МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

КАФЕДРА ЭЛЕКТРОННЫХ ВЫЧИСЛИТЕЛЬНЫХ МАШИН И СИСТЕМ

ПРОГРАММИРОВАНИЕ ОДНОКРИСТАЛЬНЫХ МИКРОЭВМ

Лабораторный практикум по учебным дисциплинам

«Проектирование контроллеров и специализированных вычислительных систем» и «Микроконтроллерные устройства» для студентов специальностей 1-40 02 01 «Вычислительные машины, системы и сети» и 1-36 04 02 «Промышленная электроника»

Часть 1

2013

УДК 004.382.7

В лабораторном практикуме излагается методика проведения лабораторных работ по программированию однокристальных микроЭВМ, выполняемых студентами специальностей «Промышленная электроника» и «Вычислительные машины, системы и сети» в рамках учебных дисциплин «Проектирование контроллеров и специализированных вычислительных систем» и «Микроконтроллерные устройства».

Первая часть лабораторного практикума предназначена для изучения структурной организации и архитектуры однокристальных микроЭВМ семейства MCS-51 и приобретения практических навыков по разработке программного обеспечения в интегрированной среде программирования Keil μVision на языке Ассемблера.

Каждая лабораторная работа рассчитана на четыре часа аудиторных занятий.

Материал лабораторного практикума может быть полезен при выполнении курсовых и дипломных проектов, в том числе студентами смежных специальностей.

Издаётся в 3-х частях. Часть 1.

Составители: Разумейчик В.С., к.т.н., доцент
Журавский В.И., доцент
Волков Е.Г., ст. преподаватель
Дереченник А.С., ст. преподаватель

Лабораторная работа № 1

ЗНАКОМСТВО С ИНТЕГРИРОВАННОЙ СРЕДОЙ ПРОГРАММИРОВАНИЯ Keil µVision

Цель работы. Изучить структуру программы на языке Ассемблера, познакомиться с интегрированной средой программирования Keil, получить навыки работы с программными проектами и средствами среды для отладки программ.

1 Теоретические сведения

1.1 Среда разработки программных проектов Keil µVision

µVision фирмы Keil является интегрированной средой разработки программного обеспечения для однокристальных микроЭВМ семейства MCS-51. Ее основные компоненты:

– стандартный интерфейс Windows, главное управляющее окно;
– полнофункциональный редактор исходных текстов;
– менеджер проектов;
– транслятор с языка Си;
– Ассемблер с поддержкой макросредств;
– компоновщик (редактор связей);
– библиотекарь объектных модулей;
– программный отладчик;
– конвертер в hex-файл;
– встроенная справочная система.

Интегрированная среда разработки µVision поддерживает проекты разработки программного обеспечения и объединяет все этапы разработки прикладной программы в единый рекурсивный процесс, когда в любой момент времени возможен быстрый возврат к любому предыдущему этапу. Этапы разработки программы:

– запись исходного текста программы на каком-либо языке программирования;

– компиляция или трансляция исходного текста в коды из системы команд микроконтроллера, используя транслятор или Ассемблер – прикладные программы, которые интерпретируют текстовый файл и создают объектные файлы, содержащие объектный код;

– компоновка объектных модулей в исполнительный файл с помощью редактора связей (компоновщика, линковщика);

– отладка программы, оптимизация и тестирование программы.

Для создания законченного приложения для микроконтроллера в среде проектирования µVision необходимо выполнить шесть следующих этапов:

– запустить интегрированную среду разработки µVision, создать файл проекта, задать тип микроконтроллера;

– создать новые исходные файлы программ на соответствующих языках программирования и включить их в проект в соответствующие группы древовидной иерархии в менеджере проекта;

– дополнить проект конфигурационными файлами и настроить их для заданной модели микроконтроллера;

– настроить опции проекта, рабочих групп и отдельных файлов, если необходимо;

– построить исполнительный файл проекта, выполнить его тестирование и отладку;

– создать hex-файл для программатора, записать программу в ПЗУ контроллера.

Для создания файла проекта в интегрированной среде разработки можно воспользоваться главным меню, выбрав команду из меню Project→ New Project, как показано на рисунке 1

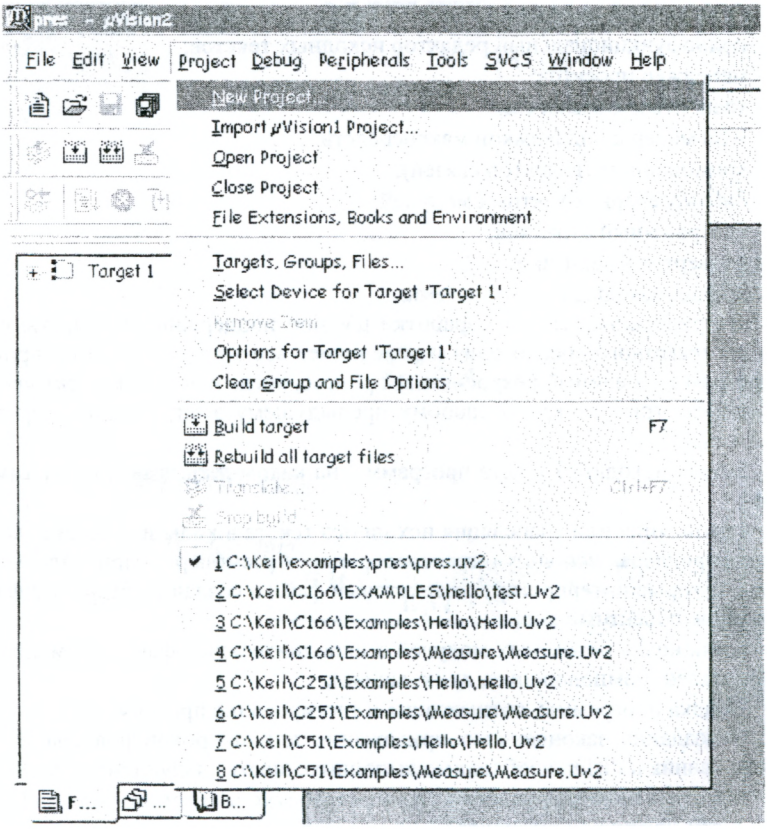

Рисунок 1 – Создание нового программного проекта

После создания новой директории и нового файла программного проекта, интегрированная среда программирования предлагает выбрать конкретный микроконтроллер из семейства MCS-51, как это показано на рисунке 2. В левом окне выбираются фирма-производитель и тип микроконтроллера, в правом окне отображается краткая характеристика выбранного устройства и его параметры.

Тип микроконтроллера очень важен для процесса компоновки программы. Тип важен отладчику для правильного отображения и обслуживания периферийных устройств.

После нажатия кнопки «OK» будет предложено добавить в рабочую папку и включить в проект файл начальной инициализации микроконтроллера (шаблон): «Copy Analog Devices Startup Code to Project Folder and Add File to Project?». Следует выбрать вариант ответа «Нет».

После создания программного проекта конечным файлом трансляции является абсолютный файл. Для загрузки в микросхему обычно используется hex-файл. Для создания такого hex-файла необходимо включить соответствующую опцию в свойствах программного проекта.

Изменение свойств программного проекта осуществляется через соответствующее диалоговое окно, вызвать которое можно, воспользовавшись главным меню, как показано на рисунке 3, либо нажав на кнопку изменения свойств программного проекта, как это показано на рисунке 4.

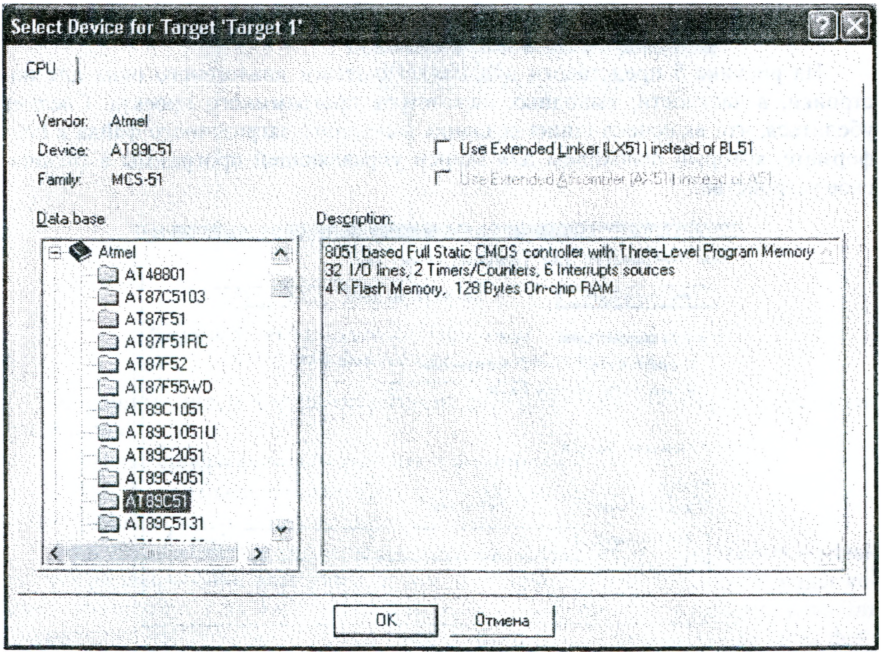

Рисунок 2 – Диалоговое окно выбора микроконтроллера

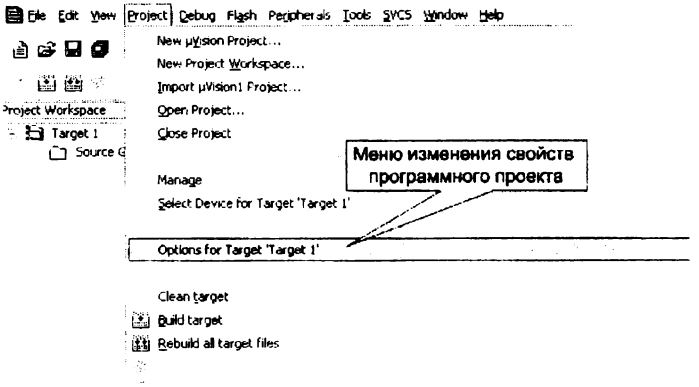

Рисунок 3 – Меню изменения свойств программного проекта

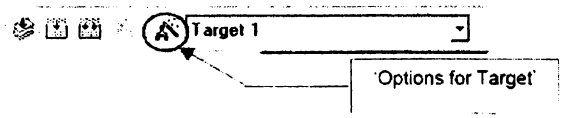

Рисунок 4 – Кнопка изменения свойств программного проекта

На рисунке 5 представлен вид появляющегося диалогового окна для настройки, в частности, выходных параметров программного проекта. Следует убедиться, что включена опция создания выходного загрузочного файла в hex-формате, который необходим для записи управляющей программы в физическое устройство.

