

Министерство образования Республики Беларусь

*Учреждение образования*

**Белорусский Государственный Университет Информатики и  
Радиоэлектроники**

кафедра радиоэлектронных средств

И.Н. Цырельчук

Практическое занятие №1

**Разработка микропроцессорной системы на основе  
микроконтроллера**

по курсу «Микропроцессорные системы и их применение»

Минск 2005

**Цель работы:** Изучить основные этапы проектирования и разработки цифровых устройств и систем на основе микроконтроллеров

## **1. Основные этапы разработки**

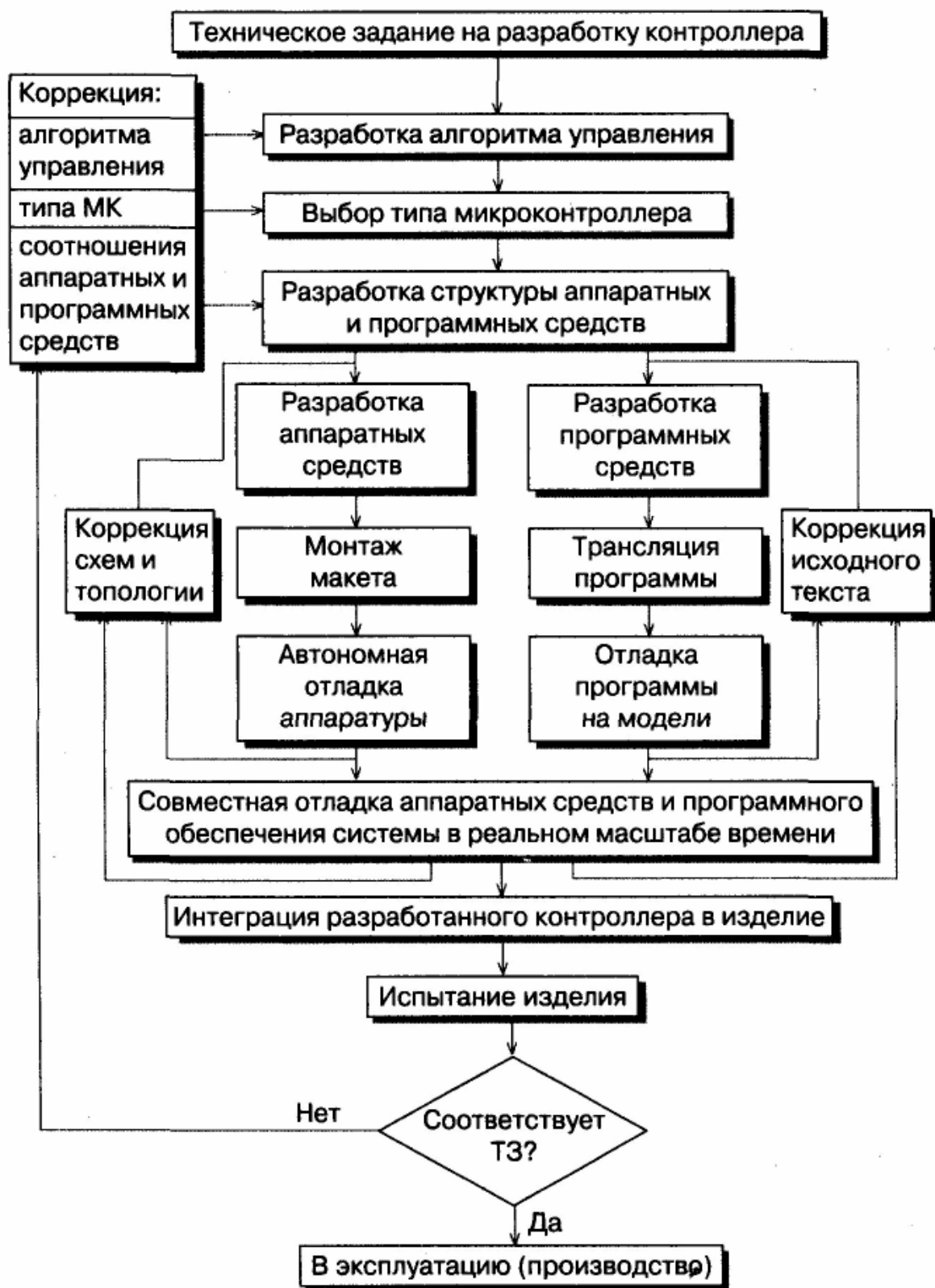
МПС на основе МК используются чаще всего в качестве встроенных систем для решения задач управления некоторым объектом. Важной особенностью данного применения является работа в реальном времени, т.е. обеспечение реакции на внешние события в течение определенного временного интервала. Такие устройства получили название контроллеров.

