Ю. И. ИВАНОВ, В. Я. ЮГАЙ

МИКРОПРОЦЕССОРНЫЕ УСТРОЙСТВА СИСТЕМ УПРАВЛЕНИЯ

Министерство образования и науки
Российской Федерации

Федеральное агентство по образованию

Государственное образовательное учреждение
высшего профессионального образования

Таганрогский государственный радиотехнический
университет

Ю.И.ИВАНОВ
В.Я.ЮГАЙ

МИКРОПРОЦЕССОРНЫЕ УСТРОЙСТВА СИСТЕМ УПРАВЛЕНИЯ

Допущено Учебно-методическим объединением вузов по образованию в области автоматизированного машиностроения (УМО АМ) в качестве учебного пособия для студентов высших учебных заведений, обучающихся по направлению подготовки дипломированных специалистов – "Автоматизированные технологии и производства" (специальность 210200 – " Автоматизация технологических процессов и производств (в энергетике)

Таганрог 2005

УДК 681.325.5–181.48:681.51(075.8)+681.51:681.325.5–181.48(075.8)

Ю.И.Иванов, В.Я.Югай. Микропроцессорные устройства систем управления: Учебное пособие. – Таганрог: Изд-во ТРТУ, 2005. – 133 с.

Учебное пособие предназначено для студентов, обучающихся по направлению 657900 «Автоматизированные технологии и производства», и содержит сведения, необходимые при изучении курсов «Микропроцессорная техника в системах управления», «Электронные устройства автоматики», «Технические средства автоматизации», «Технологические процессы и производства». В учебном пособии рассмотрены вопросы организации работы микропроцессорных средств, функциональные возможности и характеристики аппаратных средств микроконтроллеров, особенности программирования на ассемблере, приведены примеры программ для выполнения типовых функций, даны краткие рекомендации по технической реализации алгоритмов управления.

Ил.12.

Печатается по решению редакционно-издательского совета Таганрогского государственного радиотехнического университета.

Рецензенты:

А.Н.Целых – д-р техн. наук, профессор, директор регионального (областного) центра новых информационных технологий, проректор по информатике ТРТУ;

Я.Е.Ромм – д-р техн. наук, профессор, зав. кафедрой информатики ТГПИ.

ISBN 5-8327-0206-9

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Распространение новых информационных технологий, основанных на достижениях микроэлектроники последней четверти XX века, привело к значительным изменениям в самых различных областях. Традиционно рассматривается прогресс в компьютерной технике, системах компьютерной обработки информации, компьютерных сетях и т.п. Однако не менее революционные изменения произошли и в других сферах, связанных с применением технических средств обработки данных. Появление новых средств и технологий обработки данных позволило существенно расширить функциональные возможности и сложность решаемых задач в системах автоматизации.

Системы автоматического управления могут эффективно решать задачи на уровнях, начинающихся от управления отдельными узлами и устройствами и заканчивающихся управлением технологическими установками и целыми производствами. В системах автоматизации применяются различные средства реализации алгоритмов управления. В настоящее время, несмотря на определенные особенности применения, наиболее эффективными являются цифровые методы и средства.

Современная микроэлектронная элементная база позволяет при невысоких затратах на аппаратные средства использовать разнообразные алгоритмы цифровой обработки данных, а их преимущества общеизвестны. Вес и роль аналоговых средств существенно снизилась в силу недостаточной точности, стабильности, функциональной гибкости и технологичности. Основная область применения аналоговых устройств – предварительная подготовка сигналов для преобразования в цифровой формат.

Одним из важнейших факторов прогресса в средствах автоматизации является "интеллектуализация" устройств, включая и устройства, выполняющие наиболее простые функции: измерительные датчики, исполнительные устройства, средства сигнализации и т.п. Кроме необходимых основных функций, "интеллектуальные" технические средства могут реализовать множество вспомогательных, зачастую весьма сложных алгоритмов преобразования данных при относительно невысоких дополнительных затратах.

Эта функциональная избыточность позволяет использовать одни и те же технические средства при решении разнообразных задач, несмотря на различие требований, реализуемых алгоритмов и функций. Очень часто выбор определенных параметров, режимов и алгоритмов работы должен программироваться, т.е. определяться специальными процедурами настройки. Поэтому современные технические средства должны обладать соответствующей функциональной гибкостью, возможностью изменения параметров и режимов работы, поддерживать необходимые процедуры настройки.

Типовая структура системы управления приведена на рис. 1. Система управления может быть независимой локальной системой либо интегрирована как элемент в многоуровневую систему. Функции элементов системы управления можно определить следующим образом:

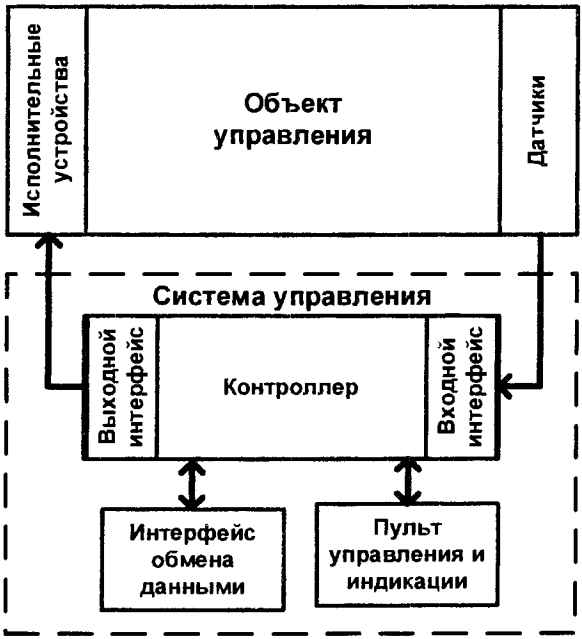

Рис. 1. Структурная схема системы управления

- входной интерфейс – прием и преобразование сигналов от датчиков в формат, удобный для дальнейшей обработки;
- выходной интерфейс – преобразование выходных данных системы в сигналы управления исполнительными устройствами;
- контроллер – основной элемент системы управления, реализующий алгоритмы управления и обработки данных в соответствии с поставленными задачами, потоком данных входного интерфейса, управляющих команд пульта управления и интерфейса обмена с другими средствами управления;
- пульт управления и индикации – средства управления и отображения данных о параметрах и режимах работы для оператора объекта управления;

- интерфейс обмена данными – средства организации взаимодействия и координации работы системы управления с другими средствами управления, в локальных системах управления необязателен, но, как правило, должен предусматриваться для возможной модернизации системы управления.

Контроллер – основное средство реализации алгоритмов управления, он должен преобразовывать поток цифровых входных данных и команд $\boldsymbol{X}$ в поток управляющих выходных данных $\boldsymbol{Y}$:

$$\mathbf{Y} = \Psi(\mathbf{X}) \quad .$$

Алгоритмы преобразования Ψ можно реализовать двумя основными способами. Могут применяться и различные комбинации этих способов.

Первый способ принято называть аппаратной реализацией: все необходимые функции преобразования выполняются специально разработанным устройством с определенным набором компонентов и их функциональных связей. Аппаратная реализация при прочих равных условиях минимальна с точки зрения требуемого числа компонентов, обладает максимально возможным быстродействием. Однако проектирование устройств управления с аппаратной реализацией требует очень высоких затрат. Кроме того, даже незначительное изменение алгоритмов преобразования требует повторной разработки нового устройства. В силу этих недостатков чисто аппаратная реализация в настоящее время применяется крайне редко.

Второй основной способ – программная реализация. Обычно даже сложные и самые разнообразные алгоритмы преобразования Ψ можно представить в виде конечных последовательностей относительно несложных операций (команд). Классический пример программной реализации – компьютеры, они и послужили прототипом микропроцессоров – основных технических средств этой реализации. Один и тот же набор аппаратных средств микропроцессорной системы позволяет решать самые разнородные задачи, алгоритм преобразования определяется программой, записанной в запоминающем устройстве.

Главным достоинством программной реализации является функциональная гибкость: для изменения алгоритмов или выполняемых функций необходимо модифицировать только последовательность команд в программе. Значительная функциональная избыточность микропроцессорных средств, как правило, не приводит к существенному увеличению сложности и стоимости. Все же такая реализация обладает и существенным недостатком: последовательное, а не параллельное, как при аппаратной реализации, выполнение необходимых операций требует значительных временных затрат, а это приводит к снижению быстродействия.

В технике обычно применяют компромиссные решения, которые в той или иной степени объединяют достоинства разных способов реализации. Программируемые логические интегральные схемы (ПЛИС) и микрокон-

троллеры – наиболее известные результаты таких компромиссов. И те, и другие средства реализации являются продуктом достижений микроэлектронных технологий и представляют большие интегральные схемы со значительными функциональными возможностями, необходимой универсальностью и гибкостью. В определенном смысле в них используются противоположные подходы, поэтому они обладают разными свойствами и разными основными областями применений.

В основе ПЛИС лежит аппаратная реализация. ПЛИС определенного типа содержит ограниченный набор типовых элементов (комбинационные логические схемы, триггеры, счетчики, регистры и т.п.). Для достижения необходимой гибкости функциональные связи между элементами не задаются жестко, а могут изменяться в довольно широких пределах с помощью процедур, аналогичных записи данных в программируемые постоянные запоминающие устройства (PROM).

Процедуры создания функциональных связей принято называть программированием. Для реализации заданного алгоритма работы необходимо выбрать тип ПЛИС с необходимым набором элементов и реализуемых функций, разработать требуемую структуру функциональных связей и далее произвести программирование этих связей. Как правило, ПЛИС допускает многократное программирование, т.е. при неудовлетворительных полученных результатах устройство можно перепрограммировать, выполнив требуемую корректировку. Естественно, что все эти операции выполняются с помощью специальных средств. Эти средства обычно включают инструментальные пакеты прикладных программ для персональных компьютеров, в которых реализуемые функции описываются с помощью формализованных процедур. Подготовленные файлы для реализации требуемых функций ПЛИС записываются через стандартные программаторы непосредственно из персонального компьютера. Таким образом, возможность программирования функций приближает ПЛИС по функциональной гибкости к программной реализации, а сама реализация функций близка к аппаратной. Обычные области применения ПЛИС – задачи, требующие высокого быстродействия, но реализующие один и тот же повторяющийся алгоритм преобразования, который не может существенно изменяться в процессе работы.

Микроконтроллеры в силу программной реализации основных функций обладают более высокой гибкостью и универсальностью, чем ПЛИС. Для повышения быстродействия и эффективности работы в микроконтроллеры дополнительно вводят средства аппаратной реализации типовых функций, что позволяет, наряду с последовательной программной реализацией необходимых алгоритмов, использовать и параллельную аппаратную реализацию. Как правило, аппаратные средства выполняют стандартные интерфейсные функции. Тогда алгоритмы преобразования, выполняемые микро-

контроллерами, можно представить в виде двух параллельно реализуемых составляющих

$$\Psi = \Psi_{SW} + \Psi_{HW},$$

где Ψ_{SW} – программная (последовательная) реализация, обеспечивающая универсальность и гибкость;

Ψ_{HW} – аппаратная (параллельная) реализация типовых функций стандартных периферийных устройств.

Такое сочетание позволяет сохранить универсальность программной реализации, повысить быстродействие, реализуя большинство интерфейсных функций аппаратно (Ψ_{HW}). Программа обработки данных (Ψ_{SW}) упрощается, она не содержит функций, реализуемых аппаратно, но должна обеспечивать корректное взаимодействие с аппаратно реализуемыми интерфейсами. Микроконтроллеры применяют для решения самых разнообразных задач, набор средств аппаратной реализации может быть различным, поэтому существует несколько классов микроконтроллеров с разными возможностями. Микроконтроллеры выпускаются многими фирмами-производителями интегральных схем и являются массовыми, относительно недорогими и доступными изделиями.

Интегрируя на одном кристалле высокопроизводительный процессор, память и стандартные периферийные устройства, микроконтроллеры позволяют с минимальными затратами создавать системы управления различными объектами и процессами. В настоящее время микроконтроллеры являются наиболее универсальными и распространенными компонентами технических средств автоматизации.

Особенности применения микроконтроллеров в системах автоматического управления в первую очередь определяются средствами программной реализации. Основой практически любой программной реализации является стандартная структура микропроцессорной системы. Рассмотрим свойства такой системы на примере классического микропроцессора Intel 8080. Несмотря на значительные изменения в микропроцессорной технике, произошедшие в течение 30 лет после его появления, свойства Intel 8080 отражают основные особенности средств программной реализации алгоритмов преобразования данных.

1.ОРГАНИЗАЦИЯ МИКРОПРОЦЕССОРНОЙ СИСТЕМЫ

Появление микропроцессорных устройств в начале 70-х годов XX века было результатом работы по созданию универсальных средств для реализации цифровых алгоритмов преобразования данных. Как известно, в основу микропроцессорной системы была положена классическая архитектура электронно-вычислительной машины – компьютера. Принципы программной реализации цифровых алгоритмов и организация работы компьютера позволяют использовать одни и те же аппаратные средства для решения весьма разнообразных задач, обеспечивая необходимую универсальность и многофункциональность. Многие архитектурные решения, отработанные в компьютерах, были использованы при создании интегральных схем – микропроцессоров, они содержат все основные компоненты универсального компьютера и используют аналогичные принципы организации работы.

1.1. Структура микропроцессорной системы

В микропроцессорной системе, как уже указывалось, используются принципы организации работы и особенности архитектуры классических компьютерных средств. Любой алгоритм преобразования данных должен быть описан в форме программы – конечной последовательности элементарных операций. Центральный элемент такой системы – микропроцессор, который выполняет операции преобразования данных, производит ввод и вывод всех необходимых данных, в том числе и кодов самой программы, и управляет взаимодействием всех элементов системы.

Набор операций (выполняемых команд) микропроцессора должен обладать функциональной полнотой, т.е. обеспечивать выполнение необходимого множества операций преобразования данных, управления, обмена данных. Так как преобразуемые данные и коды программы могут быть представлены в достаточно близких форматах, как правило, строгого разграничения между ними не делают. Такой подход расширяет возможности программной обработки: данные могут использоваться как элементы кодов программы, а элементы кодов программы можно преобразовывать в процессе работы.

Для хранения данных и кодов программы микропроцессорная система содержит запоминающее устройство. Стандартная организация микропроцессорной системы предполагает доступность для микропроцессора всех данных в запоминающем устройстве. Так как объем запоминающего устройства достаточно большой, доступ к выбранным данным обычно производится через средства адресации. Каждому элементу сохраняемых данных (например, каждому байту) соответствует определенный код хранения, называемый адресом в запоминающем устройстве. Как правило, в микропроцессорной системе предполагается постоянная готовность запоминающего устройства к обмену данными, для обмена данными микропроцессор дол-

жен передать определенный код адреса данных и соответствующие сигналы управления.

Процедуры ввода-вывода данных также должны выполняться стандартными для микропроцессора алгоритмами. Эти алгоритмы, эффективные для управления микропроцессорной системы, могут не соответствовать особенностям работы различных периферийных устройств. Поэтому микропроцессорная система обычно содержит устройства ввода-вывода, обеспечивающие согласование алгоритмов управления обменом данными.

Еще один важный компонент микропроцессорной системы – топология линий связи для передачи сигналов между микропроцессором и другими устройствами. Топология должна быть такой, чтобы изменение структуры микропроцессорной системы не требовало изменения структуры используемых линий связи и формируемых сигналов. Этому требованию удовлетворяет магистрально-модульный (шинный) принцип организации взаимодействия. При шинной организации все устройства подключаются параллельно к используемым линиям связи. Линии связи можно условно объединить в три группы – шины. По шине данных (ШД) передаются требуемые данные, шина адреса (ША) служит для выбора устройств, участвующих в обмене данными, а шина управления (ШУ) необходима для передачи управляющих сигналов.

Шинная организация максимально универсальна: подключение дополнительных устройств абсолютно не затрагивает уже действующую структуру микропроцессорной системы. Конечно, такая организация накладывает и ограничения на обмен данными. Во-первых, управление обменом данными должно производиться только одним устройством, иначе могут возникнуть конфликты между управляющими устройствами, приводящие к некорректной работе шины. Во-вторых, обязательно требуется адресация устройств, параллельное подключение к шине требует выполнения процедур выбора только одного из нескольких устройств для обмена данными. В-третьих, одновременный обмен данными с несколькими устройствами невозможен, если шины в текущий интервал времени уже заняты, передача каких-либо других данных невозможна. В микропроцессорной системе эти ограничения не создают особых проблем. Управляющим устройством системы, в том числе и для шины, обычно является микропроцессор, при нормальной работе он обменивается данными с другими устройствами системы поочередно, адресация используется не только для устройств, но и для данных.

Типичная структура микропроцессорной системы, построенной в соответствии с указанными принципами, приведена на рис. 2. В этой структуре, отражающей особенности построения типовой микропроцессорной систе-

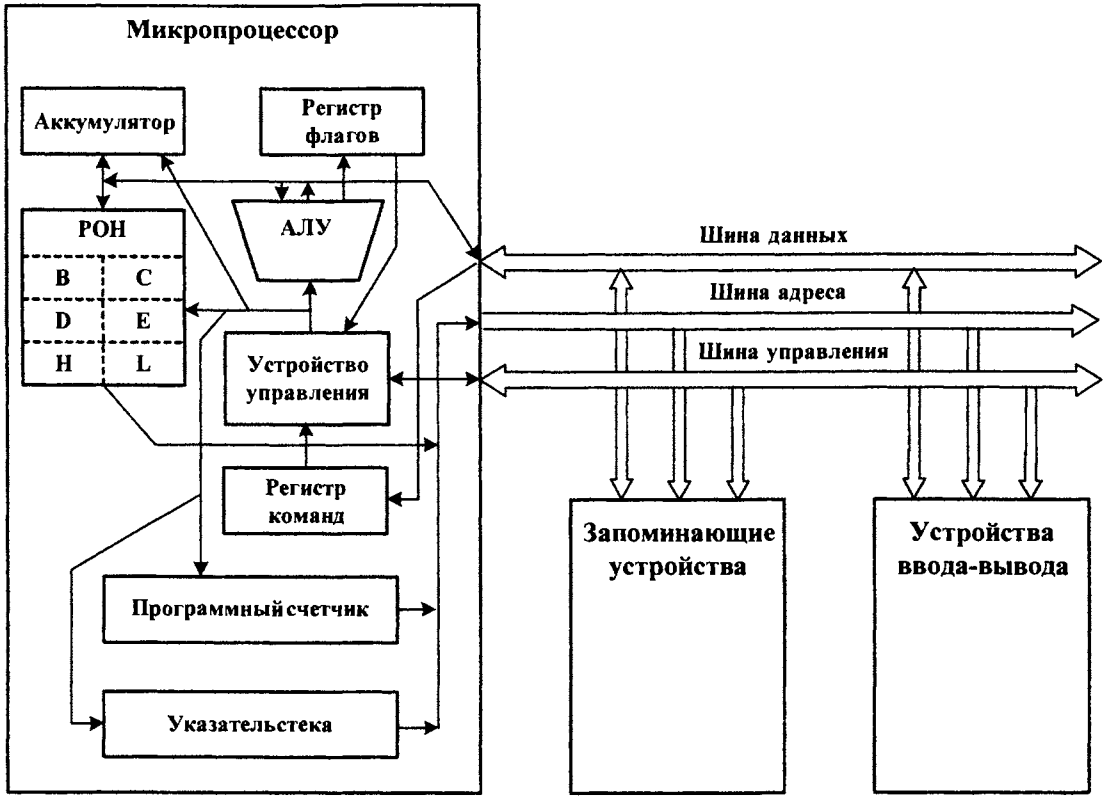

Рис. 2. Структурная схема микропроцессорной системы

мы, рассматривается работа микропроцессора Intel 8080 (отечественный аналог – К580ВМ80). Так как детальное описание этого микропроцессора можно найти в многочисленной литературе, изданной ранее (например, [1]), будем рассматривать только наиболее общие вопросы организации работы микропроцессорной системы (логическое взаимодействие), но с учетом особенностей реализации Intel 8080.

В микропроцессорной системе (рис. 2) запоминающие устройства (память) и устройства ввода-вывода (периферийные устройства) представлены в обобщенном виде, хотя в действительности они могут состоять из разнородных устройств с различными свойствами и реализуемыми функциями. Данные в параллельном формате в виде отдельных байт (8 бит) могут передаваться из микропроцессора или в микропроцессор по ШД. Сигналы ША практически всегда формируются микропроцессором и являются адресом (16 бит) байта данных, который микропроцессором передается или принимается. Сигналы ШУ в основном формируются устройством управления микропроцессора, хотя отдельные сигналы этой шины могут поступать и от других устройств, выполняя, например, функции запросов или подтверждений.

Структура микропроцессора (рис. 2) представлена в форме, отображающей только логическую организацию работы, ряд элементов, без которых невозможно корректное физическое взаимодействие, в этой структуре не показан. Микропроцессор содержит элементы, необходимые для выполнения требуемого набора команд (операций), и представляет собой средство выполнения этих команд, без программы – последовательности команд, реализующих заданный алгоритм работы микропроцессорной системы, никакие операции невозможны. Любые действия в микропроцессоре начинаются с чтения из запоминающего устройства (памяти) кода очередной команды и его ввода в регистр команд. Поэтому любую микропроцессорную систему необходимо рассматривать как программно-аппаратный комплекс. Аппаратные (hardware) и программные (software) средства тесно связаны и могут работать только в едином комплексе.

Код команды, поступивший в регистр команд микропроцессора, сигналами устройства управления определяет необходимые функции, как внутренних элементов микропроцессора, так и других устройств микропроцессорной системы через ШУ. Операции преобразования данных выполняются арифметико-логическим устройством – АЛУ (рис. 2). Набор этих операций обычно стандартный и включает сложение, вычитание, инкремент (увеличение переменной на единицу), декремент (уменьшение переменной на единицу), логические операции И, ИЛИ, исключающее ИЛИ, инверсия и т.п. В рассматриваемой системе основной формат данных АЛУ – байт (8 бит), АЛУ предназначено только для преобразования и не содержит элементов для хранения данных.

Входными данными для операций преобразования обычно служат две переменных, а результат преобразования – одна переменная. Например, для операции сложения в АЛУ входными переменными будут 2 слагаемых по одному байту, а результат-сумма – один байт и признак переноса, если сумма выходит за пределы однобайтового формата. Для хранения и входных переменных, и результатов преобразования микропроцессор содержит регистры: регистр-аккумулятор и 6 регистров блока регистров общего назначения (РОН). Одна из входных переменных (операндов) всегда размещается в аккумуляторе, а полученный результат преобразования также всегда направляется в аккумулятор. Второй операнд может храниться в одном из регистров РОН. В коде команды преобразования обычно указывается регистр, из которого поступает второй операнд, если аккумулятор единственный (рис. 2), его не указывают в коде команды.

Количество регистров данных микропроцессора невелико, в данном примере – 7 регистров (аккумулятор и 6 регистров блока РОН), эти регистры содержат данные, которые постоянно доступны для преобразования. Основной объем данных хранится в памяти микропроцессорной системы, однако для выполнения операций их необходимо переместить в регистры данных микропроцессора, используя процедуры адресации и управления обменом данных. Разрядность регистров блока РОН соответствует формату данных АЛУ – один байт. Эти 6 однобайтовых регистров (регистры B, C, D, E, H, L, рис. 2) могут использоваться независимо или объединяться в регистровые пары BC, DE, HL для выполнения определенных операций с двухбайтовыми данными. В первую очередь двухбайтовый формат необходим для операций с адресами, так как адресация данных в микропроцессорной системе производится двухбайтовым адресом ША. Такой формат адреса обеспечивает размер адресного пространства микропроцессорной системы $2^{16} = 64$ кбайт.

Организация микропроцессорной системы, кроме операций преобразования данных, требует выполнения операций пересылки данных, с помощью которых обеспечивается доступ к преобразуемым данным. Операции пересылки должны производить обмен данных между регистрами микропроцессора, между регистрами микропроцессора и памятью или устройствами ввода-вывода. Эти операции так же, как и операции преобразования, задаются командами программы с указанием кода операции и адресов приемника и источника для пересылаемых данных.

В рассматриваемой микропроцессорной системе используется память с единым адресным пространством объема 64 кбайта и для данных, и для кодов программы. Адреса данных содержатся в кодах программы, а адреса кодов программы определяются специальным регистром – программным счетчиком. Разрядность программного счетчика соответствует формату адреса (2 байта), его содержимое определяет адрес ячейки памяти, в которой хранится код очередной команды. Последовательность выполняемых ко-

манд обычно формируется в виде линейной последовательности кодов программы и размещается в ячейках памяти с последовательно нарастающими адресами.

Стандартный цикл работы микропроцессорной системы для очередной команды выполняется следующим образом:

1. По сигналам устройства управления микропроцессора производится считывание и ввод в регистр команд кода очередной команды по адресу, указанному в программном счетчике.
2. Содержимое программного счетчика автоматически инкрементируется (увеличивается на единицу) для определения следующего адреса хранения кодов программы.
3. Если код команды поступил в микропроцессор полностью, команда выполняется; если требуется чтение недостающих элементов кода, повторяется считывание с автоматическим инкрементом адреса в программном счетчике (повторение п.п. 1 и 2).
4. Когда очередная команда программы микропроцессорной системы выполнена, в программном счетчике уже содержится адрес следующей команды и начинается следующий рабочий цикл (см. п. 1).

В качестве примера программной реализации рассмотрим процедуру сложения двух переменных. Полагаем, что однобайтовые слагаемые D1 и D2 хранятся в ячейках памяти по двухбайтовым адресам A1 и A2, полученный однобайтовый результат необходимо сохранить по адресу A3, а последовательность кодов программы сложения также хранится в ячейках памяти с начальным адресом A4. С учетом особенностей работы микропроцессорной системы алгоритм должен предусматривать ввод из памяти в регистры микропроцессора обоих слагаемых, сложение содержимого двух регистров данных и пересылку в память полученного результата.

Эта последовательность команд при использовании мнемонических обозначений операций для микропроцессора Intel 8080 будет следующая:

```
m1: lda  A1     ; чтение байта данных D1 из ячейки памяти по адресу A1 и
                ; пересылка в аккумулятор микропроцессора
m2: mov  B, A   ; пересылка данных из аккумулятора в регистр B
l3:lda   A2     ; чтение байта данных D2 из ячейки памяти по адресу A2 и
                ; пересылка в аккумулятор микропроцессора.
m4: add  B      ; сложение содержимого аккумулятора и регистра B
                ; (D1+D2), результат сложения помещается в
                ; аккумулятор (A=A+B)
m5: sta  A3     ; пересылка байта данных из аккумулятора в память и
                ; запись в ячейку памяти по адресу A3
```

Для выполнения программы сложения начальный адрес программы A4 должен быть указан в программном счетчике. После завершения предыдущей команды устройство управления производит чтение кода из памяти по адресу A4 и его запись в регистр команд микропроцессора. Так как код этой команды содержит двухбайтовый адрес (A1), устройство управления дополнительно производит чтение остальных двух байт с автоинкрементом адреса в программном счетчике. Только после чтения всех трех байт кода команды операция может быть выполнена, т.е. адрес A1 поступит в шину адреса, байт данных D1 будет помещен в аккумулятор, а в программном счетчике – адрес следующей команды (A4+3). Далее микропроцессор производит чтение и выполнение следующей команды, автоматически формируя адреса последующих команд в программном счетчике, и т.д.

Необходимо обратить внимание на две важные особенности программной реализации. Во-первых, выполнение любой операции состоит из двух этапов: чтение и ввод кода выполняемой команды, и непосредственное выполнение команды. Для ускорения работы в современных микропроцессорах эти этапы обычно выполняют параллельно, выполнение текущей команды совмещают с вводом кода следующей команды. Во-вторых, порядок выполнения операций определяется программным счетчиком. При работе микропроцессора адрес в программном счетчике автоматически инкрементируется, что и определяет линейную очередность выполнения команд с нарастающими адресами в памяти. Если в программе требуется изменить порядок выполнения команд, необходимо изменить содержимое программного счетчика. Замена адреса в программном счетчике позволяет вызвать для выполнения другие фрагменты программы, изменяя линейную очередность выполнения команд.

Линейные алгоритмы с последовательным выполнением программы недостаточно функциональны, поэтому в реализуемых алгоритмах практически всегда требуется управление выполнением программы. Для решения этой задачи набор команд любого микропроцессора содержит специальную группу команд управления. Назначение команд управления – изменение в программном счетчике адресов выполняемого в данный момент фрагмента программы. Такое управление программным счетчиком и позволяет выполнять все необходимые функции управления работой программы.

Рассмотрим алгоритм, требующий управления работой программы, например, для задачи поддержания микроклимата в помещении. Типичная постановка задачи может быть следующая: нормальное состояние помещения при температуре в диапазоне $t_{min} - t_{max}$; если температура превышает t_{max}, должна включаться система охлаждения; если температура ниже t_{min}, должна включаться система подогрева. Очевидно, что средства охлаждения и подогрева не должны работать одновременно, компенсируя работу друг друга. Возможный вариант алгоритма представлен на рис. 3.

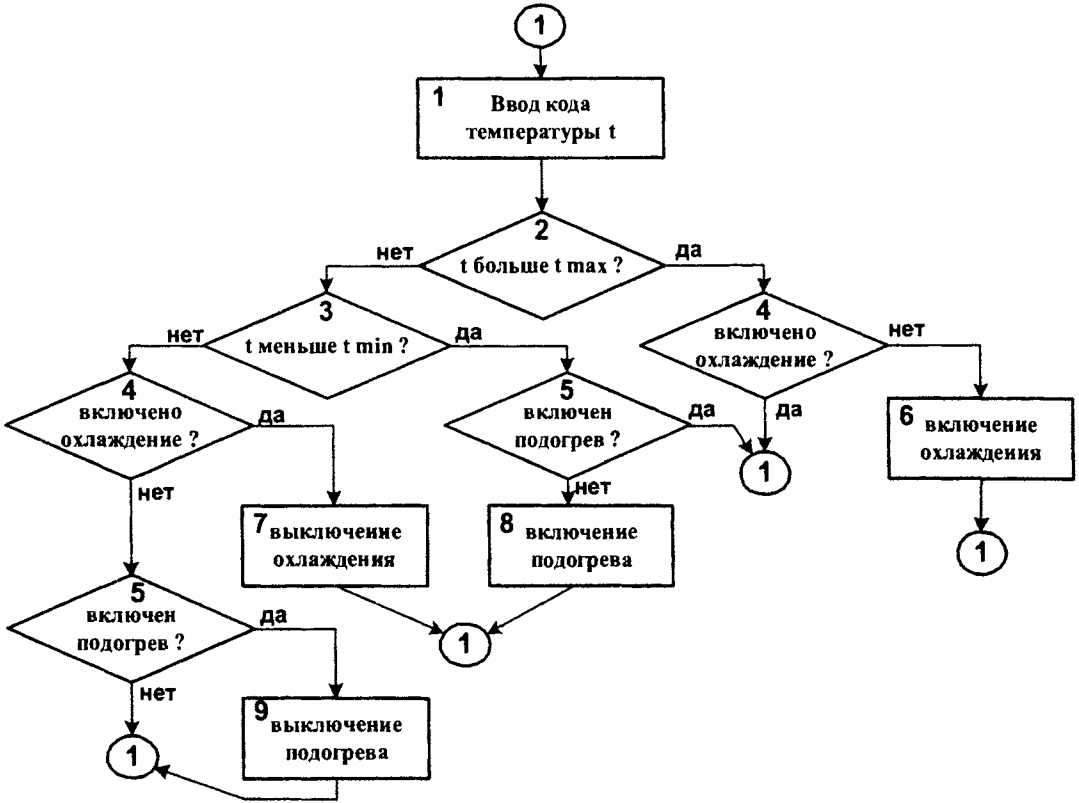

Рис. 3. Блок-схема алгоритма управления температурой

По-видимому, сам алгоритм работы в комментариях не нуждается. Однако, с точки зрения программной реализации, имеются определенные особенности. По результатам операций контроля (блоки 2 – 5 алгоритма на рис. 3) работа программы должна продолжаться по одной из двух возможных ветвей алгоритма. Операции программы, представленные в блоках 4, 5 алгоритма, повторяются в разных ветвях. Количество точек для завершения цикла контроля температуры и возврата к началу повторного цикла (точки 1 в алгоритме) равно 6. Очевидно, что такой алгоритм можно реализовать только с помощью команд управления, выполняя в каждой точке разветвления алгоритма операции управления для выбора требуемых функций в программе.

Возврат к началу цикла (блок 1 алгоритма) из точек завершения рабочего цикла (точки 1) можно выполнить командами безусловной передачи управления, которые загружают в программный счетчик микропроцессора начальный адрес кодов блока 1. В точках ветвления алгоритма (блоки 2-5 алгоритма) продолжение работы зависит от результатов контроля (ветви "да" – "нет"). Управление работой программы в этих случаях необходимо производить командами передачи управления по выполнению определенного условия (условная передача управления).

Признаки этих условий формируются в АЛУ микропроцессора автоматически при выполнении операций и содержатся в регистре флагов (рис. 2). Каждый используемый бит регистра флагов является признаком (флагом), который отображает особенности результата операции, выполненной АЛУ. Флаги этого регистра поступают в устройство управления микропроцессора и влияют на выполнение команд условной передачи управления. В Intel 8080 используется 5 флагов:

- Z – флаг нулевого результата, флаг Z = 1, если результат операции (байт аккумулятора) равен нулю;
- S – флаг знака результата, флаг S равен старшему биту результата, который при кодировании чисел со знаком используется как знаковый бит;
- C – флаг переноса, флаг C = 1, когда формируется перенос из старшего разряда результата операции;
- AC – флаг вспомогательного переноса, флаг AC = 1, когда происходит перенос из младшей тетрады результата в старшую тетраду;
- P – флаг паритета, флаг P = 1, когда количество единиц в байте результата четно.

