Министерство образования и науки Самарской области
Государственное бюджетное профессиональное образовательное учреждение
Самарской области
«САМАРСКИЙ ЭНЕРГЕТИЧЕСКИЙ КОЛЛЕДЖ»
(ГБПОУ «СЭК»)

ОП.08 МИКРОПРОЦЕССОРНЫЕ СИСТЕМЫ

Конспект лекций для студентов специальности 11.02.16
Монтаж, техническое обслуживание и ремонт
электронных приборов и устройств

Самара

Конспект лекций по дисциплине ОП.08 Микропроцессорные системы для студентов специальности 11.02.16. – Самара: ГБПОУ «СЭК». – 60 с.

Издание содержит конспект лекций по дисциплине ОП.08 Микропроцессорные системы.

Замечания, предложения и пожелания направлять в ГБПОУ «СЭК» по адресу: 443001, г. Самара, ул. Самарская 205-А или по электронной почте info@sam-ek.ru

ГБПОУ «СЭК»

Раздел 1 Микропроцессорные системы. Основные понятия

Тема 1.1 Микропроцессорные системы (МПС). Виды и характеристики

1.Основные виды МПС и их особенности. Обобщенная структура МПС. Основные характеристики и параметры МПС. Краткая характеристика возможностей и применений микропроцессорных систем

Микропроцессорная система – цифровое устройство или цифровая система (система обработки данных, контроля и управления), построенная на базе одного или нескольких микропроцессоров. *Программно-аппаратный принцип* построения МПС – один из основных принципов их организации. Этот принцип заключается в том, что *реализация целевого назначения МПС достигается не только аппаратными средствами, но и с помощью программного обеспечения*.

Магистрально-модульный принцип организации микропроцессорной системы

Второй принцип, лежащий в основе организации микропроцессорных систем, – *магистрально-модульный принцип*. В соответствии с этим принципом основные структурные компоненты МПС выполняются в виде отдельных функционально законченных модулей, которые подключаются к единой внутрисистемной магистрали (шине).

Микропроцессорный модуль включает *микропроцессор*, *генератор тактовых импульсов* (ГТИ), а также, возможно, и другие устройства, например, *таймер*, *контроллер прерываний* и т.п.

В *подсистеме памяти* выделяют *модули ОЗУ*, предназначенных для хранения переменных и загружаемого извне программ, и *модули ПЗУ*, которые используются для хранения программ и констант.

В составе *подсистемы ввода/вывода* в простейшем случае выделяются адресуемые процессором буферные схемы и регистры – *порты ввода/вывода*. Они предназначены для связи с простыми внешними устройствами, такими как светодиодные индикаторы, переключатели и т.п. Более сложные модули подсистемы ввода/вывода, предназначенные для управления внешним интерфейсным оборудованием и реализации специальных функций ввода/вывода, строятся на основе портов ввод/вывод и называются *адаптерами* или *контроллерами периферийных устройств*. Наиболее сложными из модулей подсистемы ввода/вывода являются *процессоры* (*сопроцессоры*) ввода/вывода, которые работают по собственным программам, хранящимся в памяти, и по сути дела представляют собой отдельные микропроцессорные системы.

Основные классы микропроцессорных средств

Микропроцессор (МП) – функционально законченное программно-управляемое устройство, предназначенное для обработки данных и управления процессом этой обработки, выполненное в виде одной или нескольких микро-

схем. С точки зрения организации микропроцессорной системы – это ЦП, реализованный в виде одной или нескольких микросхем.

Микропроцессорная схема (МП схема) – интегральная микросхема (ИМС), выполняющая функцию МП или его части. По существу – это интегральная схема с процессорной организацией, разработанная для построения микропроцессорных систем. Микропроцессорные схемы относятся к особому классу микросхем, одной из особенностей которых является возможность программного управления ими с помощью определенного набора команд.

МикроЭВМ – ЭВМ, содержащая одну или несколько МП схем, микросхемы памяти (ПЗУ и ОЗУ), интерфейсов периферийных устройств, а также некоторые другие схемы (рис. 1).

Рис. 1 – Структура микроЭВМ

Микропроцессорный комплект (*семейство, набор*) (МПК) – совокупность МП и других микросхем, совместимых по конструктивно-технологическому исполнению и предназначенных для совместного применения при построении МП, микроЭВМ и микропроцессорных систем.

Однокристальная микроЭВМ – микроЭВМ, выполненная в виде одной интегральные схемы. Однокристальные микроЭВМ широко используются для управления различной аппаратурой и оборудованием, например в бытовых приборах.

Микроконтроллер – однокристальная микроЭВМ с небольшими вычислительными ресурсами и упрощенной системой команд, ориентированная на выполнение процедур логического управления различным оборудованием (а не на производство вычислений). Особенностью микроконтроллеров является расширенная реализация периферийных средств на кристалле.

Одноплатная микроЭВМ – микроЭВМ, выполненная в виде одной печатной платы и предназначенная для встраивания в различную аппаратуру. В отличие от однокристальной ЭВМ, размещая на одной плате несколько микросхем, можно получить микроЭВМ с достаточно большими вычислительными ресурсами.

Микропроцессорные средства – МПК, однокристальные (включая микроконтроллеры) и одноплатные микроЭВМ.

Тема 1.2.Организация функционирования МПС

1. Обобщенная структурная схема МПС. Алгоритм работы. Механизмы прерываний. Прямой доступ к памяти.

Прямой доступ к памяти (ПДП) является одним из способов обмена данными с ПУ. В этом режиме обмен данными между ПУ и памятью микропроцессорной системы происходит безучастия процессора. Обменом в режиме ПДП управляет не программа, выполняемая процессором, а внешнее по отношению к процессору специальное устройство, называемое *контроллером ПДП* (КПДП). ПДП используется для быстрого ввода/вывода блоков данных и разгрузки процессора от управления операциями ввода/вывода. Обмен блоками данных с помощью программно-управляемого обмена осуществляется относительно медленно, так как на обмен каждым байтом затрачивается несколько команд процессора. ПДП освобождает процессор от управления операциями ввода/вывода, позволяя тем самым

осуществлять параллельно во времени выполнение процессором программы и обмен данными между ПУ и памятью;

производить этот обмен со скоростью, ограниченной только пропускной способностью памяти или ПУ.

Таким образом, ПДП, разгружая процессор от обслуживания операций ввода/вывода, способствует возрастанию общей производительности микропроцессорной системы.

Для реализации режима ПДП необходимо обеспечить непосредственную связь контроллера ПДП и памяти микропроцессорной системы, т.е. специальный информационный канал, по которому осуществляется обмен в режиме ПДП, – *канал ПДП*. Для этой цели можно использовать специально выделенную магистраль, связывающую контроллер ПДП с памятью. Однако это приведет к значительному усложнению микропроцессорной системы в целом, особенно при подключении нескольких ПУ. Поэтому с целью сокращения количества линий в шинах микропроцессорной системы контроллер ПДП подключается к памяти посредством шин системной магистрали. При этом возникает проблема совместного использования шин системной магистрали процессором и контроллером ПДП. Можно выделить два основных способа ее решения:

реализация обмена в режиме ПДП с захватом цикла;

реализация обмена в режиме ПДП с блокировкой процессора. Режим ПДП с захватом цикла

Существует две разновидности прямого доступа к памяти с захватом цикла. Наиболее простой способ организации ПДП состоит в том, что для обмена используются те циклы процессора, в которых он не обменивается данными с памятью. В такие циклы контроллер ПДП может обмениваться данными с памятью, не мешая работе процессора. Однако возникает необходимость вы-

деления таких циклов, чтобы не произошло временного перекрытия обмена ПДП с операциями обмена, инициируемыми процессором. В некоторых процессорах формируется специальный управляющий сигнал, указывающий циклы, в которых процессор не использует память. Если процессор не формирует такого сигнала, то для выделения свободных циклов необходимо применение в контроллере ПДП специальной схемы, что приводит к его усложнению. Применение этого способа организации ПДП не снижает производительности системы, но при этом обмен в режиме ПДП возможен только в случайные моменты времени одиночными словами.

Наиболее распространенным является *ПДП с захватом цикла и принудительным отключением процессора от шин системной магистрали*. Для реализации такого режима ПДП системная магистраль дополняется двумя управляющими сигналами – *требование прямого доступа к памяти* HOLD и *предоставление прямого доступа к памяти* HLDA.

Управляющий сигнал HOLD формируется контроллером ПДП. Процессор, получив этот сигнал, приостанавливает выполнение текущей команды, не дожидаясь ее завершения, отключается от шин системной магистрали и выдает контроллеру ПДП управляющий сигнал HLDA. С этого момента все шины системной магистрали управляются контроллером ПДП. Контроллер ПДП, используя шины системной магистрали, осуществляет обмен одним словом данных с памятью и затем, сняв сигнал HOLD, возвращает управление системной магистралью процессору. Как только контроллер ПДП будет готов к обмену следующим словом данных, он вновь захватывает цикл процессора и т.д. В промежутках между захватами циклов контроллером ПДП процессор продолжает выполнять команды программы. Тем самым выполнение программы замедляется, но значительно в меньшей степени, чем при обмене в режиме прерывания.

Передача блока данных с использованием ПДП предполагает выполнение определенной последовательности действий (рис. 2):

1. начальная установка (предварительная подготовка) контроллера ПДП;
2. запуск контроллера ПДП;
3. многократное занятие цикла процессора;
4. завершение обмена.

Рис. 2 – Передача блока данных с использованием ПДП

Программа используется только для начальной установки и пуска обмена через канал ПДП. После этого процессор может выполнять основную программу, которая не связана с обменом. Во время выполнения этой программы каждый раз при поступлении запроса на ПДП контроллер ПДП будет занимать цикл процессора и осуществлять передачу. После окончания обмена для передачи управления программе завершения обмена в режиме ПДП используется прерывание. Затем основная программа может быть продолжена.

Начальная подготовка к обмену в режиме ПДП состоит в выделении ПУ области памяти, используемой при обмене, и указании ее размера, т.е. количества записываемых в память или читаемых из памяти слов информации. Для этого контроллер ПДП имеет в своем составе регистр адреса и счетчик слов. Перед началом обмена с ПУ в режиме ПДП процессор должен выполнить

программу загрузки, которая обеспечивает запись в указанные регистры контроллера ПДП начального адреса выделенной ПУ области памяти и ее размера в словах заданной разрядности.

Запуск контроллера ПДП осуществляется командой вывода, по которой ПУ подключается к контроллеру ПДП. После команды пуска контроллера ПДП должна быть команда разрешения прерывания. В дальнейшем ввод блока данных через канал ПДП осуществляется без участия команд программы.

Когда ПУ подготовит слово данных, оно посылается в регистр данных контроллера. При этом каждое слово сопровождается управляющим сигналом *ввод данных* из ПУ, который обеспечивает запись слова данных в регистр данных контроллера и формирование сигнала *требование прямого доступа к памяти* HOLD. В ответ процессор формирует сигнал *предоставление прямого доступа к памяти* HLDA, после чего следующий машинный цикл зани-

мается под обмен. При этом осуществляется одна пересылка слова данных в ячейку памяти по адресу, находящемуся в регистре адреса контроллера. По сигналу HLDA контроллер выставляет на шины адреса и данных системной магистрали содержимое своих регистров адреса и данных соответственно.

Формируя управляющий сигнал MEMRW, контроллер ПДП обеспечивает запись слова данных из своего регистра данных в память.

После передачи каждого слова данных из содержимого счетчика слов контроллера вычитается 1, и когда оно становится равным 0, устанавливается запрос на прерывание, который поступает на соответствующий вход процессора. Процессор прерывает выполнение программы и передает управление подпрограмме обработки прерывания для завершения обмена.

Завершение обмена осуществляется путем отключения ПУ от контроллера ПДП командой вывода. По окончании обработки прерывания управление возвращается основной программе.

Если нет необходимости продолжать выполнение некоторой программы параллельно с передачей в режиме ПДП, то прерывание не используется. В течение обмена через канал ПДП процессор находится в цикле ожидания окончания передачи, опрашивая соответствующий разряд готовности регистра состояния контроллера ПДП по команде ввода. Как только процессор обнаружит готовность, он переходит к процедуре завершения обмена (шаг 4 рассмотренной выше последовательности), после чего выполнение программы продолжается.

Выше были рассмотрены только процесс подготовки контроллера ПДП и непосредственно передача данных в режиме ПДП. На практике любой сеанс обмена данными с ПУ в режиме ПДП всегда включает также и этап подготовки ПУ к обмену. На этом этапе процессор в режиме программно-управляемого обмена опрашивает состояние ПУ, проверяя его готовность к обмену, и посылает в ПУ команды, обеспечивающие его подготовку к обмену данными по каналу ПДП. Такая подготовка может сводиться, например, к перемещению головок на требуемую дорожку в НМД. Затем выполняется загрузка регистров контроллера ПДП, после чего начинается обмен данными в режиме ПДП.

Следует отметить, что использование в микропроцессорной системе обмена в режиме ПДП с захватом цикла требует от программиста очень ясного понимания процессов, происходящих в системе при выполнении программы, и четкой синхронизации процесса выполнения программы и ввода/выводапо каналу ПДП.

Режим ПДП с блокировкой процессора

Прямой доступ к памяти с блокировкой процессора отличается от режима ПДП с захватом цикла тем, что управление системной магистралью передается контроллеру ПДП не на время передачи одного слова, а на время обмена

блоком данных. Такой режим ПДП необходим в тех случаях, когда время между двумя сигналами *требование прямого доступа к памяти* HOLD сопоставимо с циклом процессора. В этом случае процессор не успевает выполнить хотя бы одну команду между очередными операциями обмена в режиме ПДП.

В микропроцессорной системе можно использовать несколько ПУ, работающих в режиме ПДП. Предоставление таким ПУ шин системной магистрали для обмена данными производится на приоритетной основе. В этом случае приоритеты ПУ реализуются так же, как и при обмене данными в режиме прерывания. Как правило, для каждого ПУ используется своя пара управляющих сигналов *требование прямого доступа к памяти* HOLD и *предоставление прямого доступа к памяти* HLDA и отдельный канал в контроллере ПДП

Тема 1.3 Микропроцессоры (МП)

1. Классификация и характеристики МП. Понятие об архитектуре микропроцессора. Основные элементы архитектуры. Поколения МП.

Микропроцессор(МП) - это программно управляемое устройство, которое предназначено для обработки цифровой информации и управления процессом этой обработки и выполнено в виде одной или нескольких больших интегральных схем (БИС).

Понятие **большая интегральная схема**в настоящее время четко не определено. Ранее считалось, что к этому классу следует относить микросхемы, содержащие более 1000 элементов на кристалле. И действительно, в эти параметры укладывались первые микропроцессоры. Например, 4-разрядная процессорная секция микропроцессорного комплекта К584, выпускавшегося в конце 1970-х годов, содержала около 1500 элементов. Сейчас, когда микропроцессоры содержат десятки миллионов транзисторов и их количество непрерывно увеличивается, под БИС будем понимать функционально сложную интегральную схему.

Микропроцессорная система(МПС) представляет собой функционально законченное изделие, состоящее из одного или нескольких устройств, основу которой составляет микропроцессор.

Микропроцессор характеризуется большим количеством параметров и свойств, так как он является, с одной стороны, функционально сложным вычислительным устройством, а с другой - электронным прибором, изделием электронной промышленности. Как средство вычислительной техники он характеризуется прежде всего своей **архитектурой**, то есть совокупностью программно-аппаратных свойств, предоставляемых пользователю. Сюда относятся система команд, типы и форматы обрабатываемых данных, режимы адресации, количество и распределение регистров, принципы взаимодействия с оперативной памятью и внешними устройствами (характеристики сис-

темы прерываний, прямой доступ к памяти и т. д.). По своей архитектуре микропроцессоры разделяются на несколько типов (рис. 3).

Универсальные микропроцессоры предназначены для решения задач цифровой обработки различного типа информации от инженерных расчетов до работы с базами данных, не связанных жесткими ограничениями навремя выполнения задания. Этот класс микропроцессоров наиболее широко известен. К нему относятся такие известные микропроцессоры, как МП ряда Pentium фирмы Intel и МП семейства Athlon фирмы AMD.

Рис. 3. Классификация микропроцессоров

Характеристики универсальных микропроцессоров:

- разрядность: определяется максимальной разрядностью целочисленных данных, обрабатываемых за 1 такт, то есть фактически разрядностью арифметико-логического устройства (АЛУ);
- виды и форматы обрабатываемых данных;
- система команд, режимы адресации операндов;
- емкость прямоадресуемой оперативной памяти: определяется разрядностью шины адреса;
- частота внешней синхронизации. Для частоты синхронизации обычно указывается ее максимально возможное значение, при котором гарантируется работоспособность схемы. Для функционально сложных схем, к которым относятся и микропроцессоры, иногда указывают также минимально возможную частоту синхронизации. Уменьшение частоты ниже этого предела может привести к отказу схемы. В то же время в тех применениях МП, где не требуется высокое быстродействие, снижение частоты синхронизации - одно из направлений энергосбережения. В ряде современных микропроцессоров при уменьшении частоты он переходит в <спящий режим>, при котором сохраняет свое состояние. Частота синхронизации в рамках одной архитектуры позволяет сравнить производительность микропроцессоров. Но разные архитектурные решения влияют на производительность гораздо больше, чем частота;
- производительность: определяется с помощью специальных тестов, при этом совокупность тестов подбирается таким образом, чтобы они по возможности покрывали различные характеристики микроархитектуры процессоров, влияющие на производительность.

