересылка константы в Rn ( $n=0 \div 7$ )
mov ad,A	$(ad) \leftarrow (A)$	2	Пересылка байта данных из аккумулятора в ячейку ad
mov ad,Rn	$(ad) \leftarrow (Rn)$	2	Пересылка байта данных из регистра Rn ( $n=0 \div 7$ ) в ячейку ad
mov add,ads	$(add) \leftarrow (ads)$	3	Пересылка байта данных из ячейки ads в ячейку add
mov ad,@Ri	$(ad) \leftarrow ((Ri))$	2	Пересылка бата данных из ячейки ПД ( $i=0,1$ ) в ячейку ad
mov ad,#d	$(ad) \leftarrow \#d$	3	Пересылка константы в ячейку ad
mov @Ri,A	$((Ri)) \leftarrow (A)$	1	Пересылка байта данных из аккумулятора в ячейку ПД ( $i=0,1$ )
mov @Ri,ad	$((Ri)) \leftarrow (ad)$	2	Пересылка байта данных из ячейки ad в ячейку ПД ( $i=0,1$ )
mov @Ri,#d	$((Ri)) \leftarrow \#d$	2	Пересылка константы в ячейку ПД ( $i=0,1$ )
mov DPTR,#d16	$(DPTR) \leftarrow \#d16$	3	Загрузка указателя данных
movc A, @A+DPTR	$(A) \leftarrow ((A+DPTR))$	1	Пересылка байта константы из ПП в аккумулятор
movc A, @A+PC	$(A) \leftarrow ((A+PC+1))$	1	Пересылка байта константы из ПП в аккумулятор
movx A,@Ri	$(A) \leftarrow ((Ri))$	1	Пересылка байта данных из ячейки ВПД ( $i=0,1$ ) в аккумулятор
movx A, @DPTR	$(A) \leftarrow ((DPTR))$	1	Пересылка байта данных из ячейки расширенной ВПД в аккумулятор



## Окончание таблицы П1

Мнемокод	Операция	Кол-во байт	Описание
movx @Ri,A	$((Ri)) \leftarrow (A)$	1	Пересылка байта данных аккумулятора в ячейку ВПД ( $i=0,1$ )
movx @DPTR,A	$((DPTR)) \leftarrow (A)$	1	Пересылка байта данных из аккумулятора в расширенную ВПД
push ad	$(SP) \leftarrow (SP)+1,$ $((SP)) \leftarrow (ad)$	2	Пересылка байта данных из ячейки ad в стек
pop ad	$(ad) \leftarrow ((SP)),$ $(SP) \leftarrow (SP)-1$	2	Пересылка байта данных из стека в ячейку ad
xch A,Rn	$(A) \leftrightarrow (Rn)$	1	Обмен данными аккумулятора и регистра Rn ( $n=0 \div 7$ )
xch A,ad	$(A) \leftrightarrow (ad)$	2	Обмен данными аккумулятора и ячейки ad
xch A,@Ri	$(A) \leftrightarrow ((Ri))$	1	Обмен данными аккумулятора и ячейки (Ri) ( $i=0,1$ )
xchd A,@Ri	$(A_{0...3}) \leftrightarrow$ $((Ri)_{0...3})$	1	Обмен младших тетрад аккумулятора и ячейки (Ri) ( $i=0,1$ )

## КОМАНДЫ АРИФМЕТИЧЕСКИХ ОПЕРАЦИЙ

Группа содержит 24 команды. Краткое описание команд представлено в таблице П2.

Таблица П2 – Команды арифметических операций

Мнемокод	Операция	Кол-во байт	Описание
add A,Rn	$(A) = (A) + (Rn)$	1	Сложение байта данных аккумулятора и регистра Rn ( $n=0 \div 7$ )
add A,ad	$(A) = (A) + (ad)$	2	Сложение байта данных аккумулятора и ячейки ad
add A,@Ri	$(A) = (A) + ((Ri))$	1	Сложение байта данных аккумулятора и ячейки (Ri) ( $i=0,1$ )
add A,#d	$(A) = (A) + \#d$	2	Сложение байта данных аккумулятора с константой