Любой бит регистра флагов может быть использован в качестве условия для команд условной передачи управления. Если условие для такой команды управления выполняется, то производится загрузка в программный счетчик микропроцессора адреса, указанного в команде. Если условие не выполняется, адрес в программном счетчике автоинкрементируется и, как обычно, выполняется следующая по порядку команда.

Рассмотрим примеры применения команд управления.

Пример 1:

```
m1:     add B  ; сложение (аккумулятор=аккумулятор+регистрB) и
               ; формирование флагов
m2:  jmp  A1       ; переход к программе по адресу A1
m3:  .  .  .  .  ; продолжение текста программы
```

Команда m1 производит сложение двух байтов и формирование всех 5 флагов. Команда m2 запишет в программный счетчик адрес A1, поэтому далее фрагментт3 выполняться микропроцессором не будет, а выполнение программы всегда продолжится по адресу A1 (безусловная команда управления).

Пример 2:

```
m1:     add B ; сложение (аккумулятор=аккумулятор+регистрB)
              ; и формирование флагов
m2:    jC   A2      ; переход к программе по адресу A2, если флаг
                    ; переноса C=1
m3:    jZ   A3      ; переход к программе по адресу A3, если флаг
                    ;нулевого результата Z=1
m4:  .  .  .  .  ; продолжение текста программы
```

Команда m1 производит сложение двух байт и формирование всех 5 флагов. Команда m2 запишет в программный счетчик адрес A2, если в результате сложения флаг перенвса C=1. Если переноса нет (C=0), команда m2 не выполняется и, следовательно, будет продолжено выполнение программы по m3. Аналогичным образом команда m3 анализирует флаг Z. Работа программы по m4 будет продолжена, если результат сложения не нулевой и не формируется перенос – в данном случае признак превышения результатом однобайтового формата. Таким образом, в примере 2 работа программы, в зависимости от результата сложения, может быть продолжена по одному из трех вариантов: по адресу A2, по адресу A3 или по m4.

Кроме указанных команд, в группе команд управления используются и другие команды, особенности выполнения которых будут рассмотрены позднее. В целом, система команд микропроцессора, включающая команды преобразования данных, команды пересылки данных и команды управления, является важнейшей характеристикой микропроцессора и, как было показано, система команд построена для выполнения всех необходимых

функций программной реализации при разнообразных алгоритмах преобразования данных. Компоненты микропроцессора и микропроцессорной системы фактически являются средствами аппаратной поддержки системы команд и функций, реализуемых программно.

Последний элемент микропроцессора, который не был рассмотрен, – регистр указатель стека (рис. 2). Указатель стека – специальный адресный регистр с форматом, соответствующим формату ША (16 бит), он используется для специального способа адресации памяти микропроцессорной системы. При стековой организации хранения данных всегда используется адрес, содержащийся в указателе стека. Поэтому в явном виде в операциях со стеком адрес не указывается. Более подробно использование стека будет рассмотрено позднее.

1.2. Форматы данных микропроцессорной системы

В микропроцессорной системе (рис. 2) разрядность ШД и регистров данных – 8, поэтому способы кодирования данных будем рассматривать применительно к однобайтовому формату. Этот формат данных является достаточно типичным для микропроцессорных средств. Байт данных D обозначим, начиная со старших разрядов, следующим образом:

$$D=D_7D_6D_5D_4D_3D_2D_1D_0 \ .$$

Байт D при выполнении команд преобразования данных в микропроцессоре можно интерпретировать как целое число без знака, целое число со знаком, логическую переменную или символьную переменную. Операции в других форматах данных, например, для дробных чисел, чисел с плавающей точкой и т.п., требуют применения специальных алгоритмов преобразования и способов кодирования. Рассмотрим выполнение операций преобразования данных для разных способов кодирования.

Для целых чисел без знака каждый бит D_i имеет вес 2^i в соответствии со стандартным двоичным кодированием, диапазон изменения чисел 0 – 255 (2^8-1). В текстах программ числа без знака можно указывать в десятичном формате (0 – 255), в двоичном формате (0b00000000 – 0b11111111, 0b – стандартный признак двоичного формата) и шестнадцатеричном (hex) формате (0x00 – 0xff, 0x – стандартный признак hex-формата). В hex-формате для основания системы счисления 16 в дополнение к десятичным символам используют первые 6 букв латинского алфавита a, b, c, d, e, f. Наиболее функционален hex-формат, так как он лучше соответствует исходному двоичному кодированию байта данных. Для hex-формата байт условно разбивается на две тетрады, а каждая тетрада записывается как отдельный hex-символ.

Иногда применяют двоично-десятичное кодирование (упакованный bcd-формат) с кодированием двух десятичных символов в каждой тетраде. Для выполнения арифметических операций bcd-формат неудобен, так как ре-

зультат формируется в двоичном формате и требуется его коррекция для восстановления исходного bcd-формата. Диапазон чисел для однобайтового bcd-формата 0 – 99.

Для кодирования чисел со знаком применяются стандартные арифметические коды. Старший разряд байта D_7 становится битом знака числа, а величина числа определяется оставшимися 7 битами D_0-D_6.В дополнительном коде знак "+" соответствует нулевому знаковому биту, а знак "–" соответствует единичному знаковому биту. Величина числа в дополнительном коде для положительного и отрицательного числа формируется различными способами.

Для положительных чисел применяют обычное двоичное кодирование величины. Для отрицательных чисел формируют дополнение, т.е. инверсию двоичного кода величины с прибавлением 1 младшего разряда ($INV(D_6D_5D_4D_3D_2D_1D_0)+1$) . Восстановление двоичного кода величины отрицательного числа также требует инверсии с прибавлением 1 младшего разряда. Диапазон однобайтовых чисел для дополнительного кода (–128) – (+127). Применение дополнительного кода позволяет использовать одни и те же арифметические операции для чисел без знака и чисел со знаком, так как знаковый бит преобразуется в этих операциях вместе с другими битами чисел.

Примеры однобайтового кодирования чисел приведены в табл. 1.

Таблица 1

двоичный код байта данных	*десятичный формат (число без знака)*	*hex-формат (число без знака)*	*bcd-формат (число без знака)*	*десятичный формат (число со знаком)*
0b01011001	*89*	*0x59*	*59*	*+ 89*
0b11011001	*217*	*0xd9*	*недопустимый символ в старшей тетраде*	*– 39*
0b01001011	*75*	*0x4b*	*недопустимый символ в младшей тетраде*	*+ 75*
0b10000011	*131*	*0x83*	*83*	*– 125*

В первом столбце примеров приведен двоичный код байта данных, в остальных столбцах – интерпретация этого байта для разных способов кодирования.

При выполнении в микропроцессоре операции сложения байтов 1 и 2 получим:

01011001+11011001=00110010 с флагами Z=0, S=0, C=1, AC=1, P=0.

- *Для чисел без знака (89+217 или 0x59+0xd9) байт результата: 50 (0x32), а флаг переноса C=1 означает, что результат суммирования вышел за пределы однобайтового формата и этот флаг должен интерпретироваться, как бит 9 (2^8=256) суммы.*
- *Для чисел со знаком ((+89)+(–39)) байт результата положительный: +50, флаг переноса должен игнорироваться, как не имеющий значения в этом случае.*

При выполнении в микропроцессоре операции сложения байтов 3 и 4 получим

01001011+10000011=11001110 с флагами Z=0, S=1, C=0, AC=0, P=0.

- *Для чисел без знака (75+131 или 0x4b+0x83) байт результата: 206 (0xce), флаг переноса C=0 означает, что результат суммирования не вышел за пределы однобайтового формата, а флаг знака S=1 просто повторяет значение старшего бита результата D_7.*
- *Для чисел со знаком ((+75)+(–125)) байт результата отрицательный: – 50, флаг знака S=1, повторяя значение D_7, также является признаком отрицательной суммы.*

Конечно, однобайтовый формат дает весьма ограниченный диапазон изменения чисел. Если требуется расширить этот диапазон, необходимо переходить к двухбайтовому (многобайтовому) формату. Для преобразования многобайтовых чисел в системе команд микропроцессора существуют команды операций, которые учитывают при выполнении значение флага переноса C. Например, команда сложения **add B** производит суммирование байта аккумулятора (A8) и байта регистра B (A8=A8+B8) с формированием всех флагов. Модификация команды сложения **adc B** при суммировании добавляет сформированное ранее значение флага переноса C (A8=A8+B8+C) и формирует новые значения флагов.

В операциях над многобайтовыми числами преобразование всех младших байтов должно интерпретироваться, как операция над числами без знака, но с учетом значения флага переноса. Знак числа может присутствовать только в старшем бите самого старшего байта. Рассмотрим программную реализацию суммирования двухбайтовых чисел, интерпретация результата, как было показано для однобайтового формата, зависит только от кодирования исходных данных.

Пусть двухбайтовые слагаемые находятся в регистровых парах BC, DE (первый регистр каждой пары содержит старший байт): результат суммирования должен быть помещен в регистровую пару HL.

```
m1:  mov  A, C  ; пересылка младшего байта  слагаемого 1
                ;  в аккумулятор (A8=C8),
m2:  add    E     ; суммирование младших байтов (A8=A8+E8)
                ;  и формирование флагов,
m3:  mov  L, A  ; пересылка младшего байта суммы
                ; в регистр L (L8=A8).
m4:  mov  A, B  ; пересылка старшего байта слагаемого 1
                ;в аккумулятор (A8=B8),
;примечание: команды пересылки значения сформированных ранее
; флагов не изменяют,
m5:  adc    D  ; суммирование старших байтов и флага переноса,
              ; сформированного при суммировании младших
              ; байтов, (A8=A8+D8+C),
m6:  mov  H, A  ; пересылка старшего байта суммы
                ; в регистр H (B8=A8).
```

Как видно из примера, переход к двухбайтовому формату увеличивает объем программы, как минимум в два раза.

Для логических операций байт данных D должен интерпретироваться как однобайтовая логическая переменная, в которой каждый бит D, является независимым элементом. Как правило, логические операции выполняются побитно и также побитно формируют байт результата. Например, операция "логическое И" **ana B** выполняет побитную конъюнкцию содержимого аккумулятора A и регистра B (A_7=A_7&B_7, A_6=A_6&B_6, . . . A_0=A_0&B_0), флаги S, Z, P соответствуют байту результата в аккумуляторе, а флаги переноса – нулевые (C=AC=0), так как перенос не производится.

Операции логического сдвига могут выполнять побитовое смещение байта данных аккумулятора и в направлении старших разрядов (сдвиг влево), и в направлении младших разрядов (сдвиг вправо). Значение флага переноса в операциях сдвига зависит от направления: при сдвиге влево – C=A_7, при сдвиге вправо – C=A_0. Использование флага переноса позволяет выполнять сдвиг многобайтовых переменных по алгоритмам, аналогичным алгоритмам арифметических операций над многобайтовыми числами.

Если байт данных при сдвиге интерпретировать как число без знака, сдвиг вправо эквивалентен делению числа на 2, а сдвиг влево – умножению на 2. Для чисел со знаком, так называемый арифметический сдвиг, при сдвиге необходимо обеспечить сохранение знака, т.е. старший (знаковый) бит числа при сдвиге должен быть неизменным.

Символьный формат данных в основном предназначен для текстовых сообщений. Специальных команд обработки символьных данных нет, они могут преобразоваться как числовые или логические переменные. Так как для ввода текстов или их отображения используются компьютерные средства, кодирование в символьном формате обычно производят стандартными компьютерными кодами. Таким стандартным символьным кодом однобайтового формата является, например, код ASCII. ASCII предназначен для кодирования всех символов, используемых на клавиатуре персонального компьютера. Например, символьной переменной 'A' соответствует однобайтовый ASCII-код 0x41, символьной переменной 'k' – ASCII-код 0x6b, символьной переменной '2' – ASCII-код 0x32 и т.п. Полную таблицу кода ASCII можно найти в любой литературе, описывающей форматы компьютерного кодирования данных.

Данные других форматов, кроме рассмотренных в данном разделе, также можно обрабатывать в микропроцессорной системе. При выборе алгоритмов преобразования необходимо учитывать, что система команд микропроцессора содержит операции над числовыми и логическими переменными. Следовательно, операции преобразования данных других форматов необходимо приводить к эквивалентным операциям, реализуемым в системе команд.

В целом, система команд и форматы данных микропроцессора являются универсальными средствами преобразования. Очевидно, что эффективность работы микропроцессорной системы в силу ограниченного набора команд различна для различных форматов данных. Поэтому применяемые форматы данных – весьма важный компонент реализуемых алгоритмов преобразования. Следующий важный элемент в организации работы микропроцессорной системы – процедуры адресации данных, именно адресация обеспечивает доступ к данным для алгоритмов преобразования.

1.3. Организация памяти и адресация данных

Запоминающие устройства (память) микропроцессорной системы по выполняемым функциям и свойствам можно разделить на две области: оперативное запоминающее устройство (RAM) и постоянное запоминающее устройство (ROM). В рассматриваемой структуре (рис. 2) эти области образуют единое адресное пространство, и для них используется общая система адресации с 16 битами ША. В других вариантах микропроцессорных систем разные области памяти могут иметь самостоятельные системы адресации.

Для единого адресного пространства нет какого-либо начального разделения функционального назначения областей памяти. Так называемая классическая архитектура компьютера предполагает возможность хранения и кодов программ, и данных для преобразования по любым адресам запоминающих устройств. Запоминающие устройства в этом случае могут рас-

сматриваться как набор типовых ячеек памяти, каждая из которых позволяет хранить один байт данных и имеет уникальный 16-битовый адрес, передаваемый в микропроцессорной системе по ША. Уникальный адрес позволяет выбрать для обмена данными одну ячейку памяти из всего множества ячеек запоминающих устройств и обеспечить доступ только к выбранному байту данных. Максимальный объем адресного пространства 2^{16} ячеек памяти (64 кбайт).

Оперативное запоминающее устройство (RAM) позволяет записывать однобайтовые данные для хранения и считывать записанные ранее данные. Направление передачи данных (микропроцессор → RAM или RAM → микропроцессор) определяется микропроцессором с помощью сигналов ШУ. Вместе с сигналами управления микропроцессор должен передать адрес выбранной ячейки памяти по ША. Ячейки RAM должны обеспечивать запись данных, хранение данных и считывание хранящихся данных.

При работе микропроцессорной системы RAM позволяет выполнять все необходимые функции по хранению информации и, следовательно, является обязательным элементом. Однако RAM производит хранение данных только при включенном питающем напряжении. При выключении питания все записанные данные будут потеряны и не могут быть восстановлены повторным включением питания, т.е. RAM – память энергозависимая.

Как было показано ранее, в микропроцессорной системе все операции, в том числе и загрузка данных в память, выполняются программно. При отсутствии программы в памяти реализация каких-либо функций невозможна. Эта особенность работы микропроцессорной системы требует постоянного наличия программы в памяти, даже при отключении питания. RAM, как известно, такую возможность не обеспечивает, поэтому микропроцессорная система обязательно содержит и постоянное запоминающее устройство – ROM.

ROM – память энергонезависимая, т.е. сохраняет записанные данные и при отключении питания. Другой ее особенностью является возможность только считывать данные при работе микропроцессорной системы. Стандартные процедуры и сигналы управления не позволяют выполнять запись новых данных в ROM. Запись данных ROM выполняется специальными средствами и обычно называется программированием. Существуют ROM с однократным программированием данных, позволяющие выполнять запись только один раз, и репрограммируемые ROM, с возможностью многократной перезаписи данных (обычно с ограниченным количеством циклов перезаписи).

Таким образом, данные в ROM должны быть записаны до начала работы микропроцессорной системы, а при включении – обеспечивать выполнение необходимых алгоритмов работы в соответствии с записанными кодами программы. ROM называют также памятью программ и констант, так как и

коды программы, и константы – данные, которые не изменяются в процессе выполнения программы микропроцессорной системы.

Соотношение между объемами RAM и ROM в адресном пространстве памяти зависит от назначения микропроцессорной системы. В универсальных средствах таких, как персональный компьютер, объем ROM относительно небольшой. Из программ, необходимых для работы, в ROM персонального компьютера постоянно хранится только BIOS – комплекс программ, обеспечивающих загрузку в RAM других программ с внешних носителей информации (например, с hard-диска). Набор задач жестко не ограничен и для решаемых задач выбираются и загружаются в RAM с помощью BIOS требуемые компоненты программного обеспечения.

Для специализированных микропроцессорных систем, к которым относятся и средства автоматизации, обычно и круг задач, и программы, требуемые для их решения, неизменны и могут постоянно храниться в памяти. Поэтому в таких средствах удельный вес ROM в адресном пространстве практически всегда существенно превышает удельный вес RAM. В дальнейшем будем рассматривать вопросы организации работы микропроцессорной системы, полагая, что все программы хранятся в ROM и имеется RAM относительно небольшого объема только для хранения данных.

Задание определенных физических адресов для областей RAM и ROM в общем случае может быть произвольным. Однако необходимо учитывать следующую особенность работы микропроцессора: процедура начального сброса (reset), с которой обязательно должна начинаться работа, как правило, задает в программном счетчике нулевой начальный адрес программы. По начальному адресу памяти должно размещаться начало рабочей программы микропроцессора, которая обеспечивает корректный старт работы микропроцессорной системы. Область памяти с этими начальными адресами, следовательно, должна принадлежать области ROM. Чтобы адресные пространства сделать непрерывными, обычно адреса ROM задают, начиная с наименьших адресов, а адреса RAM – в оставшемся диапазоне наибольших физических адресов.

Ранее было показано, что адресация кодов программы производится программным счетчиком микропроцессора. Адресация всех остальных данных должна производиться командами программы. Организация работы микропроцессорной системы предполагает использование нескольких способов адресации данных. Если память микропроцессорной системы образует единое адресное пространство, способы адресации не зависят от того, с какой областью памяти мы работаем.

Применяемые в микропроцессорной системе алгоритмы адресации данных можно привести к одному из четырех видов: непосредственная адресация, прямая адресация, регистровая адресация, стековая адресация. Особенности адресации связаны с разным форматом данных и адресов: данные – один байт, адреса – два байта.

Самый простой вид – непосредственная адресация. В этом случае байт данных помещается непосредственно в коде команды. *Примеры: команда* ***mvi B, D8*** *производит загрузку в регистр B байта данных D8 (B8=D8), содержащегося в коде команды; команда* ***sui D8*** *производит вычитание из байта аккумулятора байта данных D8 (A8=A8–D8).* Команды с непосредственной адресацией часто называют операциями с константами. Так как данные при непосредственной адресации являются элементами кодов программы, эти данные не могут быть изменены. Неизменность данных является свойством и основным ограничением этого вида адресации, который в действительности применяется только для операций с константами.

Для прямой адресации физический адрес ячейки памяти (ША) указывается в коде команды*: например, команда* ***sta A16*** *производит запись байта данных из аккумулятора в ячейку памяти с двухбайтовым адресом A16 (A8=M(A16)).* При такой адресации передаваемые байты данных являются обычными переменными и могут принимать любые допустимые значения. Существенное ограничение – обмен данными по командам с прямой адресации всегда происходит с одной и той же ячейкой памяти, адрес которой указан в команде. Из-за этого ограничения и громоздкого формата команд, содержащих в своем коде двухбайтовый адрес, прямая адресация также имеет ограниченное применение. В алгоритмах обработки данных достаточно часто требуется адрес сделать переменным, чтобы можно было им программно управлять.

Более универсальна регистровая адресация, называемая также косвенной адресацией. При регистровой адресации в качестве кода адреса используется содержимое регистровой пары (два байта). *Примеры: команда* ***ldax DE*** *производит загрузку в аккумулятор байта данных из ячейки памяти с адресом, указанным в регистровой паре DE (A8=M(DE)); команда* ***ana M*** *выполняет конъюнкцию байтов аккумулятора и ячейки памяти с адресом, указанным в регистровой паре HL (A8=(A8)&M(HL)).* В большинстве команд с регистровой адресацией микропроцессора Intel 8080 код адреса хранится в регистровой паре HL (H – старший байт адреса, L – младший байт адреса). Регистровая адресация, во-первых, упрощает формат команд – в явном виде код команды не содержит двухбайтового адреса; и, во-вторых, адрес можно изменять, выполняя с байтами адреса любые операции из набора команд микропроцессора. Возможность преобразования адреса при работе программы является важнейшим свойством этого вида адресации и широко используется в различных алгоритмах.

Очевидно, что регистровая адресация требует предварительной подготовки (загрузки или преобразования) кода адреса в регистровой паре. Для преобразования адресов в регистровых парах система команд содержит специальные операции с двухбайтовыми данными.

Например, команда ***lxi HL, D_LD_H*** *производит загрузку двух байт (D_L8 и D_H8) в регистры L и H (L8 = D_L8, H8 = D_H8), используя непосредственную адресацию, а команда* ***inx HL*** *выполняет инкремент содержимого регистровой пары HL, как двухбайтовой переменной (HL=HL+1).*

Четвертый вид – стековая адресация, в первую очередь, предназначена для работы с подпрограммами. Более подробно процедуры с подпрограммами будут рассмотрены в следующем разделе. Следует помнить, что стек в микропроцессорной системе – не отдельное запоминающее устройство, а способ адресации RAM с особым алгоритмом формирования адреса. Адрес RAM для операций со стеком всегда определяется специальным адресным регистром – указателем стека (SP) микропроцессора (рис. 2).

В командах, использующих стековую адресацию, адрес в явном виде никогда не указывается и определяется следующим алгоритмом доступа к данным: "последний вошел – первый вышел". Обычно этот алгоритм реализуется преобразованием содержимого указателя стека: при записи данных указатель стека (SP) автоматически декрементируется, а при чтении данных указатель стека (SP) автоматически инкрементируется. Такое преобразование адреса в указателе стека обеспечивает в командах чтения следующий порядок доступа: первым считывается байт данных, записанный последним, вторым – записанный предпоследним, и т.д. Следовательно, порядок записи данных в стек однозначно определяет порядок поступления данных при чтении, а стековая адресация не позволяет производить независимый доступ к любым данным, как другие виды адресации.

Стековая адресация применяется из-за упрощенного формата команд и процедур обмена данными в алгоритмах, требующих однократной записи в память и затем однократного считывания данных из памяти в строго заданной последовательности. Команды со стековой адресацией микропроцессора Intel 8080 производят обмен двухбайтовыми данными.

Рассмотрим пример применения стека для хранения данных:

```
m1: push BC   ; B8→RAM(SP–1), C8→RAM(SP–2),   SP=SP–2
m2: push DE   ; D8→RAM(SP–3), E8→RAM(SP–4),   SP=SP–4
  . . . . . ; команды программы, изменяющие данные в
  . . . . . ; регистрах B, C, D, E
  . . . . . ; однако данные в этих регистрах должны быть восста-
  . . . . . ; новлены для дальнейшего использования в программе.
m3: pop DE    ; RAM(SP–4)→E8, RAM(SP–3)→D8,   SP=SP–2
m4: pop BC    ; RAM(SP–2)→C8, RAM(SP–1)→B8,   SP=SP
```

Комментарии в каждой строке примера поясняют характер выполняемых операций.

Стековая адресация требует обязательной инициализации указателя стека (SP) в начале работы микропроцессорной системы. В качестве начального адреса в SP обычно загружается наибольший физический адрес RAM, запись данных в стек автоматически производится от больших адресов к меньшим адресам. Корректное выполнение рабочей программы микропроцессорной системы требует, чтобы количество команд записи данных в стек было строго равно количеству команд чтения из стека. Кроме того, адресное пространство RAM, выделенное для стека, не должно использоваться остальными компонентами программы. Другие особенности организации обработки данных в микропроцессорной системе рассматриваются в следующем разделе.

1.4. Организация обработки данных

В разделе 1.1 показано, что в микропроцессорной системе естественный порядок выполнения команд рабочей программы, определяющийся автоматическим инкрементированием программного счетчика, – последовательный, выполняемые команды размещены в ячейках ROM с последовательно нарастающими адресами. При необходимости этот порядок может быть изменен командами управления, основные функции которых и заключаются в изменении адресов выполняемых команд в программном счетчике. С помощью команд управления можно производить обработку данных с различной организацией алгоритмов выполнения.

Один из стандартных алгоритмов организации обработки данных – замена линейной последовательности однотипных операций циклическим повторением одного и того же фрагмента программы, так называемая организация циклов. Циклы в программах могут быть с заданным числом повторений или с завершением цикла по заданному условию. Рассмотрим организацию циклов на примерах.

Пример 1 (цикл с фиксированным числом повторений):
Вычислить сумму массива из 16 чисел (целые числа без знака), которые хранятся в последовательных ячейках памяти, начиная с адреса A1, и записать в память по адресу A2.

Примечание 1. Задачу можно решить линейным алгоритмом, выполнив 16 команд с прямой адресацией для чтения из памяти каждого байта данных и соответствующее число команд суммирования, получив простую, но громоздкую программу.

Примечание 2. При суммировании 16 байтов довольно высока вероятность того, что сумма выйдет за однобайтовый формат. Программа должна предусматривать формирование двухбайтовой суммы.

Для решения задачи определим функции регистров микропроцессора: регистр B – второй байт суммы, регистр C – счетчик числа циклов сумми-

рования, регистровую пару HL используем для адресации, так как в каждом следующем цикле суммирования байт данных должен поступать из следующей ячейки памяти.

```
m1:  lxi HL, A1 ; начальный адрес массива чисел в HL,
m2:  mvi C, 0x10  ; количество чисел массива (количество циклов),
m3:  xra A        ; очистка аккумулятора (A8=0),
m4:  mov B, A   ; очистка регистра B (B8=A8=0),
m5:  add M        ; сложение (A8=A8+M(HL)),
m6:  jnC m8       ; перейти к m8, если флаг переноса 0,
m7:  inr B      ; увеличить старший байт на 1 (если флаг переноса 1),
m8:  inx HL       ; инкрементировать адрес в HL для следующего
                  ; цикла суммирования,
m9:  dcr C      ; декрементировать количество оставшихся циклов
                ; суммирования,
m10: jnZ m5       ; вернуться к m5, если цикл суммирования не последний,
m11: sta A2       ; сохранить в памяти младший байт суммы,
m12: mov A, B   ; переслать в аккумулятор старший байт суммы,
m13: sta (A2+1) ; сохранить в памяти старший байт суммы,
m14: . . . .
```

Команды m1-m4 выполняют инициализацию (подготовку параметров) цикла; команды m5-m10 составляют тело цикла с формированием двух байтов суммы, повторяющееся 16 раз; команды m11-m13 производят сохранение полученной суммы в памяти по завершению цикла.

Пример 2 (цикл с завершением по условию):

Дополним задачу примера 1 следующим условием: суммирование чисел должно производиться только до появления переноса в старший байт. Следовательно, суммирование может быть выполнено и для всех чисел массива, если нет переноса, и может быть прервано раньше при ненулевом переносе. Сумма в любом случае имеет однобайтовый формат.

Назначение регистров C, HL микропроцессора аналогично примеру 1, регистр B будем использовать для сохранения промежуточного значения суммы.

```
m1:  lxi HL, A1   ; начальный адрес массива чисел в HL,
m2:  mvi C, 0x10  ; количество чисел массива (количество циклов),
m3:  xra A        ; очистка аккумулятора (A8=0),
m4:  mov B, A   ; сохранение предыдущей суммы в регистре B (B8=A8),
m5:  add M        ; сложение (A8=A8+M(HL)),
m6:  jC m11       ; перейти к завершению по m11, если флаг переноса 1,
m7:  inx HL       ; инкрементировать адрес в HL для следующего
                  ; цикла суммирования,
```

m8: ***dcr C*** *; декрементировать количество оставшихся циклов*
; суммирования,
m9: ***jnZ m4*** *; вернуться к m4, если цикл суммирования не последний,*
m10: ***jmp m12*** *; перейти к завершению по m12 для последнего цикла*
; суммирования,
m11: ***mov A, B*** *; переслать в аккумулятор предыдущее значение*
;суммы при переносе,
m12: ***sta A2*** *; сохранить в памяти сумму,*
m13: . . . .
В примере 2 цикл может завершиться по m6 при появлении переноса, и по m12, если перенос при суммировании не формируется.

Еще одной важной процедурой обработки данных является подпрограмма. Для выполнения этой процедуры в командах управления имеются специальные команды: "call A16" – вызов подпрограммы и "ret" – возврат из подпрограммы. При вызове подпрограммы работа основной программы временно приостанавливается, выполняются команды подпрограммы, затем работа основной программы продолжается. Подпрограммы обычно применяют для реализации повторяющихся в разных частях основной программы функций.

Корректное продолжение работы приостановленной программы требует восстановления данных в определенных регистрах микропроцессора. Этот набор данных, необходимых для работы программы, принято называть вектором состояния программы. В общем случае компонентами вектора состояния являются:

- адрес очередной команды в программном счетчике;
- регистр флагов, так как флаги могут измениться при выполнении подпрограммы;
- аккумулятор, его содержимое практически всегда изменяется при выполнении подпрограммы;
- регистры блока РОН, содержимое регистров также может измениться при выполнении подпрограммы.

Следовательно, любые компоненты вектора состояния, изменяющиеся при выполнении подпрограммы, должны сохраняться в памяти и при завершении подпрограммы восстанавливаться в регистрах микропроцессора. Как правило, эти операции с вектором состояния производят с помощью стека, поэтому стековая адресация всегда поддерживается в микропроцессорных системах.

Очевидно, что вызываемая подпрограмма также требует формирования своего вектора состояния. Вектор состояния вызываемой подпрограммы принято называть вектором прерывания. Так как подпрограмма только начинает работать с момента вызова, в векторе прерывания обычно указыва-

ют только начальный адрес вызываемой подпрограммы для программного счетчика.

Таким образом, процедура использования подпрограммы включает следующие операции: остановка выполнения и сохранение вектора состояния программы, загрузка вектора прерывания и выполнение подпрограммы по вектору прерывания, восстановление вектора состояния и продолжение работы программы. Выполнение всех этих операций поддерживается системой команд микропроцессора.

Команда вызова подпрограммы "call A16" производит автоматическое сохранение в стеке текущего адреса команды из программного счетчика микропроцессора и загрузку адреса A16 вектора прерывания в программный счетчик, а команда завершения прерывания "ret" – такое же автоматическое восстановление адреса в программном счетчике из стека. Из-за такого алгоритма вызова и завершения прерывания задача сохранения и восстановления остальных компонентов вектора состояния программы обычно решается в подпрограмме прерывания. Подпрограмма должна начинаться с сохранения в стеке содержимого тех регистров микропроцессора, которые будут использоваться при ее работе. А перед командой возврата из подпрограммы (ret) должно выполняться восстановление содержимого этих же регистров из стека. Коды подпрограммы могут храниться в любой области адресного пространства памяти микропроцессорной системы.

Работу с подпрограммой рассмотрим на следующем примере:

```
m1. . . . .
m2: call Asb1   ; вызов подпрограммы по адресу Asb1 с
        ; сохранением в стеке адреса m3 из программного счетчика,
m3. . . . .


; текст подпрограммы с командами сохранения и восстановления
; вектора состояния,
Asb1:   push PSW ; сохранение в стеке аккумулятора и регистра флагов,
Asb2:   push BC  ; сохранение в стеке регистровой пары BC,
Asb3:   push HL  ; сохранение в стеке регистровой пары HL,
  . . . .; команды преобразования данных в подпрограмме,
  . . . .
Asb4:   pop HL ; восстановление из стека регистровой пары HL,
Asb5:   pop BC ; восстановление из стека регистровой пары BC,
Asb6:   pop PSW  ; восстановление из стека аккумулятора
                 ; и регистра флагов,
Asb7:   ret      ; возврат из подпрограммы с восстановлением
                 ; из стека адреса m3; в программном счетчике
```

Выполнение команды m2 приостанавливает дальнейшую работу программы, при изменении содержимого программного счетчика начинает работать подпрограмма с команды Asb1. Завершение подпрограммы командой ret восстанавливает адрес команды m3 в программном счетчике и обеспечивает продолжение работы основной программы. В приведенном примере предполагается использование в подпрограмме текста программы из предыдущего примера, которая в процессе работы изменяет данные в аккумуляторе, регистре флагов и регистровых парах BC, HL.

Процедура прерывания с вызовом подпрограммы также имеет большое значение в операциях ввода-вывода данных и организации взаимодействия с периферийными устройствами. Для реализации прерываний по запросам периферийных устройств предусмотрены специальные сигналы ШУ микропроцессорной системы. Подробнее эти вопросы рассмотрены в следующем разделе.

1.5. Алгоритмы ввода-вывода данных

Важным элементом организации эффективной работы микропроцессорной системы являются процедуры ввода-вывода данных. Выполнение алгоритмов преобразования невозможно без ввода исходных данных через входные интерфейсы и бессмысленно без выдачи полученных результатов через выходные интерфейсы. Различные устройства, с разными свойствами, самостоятельной организацией и собственными внутренним алгоритмами работы могут быть элементами интерфейсов ввода-вывода (рис. 1).

В отличие от памяти микропроцессорной системы, предназначенной только для обеспечения доступа к требуемым данным, интерфейсные средства должны, с одной стороны, работать по сигналам управления микропроцессора, и, с другой стороны, учитывать особенности работы внешних (периферийных) устройств. Поэтому для ввода-вывода данных от периферийных устройств требуются специальные процедуры, обеспечивающие согласование внутренних и внешних алгоритмов работы. Процедуры ввода-вывода в микропроцессорной системе можно свести к трем типам: программный ввод-вывод, ввод-вывод по прерываниям, ввод-вывод в режиме прямого доступа к памяти (режим DMA). Эти процедуры выполняются с помощью специальных средств и специальных сигналов ШУ микропроцессорной системы.