Универсальные микропроцессоры принято разделять на **CISC**- и**RISC-микропроцессоры.CISC-микропроцессоры**(Completed Instruction Set Computing - вычисления с полной системой команд) имеют в своем составе весь классический набор команд с широко развитыми режимами адресации операндов. Именно к этому классу относятся, например, микро процессоры типа Pentium. В то же время**RISC-микропроцессоры**(reduced instruction set computing - вычисления с сокращенной системой команд) используют, как следует из определения, уменьшенное количество команд и режимов адресации. Здесь прежде всего следует выделить такие микропроцессоры, как Alpha 21x64, Power PC. Количество команд в системе команд - наиболее очевидное, но на сегодняшний день не самое главное различие в этих направлениях развития универсальных микропроцессоров. Другие различия мы будем рассматривать по мере изучения особенностей их архитектуры.

Однокристальные микроконтроллеры(ОМК или просто МК) предназначены для использования в системах промышленной и бытовой автоматики. Они представляют собой большие интегральные схемы, которые включают в себя все устройства, необходимые для реализации цифровой системы управления минимальной конфигурации: процессор (как правило, целочисленный), ЗУ команд, ЗУ данных, генератор тактовых сигналов, программируемые устройства для связи с внешней средой (контроллер прерывания, таймеры-счетчики, разнообразные порты ввода/вывода), иногда аналого-цифровые и цифро-аналоговые преобразова- тели и т. д. В некоторых источниках этот класс микропроцессоров назы- вается однокристальными микро-ЭВМ (ОМЭВМ).

В настоящее время две трети всех производимых микропроцессорных БИС в мире составляют МП этого класса, причем почти две трети из них имеет разрядность, не превышающую 16 бит. К классу однокристальных микроконтроллеров прежде всего относятся микропроцессоры серии MCS-51 фирмы Intel и аналогичные микропроцессоры других производителей, архитектура которых де-факто стала стандартом.

Отличительные особенности архитектуры однокристальных микроконтроллеров:

- физическое и логическое разделение памяти команд и памяти данных (гарвардская архитектура), в то время как в классической неймановской архитектуре программы и данные находятся в общем запоминающем устройстве и имеют одинаковый механизм доступа;
- упрощенная и ориентированная на задачи управления система команд: в МК, как правило, отсутствуют средства обработки данных с плавающей точкой, но в то же время в систему команд входят команды, ориентированные на эффективную работу с датчиками и исполнительными устройствами, например, команды обработки битовой информации;
- простейшие режимы адресации операндов.

Основные характеристики микроконтроллеров(в качестве примера численные значения представлены для МК-51):

1. Разрядность (8 бит).
2. Емкость внутренней памяти команд и памяти данных, возможности и пределы их расширения:

- внутренняя память команд - 4 Кбайт (в среднем команда имеет длину 2 байта, таким образом, во внутренней памяти может быть размещена программа длиной около 2000 команд); возможность наращивания за счет подключения внешней памяти до 64 Кбайт;
- память данных на кристалле 128 байт (можно подключить внешнюю память общей емкостью до 64 Кбайт).

 Тактовая частота:

- внешняя частота 12 МГц;
- частота машинного цикла 1 МГц.

 Возможности взаимодействия с внешними устройствами: количество и назначение портов ввода-вывода, характеристики системы прерывания, программная поддержка взаимодействия с внешними устройствами.

Наличие и характеристики встроенных аналого-цифровых преобразователей (АЦП) и цифро-аналоговых преобразователей (ЦАП) для упрощения согласования с датчиками и исполнительными устройствами системы управления.

Секционированные микропроцессоры(другие названия: микропрограммируемые и разрядно-модульные) - это микропроцессоры, предназначенные для построения специализированных процессоров. Они представляют собой микропроцессорные секции относительно небольшой (от 2 до 16) разрядности с пользовательским доступом к микропрограммному уровню управления и средствами для объединения нескольких секций.

Такая организация позволяет спроектировать процессор необходимой разрядности и со специализированной системой команд. Из-за своей малой разрядности микропроцессорные секции могут быть построены с использованием быстродействующих технологий. Совокупность всех этих факторов обеспечивает возможность создания процессора, наилучшим образом ориентированного на заданный класс алгоритмов как по системе команд и режимам адресации, так и по форматам данных.

Одним из первых комплектов секционированных микропроцессоров были МП БИС семейства Intel 3000. В нашей стране они выпускались в составе серии К589 и 585. Процессорные элементы этой серии представляли собой двухразрядный микропроцессор. Наиболее распространенным комплектом секционированных микропроцессоров является Am2900, основу которого составляют 4-разрядные секции. В нашей стране аналог этого комплекта выпускался в составе серии К1804. В состав комплекта входили следующие БИС:

- разрядное секционное АЛУ;
- блок ускоренного переноса;
- разрядное секционное АЛУ с аппаратной поддержкой умножения;
- типа схем микропрограммного управления;
- контроллер состояния и сдвига;
- контроллер приоритетных прерываний.

Основным недостатком микропроцессорных систем на базе секционированных микропроцессорных БИС явилась сложность проектирования, отладки и программирования систем на их основе. Использование специализированной системы команд приводило к несовместимости разрабатываемого ПО для различных микропроцессоров. Возможность создания оптимального по многим параметрам специализированного процессора требовала труда квалифицированных разработчиков на протяжении длительного времени. Однако бурное развитие электронных технологий привело к тому, что за время проектирования специализированного процессора разрабатывался универсальный микропроцессор, возможности которого перекрывали гипотетический выигрыш от проектирования специализированного устройства. Это привело к тому, что в настоящее время данный класс микропроцессорных БИС практически не используется.

Процессоры цифровой обработки сигналов, или **цифровые сигнальные процессоры**, представляют собой бурно развивающийся класс микропроцессоров, предназначенных для решения задач цифровой обработки сигналов - обработки звуковых сигналов, изображений, распознавания образов и т. д. Они включают в себя многие черты однокристальных микро контроллеров: гарвардскую архитектуру, встроенную память команд и данных, развитые возможности работы с внешними устройствами. В то же время в них присутствуют черты и универсальных МП, особенно с RISC-архитектурой: конвейерная организация работы, программные и аппаратные средства для выполнения операций с плавающей запятой, аппаратная поддержка сложных специализированных вычислений, особенно умножения.

Как электронное изделие микропроцессор характеризуется рядом параметров, наиболее важными из которых являются следующие:

1. Требования к синхронизации: максимальная частота, стабильность.
2. Количество и номиналы источников питания, требования к их стабильности. В настоящее время существует тенденция к уменьшению напряжения питания, что сокращает тепловыделение схемы и ведет к повышению частоты ее работы. Если первые микропроцессоры работали при напряжении питания+-15В, то сейчас отдельные схемы используют источники менее 1 В.
3. **Мощность рассеяния** - это мощность потерь в выходном каскаде схемы, превращающаяся в тепло и нагревающая выходные транзисторы. Иначе говоря, она характеризует показатель тепловыделения БИС, что во многом определяет требования к конструктивному оформлению микропро-

цессорной системы. Эта характеристика особенно важна для встраиваемых МПС.

4. Уровни сигналов логического нуля и логической единицы, которые связаны с номиналами источников питания.
5. Тип корпуса - позволяет оценить пригодность схемы для работы в тех или иных условиях, а также возможность использования новой БИС в качестве замены существующей на плате.
6. Температура окружающей среды, при которой может работать схема. Здесь выделяют два диапазона:
   - коммерческий (0 ^{0}C … +70^{0}C);
   - расширенный (-40 ^{0}C … +85 ^{0}C).

 Помехоустойчивость - определяет способность схемы выполнять свои функции при наличии помех. Помехоустойчивость оценивается интенсивностью помех, при которых нарушение функций устройства еще не превышает допустимых пределов. Чем сильнее помеха, при которой устройство остается работоспособным, тем выше его помехоустойчивость.

 Нагрузочная способность, или коэффициент разветвления по выходу, определяется числом схем этой же серии, входы которых могут быть присоединены к выходу данной схемы без нарушения ее работоспособности. Чем выше нагрузочная способность, тем шире логические возможности схемы и тем меньше таких микросхем необходимо для построения сложного вычислительного устройства. Однако с увеличением этого коэффициента ухудшаются помехоустойчивость и быстродействие.

 Надежность - это способность схемы сохранять свой уровень качества функционирования при установленных условиях за установленный период времени. Обычно характеризуется интенсивностью отказов (час-1) или средним временем наработки на отказ (час). В настоящее время этот параметр для больших интегральных схем обычно не указывается изготовителем. О надежности МП БИС можно судить по косвенным показателям, например, по приводимой разработчиками средств вычислительной техники надежности изделия в целом.

 Характеристики технологического процесса. Основной показатель здесь - разрешающая способность процесса. В настоящее время она составляет 32 нм, то есть около 30 тыс. линий на 1 мм. Более совершенный технологический процесс позволяет создать микропроцессор, обладающий большими функциональными возможностями.

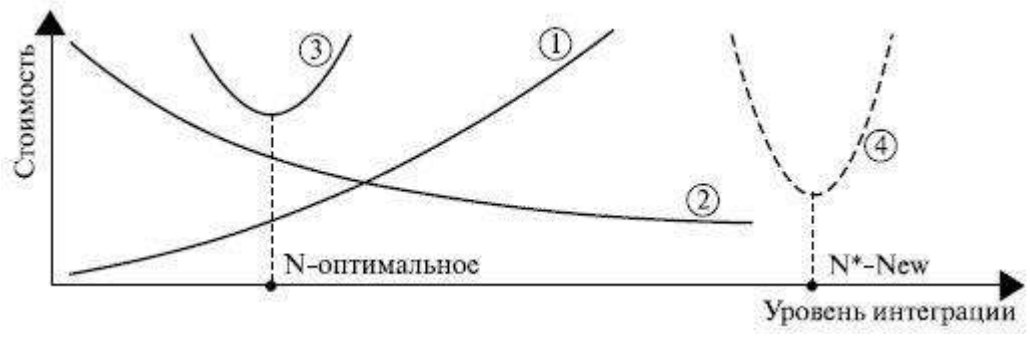


Рис. 4. Затраты на производство микропроцессорной системы

Затраты на изготовление устройств, использующих микропроцессорные БИС, представлены на рис. 1.2. Здесь:

1. затраты на изготовление БИС (чем больше степень интеграции элементов на кристалле, тем дороже обходится производство схемы);

2. затраты на сборку и наладку микропроцессорной системы (с увеличением функциональных возможностей МП потребуется меньше схем для создания МПС);

3. общая стоимость микропроцессорной системы, которая складывается из затрат (1) и (2). Она имеет некоторое оптимальное значение для данного уровня развития технологии;

4. переход на новую технологию (оптимальным будет уже другое количество элементов на кристалле, а общая стоимость изделия снижается).

В 1965 году Гордон Мур сформулировал гипотезу, известную в настоящее время как <закон Мура>, согласно которой каждые 1,5-2 года число транзисторов в расчете на одну интегральную схему будет удваиваться. Это обеспечивается непрерывным совершенствованием технологических процессов производства микросхем.

Наиболее развитая в технологическом отношении фирма Intel в жизненном цикле полупроводниковых технологий, создаваемых и применяемых в корпорации, выделяет шесть стадий.

Самая ранняя стадия проходит за пределами Intel - в университетских лабораториях и независимых исследовательских центрах, где ведутся поиски новых физических принципов и методов, которые могут стать основой научно-технологического задела на годы вперед. Корпорация финансирует эти исследования.

На второй стадии исследователи Intel выбирают наиболее перспективные направления развития новых технологий. При этом обычно рассматривается 2-3 варианта решения.

Главная задача третьей стадии - полная черновая проработка новой технологии и демонстрация ее осуществимости.

После этого начинается четвертая стадия, главная цель которой - обеспечить достижение заданных значений таких ключевых технических и экономических показателей, как выход годных изделий, надежность, стоимость и неко-

торые другие. Завершение этапа подтверждается выпуском первой промышленной партии новых изделий.

Пятая стадия - промышленное освоение новой технологии. Эта проблема не менее сложна, чем разработка самой технологии, поскольку необычайно трудно в точности воспроизвести в условиях реального производства то, что было получено в лаборатории. Обычно именно здесь возникают задержки со сроками выпуска новых изделий, с достижением запланированного объема поставок и себестоимости продукции.

Последняя, шестая стадия жизненного цикла технологии (перед отказом от ее применения) - зрелость. Зрелая технология, подвергаясь определенному совершенствованию с целью повышения производительности оборудования и снижения себестоимости продукции, обеспечивает основные объемы производства. По мере внедрения новых, более совершенных технологий <старые> производства ликвидируются.

Но не сразу: сначала они переводятся на выпуск микросхем с меньшим быстродействием или с меньшим числом транзисторов, например, периферийных БИС.

Структура и особенности архитектуры микропроцессора Pentium 4

Микропроцессор Pentium 4 является завершающей моделью 32-разрядных микропроцессоров фирмы INTel с архитектурой IA-32. Основные особенности этого процессора:

- новая микроархитектура процессора NetBurst (пакетно-сетевая);
- новая системная шина FSB.

Микроархитектура процессора определяет реализацию его внутренней структуры, принципы выполнения поступающих команд, способы размещения и обработки данных. Микроархитектура NetBurst отличается от своих предшественников по целому ряду позиций:

1. Применена гарвардская структура с разделением потоков команд и данных.
2. Используется гиперконвейерная технология (Hyper-PIPelINed Technology) выполнения команд, при которой число ступеней конвейера достигает 31 (в Pentium III - 11 ступеней). Таким образом, одновременно в процессе выполнения на разных стадиях реализации может находиться свыше 30 команд.
3. Используется динамическое выполнение команд (dynamic execution), построенное на трех базовых концепциях: предсказание переходов (branch prediction), динамический анализ потока данных (dynamic data flow analysis) и спекулятивное выполнение (OUT-oforder execution). Аналогичный механизм, названный Dynamic Execution, используется в МП Pentium III, однако в INTel Pentium 4 он улучшен.

4. Выполнение арифметических и логических операций происходит с удвоенной тактовой частотой процессора, что позволяет за один такт получить результаты для двух команд.
5. Кеш-память 2-го уровня емкостью 256 Кбайт размещается непосредственно на кристалле процессора, что позволяет сократить время выборки по сравнению с Pentuim III, где эта кэш-память располагается на отдельном кристалле в общем корпусе с процессором.
6. Значительно расширены возможности обработки чисел по принципу SIMD в новом блоке SSE-2.

Любой процессор архитектуры x86 обязательно оснащен процессорной шиной. Эта шина служит каналом связи между процессором и всеми остальными устройствами в компьютере: памятью, видеокартой, жестким диском и так далее. Так, классическая схема организации внешнего интерфейса процессора предполагает, что параллельная мультиплексированная процессорная шина, которую принято называть FSB (Front Side Bus), соединяет процессор (иногда два процессора или даже больше) и контроллер, обеспечивающий доступ к оперативной памяти и внешним устройствам. Этот контроллер обычно входит в состав северного моста набора системной логики (чипсета). Для ускорения обмена с памятью в Pentium 4 используется новая реализация системной шины, обеспечивающая обмен с эквивалентной частотой 400 МГц. Такая скорость достигается путем применения нового типа сверхбыстродействующей двухканальной памяти типа RDRAM и специальной микросхемы MCH (Memory ConTRoller Hub), реализующей 4 канала передачи данных. При тактовой частоте каждого канала 100 МГц обеспечивается общая частота обмена, эквивалентная 400 МГц. Шина включает 64-разрядную двунаправленную шину данных, дающую пропускную способность в 3,2 Гбайт/с, и 36-разрядную шину адреса (33 адресных линии A35-A3 и 8 линий выбора байтов BE7-BE0), что позволяет адресовать физическую память емкостью до 64 Гбайт. Именно учетверенная результирующая частота передачи данных является одним из главных предметов гордости разработчиков Pentium 4. Однако для многочисленных мелких запросов, где данные в большинстве своем умещаются в одну 64-байтную порцию (и, соответственно, не используются возможности многоканальной передачи), важнее именно частота тактирования. Последние модели Pentium 4 работают на частоте системной шины 150 МГц, что обеспечивает эквивалентную частоту FSB в 600 МГц и пропускную способность в 4,8 Гбайт/с.

Полученная по системной шине информация сохраняется в кэш-памяти 2-го уровня (L2) емкостью 256 Кбайт, общей для команд и данных, которая размещается непосредственно на кристалле МП. Ширина шины, по которой идет обмен данными между кэш-памятью L2 и процессором, составляет 256 бит (32 байта), а ее тактовая частота совпадает с тактовой частотой ядра процессора.