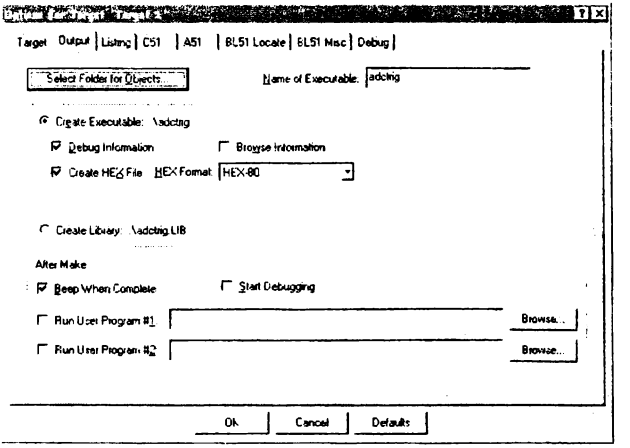

Рисунок 5 – Диалоговое окно настройки свойств программного проекта

Создание исходного текста программы начинается с создания нового текстового файла, воспользовавшись главным меню: File→ New. В появившемся новом окне можно набрать текст программы или скопировать уже заготовленный текст, после чего выбрать File→ Save as… для сохранения его в файле с расширением asm или с. После выполнения этих действий открывается окно текстового редактора, в котором можно редактировать исходный текст программы.

Теперь можно подключать к программному проекту файлы с исходным текстом программных модулей. Для этого можно щёлкнуть правой кнопкой мыши по значку группы файлов в окне менеджера проектов и выбрать опцию добавления файлов к программному проекту, как это показано на рисунке 6. Если проект состоит из нескольких программных модулей, опцию добавления файлов следует повторить соответствующее число раз.

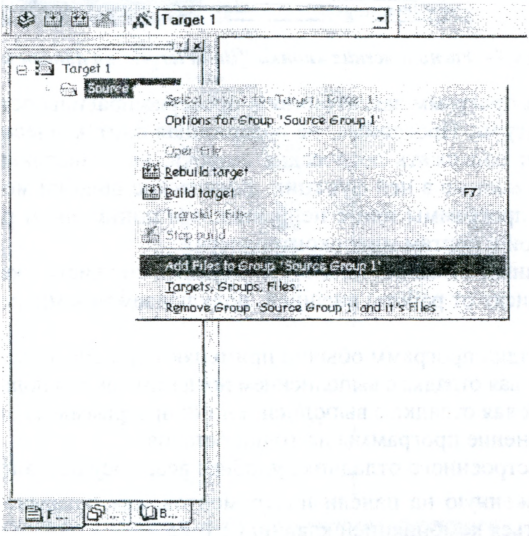

Рисунок 6 – Меню менеджера проектов с выбранной опцией добавления файлов к программному проекту

Хорошим стилем программирования считается наличие в проекте трех видов файлов:

– файлы, содержащие объявления переменных;

– файлы, содержащие описание пользовательских функций;

– файл с основным циклом программы.

При трансляции и компоновке программных модулей, написанных на языке программирования ассемблер, в листинг помещаются машинные коды команд процессора и их адреса относительно начала программного модуля. Для компиляции необходимо выбрать команду из меню Project→ Traslate<имя файла>. Все ошибки (Errors) и предупреждения (Warnings), возникшие при компиляции программного проекта, появятся в окне “Output Window”.

При трансляции исходного текста программы, написанной на языке программирования высокого уровня, таком как С, программа-транслятор может быть настроена так, что она будет создавать файл с текстом исходного модуля, написанного на ассемблере, или помещать этот текст в листинг программного модуля. Таким образом, появляется возможность создавать и отлаживать программу на языке высокого уровня, а затем переводить её на язык ассемблер. Это делает возможной дальнейшую оптимизацию программы.

Для компоновки (линковки) программы следует выбрать команду из меню Project→ Build target, либо нажать кнопку "Build target", расположенную на панели инструментов (рисунок 7). Для компоновки предусмотрена также клавиша F7. Компиляцию и компоновку можно объединить, нажав только эту клавишу.

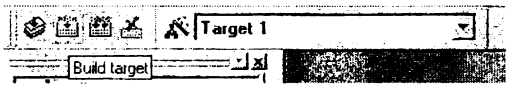

Рисунок 7 – Расположение кнопки "Build target" на панели инструментов

Отладка программ заключается в проверке правильности работы программы и аппаратуры. Программа, не содержащая синтаксических ошибок, тем не менее, может содержать логические ошибки, не позволяющие программе выполнять заложенные в ней функции. Логические ошибки могут быть связаны с алгоритмом программы или с неправильным пониманием работы аппаратуры, подключённой к портам микроконтроллера.

Встроенный отладчик позволяет отладить те участки кода программы, которые не зависят от работы внешней, подключаемой к микроконтроллеру аппаратуры.

Для отладки программ обычно применяют три способа:

– пошаговая отладка с выполнением по шагам команд подпрограммы (F10);

– пошаговая отладка с выполнением подпрограммы за один шаг (F11);

– выполнение программы до точки останова.

Вызов встроенного отладчика удобнее всего осуществить, нажав на кнопку , расположенную на панели инструментов, как показано на рисунке 8, или воспользоваться комбинацией клавиш Ctrl+F5.

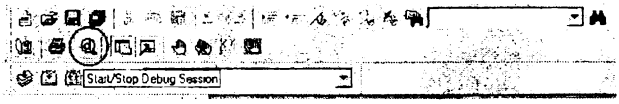

Рисунок 8 – Расположение кнопки вызова встроенного отладчика

Для более эффективной работы в режиме моделирования можно добавлять или скрывать дополнительные окна при помощи опции “View” командного меню. Перечислим основные окна, которые могут потребоваться при выполнении лабораторного курса:

– “Progect Window” (по умолчанию окно находится в левой части экрана) позволяет следить за изменением регистров микроконтроллера, например, при пошаговом выполнении программы;

– “Watches and Call Stack Window” (по умолчанию окно находится в правом нижнем углу) - окно контроля переменных и значений стека;

– “Memory Window” (по умолчанию скрыто) позволяет просматривать содержимое памяти микроконтроллера (рисунок 10). Адрес начала интересующего блока памяти задается в строке “Address” в шестнадцатеричном (0x0011) или в десятичном виде, при этом использование ключей c, d и x открывает доступ к памяти команд, данных и к внешней памяти соответственно. К примеру:

– d:0x0001 – просмотреть память данных, начиная с первой ячейки ;

– c:0x0011 – просмотреть память программ, начиная с 17-й ячейки;

– x:0x0000 – просмотреть внешнюю память данных, начиная с нулевой ячейки.

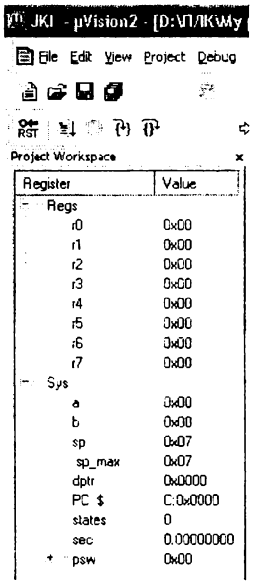

Рисунок 9 – Окно просмотра содержимого регистров

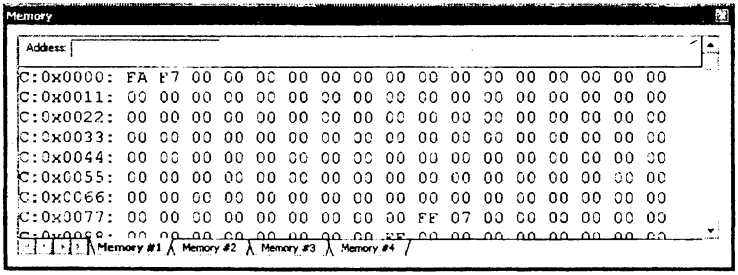

Рисунок 10 – Окно просмотра содержимого памяти

В окне предусмотрены четыре закладки, что позволяет одновременно отслеживать содержимое четырех типов памяти микроконтроллера.

1.2 Правила записи программ на языке Ассемблера

В любом вычислительном устройстве работать может лишь программа, написанная на машинном языке в абсолютном формате с конкретной привязкой всех адресов. Язык Ассемблера – это в большей степени машинный язык, но обильно использующий символические имена практически для всех элементов машинных команд, что существенно облегчает работу при написании программ. Каждой команде на языке Ассемблера ставится в соответствие всего одна машинная команда.

Исходный текст программы имеет стандартный формат. Каждая команда или директива записывается с новой строки и состоит из ограниченного числа полей: поле метки, поле операции, поля операндов, поле комментария. Эти поля могут отделяться друг от друга произвольным числом пробелов и табуляций:

Метка: операция операнд(ы) ; комментарий

В поле метки указывается произвольное символическое имя ячейки памяти, в которой хранится отмеченная меткой команда или операнд. Директивы ассемблера не преобразуются в машинные коды и не хранятся в памяти команд, а потому не могут иметь меток. За исключением директив резервирования и инициализации данных – метка (но уже без двоеточия) является именем ячейки памяти с данными.

В поле операции записывается мнемоническое обозначение команды или директивы Ассемблера, в поле операндов указываются операнды, разделяемые запятой. Команды могут иметь всего один операнд, либо вообще не иметь операндов.

Операнд может быть задан непосредственно (числом либо именем с префиксом #) или в виде его адреса в памяти. Все используемые в качестве операндов символические имена должны быть определены. Для чисел, представленных в системе счисления, отличной от десятичной системы (по умолчанию), используются суффиксы b (двоичная система счисления), q (восьмеричная), h (шестнадцатеричная). Современные Ассемблеры допускают указание в поле операнда выражения с использованием допустимых операторов (+, -, *, /, mod, or, and, xor, not, low, high), окончательное значение которого вычисляется в процессе трансляции.