Первая особенность этих процедур – устройства ввода-вывода образуют самостоятельное адресное пространство с собственными сигналами управления, независимыми адресами и командами микропроцессора для доступа к данным. В рассматриваемой микропроцессорной системе (рис. 2) адресация производится только младшим байтом ША, по ШУ предусмотрена передача независимых от управления доступом к памяти сигналов ввода и вывода данных. Однобайтовый адрес (A8) и сигналы управления вводом-

выводом передаются микропроцессором при выполнении команд следующего формата: "in A8" – ввода байта данных из устройства с адресом A8 в микропроцессор, "out A8" – вывод байта данных из микропроцессора в устройство с адресом A8.

Без применения дополнительных средств этими командами можно реализовать только программный ввод-вывод данных. Основная рабочая программа должна предусматривать периодическое выполнение команд ввода-вывода данных для всех используемых периферийных устройств. Период обмена данными должен соответствовать максимально возможной скорости работы периферийных устройств. При несвоевременном обмене данными работа этих устройств может нарушаться с недопустимой потерей данных и некорректным выполнением требуемых функций.

Однако скорость работы периферийных устройств не всегда однозначно известна и зачастую изменяется в довольно широких пределах. Если устройство не готово к обмену данными, возможно формирование специального сигнала ШУ, который переведет микропроцессор в состояние ожидания. Такой способ согласования весьма неэффективен, так как период ожидания может привести к недопустимо большой задержке выполнения других требуемых функций. Возможен и другой вариант организации взаимодействия: перед обменом данными микропроцессор предварительно выполняет программный контроль готовности, при готовности производит программный обмен данными, а при отсутствии готовности переходит к выполнению других функций программы с повторением контроля готовности устройства через определенный интервал времени. В любом из этих вариантов взаимодействия процедуры программного ввода-вывода заметно увеличивают объем функций рабочей программы микропроцессора, замедляют выполнение требуемых функций, не всегда обеспечивают своевременный обмен данными с периферийными устройствами и, следовательно, могут отрицательно повлиять на эффективность работы микропроцессорной системы.

Отказаться от программного контроля состояния периферийных устройств можно в тех случаях, когда эти устройства сами формируют и передают запросы на обработку данных в микропроцессор. В микропроцессорной системе запросы от периферийных устройств называют прерываниями (interrupt), а процедуры с их использованием – вводом-выводом по прерываниям. Такая процедура предполагает прием микропроцессором запроса прерывания, остановку выполнения текущих операций основной программы, вызов подпрограммы обслуживания прерывания с выполнением необходимых операций взаимодействия с устройством, отправившим запрос прерывания, и затем продолжение работы основной программы. Главное достоинство такой организации взаимодействия – отказ от постоянного программного контроля периферийных устройств в основной программе

микропроцессора; подпрограмма обработки прерывания вызывается для работы только тогда, когда возникает необходимость.

Для реализации процедуры прерывания в ШУ микропроцессорной системы предусмотрены специальные сигналы: запрос прерывания INT, поступающий в микропроцессор от устройств ввода-вывода, подтверждение прерывания INTA, формируемый микропроцессором при переходе в режим обработки прерывания. В ряде случаев остановка рабочей программы для обслуживания прерываний недопустима, поэтому в системе команд микропроцессора обычно содержатся команды запрета ("di") и разрешения ("ei") прерываний. Если командой "di" прерывания запрещены, поступающие запросы INT игнорируются и работа основной программы продолжается.

После команды разрешения прерываний "ei" в основной программе каждый запрос INT будет обрабатываться микропроцессором. Завершив рабочий цикл очередной команды, выполнявшейся в момент поступления запроса INT, микропроцессор приостанавливает дальнейшую работу основной программы, переходит в режим обработки прерывания и выдает в ШУ сигнал подтверждения INTA. Обработка прерывания производится по стандартной процедуре вызова подпрограммы, рассмотренной в предыдущем разделе.

Для обработки прерывания устройство ввода-вывода должны передать по ШД код команды "call A16", где A16 – начальный адрес подпрограммы обслуживания прерывания, и прекратить передачу запроса прерывания INT. Организация работы подпрограммы прерывания, выполняющей все необходимые операции взаимодействия с периферийным устройством, практически ничем не отличается от стандартной процедуры вызова подпрограммы. Завершается работа подпрограммы прерывания стандартной командой возврата "ret", которая обеспечивает продолжение приостановленной работы основной программы. Следует помнить, что работа подпрограммы прерывания ничем не отличается от работы основной программы и поступлением нового запроса прерывания может быть приостановлена. Для управления процедурами вызова прерываний в этих подпрограммах также необходимо использовать команды управления "ei" и "di".

Организация обработки прерываний в микропроцессорной системе осложняется тем, что запросы прерываний могут поступать от нескольких разных периферийных устройств. Во-первых, каждое устройство требует вызова индивидуальной подпрограммы прерывания для облуживания запроса, во-вторых, запросы от разных устройств могут поступать одновременно, и требуется арбитраж для определения очередности обслуживания. Эти функции обычно выполняют специальные устройства, называемые контроллерами прерываний [1].

Контроллер прерываний является стандартным периферийным устройством, обеспечивает прием запросов прерываний от нескольких других устройств, выполняет арбитраж поступивших запросов, формирование об-

щего сигнала запроса INT и команды вызова индивидуальной для каждого запроса подпрограммы прерывания. Индивидуальные адреса подпрограмм прерываний могут указываться программно в процедурах инициализации [1], а при другой организации микропроцессорных систем могут задаваться фиксировано в виде специальных таблиц векторов прерываний. Контроллер прерываний, не являясь самостоятельным устройством и выполняя указанные выше вспомогательные функции, практически всегда необходим для эффективной организации процедур ввода-вывода.

Ввод-вывод данных по прерываниям, исключая из основной программы микропроцессора целый ряд операций по управлению взаимодействием с периферийными устройствами, позволяет существенно увеличить скорость обработки данных, уменьшить время реакции на запросы периферийных устройств, расширить набор реализуемых функций и, следовательно, повысить эффективность работы микропроцессорной системы в целом. Общим же в рассмотренных процедурах ввода-вывода является обмен данными между любыми устройствами через микропроцессор.

Во всех операциях обмена данными микропроцессор считывает из памяти команды пересылки данных и, выполняя эти команды, производит передачу данных: байт данных по первой команде поступает в регистр микропроцессора и только по следующей команде из микропроцессора передается в другое устройство. При передаче массивов данных такой алгоритм обмена неэффективен, для передачи каждого байта требуется чтение из памяти и выполнение нескольких команд микропроцессора. Для ускорения обмена данными предусмотрена процедура прямого доступа к памяти – режим DMA. В этом режиме байты данных передаются непосредственно между памятью микропроцессорной системы и устройствами ввода-вывода без участия микропроцессора.

В режиме DMA управление процедурой с формированием управляющих сигналов и адресов производится контроллером DMA [1], который реализует необходимые функции не программно, как микропроцессор, а аппаратно. Контроллер DMA, следовательно, является специализированным устройством с ограниченным набором аппаратно реализуемых функций. Благодаря этой особенности передача данных в режиме DMA выполняется в несколько раз быстрее, чем микропроцессором.

Перед запуском процедура DMA требует инициализации: необходимо указать начальный адрес памяти для обмена данными, количество циклов обмена и направление передачи данных. Кроме того, контроллер DMA может использовать для обмена данными только те сигналы, которые в основных режимах работы формирует микропроцессор. Поэтому на период выполнения процедуры DMA необходимо работу микропроцессора приостановить и передать функции формирования требуемых сигналов контроллеру DMA. Для остановки микропроцессора при запуске режима DMA в ШУ предусмотрены специальные сигналы. Эти сигналы, поступающие че-

рез контроллер DMA, обрабатываются микропроцессором по алгоритмам, аналогичным алгоритмам обработки запросов прерываний.

Передача данных в режиме DMA выполняется следующим образом:

- запрос от периферийных устройств поступает через контроллер DMA в микропроцессор;
- если прерывание разрешено, микропроцессор приостанавливает работу и подтверждает право на управление работой ШД, ША и ШУ контроллеру DMA передачей специальных сигналов разрешения;
- контроллер, принимая управление работой системы, производит ускоренный обмен данными между памятью и периферийным устройством по параметрам, которые были заданы микропроцессором программно перед запуском процедуры;
- после завершения обмена данными контроллер DMA возвращает управление микропроцессору, который продолжает выполнение приостановленной программы.

Особенности режима DMA заключаются в том, что обмен данными выполняется с большей скоростью, но требует предварительной программной настройки. Применение этого режима эффективно, если требуется передать подготовленный массив данных относительно большого объема с максимальной скоростью, используя единые параметры настройки обмена данными.

Организация ввода-вывода является важнейшим элементом эффективной работы для микропроцессорной системы. Рассмотренные процедуры взаимодействия с периферийными устройствами обладают разными свойствами, обеспечивают разную организацию ввода-вывода данных и в соответствии с определенными условиями могут удовлетворять требованиям различных задач преобразования данных. Несмотря на вспомогательную роль операций ввода-вывода в работе самой микропроцессорной системы, очень часто в решаемых микропроцессорными средствами задачах эти операции имеют первостепенное значение и поэтому требуют самого серьезного внимания. Достаточно типичны ситуации, когда алгоритмы преобразования данных заметно проще алгоритмов организации ввода-вывода, когда микропроцессорная система выполняет функции ведомого средства, работая под управлением команд, поступающих через интерфейсы ввода-вывода, и т.п.

Рассмотренная в первой главе организация микропроцессорной системы показывает основные свойства и возможности программной реализации алгоритмов преобразования. Корректное использование особенностей программной реализации позволяет получать весьма эффективные решения в самых различных применениях. Важнейшее свойство микропроцессорных средств – возможность качественной перестройки алгоритмов работы без изменения структуры и набора применяемых аппаратных средств.

2. МИКРОКОНТРОЛЛЕРЫ AVR

Микроконтроллеры являются весьма универсальным средством, могут применяться и применяются для решения самых разнородных задач. Их универсальность определяется особенностями микропроцессорной организации обработки данных, рассмотренными в первой главе. Функциональные возможности многих микроконтроллеров, в том числе и их аппаратных средств, ориентированы на реализацию алгоритмов управления. Весьма существенно то, что микроконтроллеры относятся к числу массовых, недорогих и доступных интегральных схем, номенклатура микроконтроллеров постоянно расширяется.

Среди выпускаемых в настоящее время микроконтроллеров выделяются микроконтроллеры семейства AVR фирмы ATMEL. Эти микроконтроллеры обладают низким уровнем потребления, невысокой стоимостью при весьма значительных функциональных возможностях, высоким быстродействием и возможностью многократной перезаписи программ. Хотя и аналогичные по характеристикам микроконтроллеры выпускаются многими фирмами, по общему комплексу свойств семейство AVR одно из наиболее эффективных в классе недорогих 8-разрядных микроконтроллеров. Основная область применения таких микроконтроллеров в системах автоматизации – реализация в реальном масштабе времени алгоритмов управления, не требующих сложных вычислительных процедур и временем реакции от единиц миллисекунд и более.

Естественно, что микроконтроллеры AVR не являются универсальным средством "на все случаи". При определенных требованиях может быть необходимым применение значительно более дорогих 16- или 32-разрядных микроконтроллеров, процессоров цифровой обработки сигналов, программируемых логических интегральных схем и т.п. Тем не менее, существует большое число разнообразных задач, при решении которых недорогие и простые в применении 8-разрядные микроконтроллеры вне конкуренции.

Очевидно, что для построения системы автоматического управления одного микроконтроллера недостаточно. Допустимый диапазон изменения входных и выходных сигналов, особенности интерфейсов ввода-вывода микроконтроллера, как правило, требуют различных дополнительных средств управления, устройств сопряжения и индикации, с помощью которых можно обеспечить управление различными объектами (электроприводы постоянного и переменного токов, системы стабилизации температуры, системы контроля параметров и т.п.). Состав и характеристики этих дополнительных средств определяются свойствами конкретных объектов управления и используемыми алгоритмами работы, вопросы проектирования этих дополнительных устройств выходят за рамки данного учебного пособия и изучаются в соответствующих курсах.

Данный раздел подготовлен на основе информационных материалов по микроконтроллерам семейства AVR, распространяемых фирмой ATMEL, и является кратким справочным руководством по применению микроконтроллера AT90S8535. На примере этого микроконтроллера рассмотрены особенности организации и архитектуры семейства AVR, параметры и режимы работы аппаратных средств. В следующих главах будут показаны особенности системы команд семейства AVR, средства подготовки программ на ассемблере и примеры программ для реализации типовых процедур. Знание особенностей применения микроконтроллера семейства AVR позволяет достаточно легко перейти к изучению вопросов применения любого другого микроконтроллера этого класса.

2.1. Общая характеристика микроконтроллеров семейства AVR

Микроконтроллеры фирмы ATMEL с усовершенствованной RISC-архитектурой обладают эффективными программно-аппаратными ресурсами для решения различных задач. Семейство микроконтроллеров AVR содержит и простые модели (AT90S1200, AT90S2313) с минимумом необходимых ресурсов, и весьма сложные модели megaAVR с существенно увеличенным объемом памяти, количеством портов ввода-вывода и других средств. Высокая эффективность микроконтроллеров AVR обеспечивается развитой системой команд, выполняющихся, как правило, за один рабочий такт, аппаратной реализацией многих стандартных функций (таймеры, модуляторы ШИМ, параллельные и последовательные порты ввода-вывода, компаратор, АЦП и др.) и возможностью внутрисистемного программирования, т.е. записи программ и данных в память микроконтроллера непосредственно в схеме работающего устройства.

Общие архитектурные особенности и программная совместимость микроконтроллеров AVR позволяют использовать одни и те же алгоритмы и рабочие программы на разных моделях. Единственным ограничением их применимости может служить только отсутствие необходимых для исполнения программ аппаратных средств в более простых моделях микроконтроллеров.

Архитектура и общая организация микроконтроллеров AVR достаточно типична для микропроцессорных средств, обладает необходимой универсальностью для программной реализации различных алгоритмов и в общих чертах аналогична описанной в главе 1. Важным дополнением являются средства аппаратной реализации стандартных интерфейсных функций и соответствующие алгоритмы и процедуры обеспечения эффективного взаимодействия с ними. Определенные отличия в других деталях архитектуры и организации работы, связанные, как правило, с повышением эффективности используемых средств и спецификой реализуемых алгоритмов, будут описаны ниже.

Существенной общей особенностью всего семейства AVR является использование 32 регистров общего назначения и гарвардской архитектуры с тремя раздельными адресными пространствами: памяти программ (FLASH), оперативной памяти данных (RAM), программируемой постоянной памяти данных (EEPROM). FLASH и EEPROM являются энергонезависимыми и, как обычно, сохраняют данные при отсутствии питающих напряжений. RAM – это стандартная энергозависимая оперативная память. Система команд поддерживает стандартные операции с однобайтовыми данными, возможны определенные операции с двухбайтовыми словами и отдельными битами. Каждый из 32 восьмиразрядных регистров общего назначения может служить регистром-аккумулятором. Основной формат кодов команд – 2 байта, формат данных – 1 байт.

Управление и доступ к дополнительным аппаратным средствам микроконтроллеров производятся с помощью специальных управляющих регистров – регистров ввода-вывода. Эти регистры определяют параметры и режимы работы устройств микроконтроллеров, обеспечивают необходимый обмен данными с ними. Каждое устройство аппаратной реализации содержит индивидуальный набор регистров ввода-вывода, которые и являются основными средствами обеспечения взаимодействия. Взаимодействие между функциями, реализуемыми программно и аппаратно, производится формированием запросов прерываний. Система прерываний позволяет обслуживать программные прерывания, внутренние прерывания всех устройств микроконтроллеров через регистры ввода-вывода и внешние прерывания. Внутренние аппаратные и внешние прерывания фиксируются в таблице векторов прерываний со строго закрепленными адресами.

Общие особенности микроконтроллеров всего семейства AVR позволяют более детальное их изучение проводить для одной из моделей семейства. В дальнейшем работа микроконтроллеров AVR будет рассматриваться на примере классической модели AT90S8535 со средними программно-аппаратными ресурсами.

2.2. Микроконтроллер AT90S8535

Микроконтроллер **AT90S8535** (рис. 4) со 118 командами в системе команд содержит 8-разрядное арифметико-логическое устройство (АЛУ), память программ (FLASH) объемом 8 Кбайт, электрически программируемое ПЗУ (EEPROM) объемом 0,5 Кбайт (также с возможностью внутрисистемного программирования), статическое ОЗУ (RAM) объемом 0,5 Кбайт, 32 регистра общего назначения, 4 двунаправленных параллельных восьмиразрядных порта ввода-вывода, последовательный синхронный интерфейс SPI, последовательный асинхронный интерфейс UART, два восьмиразрядных и один шестнадцатиразрядный таймеры с возможностью реализации модуляторов ШИМ, сторожевой таймер с автономным генератором, аналоговый компаратор, восьмиканальный АЦП.

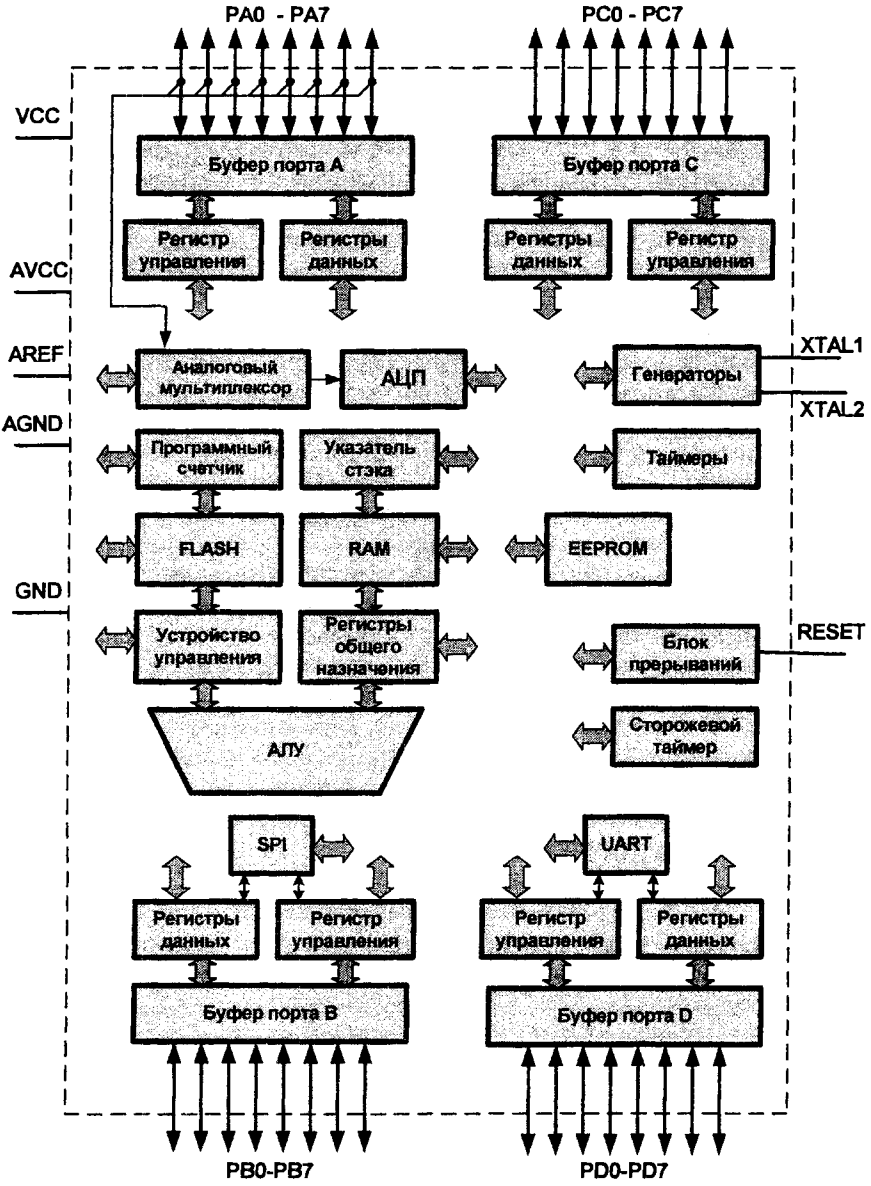

Рис. 4. Структурная схема микроконтроллера AT90S8535

Структурная схема (рис. 4) отображает только логическую организацию микроконтроллера и его основные средства программной и аппаратной реализации требуемых функций. Средства программной реализации аналогичны рассмотренным в главе 1, хотя и имеют определенные особенности. Все остальные устройства, показанные на рис. 4, предназначены для аппаратной реализации различных интерфейсных функций и будут рассмотрены в следующей главе.

АЛУ микроконтроллера выполняет стандартный набор операций преобразования данных однобайтового формата. Практически все эти операции выполняются за один такт микроконтроллера. В некоторых других моделях микроконтроллеров семейства AVR дополнительно содержится умножитель, выполняющий умножение аппаратно с формированием двухбайтового произведения за два такта. Регистр флагов, не показанный на рис. 4, также имеет однобайтовый формат, в микроконтроллере рассматривается, как один из регистров ввода-вывода, и в данной структуре называется регистр SREG.

Количество регистров общего назначения однобайтового формата увеличено до 32, регистры r0 – r31. Существенные особенности этих регистров: каждый регистр может выполнять функции аккумулятора, для регистровой адресации шесть последних регистров общего назначения r26 – r 31 могут объединяться в регистровые пары X, Y и Z соответственно.

Так как память разделена на три независимых области – FLASH, RAM и EEPROM, каждая из этих областей имеет разное функциональное назначение, обладает самостоятельной системой и средствами адресации, требует применения разных команд доступа. Подробнее организация памяти рассмотрена в следующем разделе.

С точки зрения логической организации микроконтроллера каждое из устройств аппаратной реализации интерфейсных функций можно рассматривать как определенный набор регистров ввода-вывода, изменение данных в этих регистрах и является средством координации и управления их работой.

В контроллере предусмотрены программируемые режимы снижения потребляемой мощности (**sleep mode**) с отключением отдельных устройств. В режиме **idle** процессор микроконтроллера остановлен, все остальные устройства работают, и их внутренние аппаратные прерывания или внешние прерывания переводят микроконтроллер в активный режим немедленно. В режиме **power-down** процессор и тактовый генератор остановлены и, следовательно, не работают все устройства микроконтроллера, тактируемые основным генератором, микроконтроллер переводится в активный режим сигналами внешних прерываний, сторожевым таймером или сигналом RESET. Длительность внешних сигналов для надежного запуска тактового генератора должна превышать требуемую длительность сигнала RESET. В режиме **power-save** в отличие от **power-down** запуск микроконтроллера

дополнительно может производиться прерываниями таймера 2, если в этом таймере задан асинхронный режим работы от вспомогательного генератора.

На структурной схеме (рис. 4) и в дальнейшем при описании микроконтроллеров будут использованы названия и обозначения, используемые фирмой ATMEL. Внешние выводы (рис.4) микроконтроллера:

- VCC и GND (общий) – источник питания цифровых элементов;
- AVCC, AGND (общий аналоговый), AREF – питание и опорное напряжение АЦП и его мультиплексора;
- RESET – сигнал внешнего сброса (низкий уровень длительностью более 50 нс), при включении питания сброс микроконтроллера производится автоматически;
- XTAL1 и XTAL2 – соответственно вход и выход тактового генератора (для подключения частотозадающего кварцевого резонатора и общей синхронизации с другими устройствами), аналогичные электроды вспомогательного генератора асинхронного режима таймера 2 – выводы PC6 и PC7;
- PA0-PA7, PB0-PB7, PC0-PC7, PD0-PD7 – 32 линии ввода-вывода, объединены в 4 восьмиразрядных порта (PORTA, PORTB, PORTC, PORTD).

PORTA, PORTB, PORTC, PORTD могут использоваться как стандартные двунаправленные порты ввода-вывода либо для передачи сигналов других устройств микроконтроллера. Альтернативные функции PORTA: передача аналоговых сигналов через мультиплексор на вход АЦП. Альтернативные функции PORTB: PB0 и PB1 – внешние входы T0 и T1 таймеров 0 и 1 соответственно, PB2 и PB3 – входы AIN0 и AIN1 аналогового компаратора, остальные – сигналы синхронного последовательного интерфейса SPI (PB4 – SS, PB5 – MOSI, PB6 – MISO, PB7 - SCK). Альтернативные функции PORTC: PC6 и PC7 – вход и выход вспомогательного генератора таймера 2. Альтернативные функции PORTD: PD0 и PD1 – сигналы RXD и TXD асинхронного последовательного интерфейса UART соответственно, PD2 и PD3 – сигналы внешних прерываний INT0 и INT1, PD4-PD7 – сигналы OC1B, OC1A, ICP, OC2 таймеров 1 и 2.

2.3. Запоминающие устройства микроконтроллера AT90S8535

Как указывалось ранее, запоминающие устройства микроконтроллера – FLASH, RAM, EEPROM, регистры общего назначения и регистры ввода-вывода образуют три отдельных адресных пространства (рис. 5).

Рис. 5. Память микроконтроллера AT90S8535

FLASH – постоянное запоминающее устройство и предназначено для хранения кодов программы, адресация в этой области памяти производится программным счетчиком. Изменение адреса FLASH в программном счетчике производится в соответствии со стандартной организацией микропроцессорной системы. Как и обычно, для изменения адреса в программном счетчике можно применять команды управления микроконтроллера (см. главу 1). FLASH можно использовать и для хранения констант, чтение этих констант производится специальной командой "lpm" с регистровой адресацией через регистр Z.

FLASH объемом 8 Кбайт предназначена для хранения кодов программ и констант, в том числе и при выключенном питающем напряжении, и состо-

ит из 4 К двухбайтовых ячеек с адресами $000 – $FFF*. Запись данных производится специальным программатором в процедуре внутрисистемного программирования, допустимое количество циклов перезаписи не менее 1000. Большинство кодов команд имеют 16-битовый формат, поэтому, как правило, одна ячейка флэш-памяти хранит код 1 команды или две 8-битовых константы.

В памяти данных EEPROM образует отдельное адресное пространство объемом 0,5 К однобайтовых ячеек с адресами $000 – $1FF и хранением данных при выключении питающего напряжения. Доступ к данным в EEPROM производится специальной процедурой с особым алгоритмом адресации через соответствующие регистры из файла регистров ввода-вывода. Эта процедура по организации ближе к процедурам ввода-вывода (глава 1) и будет рассмотрена позднее, программно доступно и чтение, и запись данных. Запись данных также может производиться программатором в процедуре внутрисистемного программирования.

Третье адресное пространство является основной рабочей областью памяти данных и включает 32 регистра общего назначения с адресами $000 – $01F, 64 регистра ввода-вывода с адресами $020 – $05F и, собственно, RAM объемом 0,5 К однобайтовых ячеек с адресами $060 - $25F. Такая организация памяти данных позволяет адресовать все регистры как ячейки RAM с известными адресами либо использовать для их адресации соответствующие обозначения или адреса в регистровых файлах. Например, регистр управления АЦП имеет адрес RAM $026, адрес в файле регистров ввода-вывода $06 или символическое имя ADCSR.

Адресация ячеек памяти в третьем адресном пространстве производится стандартными способами. При прямой адресации адрес содержится в командах **sts k, Rr** и **lds Rd, k** (см. главу 4), где k – адрес в диапазоне $000-$25F, Rr и Rd любые из регистров общего назначения. Косвенная адресация возможна регистрами **X, Y, Z**. Каждый из этих адресных регистров образован парой регистров общего назначения: **X – r27-r26, Y – r29-r28, Z – r31-r30**. Также возможна стековая адресация с помощью указателя стека (пара регистров **SPH-SPL** файла регистров ввода-вывода), обычно в качестве вершины стека используется наибольший адрес ОЗУ с символическим именем RAMEND, для данного микроконтроллера – это физический адрес $25F. Данные в командах с непосредственной адресацией являются элементами кодов программы и вместе с ними хранятся в FLASH.

Адресация регистров общего назначения стандартная: в команде указывается номер регистра (например, **mov r2, r0** – передать байт данных из r0 в r2) или присвоенное ему в программе символическое имя (например, **mov**

---

* Символ **$**, как и символ **0x** применяется для обозначения шестнадцатеричного формата данных

r2, temp, где temp – символическое имя регистра). Обращение к файлу регистров ввода-вывода производится командами **in, out** (например, out SPL, r16 – передать байт данных из регистра r16 в регистр ввода-вывода SPL) с использованием, как правило, символических имен (символические имена регистров ввода-вывода приведены в приложении).

Необходимо обратить внимание не только на отличия в командах адресации регистров, но и на их совершенно разное функциональное назначение. Регистры общего назначения, как и в любой микропроцессорной системе, содержат данные для программного преобразования. Следовательно, команды преобразования данных выполняются только над содержимым регистров общего назначения. Дополнительная особенность микроконтроллера – каждый регистр общего назначения может быть аккумулятором при выполнении любых команд преобразования данных. Система команд и организация программной обработки данных микроконтроллеров семейства AVR достаточно типична для микропроцессорных средств и будет рассмотрена позднее.

С точки зрения программного преобразования регистры ввода-вывода, как правило, доступны только для операций чтения и записи. Байты данных в регистрах ввода-вывода строго связаны с работой соответствующих аппаратных интерфейсных средств. Поэтому обмен данными с этими регистрами производится либо для программного управления функциями и режимами аппаратных интерфейсных средств, либо для приема от них данных, сформированных в результате выполнения заданных функций.

Функции, выполняемые комплексом аппаратных интерфейсных средств, являются важнейшими характеристиками микроконтроллера и в значительной степени определяют эффективность его работы. Особенности применения регистров ввода-вывода для организации взаимодействия с аппаратными интерфейсными средствами рассматриваются в следующей главе.

3. АППАРАТНЫЕ ИНТЕРФЕЙСЫ МИКРОКОНТРОЛЛЕРА AT90S8535

Дополнительные аппаратные средства микроконтроллеров, реализующие стандартные интерфейсные функции, позволяют существенно расширить возможности программной обработки данных. Возможность решения нескольких задач параллельно с программным преобразованием данных не только повышает скорость работы, но и заметно упрощает алгоритмы и рабочие программы. Следует помнить, что любые средства аппаратной реализации обладают весьма ограниченными пределами изменения алгоритмов и параметров работы, их функции строго специализированы, поэтому они и должны рассматриваться как вспомогательные средства.

Хотя основные свойства микропроцессорных средств определяются программной реализацией функций, их эффективность может сильно зависеть от набора вспомогательных средств. Наиболее стандартизованы различные интерфейсные функции, кроме того, интерфейсы в системах автоматизации являются важнейшим элементом обеспечения корректного взаимодействия. Именно по этим причинам вспомогательные аппаратные средства микроконтроллеров предназначены в первую очередь для выполнения стандартных интерфейсных функций.

Интерфейсные средства микроконтроллера AT90S8535 достаточно типичны и поддерживают функции ввода-вывода в параллельном и последовательном форматах, обработки аналоговых сигналов, формирования заданных временных интервалов. В данной главе рассматриваются режимы работы этих средств и вопросы применения регистров ввода-вывода для организации взаимодействия с ними. Используемые для пояснений фрагменты программ ориентированы на мнемонику и систему команд микроконтроллера AT90S8535, которая рассматривается детальнее в главе 4.

3.1. Параллельные порты ввода-вывода

Как указывалось ранее, микроконтроллер содержит 32 линии ввода-вывода (рис. 4), объединенные в 4 двунаправленных порта (A, B, C, D). Управление каждым портом производится тремя регистрами порта из файла регистров ввода-вывода, символические имена этих регистров содержат наименование порта. Так как процедуры управления для всех портов аналогичны, рассмотрим их на примере порта A.

Регистр управления порта A с символическим именем **DDRA**, адресом в файле регистров ввода-вывода $1A, адресом RAM $03A программно доступен и для чтения, и для записи и определяет направление передачи дан-

ных: **0 – ввод, 1 – вывод**. Каждый бит DDRA* (DDA0 – DDA7) управляет соответствующей линией ввода-вывода и программируется независимо, начальное значение всех битов DDRA – нулевое. То есть в процессе работы значение каждого бита не зависит от значений других битов, чтением содержимого регистра DDRA можно определить направление передачи данных по соответствующим линиям ввода-вывода в данный момент времени.

Пример 1:

```
m1:  ldir16, 0b00011110   ; запись в r16 константы, указанной
                          ; в двоичном формате,
m2:  out   DDRA, r16      ; пересылка из r16 в регистр управления DDRA.
```

В соответствии с байтом управления в регистре DDRA линии PA7, PA6, PA5, PA0 будут работать в режиме ввода, а PA4, PA3, PA2, PA1 – в режиме вывода сигналов.

Регистр ввода данных **PINA** (адрес в файле регистров ввода-вывода – $19, адрес RAM – $039) программно доступен только для чтения (**in r16, PINA** – пример команды чтения данных из PORTA) и обеспечивает считывание сигналов, поступающих в данный момент времени на соответствующие линии ввода-вывода (например, линия PA5 в режиме ввода формирует бит PINA5 регистра ввода данных). Никакого хранения данных регистр PINA не выполняет.

Регистр вывода данных **PORTA** (адрес в файле регистров ввода-вывода - $1B, адрес ОЗУ – $03B) программно доступен и для чтения, и для записи, обеспечивает хранение данных и выдачу их в режиме вывода на соответствующие линии ввода-вывода (например, PORTA4 – PA4). При чтении PORTA передает данные, ранее записанные в этот регистр для вывода.

Пример 2:

```
m3:  ldir16, 0b01010101   ; запись в r16 константы, указанной
                          ; в двоичном формате,
m4:  out   PORTA, r16     ; пересылка из r16 в выходной регистр PORTA,
m5:  in r15, PINA         ; пересылка из входного регистра PINA в r16.
```

Если полагать, что режим порта А задан в примере 1, то PA4=1, PA3=0, PA2=1, PA1=0 и определяются соответствующими битами константы,

---

* В дальнейшем будут использоваться только символические имена регистров, в программах на ассемблере эти стандартные имена определяются специальным файлом с расширением .inc. Такие файлы существуют для каждой модели микроконтроллера (например, "8535def.inc") и включаются в тексты программ. Таблица соответствия символических имен и физических адресов приведена в приложении.