Гарвардская внутренняя структура реализуется на уровне кэш-памяти 1-го уровня (L1) путем разделения потоков команд и данных. Кэш-память данных 1-го уровня имеет емкость 8 Кбайт. Вместо кэш-памяти команд 1-го уровня в Pentium 4 используется кэш-память для декодированных команд (микрокоманд). Execution TRace Cache - это название и одновременно способ реализации L1-кэша инструкций в архитектуре NetBurst. Смысловое содержание этого термина можно перевести как "кэш трассировки выполняемых микрокоманд". В Execution TRace Cache хранятся микрокоманды (?ops), которые были получены в результате декодирования входного потока инструкций исполняемого кода и готовы для передачи на выполнение конвейеру. Емкость Execution TRace Cache составляет 12 Кбайт.

После заполнения кэш-памяти микрокоманд практически любая команда будет храниться в ней в декодированном виде. Поэтому при поступлении очередной команды блок трассировки выбирает из этой кэшпамяти необходимые микрокоманды, обеспечивающие ее выполнение.

Если в потоке команд оказывается команда условного перехода, то включается механизм предсказания ветвления, который формирует адрес следующей выбираемой команды до того, как будет определено условие выполнения перехода.

После формирования потоков микрокоманд производится выделение регистров, необходимых для выполнения декодированных команд.

Эта процедура реализуется блоком распределения регистров. Он выделяет для каждого указанного в команде логического регистра (регистра цлочисленных операндов EAX, EBX и т. д., регистра операндов с плавающей точкой ST0-ST7 или регистра блоков MMX, SSE) один из 128 физических регистров, входящих в состав блоков регистров замещения (БРЗ) целочисленного блока микропроцессора и блока обработки чисел с плавающей точкой. Эта процедура позволяет минимизировать конфликты в конвейерах и выполнять команды, использующие одни и те же логические регистры, одновременно или с изменением их последовательности.

Ступени распределения/переименования конвейера могут выпустить три микрокоманды за такт на следующую ступень конвейера.

Выбранные микрокоманды размещаются в очереди микрокоманд. В ней содержатся микрокоманды, реализующие выполнение до 120 поступивших и декодированных команд, которые затем направляются в исполнительные устройства. Отметим, что в процессорах Pentium III в очереди находятся микрокоманды для 40 поступивших команд. Значительное увеличение числа команд, стоящих в очереди, позволяет более эффективно организовать поток их исполнения, изменяя последовательность выполнения команд и выделяя команды, которые могут выполняться параллельно. Эти функции реализует блок распределения микрокоманд. Он выбирает микрокоманды из очереди не в порядке их поступления, а по мере готовности соответствующих операндов

и исполнительных устройств. В результате команды, поступившие позже, могут быть выполнены до ранее выбранных команд. При этом реализуется одновременное выполнение нескольких микрокоманд (команд) в параллельно работающих исполнительных устройствах. Таким образом, естественный порядок следования команд (микрокоманд) нарушается, чтобы обеспечить более полную загрузку параллельно включенных исполнительных устройств и повысить производительность процессора.

Адреса операндов, выбираемых из памяти, вычисляются блоком формирования адреса (БФА), который реализует интерфейс с кэш-памятью данных 1-го уровня. В соответствии с заданными в декодированных командах способами адресации формируются 48 адресов для загрузки операндов из памяти в регистр БРЗ и 24 адреса для записи из регистра в память (в Pentium III формируются 16 адресов для загрузки регистров и 12 адресов для записи в память). При этом БФА формирует адреса операндов для команд, которые еще не поступили на выполнение. При обращении к памяти БФА одновременно выдает адреса двух операндов: один для загрузки операнда в заданный регистр БРЗ, второй - для пересылки результата из БРЗ в память. Таким образом реализуется процедура предварительного чтения данных для последующей их обработки в исполнительных блоках (спекулятивная выборка).

Аналогичным образом организуется параллельная работа блоков SSE, FPU, MMX, которые используют отдельный набор регистров и блок формирования адресов операндов.

При выборке операнда из памяти производится обращение к кэшпамяти данных (L1), которая имеет отдельные порты для чтения и записи. За один такт производится выборка операндов для двух команд.

При формировании адресов обеспечивается обращение к заданному сегменту памяти. Каждый сегмент может делиться на страницы. Для сокращения времени трансляции используется **буфер ассоциативной трансляции страничного адреса** TLB, который хранит базовые адреса наиболее часто используемых страниц.

Микрокоманды поступают в исполнительное ядро из блока распределения по 4 портам в 8 исполнительных блоков. Эти порты выполняют функцию шлюзов к функциональным устройствам. Для обработки целочисленных данных и выполнения логических операций в Pentium 4 используются 4 однотипных арифметико-логических устройства (ALU). Обработка чисел с плавающей запятой проходит в FPU. Блоки MMX и SSE предназначены для выполнения команд этих типов.

За один такт через порты может пройти до шести микрокоманд. Это больше, чем может выполнить препроцессор (3 микрокоманды за такт), что дает некоторую свободу в случае резкого увеличения количества готовых к исполнению микрокоманд. Суперскалярная архитектура микропроцессора реали-

зуется путем организации исполнительного ядра МП в виде ряда параллельно работающих блоков.

Арифметико-логические блоки ALU производят обработку целочисленных операндов, которые поступают из заданных регистров БРЗ. В эти же регистры заносится и результат операции. При этом проверяются условия ветвления для команд условных переходов и выдаются сигналы перезагрузки конвейера команд в случае неправильно предсказанного ветвления. Рабочая тактовая частота модулей ALU в два раза выше тактовой частоты процессора. Это достигается за счет срабатывания как по переднему, так и по заднему фронтам задающего тактового сигнала. Таким образом, каждый ALU-модуль способен выполнить до двух целочисленных операций за один рабочий такт процессора.

Эффективность конвейера резко снижается из-за необходимости его перезагрузки при выполнении условных ветвлений, когда требуется произвести очистку всех предыдущих ступеней и выбрать команду из другой ветви программы. Чтобы сократить потери времени, связанные с перезагрузкой конвейера, используется улучшенный **блок предсказания ветвлений**. Его основной частью является ассоциативная память, называемая буфером адресов ветвлений BTB, в которой хранятся 4092 адреса ранее выполненных переходов. Отметим, что в BTB процессора Pentium III хранятся адреса только 512 переходов. Кроме того, BTB содержит биты, хранящие предысторию ветвления, которые указывают, выполнялся ли переход при предыдущих выборках данной команды. При поступлении очередной команды условного перехода указанный в ней адрес сравнивается с содержимым BTB. Если этот адрес не содержится в BTB, то есть ранее не производились переходы по данному адресу, то предсказывается отсутствие ветвления. В этом случае продолжается выборка и декодирование команд, следующих за командой перехода. При совпадении указанного в команде адреса перехода с каким-либо из адресов, хранящихся в BTB, производится анализ предыстории. В процессе анализа определяется чаще всего реализуемое направление ветвления, а также выявляются чередующиеся переходы. Если предсказывается выполнение ветвления, то выбирается и загружается в конвейер команда, размещенная по предсказанному адресу. Более совершенный механизм предсказания переходов в МП Pentium 4 обеспечивает уменьшение количества ошибочно предсказанных переходов в среднем на 33 % по сравнению с Pentium III. Таким образом, резко уменьшается число перезагрузок конвейера при неправильном предсказании ветвления.

В Pentium 4 также интегрирован набор из 144 новых SIMD-инструкций, обеспечивающих одновременное выполнение одной операции над несколькими операндами. Рассмотрим особенности использования этой схемы обработки данных подробнее.

Технология MMX - итог совместной работы создателей архитектуры микропроцессоров INTel и программистов. При ее разработке был исследован ши-

рокий круг программ аудиовизуальной обработки информации: обработка изображений, MPEG-видео, синтеза музыки, сжатия речи и ее распознавания, поддержка видеоконференций, компьютерные игровые программы и т. д. В результате этого анализа были выявлены **основные особенности таких программ:**

- использование данных целого типа небольшой разрядности, например, 8-разрядные графические пиксели и 16-разрядная оцифровка звука;
- короткие циклы с высокими коэффициентами повторяемости;
- большое количество операций умножения и суммирования, в том числе из-за широкого использования быстрого преобразования Фурье;
- применение алгоритмов, требующих интенсивных вычислений;
- широкое использование операций с высоким уровнем параллелизма.

Было отмечено, что в мультимедийных приложениях 80 % времени выполнения программы приходится на 10-20 % программного кода.

Малая разрядность данных требует дополнительных действий при их обработке на 32-разрядном микропроцессоре, не позволяя в то же время использовать всю мощь 32-разрядной архитектуры.

Простым и наглядным примером такого рода обработки может служить изменение значений всех пикселей видеопамяти на определенную величину. Пусть емкость видеопамяти составляет 1 Мбайт, а каждый пиксель кодируется 1 байтом. Тогда для выполнения указанного действия потребуется выполнить примерно 1 млн операций по прибавлению константы к однобайтовому операнду, который выбирается из памяти. Одновременное выполнение таких действий над 4 операндами, что сократило бы количество операций в 4 раза, невозможно в классической архитектуре IA-32 из-за отсутствия соответствующих команд в системе команд и форматов используемых данных.

На устранение этих противоречий и были направлены основные усилия разработчиков технологии MMX. Процессор Pentium MMX, в котором впервые была реализована новая технология, был представлен фирмой INTel в январе 1997 года. Он позволил на 10-20 % повысить производительность на стандартных тестах, а для специализированных мультимедийных приложений - на 50 %.

Тема 1.4 Микроконтроллеры (МК). Общие сведения

1. Классификация. Архитектура. Обобщенная структурная схема микроконтроллера серии AVR. Основные элементы структурной схемы. Назначение. Характеристика. Логические основы построения микроконтроллеров; классификацию устройств памяти систему команд.

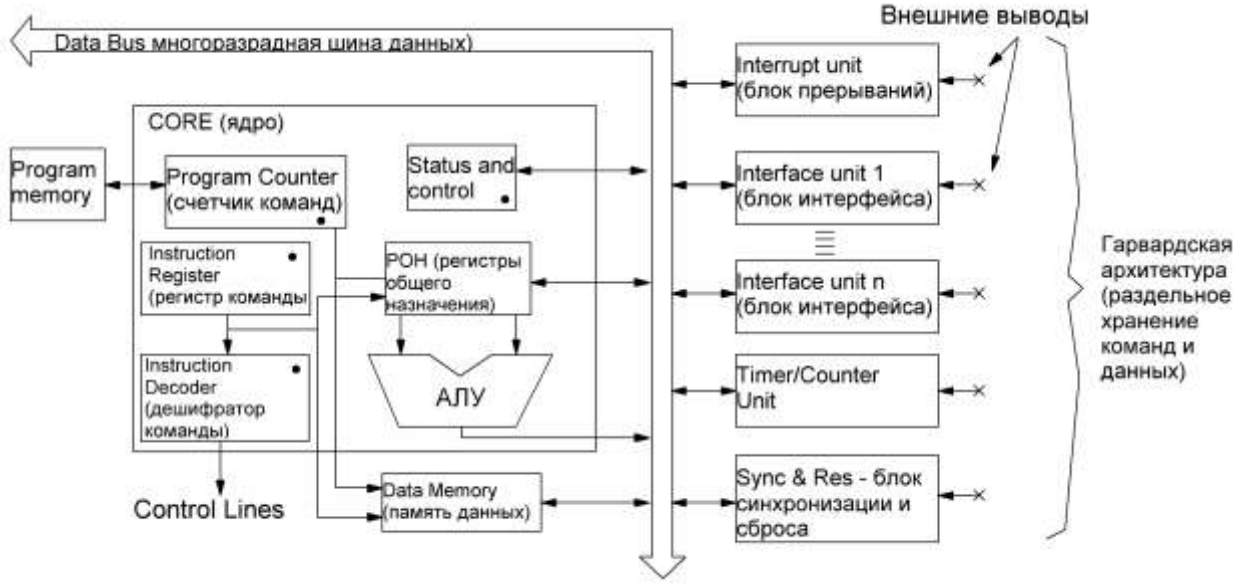


Рис.4

- -устройство управления, остальное – операционная часть.

На примере МК AVR:

Процессор

ядро - ЦПУ, построенное на принципах RISK-архитектуры. Основа этого блока – АЛУ).По системному тактовому сигналу из памяти программ в соответствии с содержимым счетчика команд (PC) выбирается очередная команда и выполняется АЛУ. Во время выбора команды из памяти программ происходит выполнение предыдущей выбранной команды, что и позволяет достичь быстродействия. АЛУ подключено к РОН (GPR). Регистров общего назначения всего 32, они имеют байтовый формат.

Память

В МК AVR реализована Гарвардская архитектура, по кот.с которой разделены не только адресные пространства памяти программ и памяти данных, но и шины доступа к ним. Каждая из областей памяти данных (RAM и EEPROM) также расположена в своем адресном пространстве. **Память программ (Flash ROM или Flash ПЗУ)**

Память программ предназначена для хранения последовательности команд, управляющих функционированием микроконтроллера, и имеет 16-ти битную организацию. Все AVR имеют Flash-память программ, которая может быть различного размера - от 1 до 256 КБайт. Ее главное достоинство в том, что она построена на принципе электрической перепрограммируемости. Программа заносится во Flash-память AVR как с помощью обычного программатора, так и с помощью SPI-интерфейса.

Память данных

Память данных разделена на три части: регистровая память, оперативная память (ОЗУ или RAM) и энергонезависимая память (ЭСППЗУ или EEPROM).

Регистровая память (РОН и РВВ)

Регистровая память включает 32 регистра общего назначения (РОН или GPR), объединенных в файл, и служебные регистры ввода/вывода (РВВ). И те и другие расположены в адресном пространстве ОЗУ, но не являются его частью.В области РВВ расположены различные служебные регистры (регистры управления микроконтроллером, регистры состояния и т. п.), а также регистры управления периферийными устройствами, входящими в состав микроконтроллера. *По сути, управление микроконтроллером заключается в управлении этими регистрами.*

Энергонезависимая память данных (EEPROM)

Для долговременного хранения различной информации, которая может изменяться в процессе функционирования микроконтроллерной системы, используется EEPROM-память. Этот тип памяти удобен для хранения промежуточных данных, различных констант, коэффициентов, серийных номеров, ключей и т.п.

Оперативная память (ОЗУ или RAM)

Используется для оперативного хранения данных. Число циклов чтения и записи в RAM не ограничено, но при отключении питающего напряжения вся информация теряется.

Периферия

Периферия МК AVR включает: порты, поддержку внешних прерываний, таймеры-счетчики, сторожевой таймер, аналоговые компараторы, 10-разрядный 8-канальный АЦП, интерфейсы UART, JTAG и SPI, устройство сброса по понижению питания, широтно-импульсные модуляторы.

Таймеры/счетчики (TIMER/COUNTERS)

Микроконтроллеры AVR имеют в своем составе от 1 до 4 таймеров/счетчиков (Т/С) с разрядностью 8 или 16 бит, которые могут работать и как таймеры от внутреннего источника тактовой частоты, и как счетчики внешних событий. Их можно использовать для точного формирования временных интервалов, подсчета импульсов на выводах МК, формирования последовательности импульсов, тактирования приемопередатчика последовательного канала связи. В режиме ШИМ (PWM) Т/С - широтно-импульсный модулятор и используется для генерирования сигнала с программируемыми частотой и скважностью. Т/С способны вырабатывать запросы прерываний.

Сторожевой таймер (WDT) Сторожевой таймер (WatchDogTimer) предназначен для предотвращения катастрофических последствий от случайных сбоев программы. Он имеет свой собственный RC-генератор, работающий на частоте 1 МГц

Аналого-цифровой преобразователь (A/D CONVERTER) Аналого-цифровой преобразователь (АЦП) служит для получения числового значения

напряжения, поданного на его вход. Этот результат сохраняется в регистре данных АЦП. входом АЦП, определяется числом, занесенным в соответствующий регистр.

Универсальный последовательный приемопередатчик (UART или USART) Универсальный асинхронный или синхронно/асинхронный приемопередатчик (UART или USART) - последовательный интерфейс для организации информационного канала обмена микроконтроллера с внешним миром. Способен работать в дуплексном режиме. Он поддерживает протокол стандарта RS-232, что дает возможность связываться с ПК.

Последовательный периферийный трехпроводный интерфейс SPI SPI (SerialPeripheralInterface) - для организации обмена данными между 2 устройствами. С его помощью может осуществляться обмен данными между микроконтроллером и различными устройствами, такими, как цифровые потенциометры, ЦАП/АЦП, FLASH-ПЗУ и др, а также между несколькими микроконтроллерами AVR. Также, через SPI может осуществляться программирование микроконтроллера.

Двухпроводной последовательный интерфейс TWI

TWI (Two-wireSerialInterface) позволяет объединить вместе до 128 различных устройств с помощью двунаправленной шины, состоящей из линии тактового сигнала (SCL) и линии данных (SDA).

Тактовый генератор Тактовый генератор вырабатывает импульсы для синхронизации работы всех узлов микроконтроллера. Внутренний тактовый генератор AVR может запускаться от нескольких источников опорной частоты (внешний генератор, внешний кварцевый резонатор, внутренняя или внешняя RC-цепочка). Минимальная допустимая частота ничем не ограничена (вплоть до пошагового режима).