Ассемблер транслирует исходную программу и берет на себя многие рутинные задачи, такие как присвоение действительных адресов, преобразование чисел и вычисление выражений, программист все же должен указать некоторые параметры: начальный адрес прикладной программы, конец программы, форматы данных и т.д. Для этого предназначены директивы, которые являются указаниями транслятору и не преобразуются в машинный код.

Директивы выбора сегмента (сегмент – блок кода или данных в памяти, который компилятор создает из исходного текста) позволяют переставлять сегменты в программе удобным пользователю образом, а также составлять из одноименных сегментов из разных модулей сегменты большего размера (этим занимается компоновщик). Так, директива BSEG выбирает абсолютный битовый

сегмент, CSEG – сегмент программы в машинном коде, DSEG – абсолютный сегмент резидентной памяти данных, RSEG – предварительно определенный перемещаемый сегмент, XSEG – абсолютный сегмент внешней памяти данных.

В основном модуле программы необходимо использовать директиву, указывающую абсолютный адрес кодового сегмента в памяти кода:

```
Cseg AT 0
```

Компоновщик этот сегмент не подвинет. Нулевой адрес памяти кода – адрес, откуда начинается обработка аппаратного сброса. В таблице 1 приведены основные блоки, из которых обычно состоит программа на языке Ассемблера.

Таблица 1 – Блоки программы на языке Ассемблера

Название блока	Примеры написания	Назначение
Блок директив препроцессору	$Include (Dat.asm)	подключить файл
Блок объявления сегмента данных (необязательно)	DataSeg segment code	объявление перемещаемого сегмента данных в памяти кода
Блок объявления переменных и глобальных функций	Perem data 20h	объявляется переменная размером в байт по адресу 20h
	Bit_E bit 0A4h	объявляется переменная размером в бит по адресу 0A4h
	Extrn Code (Func1, Func2)	объявляются внешние функции
	Extrn Data (Perem1, Perem2)	объявляются внешние переменные
	Perem3 equ 256	объявляется константа
Блок объявления сегмента кода	CodeSeg segment code	объявление перемещаемого сегмента кода
Команды компилятору и компоновщику	org 0h	изменение значения счетчика адреса текущего сегмента программ (записать следующую строку в память кода по адресу 0h)
	rseg DataSeg	выбор предварительно определенного перемещаемого сегмента
	rseg CodeSeg	
Программный блок	Start: … end	начало и конец основного цикла программы
	M1: … ret	организация функции
	Int0: … iret	организация обработчика прерывания

2 Порядок выполнения работы

2.1 Изучить теоретические сведения, изложенные в п.1.

2.2 Создать проект с именем Lab01 для микроконтроллера AT89C51.

2.3 Присоединить к проекту следующий программный код на языке Ассемблера:

```
Perem  data 20h
LCD segment code
rseg   LCD
       org 0h
       ljmp Start
Start:
       mov  R0,#20
       mov  Perem,#10
       mov  A, R0
       add  A, Perem
end
```

2.4 Разделить программу на два файла: основной и содержащий объявление переменной Perem.

2.5 Перейти в режим отладки и описать изменение состояния регистров R0 и A, а также переменной Perem. Объяснить полученные значения. Записать в отчет содержимое регистров общего назначения и системных регистров после выполнения основного цикла программы.

3 Содержание отчета

Отчет о проделанной работе оформляется в соответствии с общими требованиями и должен состоять из следующих пунктов:

3.1 Цель работы.

3.2 Отчетные материалы и пояснительные замечания при выполнении заданий п.п. 2.1 ... 2.5.

3.3 Выводы по результатам проделанной работы.

4 Контрольные вопросы

4.1 Назначение и состав среды разработки Keil uVision.

4.2 Описание процесса создания нового проекта.

4.3 Необходимость создания выходного загрузочного файла в hex-формате.

4.4 Последовательность действий при создании и подключении к программному проекту файлов программных модулей.

4.5 Основные блоки программы на языке Ассемблера.

4.6 Способы отладки программ.

4.7 Дополнительные окна среды Keil uVision и их назначение.

5 Рекомендуемая литература

5.1 Каспер, Э. Программирование на языке ассемблера для микроконтроллеров семейства i8051. – М.: Горячая линия-Телеком, 2003.

5.2 Сташин, В.В. Проектирование цифровых логических устройств на однокристальных микроконтроллерах / В.В. Сташин, А.В. Урусов. – М.: Энергоатомиздат, 1990.

Лабораторная работа №2

АРХИТЕКТУРА МИКРОКОНТРОЛЛЕРОВ СЕМЕЙСТВА MCS-51. КОМАНДЫ ПЕРЕСЫЛКИ ДАННЫХ

Цель работы. Изучить структуру и характеристики микроконтроллера AT89C51 и программную модель микроконтроллеров семейства MCS-51 в целом. Изучить особенности организации памяти программ и памяти данных, а также основные команды обращения к ним.

1 Теоретические сведения

1.1 Внутренняя структура и характеристики микроконтроллера AT89C51

Микроконтроллер AT89C51 выполнен на основе высокоуровневой n-МОП технологии. Его основные характеристики:

– восьмиразрядный центральный процессор, оптимизированный для реализации функций управления;
– максимальная частота встроенного тактового генератора – 12 МГц;
– адресное пространство памяти программ - 64 Кбайт;
– адресное пространство памяти данных - 64 Кбайт;
– внутренняя память программ - 4 Кбайт;
– внутренняя память данных - 128 байт;
– дополнительные возможности по выполнению операций булевой алгебры;
– 32 двунаправленные и индивидуально адресуемые линии ввода/вывода;
– два шестнадцатиразрядных многофункциональных таймера/счетчика;
– полнодуплексный асинхронный последовательный приемопередатчик;
– векторная система прерываний с двумя уровнями приоритета и пятью источниками событий.

Структурная схема микроконтроллера, представленная на рисунке 11, демонстрирует его внутреннее устройство – основные узлы и устройства микроконтроллера и их взаимосвязь через двунаправленную восьмиразрядную внутреннюю шину. В состав микроконтроллера AT89C51 входят:

– арифметико-логическое устройство (ALU),
– устройство управления (CU),
– DCU,
– OSC,
– IR,
– параллельные порты ввода/вывода (P0-P3),
– резидентная память программ (RPM),
– резидентная память данных (RDM),
– блок прерываний, таймеров и последовательного порта,
– блок регистров специальных функций.

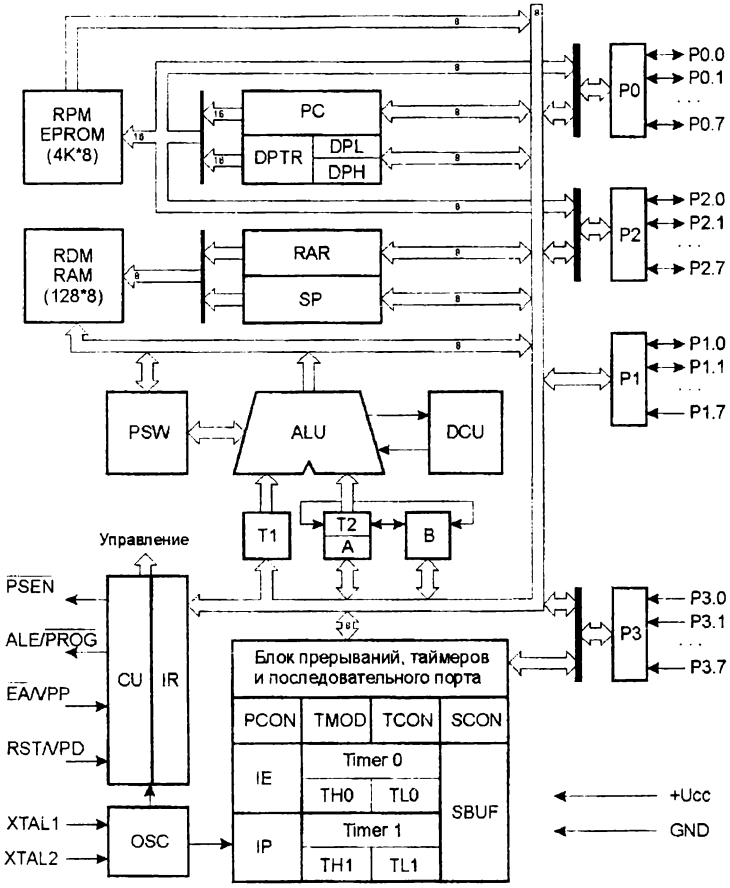

Рисунок 11 – Структурная схема микроконтроллера AT89C51

Резидентные (внутренние) память программ и память данных физически и логически разделены, имеют различные механизмы адресации, работают под управлением различных управляющих сигналов и выполняют разные функции. Память программ предназначена для хранения команд, констант, управляющих слов инициализации, таблиц перекодировки входных и выходных переменных и т.п. Она имеет 16-битную шину адреса, а доступ к ее адресному пространству может осуществляться как через 16-разрядный программный счётчик (PC), так и через 16-разрядный регистр-указатель данных (DPTR). Резидентная память данных предназначена для хранения изменяемых в процессе выполнения прикладной программы данных, адресуется одним байтом и имеет емкость 128 байт. Память программ, так же как и память данных, может быть расширена до 64 Кбайт путем подключения внешних микросхем.

Параллельные порты ввода/вывода могут быть использованы для организации ввода/вывода информации по двунаправленным линиям передачи, при этом порт P0 является двунаправленным, а порты P1...P3 – квазидвунаправленными. Каждый порт содержит программно-управляемые регистры: регистр-защёлку, входной буфер и выходной драйвер. Большим преимуществом является возможность использования всех линий порта по отдельности, независимо от остальных. По сигналу RST в регистры-защёлки всех портов автоматически записываются единицы, тем самым осуществляется их настройка на режим ввода.

Выходные драйверы портов P0 и P2, а также входной буфер порта P0 используются при обращении к внешней памяти, при наличии таковой. При этом через порт P0 в режиме временного мультиплексирования передается сначала младший байт адреса, а затем передается либо принимается байт данных. Через порт P2 передается старший байт 16-разрядного адреса.

Выводы порта P3 также имеют альтернативное назначение и могут быть использованы для работы с блоком прерываний, таймеров и последовательного порта и при обращении к внешней памяти данных.