записанной командами m3, m4 в выходной регистр порта. Чтение байта из входного регистра (команда m5) в этих битах, настроенных в режим вывода, даст такие же значения сигналов; в остальных битах, настроенных в режим ввода, будет определяться поступающими на них внешними сигналами.

Буферы портов в режиме вывода позволяют формировать логические сигналы с током нагрузки до 20 мА. В режиме ввода, кроме приема внешних сигналов, буферы позволяют подключать к входам портов внутренние резисторы (pull-up), задавая на входах при отсутствии внешних сигналов уровень логической единицы. Например, DDA7=0 (ввод данных) при PORTA7=1 внутренний резистор буфера подключен и при отсутствии внешнего сигнала будет задавать на PA7 высокий уровень. Это позволяет устранить возможные неопределенности входных сигналов, в частности, при отключении внешних источников сигналов. При появлении сигнала внешнего источника уровень PA7 будет задаваться этим внешним сигналом. При PORTA7=0 внутренний резистор отключен и никакого влияния не оказывает.

Для остальных портов ввода-вывода используются такие же регистры с аналогичными именами и заменой буквы, обозначающей наименование порта. Логическая организация параллельных портов достаточно проста и очевидна: пересылка данных в регистры ввода-вывода любого порта отображается на его работе в течение одного такта микроконтроллера. К их существенным свойствам можно отнести как полную взаимную независимость портов, так и полную независимость отдельных линий ввода-вывода: функции каждой линии ввода-вывода можно программировать и использовать абсолютно независимо. Следует учитывать, что другие интерфейсные средства формируют сигналы через эти же линии ввода-вывода. Передачу этих сигналов в дальнейшем будем называть альтернативными функциями портов.

При использовании альтернативных функций портов ввода-вывода в регистрах управления должны задаваться соответствующие этим функциям направления передачи сигналов. Очевидно, что одновременное программирование нескольких функций для одной и той же линии ввода-вывода невозможно. Более детальное описание альтернативных функций приведено далее при описании соответствующих аппаратных средств микроконтроллера.

3.2. Последовательный интерфейс SPI

Последовательный интерфейс SPI обеспечивает синхронный ввод-вывод данных в последовательном формате через линии ввода-вывода порта B сигналами **SCK** (альтернативная функция линии ввода-вывода PB7), **MOSI** (альтернативная функция линии PB5), **MISO** (альтернативная функция

PB6), SS (альтернативная функция PB4). Контроллер при обмене данными может работать в режиме ведущий (master) или ведомый (slave). Структурная схема, поясняющая алгоритмы работы интерфейса, приведена на рис. 6. Этот же интерфейс используется для внутрисистемного программирования микроконтроллера с записью данных в FLASH и EEPROM.

SPI-master управляет обменом данных. Он формирует информационную 8-битовую последовательность на выходе MOSI, стробирующую выдачу данных, последовательность тактовых импульсов на выходе SCK и одновременно может принимать на входе MISO 8-битовую последовательность, стробируемую импульсами SCK. Если PB4 конфигурируется как выход (DDB4=1), сигнал SS не используется, и PB4 может работать в стандартном режиме вывода. Если PB4 конфигурируют как вход (DDB4=0), при SS=1 продолжается работа в режиме SPI-master. Сигнал SS=0, поступающий от другого устройства, должен рассматриваться как запрос на переход в режим SPI-slave для приема данных.

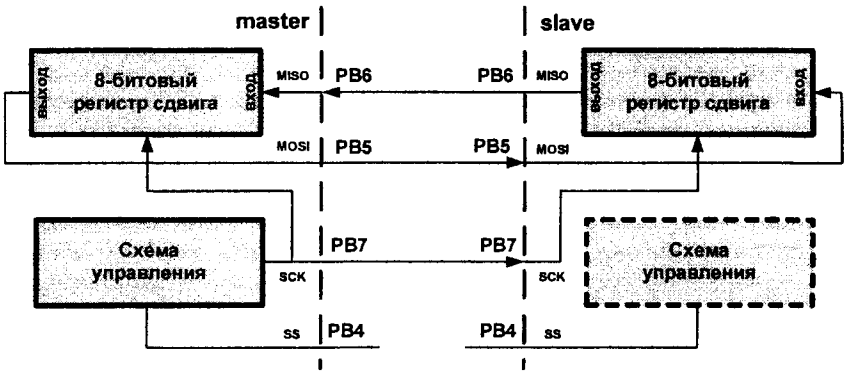

Рис. 6. Интерфейс SPI

SPI-slave управляется последовательностью импульсов на входе SCK и принимает информационную 8-битовую последовательность, подаваемую на его вход MOSI. Параллельно с приемом данных по этим же стробирующим импульсам SCK может формироваться выходная информационная последовательность на выходе MISO. Для этого режима PB4 должен конфигурироваться входом. При SS=0 реализуется режим SPI-slave, если SS=1, интерфейс переходит в пассивное состояние и перестает работать.

Для режима SPI-master конфигурирование порта B следующее: MISO – вход, MOSI – выход, SCK – выход, SS – вход (как вход используется только при необходимости), для режима SPI-slave: MISO – выход, MOSI – вход, SCK – вход, SS – вход (задается сигнал логического нуля для работы ин-

терфейса). Обмен данными между SPI-master и SPI-slave производится по алгоритму кольцевого регистра сдвига (рис. 6), по каждому такту SCK данные сдвигаются на один бит, после 8 тактов содержимое регистра-master и регистра-slave меняется местами. Соединение MOSI – MOSI обеспечивает передачу от SPI-master к SPI-slave, а соединение MISO – MISO используется при необходимости передачи данных в обратном направлении.

Управление интерфейсом SPI производится тремя регистрами файла регистров ввода-вывода: регистром данных – **SPDR**, регистром управления – **SPCR**, регистром состояния – **SPSR.** SPDR и SPCR программно доступны и для чтения, и для записи, SPSR доступен только для чтения. Логическая организация этого интерфейса, как и других интерфейсных средств, рассматриваемых далее, существенно сложнее, чем в параллельных портах ввода-вывода. Передача данных выполняется не за один такт микроконтроллера, а требует формирования целой последовательности сигналов. Поэтому необходимо согласование и логического взаимодействия, и временного взаимодействия с другими реализуемыми функциями.

В главе 1 было показано, что наиболее успешно эти задачи решаются применением процедур ввода-вывода по прерываниям. Практически все интерфейсные устройства микроконтроллера содержат средства поддержки для выполнения процедур прерываний: индивидуально формируемые запросы прерываний, соответствующая им таблица векторов прерываний, флаги разрешения и запрета обработки прерываний, аппаратно реализуемая процедура вызова прерываний. Более полно особенности системы прерываний микроконтроллера будут рассмотрены в одном из последующих разделов. В описании же работы каждого из интерфейсных средств, в том числе и интерфейса SPI, будут упоминаться только необходимые элементы системы прерываний.

SPDR служит для записи передаваемых данных и чтения данных, которые поступили в регистр сдвига при обмене. SPCR содержит 8 бит управления интерфейсом (слева в таблице символических имен старший бит).

Символические имена битов управления регистра SPCR

SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0

SPIE – бит разрешения прерывания SPI, 0 запрещает прерывания.

SPE – бит разрешения работы; при 0 запрещены любые операции в SPI, 1 разрешает работу интерфейса.

DORD – при 1 первым передается младший бит слова данных, при 0 – старший бит.

MSTR – при 1 определяется режим SPI-master, при 0 – режим SPI-slave.

Младшие 4 бита SPCR определяют параметры тактового сигнала SCK:

CPOL – определяет пассивный уровень сигнала SCK в перерывах передачи данных, т.е. при 0 тактовый сигнал в пассивном состоянии интерфейса тоже нулевой.

CPHA – при 0 запись данных должна производиться каждым первым фронтом сигнала SCK после пассивного уровня, при 1 – каждым вторым фронтом сигнала после пассивного уровня.

SPR1 и **SPR0** задают частоту сигнала SCK; коэффициент деления тактовой частоты микроконтроллера для интерфейса определяется этими битами следующим образом:

00 – коэффициент деления 4,
01 – коэффициент деления 16,
10 – коэффициент деления 64,
11 – коэффициент деления 128.

В регистре SPSR используется только 2 старших бита. **SPIF** (бит 7) – флаг прерывания, **WCOL** (бит 6) – флаг коллизии. SPIF устанавливается в 1 после каждого цикла передачи данных или после отмены режима SPI-master сигналом SS. SPIF вызывает **вектор прерывания SPI_STC (адрес вектора $00a)**, если бит SPIE=1 (разрешение прерывания в SPCR) и установлен флаг глобального прерывания I в регистре состояния SREG*. Флаг прерывания очищается автоматически при вызове вектора прерывания SPI_STC либо одновременно с очисткой флага WCOL. Флаг коллизии устанавливается в случае чтения регистра SPDR в период передачи данных в SPI (некорректное чтение данных) и автоматически очищается одновременно с флагом SPIF после чтения регистра SPSR и последующего обращения к регистру SPDR.

Если биты в регистре управления SPCR для выбора необходимого режима заданы, запись байта данных в регистр SPDR микроконтроллера в режиме SPI-master приводит к началу рабочего цикла интерфейса. SPI-master (рис. 6) передает на MOSI данные, с входа MISO может записывать данные от SPI-slave и на выходе SCK формирует 8 импульсов, управляющих передачей байта данных. Под управлением этих же сигналов SCK SPI-slave (рис. 6) принимает данные с входа MOSI и может передавать из своего регистра SPDR данные на выход MISO.

После завершения рабочего цикла обмена данными в микроконтроллере, работающем в любом режиме, устанавливается флаг прерывания для вызова вектора прерывания SPI_STC. Подпрограмма обработки этого вектора должна выполнять операции доступа к регистрам SPI для реализации необходимых функций интерфейса. Например, чтение из SPDR, поступивших в предыдущем цикле данных, и запись в него новых данных для передачи в следующем цикле. Управление битом разрешения прерываний SPIE позволяет отказаться от вызова вектора прерывания SPI_STC и производить

---

* регистр состояния SREG будет описан позднее

только программный ввод-вывод данных с программным опросом состояния интерфейса SPI. К микроконтроллеру SPI-master для обмена данными вместо второго контроллера может быть подключен и обычный регистр сдвига разрядности, соответствующей формату данных.

Таким образом, интерфейс SPI обеспечивает за один рабочий цикл и передачу байта данных через SPDR на выход, и запись в этот же регистр нового байта данных, поступившего на вход. Запуск рабочего цикла производится записью в SPI-master очередного байта данных в регистр SPDR. До первого рабочего цикла параметры работы интерфейса должны быть указаны записью соответствующего байта управления в регистр SPCR. Команды обмена данными с регистрами интерфейса SPI аналогичны командам, рассмотренным в примерах работы с параллельными портами. Программы управления работой интерфейса SPI рассмотрены в примерах главы 5.

3.3. Последовательный интерфейс UART

Интерфейс UART обеспечивает полнодуплексную передачу данных через линии ввода-вывода порта D сигналами: вход приемника – **RxD (альтернативная функция PD0)** и выход передатчика – **TxD (альтернативная функция PD1).** Приемник и передатчик интерфейса работают независимо друг от друга в асинхронном режиме со стандартными скоростями передачи данных и стандартным форматом сообщения (более подробную информацию можно найти в описании любого асинхронного приемопередатчика, например, К580ВВ51 [1] или интерфейса RS-232C). Длина одного сообщения – 8 или 9 бит, минимальная скорость передачи – 2400 бит/с., другие стандартные скорости кратны минимальной (например, 4800, 9600, 14400 и т.д.).

Управление интерфейсом UART производится 4 регистрами из файла регистров ввода-вывода: **UDR** – регистр данных, **UCR** – регистр управления, **USR** – регистр состояния, **UBRR** – регистр скорости передачи данных. Все регистры интерфейса программно доступны и для чтения, и для записи. Основные алгоритмы управления интерфейсом предполагают применение процедур ввода-вывода по прерываниям. Таблица векторов прерываний микроконтроллера включает соответствующие вектора прерываний. Обработку и формирование других сигналов интерфейса кроме RxD и TxD необходимо производить программно в подпрограммах векторов прерываний.

Регистр данных UDR физически состоит из двух независимых регистров; при записи в UDR данные поступают в регистр передачи интерфейса, при чтении считывается содержимое другого регистра, обеспечивающего прием данных интерфейсом. Запуск передатчика интерфейса производится записью байта данных в UDR, а по завершению приема запускается процедура вызова вектора прерывания для считывания байта данных, поступившего в приемник.

Регистр управления UCR содержит 8 бит управления работой интерфейса.

Символические имена битов управления регистра UCR

RXCIE	TXCIE	UDRIE	RXEN	TXEN	CHR9	RXB8	TXB8

RXCIE – бит разрешения прерывания по окончании рабочего цикла приема данных **(вектор прерывания UART_RXC с адресом $00b).**

TXCIE – бит разрешения прерывания по окончании рабочего цикла передачи данных **(вектор прерывания UART_TXC с адресом $00d).**

UDRIE – бит разрешения прерывания по перезаписи данных из регистра UDR в буфер передатчика, т.е. предыдущее сообщение еще обрабатывается передатчиком интерфейса UART, но уже разрешена запись следующего сообщения для подготовки к передаче **(вектор прерывания UART_DRE с адресом $00c).**

RXEN – бит разрешения приема, разрешает работу приемника интерфейса.

TXEN – бит разрешения передачи, разрешает работу передатчика интерфейса.

CHR9 – бит разрешения 9-битового формата сообщения.

RXB8 – дополнительный 9-й бит сообщения приемника (бит 8).

TXB8 – дополнительный 9-й бит сообщения передатчика (бит 8).

В регистре UCR все биты управления при единичном значении разрешают выполнение соответствующей функции, при нулевом значении – запрещают. Эти значения и, следовательно, выполнение указанных функций задаются программно.

Регистр состояния USR содержит флаги выполнения интерфейсом различных операций.

Символические имена флагов в регистре USR

RXC	TXC	UDRE	FE	OR	–	–	–

RXC – флаг окончания приема сообщения; устанавливается после приема очередного сообщения в UDR и вызывает вектор прерывания UART_RXC, если биты разрешения RXCIE (в регистре UCR) и I (в регистре SREG) установлены. Флаг очищается при чтении регистра UDR, т.е. подпрограмма обработки прерывания RXC должна обязательно предусматривать чтение регистра UDR для очистки флага.

TXC – флаг завершения передачи сообщения; аналогично флагу RXC вызывает вектор прерывания UART_TXC. Флаг TXC очищается автоматически при вызове вектора прерывания или программно записью "1" в этот бит.

UDRE – флаг готовности регистра UDR к приему нового сообщения для передачи (до завершения передачи TXC); аналогично флагу RXC вызывает вектор прерывания UART_DRE. Флаг очищается при записи в регистр UDR, что должно обязательно выполняться в подпрограмме обработки прерывания.

FE – флаг ошибки формата сообщения; устанавливается, например, когда стоповый бит принятого сообщения имеет нулевой уровень. Очищается при приеме сообщения установленного формата.

OR – флаг перегрузки; устанавливается, когда предыдущее принятое сообщение еще не прочитано из регистра UDR, а следующее уже поступает в приемник. Очищается, когда сообщение принято и передано в UDR.

Регистр UBRR определяет скорость обмена данными интерфейса. Скорость обмена данными BAUD определяется формулой

$$BAUD = \frac{f_{тн}}{16(UBRR + 1)},$$

где $f_{тн}$ – частота тактового генератора микроконтроллера,

UBRR – содержимое регистра скорости (число в диапазоне $00 - $FF).

Вычисленное значение скорости округляется до ближайшего стандартного значения (не рекомендуется использование скоростей, отличающихся от стандартных более, чем на 1 %).

Таким образом, в интерфейсе UART скорости передачи и приема сообщения одинаковы и определяются UBRR, разрешение на выполнение различных операций задается программно изменением содержимого регистра UCR. Операции обмена данными выполняются по флагам регистра USR векторами соответствующих прерываний. Подпрограммы обработки прерываний должны выполнять обмен данными с интерфейсом UART через регистр UDR.

3.4. Таймеры микроконтроллера

Микроконтроллер содержит 3 таймера-счетчика, которые работают независимо друг от друга. Таймер 0 и таймер 2 состоят из 8-разрядных счетчиков, таймер 1 содержит 16-разрядный счетчик. Счетчики таймеров работают в режиме суммирования. Каждый из таймеров может работать от тактового генератора микроконтроллера с программируемым коэффициентом деления частоты (от 1 до 1024) либо от внешнего источника входного сигнала. При использовании внешнего источника входные сигналы стробируются тактовыми импульсами микроконтроллера. Кроме того, таймер 2 мо-

жет работать в асинхронном режиме от дополнительного генератора, содержащегося в микроконтроллере. Этот генератор оптимизирован для работы с кварцевым резонатором часов частотой 32,768 кГц, что позволяет получать с помощью таймера 2 импульсы с периодом 1 с.

Управление работой таймеров и обработка их сигналов производится через соответствующие регистры файла регистров ввода-вывода. Регистр масок TIMSK и регистр флагов прерываний TIFR – общие для всех таймеров и программно доступны для чтения и записи. Остальные регистры управления индивидуальны для каждого таймера. Взаимодействие с таймерами при формировании заданных временных интервалов практически всегда необходимо производить процедурами ввода-вывода по соответствующим прерываниям. Основное назначение регистров TIMSK и TIFR – управление прерываниями таймеров.

Регистр TIMSK содержит биты разрешения, а регистр TIFR содержит флаги всех прерываний, вырабатываемых таймерами. Биты одноименных разрядов обоих регистров обеспечивают обработку соответствующих прерываний таймеров. Эти вектора прерываний будут описаны далее для каждого таймера.

Символические имена битов разрешения в регистре TIMSK

OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	–	TOIE0

Символические имена битов разрешения в регистре TIFR

OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	–	TOV0

Например, с помощью битов 6 обоих регистров (TOIE2 – TOV2) производится обработка прерывания по переполнению таймера 2. Установленное программно единичное значение бита TOIE2 разрешает прерывание, а при переполнении таймера 2 устанавливается единичное значение флага TOV2, которое вызывает **вектор прерывания TIM2_OVF (адрес вектора $004)**, если общее разрешение прерывания установлено. Все флаги регистра TIFR очищаются аппаратно при вызове векторов прерываний либо программно записью единицы в соответствующий бит этого регистра. Аналогично и попарно работают все остальные биты регистров TIMSK и TIFR.

3.4.1. Таймер 0

Таймер 0 содержит 8-разрядный суммирующий двоичный счетчик и формирует прерывание только по переполнению с помощью битов TOIE0 и TOV0 регистров TIMSK и TIFR. Кроме этих регистров, таймер 0 использу-

ет регистры **TCCR0 и TCNT0** из файла регистров ввода-вывода. Флаг TOV0 при выполнении условий, указанных выше, вызывает **вектор прерывания TIM0_OVF с адресом $009.** Регистр TCCR0 управляет работой таймера, а регистр TCNT0 содержит текущее состояние счетчика таймера, оба регистра программно доступны и для чтения, и для записи.

В регистре **TCCR0** используются только три младших бита **CS02, CS01, CS00**. Эти биты управляют входным сигналом таймера 0:

000 – входной сигнал блокируется, таймер остановлен.

001 – таймер работает (при всех последующих значениях управляющих битов также работает), входной сигнал счетчика – тактовые импульсы микроконтроллера с частотой $f_{ти}$.

010 – частота входного сигнала $f_{ти}/8$.

011 – частота входного сигнала $f_{ти}/64$.

100 – частота входного сигнала $f_{ти}/256$.

101 – частота входного сигнала $f_{ти}/1024$.

110 – внешний входной сигнал, поступающий на **вход T0 (альтернативная функция PB0)**, падающий фронт.

111 – внешний входной сигнал T0 (PB0), нарастающий фронт.

С помощью регистра **TCNT0** можно производить программную обработку состояния таймера, запись байта в этот регистр не останавливает работу счетчика, а чтение этого же регистра позволяет программно анализировать изменение состояния счетчика от момента запуска битами управления. Прерывание в таймере 0 с вызовом соответствующего вектора прерывания формируется только флагом переполнения TOV0.

Пример программного управления таймером 0:

```
m1:  clr   r1          ; очистка r1 (r1=0),
m2:  out   TCCR0, r1; пересылка байта из r1 (0) в регистр
                ; управления TCCR0,
m3:  out   TCNT0, r1 ; пересылка байта из r1 (0) в регистр
                ; таймера TCNT0,
m4:  ldi   r16, 1      ; загрузка в r16 константы 1,
m5:  out   TIMSK, r16      ; пересылка в регистр TIMSK константы
                ; для разрешения прерывания таймера 0,
m6:  ldi   r16, 3      ; загрузка в r16 константы управления таймера 0,
m7:  out   TCCR0, r16      ; запуск таймера 0,
```

Команды m1-m3 выполняют подготовку таймера 0 для работы (m2 – остановка таймера, m3 – сброс счетчика в нулевое состояние). Команды m4, m5 необходимы для разрешения прерывания таймера 0. Команды m6, m7 производят запуск таймера 0. После выполнения команды m7 таймер начинает работать: частота входного сигнала $f_{ти}/64$ в соответствии с константой управления, равной 3, через каждые 256 входных импульсов фор-

*мируется флаг прерывания TOV0. Следовательно, вызов соответствующего вектора прерывания будет производиться каждые 64*256=16384 тактов работы микроконтроллера.*

3.4.2. Таймер 1

Таймер 1 построен на основе 16-разрядного двоичного счетчика, формирует прерывания по 4 различным векторам прерываний, может использоваться в качестве широтно-импульсного модулятора (ШИМ) и обладает существенно расширенными возможностями по сравнению с таймером 0. Управление прерываниями, в соответствии с ранее описанными процедурами, производится соответствующими битами регистров TIMSK и TIFR. Кроме этих двух регистров, режимы и параметры работы таймера 1 определяют еще 10 регистров файла регистров ввода-вывода.

Переполнение таймера 1 формирует флаг прерывания TOV1 (аналогичен флагу TOV0), который при выполнении необходимых условий вызывает **вектор прерывания TIM1_OVF с адресом $008.** В таймере 1 также предусмотрены два канала сравнения эталонных значений, хранящихся в регистрах OCR1A и OCR1B, с текущим состоянием счетчика (Compare A, Compare B). При совпадении эталонного значения с текущим состоянием счетчика формируется соответствующий флаг прерывания OCF1A или OCF1B в регистре TIFR и выполняется такая же стандартная процедура обработки аппаратных прерываний, как и для остальных флагов. **Вектор прерывания канала A – TIM1_COMPA с адресом $006, вектор прерывания канала B – TIM1_COMPB с адресом $007.**

Последний флаг прерывания таймера 1 – прерывание режима **захват ICF1**. Когда на **вход режима захват ICP (альтернативная функция линии ввода-вывода PD6)** поступает сигнал, текущее состояние таймера записывается в специальный **регистр захвата ICR1** и формируется флаг прерывания ICF1 регистра TIFR со стандартной процедурой вызова **вектора прерывания TIM1_CAPT с адресом $005.** Основное назначение этого режима – отсчет интервала времени от программного запуска таймера до внешнего события, изменившего входной сигнал PD6.

Двухбайтовый регистр ICR1 программно доступен только для чтения и содержит оба байта текущего состояния 16-битового счетчика (ICR1L – регистр младшего байта, ICR1H – регистр старшего байта). При чтении содержимого регистра ICR1 в подпрограмме обработки прерывания сначала должна выполняться команда чтения младшего байта (регистр ICR1L) и только после этого – команда чтения старшего байта (регистр ICR1H) *.

---

* Чтение и запись содержимого всех 16-битовых регистров необходимо производить в определенной последовательности для обеспечения одновременного доступа к обоим байтам. В процедурах доступа старший байт всегда помещается в регистр временного хранения TEMP.

Процедуры доступа ко всем двухбайтовым регистрам должны выполняться в соответствии с правилами данного примечания.

Остальные параметры и режимы задаются регистрами TCCR1A и TCCR1B.

Символические имена битов управления в регистре TCCR1A

COM1A1	COM1A0	COM1B1	COM1B0	-	-	PWM11	PWM10

Каждый канал сравнения в дополнение к флагу прерывания может формировать на выводах порта D выходной сигнал: **OC1A (альтернативная функция PD5), OC1B (альтернативная функция PD4)** с параметрами, определяемыми битами **COM1A1, COM1A0, COM1B1, COM1B0** регистра TCCR1A. Режимы формирования выходных сигналов для каналов Compare A, Compare B одинаковы и соответствуют битам управления COM1X1, COM1X0 (X – это или канал A, или канал B) следующим образом:

00 – выходной сигнал OC1X не формируется, выходы порта D не используются.

01 – выходной сигнал OC1X импульсный.

10 – выходной сигнал OC1X имеет активный нулевой уровень.

11 – выходной сигнал OC1X имеет активный единичный уровень.

Рекомендуется при использовании сигнала OC1X запретить прерывание по этому каналу очисткой соответствующего бита разрешения в регистре TIMSK, иначе произойдет обработка прерывания при формировании этого сигнала. Эти же сигналы являются выходными для таймера, работающего в режиме модулятора ШИМ.

Последние биты **PWM11, PWM10** регистра TCCR1A определяют работу таймера 1 в режиме модулятора ШИМ следующим образом:

00 – режим модулятора ШИМ запрещен (работа в стандартных режимах таймера).

01 – 8-битовый модулятор ШИМ.

10 – 9-битовый модулятор ШИМ.

11 – 10-битовый модулятор ШИМ.

Работа в режиме модулятора ШИМ будет описана позднее.

Команда чтения младшего байта одновременно записывает старший байт в TEMP, а следующая за этим операция чтения старшего байта фактически выполняется из регистра TEMP. При записи последовательность обратная: сначала производится запись старшего байта (в действительности выполняется в TEMP), а следующая за ней операция записи младшего байта производит одновременную передачу старшего байта из TEMP. Для корректного выполнения доступа к таким регистрам **прерывание должно быть запрещено перед началом обмена**.

Символические имена битов управления в регистре TCCR1B

ICNC1	ICES1	-	-	CTC1	CS12	CS11	CS10

Биты **ICNC1** и **ICES1** второго регистра управления TCCR1B определяют работу таймера 1 в режиме "захват" (capture). ICNC1 определяет функцию дополнительного контроля входного сигнала ICP. При нулевом уровне этого бита функция "захват" включается первым тактом микроконтроллера после активного уровня сигнала ICP. При единичном значении бита требуется, чтобы входной сигнал ICP был активным не менее 4 тактов микроконтроллера. Этот режим позволяет устранить ложное включение под действием кратковременных помех. Бит ICES1 определяет активный уровень сигнала ICP, при нуле – переход сигнала в активное состояние по падающему фронту, при единице – по нарастающему фронту.

Бит **CTC1** может разрешить сброс таймера 1 по сигналу сравнения канала А. При нулевом значении бита сброс таймера по каналу сравнения А не производится, при единичном значении бита и совпадении эталонного значения канала А с текущим состоянием таймера производится сброс его счетчика в исходное состояние ($0000). Биты **CS12, CS11, CS10** задают параметры входного сигнала для счетчика в таймере 1:

000 – входной сигнал блокируется, таймер остановлен.

001 – таймер работает (при всех последующих значениях управляющих битов также работает), входной сигнал счетчика – тактовые импульсы микроконтроллера с частотой $f_{ти}$.

010 – частота входного сигнала $f_{ти}/8$.

011 – частота входного сигнала $f_{ти}/64$.

100 – частота входного сигнала $f_{ти}/256$.

101 – частота входного сигнала $f_{ти}/1024$.

110 – внешний входной сигнал, поступающий на **вход T1 (альтернативная функция линии PB1)**, падающий фронт.

111 – внешний входной сигнал T1 (PB1), нарастающий фронт.

При остановке таймера этими битами (000) все функции блокируются так же, как и в таймере 0. При работе таймера от внешнего источника сигнала тактовые импульсы микроконтроллера стробируют входные сигналы счетчика.

Кроме регистра ICR1, работу таймера 1 обеспечивают еще три 16-разрядных регистра: регистр текущего состояния таймера **TCNT1**, регистр эталонного значения канала сравнения А **OCR1A**, регистр эталонного значения канала сравнения В **OCR1B**. Эти регистры программно доступны и для чтения, и для записи и фактически образуются регистровыми парами (индекс L – регистр младшего байта, индекс H – регистр старшего байта). Процедуры доступа, как и ко всем двухбайтовым регистрам, должны вы-

полняться в соответствии с правилами, изложенными в примечании на предыдущей странице.

Чтение регистров TCNT1L и TCNT1H позволяет программно контролировать текущее состояние таймера, запись в эти регистры программно задает текущее состояние. При программном обращении к TCNT1 счетчик таймера сохраняет режим работы, заданный соответствующими битами регистров управления. Регистры OCR1A и OCR1B (по 2 байта каждый) служат для записи и хранения эталонных значений обоих каналов сравнения. Если в момент записи выполняется условие сравнения, прерывание по каналу сравнения не формируется.

Если выбран режим модулятора ШИМ (биты управления PWM11, PWM10), счетчик таймера 1 из суммирующего становится реверсивным. Его состояние в режиме суммирования возрастает от начального $0000 до максимального, определяемого разрядностью кода управления (при 8-битовом управлении – $00ff, при 9-битовом – $01ff, при 10-битовом – $03ff). Затем счетчик переходит в режим вычитания и уменьшает состояние до начального, возвращаясь далее в режим суммирования, и т.д.

Частота сигнала ШИМ определяется делением тактовой частоты микроконтроллера на коэффициент, зависящий от разрядности кода управления (8 бит – 510, 9 бит – 1022, 10 бит – 2046). Корректная работа в режиме ШИМ возможна только при значении битов управления входным сигналом таймера (CS12, CS11,CS10) – 001, т.е. в случае подачи на вход таймера тактовых сигналов микроконтроллера. Коды управления ШИМ записываются в регистры эталонных значений обоих каналов сравнения OCR1A и OCR1B, а выходные сигналы ШИМ поступают на выходы каналов сравнения OC1A (PD5), OC1B (PD4) и, следовательно, таймер 1 реализует два канала ШИМ с независимыми кодами управления одинаковой разрядности и раздельными выходными сигналами.

Биты управления выходными сигналами **COM1X1, COM1X0** (X – это или A, или B) регистра TCCR1A задают параметры сигналов ШИМ следующим образом:

00, 01 – выходные сигналы ШИМ не формируются, выходы OC1X не используются.

10 – неинвертированный ШИМ, длительность логической единицы в каждом периоде выходного сигнала пропорциональна коду управления ШИМ.

11 – инвертированный ШИМ, длительность логического нуля в каждом периоде выходного сигнала пропорциональна коду управления ШИМ.

Флаг переполнения TOV1 в режиме ШИМ устанавливается, когда счетчик переключается из состояния $0000. Этот флаг позволяет определить начало каждого периода сигнала ШИМ.

Таким образом, таймер 1, построенный на 16-разрядном счетчике, формирует 4 флага прерывания, может работать в режиме двухканального модулятора ШИМ и обеспечивает аппаратную реализацию разнообразных функций микроконтроллера. Программное управление всеми функциями таймера 1 производится аналогично показанному в примере для таймера 0.

3.4.3. Таймер 2

Таймер 2 построен на основе 8-разрядного счетчика, формирует два флага прерываний: флаг переполнения TOV2, флаг канала сравнения OCF2, может работать в режиме 8-битового модулятора ШИМ. Флаги прерываний обрабатываются в соответствии со стандартной описанной ранее процедурой с вызовом векторов прерывания: **вектор переполнения TIM2_OVF с адресом $004, вектор сравнения TIM2_COMP с адресом $003.** Дополнительно таймер 2 может работать в асинхронном режиме, получая входные сигналы счетчика от дополнительного генератора микроконтроллера.

Параметры и режимы работы таймера 2 определяют 4 регистра файла регистров ввода-вывода: **регистр управления TCCR2, регистр текущего состояния таймера TCNT2, регистр эталонного значения канала сравнения OCR2, регистр асинхронного режима ASSR.** Работа таймера 2 аналогична работе таймера 1 в соответствующих режимах.

В регистре управления TCCR2, программно доступном для записи и чтения, используется 7 бит.

Символические имена битов управления в регистре TCCR2

-	PWM2	COM21	COM20	CTC2	CS22	CS21	CS20

Бит **PWM2** управляет режимом ШИМ. 0 – запрещает режим ШИМ и определяет стандартную работу таймера, 1 – переводит таймер 2 в режим ШИМ.

Биты **COM21, COM20** определяют процедуры формирования выходного сигнала канала сравнения **OC2 (альтернативная функция линии ввода-вывода PD7).** Работа канала сравнения таймера 2 аналогична работе каналов сравнения таймера 1. Параметры сигнала OC2 для таймера 2 в зависимости от битов управления следующие:

00 – выходной сигнал OC2 не формируется, выход PD7 порта D не используется.

01 – выходной сигнал OC2 импульсный.

10 – выходной сигнал OC2 имеет активный нулевой уровень.

11 – выходной сигнал OC2 имеет активный единичный уровень.

Рекомендуется при использовании сигнала OC2 запретить прерывание очисткой соответствующего бита разрешения в регистре TIMSK, иначе произойдет обработка прерывания при формировании этого сигнала. Этот же

сигнал является выходным для таймера 2, работающего в режиме модулятора ШИМ.

Бит **CTC2** может разрешить сброс таймера 2 по сигналу сравнения. При нулевом значении бита сброс таймера по каналу сравнения не производится, при единичном значении бита и совпадении эталонного значения с текущим состоянием таймера производится сброс его счетчика в исходное состояние ($00).