Мнемокод	Операция	Кол-во байт	Описание
addc A,Rn	$(A)=(A)+(Rn)+(C)$	1	Сложение байта данных аккумулятора, регистра Rn ( $n=0\div 7$ ) и переноса
addc A,ad	$(A)=(A)+(ad)+(C)$	2	Сложение байта данных аккумулятора, ячейки ad и переноса
addc A,@Ri	$(A)=(A)+((Ri))+(C)$	1	Сложение байта данных аккумулятора, ячейки (Ri) ( $i=0,1$ ) и переноса
addc A,#d	$(A)=(A)+\#d+(C)$	2	Сложение байта данных аккумулятора с константой и переносом
da A	-	1	Десятичная коррекция аккумулятора
subb A,Rn	$(A)=(A)-(Rn)-(C)$	1	Вычитание из данных аккумулятора байта из регистра Rn ( $n=0\div 7$ ) и бита заема
subb A,ad	$(A)=(A)-(ad)-(C)$	2	Вычитание из данных аккумулятора байта из ячейки ad и заема
subb A,@Ri	$(A)=(A)-((Ri))-(C)$	1	Вычитание из данных аккумулятора байта ячейки (Ri) ( $i=0,1$ ) и заема
subb A,#d	$(A)=(A)-\#d-(C)$	2	Вычитание из данных аккумулятора константы и заема
inc A	$(A)=(A)+1$	1	Инкремент аккумулятора
inc Rn	$(Rn)=(Rn)+1$	1	Инкремент регистра Rn ( $n=0\div 7$ )
inc ad	$(ad)=(ad)+1$	2	Инкремент ячейки ad
inc @Ri	$((Ri))=((Ri))+1$	1	Инкремент ячейки (Ri) ( $i=0,1$ )
inc DPTR	$(DPTR)=(DPTR)+1$	1	Инкремент указателя данных
dec A	$(A)=(A)-1$	1	Декремент аккумулятора
dec Rn	$(Rn)=(Rn)-1$	1	Декремент регистра Rn ( $n=0\div 7$ )
dec ad	$(ad)=(ad)-1$	2	Декремент ячейки ad
dec @Ri	$((Ri))=((Ri))-1$	1	Декремент ячейки (Ri) ( $i=0,1$ )
mul AB	$(B)(A)=(A)*(B)$	1	Умножение аккумулятора на регистр B
div AB	$(B).(A)=(A)/(B)$		Деление аккумулятора на регистр B

При выполнении команд `add`, `addc`, `subb`, `mul` и `div` устанавливаются флаги в регистре PSW. Флаг C (бит PSW.7) устанавливается при переносе (заеме) в 9-й разряд результата. Флаг AC (бит PSW.6) устанавливается при переносе между тетрадами при выполнении операций сложения и вычитания. Флаг OV (бит PSW.2) устанавливается при переносе из разряда d.6 в разряд d.7 числа. Флаг P (бит PSW.0) устанавливается, если число единичных бит в аккумуляторе нечетное, в противном случае  $P = 0$ .

### КОМАНДЫ ЛОГИЧЕСКИХ ОПЕРАЦИЙ

Группа содержит 24 команды. Краткое описание команд представлено в таблице ПЗ.

Таблица ПЗ – Команды логических операций

Мнемокод	Операция	Кол-во байт	Описание
<code>anl A,Rn</code>	$(A)=(A) \text{ AND } (Rn)$	1	Логическое И данных аккумулятора и регистра Rn ( $n=0\div 7$ )
<code>anl A,ad</code>	$(A)=(A) \text{ AND } (ad)$	2	Логическое И данных аккумулятора и ячейки ad
<code>anl A,@Ri</code>	$(A)=(A) \text{ AND } ((Ri))$	1	Логическое И данных аккумулятора и ячейки (Ri) ( $i=0,1$ )
<code>anl A,#d</code>	$(A)=(A) \text{ AND } \#d$	2	Логическое И данных аккумулятора константы
<code>anl ad,A</code>	$(ad)=(ad) \text{ AND } (A)$	2	Логическое И данных ячейки ad и аккумулятора
<code>anl ad,#d</code>	$(ad)=(ad) \text{ AND } \#d$	3	Логическое И данных ячейки ad и константы
<code>orl A,Rn</code>	$(A)=(A) \text{ OR } (Rn)$	1	Логическое ИЛИ данных аккумулятора и регистра Rn ( $n=0\div 7$ )
<code>orl A,ad</code>	$(A)=(A) \text{ OR } (ad)$	2	Логическое ИЛИ данных аккумулятора и ячейки ad
<code>orl A,@Ri</code>	$(A)=(A) \text{ OR } ((Ri))$	1	Логическое ИЛИ данных аккумулятора и ячейки (Ri) ( $i=0,1$ )
<code>orl A,#d</code>	$(A)=(A) \text{ OR } \#d$	2	Логическое ИЛИ данных аккумулятора константы
<code>orl ad,A</code>	$(ad)=(ad) \text{ OR } (A)$	2	Логическое ИЛИ данных ячейки ad и аккумулятора
<code>orl ad,#d</code>	$(ad)=(ad) \text{ OR } \#d$	3	Логическое ИЛИ данных ячейки ad и константы