Универсальный асинхронный приёмопередатчик (UART – Universal Asynchronous Receiver-Transmitter) осуществляет прием и передачу информации, представленной последовательным кодом (младшими битами вперед), в полном дуплексном режиме обмена. В состав UART, называемого часто последовательным портом, входят принимающий и передающий сдвигающие регистры, а также специальный буферный регистр (SBUF) приёмопередатчика. Регистр SBUF представляет собой два независимых регистра: буфер приёмника и буфер передатчика. Загрузка байта в SBUF приводит к автоматической перезаписи байта в сдвигающий регистр передатчика и немедленно вызывает начало процесса передачи через последовательный порт. Когда байт считывается из SBUF, это значит, что его источником является приёмник последовательного порта. Наличие буферного регистра приёмника позволяет совмещать чтение принятого ранее байта и приём очередного байта. Но если к моменту окончания приёма байта предыдущий не был считан из буфера, он будет потерян.

И, наконец, независимые программно-управляемые 16-битные таймеры/счетчики могут работать как в режиме таймера (для отсчета временных интервалов), так и в режиме счетчика (для подсчета числа импульсов). При работе в качестве таймера содержимое таймера/счетчика инкрементируется в каждом машинном цикле. При работе в качестве счётчика его содержимое инкрементируется под воздействием перехода из 1 в 0 внешнего входного сигнала, подаваемого на соответствующий вход микроконтроллера (T0 или T1).

1.2 Программная модель микроконтроллеров семейства MCS-51

Программная модель микроконтроллера – это его программно-видимые свойства: набор пользовательских и системных регистров, система команд, режимы адресации операндов и т.п.

В распоряжении пользователя имеются четыре банка по восемь регистров общего назначения (R0-R7), расположенных во внутренней (резидентной) памяти

данных по адресам 00H...1FH. При этом, под именами R0...R7 (в один момент времени) возможно использование регистров только одного из четырёх банков. Переключение между банками для доступа ко всем 32 РОН (но при помощи все тех же имен R0...R7) осуществляется с помощью бит RS1 и RS0 регистра PSW, согласно таблице 2.

К области регистров общего назначения внутренней памяти данных примыкает пространство битовой адресации (диапазон адресов 20H ... 3FH) и область пользовательских переменных, не допускающих битовой адресации (диапазон адресов 40H ... 7FH).

Доступными программе пользователя являются и регистры специальных функций, занимающие диапазон адресов 80H ... FFH внутренней памяти данных. Некоторые из них допускают адресацию отдельных бит.

Таблица 2 – Блок регистров специальных функций

Символ	Наименование	Адрес
* ACC	Аккумулятор	0E0H
* B	Регистр-расширитель аккумулятора	0F0H
* PSW	Слово состояния программы	0D0H
	Регистры-указатели	
SP	Регистр-указатель стека	81H
DPTR	Регистр-указатель данных (DPH)	83H
	(DPL)	82H
	Параллельные порты ввода/вывода	
* P0	Порт 0	80H
* P1	Порт 1	90H
* P2	Порт 2	0A0H
* P3	Порт 3	0B0H
	Регистры специальных функций	
* IP	Регистр приоритетов	0B8H
* IE	Регистр маски прерываний	0A8H
TMOD	Регистр режима таймера/счетчика	89H
* TCON	Регистр управления/статус таймера	88H
* SCON	Регистр управления приемопередатчиком	98H
PCON	Регистр управления мощностью	87H
	Таймеры/счетчики	
TH0	Таймер 0 (старший байт)	8CH
TL0	Таймер 0 (младший байт)	8AH
TH1	Таймер 1 (старший байт)	8DH
TL1	Таймер 1 (младший байт)	8BH
	Последовательный порт	
SBUF	Буфер приемопередатчика	99H
* - допускают битовую адресацию		

Аккумулятор является источником операнда и местом фиксации результата при выполнении арифметических, логических операций и ряда операций передачи данных. Кроме того, только с использованием аккумулятора могут быть выполнены операции сдвигов, проверка на нуль, формирование флага паритета и т.п.

Регистр слова состояния программы (PSW) фиксирует ряд признаков операции (флагов), формируемых при выполнении многих команд в ALU. В таблице 3 приводится перечень флагов PSW.

Таблица 3 – Формат слова состояния программы PSW

<table>
<tr><th>Имя</th><th>Разряд</th><th>Назначение</th></tr>
<tr><td>C</td><td>PSW.7</td><td>Флаг переноса. Устанавливается и сбрасывается аппаратно или программно при выполнении арифметических и логических операций</td></tr>
<tr><td>AC</td><td>PSW.6</td><td>Флаг вспомогательного переноса. Устанавливается и сбрасывается аппаратно при выполнении команд сложения и вычитания и сигнализирует о переносе в бит 3</td></tr>
<tr><td>F0</td><td>PSW.5</td><td>Флаг 0. Может быть установлен, сброшен или проверен программой как флаг, специфицируемый пользователем</td></tr>
<tr><td>RS1
RS0</td><td>PSW.4
PSW.3</td><td>Выбор банка регистров. Устанавливается и сбрасывается программно для выбора рабочего банка регистров
<table>
<tr><th>RS1</th><th>RS0</th><th>Банк</th><th>Границы адресов</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>00H – 07H</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>08H – 0FH</td></tr>
<tr><td>1</td><td>0</td><td>2</td><td>10H – 17H</td></tr>
<tr><td>1</td><td>1</td><td>3</td><td>18H – 1FH</td></tr>
</table></td></tr>
<tr><td>OV</td><td>PSW.2</td><td>Флаг переполнения. Устанавливается и сбрасывается аппаратно при выполнении арифметических операций</td></tr>
<tr><td>-</td><td>PSW.1</td><td>Не используется</td></tr>
<tr><td>P</td><td>PSW.0</td><td>Флаг паритета. Устанавливается и сбрасывается аппаратно в каждом цикле и фиксирует нечётное/чётное число единичных битов в аккумуляторе, т.е. выполняет контроль по четности</td></tr>
</table>

Наиболее “активным” флагом PSW является флаг переноса, который принимает участие и модифицируется в процессе выполнения множества операций, включая сложение, вычитание и сдвиги. Флаг переполнения (OV) фиксирует арифметическое переполнение при операциях над целыми числами со знаком и делает возможным использование арифметики в дополнительных кодах. Он устанавливается, если результат операции сложения или вычитания не укладывается в семи битах и старший бит результата не может интерпретироваться как знаковый. При выполнении операции деления флаг OV сбрасывается, а в случае деления на нуль устанавливается. При умножении флаг OV устанавливается, если результат больше 255.

Флаг C устанавливается, если в старшем бите результата возникает перенос или заем. При выполнении операций умножения и деления флаг C сбрасывается.

Флаг AC устанавливается, если при выполнении операции сложения (или вычитания) возник межтетрадный перенос (или заем из старшей тетрады).

Флаг паритета P напрямую зависит от текущего значения аккумулятора. То есть его значение изменяется всеми командами, которые изменяют содержимое аккумулятора. Если число единичных битов аккумулятора нечётное, то флаг P устанавливается, а если чётное – сбрасывается. При нулевом значении аккумулятора P=0.

Восьмибитный указатель стека SP используется для адресации любой области RDM. Его содержимое инкрементируется прежде, чем данные будут сохранены в стеке в ходе выполнения команд PUSH и CALL. Содержимое SP декрементируется после выполнения команд POP и RET.

Двухбайтный указатель данных DPTR обычно используется для фиксации 16-битного адреса в операциях обращения к внешней памяти. Командами микроконтроллера регистр-указатель данных может быть использован и как 16-битный регистр, и как два независимых восьмибитных регистра (DPH и DPL).

Регистры портов P0...P3 используются для организации параллельного ввода-вывода, при этом предоставляется возможность использования всех линий портов по отдельности, независимо от остальных. Кроме того, выходные драйверы портов P0 и P2, а также входной буфер порта P0 используются при обращении к внешней памяти, а выводы порта P3 могут быть использованы для работы с блоком прерываний, таймеров и последовательного порта и при обращении к внешней памяти данных. Альтернативное назначение выводов порта P3 приведено в таблице 4.

Таблица 4 – Альтернативное назначение выводов порта P3

Сигнал	Разряд	Назначение
RD	P3.7	Чтение. Активный сигнал низкого уровня формируется аппаратно при обращении к внешней памяти данных
WR	P3.6	Запись. Активный сигнал низкого уровня формируется аппаратно при обращении к внешней памяти данных
T1	P3.5	Вход таймера/счётчика 1 или тест-вход
T0	P3.4	Вход таймера/счётчика 0 или тест-вход
INT1	P3.3	Вход запроса прерывания 1. Воспринимается сигнал низкого уровня или срез
INT0	P3.2	Вход запроса прерывания 0. Воспринимается сигнал низкого уровня или срез
TXD	P3.1	Выход передатчика последовательного порта в режиме UART. Выход синхронизации в режиме регистра сдвига
RXD	P3.0	Вход приёмника последовательного порта в режиме UART. Ввод/вывод данных в режиме регистра сдвига

Регистры специальных функций IP, IE, TMOD, TCON, SCON, PCON используются для фиксации и программного изменения управляющих бит и бит состояния системы прерывания, таймеров/счётчиков, приёмопередатчика последовательного порта и для управления энергопотреблением.

Систему команд микроконтроллеров MCS-51 составляют 111 базовых команд (при общем количестве 255), которые по функциональному признаку могут быть разделены на пять групп:

- команды пересылки данных;
- команды арифметических операций;
- команды логических операций;
- команды битовых операций;
- команды передачи управления.

Формат машинной команды состоит из поля кода операции (КОП) и адресного поля, для адресации операнда (ов). Для кодирования всей системы команд микроконтроллера достаточным является размер поля КОП в восемь бит. Большинство команд (94 команды) имеют общий формат команды в один или два байта и выполняются за один или два машинных цикла, длительность которого при тактовой частоте 12 МГц составляет 1 мкс. Синтаксис базовых ассемблерных команд с указанием их характеристик (тип, длина команды в байтах, продолжительность исполнения команды в циклах) приведены в приложении 1. В микроконтроллере используются следующие ***режимы адресации*** данных:

- регистровый;
- непосредственный;
- неявный;
- прямой;
- символический;
- косвенный.

Регистровый способ адресации используется для операндов, хранящихся в одном из 32-х регистров общего назначения, доступных под именами R0-R7, и позволяет получать короткие команды. Например, команда декремента содержимого регистра DEC R0 имеет однобайтный формат – номер регистра кодируется в поле КОП.