Тема 1.5. Микроконтроллеры семейства серии AVR

1.Общие сведения. Архитектура. Регистры общего назначения (РОН). Регистры ввода – вывода. Память. Память программ и память данных. Счетчики команд и стековая память.

Регистры памяти – простейший вид регистров – хранят двоичную информацию. Это набор синхронных триггеров, каждый из которых хранит один разряд двоичного числа. Если регистр построен на триггерах-защелках, то такой регистр называют регистром-защелкой. Схема такого регистра представлена на рис.4.

Типовыми внешними связями регистра являются информационные входы D_i, вход сигнала записи C (или загрузки синхронизации), вход гашения (установки в 0) R, выходы триггеров Q_i, разрешение выхода (чтения) RE, разрешение приема информации (записи) WE. Возможны другие обозначение информационных и управляющих входов и выходов. Условное изображение регистра показано на рис. 8.2.

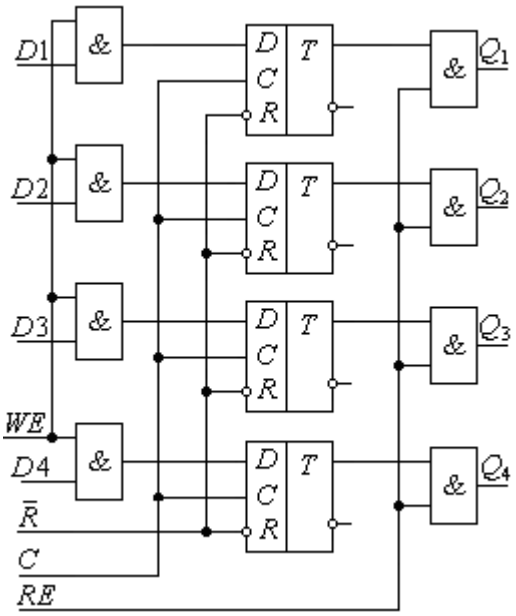
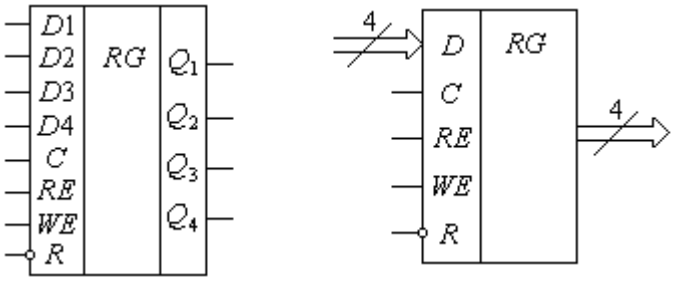


Рис. 4. Регистр памяти

Изображение по варианту *а* используется, когда нужно показать каждый вход и выход данных. Если же тракт данных рассматривается как единое, укрупненное понятие – шина данных, то используется обозначение, показанное на рис. 5 *б*.



а б

Рис. 5. Условное изображение регистра: *а* – с раздельными линиями по разрядам; *б* – с информационными линиями в виде шины

Часто регистры дополняются элементами отключения выходных шин. Тогда микросхема имеет дополнительный вход перевода в третье состояние *EZ*. Ввод (запись) и вывод (считывание) производится одновременно во всех разрядах при наличии разрешения *WE* или *RE*.

С приходом очередного тактового импульса происходит обновление информации. Считывание информации может осуществляться в однофазном виде в прямом или обратном коде (с выходов $\overline{Q}$) или в парафазном виде.

В качестве подобных регистров могут быть использованы без дополнительных элементов многие синхронные триггеры: К155ТМ5, ТМ7, ТМ8, 564ТМ3, 555ТМ8 и др.

Наращивание разрядности регистра достигается добавлением нужного числа триггеров, тактовые входы которых присоединяют к линии синхронизации.

На рис. 6 представлен регистр К155ИР15 – регистр с тремя состояниями. Здесь $D1 - D4$ – информационные входы, C – синхронизирующий вход. Запись осуществляется по фронту 0,1 (⎍),$Q_1 - Q_4$ – прямые выходы, E_1, E_2 – разрешающие входы. Запись возможна при $E_1 = E_2 = 0$ (если на одном из входов E логическая 1, то это режим хранения информации). Входы EZ_1,EZ_2 – перевод в 3-е состояние, R – установка в 0 (высоким уровнем): с приходом 1 на вход R все триггеры устанавливаются в 0. При вводе информации на входах E_1, E_2 и R должен быть 0. Если на EZ (любом) логическая 1 – выходы отключаются. Более подробные сведения о режимах работы микросхемы можно посмотреть, например, в [12, 13].

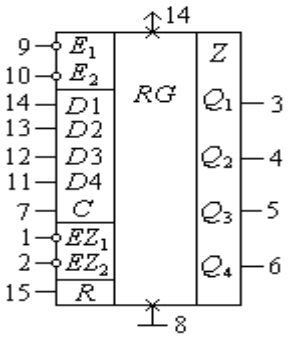


Рис. 8.3. Цоколевка регистра К155ИР15

Рис. 6. Цоколевка регистра К155ИР15

Выпускаемые промышленностью регистры иногда объединяют на кристалле микросхемы с другими узлами, в паре с которыми регистры часто используются в схемах цифровой аппаратуры. Пример такого комплексного узла – микросхема многорежимного буферного регистра К589ИР12. Условное обозначение и структура регистра представлены на рис. 7. Микросхема состоит из восьми информационных D-триггеров, восьми выходных буферных устройств с тремя состояниями, отдельного D-триггера для формирования запросов на прерывание и гибкой схемы управления режимами работы регистра.

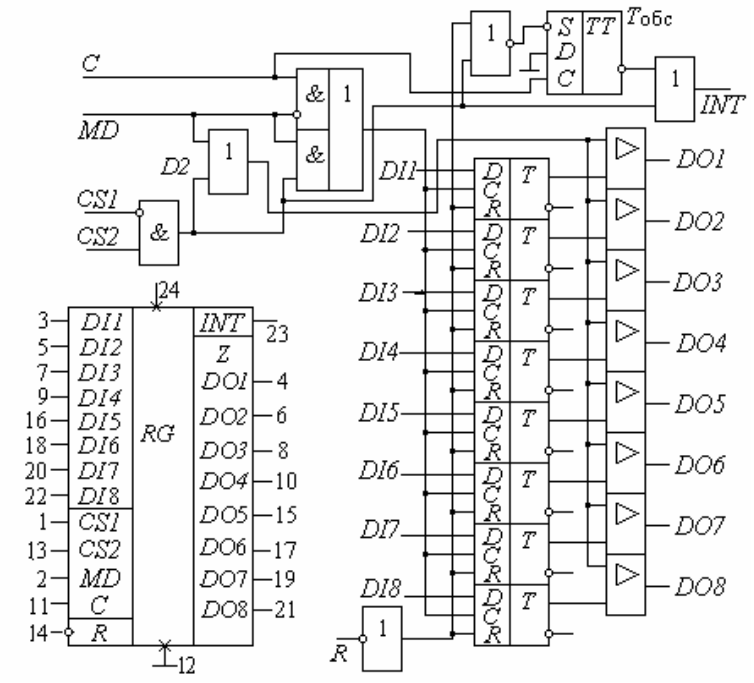


Рис. 7. Условное обозначение (*а*) и функциональная схема (*б*) регистра ИР12

Микросхема имеет 24 вывода, из которых шестнадцать служат для ввода входных данных – *DI1-DI8* и вывода выходных данных – *DO1 - DO8.*

Управляющие сигналы: *MD* – выбор режима; *CS1, CS2* – сигналы выбора кристалла; *C* – стробирующий сигнал; *R* – очистка (сброс); *INT* – сигнал прерывания, выдаваемый микропроцессору.

Запись информации в регистр обеспечивается одной из следующих комбинаций управляющих сигналов: *CS1·CS2·MD* v *C·MD*, где *CS1, CS2* – входы выборки кристалла; когда *CS1*=0 *и CS2*=1, регистр выбирается; C – вход для подачи стробирующего сигнала при записи; MD – вход выбора режима, используется для определения тактирующего сигнала на входе C триггеров регистра (CS1·*CS2* при *MD=1* или *C* при *MD=0*) и для управления состоянием выходного буфера.

Чтение информации с регистра осуществляется одной из двух комбинаций входных сигналов: *CS1, CS2* v MD. Формирование сигнала прерывания INT возможно при записи или чтении информации с регистра: *CS1·CS2 v C*. Сброс запроса прерывания INT осуществляется сигналом очистки регистра R=0, или при CS1·CS2=1 триггер прерывания устанавливается в 1.

Часто регистры объединяются в блоки регистровой памяти – *регистровые файлы*. Такие микросхемы, могут быть объединены с входным дешифратором, позволяющим принимать входные данные в соответствующий регистр, выбираемый сигналами на адресных входах микросхемы. Объединяют регистры и с выходным мультиплексором *MS*, позволяющим выбирать содержи-

мое соответствующего регистра. Пример схемы такого устройства представлен на рис. 8.

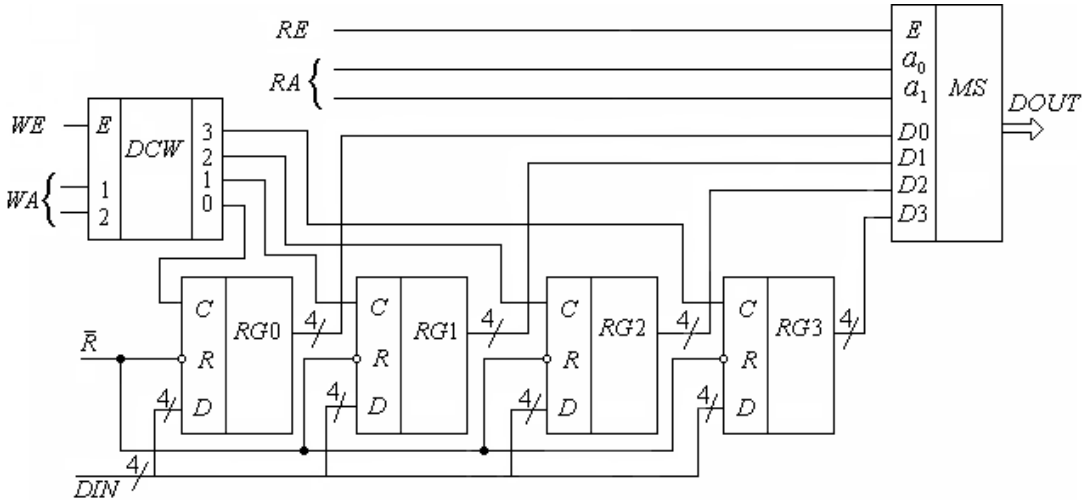


Рис. 8. Схема регистра общего назначения

Входы D_i четырёх регистров подключены к общей входной шине данных *DIN*. Вход загрузки требуемого регистра выбирается дешифратором записи *DCW* на основании поступающего на вход *DCW* адреса записи *WA* (write address), т. е. кода номера загружаемого регистра. Запись данных, присутствующих на шине *DIN*, происходит по сигналу разрешения записи *WE* (write enable).

Выходы регистров мультиплексором *MS* подключаются к выходной шине *DOUT* (data out). Номер регистра, с которого происходит чтение, определяет код адреса чтения *RA* (read address). Выдачу данных в шину *DOUT* разрешает сигнал *RE* (read enable).

Поскольку дешифрация адреса записи и адреса чтения производится независимыми узлами (*WA* и *RA*), регистровая память может одновременно записывать данные в один регистр и читать из другого.

Описанная структура использована в микросхеме 155РП1. Аналогичные, но более развитые структуры имеют регистровые памяти ИР11 и ИР12 серий К561 и К564, ИР26 155 серии и др.

Микросхемы регистровой памяти легко наращиваются по разрядности и допускают наращивание по числу регистров. Они разработаны для построения блоков регистров общего назначения (РОН) и других специализированных блоков памяти небольшого объема, предназначенных для временного хранения исходных данных и промежуточных результатов в цифровом устройстве.

Для памяти с большим числом регистров часто используют один дешифратор адреса и при записи и при чтении. Такую память называют *памятью с произвольным доступом* (по ЕСКД это RAM) – ОЗУ. В таких устройствах и ввод, и вывод данных часто осуществляется через одни и те же выводы корпуса микросхемы за счёт использования элементов с тремя состояниями. Пример реализации двунаправленной шины управления рассмотрен ранее (см. рис.

3.35, *а*). Здесь при наличии на входе *Z* сигнала низкого уровня (0) информация передается слева направо, при наличии высокого уровня – справа налево.

Микросхемы ОЗУ малой ёмкости есть в составе распространённых серий. Они имеют входы адреса (A_i), данных (D_i), режима *W*/*R*, выходы данных Q_i, вход разрешения *E* (ВМ, ВК, *CS* – выбор кристалла).

Примеры микросхем ОЗУ:

155РУ2 – 16*4 (16 слов по 4 разряда);
155РУ5 – 256*1;
564РУ2 – 256*1.

Это статические ОЗУ. Они более быстродействующие, чем динамические

2. Периферия микроконтроллера. Подсистема ввода – вывода. Система прерываний. Таймеры-счетчики, сторожевой таймер. Другие встроенные периферийные устройства. Основные понятия. Аналоговые компараторы (Analog Comparator). Аналого-цифровой преобразователь - АЦП (A/D CONVERTER). Интерфейсы. Универсальный последовательный асинхронный приемопередатчик (UART / USART) Интерфейсы UART. Последовательный периферийный интерфейс (SPI.). Последовательный двухпроводный интерфейс (TWI). Другие ячейки.

Регистровый файл SRAMзанимает младшие 32 байта в общем адресном пространстве ОМК. Шесть из 32-х регистров файла могут использоваться как три 16-разрядных указателя адреса при косвенной адресации данных. Один из этих указателей (Z Pointer) применяется также для доступа к данным, записанным в памяти программ микроконтроллера. Использование трех 16-битных указателей (X, Y и Z Pointers) существенно повышает скорость пересылки данных при работе прикладных программ.

Память программ. Все AVR- МК имеют Flash-память программ, которая может быть загружена как с помощью обычного программатора в параллельном формате, так и с помощью SPI- интерфейса, в том числе непосредственно на плате. Число циклов перезаписи – не менее 1000. Некоторые версии МК Mega имеют возможность самопрограммирования, т.е. микроконтроллер способен самостоятельно, без внешнего программатора, изменять содержимое ячеек памяти программ. Это дает возможность записать во внешнюю энергонезависимую память несколько рабочих версий программы, а потом при необходимости или по реакции на логические условия перегружать рабочие программы в тот же микроконтроллер AVR без извлечения его из печатной платы. Для этого весь массив памяти программ делится на две неравные по объему области: блок загрузчика (программа, которая управляет перезаписью Flash-памяти программ) и блок для размещения рабочих программ. Программа- загрузчик создается разработчиком и должна быть запрограммирована внешним программатором.

Память данных. Все AVR- МК имеют энергонезависимую память данных с

электрическим стиранием EEPROM. Этот тип памяти используется для хранения промежуточных данных, констант, таблиц перекодировок, калибровочных коэффициентов и т.п. Данные в EEPROM могут быть загружены как через SPI- интерфейс, так и с помощью обычного программатора. Число циклов перезаписи – не меньше 100 000. Два программируемых бита защиты информации позволяют защитить память программ и энергонезависимую память данных EEPROM от несанкционированного считывания.

Внутренняя оперативная память SRAM есть во всех AVR семейств Classic и Mega и в новом кристалле семейства Tiny – ATtiny26/L. Для некоторых МК возможно подключение внешней памяти данных объемом до 64 Кбайт.

Число независимых линий портов ввода/ вывода – от 3 до 53. Каждый разряд порта может быть запрограммирован на ввод или вывод информации. Мощные выходные драйверы обеспечивают токовую нагрузочную возможность 20 мА по линии порта при максимальном значении 40 мА. Общая токовая нагрузка на все линии одного порта не должна превышать 80 мА (для Vdd = 5 В).

Любой микроконтроллер AVR обязательно содержит набор периферийных устройств. Периферийные они по отношению к центральному процессорному устройству (ЦПУ) микроконтроллера. Но находятся они также внутри микросхемы. Ниже перечислены все возможные периферийные устройства, которые могут входить в состав микроконтроллера AVR.

Встроенные таймеры/счетчики.

Микроконтроллеры AVR могут содержать от одного до четырех таймеров/счетчиков. Причем используются как восьми-, так и шестнадцатиразрядные таймеры. Их количество на один микроконтроллер может составлять от одного до шести.

Генератор сигнала с широтно-импульсной модуляцией (ШИМ).Генерация сигнала ШИМ – это один из режимов работы таймера/счетчика. Одна микросхема может иметь от 2 до 12 каналов ШИМ, а может не иметь ни одного.

Аналоговый компаратор.

Входит в состав практически всех микроконтроллеров AVR.

Аналогово-цифровой преобразователь (АЦП).АЦП микроконтроллеров AVR могут иметь от четырех до шестнадцати каналов. То есть могут преобразовывать в цифровой эквивалент до 16 входных аналоговых сигналов. На самом деле канал АЦП всегда один. Но на его входе стоит аналоговый мультиплексор. Поэтому АЦП способен подключаться к нескольким разным источникам аналогового сигнала.