Биты **CS22, CS21, CS20** задают параметры входного сигнала для счетчика в таймере:

000 – входной сигнал блокируется, таймер остановлен.
001 – таймер работает (при всех последующих значениях управляющих битов также работает), частота входного сигнала счетчика – $f_{вх}$.
010 – частота входного сигнала $f_{вх}/8$.
011 – частота входного сигнала $f_{вх}/32$.
100 – частота входного сигнала $f_{вх}/64$.
101 – частота входного сигнала $f_{вх}/128$.
110 – частота входного сигнала $f_{вх}/256$.
111 – частота входного сигнала $f_{вх}/1024$.

При остановке таймера этими битами (000) все функции блокируются так же, как и в других таймерах.

Регистры **TCNT2** и **OCR2** выполняют такие же функции, как и аналогичные регистры таймера 1. Выход сигнала сравнения OC2 (PD7) также применяется для формирования выходного сигнала ШИМ.

Регистр **ASSR** используется для управления работой таймера 2 в асинхронном режиме, в этом режиме входной сигнал счетчика поступает от вспомогательного генератора микроконтроллера и не синхронизируется тактовыми сигналами микроконтроллера. **К входу (альтернативная функция линии PC6) и выходу (альтернативная функция линии PC7) дополнительного генератора** можно подключить отдельную частотнозадающую цепь. Например, для построения часов реального времени можно использовать кварцевый резонатор часов с частотой 32,768 кГц.

Символические имена битов в регистре ASSR

-	-	-	-	AS2	TCN2UB	OCR2UB	TCR2UB

AS2 – бит асинхронного режима. При 0 входной сигнал счетчика формируется тактовым генератором микроконтроллера. При 1 входной сигнал счетчика формируется вспомогательным генератором с использованием цепей PC6, PC7. Переключение между асинхронным и синхронным режимами может привести к изменению состояния регистров TCCR2, TCNT2,

OCR2. Для предотвращения возможных сбоев в работе могут использоваться биты TCN2UB, OCR2UB, TCR2UB. Рекомендации по предотвращению сбоев с помощью этих битов приведены в руководствах фирмы ATMEL.

Работа таймера 2 в режиме ШИМ аналогична работе таймера 1 в этом же режиме при 8-битовом коде управления, частота сигнала ШИМ на выходе OC2 (PD7) определяется делением частоты входного сигнала счетчика на 510, код управления должен поступать в регистр OCR2. Все операции управления работой таймера 2 аналогичны рассмотренным ранее.

3.5. Аналоговый компаратор

Аналоговый компаратор производит сравнение сигналов, поступающих на **неинвертирующий вход AIN0 (альтернативная функция линии ввода-вывода PB2) и инвертирующий вход AIN1 (альтернативная функция линии PB3).** Выходной сигнал компаратора может использоваться в режиме "захват" таймера 1 или выполнять вызов отдельного **вектора прерывания компаратора ANA_COMP с адресом $010.**

Управление функциями компаратора производится регистром управления компаратора **ACSR**, в котором 6 бит программно доступны для чтения и записи, а бит выходного сигнала ACO компаратора доступен только для чтения.

Символические имена битов в регистре ACSR

ACD	-	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0

ACD – бит выключения компаратора, при 1 компаратор отключен. Рекомендуется при изменении этого бита запретить прерывание компаратора битом ACIE.

ACO – бит выходного сигнала компаратора.

ACI – флаг прерывания компаратора устанавливается в 1, когда параметры выходного сигнала компаратора соответствуют битам ACIS1, ACIS0. Этот флаг работает вместе с битом разрешения ACIE так же, как и остальные флаги прерываний.

ACIE – бит разрешения прерывания компаратора.

ACIC – бит управления компаратором для режима "захват" таймера 1. При 1 сигнал компаратора поступает на вход захвата таймера 1 и действует в соответствии с параметрами и режимами таймера 1.

ACIS1, ACIS0 – определяют параметры сигнала компаратора для формирования флага прерывания:

00 – прерывание по импульсному выходному сигналу компаратора,

01 – не используется,

10 – прерывание по падающему фронту сигнала компаратора,

11 – прерывание по нарастающему фронту сигнала компаратора.

Изменение младших двух битов регистра ACSR может установить флаг прерывания, рекомендуется перед этой операцией запретить прерывание по этому флагу. Необходимо также учитывать, что использование команд SBI и CBI для регистра ACSR приведет к очистке флага прерывания ACI.

3.6. Аналого-цифровой преобразователь (АЦП)

АЦП микроконтроллера разрядностью 10 бит работает по алгоритму последовательных приближений, погрешность преобразования – не более 2 единиц младшего значащего разряда, время преобразования 65 мкс – 260 мкс. АЦП совместно со встроенным аналоговым мультиплексором обеспечивает преобразование в 10-ти разрядный двоичный код сигналов по **8 аналоговым входам (альтернативная функция линий ввода-вывода порта А)** в диапазоне напряжений от 0 (AGND) до опорного (AREF).

Для снижения уровня помех цепи питания (AGND, AVCC) схем преобразования аналоговых сигналов подключаются отдельно (рис. 4), напряжение питания AVCC не должно отличаться от напряжения питания VCC более чем на ±0,3 В. Опорное напряжение должно лежать в диапазоне от 2 В до напряжения питания AVCC. Код АЦП $000 соответствует нулевому входному сигналу, максимальный код $3FF соответствует сигналу, равному опорному, минус вес единицы младшего значащего разряда.

Номер входа мультиплексора, с которого поступает сигнал для преобразования в АЦП, определяется тремя младшими битами **MUX2, MUX1, MUX0** управляющего регистра **ADMUX.** Любой из восьми входов может быть выбран через ADMUX записью в него соответствующего кода в любой момент времени, однако переключение входов фактически производится только после завершения очередного цикла преобразования АЦП. В линиях ввода-вывода порта А, используемых для приема аналоговых сигналов, необходимо задавать режим ввода.

АЦП может работать в режиме однократного преобразования или циклически с автоматическим повторным запуском после каждого преобразования. По окончании преобразования формируется **флаг прерывания ADIF** со стандартной процедурой вызова **вектора прерывания ADC с адресом $00e** и записью 10-разрядного кода в **двухбайтовый регистр ADCL** (младшие 8 бит результата), **ADCH** (старшие 2 бита результата). Чтение данных из регистра результата ADC должно начинаться обязательно с младшего байта (см. примечание на с. 57). Точность АЦП зависит от тактовой частоты преобразования, рекомендуется диапазон тактовых частот 50 – 200 кГц, при более высоких частотах точность преобразования снижается. Стандартный цикл преобразования требует 13 тактов работы и при рекомендуемом значении тактовой частоты 100 кГц определяет время преобразования 130 мкс.

Кроме регистров ADMUX, ADCH, ADCL, работа АЦП определяется регистром **ADCSR**, который также содержится в файле регистров ввода-вывода.

Символические имена битов управления в регистре ADCSR

ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS20

ADEN – бит разрешения, 0 – АЦП выключен, 1 – АЦП включен.

ADSC – бит запуска преобразования, в режиме однократного преобразования единица должна записываться при каждом запуске, в циклическом режиме – один раз для запуска первого преобразования.

ADFR – бит режима преобразования, 1 – циклический режим, 0 – однократный.

ADIF – флаг прерывания, устанавливается после завершения преобразования и записи кода в выходной регистр АЦП. Очищается автоматически при вызове вектора прерывания либо записью 1 в этот бит.

ADIE – бит разрешения прерывания АЦП, 1 разрешает прерывание.

ADPS2, ADPS1, ADPS20 – биты управления тактовой частотой АЦП, определяют коэффициент деления тактовой частоты микроконтроллера следующим образом:

000 – коэффициент деления 2,
001 – коэффициент деления 2,
010 – коэффициент деления 4,
011 – коэффициент деления 8,
100 – коэффициент деления 16,
101 – коэффициент деления 32,
110 – коэффициент деления 64,
111 – коэффициент деления 128.

Дополнительное снижение уровня помех для повышения точности преобразования можно получить, если на время преобразования АЦП приостановить работу процессора в микроконтроллере переходом в режим "idle". Возврат в рабочее состояние должна обеспечивать подпрограмма обработки прерывания АЦП. Более подробную информацию о работе АЦП можно найти в руководствах фирмы ATMEL.

Для управления работой АЦП подпрограмма его вектора прерывания должна выполнить чтение результатов завершенного цикла преобразования из выходных регистров, определить через регистр мультиплексора номер следующего входного канала и произвести запуск следующего цикла преобразования.

3.7. Чтение и запись данных EEPROM

Доступ к данным в EEPROM производится с помощью 4 регистров файла регистров ввода-вывода: **регистра управления EECR, регистра данных EEDR, регистра старшего байта адреса EEARH, регистра младшего байта адреса EEARL.** Время доступа не превышает 4 мс и зависит от напряжения питания. Завершение процедуры доступа формирует флаг прерывания готовности EEPROM со стандартной процедурой вызова **вектора прерывания EE_RDY с адресом $00f.**

Обмен данными с EEPROM производится через регистр данных EEDR по двухбайтовому адресу, указанному в регистрах EEARH, EEARL. Процедурами доступа управляют биты регистра EECR.

Символические имена битов в регистре управления EECR

-	-	-	-	EERIE	EEMWE	EEWE	EERE

EERIE – бит разрешения прерывания EEPROM, это прерывание формируется при нулевом значении бита разрешения записи EEWE.

EEMWE – бит записи данных в EEPROM, при единичном значении EEWE записывает в EEPROM данные из регистра EEDR по адресу, указанному в регистрах адреса.

EEWE – бит разрешения записи.

EERE – бит управления чтением данных, при единичном значении производится чтение данных из EEPROM в регистр EEDR.

Процедура записи данных усложнена для защиты от несанкционированного изменения данных. Более подробную информацию можно найти в руководствах фирмы ATMEL.

3.8. Система прерываний и регистры общего управления

Аппаратные прерывания в микроконтроллере обрабатываются 17 векторами прерываний с жестко закрепленными адресами во флэш-памяти в диапазоне от $000 до $010. Уровни приоритетов векторов прерываний определяются их адресами, наименьшему адресу $000 соответствует самый высокий приоритет. Стандартный текст программы со ссылками на метки подпрограмм обработки прерываний должен быть следующим:

```
$000  rjmp RESET      ; Reset Handler
$001  rjmp EXT_INT0     ; IRQ0 Handler
$002  rjmp EXT_INT1     ; IRQ1 Handler
$003  rjmp TIM2_COMP  ; Timer2 Compare Handler
```

```
$004  rjmp TIM2_OVF    ; Timer2 Overflow Handler
$005  rjmp TIM1_CAPT ; Timer1 Capture Handler
$006  rjmp TIM1_COMPA  ; Timer1 CompareA Handler
$007  rjmp TIM1_COMPB  ; Timer1 CompareB Handler
$008  rjmp TIM1_OVF    ; Timer1 Overflow Handler
$009  rjmp TIM0_OVF    ; Timer0 Overflow Handler
$00a  rjmp SPI_STC   ; SPI Transfer Complete
                       ;  Handler
$00b  rjmp UART_RXC    ; UART RX Complete Handler
$00c  rjmp UART_DRE    ; UDR Empty Handler
$00d  rjmp UART_TXC    ; UART TX Complete Handler
$00e  rjmp ADC       ; ADC Complete Interrupt
                       ; Handler
$00f  rjmp EE_RDY    ; EEPROM Ready Handler
$010  rjmp ANA_COMP    ; Analog Comparator Handler
```

Использовать эту область памяти для хранения кодов других команд нецелесообразно, так как ошибочное формирование флага прерывания может привести к вызову соответствующего вектора с некорректным выполнением процедуры обработки. Вектора прерываний с адресами $003 - $010 описаны в предыдущих разделах и служат для вызова подпрограмм, необходимых для реализации соответствующих функций встроенных аппаратных средств микроконтроллера.

Вектор **RESET с адресом $000** выполняет инициализацию микроконтроллера при начальном сбросе. Обработка этого вектора не требует никаких условий, т.е. вектор может быть вызван в произвольный момент времени независимо от состояния и режима работы микроконтроллера. RESET загружает в программный счетчик начальный адрес $000 и вызывается в следующих случаях: при включении питания микроконтроллера или временном снижении напряжения питания в процессе работы ниже критической величины, при поступлении на вход микроконтроллера RESET сигнала низкого уровня длительностью более 50 нс., при формировании сигнала RESET сторожевым таймером.

В регистре микроконтроллера **MCUSR** биты **EXTRF** (бит 1), **PORF** (бит 0) позволяют определить причину начального сброса. EXTRF устанавливается при начальном сбросе внешним сигналом, PORF устанавливается при

сбросе по включению питания. Если эти биты обнулить программной записью содержимого этого регистра, после следующего начального сброса можно определить источник формирования этого сигнала. Сброс сторожевым таймером при чтении определится кодом 00, сброс внешним сигналом – кодом 10, сброс включением питания – кодами 01, 11.

Процедура вызова всех остальных векторов прерывания отличается от RESET. Во-первых, должен быть установлен флаг общего разрешения прерываний I в регистре SREG и бит разрешения данного прерывания в соответствующем регистре. Во-вторых, установка определенного флага прерывания вызывает соответствующий вектор прерывания (только при выполнении первого условия). При одновременном формировании нескольких флагов прерываний очередность обработки определяется уровнями приоритетов (чем меньше адрес вектора, тем выше уровень приоритета). В-третьих, при загрузке адреса прерывания в программный счетчик автоматически очищаются флаг I, запрещая другие прерывания, и флаг данного прерывания, а текущее состояние программного счетчика загружается в стек. Подпрограмма обработки прерывания должна завершаться командой **"reti"**, которая восстанавливает разрешение прерываний (флаг I) и состояние программного счетчика (загружается из стека). Возможен программный сброс флага прерывания (без вызова вектора прерывания) записью единицы в соответствующий бит регистра флагов прерываний, а также программное разрешение прерываний (флаг I) в любой подпрограмме обработки прерываний.

Регистр состояния микроконтроллера SREG, как и остальные регистры управления, находится в файле регистров ввода-вывода. Кроме флага общего разрешения прерываний, регистр SREG содержит флаги – стандартные признаки результатов преобразования данных, необходимые для выполнения ряда команд в рабочих программах микроконтроллера. Содержание регистра может измениться в процессах обработки прерываний, необходимо в подпрограммах обработки прерываний предусмотреть сохранение содержимого этого регистра в стеке с восстановлением перед выходом из подпрограммы.

Символические имена флагов в регистре состояния SREG

I	T	H	S	V	N	Z	C

I – флаг общего разрешения прерываний, может устанавливаться и очищаться программно соответствующими командами или аппаратно при обработке прерываний.

T – бит копирования, используется для временного хранения отдельных битов командами BLD, BST.

H – флаг полупереноса, формируется в некоторых арифметических операциях как результат переноса между младшей тетрадой и старшей тетрадой в байте.

S – флаг знака, его значение определяется суммой по mod 2 флагов N ⊕ V.

V – флаг дополнения до двух.

N – флаг отрицательного результата арифметических или логических операций.

Z – флаг нулевого результата арифметических или логических операций.

C – флаг переноса в арифметических или логических операциях.

Регистр – указатель стека имеет размер 2 байта и состоит из двух регистров **SPL, SPH** файла регистров ввода-вывода. При инициализации необходимо всегда определять адрес вершины стека. Рекомендуется задавать максимальный физический адрес ОЗУ с символическим именем RAMEND, тогда объем стека будет определяться физическим объемом ОЗУ. Стек всегда используется при вызове векторов прерываний для сохранения текущего состояния программного счетчика.

Сигналы внешних прерываний **INT0 (альтернативная функция линии PD2) и INT1 (альтернативная функция линии PD3)** обрабатываются с помощью регистров **GIFR, GIMSK и общего регистра управления MCUCR** файла регистров ввода-вывода. **В регистре GIMSK биты INT1 (бит 7) и INT0 (бит 6)** определяют разрешение внешних прерываний, а в регистре **GIFR флаги INTF1 (бит 7) и INTF0 (бит 6)** обеспечивают вызов векторов внешних прерываний по процедурам, аналогичным обработке других векторов прерываний.

Параметры сигналов, формирующие флаги внешних прерываний, определяются битами ISC01, ISC00 и ISC11, ISC10 регистра MCUCR. Остальные биты этого регистра определяют параметры режима **sleep,** в этом режиме приостанавливается работа основных устройств микроконтроллера для снижения энергопотребления и уровня помех.

Символические имена битов управления в регистре MCUCR

-	SE	SM1	SM0	ISC11	ISC10	ISC01	ISC00

SE – бит разрешения режима sleep; если бит установлен, микроконтроллер программно переводится в режим sleep командой **"sleep".**

SM1 и SM0 определяют вид режима sleep:

00 – режим idle,
01 – не используется,
10 – режим power-down,
11 – режим power save.

Младшие 4 бита определяют параметры сигналов внешних прерываний для формирования флагов INTF1, INTF0. Биты попарно управляют обработкой

каждого из этих сигналов, ISCX1, ISCX0 (X – это 0 или 1) могут принимать следующие значения:

00 – низкий уровень сигнала внешнего прерывания устанавливает флаг,
01 – не используется,
10 – падающий фронт сигнала внешнего прерывания устанавливает флаг,
11 – нарастающий фронт сигнала внешнего прерывания устанавливает флаг.

Необходимо учитывать, что при конфигурировании PD2 или PD3 выходом, программное изменение этих сигналов также формирует соответствующие флаги и может приводить к вызову векторов прерываний.

Сторожевой таймер (Watchdog Timer) можно использовать для контроля корректности работы микроконтроллера. Если сторожевой таймер включен битами управления регистра WDTCR, через определенные интервалы времени (задаются программно в интервале от 15 мс до 2 с) формируется сигнал RESET. Корректно работающая программа должна предусматривать периодический сброс сторожевого таймера, предотвращая повторную инициализацию микроконтроллера сигналом RESET. Отсутствие своевременного сброса сторожевого таймера может рассматриваться как некорректное выполнение каких-либо функций и приведет к повторной инициализации. Режимы и параметры работы сторожевого таймера определяются программно через **регистр управления WDTCR,** который доступен для записи и чтения.

Символические имена битов управления в регистре WDTCR

-	-	-	WDTOE	WDE	WDP2	WDP1	WDP0

WDTOE – бит разрешения для изменения состояния сторожевого таймера.
WDE – бит включения сторожевого таймера; 1 – таймер включен,
0 – таймер выключен.
WDP2, WDP1, WDP0 – биты управления периодом формирования сигнала RESET сторожевым таймером. 000 – минимальный период (около 15 мс), 111 – максимальный период (около 2 с).

Изменение состояния сторожевого таймера (включен/выключен, бит WDE) возможно только по следующему алгоритму: биты WDTOE, WDE обязательно и одновременно должны быть установлены в единицу; затем в течение 4 тактов микроконтроллера в бит WDE должно быть записано требуемое значение (0 – выключение, 1 – включение). После 4 тактов WDTOE сбрасывается автоматически. Период работы сторожевого таймера зависит

от битов управления и напряжения питания, более подробную информацию можно найти в руководствах фирмы ATMEL.

Основное назначение рассматриваемой системы прерываний – организация взаимодействия всех средств микроконтроллера при решении прикладных задач. Все аппаратные средства микроконтроллера (процессор, таймеры, интерфейсы, АЦП и т.п.) могут работать и работают параллельно и независимо друг от друга. В то же время каждое из устройств обеспечивает выполнение каких-либо ограниченных функций, которые являются только элементами общей прикладной задачи. Эта общая задача на определенных этапах требует обмена данными между устройствами, запуска или останова отдельных процедур, изменения параметров или режимов работы и т.п. Система векторов прерываний позволяет выделить моменты времени, когда завершаются очередные операции и требуется определенная реакция других элементов микроконтроллера для продолжения работы.

Очевидно, что возможность параллельной реализации нескольких функций весьма существенно расширяет возможности микроконтроллера и увеличивает быстродействие. Во всех моделях микроконтроллеров AVR содержатся средства аппаратной, а, следовательно, параллельной реализации разнообразных стандартных задач, которые должны решаться при управлении техническими объектами. Как показано ранее, все эти аппаратные средства многофункциональны, в них предусмотрены различные режимы и параметры работы, и они требуют программной настройки через соответствующие регистры ввода-вывода. Многофункциональность позволяет в процессе работы на различных ее этапах управлять параметрами и режимами аппаратных средств. Эта возможность также может использоваться для повышения эффективности применения микроконтроллеров.

Указанные дополнительные преимущества микроконтроллеров требуют эффективного и корректного взаимодействия параллельно работающих аппаратных средств. Необходимость взаимодействия накладывает определенные ограничения на алгоритмы работы, которые необходимо учитывать при подготовке рабочих программ микроконтроллеров. Очевидно, что все основные процедуры организации взаимодействия реализуются программно, должны быть предусмотрены в алгоритмах работы и требуют дополнительных программных ресурсов. Система прерываний является необходимым и важнейшим элементом управления работой аппаратных средств.

Именно процедуры обработки прерываний в микроконтроллерах AVR позволяют выполнять достаточно гибкий, программно-управляемый контроль реализуемых функций. Средства управления работой аппаратных средств в обработке прерываний следующие:

- установленные приоритеты векторов прерываний – при одновременном поступлении нескольких запросов прерываний сначала вызывается вектор прерывания с наименьшим адресом;

- программное управление флагом общего разрешения прерывания (флаг I регистра SREG) – в любых программах на произвольных этапах с помощью этого флага можно программно запрещать или разрешать обработку всех прерываний;
- аппаратное управление флагом общего разрешения прерывания – при вызове любого вектора прерывания флаг глобального разрешения I очищается аппаратно и восстанавливается (командой reti) при завершении обработки прерывания (кроме того, подпрограмма обработки любого вектора прерывания также может программно изменять состояние флага I, разрешая обработку и других прерываний);
- вектора прерываний содержат в соответствующих регистрах ввода-вывода индивидуальные флаги разрешения (маскирования) – это позволяет программно разрешать или запрещать вызов каждого из векторов прерываний на любых этапах работы независимо от других векторов;
- флаги вызова векторов прерываний в регистрах ввода-вывода очищаются аппаратно при обращении к п/программе обработки прерывания;
- флаги вызова векторов прерываний могут очищаться программно записью единицы в эти биты регистров ввода-вывода – это позволяет при необходимости отменять вызов п/программ обработки прерываний.

Стандартное распределение параллельно реализуемых в микроконтроллере функций для управления техническим объектом может быть следующим (каждый пункт в данном примере – отдельный параллельно реализуемый процесс):

1. **АЦП** выполнят преобразование поступающего от датчика аналогового сигнала (время преобразования около 100 мкс, за этот период процессор может выполнить несколько сот команд рабочей программы).
2. **Процессор**, выполняя команды основной программы, производит обработку кодов, поступивших ранее от АЦП и других интерфейсов микроконтроллера, и формирует новые значения управляющих сигналов и данных для индикации.
3. **Параллельные порты ввода-вывода** обеспечивают вывод сформированных ранее сигналов управления и индикации.
4. **Таймеры** формируют сигналы с заданными ранее в программе временными характеристиками и/или в режиме модулятора ШИМ выдают сигналы управления исполнительными устройствами.
5. **Интерфейс SPI** передает полученные от процессора данные для управления символьным или матричным индикатором.

6. **Интерфейс UART** реализует обмен данными с COM-портом персонального компьютера для координации работы микроконтроллера с другими устройствами.
7. **Внешние прерывания** обеспечивают прием и обработку сигналов цифровых датчиков и сигналов управления, требующих быстрой реакции микроконтроллера (например, сигнал аварийной остановки системы управления и т.п.).

Очевидно, что перечисленные процессы асинхронны относительно друг друга, каждое из устройств реализует заданные функции со своими режимами и временными границами. Основную координирующую роль может выполнять только процессор, обеспечивая необходимый обмен данными между устройствами и управляя их режимами и параметрами через регистры ввода-вывода.

Основное требование, которое предъявляется к организации взаимодействия, – бесконфликтный доступ к ресурсам, так как различные средства микроконтроллера могут одновременно выставлять запросы на доступ к одним и тем же ресурсам. Корректное управление должно обеспечиваться даже путем снижения эффективности работы отдельных аппаратных средств. Необходимые координирующие функции должны быть предусмотрены в алгоритме основной рабочей программы и могут быть выполнены благодаря эффективной системе обработки прерываний. Для реализации координирующих функций необходимы определенные, в первую очередь программные ресурсы микроконтроллера, эти функции обычно достаточно сложны и являются важнейшей частью общего алгоритма работы.

4. СИСТЕМА КОМАНД МИКРОКОНТРОЛЛЕРОВ AVR

Реализация необходимых функций микроконтроллером требует эффективного управления его программно-аппаратными средствами. Это управление и координация работы различных средств производится рабочими программами. Подготовка рабочих программ микроконтроллеров может выполняться на персональном компьютере с помощью инструментальных средств фирмы ATMEL, например, AVR Studio.

Инструментальный пакет AVR Studio содержит средства подготовки текстов программ на языках С и ассемблере, компиляторы для формирования загрузочных файлов, симулятор для отладки программ и драйвер программатора для внутрисистемного программирования микроконтроллеров. В связи с тем, что значительная часть функций в микроконтроллере реализуется аппаратными средствами, и ассемблер обеспечивает более рациональное программное управление этими средствами, в дальнейшем будем рассматривать программирование только на ассемблере.

Система команд микроконтроллера AT90S8535 содержит 118 команд и предусматривает выполнение стандартных операций пересылки данных, арифметических и логических операций, команд управления. К дополнительным возможностям, реализованным в системе команд, можно отнести:

- выполнение двух операций одной командой (например, команда **ld r24, X+** производит пересылку байта данных из ОЗУ в регистр R24 с последующим инкрементом адреса в регистре X);
- операции с отдельными битами (например, команда **cbr r18, 2** очищает (обращает в 0) бит 1 регистра R18);
- операции с данными не только в файле регистров общего назначения, но и в файле регистров ввода-вывода (например, команда **sbi PORTC, PC7** устанавливает (обращает в 1) бит 7 регистра PORTC);
- условные команды управления по состоянию любого бита регистра состояния **SREG** или других регистров (например, команда **brtc label** выполняет переход к метке label, если флаг копирования T очищен).

В дальнейшем будут использоваться стандартные мнемонические обозначения как для команд, так и для их параметров. Стандартные мнемонические обозначения (имена) регистров и отдельных битов в них содержатся в специальных файлах, например, "8535def.inc", и включаются с помощью директив компилятора в тексты программ. Регистр состояния (флагов) микроконтроллера SREG играет важную роль в рабочих программах, биты этого регистра определяют условия для выполнения команд управления.

Регистр состояния микроконтроллера SREG

I	T	H	S	V	N	Z	C

Флаги регистра SREG (начиная со старшего бита):

I – флаг глобального разрешения прерывания, разрешает (1) или запрещает (0) все аппаратные прерывания.

T – флаг копирования бита, может быть скопирован из любого бита (или в любой бит) любого регистра общего назначения.

H – флаг переноса между младшей и старшей тетрадой байта данных.

S – флаг знака, определяется суммой по mod 2 флагов $N \oplus V$.

V – флаг переполнения (дополнения до двух).

N – флаг отрицательного результата (соответствует значению бита 7 результата операции).

Z – флаг нулевого результата операции.

C – флаг переноса.

При использовании команд с анализом флагов регистра SREG необходимо учитывать, что эти флаги формируются не всеми операциями, например, операции пересылки данных флаги не изменяют.

4.1. Система команд микроконтроллеров AVR

В мнемонических обозначениях команд всегда первым указывается регистр, в который помещается результат операции: любой регистр файла регистров общего назначения, файла регистров ввода-вывода, ячейка ОЗУ. Ограничения на параметры и операнды команд приведены в примечании. В описании команд приняты следующие обозначения:

Rd – регистр, в который помещается результат операции (любой регистр общего назначения).

Rr – регистр, из которого поступает байт данных для операции (любой регистр общего назначения).

K – константа (байт данных), число в десятичном формате от 0 до 255, в шестнадцатеричном формате могут использоваться два варианта обозначения: 0x00 – 0xff или $00 – $ff, в двоичном формате – следующее обозначение: 0b00000000 – 0b11111111.

k – константа адрес (два байта в пределах каждого адресного пространства), также может указываться в десятичном, шестнадцатеричном или двоичном формате.

X, Y, Z – регистры косвенной адресации (в файле регистров общего назначения регистр X – R27,R26; регистр Y – R29,R28; регистр Z – R30,R31).

P – регистр файла регистров ввода-вывода.

b – бит от 0 до 7 в любом регистре (старший бит – 7, младший бит – 0).
PC – программный счетчик.
STACK – стек.
PM – память программ (FLASH).

4.1.1. Арифметические и логические команды

Таблица 2

Команда	Описание	Флаги	Примечание
add Rd, Rr	Сложение, Rd ← Rd+Rr	HVNCZ	
adc Rd, Rr	Сложение с переносом, Rd ← Rd+Rr+C	HVNCZ	
adiw Rd, K	Сложение слова (2 байта, R(d+1)Rd) с константой K, R(d+1)Rd ← R(d+1)Rd+K	SVNCZ	d=(24,26,28,30), K=(0 – 63)
sub Rd, Rr	Вычитание, Rd ← Rd–Rr	HVNCZ	
subi Rd, K	Вычитание константы, Rd ← Rd–K	HVNCZ	d=(16 – 31)
sbc Rd, Rr	Вычитание с переносом, Rd ← Rd–Rr–C	HVNCZ	
sbci Rd, K	Вычитание константы с переносом, Rd ← Rd–K–C	HVNCZ	d=(16 – 31)
sbiw Rd, K	Вычитание из слова (2 байта, R(d+1)Rd) константы K, R(d+1)Rd ← R(d+1)Rd–K	SVNCZ	d=(24,26,28,30), K=(0 – 63)
inc Rd	Инкремент, Rd ← Rd+1	VNZ	
dec Rd	Декремент, Rd ← Rd–1	VNZ	
and Rd, Rr	Логическое И, Rd ← Rd•Rr	VNZ	
andi Rd, K	Логическое И с константой, Rd ← Rd•K	VNZ	d=(16 – 31)
or Rd, Rr	Логич. ИЛИ, Rd ← Rd ∨ Rr	VNZ	

Продолжение табл. 2

Команда	Описание	Флаги	Примечание
ori Rd, K	Логическое ИЛИ с константой, Rd ← Rd ∨ K	VNZ	d=(16 – 31)
eor Rd, Rr	Исключающее ИЛИ, Rd ← Rd ⊕ Rr	VNZ	
com Rd	Инверсия, Rd ← $ff–Rd	VNZ	
neg Rd	Дополнение, Rd ← $00–Rd	HVNCZ	
asr Rd	Арифметический сдвиг вправо, старший бит Rd(7) не изменяется, остальные биты Rd сдвигаются вправо, флаг C ← Rd(0)	VNCZ	
lsl Rd	Логический сдвиг влево, все биты Rd смещаются влево, младший бит Rd(0) ← 0, флаг C ← Rd(7)	VNCZ	
lsr Rd	Логический сдвиг вправо, все биты Rd смещаются вправо, старший бит Rd(7) ← 0, флаг C← Rd(0)	VNCZ	
rol Rd	Циклический сдвиг влево с переносом, все биты Rd смещаются влево, младший бит Rd(0) ← C, флаг C ← Rd(7)	VNCZ	
ror Rd	Циклический сдвиг вправо с переносом, все биты Rd смещаются вправо, старший бит Rd(7) ← C, флаг C ← Rd(0)	VNCZ	

Окончание табл. 2

Команда	Описание	Флаги	Примечание
tst Rd	Тест на ноль или минус Rd не изменяется, флаги Z и N определяются содержимым Rd, V ← 0	SVNZ	
cp Rd, Rr	Сравнение, содержимое регистров не изменяется, флаги формируются в соответствии с результатом вычитания Rd – Rr	HVNCZ	
cpc Rd, Rr	Сравнение с учетом переноса, содержимое регистров не изменяется, флаги формируются в соответствии с результатом вычитания Rd – Rr – C	HVNCZ	
cpi Rd, K	Сравнение с константой, содержимое регистров не изменяет, флаги формируются в соответствии с результатом вычитания Rd – K	HVNCZ	d=(16 – 31)

Арифметические и логические команды реализуют стандартный набор операций с однобайтовыми данными микроконтроллера с RISC архитектурой.

Примеры:

- *команда **add R3, R16** выполняет сложение содержимого регистров R3 и R16, полученная сумма будет записана в R3, содержимое регистра R16 не изменится, результат сложения определит значения флагов HVNCZ;*
- *команда **subi R21, $F0** выполняет вычитание из содержимого регистра R21 числа 240 (в шестнадцатеричном формате $F0), полученная разность будет записана в R21, результат вычитания определит значения флагов HVNCZ (в примечании указано ограничение на используемые в этой команде регистры общего назначения R16-R31);*
- *команда **and R3, R4** выполняет поразрядную логическую операцию И (конъюнкцию) содержимого регистров R3 и R4, результат будет запи-*

сан в R3, содержимое R4 не изменится, по результатам операции формируются только флаги VNZ;

- *команда **asr R0** выполняет деление на 2 числа со знаком в регистре R0, т.е. выполнится сдвиг вправо на один разряд всех битов регистра R0, кроме старшего знакового бита 7, младший бит 0 будет перемещен во флаг C, остальные флаги VNZ определяются результатом деления.*

Некоторые модели микроконтроллеров серии AVR позволяют выполнять и другие операции. Например, модели megaAVR содержат аппаратный умножитель и выполняют операции умножения.

При подготовке программ необходимо учитывать ограничения, указанные в примечаниях. Например, операции с константами допустимы не для всех регистров общего назначения. Во всех остальных случаях так же, как и в дальнейшем, регистры Rd и Rr – любые из регистров общего назначения.