Непосредственный способ адресации служит для передачи числового значения операнда непосредственно в самой команде. Это самый быстрый способ адресации операнда, но формат машинной команды удлиняется на соответствующий размер операнда. Значение одно- или двухбайтового операнда располагается в программной памяти непосредственно за байтом КОП команды. Например, команда загрузки байта в аккумулятор MOV A, #10 имеет двухбайтный формат: байт КОП и байт данных. Команда загрузки двухбайтового значения в регистр-указатель данных MOV DPTR, #1FFFh имеет трехбайтный формат: байт КОП и два байта данных.

Неявный способ – адрес операнда не указывается в машинной команде, а предопределен самим кодом операции КОП. Этот способ адресации позволяет получать машинные команды минимального формата. Так, формат команды сброса флага переноса CLR C состоит только из одного поля КОП, а сама команда занимает 1 байт в памяти программ.

Прямой способ адресации предполагает указание в команде одно- или двухбайтового адреса, определяющего местоположения операнда во внутренней памяти данных. Таким образом формат машинной команды удлиняется на соответствующий размер адреса. Например, команда сложения ADD A, 10 (сложение аккумулятора с содержимым памяти по адресу Ah) имеет двухбайтный формат: байт КОП и байт числового значения адреса. Команда длинного перехода LJMP 1C00h имеет трехбайтный формат: первый байт КОП и два байта адреса. Регистры общего назначения, расположенные во внутренней памяти данных (в диапазоне адресов 00H ... 1FH), можно также адресовать прямым режимом адресации, но такой вариант менее предпочителен из-за более длинного формата машинной команды.

Символический способ можно рассматривать как разновидность прямого режима адресации операнда, отличие состоит в указании присвоенного ранее символического имени адреса операнда вместо числового значения этого адреса. Такой режим используется для адресации некоторых регистров специальных функций (в том числе портов) и отдельных их бит. Символическое имя бита имеет следующую структуру: <имя регистра или порта>.<номер бита>. Например, символическое имя пятого бита аккумулятора будет следующим: ACC.5. Символические имена регистров специальных функций являются зарезервированными словами, поэтому нет необходимости их определять с помощью директив ассемблера.

Косвенная адресация предполагает указание операнда посредством адреса, содержащегося в регистре либо в регистровой паре. Символ @ в ассемблерной команде перед именем регистра означает, что необходимый для исполнения команды операнд располагается в памяти по адресу, указанному в регистре, при этом в качестве регистров могут использоваться только R0, R1, DPTR, A+DPTR и A+PC. Данный способ адресации позволяет повысить гибкость программирования и уменьшить формат команды по сравнению с прямым режимом адресации. Например, команда пересылки MOV A, @R0 (загрузить в аккумулятор содержимое ячейки внутренней памяти, восьмиразрядный адрес которой содержится в регистре R0) занимает один байт памяти программ и выполняется за один командный цикл. Команда пересылки MOVX A, @DPTR (загрузить в аккумулятор содержимое ячейки внешней памяти данных, адрес которой содержится в двухбайтном регистре-указателе DPTR) и команда перехода JMP @A+DPTR (переход на адрес, вычисляемый как сумма значений, содержащихся в регистрах A и DPTR) также имеют размер один байт, но выполняются уже за два командных цикла.

Многие команды для указания операндов комбинируют различные способы адресации. Например, команда MOV @R0, #15h (загрузить байт данных 15h в ячейку внутренней памяти, восьмиразрядный адрес которой содержится в регистре R0) использует непосредственный режим адресации операнда-источника и косвенный для операнда-приемника. Но не все сочетания режимов адресации операндов допустимы. Так, в зависимости от способа адресации и местораспо-

ложения операнда можно выделить девять типов операндов, между которыми возможен информационный обмен:

- A – аккумулятор,
- Rn – один из семи регистров выбранного банка,
- @Ri – косвенно адресуемая ячейка внутренней либо внешней памяти данных,
- @DPTR – косвенно адресуемая ячейка внешней памяти данных или памяти программ,
- @PC – косвенно адресуемая ячейка памяти программ,
- #d – непосредственное значение восьмиразрядных данных,
- #d16 – непосредственное значение 16-разрядных данных,
- bit – прямоадресуемый бит,
- ad – адрес внутренней памяти данных.

Схема возможных сочетаний типов операндов в командах пересылки данных приведена на рисунке 12.

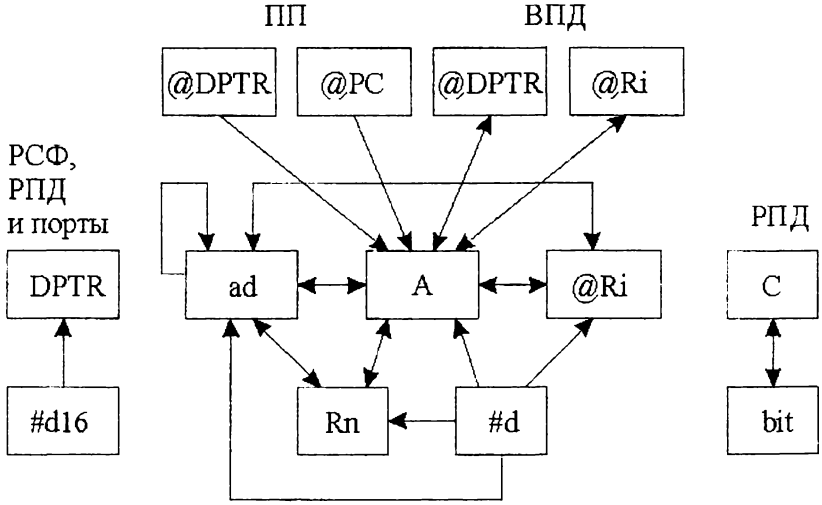

Рисунок 12 – Схема возможных путей пересылки данных

Обращение к аккумулятору может быть выполнено с использованием неявной и прямой адресации. В зависимости от способа адресации аккумулятора применяется одно из символических имен: A или ACC (символическое имя прямого адреса аккумулятора). При прямой адресации обращение к аккумулятору производится как к одному из регистров специальных функций, и его адрес указывается во втором байте команды. Использование неявной адресации аккумулятора предпочтительнее, но не всегда возможно, например, при обращении к отдельным битам аккумулятора.

1.3 Команды пересылки данных

Команды пересылки данных выполняют копирование данных из операнда-источника в операнд-приемник.

Команда пересылки байта данных в/из внутренней памяти данных:

```
MOV <операнд-приемник>, <операнд-источник>
```

Операнд-источник может адресоваться любым режимом адресации, приемник – любым, кроме непосредственного. При этом допускается пересылка данных из памяти в память, если хотя бы один из операндов адресуется прямым режимом. Наиболее предпочтительным является использование в качестве одного из операндов аккумулятора А.

Команда пересылки байта данных в/из внешней памяти:

```
MOVX <операнд_приемник>, <операнд_источник>
```

Обязательно использование аккумулятора в качестве одного из операндов. Ячейка внешней памяти данных адресуется исключительно косвенным режимом адресации, при этом для доступа к 256 байтам внешней памяти данных используется 8-битный адрес, хранящийся в регистре R0 или R1, а при обращении к расширенной внешней памяти данных объемом 65536 байт – 16-битный адрес из регистра-указателя данных DPTR.

Команда пересылки байта данных из памяти программ в аккумулятор:

```
MOVC <операнд_приемник>, <операнд_источник>
```

Это позволяет использовать ПЗУ программ для хранения констант. Операнд-приемник – исключительно аккумулятор, способ адресации ячейки программной памяти – косвенный. В качестве указателя адреса может использоваться программный счетчик PC и регистр-указатель данных DPTR.

Команда загрузки в стек содержимого прямоадресуемого байта:

```
PUSH <операнд-источник>
```

Команда извлечения из стека байта и сохранение его в операнд-приемник, адресуемый прямым режимом адресации:

```
POP <операнд-приемник>
```

Обмен байтовыми данными операндов:

```
XCH <операнд1 >, <операнд2>
```

В качестве операнда1 следует указывать только аккумулятор, операнд2 допускается адресовывать регистровым, прямым и косвенным режимами адресации.

Следующий пример демонстрирует способ копирования данных из памяти программ в память данных:

```
                org   0h
                ljmp  Start
Bank0_R1        data  1
DataRAM         data  30h
Const13         equ   13h
Box:            db    '1 Text1', 0
                db    '3 Text3', 0, 13h

Start:
LoadRAM:
                mov   R0, #DataRAM
                mov   DPTR, #Box
cyc_1:
                clr   A
                movc  A, @A+DPTR
                cjne  A, #Const13, cyc_2
                clr   c
                ret
cyc_2:
                mov   @R0, A
                inc   R0
                inc   DPTR
                clr   A
                movc  A, @A + DPTR
                jnz   cyc_2
                sjmp  cyc_1
```

Поскольку переменные *Bank0_R1, DataRAM, Const13, Box* объявлены в сегменте кода, они хранятся в памяти программ. Для посимвольного копирования из памяти программ в память данных необходимы два указателя, которые будут инкрементироваться после каждого копирования. В качестве этих указателей использованы:

– регистр *R0 для указателя на память данных,* в него изначально загружается константа *30h* из переменной *DataRAM*;

– регистр *DPTR* для указателя на память программ, в него изначально загружается адрес первого элемента массива *Box*.

В цикле *cyc_1* происходит считывание в аккумулятор *A* символа из памяти программ по адресу из регистра *DPTR*, сравнение его с содержимым переменной *Const13* и переход на метку *cyc_2* в случае, если символы не равны. Вложенный цикл *cyc_2* выполняет копирование строки символов (признаком завершения строки является значение '0') из памяти программ в память данных.