Последовательный интерфейс.

Микросхемы AVR способны поддерживать несколько разных видов последовательных интерфейсов. Каждый такой интерфейс реализует один или не-

сколько известных стандартов передачи информации. Один из видов такого интерфейса поддерживает тот же стандарт, что и COM-порт персонального компьютера. Есть также интерфейс, поддерживающий стандарт I2C шины. SPI-интерфейс может использоваться как для последовательного программирования памяти программ, так и для связи нескольких микроконтроллеров в мультипроцессорной системе.

Микроконтроллеры AVR содержат от 1 до 4 таймеров/счетчиков, которые могут работать как таймеры от внутреннего источника опорной частоты, и как счетчики внешних событий.

Таймер/счетчик RTC (есть во всех микроконтроллерах семейства Mega и в некоторых МК Classic) реализует систему реального времени. Таймер имеет собственный делитель, который может быть программным способом подключен или к основному внутреннему источнику тактовой частоты МК, или к дополнительному асинхронному источнику опорной частоты (кварцевый резонатор или внешний синхросигнал). Для этой цели МК имеет два внешних вывода. Внутренний осцилятор, нагруженный на счетный вход таймера/счетчика RTC, оптимизирован для работы с внешним "часовым" кварцевым резонатором на 32,768 кГц.

Блок SPI предназначен для последовательного ввода и вывода данных, также используется для программирования после установки МК на печатную плату.

Сторожевой таймер WDT служит для перезапуска программы при появлении сбоя в ходе ее выполнения. Программа, которая работает без сбоев, периодически сбрасывает сторожевой таймер, не допуская его переполнения. Сторожевой таймер имеет собственный RC-генератор, который работает на частоте 1 МГц. На входе WDT включен делитель с программируемым коэффициентом деления, позволяющий регулировать временной интервал переполнения таймера для сброса микроконтроллера. Таймер WDT может быть отключен программным способом во время работы микроконтроллера как в активном режиме, так и в любом из режимов пониженного энергопотребления. В последнем случае это приводит к значительному снижению потребляемого тока.

Аналоговый компаратор AC сравнивает по величине напряжения сигналы, поступающие на входы P1.0 и P1.1 Результат сравнения подается на внутреннюю линию, которая не имеет внешнего вывода.

Аналого-цифровой преобразователь ADC построен по схеме последовательного приближения с устройством выборки/хранения (УВХ). Каждый из аналоговых входов может быть соединен со входом УВХ через аналоговый мультиплексор. Устройство выборки/хранения имеет собственный усилитель. Разрядность АЦП – 10 бит при нормированной погрешности ± 2 МЗР. АЦП может работать в двух режимах – однократное преобразование по любому выбранному каналу и последовательный циклический опрос всех каналов. Время преобразования выбирается программно с помощью установки

коэффициента деления частоты специального предделителя, который входит в состав блока АЦП. Оно составляет от 60 до 280 мкс.

Внутренний тактовый генератор МК AVR может запускаться от нескольких источников опорной частоты (внешний генератор, внешний кварцевый резонатор, внутренняя или внешняя RC-цепочка). AVR-микроконтроллеры полностью статические – минимальная допустимая частота ничем не ограничена, т.е. возможно обеспечить даже пошаговый режим выполнения программы.

Микроконтроллеры AVR могут быть переведены программным путем в один из шести режимов пониженного энергопотребления:

1. Режим холостого хода (IDLE), в котором прекращает работу только процессор и фиксируется содержимое памяти данных, а внутренний генератор синхросигналов, таймеры, система прерываний и WDG-таймер продолжают функционировать.

2. Режим микропотребления (Power Down), при котором сохраняется содержимое регистрового файла, но останавливается внутренний генератор синхросигналов. Выход из Power Down возможен или по общему сбросу микроконтроллера, или по сигналу от внешнего источника прерывания. При включенном WDG-таймере ток потребления в этом режиме составляет около 80 мкА, а при выключенном – меньше 1 мкА для всех типов AVR (для величины питающего напряжения 5 В).

3. Режим хранения энергии (Power Save), реализованный только в тех AVR, которые имеют в своем составе систему реального времени. В основном режим Power Save идентичен режиму Power Down, но он допускает независимую работу таймера/счетчика RTC. Выход из режима Power Save возможен по прерыванию, вызванному или переполнением таймера/счетчика RTC, или срабатыванием блока сравнения этого счетчика. Ток потребления в этом режиме составляет » 10 мкА при напряжении питания 5 В на частоте 32,768 кГц.

4. Режим подавления шума при работе аналого-цифрового преобразователя (ADC Noise Reduction). В этом режиме останавливается работа процессора, но разрешена работа АЦП, двухпроводного интерфейса I2C и сторожевого таймера.

5. Основной режим ожидания (Standby). Отличается от режима Power Down тем, что работа тактового генератора не прекращается. Это гарантирует быстрый выход микроконтроллера из режима ожидания всего за 6 тактов генератора.

6. Дополнительный режим ожидания (Extended Standby). Идентичный режиму Power Save, но работа тактового генератора тоже не прекращается, что гарантирует быстрый выход из режима за 6 тактов генератора.

Система команд AVRнасчитывает до 133 различных команд. Различают пять

групп команд AVR: условного разветвления, безусловного разветвления, арифметические и логические операции, команды пересылки данных, команды работы с битами. В последних AVR Mega реализована функция аппаратного умножения. По разнообразию и количеству команд AVR больше похожи на CISC-, чем на RISC-процессоры.

Раздел 2 Алгоритмизация и программирование микроконтроллеров

Тема 2.1 Языки программирования

1. Основные этапы эволюции языков программирования от машинных кодов и ассемблера до языков высокого уровня.

Обратимся к истокам развития вычислительной техники. Вспомним самые первые компьютеры и программы для них. Это была эра программирования непосредственно в машинных кодах, а основным носителем информации были перфокарты и перфоленты. Программисты обязаны были знать архитектуру машины досконально. Программы были достаточно простыми, что обуславливалось, во-первых, весьма ограниченными возможностями этих машин, и, во-вторых, большой сложностью разработки и, главное, отладки программ непосредственно на машинном языке. Вместе с тем такой способ разработки давал программисту просто невероятную власть над системой. Становилось возможным использование таких хитроумных алгоритмов и способов организации программ, какие и не снились современным разработчикам. Например, могла применяться (и применялась!) такая возможность, как самомодифицирующийся код. Знание двоичного представления команд позволяло иногда не хранить некоторые данные отдельно, а встраивать их в код как команды. И это далеко не полный список приемов, владение хотя бы одним из которых сейчас сразу же продвигает вас до уровня «гуру» экстракласса.

2. Ассемблер

Первым значительным шагом представляется переход к языку ассемблера (позволим себе маленькое лирическое отступление: английское название assembly language, или assembler, на русский переводят именно тем термином, который был использован выше. При этом у новичка создается впечатление, что язык назван в честь некоего человека по имени ассемблер. Достаточно забавная ситуация, не правда ли?). Не очень заметный, казалось бы, шаг — переход к символическому кодированию машинных команд — имел на самом деле огромное значение. Программисту не надо было больше вникать в хитроумные способы кодирования команд на аппаратном уровне. Более того, зачастую одинаковые по сути команды кодировались совершенно различным образом в зависимости от своих параметров (широко известный пример из мира современных компьютеров — это кодирование инструкции mov в процессорах Intel: существует несколько совершенно по-разному кодируемых вариантов команды; выбор того или иного варианта зависит от операндов, хотя суть выполняемой операции неизменна: поместить содер-

жимое (или значение) второго операнда в первый). Появилась также возможность использования макросов и меток, что также упрощало создание, модификацию и отладку программ. Появилось даже некое подобие переносимости — существовала возможность разработки целого семейства машин со сходной системой команд и некоего общего ассемблера для них, при этом не было нужды обеспечивать двоичную совместимость.

Вместе с тем, переход к новому языку таил в себе и некоторые отрицательные (по крайней мере, на первый взгляд) стороны. Становилось почти невозможным использование всяческих хитроумных приемов сродни тем, что упомянуты выше. Кроме того, здесь впервые в истории развития программирования появились два представления программы: в исходных текстах и в откомпилированном виде. Сначала, пока ассемблеры только транслировали мнемоники в машинные коды, одно легко переводилось в другое и обратно, но затем, по мере появления таких возможностей, как метки и макросы, дизассемблирование становилось все более и более трудным делом. К концу ассемблерной эры возможность автоматической трансляции в обе стороны была утеряна окончательно. В связи с этим было разработано большой количество специальных программ-дизассемблеров, осуществляющих обратное преобразования, однако в большинстве случаев они с трудом могут разделить код и данные. Кроме того, вся логическая информация (имена переменных, меток и т.п.) теряется безвозвратно. В случае же задачи о декомпиляции языков высокого уровня примеры удовлетворительного решения проблемы и вовсе единичны.

2.Этапы разработки программы. Способы алгоритмизации и программирования работы микроконтроллеров

Разработка программ включает в себя следующие этапы:

1. Анализ и уточнение требований, предъявляемых к программе. Иногда этот этап называют постановкой задачи.
2. Проектирование алгоритма и выбор структур данных (или алгоритмизация).
3. Программирование и отладка.
4. Тестирование программы.
5. Документирование, подготовка инструкции для пользователя программы.

Первые два этапа являются определяющими этапами разработки программного обеспечения. Этап анализа и уточнения требований, предъявляемых к программе – необходимый и весьма ответственный этап , который осуществляется совместно пользователем (заказчиком) программы и ее разработчиком. На этом этапе на основе требований заказчика уточняются три основных момента:

- исходные данные программы: формат данных, источник и порядок ввода, объем исходных данных и ограничения на их значения;

- выходные данные программы: содержание, формат и порядок вывода, возможный объем исходных данных, заголовки и предполагаемые ограничения на их значения;
- аварийные ситуации при вводе и обработке данных и действия пользователя, предусматриваемые при их возникновении.

На этапе анализа и уточнения требований создается более полное, уточненное описание задачи, называемое спецификацией задачи. Спецификация задачи – это очень важный документ, который является техническим заданием на разработку программы и частью соглашения (договора) между разработчиком программного обеспечения и заказчиком.

В спецификации задачи выделяют две ее части:

- функциональную спецификацию;
- эксплуатационную спецификацию.

Функциональная спецификация описывает объекты, относящиеся к задаче: разбиение на предполагаемые подзадачи, их входные и выходные данные, связи между ними, реакции на аварийные ситуации.

Эксплуатационная спецификация содержит требования к скорости работы программы и используемым ресурсам памяти, характеристикам и конфигурации ЭВМ, на которой должна работать программа, и специальные требования к надежности и безопасности ее работы.

На втором этапе разработки программного обеспечения проектируется алгоритм решения задачи в соответствии с заданной спецификацией, формируется общая структура программы, определяются компоненты и их входные и выходные параметры.

Основным подходом к проектированию структуры программы является нисходящее проектирование или пошаговое уточнение. Этот подход заключается в том, что программа первоначально рассматривается как «черный ящик», который выполняет некоторую функцию F, преобразующую единственные входные данные в единственные выходные данные. Это общая функция F может быть разделена на ряд более простых подфункций F1,F2,…,Fk. Каждая из этих функций сама по себе представляет такой же «черный ящик».

Процесс разбиения завершается тогда, когда любая из определенных функций может быть описана с помощью простых базовых операций и управляющих структур алгоритмов.[3]В процессе проектирования формируется структура задачи в виде дерева подзадач (подфункций) (См. рис. 9).

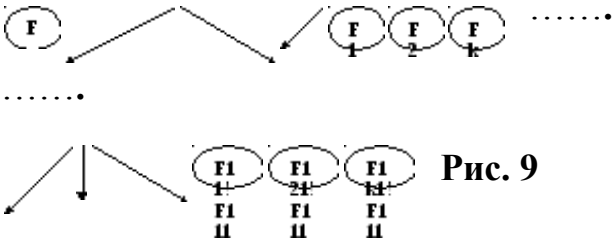


Рис. 9

Эта иерархическая структура задачи определяет структур компонент создаваемой программы. Компонента программы называется подпрограммой. В случае, если задача является сложной, целесообразно программу разделить на компоненты, одна из которых будет главной и организующей вычислительный процесс, а другие компоненты будут подпрограммами, которые будут выполняться в порядке, соответствующем алгоритму. Программы, выполняющие относительно простые задачи, могут не содержать подпрограмм.

Результатом данного этапа разработки является формализованное описание алгоритма или проект программы. Формы представления алгоритмов рассмотрены в п. 3.2.

Третьим этапом разработки программы является программирование и отладка. На этом этапе алгоритм записывается на языке программирования высокого уровня, и производится отладка программы с помощью соответствующей системы программирования (транслятора данного языка). Во время отладки исправляются обнаруженные в тексте программы ошибки. Результатом отладки является текст программы, не содержащей ошибок и выполненной хотя бы один раз на ПЭВМ. Таким образом, создается работоспособная программа.

На этапе тестирования программы выполняется проверка правильности работы программы с использованием заранее подготовленных тестов. Тест – это набор исходных данных и соответствующих им результатов, которые должна выдавать программа при обработке этих исходных данных. Для сложных задач составление тестов является весьма трудоемкой работой, включающей ручные расчеты или использование других программ-аналогов.

При обнаружении неправильных результатов на этапе тестирования происходит возврат на предыдущие этапы разработки программы. Результатом тестирования является окончательная версия программы и документация о результатах выполнения тестов.

Последний этап разработки ПО – подготовка пользовательской документации, состав которой определяется заказчиком. Обычно документация для пользователя включает в себя:

описание применения, которое содержит сведения о назначении программы, требования к техническим и программным средствам ЭВМ, состав и функции программы, применяемые математические алгоритмы;

руководство пользователя, в котором описаны входные и выходные данные, способы запуска программы, режимы работы, дан перечень сообщений и описана реакция пользователя на каждое сообщение.

Тема 2.2 Трансляция программ

1. Транслятор. Трансляция программы и получение файла прошивки для микроконтроллера. Краткий обзор содержимого файла прошивки. Разбор файла описаний и листинга программы. Размещение программы в памяти микроконтроллера

После того, как текст программы набран и записан на жесткий диск, необходимо произвести трансляцию программы. В процессе трансляции создается результирующий (объектный) файл, который представляет собой ту же программу, но в машинных кодах, предназначенную для записи в программную память микроконтроллера. Результирующий файл имеет расширение hex.

Кроме hex-файла транслятор создает еще несколько вспомогательных файлов. И главное, файл с расширением eep. Этот файл имеет точно такую же внутреннюю структуру, как файл hex. А содержит он информацию, предназначенную для записи в EEPROM. Такая информация появляется в том случае, когда в тексте программы переменным, размещенным в сегменте eeprom, присвоены начальные значения. Если в лабораторных работах этого не делать, то файл с расширением eep во всех проектах будет пустой (будет содержать лишь завершающую строку).

Теперь немного разберемся с форматом файлов hex и eep. В обоих случаях применяется так называемый HEX-формат, который практически является стандартом для записи результатов транслирования различных программ. Он поддерживается практически всеми трансляторами с любого языка программирования.

В принципе, программисту не обязательно знать структуру этого формата. Достаточно понимать, что в hex-файле определенным способом закодирована программа в машинных кодах. Именно этот файл используется программатором для «прошивки» программной памяти микроконтроллера. Любой программатор поддерживает hex-формат и распознает записанные туда коды автоматически. Приведем краткое описание hex-формата.

1.4.2 Формат hex-файла

Если вы посмотрите содержимое такого файла при помощи редактора «Блокнот», то вы увидите, что это текстовый файл, в котором данные закодированы в виде текстовых строк. Ниже приведен пример содержимого hex-файла, полученного в результате трансляции одного из вариантов программы Progl:

:020000020000FC

:100000000FE70DBF00E806BD00E006BD01BB0FEF26

:1000100007BB08BB02BB00E808B900B308BBFDCFB3

:00000001FF

Как видите, данный файл состоит из четырех строк. Первая и последняя строки несут служебную информацию. Наличие первой строки необязательно. Система AVR Studio при трансляции программы всегда добавляет в hex-файл первую строку именно такого содержания. Последняя строка – это стандартный конец для любого hex-файла.

Оставшиеся две строки как раз и содержат информацию о кодах программы. В каждой такой строке закодирована цепочка байтов и адрес в памяти, где эти байты должны размещаться.

Строка начинается с двоеточия. Двоеточие – обязательный элемент, который служит для идентификации hex-формата. Все остальные символы в строке – это шестнадцатеричные числа, записанные слитно без пробелов. Отдельные числа отличают по их позиции в строке. Так первые два знака занимает шестнадцатеричное число, означающее длину цепочки.

В нашем случае длина обеих цепочек равна 0x10 (то есть 16) байт. Следующие четыре символа — это начальный адрес, куда эти байты должны быть помещены. Первая цепочка будет размещена в памяти, начиная с нулевого адреса. Вторая цепочка — с адреса 0x0010. Очередные два знака занимает код вида строки. В интересующих нас строках он равен «00», что означает, что эти строки предназначены для записи данных (в первой строке такой код равен «02», а в последней «01»).