Технология проектирования контроллеров на базе МК полностью соответствует принципу неразрывного проектирования и отладки аппаратных и программных средств, принятому в микропроцессорной технике. Это означает, что перед разработчиком такого рода МПС стоит задача реализации полного цикла проектирования, начиная от разработки алгоритма функционирования и заканчивая комплексными испытаниями в составе изделия, а, возможно, и сопровождением при производстве. Сложившаяся к настоящему времени методология проектирования контроллеров может быть представлена так, как показано на рис. 1.

В техническом задании формулируются требования к контроллеру с точки зрения реализации определенной функции управления. Техническое задание включает в себя набор требований, который определяет, что пользователь хочет от контроллера и что разрабатываемый прибор должен делать. Техническое задание может иметь вид текстового описания, не свободного в общем случае от внутренних противоречий.

На основании требований пользователя составляется функциональная спецификация, которая определяет функции, выполняемые контроллером для пользователя после завершения проектирования, уточняя тем самым, насколько устройство соответствует предъявляемым требованиям. Она включает в себя описания форматов данных, как на входе, так и на выходе, а также внешние условия, управляющие действиями контроллера.

Функциональная спецификация и требования пользователя являются критериями оценки функционирования контроллера после завершения проектирования. Может потребоваться проведение нескольких итераций, включающих обсуждение



**Рис. 6.1.** Основные этапы разработки контроллера.

требований и функциональной спецификации с потенциальными пользователями контроллера, и соответствующую коррекцию требований и спецификации. Требования к типу используемого МК формулируются на данном этапе чаще всего в неявном виде.

Этап разработки алгоритма управления является наиболее ответственным, поскольку ошибки данного этапа обычно обнаруживаются только при испытаниях законченного изделия и приводят к необходимости дорогостоящей переработки всего устройства. Разработка алгоритма обычно сводится к выбору одного из нескольких возможных вариантов алгоритмов, отличающихся соотношением объема программного обеспечения и аппаратных средств.

При этом необходимо исходить из того, что максимальное использование аппаратных средств упрощает разработку и обеспечивает высокое быстродействие контроллера в целом, но сопровождается, как правило, увеличением стоимости и потребляемой мощности. Связано это с тем, что увеличение доли аппаратных средств достигается либо путем выбора более сложного МК, либо путем использования специализированных интерфейсных схем. И то, и другое приводит к росту стоимости и энергопотребления. Увеличение удельного веса программного обеспечения позволяет сократить число элементов контроллера и стоимость аппаратных средств, но это приводит к снижению быстродействия, увеличению необходимого объема внутренней памяти МК, увеличению сроков разработки и отладки программного обеспечения. Критерием выбора здесь и далее является возможность максимальной реализации заданных функций программными средствами при минимальных аппаратных затратах и при условии обеспечения заданных показателей быстродействия и надежности в полном диапазоне условий эксплуатации. Часто определяющими требованиями являются возможность защиты информации (программного кода) контроллера, необходимость обеспечения максимальной продолжительности работы в автономном режиме и другие. В результате выполнения этого этапа окончательно формулируются требования к параметрам используемого МК.

При выборе типа МК учитываются следующие основные характеристики:

- разрядность;
- быстродействие;
- набор команд и способов адресации;
- требования к источнику питания и потребляемая мощность в различных режимах;
- объем ПЗУ программ и ОЗУ данных;

- возможности расширения памяти программ и данных;
- наличие и возможности периферийных устройств, включая средства поддержки работы в реальном времени (таймеры, процессоры событий и т.п.);
- возможность перепрограммирования в составе устройства;
- наличие и надежность средств защиты внутренней информации;
- возможность поставки в различных вариантах конструктивного исполнения;
- стоимость в различных вариантах исполнения;
- наличие полной документации;
- наличие и доступность эффективных средств программирования и отладки МК;
- количество и доступность каналов поставки, возможность замены изделиями других фирм.