Как и обычно, флаги регистра состояния SREG определяются результатом выполнения текущей операции. Если какой-либо флаг для команды не указан, это означает, что флаг данной операцией не изменяется. Команды сравнения не изменяют данные в регистрах, но, формируя соответствующие флаги, позволяют использовать команды управления по состоянию этих флагов.

4.1.2. Команды пересылки данных

Таблица 3

Команда	Описание	Флаги	Примечание
mov Rd, Rr	Пересылка данных между регистрами, Rd ← Rr	Нет	
ldi Rd, K	Загрузка константы в регистр, Rd ← K	Нет	d=(16 – 31)
lds Rd, k	Пересылка данных в Rd из ОЗУ (адрес k), Rd ← ОЗУ(k)	Нет	
sts k, Rr	Пересылка данных из Rr в ОЗУ (адрес k), ОЗУ(k) ← Rr	Нет	
in Rd, P	Пересылка данных из регистра ввода-вывода P в регистр Rd, Rd ← P	Нет	

Таблица 3

Команда	Описание	Флаги	Примечание
out P, Rr	Пересылка данных из регистра Rr в регистр ввода-вывода P, P ← Rr	Нет	
push Rr	Пересылка данных из регистра Rr в стек, STACK ← Rr	Нет	
pop Rd	Пересылка данных в регистр Rd из стека, Rd ← STACK	Нет	
Команды пересылки косвенной адресации с использованием регистров X, Y, Z			
ld Rd, X	Пересылка данных в регистр Rd из ОЗУ с адресом, указанным регистром X, Rd ← ОЗУ(X)	Нет	
ld Rd, –X	Пересылка данных в регистр Rd из ОЗУ с адресом, указанным регистром X, и предшествующим декрементом X, Rd ← ОЗУ(X–1), X ← X–1	Нет	
ld Rd, X+	Пересылка данных в регистр Rd из ОЗУ с адресом, указанным регистром X, и последующим инкрементом X, Rd ← ОЗУ(X), X ← X+1	Нет	
st X, Rr	Пересылка данных из регистра Rr в ОЗУ с адресом, указанным регистром X, ОЗУ(X) ← Rr	Нет	

Продолжение табл. 3

Команда	Описание	Флаги	Примечание
st –X, Rr	Пересылка данных из регистра Rr в ОЗУ с адресом, указанным регистром X, и предшествующим декрементом X, ОЗУ(X–1) ← Rr, X ← X–1	Нет	
st X+, Rr	Пересылка данных из регистра Rr в ОЗУ с адресом, указанным регистром X, и последующим инкрементом X, ОЗУ(X) ← Rr, X ← X+1	Нет	
ld Rd, Y	Пересылка данных в регистр Rd из ОЗУ с адресом, указанным регистром Y, Rd ← ОЗУ(Y)	Нет	
ld Rd, –Y	Пересылка данных в регистр Rd из ОЗУ с адресом, указанным регистром Y, и предшествующим декрементом Y, Rd ← ОЗУ(Y–1), Y ← Y–1	Нет	
ld Rd, Y+	Пересылка данных в регистр Rd из ОЗУ с адресом, указанным регистром Y, и последующим инкрементом Y, Rd ← ОЗУ(Y), Y ← Y+1	Нет	
ldd Rd, Y+k	Пересылка данных в регистр Rd из ОЗУ с адресом, указанным регистром Y со смещением k, Rd ← ОЗУ(Y+k)	Нет	k=(0 – 63)
st Y, Rr	Пересылка данных из регистра Rr в ОЗУ с адресом, указанным регистром Y, ОЗУ(Y) ← Rr	Нет	

Продолжение табл. 3

Команда	Описание	Флаги	Примечание
st –Y, Rr	Пересылка данных из регистра Rr в ОЗУ с адресом, указанным регистром Y, и предшествующим декрементом Y, ОЗУ(Y–1) ← Rr, Y ← Y–1	Нет	
st Y+, Rr	Пересылка данных из регистра Rr в ОЗУ с адресом, указанным регистром Y, и последующим инкрементом Y, ОЗУ(Y) ← Rr, Y ← Y+1	Нет	
std Y+k, Rr	Пересылка данных из регистра Rr в ОЗУ с адресом, указанным регистром Y со смещением k, ОЗУ(Y+k) ← Rr	Нет	k=(0 – 63)
ld Rd, –Z	Пересылка данных в регистр Rd из ОЗУ с адресом, указанным регистром Z, и предшествую-щим декрементом Z Rd ← ОЗУ(Z–1), Z ← Z–1	Нет	
ld Rd, Z+	Пересылка данных в регистр Rd из ОЗУ с адресом, указанным регистром Z, и последующим инкрементом Z, Rd ← ОЗУ(Z), Z ← Z+1	Нет	
ldd Rd, Z+k	Пересылка данных в регистр Rd из ОЗУ с адресом, указанным регистром Z со смещением k, Rd ← ОЗУ(Z+k)	Нет	k=(0 – 63)
st Z, Rr	Пересылка данных из регистра Rr в ОЗУ с адресом, указанным регистром Z, ОЗУ(Z) ← Rr	Нет	
st –Z, Rr	Пересылка данных из регистра Rr в ОЗУ с адресом, указанным регистром Z, и предшествующим декрементом Z, ОЗУ(Z–1) ← Rr, Z ← Z–1	Нет	

Окончание табл. 3

Команда	Описание	Флаги	Примечание
st Z+, Rr	Пересылка данных из регистра Rr в ОЗУ с адресом, указанным регистром Z, и последующим инкрементом Z, ОЗУ(Z) ← Rr, Z ← Z+1	Нет	
std Z+k, Rr	Пересылка данных из регистра Rr в ОЗУ с адресом, указанным ре-гистром Z со смещением k, ОЗУ(Z+k) ← Rr	Нет	k=(0 – 63)
lpm	Пересылка данных в регистр R0 из FLASH по адресу, указанному ре-гистром Z, R0 ← PM(Z)	Нет	

Команды пересылки данных также реализуют стандартный набор операций:

- пересылка байта данных между регистрами общего назначения (mov);
- пересылка байта данных между регистром общего назначения и регистром ввода-вывода (in, out);
- пересылка байта данных (с прямой или косвенной адресацией) между регистром общего назначения и ОЗУ (ld, st) или стеком (pop, push).

Команда загрузки константы в регистр допустима только для регистров R16–R31. Команды пересылки данных флаги не формируют, формирование флагов по содержимому байтов данных при необходимости можно выполнить командами сравнения.

В мнемонических обозначениях команд можно использовать физические адреса регистров или, что обычно более удобно и наглядно, стандартные символьные имена, описанные в файле "8535def.inc". Например, команды **out $08, R16** и **out ACSR, R16** имеют один и тот же смысл: пересылка содержимого регистра общего назначения R16 в регистр управления АЦП.

Необходимо помнить, что управление всеми аппаратными средствами микроконтроллера производится пересылкой байтов управления, сформированных программно в регистрах общего назначения, в соответствующие регистры файла регистров ввода-вывода командой **out**. Анализ текущего состояния аппаратных средств можно производить чтением содержимого регистров ввода-вывода командой **in**. Как указывалось ранее, вектора прерываний для координации работы аппаратных средств должны обеспечивать программное управление параметрами и режимами их работы. Необ-

ходимый для выполнения этих функций обмен данными производится командами in, out.

Пересылка данных между регистрами общего назначения и EEPROM данных производится через соответствующие регистры ввода-вывода специальной процедурой. Команда **lpm** позволяет загружать в регистр R0 константы, которые хранятся в памяти программ (FLASH). Адрес FLASH должен указываться в регистре Z.

Каждая ячейка FLASH хранит два байта, поэтому младший бит регистра Z (бит 0) определяет байт для пересылки в регистр R0 (0 – младший байт, 1 – старший байт). Адрес FLASH находится в регистре Z, начиная с бита 1. То есть бит 1 соответствует младшему биту адреса FLASH, что необходимо учитывать при записи адреса в регистр Z. Адрес FLASH для регистра Z можно указать выражением Z=2*(адрес FLASH), а бит 0 регистра Z формировать дополнительно в зависимости от того, какой байт двухбайтовой ячейки FLASH пересылается в регистр R0.

4.1.3. Команды управления

Таблица 4

Команда	Описание	Флаги	Примечание
rjmp k	Безусловный переход (изменение PC на k), PC ← PC+1+k	Нет	–2048 < k < 2048
ijmp	Безусловный переход с адресацией по Z, PC ← Z	Нет	
rcall k	Вызов подпрограммы (изменение PC на k), STACK← PC, PC← PC+1+k	Нет	–2048 < k < 2048
icall	Вызов подпрограммы с адресацией по Z, STACK← PC, PC← Z	Нет	
ret	Возврат из подпрограммы, PC ← STACK	Нет	
reti	Возврат из прерывания, PC ← STACK, I = 1	I	

Команда	Описание	Флаги	Примечание
brbs b, k	Условный переход (изменение PC на k), если бит b в SREG установлен, PC ← PC+1+k, если SREG(b)=1	Нет	$-64 \le k \le 63$
brbc b, k	Условный переход (изменение PC на k), если бит b SREG очищен, PC ← PC+1+k, если SREG(b)=0	Нет	$-64 \le k \le 63$
breq k	Условный переход (изменение PC на k), если равно, PC ← PC+1+k, если флаг Z=1	Нет	$-64 \le k \le 63$
brne k	Переход (изменение PC на k), если не равно, PC ← PC+1+k, если флаг Z=0	Нет	$-64 \le k \le 63$
brcc k (brlo k)	Условный переход на k по флагу C=0, PC ← PC+1+k, если флаг C=0	Нет	$-64 \le k \le 63$
brmi k	N=1	Условный переход на k по флагу N	Нет	$-64 \le k \le 63$
brpl k	N=0			
brvs k	V=1	Условный переход на k по флагу V	Нет	$-64 \le k \le 63$
brvc k	V=0			
brge k	S=1	Условный переход на k по флагу S	Нет	$-64 \le k \le 63$
brlt k	S=0			
brhs k	H=1	Условный переход на k по флагу H	Нет	$-64 \le k \le 63$
brhc k	H=0			
brts k	T=1	Условный переход на k по флагу T	Нет	$-64 \le k \le 63$
brtc k	T=0			
brie k	I=1	Условный переход на k по флагу I	Нет	$-64 \le k \le 63$
brid k	I=0			

Окончание табл. 4

Команда	Описание	Флаги	Примечание
Следующие команды пропускают одну операцию в программе, если условие выполняется			
sbrs Rr, b	Пропустить, если бит b в регистре Rr установлен, PC ← PC+2(3), если Rr(b) = 1	Нет	
sbrc Rr, b	Пропустить, если бит b в регистре Rr очищен, PC ← PC+2(3), если Rr(b) = 0	Нет	
sbis P, b	Пропустить, если бит b в регистре ввода-вывода P установлен, PC ← PC+2(3), если P(b) = 1	Нет	P = (0 – 31)
cpse Rd, Rr	Пропустить, если Rd = Rr, PC ← PC+2(3), если Rd = Rr	Нет	

Команды управления необходимы для программной реализации алгоритмов со сложной и неоднозначной последовательностью действий. Эти команды позволяют обеспечить циклическое повторение отдельных фрагментов программ, ветвление программ с анализом выполнения определенных условий, вызов подпрограмм, в том числе и для обслуживания прерываний, возврат из подпрограмм с корректным продолжением выполнения прерванной программы. Основные действия при исполнении команд управления – изменение состояния программного счетчика (PC), которым производится адресация во флэш-памяти (PM).

Особенностью команд управления микроконтроллеров AVR является не абсолютная, а относительная адресация передачи управления (изменение текущего содержимого программного счетчика на величину указываемого в команде параметра k). Компиляторы при обработке текстов программ не требуют обязательного определения численного значения этого параметра.

При программировании на ассемблере в командах управления параметр k может быть заменен меткой. Ограничения на величину смещения k в ко-

мандах **rjmp k, rcall k** в микроконтроллере 8535 практически не имеют никакого значения, так как размер адресного пространства памяти программ РМ составляет 4096 двухбайтовых ячеек (8 Кбайт). Если в командах условных переходов требуется смещение больше, чем указано в примечании, можно использовать для перехода промежуточную команду rjmp.

Пример:

```
    cp  R0, R21      ; сравнить содержимое регистров R0, R21
    breq inter       ; если равно, перейти к inter (смещение не более 63)
     .  .  .  .
inter:  rjmp fin     ; перейти к fin (смещение в пределах адр. простр. РМ)
     .  .  .  .
    fin: nop         ;программа выполняется по равенству R0=R21
     .  .  .  .
```

Команды "пропустить" (последние пять команд) выполняют условный переход со смещением только на одну команду. В комбинации с командами rjmp или rcall позволяют реализовать переход в любую часть программы аналогично приведенному выше примеру.

4.1.4. Команды преобразования битов в регистрах

Таблица 5

Команда	Описание	Флаги	Примечание
sbr Rd, K	Установить в регистре Rd биты по маске K	Нет	d = (16 – 31)
cbr Rd, K	Очистить в регистре Rd биты по маске K	Нет	d = (16 – 31)
sbi P, b	Установить в регистре ввода-вывода P бит b, P(b)=1	Нет	P = (0 – 31)
cbi P, b	Очистить в регистре ввода-вывода P бит b, P(b)=0	Нет	P = (0 – 31)
bst Rr, b	Пересылка бита b из регистра Rr во флаг T, T ← Rr(b)	T	
bld Rd, b	Пересылка бита b в регистр Rd из флага T, Rr(b) ← T	Нет	
swap Rd	Поменять местами старшую и младшую тетрады в регистре Rd	Нет	

Команда	Описание	Флаги	Примеч.
ser Rd	Установить все биты регистра Rd, Rd ← $ff	Нет	d = (16 – 31)
clr Rd	Очистить все биты регистра Rd, Rd ← $00	VNZ	
bset b	Установить бит b в регистре SREG, SREG(b) ← 1	SREG(b)	
bclr b	Очистить бит b в регистре SREG, SREG(b) ← 0	SREG(b)	
clc	Очистить флаг C, C ← 0	C	
sez	Установить флаг Z, Z ← 1	Z	
clz	Очистить флаг Z, Z ← 0	Z	
sen	Установить флаг N, N ← 1	N	
cln	Очистить флаг N, N ← 0	N	
sev	Установить флаг V, V ← 1	V	
clv	Очистить флаг V, V ← 0	V	
ses	Установить флаг S, S ← 1	S	
cls	Очистить флаг S, S ← 0	S	
seh	Установить флаг H, H ← 1	H	
clh	Очистить флаг H, H ← 0	H	
set	Установить флаг T, T ← 1	T	
clt	Очистить флаг T, T ← 0	T	
sei	Установить флаг I, I ← 1	I	
cli	Очистить флаг I, I ← 0	I	

Команды преобразования битов позволяют определять значения отдельных битов в регистрах общего назначения (с ограничениями, указанными в примечании), в первых 32 регистрах ввода-вывода (0-31) и в регистре состояния SREG. Флаг глобального разрешения прерывания I управляет всеми аппаратными прерываниями микроконтроллера и может в лю-

бой части программы определять разрешение (команда **sei**) или запрет (команда **cli**) прерываний. Флаг копирования бита T обеспечивает передачу любого бита одного регистра в любой бит другого (или того же самого) регистра общего назначения без изменения остальных битов:

```
bst R12, 4   ;пересылка бита 4 регистра R12 во флаг T
bld R16, 0   ;пересылка флага T в бит 0 регистра R16
```

Команды **sbr** и **cbr** по маске K определяют биты в регистре Rd, соответствующие единичным значениям битов константы K:

```
cbr R16, 0b10100110      ;очистка битов 7, 5, 2, 1 в регистре R16
sbr R17, $0f             ;установка 4 младших битов в регистре R17
```

Команды **sbi** и **cbi** позволяют задавать значения произвольных битов в регистрах ввода-вывода (0-31), например, **sbi PORTC, PC0** устанавливает бит 0 регистра PORTC.

4.1.5. Прочие команды

Таблица 6

Команда	Описание	Флаги	Примечание
nop	Нет операции	Нет	
sleep	Переход в режим sleep	Нет	
wdr	Сброс сторожевого таймера	Нет	

Эти команды служат для управления специальными режимами микроконтроллера. Команда **sleep** переводит микроконтроллер в состояние покоя (sleep mode) с пониженным энергопотреблением и ограниченным функционированием его элементов, если в управляющем регистре ввода-вывода MCUCR этот режим разрешен. Команда **wdr** осуществляет программный сброс сторожевого таймера и используется в контрольных точках программы, когда работа сторожевого таймера разрешена.

4.2. Компилятор ассемблера микроконтроллеров AVR

Компилятор транслирует исходные коды с языка ассемблера в объектный код. Полученный объектный код можно использовать в симуляторе AVR Studio. Компилятор также генерирует код, который может быть непосредственно записан в память микроконтроллера. Текст программы для компилятора должен строго соответствовать рассмотренным ниже синтаксическим правилам. Приведенные ранее в примерах фрагменты программ соответствуют этим правилам.

Компилятор работает с исходными файлами, содержащими в каждой строке метку, команду или директиву и комментарии. Строка не должна быть длиннее 120 символов. Любая строка может начинаться с метки, которая является набором символов, заканчивающимся двоеточием. Метки используются для указания места, в которое передаётся управление при переходах, а также для задания имён переменных. Комментарий имеет следующую форму:

```
; [Текст]
```

Текст после точки с запятой и до конца строки игнорируется компилятором, эти комментарии обычно объясняют назначение и смысл выполняемых операций, что существенно облегчает понимание реализуемых в программе алгоритмов работы.

Входная строка может иметь одну из четырёх форм (позиции в квадратных скобках необязательны):

```
[метка:]    директива    [операнды]    [;Комментарий]
[метка:]    команда      [операнды]    [;Комментарий]
;Комментарий
Пустая строка
```

Перечень команд с допустимыми операндами (регистры, адреса, константы или метки) приведен выше. **Каждая команда преобразуется компилятором в соответствующий код операции программы микроконтроллера.** Мнемоника команд и операндов должна строго соответствовать указанной в описаниях. Текст в приведенных ранее примерах использования команд составлен в соответствии с требованиями компилятора. Регистр для символов (прописные, строчные) и пробелы для компилятора значения не имеют.

Директивы не транслируются непосредственно в код. Они используются компилятором для указания положения в программной памяти, определения макросов, инициализации памяти и т.д. **Признаком директивы служит точка перед текстом директивы**, например: **.CSEG**. Ниже дано описание директив компилятора, которые в дальнейшем будут применяться в текстах программ.

Директивы **.CSEG, .DSEG, .ESEG** указывают соответственно на память программ, память данных, EEPROM (EEPROM), к которым относится дальнейший текст программы (по умолчанию тексты программ предназна-

чены для памяти .CSEG). Эти директивы необходимы в связи с использованием трех раздельных адресных пространств в памяти микроконтроллеров AVR. Вместе с директивой **.ORG k** определяют конкретные физические адреса ячеек памяти для хранения кодов программы:

```
    .CSEG

    .ORG $120
    more: cpi   r19, $00    ; проверка старших битов АЦП
          brne  repAD1          ; перейти, если не равно
          mov   r17, r18
          lsl r17               ;сдвиг влево для проверки бита 7
          brcs  repAD1          ;перейти, если бит 7 не равен 0
    . . . . . .

    .ORG $090
repAD1:   nop
      . . . . . .
```

В приведенном примере коды будут записаны в памяти программ (.CSEG); метка more и код этой строки записываются по адресу $120, последующие коды – в следующие по порядку ячейки памяти. Вторая директива .ORG $090 указывает, что код, соответствующий метке repAD1, компилятор должен разместить по адресу $090.

Директивы **.DB** (определить байты), **.DW** (определить слова) позволяют записать в память программ или EEPROM массивы однобайтных или двухбайтовых констант соответственно. Могут использоваться метки для ссылок на адреса памяти, по которым директивы записывают константы. Вместе с директивой ORG позволяют задавать конкретные физические адреса памяти для хранения констант. Директивы DB и DW должны содержать список констант, разделенных запятыми:

```
    .ESEG

    const1:   .  DB  02, 255, 0b01010101, -128, 0xaa

    .CSEG

    .ORG $380

    led:.  DW  $063f, $4f5b, 6d66, a77d, $0000
```

Массив однобайтовых констант (DB) записан в EEPROM, адреса EEPROM определяются компилятором, адрес первой константы массива (02) соответствует метке const1. Массив двухбайтовых констант (DW) записан в память программ, адрес первой константы ($063f) соответствует метке led и определяется директивой .ORG $380.

Директива **.INCLUDE "имя файла"** (присоединить файл) позволяет включить в текст программы другие файлы. Например: **.INCLUDE "8535def.inc"** включает в текст программы файл определения стандартных символических имен регистров и переменных. Этот файл позволяет использовать в текстах программ в качестве операндов символические имена, определяемый директивами компилятора DEF, EQU, SET.

Директивы **.DEF** (установить регистру имя), **.EQU** (установить имени постоянное значение), **.SET** (установить имени переменное значение) позволяют установить для компилятора соответствие символьных имен и операндов в текстах программ. Директива **.EQU** указывает для имени значение, которое не может быть изменено, а директива **.SET** допускает изменение этого значения другой директивой. Значения символических имен могут задаваться в виде числовых констант или выражений. Более подробно с правилами записи выражений можно ознакомиться в документации фирмы ATMEL. Примеры применения директив:

```
.EQU  MCUCR=$35      ;установить постоянное значение
.EQU  MCUSR=$34      ;   для переменной с указанным
.EQU  PORF  =0       ;   символическим именем
.DEF  XL=r26         ;установить регистру R26 имя XL
.DEF  XH=r27         ;установить регистру R27 имя XH
```

Первая директива в данном примере назначает переменной с именем MCUCR постоянное значение шестнадцатеричного числа $35 (в десятичном формате 53), в командах, где требуется указать это значение операнда, можно приводить либо численное значение ($35), либо символическое имя (MCUCR). Например, команды out $35, r26 и out MCUCR, XL абсолютно эквивалентны; в первом варианте для операндов используются их физические значения, во втором варианте – их символические имена. Как отмечалось ранее, использование символических имен существенно упрощает мнемонику команд, при подготовке программ гораздо легче работать с именами операндов, чем запоминать большое количество адресов, числовых констант и т.п. Целесообразно также использовать стандартные симво-

лические имена, назначаемые специальными файлами для каждой модели микроконтроллера.

Полезными при подготовке текстов программ могут быть директивы **.NOLIST** (исключить из листинга), **.LIST** (включить в листинг). Эти директивы не содержат операндов и позволяют исключить из листинга программы текст, находящийся ниже директивы **.NOLIST** и выше директивы **.LIST.** Например, целесообразно исключать из листинга программы файл определения стандартных символических имен 8535def.inc, так как он не содержит исполняемых кодов и занимает в листинге объем нескольких страниц текста.

Компилятор позволяет использовать в тексте программ выражения, которые могут состоять из операндов, операторов и функций. Формат выражений – 4 байта, т.е. они могут содержать значения переменных, укладывающихся в 4-х байтовый диапазон.

Операндами в выражениях могут быть числа; символические имена, определенные директивами .SET и .EQU; метки программы (каждая метка в тексте программы соответствует определенному числу – физическому адресу памяти); PC – текущее состояние программного счетчика.

Над операндами в выражениях можно выполнять различные математические операции, задаваемые операторами. Операторы имеют стандартные для компьютерных приложений обозначения:

- арифметические операторы: + , – , * , / ;
- операторы сдвига: влево << , вправо >> ;
- логические операторы: & , && , | , || , ! , ^ ;
- операторы отношений: == , != , < , <= , > , >= .

Элементы выражений (операнды с операторами) могут быть заключены в круглые скобки, как и в любых математических преобразованиях.

Выражения могут быть аргументами следующих функций:

- LOW(выражение) – младший байт выражения,
- HIGH(выражение) – второй байт выражения,
- BYTE2(выражение) то же что и функция HIGH,
- BYTE3(выражение) – третий байт выражения,
- BYTE4(выражение) – четвёртый байт выражения,
- LWRD(выражение) – биты 0-15 выражения,
- HWRD(выражение) – биты 16-31 выражения,
- PAGE(выражение) – биты 16-21 выражения,
- EXP2(выражение) – 2 в степени (выражение),
- LOG2(выражение) – целая часть log2(выражение).

Примеры выражений:

```
ldi   r16, LOW(RAMEND)   ;RAMEND=0x025f, LOW(RAMEND)=0x5f
out   SPL, r16           ;инициализация младшего байта указателя стека
ldi   r16, HIGH(RAMEND)  ; HIGH(RAMEND)=0x02
out   SPH, r16           ;инициализация старшего байта указателя стека
ldi   r16, (1<<OCIE1A)|(1<<TOIE0)   ;загрузить в r16 константу
        ;в виде выражения, определяющего разрешения прерываний
        ;таймеров 0 и 1 с использованием символических имен флагов
        ;этих прерываний,
out   TIMSK, r16         ;передать константу из r16 в регистр TIMSK.
```

Очевидно, что назначение команды с константой в форме выражения более понятно, чем с константой в виде числа 0x11.

Для команды lpm адрес FLASH указывается в регистре Z со сдвигом на 1 разряд, так как младший бит регистра Z определяет байт, который передается по этой команде из FLASH в r0 (1 – старший байт, 0 – младший байт). Если адрес FLASH задан меткой (например, меткой ledbcd), программа чтения из FLASH будет следующая:

```
ldi   ZL, LOW(ledbcd*2)    ;определение адреса FLASH в Z со сдвигом
ldi   ZH, HIGH(ledbcd*2)   ;на 1 разряд умножением метки на 2
lpm                ;чтение в r0 младшего байта константы по метке
mov   r1, r0       ;пересылка младшего байта в r1
inc   ZL                 ;адрес старшего байта (если нет переноса в ZH)
lpm                ;чтение в r0 старшего байта константы по метке
```

Примеры показывают, что использование выражений делает текст программы более наглядным и освобождает от необходимости вычисления параметров для команд вручную.

5. РЕАЛИЗАЦИЯ ТИПОВЫХ ФУНКЦИЙ

При подготовке программ необходимо учитывать, что многие стандартные задачи могут решаться или встроенными аппаратными средствами микроконтроллера, или программно (процессором микроконтроллера под управлением соответствующей программы). В предыдущих главах приведено описание всех устройств микроконтроллера и показано, что аппаратные ресурсы, их режимы и параметры предназначены для выполнения операций преобразования и формирования сигналов, которые часто используются в задачах управления техническими объектами.

Использование аппаратных средств для реализации стандартных функций, как правило, намного эффективнее. Во-первых, ресурсы процессора освобождаются для реализации других необходимых функций, а это приводит и к повышению быстродействия, и к уменьшению объемов рабочих программ. Во-вторых, различные аппаратные средства работают параллельно и независимо друг от друга, самостоятельно выполняя необходимые функции, и это также повышает быстродействие и существенно расширяет функциональные возможности. В-третьих, система векторных прерываний позволяет достаточно просто организовать корректное взаимодействие и управление всеми ресурсами из рабочей программы микроконтроллера.

При составлении алгоритмов решения задач рекомендуется придерживаться следующей последовательности:

1. Определить функции и состав средств микроконтроллера, необходимых для решения задачи, отдавая предпочтение аппаратной реализации требуемых функций.
2. Для выбранных средств реализации определить необходимые рабочие режимы, параметры этих режимов, процедуры инициализации, средства и процедуры организации взаимодействия, а также необходимые для управления и координации работы ресурсы.
3. В начальной части рабочей программы описать вектора прерываний и процедуры инициализации с указанием режимов и параметров для всех используемых аппаратных средств микроконтроллера.
4. В основной части программы организовать выполнение всех программно реализуемых функций, включая процедуры взаимодействия как аппаратно реализуемых, так и программно реализуемых функций.
5. Дополнить основную программу подпрограммами обслуживания векторов прерываний, необходимых для программной поддержки работы аппаратных средств.
6. Проверить корректность работы всех средств, используемых для решения задачи. Конфликты между параллельно работающими средствами микроконтроллера наиболее часто могут возникать из-за конкуренции

при использовании общих ресурсов (регистров общего назначения, ячеек памяти данных, времени работы процессора и т.п.).

Предварительная проверка рабочей программы производится компилятором при ассемблировании и симулятором инструментального пакета AVR Studio. Необходимо учитывать, что компилятор проводит только синтаксический контроль текста программы, а моделирование работы аппаратных средств симулятором возможно в весьма ограниченных пределах.

5. 1. Примеры программ для микроконтроллеров AVR

Далее приводятся примеры программ для реализации стандартных функций.

Пример 1. Передать в последовательном формате из микроконтроллера во внешний регистр 4 байта данных, находящихся в RAM по адресам: $060, $061, $062, $063, с формированием для внешнего регистра сигнала завершения цикла передачи – load.

Вариант "a". Задачу передачи данных в последовательном формате можно решить использованием интерфейса микроконтроллера SPI, для адресации данных в ОЗУ целесообразно использовать один из регистров косвенной адресации, например, регистр X.

Для передачи данных необходимо определить в SPI режим master с формированием сигналов SCK (PB7), MOSI (PB5). Вход MISO (PB6) в данной задаче не используется, сигнал завершения цикла может быть сформирован только программно на одном из выходов, например, PB4. Регистр управления интерфейса SRCR должен быть инициализирован следующим образом: 0b01011100, старший бит SPIE этого регистра должен программно устанавливаться в начале цикла и очищаться в конце цикла. Необходимо также использовать вектор прерывания SPITC с адресом $00a.

В приведенном далее тексте программы начальная часть, как и во всех последующих примерах программ, содержит файл определения символических имен и обязательную процедуру инициализации стека. Текст программы (и всех остальных программ) записан в соответствии с требованиями компилятора.

```
; программа примера 1, вариант "a"
.NOLIST
.INCLUDE "8535def.inc"
.LIST
.DEF temp = r16     ;определить символическое имя temp регистру r16
.CSEG
.ORG $000
   rjmp    init             ;прерывание по reset
.ORG $00a
```

```
   rjmp    spitc        ;прерывание по завершению передачи байта
.ORG $011

init:    ldi    temp, low(RAMEND)
      out     SPL, temp
      ldi     temp, high(RAMEND)
      out  SPH, temp     ; определить в указателе стека адрес RAMEND
      ldi     temp, $f0
      out  DDRB, temp     ; 4 старших бита PORTB на вывод данных
      ldi     temp, 0b01011100
      out     SPCR, temp     ; инициализация SPI

;запуск цикла с выводом первого байта в последовательном формате
start: clr    XH
      ldi     XL, $60    ; занести начальный адрес данных в регистр X
      ld temp, X+  ; занести в temp байт с постинкрементом адреса в X
      out     SPDR, temp     ; переслать байт в регистр данных SPI
      sbi     SPCR, SPIE     ; установить разрешение прерывания SPI
      sei          ; установить флаг общего разрешения прерываний

; бессодержательный бесконечный цикл, может быть заменен любой
; программой
main:    nop
      rjmp    main

;п/программа для вектора прерывания SPI
spitc: cpi    XL, $64
      breq end       ; перейти, если цикл завершен
      ld temp, X+  ; занести в temp байт с постинкрементом адреса в X
      out  SPDR, temp       ; переслать байт в регистр данных SPI
      reti              ; возврат из прерывания
; завершение цикла
end:  sbi  PORTB, PB4      ; установить сигнал load
      cbi  SPCR, SPIE      ; запретить прерывание SPI
      cbi  PORTB, PB4  ; сбросить сигнал load
      reti              ; возврат из прерывания
```

В этом варианте программы в течение цикла выдачи 4-х байт данных в последовательном формате процессор используется для запуска цикла (метка start -- 6 команд), три раза для обслуживания прерывания (метка spitc – 5 команд) и для завершения цикла (метки spitc, end – 6 команд). То есть для решения задачи требуется всего 27 команд, остальные операции реализуется аппаратно.

Процедура вывода данных может быть запущена программно командой передачи управления на метку start в любой момент времени после завершения цикла. Пока работает интерфейс SPI можно программно реализовать параллельное выполнение других функций. В приведенном примере это отражено бессодержательным циклом (метка main), который можно заменить какой-либо прикладной программой.

Вариант "б". Задачу передачи данных в последовательном формате можно решить программно, передавая на те же выходы порта B программно формируемые сигналы, для адресации данных в RAM также будем использовать регистр косвенной адресации X.

```
; программа примера 1, вариант "б"
.NOLIST
.INCLUDE "8535def.inc"
.LIST
.DEF temp = r16    ;определить символическое имя temp регистру r16
 .CSEG
.ORG $000
   rjmp    init          ;прерывание по reset

.ORG $011
init:     ldi    temp, low(RAMEND)
      out    SPL, temp
      ldi    temp, high(RAMEND)
      out  SPH, temp     ; определить в указателе стека адрес RAMEND
      ldi  temp, $f0
      out  DDRB, temp  ; 4 старших бита PORTB на вывод данных
      sbi    PORTB, PB7

;запуск цикла с выводом первого байта в последовательном формате
start: cli            ; очистить флаг разрешения прерываний
      clr    XH
      ldi    XL, $60    ; занести начальный адрес данных в регистр X
byte: ld temp, X+  ; занести в temp байт с постинкрементом адреса в X
      ldi    r17, $08       ; определить количество бит в байте
bit:      sbrs  temp, 7     ; пропустить, если бит 7 установлен
      cbi    PORTB, PB5       ; очистить PB5
      sbrc  temp, 7     ; пропустить, если бит 7 очищен
      sbi    PORTB, PB5       ; установить PB5
      cbi    PORTB, PB7       ; очистить PB7 (сигнал SCK)
      lsl    temp        ; сдвиг влево байта данных
      sbi    PORTB, PB7       ; установить PB7 (сигнал SCK)
```

```
    dec   r17          ; декремент числа бит для вывода
    brpl  bit          ;вернуться к метке bit, если не все
                       ;биты в байте переданы
    cpi   XL, $64      ; сравнить адрес с конечным для цикла
    brne  byte         ; перейти, если цикл не завершен
; завершение цикла
end: sbi  PORTB, PB4   ; установить сигнал load
    nop
    cbi   PORTB, PB4   ; сбросить сигнал load
    sei                ; общее разрешение прерываний

; бессодержательный бесконечный цикл, может быть заменен любой программой
main:  nop
    rjmp main
```

В варианте программной реализации фрагмент программы для вывода каждого бита (метка bit – 9 команд) повторяется 32 раза, фрагмент запуска цикла (метка start – 3 команды) и завершения цикла (метка end – 4 команды) – 1 раз, фрагмент подготовки байта для вывода (метка byte – 2 команды) – 4 раза, т.е. необходимо выполнить 303 команды.