Пересылку данных из памяти программ во внешнюю память данных можно реализовать следующим образом:

```
                org   0h
                ljmp  Start
Data:           db    1, 2, 3, 4, 5, 6, 0
Addrx           equ   1234h

Start:
                mov   DPTR, #Data
                mov   R6, #Low(Addrx)
                mov   R7, #High(Addrx)
cycle:
                clr   A
                movc  A, @A+DPTR
                jnz   next
                jmp   _end
next:
                mov   R5, A
                mov   A, R6
                xch   A, DPL
                mov   R6, A
                mov   A, R7
                xch   A, DPH
                mov   R7, A
                mov   A, R5
                movx  @DPTR, A
                inc   DPTR
                mov   A, R6
                xch   A, DPL
                mov   R6, A
                mov   A, R7
                xch   A, DPH
                mov   R7, A
                inc   DPTR
                sjmp  cycle
_end:
end.
```

В массиве *Data* в памяти программ хранятся данные, предназначенные для копирования, а в переменной *Addrx* – двухбайтовый адрес внешней памяти данных. Поскольку обращение и к внешней расширенной памяти данных, и к внутренней памяти программ возможно только через регистр-указатель *DPTR*, для временного сохранения одного из 16-разрядных указателей необходимо использование двух дополнительных промежуточных регистров, например *R6* и *R7*. Загрузив в указатель DPTR адрес ячейки памяти программ, посимвольное копирование данных из памяти программ во внешнюю память можно осуществить путем выполнения в цикле следующих операций: получение символа из памяти программ *movc A, @A+DPTR*; обмен значениями 16-разрядных указателей (из пары регистров R7 и R6 и из регистра DPTR, побайтно); копирование полученного в аккумуляторе символа во внешнюю память данных *movx @DPTR, A*; обмен значениями 16-разрядных указателей.

2 Порядок выполнения работы

2.1 Изучить теоретические сведения, представленные в п1.

2.2 Изучить код, представленный в п.1.3. Включить в код программы таблицу из десяти записей разной длины (например, фамилия студента), признак завершения записи – код 13h.

2.3 Включить в основный цикл программы процедуру перекодировки таблицы фамилий – каждой букве фамилии поставить в соответствие номер буквы в русском алфавите, а признак завершения записи перекодировать в число 128. В память данных микроконтроллера, начиная с адреса 30h, разместить результат такой перекодировки.

2.4 Включить в основный цикл программы процедуру сортировки записей исходной таблицы, отсортированную таблицу разместить во внешней памяти данных по адресу 4000h.

2.5 Запустить программу на выполнение. Проверить содержимое внутренней памяти данных, внутренней памяти программ и внешней памяти данных.

3 Содержание отчета

Отчет о проделанной работе оформляется в соответствии с общими требованиями и должен содержать:

3.1 Цель работы.

3.2 Схемы возможных сочетаний типов операндов в командах пересылки данных отдельно для команд mov, movc и movx.

3.3 Текст программы, с процедурами перекодировки и сортировки.

3.4 Содержимое внутренней и внешней памяти данных и памяти программ.

3.5 Выводы по результатам проделанной работы.

4 Контрольные вопросы

4.1 Общие характеристики микроконтроллеров семейства MCS-51.

4.2 Структурная организация микроЭВМ семейства MCS-51.

4.3 Программная модель микроконтроллера.

4.4 Регистры специальных функций.

4.5 Формат слова состояния программы.

4.6 Режимы адресации операндов и их возможные сочетания.

4.7 Обращение к внутренней памяти данных.

4.8 Обращение к внутренней памяти программ.

4.9 Обращение к внешней памяти.

5 Рекомендуемая литература

5.1 Предко, М. Руководство по микроконтроллерам: в 2-х т. – М.: Постмаркет, 2001.

5.2 Каспер, Э. Программирование на языке ассемблера для микроконтроллеров семейства i8051. – М.: Горячая линия-Телеком, 2003.

Лабораторная работа №3

РАСШИРЕНИЕ РАЗРЯДНОСТИ АРИФМЕТИЧЕСКИХ ОПЕРАЦИЙ

Цель работы. Изучить форматы основных арифметических команд микроконтроллера i8051 и возможные режимы адресации их операндов. Изучить форматы команд организации ветвлений и вызова подпрограмм. Научиться реализовывать алгоритмы расширения разрядности операндов для арифметических операций, выполняемых микроконтроллером i8051.

1 Теоретические сведения

1.1 Арифметические команды

Группу арифметических команд i8051 образуют 24 команды, выполняющие операции сложения, вычитания, умножения и деления, инкремента/декремента байтов, десятичной коррекции. Выполнение арифметических команд влияет на флаги.

Команда сложения:

```
ADD <приемник>, <источник>
```

выполняет сложение двух байтовых операндов без учета флага переноса C. Результат сложения сохраняется по адресу операнда-приемника. Устанавливает флаг переноса C в единицу в случае, если результат не помещается в байт, т.е. был перенос из старшего разряда.

Команда сложения с учетом флага переноса:

```
ADDC <приемник>, <источник>
```

выполняет сложение двух байтовых операндов с учетом флага переноса C. Результат сложения сохраняется по адресу операнда-приемника. Устанавливает флаг переноса C в единицу в случае, если был перенос из старшего разряда.

Команда вычитания:

```
SUBB <приемник>, <источник>
```

выполняет вычитание с учетом флага C. Результат сложения сохраняется по адресу операнда-приемника. Если результат операции отрицательный, флаг C устанавливается в единицу. Команда модифицирует флаги, представленные в таблице 2.

Команда умножения:

```
MUL AB
```

перемножает значения из регистров-аккумуляторов A и B. При этом старшая часть результата умножения помещается в регистр B, а младшая - в регистр A.

Команда деления:

```
DIV AB
```

выполняет деление значения из регистра A на значение из регистра B. При этом целая часть результата деления помещается в регистр A, а остаток – в регистр B.

1.2 Команды передачи управления

Группу команд передачи управления составляют команды безусловных переходов на указанный в команде адрес памяти программ (абсолютный либо относительный), команды условных переходов для организации ветвлений и команды вызова подпрограммы и возврата из подпрограммы. Результатом выполнения таких команд является изменение текущего значения счетчика команд.

Команды безусловного перехода:

```
LJMP <адрес>
AJMP <адрес>
```

осуществляют абсолютный (длинный либо короткий внутри страницы) переход на указанный адрес. А следующие команды:

```
SJMP <метка>
JMP @A+DPTR
```

выполняют, соответственно, короткий относительный и косвенный относительный переход.

Команда условного перехода по результату сравнения двух операндов:

```
CJNE <операнд1>, <операнд2>, <метка>
```

выполняет сравнение операндов и осуществляет переход по метке, если они не равны. Если операнды равны, выполняется следующая за командой CJNE инструкция. Команда сравнения вычитает операнд2 из операнда1 и в зависимости от результата изменяет флаги, не сохраняя результат вычитания.

Следующие команды условного перехода проверяют на равенство нулю указанного в команде операнда:

```
DJNZ <операнд>, <метка>
JB <бит>, <метка>
JNB <бит>, <метка>
JBC <бит>, <метка>
```

при этом команда DJNZ перед сравнением выполняет декремент операнда, а команда JBC сбрасывает битовый операнд после сравнения.

Некоторые команды условного перехода подразумевают неявное задание операнда. Так, команды:

```
JZ <метка>
JNZ <метка>
```

проверяют на равенство нулю аккумулятор. А команды:

```
JC <метка>
JNC <метка>
```

проверяют состояние флага переноса С.

Для обращения к подпрограммам необходимо использовать команды вызова подпрограмм. Эти команды сохраняют в стеке адрес возврата в основную программу. Для возврата из подпрограммы необходимо выполнить команду RET (RETI для обработчика прерываний). Синтаксис команд следующий:

```
LCALL <адрес>
ACALL <адрес>
RET
RETI
```

1.3 Особенности обработки многоразрядных операндов

Поскольку микроконтроллеры семейства MCS-51 являются восьмиразрядными, разрядность обрабатываемых либо анализируемых данных не превышает байтового значения. Для обработки многоразрядных операндов, значения которых превышают 255, необходима разработка специальных алгоритмов расширения разрядности арифметических операций. Так, алгоритм сложения многоразрядных чисел можно реализовать путем циклического побайтного сложения, начиная с младшего байта, с учетом переноса от сложения предыдущих байт. Умножения легко реализовать циклическим сложением многоразрядных операндов, а деление – вычитанием.

На базе вычитания многоразрядных чисел с последующим анализом флагов состояния реализуется алгоритм сравнения. Необходимо лишь предварительно определить в битовой области памяти данных соответствующие битовые переменные, например:

```
BitL   bit   20h
BitE   bit   21h
BitG   bit   22h
```

2 Порядок выполнения работы

2.1 Изучить теоретические сведения, представленные в п1.

2.2 Разработать подпрограммы для выполнения следующих операций:

Суммирование двухбайтовых чисел ADD2_2:

R1, R0 - слагаемое;
R5, R4 - слагаемое;
R5, R4 - сумма.

Суммирование трехбайтовых чисел ADD3_3:

R2, R1, R0 - слагаемое;
R6, R5, R4 - слагаемое;
R6, R5, R4 - сумма.

Суммирование четырёхбайтовых чисел ADD4_4:

R3, R2, R1, R0 - слагаемое;
R7, R6, R5, R4 - слагаемое;
R7, R6, R5, R4 - сумма.

2.3 Разработать подпрограммы для выполнения следующих операций:

Вычитание двухбайтовых чисел SUB2_2:

R5, R4 - уменьшаемое;
R1, R0 - вычитаемое;
R5, R4 - разность.

Вычитание трехбайтовых чисел SUB3_3:

R6, R5, R4 - уменьшаемое;
R2, R1, R0 - вычитаемое;
R6, R5, R4 - разность.

Вычитание четырехбайтовых чисел SUB4_4:

R7, R6, R5, R4 - уменьшаемое;
R3, R2, R1, R0 - вычитаемое;
R7, R6, R5, R4 - разность.

2.4 Разработать подпрограммы для выполнения следующих операций:

Умножение двухбайтовых чисел MUL2_2:

R1, R0	- множитель;
R3, R2	- множитель;
R7, R6, R5, R4	- произведение.

Деление двухбайтовых чисел DIV2_1:

R1, R0	- делимое;
R4	- делитель;
R3, R2	- частное;
R4	- остаток.

2.5 Разработать подпрограммы для выполнения следующих операций:

Сравнение двухбайтовых чисел CMP2_2:

R5, R4	- число 1;
R1, R0	- число 2.

Сравнение трехбайтовых чисел CMP3_3:

R6, R5, R4	- число 1;
R2, R1, R0	- число 2.

Сравнение четырехбайтовых чисел CMP4_4:

R7, R6, R5, R4	- число 1;
R3, R2, R1, R0	- число 2.