Сразу после кода вида строки начинаются собственно данные. Каждый байт данных занимает два знака. Самые последние два символа — это контрольная сумма. Она рассчитывается по специальной формуле с использованием значений всех байтов цепочки и служит для проверки на отсутствие ошибок.

Тема 2.3 Краткий обзор программаторов

1. Программаторы. Последовательные и параллельные программаторы. Внутрисхемное программирование

Все модели программаторов ТРИТОН позволяют программировать микросхемы непосредственно в устройствах пользователя. Для этого не только микросхема, но и само устройство должны поддерживать режим внутрисхемного программирования. В устройстве должно быть предусмотрено подключение программатора. Поскольку для программирования многих микросхем используются напряжения, значительно превышающие напряжение питания, устройство должно выдерживать эти напряжения. Подключаемое устройство не должно оказывать шунтирующего влияния на сигналы программатора. Как правило, все эти требования подробно описаны в фирменных спецификациях по программированию на каждую микросхему.

Программирование микросхем внутрисхемно может осуществляться двумя способами:

- непосредственно сигналами с панельки программатора.
- с помощью специального переходника TSH-ICSP.

Программирование сигналами с панельки программатора

Для этого нужно просто соединить выводы микросхемы с соответствующими выводами панельки программатора и, если микросхема имеет несколько алгоритмов программирования, выбрать режим ICSP. Посмотреть на каких выводах программатор сформирует необходимые сигналы, можно в документации на данную микросхему или в меню “Микросхема\Параметры”.

Возможные проблемы. Ключи программатора, которые формируют логические сигналы и обеспечивают чтение данных с микросхемы, построены по схеме с открытым коллектором, в нагрузке которого стоит резистор 10k. Поэтому, если в устройстве пользователя к выводам DATA или CLOCK подключена какая-либо низкоомная нагрузка, то возможно шунтирование сигналов программатора и как следствие сбои в работе. В этом случае, непосредственно на разъем для внутрисхемного программирования, между VCC и сигналом DATA (или CLOCK) можно установить дополнительный резистор номиналом 0,5..1k, который будет подтягивать уровень сигнала и увеличивать нагрузочную способность. Для устойчивой работы необходимо отношение сопротивления нагрузки к сопротивлению в открытом коллекторе не менее, чем 5:1

Программирование через переходник TSH-ICSP

В отличии от выводов панельки программатора, которые построены по схеме с ОК, логические выходы переходника TSH-ICSP буферизированы, т.е. выполнены по схеме push-pull и обеспечивают ток нагрузки в нуле и единице до 15-20mA. Остальные сигналы: питание, земля и напряжение программирования передаются на выходы переходника без изменений. Необходимо отметить, что ток нагрузки логических выходов идет за счет источника питания в программаторе, максимальный ток которого не должен превышать 80mA.

Чтобы использовать переходник TSH-ICSP его необходимо “Включить”. Для этого, после выбора микросхемы, в меню “Микросхема\Параметры” необходимо установить флаг “Использовать TSH-ICSP”. Программа считает из файла “ICSP.CFG” начальную конфигурацию, сделает необходимые настройки и покажет на экране цоколевку разъема. Если в появившемся окне этого флага нет, значит выбранная микросхема не может программироваться в устройстве пользователя или программатор не поддерживает этот режим.

Переходник для внутрисхемного программирования TSH-ICSP поддерживает технологию универсальных алгоритмов и может быть частично переконфигурирован пользователем в меню: “Микросхема\Параметры”

выводы с 1 по 4	Vcc, Vpp, GND, логический вход/выход с ОК.
выводы 5, 7, 9	буферизированные логические входы/выходы, вход/выход с ОК или GND.
выводы 6, 8	Vpp, GND, логический вход/выход с ОК.
вывод 10	Vcc, GND, логический вход/выход с ОК.
Для увеличения нагрузочной способности выходов с ОК на выводах 6, 8 и 10 на плате переходника предусмотрена возможность установки дополнительных подтягивающих резисторов номиналом 0,5..1k.	

Если расположение сигналов на выходах панельки не совпадает с цоколевкой разъема на плате, то программа позволяет изменить конфигурацию выводов. Для этого в положениях “Начальное состояние” и “Управляющие сигналы” надо мышкой перетащить сигналы на требуемые выводы, после чего сохранить настройки. Чтобы после переназначения сигналов, переходник работал корректно, менять конфигурацию надо в двух местах:

- в “Начальном состоянии”, где задается состояние выводов, которое будет подано на микросхему в момент включения питания, и которое обеспечивает вход микросхемы в режим программирования.
- и в положении “Управляющие сигналы”, где задаются номера выводов на которых программатор будет формировать логические сигналы.

Программа учитывает схемотехнику переходника и программатора, и подсвечивает неправильно сконфигурированные сигналы желтым цветом. Кроме того, при перетаскивании управляющих сигналов, программа может исправлять некоторые ошибки.

Чтобы каждый раз не повторять эту операцию, можно сохранить текущие настройки в виде проекта или отредактировать файл “ICSP.CFG”. Чтобы изменить текущую или добавить новую конфигурацию в файл “ICSP.CFG” нужно в любом текстовом редакторе изменить соответствующую строку в конце этого файла. Более подробно о переназначении выводов и работе переходника TSH-ICSP описано в файле “ICSP.CFG”, который находится в папке, куда устанавливалась программа.

Общие замечания и рекомендации

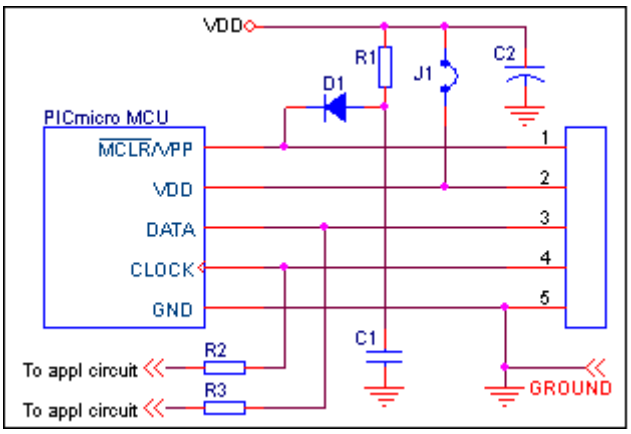
Перед подключением разъема для внутрисхемного программирования, ОБЯЗАТЕЛЬНО СОЕДИНИТЬ общий вывод платы или корпус устройства с корпусом компьютера или программатора!!! В первую очередь это касается устройств работающих от внешних импульсных иточников питания. При отсутствии надежного контакта или при неправильном заземлении, разность потенциалов будет приложена через выводы панельки к ключам программатора. Как правило, такие программаторы уже не ремонтопригодны.

- Формирователь напряжения питания в программаторе обеспечивает ток не более 80mA. При внутрисхемном программировании, с учетом емкостей по

питанию, ток потребляемый программируемым устройством не должен превышать 50mA. Если устройство потребляет больший ток, то необходимо использовать внешний источник питания. При этом подавать питание одновременно этого источника и программатора НЕЛЬЗЯ.

- Программатор формирует уровни сигналов в соответствии с заданными в программе значениями напряжения питания. Поэтому, при питании устройства от внешнего источника питания, необходимо установить значения напряжений (Vcc min, max, nom), формируемые программатором, в соответствии с напряжением этого источника.
- Программируемое устройство может иметь большие емкости в цепях питания. Программатор позволяет работать с емкостями по питанию до 500mkF, при условии, что устройство потребляет ток не более 20..30mA. Для этого в параметрах микросхемы необходимо изменить ячейку $18 - длительность задержки при включении питания. Этот параметр определяет время, в течение которого защита в программаторе находится в выключенном состоянии, обеспечивая заряд емкости максимальным током. Значение устанавливается из расчета 1ms на ~5-10mkF.
- Программатор формирует “землю” и логический ноль с помощью полевых транзисторов с очень малым сопротивлением канала. Поэтому выводы микросхемы, которые используются при программировании, не должны подключаться напрямую к цепям питания или выходам других микросхем, которые находятся в активном состоянии. Выходы этих микросхем должны быть отключены перемычками на время программирования или переведены в третье состояние с помощью дополнительных сигналов

Рекомендуемая схема подключения PIC контроллеров



- Перемычку J1 можно не устанавливать, если ток потребляемый схемой не превышает 50mA.
- При отсутствии диода D1 номинал резистора R1 должен быть не менее 20kOm.
- Вместо диода D1, резисторов R2 и R3 можно установить перемычки и размыкать их на время программирования.
- Номиналы резисторов R2 и R3 (или сопротивление нагрузки, подключенной к выводам DATA или CLOCK) должны быть не менее 1kOm при работе через TSH-ICSP и 50kOm для сигналов с панельки программатора.
- Резисторы R2 и R3 можно не устанавливать, если нагрузка, подключенная к выводам DATA или CLOCK подтянута к VDD.

Рис.10

Типовые ошибки и возможные проблемы.

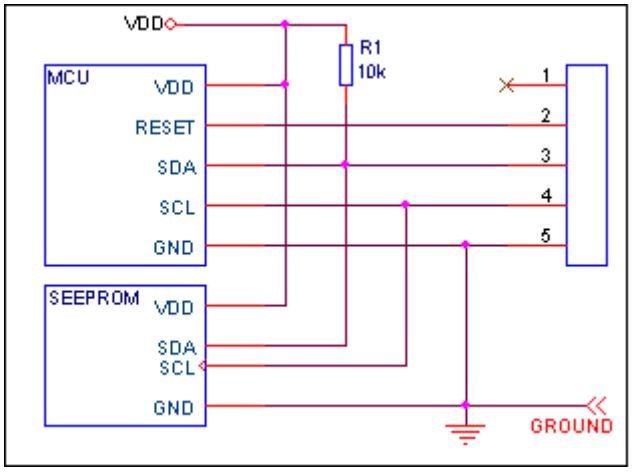
Многие PIC контроллеры (Pic12F*, Pic16F*) для входа в режим программирования требуют сначала подачу высокого напряжения на вывод MCLR и только потом напряжение питания. Некоторые из них, особенно при работе от внутреннего генератора, могут запускаться при подаче напряжения на обычные выводы микросхемы. Поэтому схема должна быть спроектирована с

учетом того, чтобы не допустить попадания Vpp в цепи питания или на другие выводы микросхемы.

Если устройство имеет большую емкость по питанию, и при этом потребляет незначительный ток, то во время работы, когда программатор отключает питание между циклами, емкость может не успеть разрядиться, что может препятствовать последующему входу в режим программирования. В этом случае можно установить дополнительный резистор (порядка 1kOm) между VDD и GND, непосредственно на разъеме для внутрисхемного программирования.

Если для сброса контроллера используется супервизор без токоограничивающего резистора, то напряжение программирования (+12..13v) окажется приложенным к его выходу, что приведет к срабатыванию защиты в программаторе и может повредить микросхему.

Рекомендуемая схема подключения Serial EEPROM 24xxx



- Если ток потребляемый схемой не превышает 50mA, то можно запитать схему от программатора.
- В устройстве на шине I2C может быть подключено несколько микросхем. Доступ к каждой микросхеме осуществляется по адресу, установить который можно в параметрах микросхемы.
- Процессор в подключаемом устройстве не должен оказывать шунтирующего влияния на сигналы программатора. Для этого используется дополнительный сигнал, который подключается ко входу RESET микроконтроллера и запрещает его работу на время программирования микросхемы. Уровень этого сигнала зависит от типа процессора и задается в параметрах микросхемы в положении "Начальное состояние".

Рис.11

Типовые ошибки и возможные проблемы.

Не подключен сигнал RESET и процессор находится в активном состоянии. Если при работе с Serial EEPROM, процессор попытается обратиться к микросхеме одновременно с программатором, то между ними возникнет конфликт. В результате в программаторе может сработать защита по питанию, но в любом случае, микросхема будет считана или записана с ошибками. Чтобы избежать этого один из выводов панельки нужно использовать в качестве сигнала сброса для процессора.

Тема 2.4. Программирование микроконтроллеров

1.Программирование в машинных кодах. Подробный разбор файлов проекта и разбор содержимого файла прошивки. Редактирование кодов команд в файле прошивки.

Любая программа для ЭВМ — системная или прикладная — воспринимается (распознается) процессором только в том случае, если она состоит из специальных команд, коды которых известны процессору определенного типа. Команды записаны в памяти компьютера в специальном формате. Каждая

команда состоит из операционной и адресной частей. В первой из них находится позиционный двоичный код, определяющий требуемое от процессора действие (сложение, вычитание и т.д.). Во второй – адресной части команды, также в виде двоичного позиционного кода, находятся адреса данных (операндов), над которыми это действие необходимо выполнить, либо сами операнды. В вольном переводе на русский язык некоторую команду можно, например, интерпретировать так: сложить два числа, находящиеся в памяти по адресам 100 и 120.

Разные типы ЭВМ имеют отличные друг от друга способы кодировки команд. Так, на персональных IBM-совместимых компьютерах некоторая команда сложения в двоичном коде может иметь вид: 0000001111000011D или в шестнадцатиричном коде 03C3H . А на «древних» компьютерах типа М-220 команда сложения двух чисел могла выглядеть так:

001 00000001100100 00000001111000 00000001111011.

Поэтому программа в кодах компьютера (машинных кодах) является машинно-зависимой и непереносимой, т.е. подготовленная для компьютера одного типа, она не сможет выполняться на других. Этот факт определяет основной недостаток программирования в машинных кодах.

Вторым недостатком программирования в кодах является сильное дробление программы. Дело в том, что логически команды процессора достаточно примитивны и обуславливают выполнение простейших операций. Так, программирование несложной формулы x=(a+b) (c+d) требовало задания серии команд типа:

— сложить a и b, промежуточный результат записать в ,
— сложить c и d, промежуточный результат записать в ,
— умножить на , результат записать в x.

При программировании в кодах визуально каждая программа состояла из большого количества команд-строк, похожих на приведенные выше двоичные коды. Это определяло третий недостаток программирования в кодах — затрудненную читаемость программы и, как следствие, сложность исправления (отладки) или доработки программы.

Однако программированию в кодах присущи и значительные плюсы. Программист управляет всеми ресурсами компьютера, полностью контролирует текущее состояние ЭВМ, выбирает наиболее оптимальный код команды. Самые короткие по объему и наиболее быстрые по выполнению программы или их фрагменты разрабатываются и сегодня в кодах. Для облегчения наглядности программы в кодах разработаны специальные символические языки — ассемблеры. В них каждой команде компьютера сопоставляется определенный символьный код, являющийся сокращением «родных» для человека слов. Специальная программа (она также называется ассемблером) переводит (транслирует)«непонятную» для компьютера (но более понятную для человека) символьную строку в коды компьютера. Так, приведенные выше коды

команд сложения на ассемблере могли выглядеть так: ADD AX, BX (сложить числа из регистров AX и BX и результат запомнить в AX). При программировании на ассемблере программист может оперировать не с адресами памяти, в которых хранятся данные, а с их символическим представлением. Например, вначале ассемблеру специальной инструкцией сообщается, что по такому-то адресу хранится число, названное для программиста как . Далее программист не задумывается над тем, по какому адресу находится соответствующее число, но просто использует его имя .

Ассемблер является машинно-зависимым языком программирования, так как его инструкции соответствуют кодам команд компьютера. Поэтому ассемблерная программа может выполняться только на тех ЭВМ, для которых она разрабатывалась. Кроме того, для работы на ассемблере требуется детальное знание особенностей конкретной ЭВМ.

2.Приемы программирования. Этапы программирования. Постановка задачи. Анализ принципиальной схемы. Разработка алгоритма программы. Операции начальной настройки. Операции, составляющие тело цикла.

Программирование (programming) - теоретическая и практическая деятельность, связанная с созданием программ. Решение задач на компьютере включает в себя следующие основные этапы, часть из которых осуществляется без участия компьютера.

1. Постановка задачи:
• сбор информации о задаче;
• формулировка условия задачи;
• определение конечных целей решения задачи;
• определение формы выдачи результатов;
• описание данных (их типов, диапазонов величин, структуры и т. п.).
2. Анализ и исследование задачи, модели:
• анализ существующих аналогов;
• анализ технических и программных средств;
• разработка математической модели;
• разработка структур данных.
3. Разработка алгоритма:
• выбор метода проектирования алгоритма;
• выбор формы записи алгоритма (блок-схемы, псевдокод и др.);
• выбор тестов и метода тестирования;
• проектирование алгоритма.
4. Программирование:
• выбор языка программирования;
• уточнение способов организации данных;
• запись алгоритма на выбранном языке
программирования.
5. Тестирование и отладка:
• синтаксическая отладка;

• отладка семантики и логической структуры;
• тестовые расчеты и анализ результатов тестирования;
• совершенствование программы.
Анализ результатов решения задачи и уточнение в случае необходимости математической модели с повторным выполнением этапов 2-5.
7. Сопровождение программы:
• доработка программы для решения конкретных задач;
• составление документации к решенной задаче, к математической модели, к алгоритму, к программе, к набору тестов, к использованию.