Таким образом, вариант программной реализации по объему операций, выполняемых процессором, существенно больше и, самое главное, полностью загружает процессор, не позволяя параллельно с выводом данных выполнять какие-либо другие функции.

Пример 2. Выполнить преобразование аналоговых сигналов, поступающих на входы PA0, PA1, PA2 и PA3, в цифровой код; преобразовать выходные коды АЦП в 8-битовый формат, пренебрегая значениями младших разрядов; записать полученные коды в ОЗУ по адресам: $060, $061, $062, $063.

Для решения задачи будем использовать АЦП в режиме однократного преобразования с повышенной тактовой частотой, так как младшие разряды выходного кода АЦП не используются. Вектор прерывания АЦП необходимо использовать для приема и обработки выходного кода АЦП. Номер бита порта A, с которого поступает сигнал для преобразования, определяется регистром ADMUX. Для адресации будем использовать регистр X.

```
; программа примера 2
.NOLIST
.INCLUDE "8535def.inc"
.LIST
.DEF temp = r16    ; определить символическое имя temp регистру r16
.CSEG
.ORG $000
     rjmp   init          ; прерывание по reset
.ORG $00e
     rjmp   adc      ; прерывание по завершению преобразования АЦП

.ORG $011
init:    ldi   temp, low(RAMEND)
     out  SPL, temp
     ldi   temp, high(RAMEND)
     out  SPH, temp     ; определить в указателе стека адрес RAMEND
     clr   temp
     out   DDRA, temp       ; определить все биты порта А на ввод
     ser   temp
     out  PORTA, temp   ; определить пассивный высокий уровень
                        ; сигнала для всех битов порта А
     ldi   temp, 0b10000100
     out   ADCSR, temp      ; инициализация АЦП

start: clr   XH
     ldi   XL, $60    ; занести начальный адрес данных в регистр X
     out  ADMUX, XH ; определить номера бита 0 порта А
                        ; в мультиплексоре
     sbi  ADCSR, ADSC  ; запустить преобразование в АЦП
     sbi   ADCSR, ADIE; разрешить прерывание АЦП
     sei          ; общее разрешение прерываний

; бессодержательный бесконечный цикл, может быть заменен любой программой
main:   nop
     rjmp   main

;п/программа обработки прерывания АЦП
adc:     in r20, ADCL       ; занести младший байт кода АЦП в r20
     in r21, ADCH      ; занести старший байт кода АЦП в r21
     lsr   r21
     ror   r20
```

```
    lsr   r21       ; двукратный сдвиг вправо с переносом
                    ; из старшего байта
    ror   r20       ; в младший для преобразования в 8-битовый формат
    st X+, r20      ; сохранить в ОЗУ по адресу X с постинкрементом
    cpi   XL, $64
    breq end        ; перейти к завершению цикла после РАЗ
    mov   temp, XL
    cbr   temp, $f8
    out   ADMUX, temp   ; занести в мультиплексор номер
                        ; текущего канала
    sbi   ADCSR, ADSC   ; запустить преобразование в АЦП
    reti            ; завершить прерывание

end: cbi  ADCSR, ADIE ; запретить прерывание АЦП
    reti            ; завершить прерывание
```

Пример 3. Реализовать таймер, отображающий с помощью светодиодов в 8-разрядном двоичном коде время замкнутого состояния контактов кнопки включения с дискретностью 0,5 с, предусмотреть возможность сброса таймера в исходное состояние и индикацию переполнения таймера.

Для управления таймером необходим отсчет временных интервалов по 0,5 с. от момента замыкания кнопки включения (в дальнейшем Кн1) до ее размыкания, а также – вторая кнопка (в дальнейшем Кн2) для управления сбросом. Для индикации выходных сигналов таймера светодиоды можно подключить через токоограничивающие резисторы к одному из портов микроконтроллера, например, порту В. Контроль кнопок Кн1 и Кн2 наиболее просто реализуется использованием сигналов внешних прерываний Int0 (PD2), Int1 (PD3). Полагая, что приоритет сброса выше, подключим замыкающий контакт Кн2 между PD2 и общим проводом, а замыкающий контакт Кн1 – между PD3 и общим проводом. Для индикации переполнения таймера дополнительный светодиод можно подключить к свободному биту порта D, например, PD1.

При тактовой частоте микроконтроллера 8 МГц временные интервалы 0,5 с можно получить в канале сравнения А таймера-счетчика 1 с коэффициентом деления тактовой частоты – 1024, кодом канала сравнения А – 3906 ($0f42) и сбросом таймера 1 по сигналу сравнения канала А. Для инициализации этого режима в регистре TCCR1A все биты очищаются, а в регистре TCCR1B записывается константа 0b00001000. Запуск таймера производится установкой трех младших битов TCCR1B, останов таймера очисткой этих же трех битов. Для возврата таймера в исходное состояние необходимо очистить биты регистров TCNT1H, TCNT1L. Используемые век-

торы прерываний: Int0 ($001), Int1 ($002), Tim1_CompA ($006). Таким образом, временной интервал 0,5 с и контроль состояния кнопок реализуются аппаратными средствами с использованием соответствующих векторов прерываний, остальные необходимые функции – программно.

```
; программа примера 3
.NOLIST
.INCLUDE "8535def.inc"
.LIST
.DEF temp = r16    ; определить символическое имя temp регистру r16
.CSEG
.ORG $000
     rjmp   init          ; прерывание по reset
     rjmp   restim        ; прерывание по сбросу таймера (int0)
     rjmp   timon         ; прерывание по включению таймера (int1)
.ORG $006
     rjmp   half     ; прерывание для формирования
                     ; интервала полсекунды

.ORG $011
init:    ldi   temp, low(RAMEND)
     out   SPL, temp
     ldi   temp, high(RAMEND)
     out   SPH, temp      ; определить в указателе стека адрес RAMEND
     ser   temp
     out   DDRB, temp     ; порт B на вывод
     out   PORTD, temp    ; пассивный высокий в порте D
     clr   temp
     out  PORTB, temp     ; очистить биты порта B (сброс)
     out   DDRD, temp     ; порт D на ввод
     sbi   DDRD, PD1   ; бит PD1 (переполнение) на вывод
     out   MCUCR, temp ; внешние прерывания по низкому уровню
     ldi   temp, $08
     out   TCCR1B, temp   ; разрешение сброса таймера 1 по каналу
                          ; сравнения A
     ldi   r18, $42
     ldi   r19, $0f
     out  OCR1AH, r19
     out   OCR1AL, r18    ; запись кода сравнения (3906) канала A
     ldi   temp, $10
     out   TIMSK, temp    ; разрешение прерывания канала A
     ldi   temp, $c0
     out   GIMSK, temp    ; разрешение внешних прерываний
```

```
    sei             ; общее разрешение прерываний

; бессодержательный бесконечный цикл, может быть заменен любой программой
main:  nop
       rjmp   main

restim: clr   r20
       out  PORTB, r20      ; очистить биты порта B (сброс)
       cbi   PORTD, PD1   ; очистить индикацию переполнения
       out   TCNT1H, r20
       out   TCNT1L, r20      ; сброс таймера 1
stop: in    temp, TCCR1B
       cbr   temp, $07
       out   TCCR1B, temp    ; остановить таймер 1
       reti                  ; завершить прерывание

timon:  in    temp, TCCR1B
       sbr   temp, $07
       out   TCCR1B, temp    ; запустить таймер 1
       reti                  ; завершить прерывание

half: sbic  PORTD, PD3       ; пропустить, если Кн1 нажата
       rjmp   stop          ; перейти к остановке таймера 1
       inc   r20
       out   PORTB, r20     ; инкремент индикатора кода таймера
       breq carry           ; перейти к формированию переполнения
       reti                 ; завершить прерывание

carry:  sbi   PORTD, PD1       ; установить бит индикации переполнения
       rjmp   stop            ; перейти к остановке таймера 1
```

В приведенных выше примерах показаны различные варианты программно-аппаратной реализации некоторых стандартных функций. Эти примеры в определенной мере иллюстрируют возможности микроконтроллеров AVR в решении задач управления техническими объектами. Каждая из рассмотренных программ использует только незначительную часть программно-аппаратных ресурсов микроконтроллера и может рассматриваться как фрагмент более полной и объемной рабочей программы. Это отражено в примерах в виде бессодержательного бесконечного цикла, который может быть заменен какой-либо прикладной программой.

5.2. Микроконтроллерная система управления температурой

В качестве примера реализации всех необходимых функций управления микроконтроллером AT90S8535 рассмотрим задачу поддержания микроклимата в помещении, алгоритм решения которой приведен в главе 1 на рис. 3. Подготовка программы для микроконтроллера требует уточнения некоторых деталей и распределения функций между различными средствами реализации.

В исходных данных задачи приведены только сведения, необходимые для составления общего алгоритма работы. Будем полагать, что управление охлаждением и подогревом производится независимыми стандартными логическими сигналами (1 – включено, 0 – выключено). Контролируемая температура измеряется двумя аналоговыми датчиками, оценка температуры производится по среднему арифметическому значению их двух выходных сигналов. Параметры аналоговых сигналов и опорное напряжение АЦП обеспечивают формирование кодов температуры в следующем формате: старший байт выходного кода АЦП соответствует значению температуры с точностью до 0,1 °С (например, код 215 (0xd7) – 21,5 °С). Пороговые значения температур следующие: t_{min}=17,0 °С, t_{max}=22,0 °С.

Любая система управления должна содержать средства управления и индикации. В минимальном варианте необходимо производить включение и выключение системы кнопками управления, индикацию состояния системы и температуры в помещении. Тестовые режимы для проверки работоспособности системы рассматривать не будем, хотя реальная система функции диагностики также должна поддерживать.

Кнопки управления можно подключить к свободным линиям параллельных портов микроконтроллера с формированием общего сигнала внешнего прерывания по схеме, показанной на рис. 7. Такая схема позволяет при необходимости подключить к этому сигналу внешнего прерывания и другие

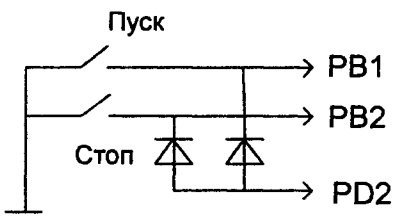

Рис. 7. Схема подключения кнопок управления

цепи. При обработке сигналов кнопок необходимо программно определить приоритеты сигналов при одновременном нажатии обеих кнопок ("защита от дурака"). Можно также указать, что в этой задаче применение алгоритма "антидребезга" необязательно, так как первое же замыкание кнопки пере-

водит систему в соответствующий режим. Последующие замыкания контактов будут только подтверждать команду перехода в этот режим.

Для индикации можно применять стандартные семисегментные индикаторы, так как требуется отображение цифровых кодов температуры. Индикация режима работы тоже может выполняться этими индикаторами. Выключением индикации можно отображать режим "Стоп", а включенная индикация температуры может отображать режим "Пуск". Для индикации состояния "подогрев", "охлаждение" можно использовать соответствующие сигналы управления.

Управление цифровыми индикаторами через параллельные порты возможно, но нецелесообразно: индикация каждой цифры требует отдельного байта управления. Рациональнее байты управления для индикации передавать через последовательный интерфейс SPI во внешние регистры, а с их выходов в параллельном формате управлять индикаторами (рис. 8).

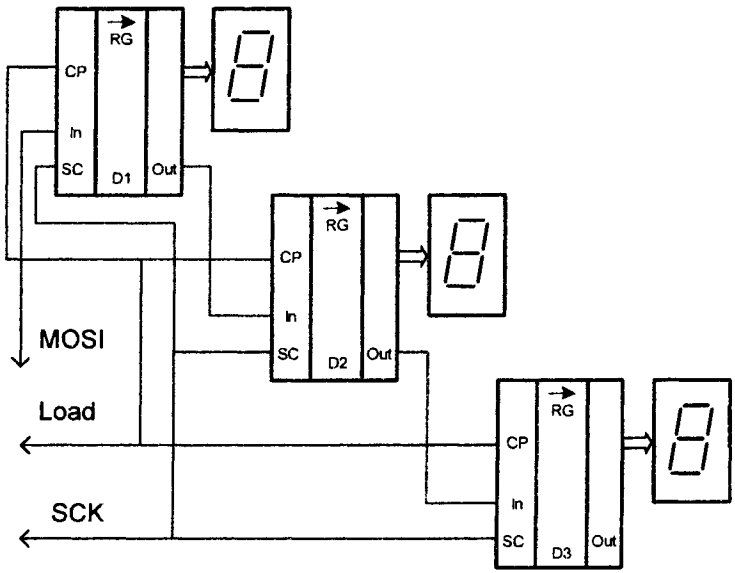

Рис. 8. Схема управления индикаторами

Регистры индикаторов – стандартные однобайтовые регистры сдвига D1-D3 с входом последовательной записи In, тактовыми импульсами последовательной записи SC, сигналом управления для вывода в параллельном формате CP и параллельными выходами, к которым подключены цифровые индикаторы. Выходные сигналы MOSI интерфейса SPI, работающе-

го в режиме master, поступают на вход In регистра D1, тактовые импульсы SCK – на входы тактирования последовательной записи SC всех регистров.

Алгоритм вывода должен обеспечивать последовательную выдачу трех байтов управления индикаторами, передача байтов должна начинаться с младшего разряда кода температуры. После трех циклов работы интерфейса SPI первый байт будет записан в регистре D3, второй байт – в регистреD2 и третий байт – в регистре D1. Для выдачи принятого трехбайтового кода в индикаторы необходимо программно выполнить передачу сигнала Load, который интерфейсом SPI не формируется.

На следующем этапе необходимо выбрать средства микроконтроллера для реализации необходимых функций управления и определить их параметры и режимы работы. Первое, что требуется оценить, достаточны ли программно-аппаратные ресурсы микроконтроллера для решения задачи? Предварительная оценка времени, требуемого для выполнения всех необходимых функций, – единицы миллисекунд (блок-схема алгоритма на рис. 3 позволяет определить ориентировочное число команд программы: от нескольких сот до 1 – 2 тысяч). Если ресурсы недостаточны, может потребоваться применение других, более сложных средств. Тепловая инерционность помещений с постоянными времени десятки и более секунд обычно не предъявляет особых требований к быстродействию. Алгоритм управления не требует выполнения каких-либо сложных операций преобразования данных, поэтому можно сделать вывод о достаточных функциональных возможностях микроконтроллера для решения данной задачи.

Мало того, предварительная оценка быстродействия микроконтроллера и инерционности объекта управления показывает, что имеет смысл искусственно увеличить интервалы времени для контроля температуры. Это позволяет и обеспечить эффективное управление, и резервировать ресурсы микроконтроллера для решения дополнительных задач. Выберем периодичность запуска функций управления, равной 1 с. Такая периодичность практически не повлияет на эффективность управления температурной (тепловые постоянные времени объекта управления на 1-2 порядка выше), обеспечит необходимую оперативность и наглядность отображения температуры на индикаторе и достаточно просто реализуется с помощью таймеров микроконтроллера.

Дополнительного уточнения также требуют пороговые значения температуры t_{min} и t_{max} . Для единственного порогового значения и для включения, и для выключения, весьма вероятен режим, при котором в каждом очередном цикле контроля будет производиться переключение сигнала управления (включение-отключение-включение-отключение и т.д.). Если ввести гистерезис (различные пороговые значения для включения t_1 и выключения t_2), такой режим постоянных переключений станет маловероятным. Разность пороговых значений выберем ±1°C, тогда t_{max1}=23,0 °C; t_{max2}=21,0 °C; t_{min1}=16,0 °C; t_{min2}=18,0 °C.

Выберем тактовую частоту микроконтроллера, которая определяется внешними частотозадающим элементами и может лежать в диапазоне 0 – 8 МГц. Типичное значение тактовой частоты – 4 МГц. Ввод аналоговых сигналов от датчиков температуры будем производить через АЦП микроконтроллера, используя входы порта А: PA0 – датчик 1, PA1 – датчик 2. Переключение входов АЦП программное с помощью регистра мультиплексора ADMUX. В байте управления АЦП необходимо определить режим однократного преобразования с программным запуском, разрешение прерывания и тактовую частоту с коэффициентом деления 32. Выходной код АЦП будем преобразовывать в однобайтовый формат и передавать для дальнейшей обработки вектором прерывания АЦП. Сразу же можно выбрать линии ввода-вывода для передачи сигналов управления: PA7 – управление охлаждением, PA6 – управление подогревом. Таким образом, в порте А две линии должны настраиваться на ввод (датчики) и две линии – на вывод (сигналы управления).

Интерфейс SPI также требует выбора параметров настройки для регистра управления SPCR. Управление интерфейсом также будем производить через его вектор прерывания с заданием режима master, порядка формирования байта данных при выводе, параметров сигнала SCK и его частоты. Высокая скорость передачи данных не требуется, поэтому выберем коэффициент деления для тактовой частоты интерфейса – 16. Интерфейс передает выходные сигналы через PB5 (MOSI), PB7 (SCK). Для сигнала управления индикацией (Load) выберем PB0. Эти три бита порта В должны быть настроены на вывод данных с заданием пассивного высокого уровня сигнала.

Алгоритм вывода кодов индикации через интерфейс SPI должен включать преобразование байтов кода температуры в коды индикации трех десятичных цифр с десятичной точкой после второй цифры. Управление индикаторами требует формирования специальных кодов для каждой десятичной цифры. Эти коды индикации можно хранить в FLASH, а для вывода кодов индикации выполнять программное преобразование трех десятичных цифр температуры в три байта кодов управления индикаторами с помощью констант из FLASH. Первым в интерфейсе SPI должен поступать код индикации младшей цифры, после вывода третьего кода индикации необходимо программно сформировать сигнал управления на PB0.

Для обработки сигналов от кнопок управления используем внешнее прерывание INT0. Определим режим формирования этого запроса прерывания в регистре MCUCR по падающему фронту сигнала PD2 и разрешение этого прерывания в регистре GIMSK. Вектор прерывания должен анализировать состояния сигналов PB1 и PB2, определяя выполняемую операцию управления (Пуск, Стоп). Для реализации этих функций необходимо задать режим ввода на PD2, PB1 и PB2 с обязательным формированием пассивного высокого уровня. Если кнопки не замкнуты, на всех этих входах – высокий

уровень сигнала, при нажатии любой кнопки – низкий уровень на PD2 и выводе порта B, к которому подключена нажатая кнопка (рис. 7)

Запуск функций управления с периодичностью 1 с можно производить таймером 0, используя его вектор прерывания по переполнению. Максимальный период для этого прерывания зависит от коэффициента деления тактовой частоты: $t=1024*256/f_{CLK}=65{,}536$ мс. Увеличение периода до требуемой величины можно выполнить программно, запуская функции управления каждым 16-м вызовом вектора прерывания таймера, тогда период $T=16*t=1{,}05$ с. Необходимым дополнением алгоритма обработки прерывания становится счетчик количества прерываний, с помощью которого и определяется номер текущего прерывания.

Функциональная схема системы управления температурой с указанным распределением функций в микроконтроллере AT90S8535 приведена на рис. 9.

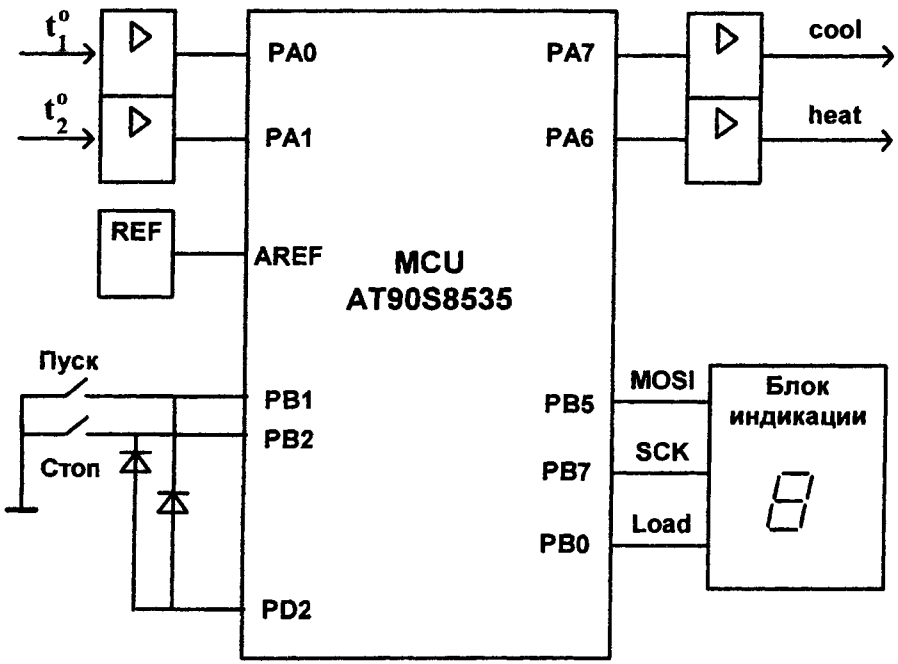

Рис. 9. Функциональная схема системы управления температурой

Аналоговые сигналы двух датчиков температуры поступают на входы PA0, PA1 микроконтроллера через согласующие усилители. Эти усилители выполняют масштабирование выходных сигналов датчиков и при необхо-

димости ограничение спектра сигналов для снижения уровня помех и повышения точности преобразования в цифровой код. Вместе с прецизионным источником опорного напряжения (REF) эти усилители определяют диапазон измеряемой температуры и значения кодов на выходе АЦП микроконтроллера. В соответствии с погрешностями дискретизации АЦП точность формирования сигналов всеми аналоговыми устройствами должна быть на уровне 0,1-0,2 %.

Усилители сигналов управления PA6, PA7 должны формировать сигналы cool, heat с требуемыми параметрами, преобразуя стандартные логические сигналы на выходах микроконтроллера. В зависимости от характеристик сигналов управления эти усилители могут содержать ключи на биполярных или полевых транзисторах, ключи с электромагнитными реле или твердотельные реле. Кнопки управления и блок индикации показаны ранее на рис. 7, 8.

Взаимодействие устройств микроконтроллера при выполнении всех указанных функций достаточно эффективно обеспечивается применением системы прерываний. Отдельные элементы задачи управления могут быть описаны в форме подпрограмм для векторов прерываний и подпрограмм для реализации некоторых дополнительных функций. Логическая организация программы микроконтроллера стандартна, программа состоит из следующих элементов: описание векторов прерываний, инициализация всех используемых компонентов микроконтроллера, комплекс подпрограмм для реализации всех необходимых функций. Вызов подпрограмм производится соответствующими аппаратными средствами микроконтроллера.

В режиме Стоп все функции системы управления отключены, аппаратные и программные средства в состоянии покоя, микроконтроллер может выполнять какие-нибудь другие функции. При нажатии кнопки "Пуск" подпрограммой вектора внешнего прерывания запускается первый цикл управления температурой и таймер 0 для выполнения повторных циклов контроля с периодом около 1 с. Каждый цикл контроля начинается с запуска АЦП, по его вектору прерывания вводится код текущей температуры. Код температуры поступает в интерфейс SPI для управления индикацией и одновременно на программную обработку с реализацией алгоритма управления, приведенного на рис. 3. После завершения этих функций, которые загружают процессор в течение единиц миллисекунд, выполнение алгоритмов управления приостанавливается до повторного запуска цикла таймером 0. Нетрудно показать, что задача управления требует менее 1% общего времени работы процессора.

Работа системы управления может быть прервана на любом этапе нажатием кнопки "Стоп", подпрограмма вектора внешнего прерывания должна остановить работу всех используемых средств: АЦП, таймера 0, через SPI

выдать коды выключения индикаторов и затем отключить интерфейс. Далее приводится текст программы, реализующий этот алгоритм работы.

```
; Программа управления температурой
.NOLIST
.INCLUDE "8535def.inc"
.LIST

; определить символические имена
.DEF    templ = r16
.DEF    temph = r17
.DEF    saveF = r4
.DEF    counT = r18
.DEF    rcnt = r19
.EQU    Mind = 0x100    ; RAM для кодов индикации
.EQU    Madc = 0x090    ; RAM для кода АЦП
.CSEG

; вектора прерываний
.ORG $000
      rjmp    init            ; прерывание по reset
      rjmp    key_cnt         ; прерывание int0
.ORG $009
      rjmp    cycle    ; прерывание таймера 0 для запуска цикла
      rjmp    spi_stc     ; прерывание SPI
.ORG $00e
      rjmp    adc_cmt         ; прерывание АЦП

.ORG $011
;инициализация
init:     ldi    templ, low(RAMEND)
      ldi    temph, high(RAMEND)
      out    SPH, temph
      out  SPL, templ    ;определить в указателе стека адрес RAMEND
   ;порты ввода-вывода
      ldi    templ, 0b11000000
      clr    temph
      out    PORTA, temph
      out    DDRA, templ
      ser    templ
      out  PORTB, templ
      out    DDRB, templ      ; порт B на вывод
      out  PORTD, templ
```

```
    out   DDRD, temph      ; порт D на ввод
  ;внешнее прерывание INT0
    ldi   templ, (1<<ISC01)|(0<<ISC00)
    out   MCUCR, templ     ; INT0 по падающему фронту
    ldi   templ, (1<<INT0)
    out   GIMSK, templ     ; разрешение INT0
  ;прерывание таймера 0
    ldi   templ, (1<<TOIE0)
    out   TIMSK, templ
  ;параметры SPI
    ldi   templ, 0b11011001
    out SPCR, templ
  ;параметры АЦП
    clr   templ
    out   ADMUX, templ   ;канал PA0 для АЦП
    ldi   templ, 0b10001101
    out   ADCSR, templ

    sei                  ; общее разрешение прерываний
    rcall   ind_off      ; выключение индикации температуры

; бессодержательный бесконечный цикл, может быть заменен любой программой
main:  nop
    rjmp   main

ind_off:
  ;вывод нулевых байтов в SPI для отключения индикации
    clr   templ
    ldi   XL, low(Mind)
    ldi   XH, high(Mind)   ;начальный адрес RAM для кодов SPI
    st    X+, templ
    st    X+, templ
    st    X+, templ
    ldi   XL, low(Mind) ;восстановление в X начального адреса
    rcall   spi_stc     ;старт передачи байтов индикации
    ret

;INT0, кнопки управления
key_cnt:
  ;сохранение в стеке регистров и флагов
    push   templ
    push   temph
```

```
      push    saveF
      in      saveF, SREG

   ;обработка сигналов "Стоп", "Пуск"
      in      templ, PINB
      sbrs    templ, PB2      ; контроль сигнала СТОП
      rjmp    cnt_off         ; перейти к процедуре СТОП
      sbrs    templ, PB1      ; контроль сигнала ПУСК
      rjmp    cnt_on          ; перейти к процедуре ПУСК
;если оба сигнала в 1, завершить без изменения режима

key_out:
   ; завершение с восстановлением из стека флагов и регистров
      out     SREG, saveF
      pop     saveF
      pop     temph
      pop     templ
      reti

cnt_on:
   ;переход в режим ПУСК
      tst     rcnt
      brne    key_out    ;завершить по признаку режима ПУСК
      sbr     rcnt, 0x80 ;установить признак режима ПУСК в rcnt
      ldi     temph, 0x05
      out     TCCR0, temph  ;запуск таймера циклов контроля
      clr     counT      ;сброс счетчика для формирования цикла
      out     ADMUX, counT - ;канал PA0 для АЦП
      sbi     ADCSR, ADSC ;запуск АЦП для 1 цикла контроля
      cbi     PORTA, PA7   ;выключение охлаждения
      cbi     PORTA, PA6   ;выключение подогрева
      rjmp    key_out       ;завершение процедуры

cnt_off:
   ;переход в режим СТОП
      tst     rcnt
      breq    key_out    ;завершить по признаку режима СТОП
      clr     rcnt     ;сбросить признаки режима ПУСК в rcnt
      clr     temph
      out     TCCR0, temph  ;останов таймера циклов контроля
      out     TCNT0, temph  ;сброс таймера циклов контроля
      clr     counT      ;сброс счетчика для формирования цикла
      cbi     PORTA, PA7   ;выключение охлаждения
```

```
      cbi     PORTA, PA6   ;выключение подогрева
   wait_spi:
      cpi     XL, low(Mind) ;контроль вывода байтов индикации
      breq cnt_end    ;продолжение процедуры, если вывод завершен
      sei   ;разрешить прерывания при ожидании
      rjmp    wait_spi   ;возврат к контролю завершения цикла вывода

   cnt_end:
      rcall   ind_off    ;выключить индикацию температуры для СТОП
      rjmp    key_out       ;завершение процедуры

;периодический запуск цикла управления в режиме ПУСК
cycle:
      push    saveF
      in    saveF, SREG
      tst   rcnt     ;контроль режима ПУСК
      breq cyc_out    ;завершить без запуска для СТОП
      cpi   counT, 15    ;контроль количества прерываний таймера 0
      breq cyc_do     ;запуск цикла по 16 прерыванию
      inc   counT     ;инкремент количества прерываний
      rjmp    cyc_out    ;завершить без запуска (<16)

   cyc_do:
      clr   counT          ;очистка счетчика прерываний
      out   ADMUX, counT   ;канал PA0 для АЦП
      sbi       ADCSR, ADSC ;запуск АЦП, начало цикла контроля

   ;завершение с восстановлением флагов
cyc_out:
      out     SREG, saveF
      pop     saveF
      reti

;прерывание по завершению преобразования АЦП
adc_cmt:
      push    templ
      push    temph
      push    saveF
      in    saveF, SREG
      tst   rcnt
      breq    adc_out    ;завершить по признаку режима СТОП
```

```
;прием кода температуры
   in    templ, ADCL       ; младший байт кода АЦП
   in    temph, ADCH       ; старший байт кода АЦП
   lsr   temph
   ror   templ
   lsr   temph      ; двукратный сдвиг вправо с переносом
   ror   templ      ; для преобразования в 8-битовый формат
   in    temph, ADMUX  ;номер канала ?
   tst   temph         ;
   brne tmtr         ;перейти к обработке, если PA1 (датчик 2)
   sbi   ADMUX, ADMUX0 ;установить канал PA1
   sbi    ADCSR, ADSC       ; запустить преобразование в АЦП
   sts   Madc, templ    ; сохранить код канала PA0 (датчик 1)
   rjmp    adc_out       ;завершить процедуру для повторения АЦП

tmtr:
; преобразование кода температуры
   lds   temph, Madc   ;считать из RAM код PA0
   add      templ, temph;сложить коды PA1 и PA0
   ror      templ          ;сдвинуть вправо с переносом сложения

;анализ кода температуры
   cpi   templ, 230    ;порог включения охлаждения
   brcs     thresh1    ;перейти к следующему порогу, если <
   rcall    cool     ;включить охлаждение
   rjmp     bcdt     ;завершить анализ

thresh1:
   cpi   templ, 210    ; порог выключения охлаждения
   brcc bcdt     ; завершить анализ, если >

hresh2:
   cpi   templ, 160    ; порог включения подогрева
   brcc     thresh3    ; перейти к следующему порогу, если >
   rcall    cool     ;включить подогрев
   rjmp     bcdt     ;завершить анализ

hresh3:
   cpi   templ, 180    ; порог выключения подогрева
   brcs     bcdt     ; завершить анализ, если <
   rcall    offch    ;выключить средства изменения температуры
   rjmp     bcdt     ; завершить анализ
```

```
;включение охлаждения
cool:
sbis    PINA, PA7       ;пропустить, если включено
sbi     PORTA, PA7      ;включить охлаждение
ret     ;возврат
;включение подогрева
heat:
sbis    PINA, PA6       ;пропустить, если включено
sbi     PORTA, PA6      ;включить подогрев
ret     ;

;выключение охлаждения и подогрева
offch:
sbic    PINA, PA7       ; пропустить, если выключено
cbi     PORTA, PA7      ; выключить охлаждение
sbic    PINA, PA6       ; пропустить, если выключено
cbi     PORTA, PA6      ; выключить подогрев
ret     ;

;преобразование кода в bcd формат для индикации
bcdt:
clr     temph       ;очистка регистра для десятичной цифры
hunst:              ;определение цифры сотен
cpi     templ, 100;
brcs    hunout      ;перейти к сохранению сотен для SPI
subi    templ, 100;вычесть 100
inc     temph       ;инкремент сотен
rjmp    hunst       ;повторить цикл вычислений

hunout:     ; код индикации и запись в RAM для SPI
rcall   code_ind

decst:
clr     temph       ; очистка регистра для десятичной цифры
cpi     templ, 10   ; определение цифры десятков
brcs    decout      ; перейти к сохранению сотен для SPI
subi    templ, 10   ; вычесть 10
inc     temph       ; инкремент десятков
rjmp    decst     ; повторить цикл вычислений

decout:     ; код индикации и запись в RAM для SPI десятков
rcall   code_ind
```

```
        ; код индикации и запись в RAM для SPI десятков
    mov     temph, templ;передать цифру единиц
    rcall   code_ind  ;
    rjmp    str_spi     ;перейти к старту SPI

    code_ind:
    ldi   ZL, low(cdind*2)  ;
    ldi   ZH, high(cdind*2); адрес FLASH для кодов индикации
    add     ZL, temph    ;смещение адреса по цифре в temph
    lpm                 ;прочитать код индикации для цифры из temph
    st    -X, r0        ;сохранить код индикации в RAM для SPI
    ret                 ;возврат

  ;старт SPI для передачи кодов индикации
  str_spi:
    ldi   XL, low(Mind)
    ldi   XH, high(Mind)   ;начальный адрес RAM для кодов SPI
    rcall   spi_stc     ;старт передачи байтов индикации

adc_out:
  ; завершение с восстановлением из стека флагов и регистров
    out     SREG, saveF
    pop     saveF
    pop     templ
    reti

;передача кодов управления индикацией
spi_stc:
    push    templ
    push    temph
    push    saveF
    in    saveF, SREG

    cpi   XL, low(Mind+3)  ;
    breq    load     ;
    ld    templ, X+ ;
    cpi   XL, low(Mind+2)  ;
    brne    skip        ;
    sbr     templ, 0x80     ;
    skip:
    out     SPDR, templ;
```

```
spi_out:
   ; завершение с восстановлением из стека флагов и регистров
      out     SREG, saveF
      pop     saveF
      pop     templ
      reti
   load:
      cbi     PORTB, PB0
      nop
      nop
      nop          ;
      sbi     PORTB, PB0   ;
      rjmp    spi_out    ;

.ORG 0x0800
cdind:      ;коды индикации цифр, начиная с нуля
.dB    0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07, 0x7f, 0x6f

;end of file
```