Оформить как одну подпрограмму с различными точками входа. Результаты сравнения оформить в битах:

BitL = 1, если “число 1” < “число 2”;

BitE = 1, если “число 1” = “число 2”;

BitG = 1, если “число 1” > “число 2”.

3 Содержание отчета

Отчет о проделанной работе оформляется в соответствии с общими требованиями и должен содержать:

3.1 Цель работы.

3.2 Текст программы с реализованными в п.п. 2.2 – 2.5 подпрограммами.

3.3 Выводы по результатам проделанной работы.

4 Контрольные вопросы

4.1 Синтаксис арифметических команд сложения и вычитания для микроконтроллеров семейства MCS-51 и режимы адресации их операндов.

4.2 Синтаксис арифметических команд умножения и деления для микроконтроллеров семейства MCS-51 и режимы адресации их операндов.

4.3 Синтаксис арифметических команд сложения и вычитания для микроконтроллеров семейства MCS-51 и режимы адресации их операндов.

4.4 Синтаксис команд безусловных переходов и команд вызова подпрограмм для микроконтроллеров семейства MCS-51 и режимы адресации их операндов.

4.5 Синтаксис команд условных переходов для микроконтроллеров семейства MCS-51 и режимы адресации их операндов.

5 Рекомендуемая литература

5.1 Предко, М. Руководство по микроконтроллерам: в 2-х т. – М.: Постмаркет, 2001.

5.2 Каспер, Э. Программирование на языке ассемблера для микроконтроллеров семейства i8051. – М.: Горячая линия – Телеком, 2003.

СИСТЕМА КОМАНД MCS-51

Группа команд передачи данных

Название команды	Мнемокод	КОП	Т	Б	Ц	Операция
Пересылка в аккумулятор из регистра (n = 0 - 7)	MOV A, Rn	11101rrr	1	1	1	(A) = (Rn)
Пересылка в аккумулятор прямоадресуемого байта	MOV A, ad	11100101	3	2	1	(A) = (ad)
Пересылка в аккумулятор байта из РДП (i = 0, 1)	MOV A, @Ri	1110011i	1	1	1	(A) = ((Ri))
Загрузка в аккумулятор константы	MOV A, #d	01110100	2	2	1	(A) = #d
Пересылка в регистр из аккумулятора	MOV Rn, A	11111rrr	1	1	1	(Rn) = (A)
Пересылка в регистр прямоадресуемого байта	MOV Rn, ad	10101rrr	3	2	2	(Rn) = (ad)
Загрузка в регистр константы	MOV Rn, #d	01111rrr	2	2	1	(Rn) = #d
Пересылка по прямому адресу аккумулятора	MOV ad, A	11110101	3	2	1	(ad) = (A)
Пересылка по прямому адресу регистра	MOV ad, Rn	10001rrr	3	2	2	(ad) = (Rn)
Пересылка прямоадресуемого байта по прямому адресу	MOV add, ads	10000101	9	3	2	(add) = (ads)
Пересылка байта из РДП по прямому адресу	MOV ad, @Ri	1000011i	3	2	2	(ad) = ((Ri))
Пересылка по прямому адресу константы	MOV ad, #d	01110101	7	3	2	(ad) = #d
Пересылка в РДП из аккумулятора	MOV @Ri, A	1111011i	1	1	1	((Ri)) = (A)
Пересылка в РДП прямоадресуемого байта	MOV @Ri, ad	0110011i	3	2	2	((Ri)) = (ad)
Пересылка в РДП константы	MOV @Ri, #d	0111011i	2	2	1	((Ri)) = #d
Загрузка указателя данных	MOV DPTR, #d16	10010000	13	3	2	(DPTR) = #d16
Пересылка в аккумулятор байта из ПП	MOVC A, @A + DPTR	10010011	1	1	2	(A) = ((A) + (DPTR))
Пересылка в аккумулятор байта из ПП	MOVC A, @A + PC	10000011	1	1	2	(PC) = (PC) + 1 (A) = ((A) + (PC))
Пересылка в аккумулятор байта из ВПД	MOVX A, @Ri	1110001i	1	1	2	(A) = ((Ri))
Пересылка в аккумулятор байта из расширенной ВПД	MOVX A, @DPTR	11100000	1	1	2	(A) = ((DPTR))
Пересылка в ВПД из аккумулятора	MOVX @Ri, A	1111001i	1	1	2	((Ri)) = (A)
Пересылка в расширенную ВПД из аккумулятора	MOVX @DPTR, A	11110000	1	1	2	((DPTR)) = (A)
Загрузка в стек	PUSH ad	11000000	3	2	2	(SP) = (SP) + 1 ((SP)) = (ad)

Продолжение таблицы

Название команды	Мнемокод	КОП	Т	Б	Ц	Операция
Извлечение из стека	POP ad	11010000	3	2	2	(ad) = (SP) (SP) = (SP) - 1
Обмен аккумулятора с регистром	XCH A, Rn	11001rrr	1	1	1	(A) <-> (Rn)
Обмен аккумулятора с прямоадресуемым байтом	XCH A, ad	11000101	3	2	1	(A) <-> (ad)
Обмен аккумулятора с байтом из РДП	XCH A, @Ri	1100011i	1	1	1	(A) <-> ((Ri))
Обмен младшей тетрады аккумулятора с младшей тетрадой байта РДП	XCHD A, @Ri	1101011i	1	1	1	(A_{0-3}) <-> $((Ri)_{0-3})$

Группа команд арифметических операций

Название команды	Мнемокод	КОП	Т	Б	Ц	Операция
Сложение аккумулятора с регистром (n = 0 - 7)	ADD A, Rn	00101rrr	1	1	1	(A) = (A) + (Rn)
Сложение аккумулятора с прямоадресуемым байтом	ADD A, ad	00100101	3	2	1	(A) = (A) + (ad)
Сложение аккумулятора с байтом из РПД (i = 0, 1)	ADD A, @Ri	0010011i	1	1	1	(A) = (A) + ((Ri))
Сложение аккумулятора с константой	ADD A, #d	00100100	2	2	1	(A) = (A) + #d
Сложение аккумулятора с регистром и переносом	ADDC A, Rn	00111rrr	1	1	1	(A) = (A) + (Rn) + (C)
Сложение аккумулятора с прямоадресуемым байтом и переносом	ADDC A, ad	00110101	3	2	1	(A) = (A) + (ad) + (C)
Сложение аккумулятора с байтом из РПД и переносом	ADDC A, @Ri	0011011i	1	1	1	(A) = (A) + ((Ri)) + (C)
Сложение аккумулятора с константой и переносом	ADDC A, #d	00110100	2	2	1	(A) = (A) + #d + (C)
Десятичная коррекция аккумулятора	DA A	11010100	1	1	1	Если $(A_{0-3}) > 9 \vee ((AC) = 1)$, то $(A_{0-3}) = (A_{0-3}) + 6$, затем если $(A_{4-7}) > 9 \vee ((C) = 1)$, то $(A_{4-7}) = (A_{4-7}) + 6$
Вычитание из аккумулятора регистра и заема	SUBB A, Rn	10011rrr	1	1	1	(A) = (A) - (C) - (Rn)
Вычитание из аккумулятора прямоадресуемого байта и заема	SUBB A, ad	10010101	3	2	1	(A) = (A) - (C) - ((ad))
Вычитание из аккумулятора байта РПД и заема	SUBB A, @Ri	1001011i	1	1	1	(A) = (A) - (C) - ((Ri))
Вычитание из аккумулятора константы и заема	SUBB A, #d	10010100	2	2	1	(A) = (A) - (C) - #d

Продолжение таблицы

Название команды	Мнемокод	КОП	Т	Б	Ц	Операция
Инкремент аккумулятора	INC A	00000100	1	1	1	(A) = (A) + 1
Инкремент регистра	INC Rn	00001rrr	1	1	1	(Rn) = (Rn) + 1
Инкремент прямоадресуемого байта	INC ad	00000101	3	2	1	(ad) = (ad) + 1
Инкремент байта в РПД	INC @Ri	0000011i	1	1	1	((Ri)) = ((Ri)) +1
Инкремент указателя данных	INC DPTR	10100011	1	1	2	(DPTR) = (DPTR) + 1
Декремент аккумулятора	DEC A	00010100	1	1	1	(A) = (A) - 1
Декремент регистра	DEC Rn	00011rrr	1	1	1	(Rn) = (Rn) - 1
Декремент прямоадресуемого байта	DEC ad	00010101	3	2	1	(ad) = (ad) - 1
Декремент байта в РПД	DEC @Ri	0001011i	1	1	1	((Ri)) = ((Ri)) -1
Умножение аккумулятора на регистр В	MUL AB	10100100	1	1	4	(B)(A) = (A)*(B)
Деление аккумулятора на регистр В	DIV AB	10000100	1	1	4	(A).(B) = (A)/(B)

Группа команд логических операций

Название команды	Мнемокод	КОП	Т	Б	Ц	Операция
Логическое И аккумулятора и регистра	ANL A, Rn	01011rrr	1	1	1	(A) = (A) ∧ (Rn)
Логическое И аккумулятора и прямоадресуемого байта	ANL A, ad	01010101	3	2	1	(A) = (A) ∧ (ad)
Логическое И аккумулятора и байта из РПД	ANL A, @Ri	0101011i	1	1	1	(A) = (A) ∧ ((Ri))
Логическое И аккумулятора и константы	ANL A, #d	01010100	2	2	1	(A) = (A) ∧ #d
Логическое И прямоадресуемого байта и аккумулятора	ANL ad, A	01010010	3	2	1	(ad) = (ad) ∧ (A)
Логическое И прямоадресуемого байта и константы	ANL ad, #d	01010011	7	3	2	(ad) = (ad) ∧ #d
Логическое ИЛИ аккумулятора и регистра	ORL A, Rn	01001rrr	1	1	1	(A) = (A) ∨ (Rn)
Логическое ИЛИ аккумулятора и прямоадресуемого байта	ORL A, ad	01000101	3	2	1	(A) = (A) ∨ (ad)
Логическое ИЛИ аккумулятора и байта из РПД	ORL A, @Ri	0100011i	1	1	1	(A) = (A) ∨ ((Ri))
Логическое ИЛИ аккумулятора и константы	ORL A, #d	01000100	2	2	1	(A) = (A) ∨ #d
Логическое ИЛИ прямоадресуемого байта и аккумулятора	ORL ad, A	01000010	3	2	1	(ad) = (ad) ∨ (A)
Логическое ИЛИ прямоадресуемого байта и константы	ORL ad, #d	01000011	7	3	2	(ad) = (ad) ∨ #d
Исключающее ИЛИ аккумулятора и регистра	XRL A, Rn	01101rrr	1	1	1	(A) = (A) ⊻ (Rn)
Исключающее ИЛИ аккумулятора и прямоадресуемого байта	XRL A, ad	01100101	3	2	1	(A) = (A) ⊻ (ad)