Понятие об алгоритмах

Алгоритм - система команд, определяющая процесс преобразования исходных данных в результат решения поставленной задачи

3. Программа на языке Ассемблер. Алгоритм создания программы. Форма записи. Директивы. Операторы. Описание программы(листинг)

Определение. ***Алгоритм*** *— это последовательность действий, которую должен произвести наш микроконтроллер, чтобы достичь требуемого результата. Для простых задач алгоритм можно просто описать словами. Для более сложных задач алгоритм рисуется в графическом виде.*

В нашем случае алгоритм таков: После операций начальной настройки портов микроконтроллер должен войти в непрерывный цикл, в процессе которого он должен опрашивать вход, подключенный к нашей кнопке, и в зависимости от ее состояния управлять светодиодом. Опишем это подробнее.

Операции начальной настройки:

- установить начальное значение для вершины стека микроконтроллера;
- настроить порт B на вывод информации;
- подать на выход PB.0 сигнал логической единицы (потушить светодиод);
- сконфигурировать порт D на ввод;
- включить внутренние нагрузочные резисторы порта D.

Операции, составляющее тело цикла:

- прочитать состояние младшего разряда порта PD (PD.0);
- если значение этого разряда равно единице, выключить светодиод;
- если значение разряда PD.0 равно нулю, включить светодиод;
- перейти на начало цикла.

3 Программа на Ассемблере

Для создания программ мы используем версию Ассемблера, предложенную разработчиком микроконтроллеров AVR — фирмой Atmel. А также воспользуемся программным комплексом «AVR Studio», разработанным той же фирмой и предназначенным для создания, редактирования, трансляции и отладки программ для AVR на Ассемблере. Подробнее программа «AVR Studio» описана в предыдущей главе.

Главная **цель** лабораторных работ – научиться создавать программы. Изучение языка программирования будет происходить следующим образом. В методическом пособии приводится примерный вариант текста программы для каждой конкретной задачи, а затем подробно описываются все его элементы и объясняется, как программа работает. Студент должен составить свой вариант программы.

Текст возможного варианта программы, реализующий поставленную выше задачу, приведен в **листинге 1.1**. Прежде чем приступить к описанию данного примера, необходимо напомнить несколько общих понятий о языке Ассемблер.

Программа на Ассемблере представляет собой набор команд и комментариев (иногда команды называют инструкциями). Каждая команда занимает одну отдельную строку. Их допускается перемежать пустыми строками. Команда обязательно содержит оператор, который выглядит как имя выполняемой операции.

Некоторые команды состоят только из одного оператора. Другие же команды имеют один или два операнда (параметра). Операнды записываются в той же строке сразу после оператора, через пробел. Если операндов два, их записывают через запятую. Так, в строке 6 нашей программы записана команда загрузки константы в регистр общего назначения. Она состоит из оператора ldi и двух операндов temp и RAMEND.

В случае необходимости перед командой допускается ставить так называемую **метку**. Она состоит из имени метки, заканчивающимся двоеточием. Метка служит для именования данной строки программы. Затем это имя используется в различных командах для обращения к помеченной строке.

При выборе имени метки необходимо соблюдать следующие **правила:**

- имя должно состоять из одного слова, содержащего только латинские буквы и цифры;
- допускается также применять символ подчеркивания;
- первым символом метки обязательно должна быть буква или символ подчеркивания.

Строка 16 нашей программы содержит метку с именем main. Метка не обязательно должна стоять в строке с оператором. Допускается ставить метку в любой строке программы. Кроме команд и меток, программа содержит комментарии.

Определение. ***Комментарий*** *— это специальная запись в теле программы, предназначенная для человека. Компьютер в процессе трансляции программы игнорирует все комментарии. Комментарий может занимать отдельную строку, а может стоять в той же строке, что и команда. Начинается комментарий с символа «точка с запятой». Все, что находится после точки с запятой до конца текущей строки программы, считается комментарием.*

Если в уже готовой программе вы поставите точку с запятой в начале строки перед какой-либо командой, то данная строка для транслятора как бы исчезнет. С этого момента транслятор будет считать всю эту строку комментарием. Таким образом, можно временно отключать отдельные строки программы в процессе отладки (то есть при поиске ошибок в программе).

Кроме операторов, в языке Ассемблер применяются **псевдооператоры,** или **директивы.** Если оператор — это некий эквивалент реальной команды микроконтроллера и в процессе трансляции заменяется соответствующим машинным кодом, который помещается в файл результата трансляции, то директива, хотя по форме и напоминает оператор, но не является командой процессора.

Определение. ***Директивы*** *— это специальные вспомогательные команды для транслятора, определяющие режимы трансляции и реализующие различные вспомогательные функции.*

Далее из конкретных примеров можно понять, о чем идет речь. В данной конкретной версии Ассемблера директивы выделяются особым образом. Имя каждой директивы начинается с точки. (Смотри **листинг 1.1, строки с 1 по 5).**

При написании программ на Ассемблере принято соблюдать **особую форму записи:**

- программа записывается в несколько колонок (см. **листинг 1.1);**
- аналогичные элементы разных команд принято размещать друг под другом;
- самая первая (левая) колонка зарезервирована для меток;
- если метка отсутствует, место в колонке пустует;
- следующая колонка предназначена для записи операторов;
- затем идет колонка для операндов;
- оставшееся пространство (крайняя колонка справа) предназначено для комментариев.

В некоторых случаях, **например,** когда текст команды очень длинный, допускается нарушать этот порядок. Но, по возможности, нужно оформлять программу именно так. Оформленная подобным образом программа более наглядна и гораздо лучше читается.

4.Программа на языке Си. Программная среда Code Vision AVR. Мастер Программ и его свойства. Настройка портов. Работа программа на языке Си. Описание. Комментарии.

Минимальная программа на Си может быть такой: **main(){}.**

Эта программа не делает ничего полезного – но это уже программа и она показывает, что в программе на языке Си должна быть главная функция main – обязательно. Реальные программы на Си конечно больше.

Регистры МК в программе на Си имеют названия, как и в оригинальной технической документации фирмы ATMEL AVR Data Sheets (ДШ) и так как числа в большинстве из них можно менять – для программы регистры являются по сути переменными.

Чтобы поместить число в переменную (в регистр), в Си есть **оператор присваивания**. Это знак = (называемый в математике "равно"). В Си этот знак не означает равенство. Знак = в Си означает вычислить результат того, что справа от оператора присваивания и поместить этот результат в переменную, находящуюся левее оператора присваивания. Ниже приведены примеры команд на Си, использующие оператор присваивания.

PORTB = PINB + 57; /*Взять (прочитать, считать) значение переменной (регистра) PINB, затем прибавить к нему число 57 и поместить результат в переменную PORTB */

PORTB&=0x5A; /*Прочитать значение переменной PORTB, затем выполнить "поразрядное (побитное) логическое И" между прочитанным значением и числом 0x5A и записать результат в переменную PORTB */

PORTB = 0x23; /*Не читая содержимое переменной PORTB присвоить ей значение 0x23 */

Вместо & "И" могут быть и другие побитные логические операции: | "ИЛИ", ^ "Исключающее ИЛИ", ~ "инвертирование битов" и арифметические операции: + - * / %.

Запомните! Результатом поразрядных (побитных) логических операций (& | ^ ~) является число, которое может быть интерпретировано компилятором как "истина", если оно не ноль, и "ложь", если число ноль.

Целые числа в компиляторе могут быть записаны:

- в десятичной форме: 1234;
- в двоичной форме с префиксом 0b: 0b101001;
- в шестнадцатеричной форме с префиксом 0x: 0x5A;
- в восьмеричной форме с префиксом 0: 0775.

С оператором присваивания используются вот такие сокращения:

Длинная запись	Смысл	Сокращается до
x = x + 1;	добавить 1	x++; или ++x;
x = x - 1;	вычесть 1	x--; или --x;
x = x + y;	прибавить y	x += y;
x = x - y;	вычесть y	x -= y;
x = x * y;	умножить на y	x *= y;
x = x / y;	поделить на y	x /= y;
x = x % y;	остаток от деления	x %= y;
x--;	вычесть 1	x -= 1;
x++;	добавить 1	x += 1;

Есть в Си операции, которые изменяют значение переменной и без оператора присваивания:

PORTA++; /* Взять значение переменной PORTA, добавить к ней 1 и записать результат обратно в PORTA –инкрементировать регистр PORTA */

PORTC--; /* Эта строчка на Си означает обратное действие – декрементировать значение регистра PORTC */

Инкремент и декремент удобно использовать для изменения значения различных переменных-счетчиков. Важно помнить, что они имеют очень низкий приоритет. Поэтому, чтобы быть уверенным в порядке выполнения, желательно писать их отдельной строчкой программы.

Когда инкремент или декремент используется в выражении, то важно, где стоят два знака + или – (перед переменной или после переменной):

A=4; B=7;

A=B++; /* Взять значение переменной B, присвоить его переменной A, затем добавить 1 к переменной B и сохранить результат в B. Теперь A будет содержать число 7, B будет содержать число 8 */

A=4; B=7;

A=++B; /* Взять значение переменной B, затем добавить к нему 1 и сохранить результат в B и этот же результат присвоить переменной A. Теперь A будет содержать число 8 и B будет содержать число 8 */

Арифметические операции в Си:

x+y //сложение
x-y // вычитание
x * y // умножение
x / y /* деление. Если числа целые, результат – целое число с отброшенной дробной частью – не округленное! Т.е. если в результате деления на калькуляторе получается 6.23411 или 6.94, то результат будет просто целое число 6. Если числа с плавающей точкой, то есть float или double и записываются с точкой и числом после точки, то и результат будет число с плавающей точкой */
x % y // вычислить остаток от деления нацело
Примеры:
5 / 2 // даст 2
5 % 2 // даст 1
75 / 29 // даст 2
75 % 29 // даст 17

Операторы сравнения (или отношения) используются для сравнения переменных, чисел (констант) и выражений:

x < y // x меньше y x > y // больше x <= y // меньше или равно x >= y // больше или равно x == y // равно x != y // не равно

Результат выполнения этих операторов: "истина" это "1" (точнее "не ноль"), "ложно" это "0". Значения, хранимые в переменных (в регистрах) x и y, не изменяются!

Логические операции:

|| // "ИЛИ"
&& // "И"
! // "НЕ"
!(истина) // дает "ложь"
!(ложь) // дает "истина"

В результате логической операции вы получаете не число, а логическое значение "истина" или "ложь". Для логических операций && и || берутся результаты выражений слева и справа от знака операции, преобразованные в "истину" или "ложь", и определяется логический результат операции. Компилятор результат "истина" превращает в 1, а "ложь" в 0.

Ходовые конструкции на Си (в компиляторе CVAVR заготовки этих конструкций находятся под ярлыком "Code Templates" слева вверху. Вы можете выбирать нужные заготовки и вставлять их в свою программу):

if(){}else{}; идеальная конструкция, если вам нужно выполнить какую-то часть программы при наличии каких либо условий:

```
if (выражение) { /* делать этот код, если выражение "истина" – т.е. результат его вычисления не ноль */ } else { /* делать этот код, если выражение "ложь" – т.е. результат его вычисления равен нулю */ };
```

} else { это не обязательный элемент конструкции:

```
if (выражение) { /* делать этот код, если выражение "истина" – т.е. результат его вычисления не ноль */ };
```

while(){}; условный цикл – используйте, если вам нужно выполнять какой-то код программы, пока выполняется (существует, справедливо, не ноль) некоторое условие:

```
while (выражение) { /* делать этот код, если выражение "истина" – т.е. результат его вычисления не ноль. Пока выполняется этот код, выражение не проверяется на истинность. После выполнения кода происходит переход к строке while, чтобы снова проверять истинность выражения */
};
```

Цикл while имеет вариант do – while, при котором код в { } выполняется по меньшей мере один раз независимо от истинности условия в скобках:

```
do { /* сделать этот код один раз, затем, если выражение есть "истина" –
т.е. результат его вычисления не ноль – опять делать код с начала, и так
до тех пор, пока выражение "истина" */
} while (выражение);
```

for(;;){}; – этот цикл позволяет выполнить часть программы нужное число раз:

```
char i; /* объявление переменной для for. Это обычная переменная и, зна-
чит, может иметь любое  допустимое имя по вашему желанию */
for (i=5;i<20;i+=4) {  /*  код цикла for. i=5 – это начальное выражение.
Число 5 просто для примера, может быть таким, как позволяет объявле-
ние переменной i, в нашем случае от 0 до 255. i<20 – контрольное выра-
жение. Может быть с разными операторами отношения, важно лишь,
чтобы по ходу цикла оно становилось когда-то "ложью" – иначе цикл "за-
циклится", т.е. никогда не кончится. i+=4 – счетчик. Обычно это i++, т.е. к
переменной добавляется 1 каждый "прогон" цикла. Но может быть таким,
какое вам требуется, важно лишь достижение когда-либо условия, огово-
ренного выше! Иначе цикл станет бесконечным. Код цикла for будет пер-
вый раз выполнен для i=5, затем по выражению i+=4 i станет 9. Теперь
будет проверено контрольное выражение i<20 и так как 9<20 код цикла
for будет выполнен еще раз. Так будет происходить до тех пор, пока кон-
трольное выражение "истинно". Когда оно станет "ложно" цикл for за-
кончится и программа пойдет дальше.  */
            };
```

Начальным условием может быть любое допустимое в Си выражение, результатом которого является целое число. Контрольное выражение определяет, до каких пор будет выполняться цикл. Счетчик показывает, как изменяется начальное выражение перед каждым новым выполнением цикла.

Циклы for(;;) и while() часто используют вот так:

```
while(1);
for (;;);
/* Так написанные эти циклы означают: МК выполнять эту строчку пока
есть питание, нет сброса и нет прерывания. Когда возникает прерывание,
программа переходит на обработчик прерывания и (если в обработчике нет
перехода в другое место программы) по завершении кода обработчика
опять возвращается в такой цикл */
```

switch(){}; – оператор множественного выбора, позволяет сделать выбор из нескольких вариантов.

```
switch (выражение) { case 7: /* этот код будет выполняться, если результат вычисления выражения равен числу 7. На этом работа оператора switch закончится */ break;
case -28: /* этот код будет выполняться, если результат вычисления выражения равен отрицательному числу -28. На этом работа оператора switch закончится */ break;
case 'G': /* этот код будет выполняться, если результат вычисления выражения равен числу, соответствующему символу G в таблице ASCII. На этом работа оператора switch закончится */ break; default: /* этот код будет выполняться, если результат вычисления выражения не равен ни 7, ни -28, ни 'G'. А так же после выполнения кода, не имеющего в конце  break. На этом работа оператора switch закончится */ };
/* switch закончен - выполняется дальнейший код программы */
```

case может быть столько, сколько вам нужно. Чтобы программа работала быстрее, старайтесь наиболее вероятные варианты располагать выше!

default - не обязателен. Его можно расположить и не в конце.

break; - если его не использовать, то, найдя нужный вариант, программа будет выполнять и следующие ниже условия case.

goto – оператор безусловного (немедленного) перехода.

```
mesto_5: /* сюда мы попадем после выполнения строки программы goto mesto_5   */
goto mesto_1; /* перейти в то место программы, где в начале строки написано mesto_1:     */
goto mesto_5; /* перейти в то место программы, где в начале строки написано mesto_5:     */
mesto_1: /* сюда мы попадем после выполнения строки программы goto mesto_1   */
```

goto существует наверно во всех языках и в ассемблере в том числе. Используйте его с осторожностью! Например: если вы покинете функцию-обработчик прерывания по goto, не завершив ее, то не произойдет автоматического включения прерываний глобально – т.е. не установится бит I в регистре SREG, Этот бит устанавливается автоматически после полного выполнения функции обработки прерывания и "естественного" выхода из неё.

Тема 2.5. Среда разработки AVR Studio

1.Детальный обзор программы AVR Studio. Изучение режима отладки программы

AVR Studio – это интегрированная отладочная среда разработки приложений (IDE) для микроконтроллеров AVR компании Atmel (рис. 1). IDE AVR Studio содержит:

средства создания и управления проектом;
редактор кода на языке Ассемблер;

транслятор языка Ассемблера (Atmel AVR);
отладчик (Debugger);
программное обеспечение верхнего уровня для поддержки внутрисхемного программирования (In-System Programming, ISP) с использованием стандартных отладочных средств Atmel AVR.

Работа с AVR Studio начинается с создания проекта. Сначала необходимо указать используемый микроконтроллер (МК) и платформу, на которой будет производиться отладка программы.

Написание программы производится в окне редактора текста программы. Для использования символических имен регистров специального назначения вместо их адресов необходимо подключить (директива .include) к проекту файл определения регистров специального назначения (например, m16def.inc для ATmega16). Включаемые файлы входят в прикладное программное обеспечение AVR Studio и при инсталляции помещаются в папку Appnotes в директории установки AVR Studio.

Написание программы в AVR Studio производится на языке Ассемблер. Последние версии AVR Studio содержат AVR Ассемблер второй версии, который в дополнение к стандартному Ассемблеру поддерживает новые директивы Ассемблера, Си - подобные директивы препроцессора, создание переменных определенного типа.