Список этот не является исчерпывающим, поскольку специфика проектируемого устройства может перенести акцент требований на другие параметры МК. Определяющими могут оказаться, например, требования к точности внутреннего компаратора напряжений или наличие большого числа выходных каналов ШИМ (широтно-импульсной модуляции).

Номенклатура выпускаемых в настоящее время МК исчисляется тысячами типов изделий различных фирм. Современная стратегия модульного проектирования обеспечивает потребителя разнообразием моделей МК с одним и тем же процессорным ядром. Такое структурное разнообразие открывает перед разработчиком возможность выбора оптимального МК, не имеющего функциональной избыточности, что минимизирует стоимость комплектующих элементов.

Однако для реализации на практике возможности выбора оптимального МК необходима достаточно глубокая проработка алгоритма управления, оценка объема исполняемой программы и числа линий сопряжения с объектом на этапе выбора МК. Допущенные на данном этапе просчеты могут впоследствии привести к необходимости смены модели МК и повторной разводки печатной платы макета контроллера. В таких условиях целесообразно выполнять предварительное моделирование основных элементов прикладной программы с использованием программно-логической модели выбранного МК.

При отсутствии МК, обеспечивающего требуемые по ТЗ характеристики проектируемого контроллера, необходим возврат к этапу разработки алгоритма управления и пересмотр выбранного соотношения между объемом программного обеспечения и аппаратных средств. Отсутствие подходящего МК чаще всего означает, что для реализации необходимого объема вычислений (алгоритмов

управления) за отведенное время нужна дополнительная аппаратная поддержка. Отрицательный результат поиска МК с требуемыми характеристиками может быть связан также с необходимостью обслуживания большого числа объектов управления. В этом случае возможно использование внешних схем обрaмления МК.

На этапе разработки структуры контроллера окончательно определяется состав имеющихся и подлежащих разработке аппаратных модулей, протоколы обмена между модулями, типы разъемов. Выполняется предварительная проработка конструкции контроллера. В части программного обеспечения определяются состав и связи программных модулей, язык программирования. На этом же этапе осуществляется выбор средств проектирования и отладки.

Возможность перераспределения функций между аппаратными и программными средствами на данном этапе существует, но она ограничена характеристиками уже выбранного МК. При этом необходимо иметь в виду, что современные МК выпускаются, как правило, сериями (семействами) контроллеров, совместимых программно и конструктивно, но различающихся по своим возможностям (объем памяти, набор периферийных устройств и т.д.). Это дает возможность выбора структуры контроллера с целью поиска наиболее оптимального варианта реализации.

Нельзя не упомянуть здесь о новой идеологии разработки устройств на базе МК, предложенной фирмой «Scenix». Она основана на использовании высокоскоростных RISC-микроконтроллеров серии SX с тактовой частотой до 100 МГц. Эти МК имеют минимальный набор встроенной периферии, а все более сложные периферийные модули эмулируются программными средствами. Такие модули программного обеспечения называются «виртуальными периферийными устройствами», они обеспечивают уменьшение числа элементов контроллера, времени разработки, увеличивают гибкость исполнения. К настоящему времени разработаны целые библиотеки виртуальных устройств, содержащие отлаженные программные модули таких устройств как модули ШИМ и ФАПЧ, последовательные интерфейсы, генераторы и измерители частоты, контроллеры прерываний и многие другие.

## 2. Разработка и отладка аппаратных средств

После разработки структуры аппаратных и программных средств дальнейшая работа над контроллером может быть распараллелена. Разработка аппаратных средств включает в себя разработку общей принципиальной схемы, разводку топологии плат, монтаж макета и его автономную отладку. Время выполнения этих этапов зависит от имеющегося набора апробированных функционально-топологических модулей, опыта и квалификации разработчика. На этапе ввода принципиальной схемы и разработки топологии используются, как правило, распространенные системы проектирования типа «ACCEL EDA» или «OrCad».