Мнемокод	Операция	Кол-во байт	Описание
hrl A,Rn	$(A)=(A) \text{ OR } (Rn)$	1	Исключающее ИЛИ данных аккумулятора и регистра Rn ( $n=0\div 7$ )
hrl A,ad	$(A)=(A) \text{ OR } (ad)$	2	Исключающее ИЛИ данных аккумулятора и ячейки ad
hrl A,@Ri	$(A)=(A) \text{ OR } ((Ri))$	1	Исключающее ИЛИ данных аккумулятора и ячейки (Ri) ( $i=0,1$ )
hrl A,#d	$(A)=(A) \text{ OR } \#d$	2	Исключающее ИЛИ данных аккумулятора константы
hrl ad,A	$(ad)= (ad) \text{ OR } (A)$	2	Исключающее ИЛИ данных ячейки ad и аккумулятора
hrl ad,#d	$(ad)= (ad) \text{ OR } \#d$	3	Исключающее ИЛИ данных ячейки ad и константы
clr A	$(A)=0$	1	Сброс аккумулятора
cpl A	$(A)=\text{NOT } (A)$	1	Инверсия бит аккумулятора
rl A	$(A_{n+1})= (A_n),$ $(A_0)= (A_7)$	1	Циклический сдвиг аккумулятора влево
rlc A	$(A_{n+1})= (A_n),$ $(A_0)= (C), C)=(A_7)$	1	Циклический сдвиг аккумулятора влево через перенос
rr A	$(A_n)= (A_{n+1}),$ $(A_7)= (A_0)$	1	Циклический сдвиг аккумулятора вправо
rrc A	$(A_n)= (A_{n+1}),$ $(A_7)= (C), (C)=(A_0)$	1	Циклический сдвиг аккумулятора вправо через перенос
swap A	$(A_{0\dots 3})\leftrightarrow (A_{4\dots 7})$	1	Обмен местами тетрад в аккумуляторе

### КОМАНДЫ ОПЕРАЦИЙ НАД БИТАМИ

Группа содержит 12 команд. Эти команды позволяют выполнять операции над отдельными битами: сброс, установку, инверсию бита, а также логические И и ИЛИ. Краткое описание команд представлено в таблице П4.

Таблица П4 – Команды операций над битами

Мнемокод	Операция	Кол-во байт	Описание
clr C	$(C)\leftarrow 0$	1	Сброс переноса
clr bit	$(bit)\leftarrow 0$	2	Сброс бита
setb C	$(C)\leftarrow 1$	1	Установка переноса

Мнемокод	Операция	Кол-во байт	Описание
setb bit	(bit) $\leftarrow$ 1	2	Установка бита
cpl C	(C) $\leftarrow$ NOT (C)	1	Инверсия переноса
cpl bit	(bit) $\leftarrow$ NOT (bit)	2	Инверсия бита
anl C,bit	(C) $\leftarrow$ (C) AND(bit)	2	Логическое И переноса и бита
anl C,/bit	(C) $\leftarrow$ (C) AND (NOT (bit))	2	Логическое И переноса и инверсии бита
orl C,bit	(C) $\leftarrow$ (C) OR (bit)	2	Логическое ИЛИ переноса и бита
orl C,/bit	(C) $\leftarrow$ (C) OR (NOT (bit))	2	Логическое ИЛИ переноса и инверсии бита
mov C,bit	(C) $\leftarrow$ (bit)	2	Пересылка бита в перенос
mov bit,C	(bit) $\leftarrow$ (C)	2	Пересылка переноса в бит

### КОМАНДЫ УПРАВЛЕНИЯ ХОДОМ ВЫЧИСЛЕНИЙ

Группа содержит 23 команды безусловного и условного переходов, вызова подпрограмм и возврата из подпрограмм. Краткое описание команд представлено в таблице П5.