В результате трансляции создается выходной файл в формате HEX (расширение .hex). Если не выдается сообщений об ошибках, можно приступать к отладке проекта.

Отладчик AVR Studio поддерживает все типы МК AVR и имеет два режима работы: режим программной симуляции и режим управления различными типами внутрисхемных эмуляторов (In-Circuit Emulators) производства фирмы Atmel. Важно отметить, что интерфейс пользователя не изменяется в зависимости от выбранного режима отладки.

Отладочная среда поддерживает выполнение программ как в виде ассемблерного текста, так и в виде исходного текста языка Си, загруженного в объектном коде.

Управление отладкой производится командами меню DEBUG либо соответствующими иконками на панели инструментов AVR Studio.

Пользователь может выполнять программу полностью в пошаговом режиме, трассируя блоки функций или выполняя программу до того места, где стоит курсор. В дополнение можно определять неограниченное число точек останова, каждая из которых может быть включена или выключена. Точки останова сохраняются между сессиями работы.

В AVR Studio для отладки программы предусмотрены две команды пошагового режима: Step Over и Trace into. Разница между ними в том, что при выполнении команды Step Over выполнение подпрограмм происходит до пол-

ного окончания без отображения процесса выполнения. К командам шагового режима также относится команда Auto Step.

С помощью команд пошагового режима можно проследить изменения значений в переменных, регистрах ввода/вывода, памяти и регистрового файла. Для этого предназначены раздел I/O рабочей области AVR Studio (см. рис. 1), окно Watch (меню Debug \ Quick watch).

Интегрированная среда разработки AVR Studio также поддерживает просмотр (меню View / Memory) ячеек памяти программ, памяти данных, EEPROM и регистров портов ввода/вывода в ходе исполнения. Выпадающее меню диалогового окна позволяет выбрать один из четырех массивов ячеек памяти: Data, IO, EEPROM, Program Memory. Для одновременного просмотра нескольких областей окно Memory может быть открыто несколько раз. Информация в диалоговом окне может быть представлена в виде байтов или в виде слов в шестнадцатеричной системе счисления, а также в виде ASCII-символов.

В процессе отладки пользователь может инициализировать внутреннее ОЗУ, или EEPROM МК (например, данными, содержащимися в полученном при трансляции файле .eep), или сохранить содержимое ОЗУ и

EEPROM в виде файлов в формате Intel Hex (меню File -> Up/Download Memory).

Помимо шагового режима, возможна отладка программы с использованием точек останова (меню Breakpoints -> Toggle Breakpoint). Командой Start Debugging запускается исполнение программы. Программа будет выполняться до остановки пользователем или до обнаружения точки останова.

Для наблюдения за работой программы можно открыть несколько окон, отображающих состояние различных узлов МК (см. рис. 1). Окна открываются нажатием соответствующих кнопок на панели инструментов или при выборе соответствующего пункта меню View. Если в процессе выполнения программы в очередном цикле значение какого-либо регистра изменится, то этот регистр будет выделен красным цветом. При этом если в следующем цикле значение регистра останется прежним, то цветовое выделение будет снято. Такое же цветовое выделение реализовано в окнах устройств ввода/вывода, памяти и переменных.

Состояние встроенных периферийных устройств МК, а также состояния программного счетчика, указателя стека, содержимого регистра статуса SREG и индексных регистров X, Y и Z отображено в окне I/O View. В этом окне отражаются все функциональные блоки микроконтроллера. Любой блок может быть раскрыт нажатием на его значок. При раскрытии блока в окне отражаются адреса и состояния всех его регистров и отдельных, доступных для модификации, битов. Каждый доступный для модификации бит может быть установлен или сброшен как программой по ходу ее исполнения, так и пользователем вручную (указав курсором мыши нужный бит и щелкнув левой

кнопкой мыши, пользователь может изменить значение бита на обратное), а в режиме программной симуляции это является способом имитации входного воздействия на МК.

Тема 2.6 Отладка программ

1.Основные виды отладки и их возможности. Этапы процесса отладки программ

Отладка ПС - это деятельность, направленная на обнаружение и исправление ошибок в ПС. Тестирование ПС - это процесс выполнения программ ПС на некотором наборе данных, для которого заранее известен результат применения или известны правила поведения этих программ. Указанный набор данных называется тестовым или просто тестом. Таким образом, отладку можно представить в виде многократного повторения трех процессов: тестирования, в результате которого может быть констатировано наличие в ПС ошибки, поиска места ошибки в программах и документации ПС и редактирования программ и документации с целью устранения обнаруженной ошибки. Другими словами:

Отладка = Тестирование + Поиск ошибок + Редактирование.

Принципы и виды отладки программного средства

Успех отладки ПС в значительной степени предопределяет рациональная организация тестирования. Тестирование не может доказать правильность ПС, в лучшем случае оно может продемонстрировать наличие в нем ошибки. Поэтому возникает две задачи. Первая задача: подготовить такой набор тестов, чтобы обнаружить в ПС по возможности большее число ошибок. Однако чем дольше продолжается процесс тестирования (и отладки в целом), тем большей становится стоимость ПС. Отсюда вторая задача: определить момент окончания отладки ПС (или отдельной его компоненты). Признаком возможности окончания отладки является полнота охвата пропущенными через ПС тестами (т.е. тестами, к которым применено ПС) множества различных ситуаций, возникающих при выполнении программ ПС, и относительно редкое проявление ошибок в ПС на последнем отрезке процесса тестирования. Последнее определяется в соответствии с требуемой степенью надежности ПС, указанной в спецификации его качества.

Для оптимизации набора тестов, т.е. для подготовки такого набора тестов, который позволял бы при заданном их числе (или при заданном интервале времени, отведенном на тестирование) обнаруживать большее число ошибок в ПС, необходимо, во-первых, заранее планировать этот набор и, во-вторых, использовать рациональную стратегию планирования (проектирования) тестов. Возможны разные подходы к выработке стратегии проектирования тестов, которые можно условно графически разместить между следующими двумя крайними подходами. Левый крайний подход заключается в том, что тесты проектируются только на основании изучения спецификаций ПС (внешнего описания, описания архитектуры и спецификации модулей).

Строение модулей при этом никак не учитывается, т.е. они рассматриваются как черные ящики. Фактически такой подход требует полного перебора всех наборов входных данных, так как в противном случае некоторые участки программ ПС могут не работать при пропуске любого теста, а это значит, что содержащиеся в них ошибки не будут проявляться. Однако тестирование ПС полным множеством наборов входных данных практически неосуществимо. Правый крайний подход заключается в том, что тесты проектируются на основании изучения текстов программ с целью протестировать все пути выполнения каждой программ ПС. Если принять во внимание наличие в программах циклов с переменным числом повторений, то различных путей выполнения программ ПС может оказаться также чрезвычайно много, так что их тестирование также будет практически неосуществимо.

Оптимальная стратегия проектирования тестов расположена внутри интервала между этими крайними подходами, но ближе к левому краю. Она включает проектирование значительной части тестов по спецификациям, но она требует также проектирования некоторых тестов и по текстам программ. При этом в первом случае эта стратегия базируется на принципах:

• на каждую используемую функцию или возможность - хотя бы один тест,
• на каждую область и на каждую границу изменения какой-либо входной величины - хотя бы один тест,
• на каждую особую (исключительную) ситуацию, указанную в спецификациях, - хотя бы один тест.

Во втором случае эта стратегия базируется на принципе: каждая команда каждой программы ПС должна проработать хотя бы на одном тесте.

В нашей стране различаются два основных вида отладки (включая тестирование): автономную и комплексную отладку ПС.

Автономная отладка ПС означает последовательное раздельное тестирование различных частей программ, входящих в ПС, с поиском и исправлением в них фиксируемых при тестировании ошибок. Она фактически включает отладку каждого программного модуля и отладку сопряжения модулей. В процессе автономной отладки ПС производится наращивание тестируемой программы отлаженными модулями: при переходе к отладке следующего модуля в его программное окружение добавляется последний отлаженный модуль. Такой процесс наращивания программного окружения отлаженными модулями называется интеграцией программы.

При восходящем тестировании это окружение будет содержать только один отладочный модуль (кроме случая, когда отлаживается последний модуль отлаживаемой программы), который будет головным в тестируемой программе. Такой отладочный модуль называют ведущим (или драйвером). Ведущий отладочный модуль подготавливает информационную среду для тестирования отлаживаемого модуля (т. е. формирует ее состояние, требуемое для тестирования этого модуля, в частности, путем ввода некоторых тесто-

вых данных), осуществляет обращение к отлаживаемому модулю и после окончания его работы выдает необходимые сообщения.

При нисходящем тестировании окружение отлаживаемого модуля в качестве отладочных модулей содержит отладочные имитаторы (заглушки) некоторых еще не отлаженных модулей. К таким модулям относятся, прежде всего, все модули, к которым может обращаться отлаживаемый модуль, а также еще не отлаженные модули, к которым могут обращаться уже отлаженные модули (включенные в это окружение). Некоторые из этих имитаторов при отладке одного модуля могут изменяться для разных тестов.

На практике в окружении отлаживаемого модуля могут содержаться отладочные модули обоих типов, если используется смешанная стратегия тестирования. Это связано с тем, что как восходящее, так и нисходящее тестирование имеет свои достоинства и свои недостатки.

К достоинствам восходящего тестирования относятся:

• простота подготовки тестов,

• возможность полной реализации плана тестирования модуля.

Это связано с тем, что тестовое состояние информационной среды готовится непосредственно перед обращением к отлаживаемому модулю (ведущим отладочным модулем).

Недостатками восходящего тестирования являются следующие его особенности:

• тестовые данные готовятся, как правило, не в той форме, которая рассчитана на пользователя (кроме случая, когда отлаживается последний, головной, модуль отлаживаемой программ);

• большой объем отладочного программирования (при отладке одного модуля приходится составлять много ведущих отладочных модулей, формирующих подходящее состояние информационной среды для разных тестов);

• необходимость специального тестирования сопряжения модулей.

• К достоинствам нисходящего тестирования относятся следующие его особенности:

• большинство тестов готовится в форме, рассчитанной на пользователя;

• во многих случаях относительно небольшой объем отладочного программирования (имитаторы модулей, как правило, весьма просты и каждый пригоден для большого числа, нередко - для всех, тестов);

• отпадает необходимость тестирования сопряжения модулей.

Недостатком нисходящего тестирования является то, что тестовое состояние информационной среды перед обращением к отлаживаемому модулю готовится косвенно - оно является результатом применения уже отлаженных модулей к тестовым данным или данным, выдаваемым имитаторами. Это, во-первых, затрудняет подготовку тестов и требует высокой квалификации тес-

товика (разработчика тестов), а во-вторых, делает затруднительным или даже невозможным реализацию полного плана тестирования отлаживаемого модуля. Указанный недостаток иногда вынуждает разработчиков применять восходящее тестирование даже в случае нисходящей разработки. Однако чаще применяют некоторые модификации нисходящего тестирования, либо некоторую комбинацию нисходящего и восходящего тестирования.

Часто применяют также комбинацию восходящего и нисходящего тестирования, которую называют методом сандвича. Сущность этого метода заключается в одновременном осуществлении как восходящего, так и нисходящего тестирования, пока эти два процесса тестирования не встретятся на какомлибо модуле где-то в середине структуры отлаживаемой программы. Этот метод при разумном порядке тестирования позволяет воспользоваться достоинствами как восходящего, так и нисходящего тестирования, а также в значительной степени нейтрализовать их недостатки.

Автономное тестирование модуля целесообразно осуществлять в четыре последовательно выполняемых шага.

• На основании спецификации отлаживаемого модуля подготовьте тесты для каждой возможности и каждой ситуации, для каждой границы областей допустимых значений всех входных данных, для каждой области изменения данных, для каждой области недопустимых значений всех входных данных и каждого недопустимого условия.

• Проверьте текст модуля, чтобы убедиться, что каждое направление любого разветвления будет пройдено хотя бы на одном тесте. Добавьте недостающие тесты.

• Проверьте текст модуля, чтобы убедиться, что для каждого цикла существуют тесты, обеспечивающие, по крайней мере, три следующие ситуации: тело цикла не выполняется ни разу, тело цикла выполняется один раз и тело цикла выполняется максимальное число раз. Добавьте недостающие тесты.

• Проверьте текст модуля, чтобы убедиться, что существуют тесты, проверяющие чувствительность к отдельным особым значениям входных данных. Добавьте недостающие тесты.

Комплексная отладка означает тестирование ПС в целом с поиском и исправлением фиксируемых при тестировании ошибок во всех документах (включая тексты программ ПС), относящихся к ПС в целом. К таким документам относятся определение требований к ПС, спецификация качества ПС, функциональная спецификация ПС, описание архитектуры ПС и тексты программ ПС. Тестирование этих документов производится, как правило, в порядке, обратном их разработке. Исключение составляет лишь тестирование документации по применению, которая разрабатывается по внешнему описанию параллельно с разработкой текстов программ - это тестирование лучше производить после завершения тестирования внешнего описания. Тестирование при комплексной отладке представляет собой применение ПС к конкретным

данным, которые в принципе могут возникнуть у пользователя (в частности, все тесты готовятся в форме, рассчитанной на пользователя), но, возможно, в моделируемой (а не в реальной) среде. Например, некоторые недоступные при комплексной отладке устройства ввода и вывода могут быть заменены их программными имитаторами.

Тестирование архитектуры ПС. Целью тестирования является поиск несоответствия между описанием архитектуры и совокупностью программ ПС. К моменту начала тестирования архитектуры ПС должна быть уже закончена автономная отладка каждой подсистемы. Ошибки реализации архитектуры могут быть связаны, прежде всего, с взаимодействием этих подсистем, в частности, с реализацией архитектурных функций (если они есть). Поэтому хотелось бы проверить все пути взаимодействия между подсистемами ПС. При этом желательно хотя бы протестировать все цепочки выполнения подсистем без повторного вхождения последних. Если заданная архитектура представляет ПС в качестве малой системы из выделенных подсистем, то число таких цепочек будет вполне обозримо.

Тестирование внешних функций. Целью тестирования является поиск расхождений между функциональной спецификацией и совокупностью программ ПС. Несмотря на то, что все эти программы автономно уже отлажены, указанные расхождения могут быть, например, из-за несоответствия внутренних спецификаций программ и их модулей (на основании которых производилось автономное тестирование) функциональной спецификации ПС. Как правило, тестирование внешних функций производится так же, как и тестирование модулей на первом шаге, т.е. как черного ящика.

Тестирование качества ПС. Целью тестирования является поиск нарушений требований качества, сформулированных в спецификации качества ПС. Это наиболее трудный и наименее изученный вид тестирования. Ясно лишь, что далеко не каждый примитив качества ПС может быть испытан тестированием. Завершенность ПС проверяется уже при тестировании внешних функций. На данном этапе тестирование этого примитива качества может быть продолжено, если требуется получить какую-либо вероятностную оценку степени надежности ПС. Однако, методика такого тестирования еще требует своей разработки. Могут тестироваться такие примитивы качества, как точность, устойчивость, защищенность, временная эффективность, в какой-то мере - эффективность по памяти, эффективность по устройствам, расширяемость и, частично, независимость от устройств. Каждый из этих видов тестирования имеет свою специфику и заслуживает отдельного рассмотрения. Мы здесь ограничимся лишь их перечислением. Легкость применения ПС (критерий качества, включающий несколько примитивов качества) оценивается при тестировании документации по применению ПС.

Тестирование документации по применению ПС. Целью тестирования является поиск несогласованности документации по применению и совокупностью программ ПС, а также выявление неудобств, возникающих при приме-

нении ПС. Этот этап непосредственно предшествует подключению пользователя к завершению разработки ПС (тестированию определения требований к ПС и аттестации ПС), поэтому весьма важно разработчикам сначала самим воспользоваться ПС так, как это будет делать пользователь. Все тесты на этом этапе готовятся исключительно на основании только документации по применению ПС. Прежде всего, должны тестироваться возможности ПС как это делалось при тестировании внешних функций, но только на основании документации по применению. Должны быть протестированы все неясные места в документации, а также все примеры, использованные в документации. Далее тестируются наиболее трудные случаи применения ПС с целью обнаружить нарушение требований относительности легкости применения ПС.

Тестирование определения требований к ПС. Целью тестирования является выяснение, в какой мере ПС не соответствует предъявленному определению требований к нему. Особенность этого вида тестирования заключается в том, что его осуществляет организация-покупатель или организация-пользователь ПС как один из путей преодоления барьера между разработчиком и пользователем. Обычно это тестирование производится с помощью контрольных задач - типовых задач, для которых известен результат решения. В тех случаях, когда разрабатываемое ПС должно придти на смену другой версии ПС, которая решает хотя бы часть задач разрабатываемого ПС, тестирование производится путем решения общих задач с помощью как старого, так и нового ПС (с последующим сопоставлением полученных результатов). Иногда в качестве формы такого тестирования используют опытную эксплуатацию ПС - ограниченное применение нового ПС с анализом использования результатов в практической деятельности. По существу, этот вид тестирования во многом перекликается с испытанием ПС при его аттестации, но выполняется до аттестации, а иногда и вместо аттестации.