Автономная отладка аппаратуры на основе МК с открытой архитектурой предполагает контроль состояния многоразрядных магистралей адреса и данных с целью проверки правильности обращения к внешним ресурсам памяти и периферийным устройствам. Закрытая архитектура МК предполагает реализацию большинства функций разрабатываемого устройства внутренними средствами микроконтроллера. Поэтому разрабатываемый контроллер будет иметь малое число периферийных ИС, а обмен с ними будет идти преимущественно по последовательным интерфейсам. Здесь на первый план выйдут вопросы согласования по нагрузочной способности параллельных портов МК и отладка алгоритмов обмена по последовательным каналам.

### 3. Разработка и отладка программного обеспечения

Содержание этапов разработки программного обеспечения, его трансляции и отладки на моделях существенно зависит от используемых системных средств. В настоящее время ресурсы 8-разрядных МК достаточны для поддержки программирования на языках высокого уровня. Это позволяет использовать все преимущества структурного программирования, разрабатывать программное обеспечение с использованием отдельно транслируемых модулей. Одновременно продолжают широко использоваться языки низкого уровня типа ассемблера, особенно при необходимости обеспечения контролируемых интервалов времени. Задачи предварительной обработки данных часто требуют использования вычислений с плавающей точкой, трансцендентных функций.

В настоящее время самым мощным средством разработки программного обеспечения для МК являются интегрированные среды разработки, имеющие в своем составе менеджер проектов, текстовый редактор и симулятор, а также допускающие подключение компиляторов языков высокого уровня типа Паскаль или Си. При этом необходимо иметь в виду, что архитектура многих 8-разрядных МК вследствие малого количества ресурсов, страничного распределения памяти, неудобной индексной адресации и некоторых других архитектурных ограничений не обеспечивает компилятору возможности генерировать эффективный код. Для обхода этих ограничений разработчики ряда компиляторов вынуждены были перекладывать на пользователя заботу об оптимизации кода программы.

Для проверки и отладки программного обеспечения используются так называемые программные симуляторы, предоставляющие пользователю возможность выполнять разработанную программу на программно-логической модели МК. Программные симуляторы распространяются, как правило, бесплатно и сконфигурированы сразу на несколько МК одного семейства. Выбор конкретного типа МК среди моделей семейства обеспечивает соответствующая опция меню конфигурации симулятора. При этом моделируется работа ЦП, всех портов ввода/вывода, прерываний и другой периферии. Карта памяти моделируемого МК загружается в симулятор автоматически, отладка ведется в символьных обозначениях регистров.

Загрузив программу в симулятор, пользователь имеет возможность запускать ее в пошаговом или непрерывном режимах, задавать условные или безусловные точки останова, контролировать и свободно модифицировать содержимое ячеек памяти и регистров симулируемого МК.



#### **4. Методы и средства совместной отладки аппаратных и программных средств**

Этап совместной отладки аппаратных и программных средств в реальном масштабе времени является самым трудоемким и требует использования инструментальных средств отладки. К числу основных инструментальных средств отладки относятся:

- внутрисхемные эмуляторы;
- платы развития (оценочные платы);
- мониторы отладки;
- эмуляторы ПЗУ.

Внутрисхемный эмулятор - программно-аппаратное средство, способное заменить эмулируемый МК в реальной схеме. Стыковка внутрисхемного эмулятора с отлаживаемой системой производится при помощи кабеля со специальной эмуляционной головкой, которая вставляется вместо МК в отлаживаемую систему. Если МК нельзя удалить из отлаживаемой системы, то использование эмулятора возможно, только если этот микроконтроллер имеет отладочный режим, при котором все его выводы находятся в третьем состоянии. В этом случае для подключения эмулятора используют специальный адаптер-клипсу, который подключается непосредственно к выводам эмулируемого МК.

Внутрисхемный эмулятор - это наиболее мощное и универсальное отладочное средство, которое делает процесс функционирования отлаживаемого контроллера прозрачным, т.е. легко контролируемым, произвольно управляемым и модифицируемым.

Платы развития, или, как принято их называть в зарубежной литературе, оценочные платы (Evaluation Boards), являются своего рода конструкторами для макетирования электронных устройств. Обычно это печатная плата с установленным на ней МК и всей необходимой ему стандартной периферией. На этой плате также устанавливают схемы связи с внешним компьютером. Как правило, там же имеется свободное поле для монтажа прикладных схем пользователя. Иногда предусмотрена уже готовая разводка для установки дополнительных устройств, рекомендуемых фирмой. Например, ПЗУ, ОЗУ, ЖКИ-дисплей, клавиатура, АЦП и др. Кроме учебных или макетных целей, такие доработанные пользователем платы можно использовать в качестве одноплатных контроллеров, встраиваемых в малосерийную продукцию.