Таблица П5 – Команды операций над битами

Мнемокод	Операция	Кол-во байт	Описание
ljmp ad16	(PC) $\leftarrow$ ad16	3	Переход во всем пространстве памяти программ
ajmp ad11	(PC <sub>0...10</sub> ) $\leftarrow$ ad11	2	Переход в пределах страницы в 2 КБ
sjmp rel	(PC) $\leftarrow$ (PC)+ rel	2	Переход в пределах страницы в 256 байт
jmp @A+DPTR	(PC) $\leftarrow$ (A)+ (DPTR)	1	Косвенный относительный переход
jz rel	If(A)=0: (PC) $\leftarrow$ (PC)+ rel	2	Переход, если в аккумуляторе ноль
jnz rel	If(A) $\neq$ 0: (PC) $\leftarrow$ (PC)+ rel	2	Переход, если в аккумуляторе не ноль
jc rel	If(C)=1: (PC) $\leftarrow$ (PC)+ rel	2	Переход, если в бите переноса единица

Мnemonic	Operation	Count byte	Description
jnc rel	If(C)=0: (PC) $\leftarrow$ (PC)+ rel	2	Transition, if the carry bit is zero
jb bit,rel	If(bit)=1: (PC) $\leftarrow$ (PC)+ rel	3	Transition, if the specified bit is one
jnb bit,rel	If(bit)=0: (PC) $\leftarrow$ (PC)+ rel	3	Transition, if the specified bit is zero
jbc bit,rel	If(bit)=1: (PC) $\leftarrow$ (PC)+ rel	3	Transition, if the specified bit is one, and the following bit is cleared
djnz Rn,rel	(Rn)=(Rn)-1, If(Rn) $\neq$ 0: (PC) $\leftarrow$ (PC)+ rel	2	Decrement register and transition, if register is not zero
djnz ad,rel	(ad)=(ad)-1, If(ad) $\neq$ 0: (PC) $\leftarrow$ (PC)+ rel	3	Decrement data in cell ad and transition, if cell is not zero
cjne A,ad,rel	If(A) $\neq$ (ad): (PC) $\leftarrow$ (PC)+ rel	3	Comparison of accumulator and data in cell ad; transition, if not equal
cjne A,#d,rel	If(A) $\neq$ #d: (PC) $\leftarrow$ (PC)+ rel	3	Comparison of accumulator and constant; transition, if not equal
cjne @Ri,#d,rel	If(@Ri) $\neq$ #d: (PC) $\leftarrow$ (PC)+ rel	3	Comparison of cell data and constant; transition, if not equal
lcall ad16	(PC) $\leftarrow$ ad16	3	Transition to subprogram, located within the limits of the program memory area
acall ad11	(PC) $\leftarrow$ ad11	2	Transition to subprogram, located within the limits of the page in 2 KB
ret	(PC) $\leftarrow$ (SP)	1	Return from subprogram
reti	(PC) $\leftarrow$ (SP)	1	Return from subprogram interrupt processing
nop	(PC) $\leftarrow$ (PC)+1	1	No operation

Команда безусловного перехода `ljmp address` выполняет переход по абсолютному 16-битовому адресу, указанному в теле команды. Действие команды `ajmp address` аналогично действию команды `ljmp`, но используется

только 11-битовый адрес. Поэтому переход может быть выполнен только в пределах страницы памяти размером в 2 КБ.

В команде `sjmp rel` адрес перехода относительный. Смещение `rel` размером в 1 байт позволяет выполнить смещение в пределах -128 ... +127 байт относительно адреса текущей исполняемой команды.

Действие команд вызова подпрограмм полностью аналогично действию команд безусловного перехода. Отличие состоит в том, что адрес точки возврата сохраняется в стеке.

Большинство современных трансляторов программ с языка Ассемблер позволяют использовать обобщенную мнемонику `jmp address` для команды безусловного перехода и мнемонику `call address` для вызова подпрограммы. Конкретный тип команды транслятор определяет самостоятельно, исходя из длины перехода.

Бойков Владимир Иванович  
Быстров Сергей Владимирович  
Власов Сергей Михайлович  
Николаев Николай Анатольевич

**Микроконтроллерная техника систем управления.  
Методические указания к выполнению  
лабораторных работ**

**Учебно-методическое пособие**

В авторской редакции

Редакционно-издательский отдел Университета ИТМО

Зав. РИО

Н.Ф. Гусарова

Подписано к печати

Заказ №

Тираж

Отпечатано на ризографе



**Редакционно-издательский отдел**  
**Университета ИТМО**  
197101, Санкт-Петербург, Кронверкский пр., 49