Фрагменты листинга программы, создаваемого компилятором AVR Studio, приведены ниже. Первые 5 символов – адрес FLASH, который присваивается компилятором коду каждой команды в программе. Следующие 4 символа – код команды в hex-формате, далее в каждой строке следует исходный текст программы.

```
          .LIST

          ;определить символические имена
          .DEF    templ = r16
          .DEF    temph = r17
          .DEF    saveF = r4
          .DEF    counT = r18
          .DEF    rcnt = r19
          .EQU    Mind = 0x100  ; RAM для кодов индикации
          .EQU    Madc = 0x090  ; RAM для кода АЦП
          .CSEG

          ; вектора прерываний
          .ORG $000
000000 c010          rjmp  init              ; прерывание по reset
000001 c035          rjmp      key_cnt         ; прерывание int0
          .ORG $009
000009 c055          rjmp  cycle     ; прерывание таймера 0 для запуска цикла
00000a c0b2          rjmp  spi_stc       ; прерывание SPI
```

```
            .ORG $00e
00000e c05e         rjmp  adc_cmt     ; прерывание АЦП

            .ORG $011
            ; инициализация
000011 e50f   init:     ldi templ, low(RAMEND)
000012 e012         ldi    temph, high(RAMEND)
000013 bf1e         out    SPH, temph
000014 bf0d         out SPL, templ   ;  определить  в  указателе  стека  адрес
RAMEND
            ; порты ввода-вывода
000015 ec00         ldi  templ, 0b11000000
000016 2711         clr  temph
000017 bb1b         out    PORTA, temph
000018 bb0a         out    DDRA, templ
000019 ef0f         ser templ
00001a bb08         out  PORTB, templ
00001b bb07         out    DDRB, templ     порт B на вывод
00001c bb02         out  PORTD, templ
00001d bb11         out    DDRD, temph     ; порт D на ввод
            ; внешнее прерывание INT0
00001e e002         ldi    templ, (1<<ISC01)|(0<<ISC00)
00001f bf05         out MCUCR, templ       ; INT0 по падающему фронту
000020 e400         ldi    templ, (1<<INT0)
000021 bf0b         out    GIMSK, templ    ; разрешение INT0
            ; прерывание таймера 0
000022 e001         ldi    templ, (1<<TOIE0)
000023 bf09         out    TIMSK, templ
            ; параметры SPI
000024 ed09         ldi    templ, 0b11011001
000025 b90d         out  SPCR, templ
            ; параметры АЦП
000026 2700         clr    templ
000027 b907         out    ADMUX, templ  ; канал PA0 для АЦП
000028 e80d         ldi    templ, 0b10001101
000029 b906         out    ADCSR, templ

00002a 9478         sei                  ; общее разрешение прерываний
00002b d002         rcall     ind_off    ; выключение индикации температуры

            ; бессодержательный бесконечный цикл, может быть заменен любой про-
граммой
00002c 0000     main:   nop
00002d cffe         rjmp     main

            ind_off:
              ; вывод нулевых байтов в SPI для отключения индикации
00002e 2700         clr templ
```

```
00002f e0a0          ldi XL, low(Mind)
000030 e0b1          ldi XH, high(Mind)   ; начальный адрес RAM для кодов SPI
000031 930d          st  X+, templ
000032 930d          st  X+, templ
000033 930d          st  X+, templ
000034 e0a0          ldi XL, low(Mind); восстановление в X начального адреса
000035 d087          rcall     spi_stc      ; старт передачи байтов индикации
000036 9508          ret

          ;INT0, кнопки управления
          key_cnt:
            ; сохранение в стеке регистров и флагов
000037 930f          push  templ
000038 931f          push  temph
000039 924f          push  saveF
00003a b64f          in    saveF, SREG
          ; обработка сигналов "Стоп", "Пуск"
00003b b306          in    templ, PINB
00003c ff02          sbrs      templ, PB2     ; контроль сигнала СТОП
00003d c012          rjmp      cnt_off        ; перейти к процедуре СТОП
00003e ff01          sbrs      templ, PB1     ; контроль сигнала ПУСК
00003f c005          rjmp      cnt_on         ; перейти к процедуре ПУСК
          ; если оба сигнала в 1, завершить без изменения режима

          key_out:
            ; завершение с восстановлением из стека флагов и регистров
000040 be4f          out       SREG, saveF
000041 904f          pop       saveF
000042 911f          pop       temph
000043 910f          pop       templ
000044 9518          reti

          cnt_on:
            ; переход в режим ПУСК
000045 2333          tst       rcnt
000046 f7c9          brne      key_out      ; завершить по признаку режима ПУСК
000047 6830          sbrrcnt, 0x80   ; установить признак режима ПУСК в rcnt
000048 e015          ldi    temph, 0x05
000049 bf13          out       TCCR0, temph ; запуск таймера циклов контроля
00004a 2722          clr    counT    ; сброс счетчика для формирования цикла
00004b b927          out       ADMUX, counT  ; канал PA0 для АЦП
00004c 9a36          sbi    ADCSR, ADSC     ; запуск АЦП для 1 цикла контроля
00004d 98df          cbi    PORTA, PA7   ; выключение охлаждения
00004e 98de          cbi    PORTA, PA6   ; выключение подогрева
00004f cff0        rjmp     key_out          ; завершение процедуры
```

```
            cnt_off:
              ; переход в режим СТОП
000050 2333           tst     rcnt
000051 f371           breq      key_out      ; завершить по признаку режима СТОП
000052 2733           clr     rcnt       ; сбросить признаки режима ПУСК в rcnt
000053 2711           clr     temph
000054 bf13           out       TCCR0, temph  ; останов таймера циклов контроля
000055 bf12           out       TCNT0, temph  ; сброс таймера циклов контроля
000056 2722           clr     counT    ; сброс счетчика для формирования цикла
000057 98df           cbi     PORTA, PA7   ; выключение охлаждения
000058 98de           cbi     PORTA, PA6   ; выключение подогрева
              wait_spi:
000059 30a0           cpi     XL, low(Mind); контроль вывода байтов индикации
00005a f011           breq     cnt_end  ;продолжение процедуры, если вывод завершен

00005b 9478           sei     ; разрешить прерывания при ожидании
00005c cffc         rjmp      wait_spi ; возврат к контролю завершения цикла вывода

              cnt_end:
00005d dfd0           rcall    ind_off ;выключить индикацию температуры для СТОП
00005e cfe1           rjmp      key_out          ; завершение процедуры



. . . . . .
. . . . . .
. . . . . .


        ;передача кодов управления индикацией
            spi_stc:
0000bd 930f           push  tempI
0000be 924f           push  saveF
0000bf b64f           in      saveF, SREG

0000c0 30a3           cpi     XL, low(Mind+3) ;
0000c1 f051           breq      load      ;
0000c2 910d           ld      tempI, X+   ;
0000c3 30a2           cpi     XL, low(Mind+2) ;
0000c4 f409             brne    skip          ;
0000c5 6800           sbr     tempI, 0x80     ;
                skip:
0000c6 b90f           out       SPDR, tempI   ;
0000c7 b90f           out       SPDR, tempI   ;
```

```
            spi_out:
              ; завершение с восстановлением из стека флагов и регистров
0000c8 be4f         out     SREG, saveF
0000c9 904f         pop     saveF
0000ca 910f         pop     templ
0000cb 9518         reti

              load:
0000cc 98c0         cbi   PORTB, PB0
0000cd 0000         nop
0000ce 0000         nop             ;
0000cf 9ac0         sbi   PORTB, PB0  ;
0000d0 cff7         rjmp  spi_out     ;


            .ORG 0x0800
            cdind:      ; коды индикации цифр, начиная с нуля
000800 063f
000801 4f5b
000802 6d66
000803 077d
000804 6f7f     .dB     0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07, 0x7f, 0x6f

            ;End Of File


RESOURCE USE INFORMATION
------------------------

Notice:
The register and instruction counts are symbol table hit counts,
and hence implicitly used resources are not counted, eg, the
'lpm' instruction without operands implicitly uses r0 and z,
none of which are counted.

x,y,z are separate entities in the symbol table and are
counted separately from r26..r31 here.

.dseg memory usage only counts static data declared with .byte

AT90S8535 register use summary:
r0 : 1  r1 : 0   r2 :  0   r3 :  0   r4 : 16   r5 :  0   r6 :  0   r7 :  0
r8 : 0 r9 :  0   r10:  0   r11:  0   r12:  0   r13:  0   r14:  0   r15:  0
r16: 50  r17: 29  r18:  7   r19:  6   r20:  0   r21:  0   r22:  0   r23:  0
r24:  0   r25:  0   r26:  5  r27:  2   r28:  0   r29:  0   r30:  2   r31:  1
x :  5   y :  0   z :  0
Registers used: 11 out of 35 (31.4%)
```

```
AT90S8535 instruction use summary:
adc  : 0   add  : 4   adiw : 0   and  : 0   andi : 0   asr  : 0
bclr : 0   bld  : 0   brbc : 0   brbs : 0   brcc : 4   brcs : 8
breq : 12  brge : 0   brhc : 0   brhs : 0   brid : 0   brie : 0
brlo : 0   brlt : 0   brmi : 0   brne : 4   brpl : 0   brsh : 0
brtc : 0   brts : 0   brvc : 0   brvs : 0   bset : 0   bst  : 0
cbi  : 14  cbr  : 0   clc  : 0   clh  : 0   cli  : 0   cln  : 0
clr  : 20  cls  : 0   clt  : 0   clv  : 0   clz  : 0   com  : 0
cp   : 0   cpc  : 0   cpi  : 18  cpse : 0   dec  : 0   eor  : 0
icall: 0   ijmp : 0   in   : 16  inc  : 6   ld   : 21  dd   : 0
ldi  : 32  lds  : 2   lpm  : 2   lsl  : 0   lsr  : 4   mov  : 2
neg  : 0   nop  : 6   or   : 2   ori  : 0   out  : 48  pop  : 20
push : 20  rcall: 20  ret  : 10  reti : 8   rjmp : 40  rol  : 0
ror  : 6   sbc  : 0   sbci : 0   sbi  : 14  sbic : 4   sbis : 4
sbiw : 0   sbr  : 2   sbrc : 0   sbrs : 4   sec  : 0   seh  : 0
sei  : 4   sen  : 0   ser  : 2   ses  : 0   set  : 0   sev  : 0
sez  : 0   sleep: 0   st   : 8   std  : 0   sts  : 2   sub  : 0
subi : 4   swap : 0   tst  : 10  wdr  : 0
Instructions used: 37 out of 100 (37.0%)

AT90S8535 memory use summary [bytes]:
Segment  Begin    End      Code  Data  Used  Size  Use%
----------------------------------------------------------
[.cseg] 0x000000 0x00100a  394    10   400  8192  4.9%
[.dseg] 0x000060 0x000060    0     0     0   512  0.0%
[.eseg] 0x000000 0x000000    0     0     0   512  0.0%

Assembly complete, 0 errors, 1 warnings
```

В выходном сообщении компилятора указано, что программа содержит 394 команды, используя менее 5% адресного пространства FLASH. В алгоритме работы все необходимые функции управления реализованы векторами прерываний, основной программный цикл (метка main) при работе системы управления температурой не используется, с его помощью могут решаться любые дополнительные задачи. Управление аппаратными средствами микроконтроллера, обеспечивающими выполнение всех необходимых функций, достаточно просто и эффективно производится программной обработкой прерываний, суммарное время загрузки процессора подпрограммами обработки прерываний в каждом цикле работы составляет менее 1%. Таким образом, суммарное использование программных ресурсов микроконтроллера (процессора, памяти, регистров и т.п.) не превышает нескольких процентов.

Оценку использования аппаратных ресурсов микроконтроллера можно произвести по функциональной схеме (рис. 9). Для передачи сигналов необходимы 10 линий ввода-вывода из 32. Большая часть входов АЦП свободна, также не используются внешнее прерывание INT1, таймеры 1 и 2,

компаратор и интерфейс UART. Следует отметить, что значительный объем резервных ресурсов микроконтроллера не является признаком неэффективного решения задачи. Напротив, эти резервные ресурсы практически всегда должны оставаться для возможной модернизации: изменения алгоритмов работы или введения каких-либо дополнительных функций. Некоторые варианты модернизации функций системы управления приведены ниже.

Программу несложно доработать при изменении и усложнении требуемых алгоритмов управления. Например, управление нагревом и охлаждением будет более эффективным, если использовать не релейное управление (включено/выключено), а ПИД-регулятор с управлением тепловой мощностью по отклонению температуры. Аналоговый сигнал управления в этом случае можно формировать таймером 1 или 2 в режиме модулятора ШИМ.

Пороговые значения температур в программе фиксированные, так как определены в виде констант. Несложно реализовать процедуру ввода, индикации и хранения в EEPROM изменяемых пороговых значений температуры. В этой процедуре можно использовать уже имеющийся цифровой индикатор температуры, а для управления вводом необходимо подключить дополнительные кнопки на свободные входы портов микроконтроллера по схеме, которая аналогична приведенной на рис. 7.

Неиспользованный интерфейс UART позволяет создать средства обмена данными с другими устройствами управления. Например, в рассматриваемую систему управления температурой можно ввести средства поддержки дистанционного контроля работы через интерфейс UART. Функции дистанционного контроля могут выполняться из персонального компьютера с передачей команд управления параметрами и режимами работы и с запросом данных о текущем состоянии системы.

5.3. Средства подготовки программ

Подготовка файла программы микроконтроллера для процедуры внутрисистемного программирования может выполняться инструментальным пакетом **AVR Studio 4.X.** Пользовательский интерфейс пакета AVR Studio стандартный для Window's-приложений (рис. 10) и содержит полоску меню, панель инструментов и рабочую область. Для активизации функций AVR Studio, необходимых для работы с текстом программы, должны быть выполнены следующие операции:

1. Для работы с текстом программы необходимо создать проект (project). Проект создается с помощью меню **Project** (пункт **New Project**). В появившемся на рабочем поле окне (рис. 11) указать тип проекта – AVR Assembler, имя и путь папки, имя проекта. Кнопкой "Next" необходимо

Рис. 10. Интерфейс AVR Studio

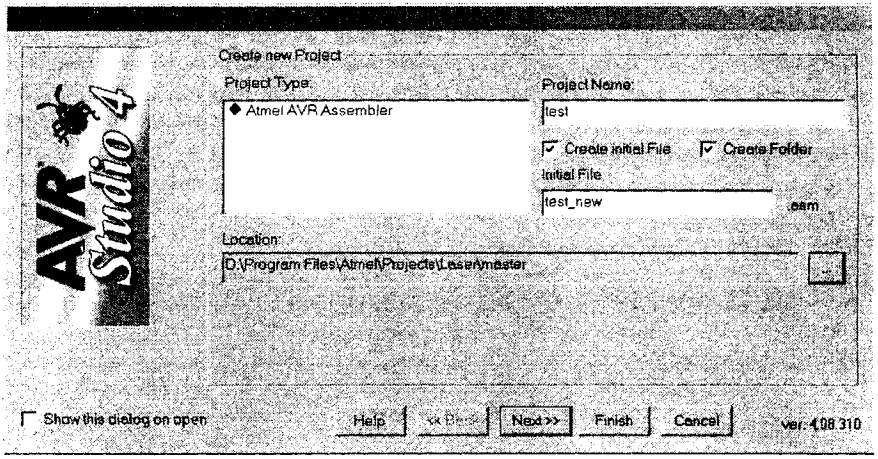

Рис. 11. Окно "Create new Project"

определить опции проекта: Debug Platform – AVR Simulator, тип микроконтроллера Device – AT90S8535. Если проект уже существует, можно открыть его для работы тем же меню **Project** (или меню File). Окно с открытым для работы проектом появится в рабочем поле.

2. Если в открытом проекте уже есть файлы программ на ассемблере (с расширением .asm), они будут отображены в окне "Workspace" и их можно открыть в рабочем поле двойным щелчком левой кнопки мыши. Создать новые файлы (Create new file) или добавить к проекту созданные ранее файлы (Add existing file) можно с помощью контекстного меню (или меню **Project**), доступ к контекстному меню открывается щелчком правой кнопки мыши в окне Workspace (рис. 10). Каждый файл на рабочем поле отображается отдельным окном. Работа с окнами, подготовка и редактирование текстов выполняются стандартными средствами Window's–приложений.

Компилятор ассемблера может работать только с одним файлом, поэтому необходимо указать для компилятора исходный файл в контекстном меню (Set as Entry File). После этого можно выполнить запуск компилятора из меню **Project** – пункт **Build (F7)** или соответствующей кнопкой панели инструментов. По завершении ассемблирования компилятор сформирует в окне "Output" (рис. 10) сообщение о результатах. Если синтаксических ошибок нет, выходной файл сформирован.

Обнаруженные ошибки будут указаны компилятором (окно "Output"), в сообщении указываются тип ошибки и номер строки в исходной программе. Отметить строку с ошибкой в исходном тексте можно двойным щелчком левой кнопки мыши на сообщении об ошибке.

Предварительная проверка подготовленной программы после устранения синтаксических ошибок производится с помощью симулятора (Debug Platform – AVR Simulator). Запуск симулятора производится из меню **Project** – пункт **Build and Run (Ctrl+F7)** или соответствующей кнопкой панели инструментов. Включение различных функций симулятора, режимы их моделирования определяются различными меню пакета AVR Studio или кнопками панели инструментов (более подробную информацию об этом можно найти в меню Help).

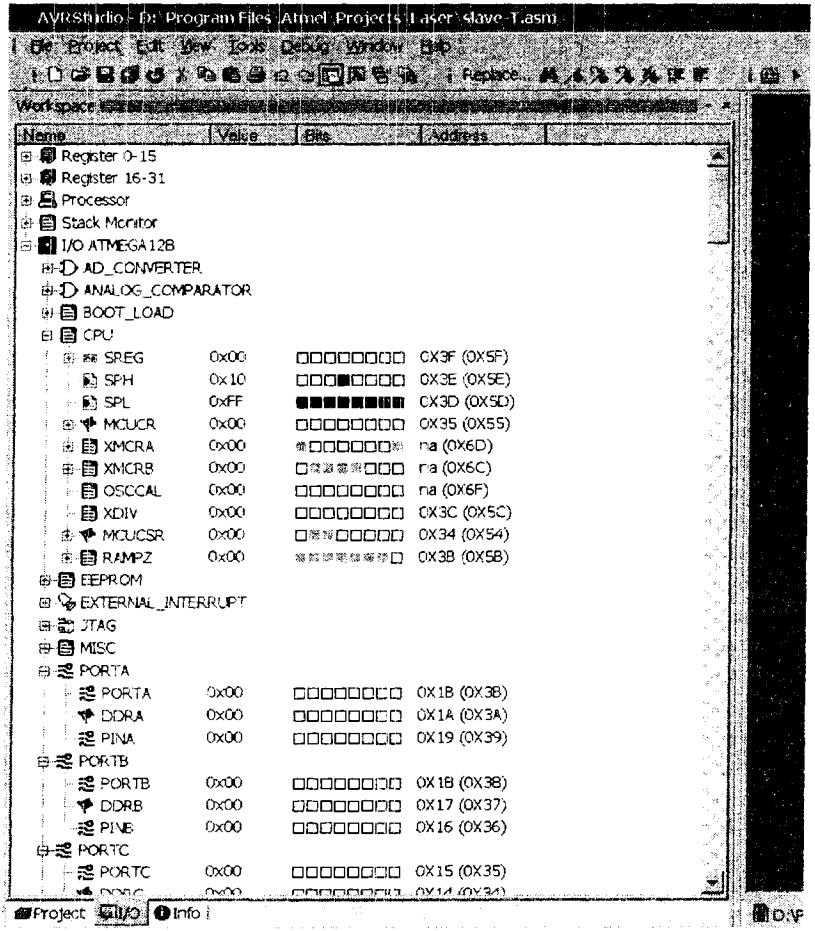

Рис. 12. Окно I/O

Отображение результатов работы программы в симуляторе производится в окне "Workspace" переключением в режим I/O (рис. 12). В этом окне можно отображать содержимое всех регистров и ячеек памяти микроконтроллера. При использовании пошагового режима выполнения программ на любом этапе можно производить изменение данных вводом с клавиатуры PC новых значений в двоичном, шестнадцатеричном или десятичном формате. Аналогичные операции можно выполнять с помощью окон "Watch", формируя в этих окнах произвольный и более удобный для работы с программой набор отображаемых элементов микроконтроллера.

После проверки корректности программы симулятором можно с помощью программатора произвести запись подготовленного Hex–файла в память микроконтроллера. Необходимо учитывать, что симулятор не позволяет моделировать все необходимые функции. Окончательное тестирование программы может проводиться только на лабораторном стенде с микроконтроллером [2].

Для записи подготовленного Hex–файла в микроконтроллер необходимо подключить лабораторный стенд к COM-порту компьютера (в стенде используется разъем DB15F с надписью AVR studio Prog) и включить питание. После включения стенда программатор AVR Studio может работать, его запуск производится из меню **Tools** пунктом **AVR Prog**. Управление программатором производится в открывшемся после запуска окне **AVR-prog** на рабочем поле.

В окне программатора необходимо выбрать записываемый в микроконтроллер Hex–файл (кнопка Browse, файл после компиляции находится в папке проекта, с которым Вы работаете). В строке **Device** выбрать тип микроконтроллера (AT90S8515) и запустить программатор в режиме программирования флэш-памяти программ кнопкой **Program** в области **Flash** окна программатора. Для управления программированием EEPROM в окне программатора предусмотрена специальная область. Устройство начинает работать сразу после завершения программирования.

5.4. Особенности применения микроконтроллеров AVR

Как устройство управления микроконтроллер должен обеспечивать прием сигналов, характеризующих состояние объекта управления, обрабатывать эти сигналы в соответствии с реализуемым алгоритмом управления, формировать и передавать управляющие воздействия, отображать в устройствах индикации информацию о текущем состоянии объекта и режимах его работы. Кроме этого, практически всегда необходимо управление параметрами и режимами работы, которое производится с пульта управления, содержащего кнопки пуска, останова, сброса, изменения заданных значений рабочих параметров и т.п. Все необходимые элементы должны проектироваться параллельно с разработкой общего алгоритма работы микро-

контроллерной системы управления. Пример такого решения задачи рассмотрен в разделе 5,2.

Рекомендации по применению программно-аппаратных средств микроконтроллера приведены ранее. Подключение к микроконтроллеру дополнительных внешних элементов и устройств должно производиться с согласованием электрических и временных параметров сигналов, алгоритмов обмена, нагрузочной способности и т.п. Если в сигнальных цепях возможна передача напряжений, превышающих напряжение питания микроконтроллера, применение устройств гальванической развязки обязательно. Приведенные далее параметры сигналов соответствуют стандартному напряжению питания микроконтроллера +5 В.

При выборе портов ввода-вывода микроконтроллера для приема или выдачи логических сигналов необходимо учитывать альтернативные функции этих портов. Если определенные аппаратные средства применяются при решении задачи, соответствующие их функциям входы и выходы портов жестко закреплены. Поэтому перед распределением функций портов необходимо определить функции и режимы аппаратных средств микроконтроллера и используемые ими линии ввода-вывода.

Уровни сигналов микроконтроллера соответствуют стандартным логическим сигналам для напряжения питания +5 В. Нагрузочная способность портов в режиме вывода – 20 мА, однако не рекомендуется нагружать этим максимальным током более трех выходов одновременно. Входное сопротивление в режиме ввода высокое, даже при использовании внутренних резисторов портов для задания пассивного высокого уровня сигнала дополнительный ток входных цепей не превышает нескольких мкА. Если требуются логические сигналы с другими параметрами на входах или выходах микроконтроллера, должны использоваться схемы сопряжения (транзисторные ключи, интегральные схемы преобразователей уровня, усилители мощности и т.п.) с необходимыми характеристиками преобразования.

Логические сигналы для управления параметрами и режимами работы удобно формировать кнопками с замыкающими контактами. Если на входе управления задать пассивный высокий уровень и второй контакт кнопки подключить к общему проводу, то замыкание контактов кнопки легко идентифицируется низким уровнем сигнала на входе порта и не требуется подключения никаких дополнительных элементов. При обработке сигналов кнопок или других устройств с механическими контактами (реле, концевые выключатели, клавиатура и т.п.) необходимо учитывать "дребезг" контактов. В микроконтроллере устранение "дребезга" можно выполнить без применения дополнительных внешних устройств, установив период повторного опроса этих линий ввода около 100 мс с помощью одного из таймеров.

Индикацию простых логических сигналов можно выполнить подключением светодиодов через токоограничивающие резисторы непосредственно

к выходам портов микроконтроллера. Отображение более сложной информации (цифровые, символьные или матричные индикаторы) целесообразнее обеспечивать с помощью дополнительных внешних устройств и выполнять вывод необходимых данных в последовательном формате, например, через интерфейс SPI.

Преобразование аналоговых сигналов, поступающих на линии ввода порта A, в цифровой формат производится 8-канальным десятиразрядным АЦП. Диапазон преобразуемых напряжений зависит от опорного напряжения U_{REF}, подаваемого на вход AREF микроконтроллера. Цифровой код N и амплитуда входного сигнала $U_{вх}$ определяются следующим образом:

$$U_{BX} = U_{REF} \frac{N}{2^{10}} \quad .$$

Рекомендуется значение U_{REF} около 4 В, тогда при изменении амплитуды сигнала $U_{вх}$ от 0 до 4 В цифровой код N будет изменяться от $000 до $3FF, соответствие между амплитудой сигнала и цифровым кодом можно найти по приведенной выше формуле. Так как точность формирования U_{REF} существенно влияет на точность преобразования АЦП, в качестве источников опорного напряжения необходимо применение специализированных прецизионных интегральных микросхем. Например, микросхема REF198E (Analog Devices) при напряжении питания 5 В позволяет получить напряжение 4,096 В с точностью ±2 мВ и весьма низким температурным и временным дрейфом.

Нельзя подавать на входы микроконтроллера напряжение более 4,7 В, использование же слишком низких значений опорного напряжения приведет к снижению точности преобразования особенно при малых уровнях сигналов. Рекомендуемое значение тактовой частоты АЦП 50 кГц – 200 кГц, при более высоких тактовых частотах работа возможна, но снижается точность преобразования.

ЦАП для формирования аналоговых выходных сигналов в составе микроконтроллера не предусмотрен. При необходимости можно использовать только внешний ЦАП, передавая для него цифровые сигналы через интерфейсы микроконтроллера. Однако в большинстве применений необходимые аналоговые сигналы можно формировать с помощью таймеров микроконтроллера в режиме модуляторов ШИМ. Как известно, при использовании ШИМ усиление сигналов по мощности производится более простыми средствами, необходимы только дополнительные силовые сглаживающие фильтры для получения приемлемых уровней пульсаций аналоговых выходных сигналов.

ЗАКЛЮЧЕНИЕ

Высокая эффективность современных средств автоматизации в значительной степени определяется применяемыми алгоритмами управления. Наиболее универсальны для выполнения разнообразных процедур преобразования данных, лежащих в основе алгоритмов управления, микропроцессорные устройства. Следует учитывать, что особенности программной реализации этих алгоритмов в микропроцессорных системах накладывают определенные ограничения. Рациональное сочетание возможностей программной и аппаратной реализации позволяет существенно смягчить эти ограничения. Именно такое сочетание возможностей лежит в основе микроконтроллеров – наиболее эффективных, универсальных и популярных компонентов современных средств автоматизации.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Каган Б.М., Сташин В.В. Основы проектирования микропроцессорных устройств автоматики. – М.: Энергоатомиздат, 1987.
2. Иванов Ю.И., Югай В.Я. Микропроцессорные устройства систем управления: Методическое руководство к лабораторным работам. – Таганрог: Изд-во ТРТУ, 2004.

Регистр	Адрес	Бит 7	Бит 6	Бит 5	Бит 4	Бит 3	Бит 2	Бит 1	Бит 0
SREG	3F (5F)	I	T	H	S	V	N	Z	C
SPH	3E (5E)	-	-	-	-	-	-	SP9	SP8
SPL	3D (5D)	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0
GIMSK	3B (5B)	INT1	INT0	Не используются					
GIFR	3A (5A)	INTF1	INTF0	Не используются					
TIMSK	39 (59)	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	-	TOIE0
TIFR	38 (58)	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	-	TOV0
MCUCR	35 (55)	-	SE	SM1	SM0	ISC11	ISC10	ISC01	ISC00
MCUSR	34 (54)	Не используются						EXTRF	PORF
TCCR0	33 (53)	Не используются					CS02	CS01	CS00
TCNT0	32 (52)	Регистр таймера 0 (8 бит)							
TCCR1A	2F (4F)	COM1A1	COM1A0	COM1B1	COM1B0	-	-	PWM11	PWM10
TCCR1B	2E (4E)	ICNC1	ICES1	-	-	CTC1	CS12	CS11	CS10
TCNT1H	2D (4D)	Регистр таймера 1 (старший байт)							
TCNT1L	2C (4C)	Регистр таймера 1 (младший байт)							
OCR1AH	2B (4B)	Регистр канала сравнения А таймера 1 (старший байт)							
OCR1AL	2A (4A)	Регистр канала сравнения А таймера 1 (младший байт)							

Продолжение прил.

Регистр	Адрес	Бит 7	Бит 6	Бит 5	Бит 4	Бит 3	Бит 2	Бит 1	Бит 0
OCR1BH	29 (49)	Регистр канала сравнения B таймера 1 (старший байт)							
OCR1BL	28 (48)	Регистр канала сравнения B таймера 1 (младший байт)							
ICR1H	27 (47)	Регистр режима захват таймера 1 (старший байт)							
ICR1L	26 (46)	Регистр режима захват таймера 1 (младший байт)							
TCCR2	25 (45)	-	PWM2	COM21	COM20	CTC2	CS22	CS21	CS20
TCNT2	24 (44)	Регистр таймера 2 (8 бит)							
OCR2	23 (43)	Регистр режима захват таймера 2 (8 бит)							
ASSR	22 (42)	Не используются				AS2	TCN2UB	OCR2UB	TCR2UB
WDTCR	21 (41)	Не используются			WDTOE	WDE	WDP2	WDP1	WDP0
EEARH	1F (3F)	Не используются							EEAR8
EEARL	1E (3E)	EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0
EEDR	1D (3D)	Регистр данных ППЗУ (EEPROM)							
EECR	1C (3C)	Не используются				EERIE	EEMWE	EEWE	EERE
PORTA	1B (3B)	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0
DDRA	1A (3A)	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0
PINA	19 (39)	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0
PORTB	18 (38)	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
DDRB	17 (37)	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0

Регистр	Адрес	Бит 7	Бит 6	Бит 5	Бит 4	Бит 3	Бит 2	Бит 1	Бит 0
PINB	16 (36)	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
PORTC	15 (35)	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0
DDRC	14 (34)	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0
PINC	13 (33)	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0
PORTD	12 (32)	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
DDRD	11 (31)	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
PIND	10 (30)	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
SPDR	0F (2F)	Регистр данных SPI (8 бит)							
SPSR	0E (2E)	SPIF	WCOL	Не используются					
SPCR	0D (2D)	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0
UDR	0C (2C)	Регистр данных UART (8 бит)							
USR	0B (2B)	RXC	TXC	UDRE	FE	OR	Не используются		
UCR	0A (2A)	RXCIE	TXCIE	UDRIE	RXEN	TXEN	CHR9	RXB8	TXB8
UBRR	09 (29)	Регистр скорости UART (8 бит)							
ACSR	08 (28)	ACD	-	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0
ADMUX	07 (27)	Не используются					ADMUX2	ADMUX1	ADMUX0
ADCSR	06 (26)	ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0
ADCH	05 (25)	Не используются						ADC9	ADC8
ADCL	04 (24)	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0

Иванов Юрий Иванович
Югай Владислав Яковлевич

МИКРОПРОЦЕССОРНЫЕ УСТРОЙСТВА СИСТЕМ УПРАВЛЕНИЯ

Учебное пособие

Ответственный за выпуск Югай В.Я.
Редактор Кочергина Т.Ф..
Корректор Надточий З.И.

ЛП № 020565 от 23.06.1997 г. Подписано к печати 11.04.05
Офсетная печать Усл. п.л. – 8,3 Уч.-изд.л. – 8,1
Заказ № 154 Тираж 150 экз.

"С"

Издательство Таганрогского государственного
радиотехнического университета
ГСП 17А, Таганрог, 28, Некрасовский, 44
Типография Таганрогского государственного
радиотехнического университета
ГСП 17А, Таганрог, 28, Энгельса, 1