Для большего удобства платы развития комплектуются еще и простейшим средством отладки на базе монитора отладки. Используются два типа мониторов отладки: один для МК, имеющих внешнюю шину, а второй - для МК, не имеющих внешней шины.

В первом случае отладочный монитор поставляется в виде микросхемы ПЗУ, которая вставляется в специальную розетку на плате развития. Плата также имеет ОЗУ для программ пользователя и канал связи с внешним компьютером или терминалом. Во втором случае плата развития имеет встроенные схемы программирования внутреннего ПЗУ МК, которые управляются от внешнего компьютера. При этом программа монитора просто заносится в ПЗУ МК совместно с прикладными кодами пользователя. Прикладная программа должна быть специально подготовлена: в нужные места необходимо вставить вызовы отладочных подпрограмм монитора. Затем осуществляется пробный прогон. Чтобы внести в программу исправления, пользователю надо стереть ПЗУ и произвести повторную запись. Готовую прикладную программу получают из отлаженной путем удаления всех вызовов мониторных функций и самого монитора отладки.

Возможности отладки, предоставляемые комплектом «плата развития плюс монитор», не столь универсальны, как возможности внутрисхемного эмулятора, да и некоторая часть ресурсов МК в процессе отладки отбирается для работы монитора. Тем не менее, наличие набора готовых программно-аппаратных средств, позволяющих без потери времени приступить к монтажу и отладке проектируемой системы, во многих случаях является решающим фактором. Особенно если учесть, что стоимость такого комплекта несколько меньше, чем стоимость более универсального эмулятора.

Эмулятор ПЗУ — программно-аппаратное средство, позволяющее замещать ПЗУ на отлаживаемой плате, и подставляющее вместо него ОЗУ, в которое может быть загружена программа с компьютера через один из стандартных каналов связи. Это устройство позволяет пользователю избежать многократных циклов перепрограммирования ПЗУ. Эмулятор ПЗУ нужен только для МК, которые могут обращаться к внешней памяти программ. Это устройство сравнимо по сложности и по стоимости с платами развития и имеет одно большое достоинство: универсальность. Эмулятор ПЗУ может работать с любыми типами МК.

Эмулируемая память доступна для просмотра и модификации, но контроль над внутренними управляющими регистрами МК был до недавнего времени невозможен.

В последнее время появились модели интеллектуальных эмуляторов ПЗУ, которые позволяют «заглядывать» внутрь МК на плате пользователя. Интеллектуальные эмуляторы представляют собой гибрид из обычного эмулятора ПЗУ, монитора отладки и схем быстрого переключения шины с одного на другой. Это создает эффект, как если бы монитор отладки был установлен на плате пользователя и при этом он не занимает у МК никаких аппаратных ресурсов, кроме небольшой зоны программных шагов, примерно 4К.

Этап совместной отладки аппаратных и программных средств в реальном масштабе времени завершается, когда аппаратура и программное обеспечение совместно обеспечивают выполнение всех шагов алгоритма работы системы. В конце этапа отлаженная программа заносится с помощью программатора в энергонезависимую память МК, и проверяется работа контроллера без эмулятора. При этом используются лабораторные источники питания. Часть внешних источников сигналов может моделироваться.

Этап интеграции разработанного контроллера в изделие заключается в повторении работ по совместной отладке аппаратуры и управляющей программы, но при работе в составе изделия, питании от штатного источника и с информацией от штатных источников сигналов и датчиков.

Состав и объем испытаний разработанного и изготовленного контроллера зависит от условий его эксплуатации и определяется соответствующими нормативными документами. Проведение испытаний таких функционально сложных изделий, как современные контроллеры, может потребовать разработки специализированных средств контроля состояния изделия во время испытаний.