Продолжение таблицы

Название команды	Мнемокод	КОП	Т	Б	Ц	Операция
Исключающее ИЛИ аккумулятора и байта из РПД	XRL A, @Ri	0110011i	1	1	1	(A) = (A) ⊻ ((Ri))
Исключающее ИЛИ аккумулятора и константы	XRL A, #d	01100100	2	2	1	(A) = (A) ⊻ #d
Исключающее ИЛИ прямоадресуемого байта и аккумулятора	XRL ad, A	01100010	3	2	1	(ad) = (ad) ⊻ (A)
Исключающее ИЛИ прямоадресуемого байта и константы	XRL ad, #d	01100011	7	3	2	(ad) = (ad) ⊻ #d
Сброс аккумулятора	CLR A	11100100	1	1	1	(A) = 0
Инверсия аккумулятора	CPL A	11110100	1	1	1	(A) = (¯A)
Сдвиг аккумулятора влево циклически	RL A	00100011	1	1	1	$(A_{n+1}) = (A_n)$, n = 0 ? 6, $(A_0) = (A_7)$
Сдвиг аккумулятора влево через перенос	RLC A	00110011	1	1	1	$(A_{n+1}) = (A_n)$, n = 0 ? 6, $(A_0) = (C)$, $(C) = (A_7)$
Сдвиг аккумулятора вправо циклически	RR A	00000011	1	1	1	$(A_n) = (A_{n+1})$, n = 0 ? 6, $(A_7) = (A_0)$
Сдвиг аккумулятора вправо через перенос	RRC A	00010011	1	1	1	$(A_n) = (A_{n+1})$, n = 0 ? 6, $(A_7) = (C)$, $(C) = (A_0)$
Обмен местами тетрад в аккумуляторе	SWAP A	11000100	1	1	1	(A_{0-3}) <-> (A_{4-7})

Группа команд операции с битами

Название команды	Мнемокод	КОП	Т	Б	Ц	Операция
Сброс переноса	CLR C	11000011	1	1	1	(C) = 0
Сброс бита	CLR bit	11000010	4	2	1	(b) = 0
Установка переноса	SETB C	11010011	1	1	1	(C) = 1
Установка бита	SETB bit	11010010	4	2	1	(b) = 1
Инверсия переноса	CPL C	10110011	1	1	1	(C) = (¯C)
Инверсия бита	CPL bit	10110010	4	2	1	(b) = (¯b)
Логическое И бита и переноса	ANL C, bit	10000010	4	2	2	(C) = (C) ∧ (b)
Логическое И инверсии бита и переноса	ANL C, /bit	10110000	4	2	2	(C) = (C) ∧ (¯b)
Логическое ИЛИ бита и переноса	ORL C, bit	01110010	4	2	2	(C) = (C) ∨ (b)
Логическое ИЛИ инверсии бита и переноса	ORL C, /bit	10100000	4	2	2	(C) = (C) ∨ (¯b)
Пересылка бита в перенос	MOV C, bit	10100010	4	2	1	(C) = (b)
Пересылка переноса в бит	MOV bit, C	10010010	4	2	2	(b) = (C)

Группа команд передачи управления

Название команды	Мнемокод	КОП	Т	Б	Ц	Операция
Длинный переход в полном объеме памяти программ	LJMP ad16	00000010	12	3	2	(PC) = ad16
Абсолютный переход внутри страницы в 2 Кбайта	AJMP ad11	$a_{10}a_9a_8$00001	6	2	2	(PC) = (PC) + 2 (PC_{0-10}) = ad11
Короткий относительный переход внутри страницы в 256 байт	SJMP rel	10000000	5	2	2	(PC) = (PC) + 2 (PC) = (PC) + rel
Косвенный относительный переход	JMP @A+DPTR	01110011	1	1	2	(PC) = (A) + (DPTR)
Переход, если аккумулятор равен нулю	JZ rel	01100000	5	2	2	(PC) = (PC) + 2, если (A) = 0, то (PC) = (PC) + rel
Переход, если аккумулятор не равен нулю	JNZ rel	01110000	5	2	2	(PC) = (PC) + 2, если (A) ? 0, то (PC) = (PC) + rel
Переход, если перенос равен единице	JC rel	01000000	5	2	2	(PC) = (PC) + 2, если (C) = 1, то (PC) = (PC) + rel
Переход, если перенос равен нулю	JNC rel	01010000	5	2	2	(PC) = (PC) + 2, если (C) = 0, то (PC) = (PC) + rel
Переход, если бит равен единице	JB bit, rel	00100000	11	3	2	(PC) = (PC) + 3, если (b) = 1, то (PC) = (PC) + rel
Переход, если бит равен нулю	JNB bit, rel	00110000	11	3	2	(PC) = (PC) + 3, если (b) = 0, то (PC) = (PC) + rel
Переход, если бит установлен, с последующим сбросом бита	JBC bit, rel	00010000	11	3	2	(PC) = (PC) + 3, если (b) = 1, то (b) = 0 и (PC) = (PC) + rel
Декремент регистра и переход, если не нуль	DJNZ Rn, rel	11011rrr	5	2	2	(PC) = (PC) + 2, (Rn) = (Rn) -1, если (Rn) ? 0, то (PC) = (PC) + rel
Декремент прямоадресуемого байта и переход, если не нуль	DJNZ ad, rel	11010101	8	3	2	(PC) = (PC) + 2, (ad) = (ad) -1, если (ad) ? 0, то (PC) = (PC) + rel
Сравнение аккумулятора с прямоадресуемым байтом и переход, если не равно	CJNE A, ad, rel	10110101	8	3	2	(PC) = (PC) + 3, если (A) ? (ad), то (PC) = (PC) + rel, если (A) < (ad), то (C) = 1, иначе (C) = 0
Сравнение аккумулятора с константой и переход, если не равно	CJNE A, #d, rel	10110100	10	3	2	(PC) = (PC) + 3, если (A) ? #d, то (PC) = (PC) + rel, если (A) < #d, то (C) = 1, иначе (C) = 0

Продолжение таблицы

Сравнение регистра с константой и переход, если не равно	CJNE Rn, #d, rel	10111rrr	10	3	2	(PC) = (PC) + 3, если (Rn) ? #d, то (PC) = (PC) + rel, если (Rn) < #d, то (C) = 1, иначе (C) = 0
Сравнение байта в РПД с константой и переход, если не равно	CJNE @Ri, #d, rel	1011011i	10	3	2	(PC) = (PC) + 3, если ((Ri)) ? #d, то (PC) = (PC) + rel, если ((Ri)) < #d, то (C) = 1, иначе (C) = 0
Длинный вызов подпрограммы	LCALL ad16	00010010	12	3	2	(PC) = (PC) + 3, (SP) = (SP) + 1, ((SP)) = (PC_{0-7}), (SP) = (SP) + 1, ((SP)) = (PC_{8-15}), (PC) = ad16
Абсолютный вызов подпрограммы в пределах страницы в 2 Кбайта	ACALL ad11	$a_{10}a_9a_8$10001	6	2	2	(PC) = (PC) + 2, (SP) = (SP) + 1, ((SP)) = (PC_{0-7}), (SP) = (SP) + 1, ((SP)) = (PC_{8-15}), (PC_{0-10}) = ad11
Возврат из подпрограммы	RET	00100010	1	1	2	(PC_{8-15}) = ((SP)), (SP) = (SP) - 1, (PC_{0-7}) = ((SP)), (SP) = (SP) - 1
Возврат из подпрограммы обработки прерывания	RETI	00110010	1	1	2	(PC_{8-15}) = ((SP)), (SP) = (SP) - 1, (PC_{0-7}) = ((SP)), (SP) = (SP) - 1
Холостая команда	NOP	00000000	1	1	1	(PC) = (PC) + 1

Приняты следующие обозначения:

A - аккумулятор

R_i – регистр выбранного банка

i – номер регистра

direct – прямо адресуемый 8-битовый внутренний адрес

@ R_i – косвенно адресуемая 8-битовая ячейка ОЗУ

d8, #d – 8-битовое непосредственное данное

d16 – 16-битовое непосредственное данное

dH, dL – старшие, младшие биты непосредственных 16-битных данных

adr11 – 11-битовый адрес

adr16 – 16-битовый адрес

adrL – младшие биты адреса

disp8, rel – 8-битовый байт смещения

bit – прямо адресуемый бит

(x) – содержимое элемента x

((x)) – содержимое по адресу, хранящемуся в x

(x)[M] – разряд M элемента x

/x – инверсия

ads – адрес источника данных

add – адрес приемника данных

ad16h – старший байт адреса

ad16l – младший байт адреса

#d16h – старший байт данных

ad – 8-разрядный адрес

#d16l – младший байт данных

Учебное издание

Составители:
Разумейчик Вита Станиславовна
Журавский Владимир Иванович
Волков Евгений Геннадьевич
Дереченник Анна Станиславовна

ПРОГРАММИРОВАНИЕ ОДНОКРИСТАЛЬНЫХ МИКРОЭВМ

Лабораторный практикум по учебным дисциплинам

«Проектирование контроллеров и специализированных вычислительных систем» и «Микроконтроллерные устройства» для студентов специальностей 1-40 02 01 «Вычислительные машины, системы и сети» и 1-36 04 02 «Промышленная электроника»

Часть 1

Ответственный за выпуск: Разумейчик В.С.
Редактор: Боровикова Е.А.
Компьютерная вёрстка: Соколюк А.П.
Корректор: Никитчик Е.В.

Подписано к печати 06.02.2014 г. Формат 60x84 $^{1}/_{16}$. Гарнитура Times New Roman. Бумага «Снегурочка». Усл. п. л. 2,1. Уч. изд. 2,25. Заказ № 1323. Тираж 50 экз. Отпечатано на ризографе учреждения образования «Брестский государственный технический университет». 224017, г. Брест, ул. Московская, 267.