## **Контрольные вопросы**

1. Какая сфера применения является наиболее типичной для цифровых устройств на микроконтроллерах?
2. Что такое «программный симулятор»?
3. Какую функцию выполняет «монитор» на плате развития?
4. Что включает в себя понятие «работа в реальном масштабе времени»?
5. Что такое «внутрисхемный эмулятор»?
6. Что такое «эмулятор ПЗУ»?
7. Что включает в себя понятие «закрытая архитектура» микроконтроллера?
8. Что такое «плата развития»?
9. Что такое «виртуальное» периферийное устройство МК?

Министерство образования РБ

Учреждение образования «Белорусский государственный  
университет информатики и радиоэлектроники»

Кафедра радиоэлектронных средств

**ПРАКТИЧЕСКАЯ РАБОТА №2**

**«Разработка программного обеспечения  
для PIC-микроконтроллеров»**

## **Разработка программного обеспечения для PIC-микроконтроллеров**

Разработка программного обеспечения является центральным моментом общего процесса проектирования. Центр тяжести функциональных свойств современных цифровых систем находится именно в программных средствах.

Основным инструментом для профессиональной разработки программ является ассемблер, предполагающий детализацию на уровне команд МК. Только ассемблер позволяет максимально использовать ресурсы кристалла.

Для микроконтроллеров PIC выпущено большое количество различных средств разработки. В данной главе речь пойдет о средствах, предоставляемых фирмой Microchip, которые весьма эффективны и широко используются на практике.

### **1. Ассемблер MPASM**

Ассемблер MPASM представляет собой интегрированную программную среду для разработки программных кодов PIC микроконтроллеров всех семейств. Выпускается фирмой Microchip в двух вариантах: для работы под DOS и для работы под Windows 95/98/NT. Ассемблер MPASM может использоваться как самостоятельно, так и в составе интегрированной среды разработки MPLAB. Он включает несколько программ: собственно MPASM, MPLINK и MPLIB, причем каждая из них обладает собственным интерфейсом.

Программа MPASM может использоваться для двух целей:

- генерации исполняемого (абсолютного) кода, предназначенного для записи в МК с помощью программатора;
- генерации перемещаемого объектного кода, который затем будет связан с другими ассемблированными или скомпилированными модулями.

Исполняемый код является для MPASM выходным кодом по умолчанию. При этом все переменные источника должны быть явно описаны в тексте программы или в файле, подключаемом с помощью директивы INCLUDE <filename>. Если при ассемблировании не выявляется ошибок, то генерируется выходной .hex-файл, который может быть загружен в МК с помощью программатора.

При использовании ассемблера MPASM в режиме генерации перемещаемого объектного кода формируются объектные модули, которые могут быть впоследствии объединены с другими модулями при помощи компоновщика MPLINK. Программа-компоновщик MPLINK преобразует перемещаемые объектные коды в исполняемый бинарный код, привязанный к абсолютным адресам МК. Библиотечная утилита MPLIB позволяет для удобства работы сгруппировать перемещаемые объекты в один файл или библиотеку. Эти библиотеки могут быть связаны компоновщиком MPLINK в файл

выходного объектного кода ассемблера MPASM.

Программы MPASM и MPLINK доступны через оболочку MPASM, тогда как MPLIB доступна только со своей командной строки.

Исходным файлом для ассемблера MPASM по умолчанию является файл с расширением .ASM. Текст исходного файла должен соответствовать требованиям синтаксиса, приведенным далее.

Ассемблер MPASM может быть вызван командной строкой

MPASM [/<Option>[ /<Option>...]] <file\_name>

где /<Option> означает выбор режима работы ассемблера в командной строке; <file\_name> - имя файла на ассемблирование.

Режимы работы ассемблера, выбранные по умолчанию, приведены в табл. 1.

Выбор	Значение по умолчанию	Описание
?	N/A	Вызвать помощь
a	INHx8M	Генерировать абсолютный .COD и hex выход непосредственно из ассемблера:
c	On	Выбрать/запретить случай чувствительности
e	On	Выбрать/запретить файл ошибок
h	N/A	Отобразить панель помощи MPASM
l	On	Выбрать/запретить файл листинга, генерированный из макроассемблера.
m	On	Вызвать/запретить макрорасширение
o	N/A	Установить путь для объектных файлов /o<path>\object.file
p	None	Установить тип процессора: /p<processor_type>
q	Off	Разрешить/Запретить скрытый режим (запретить вывод на экран)
r	Hex	Определяет тип числа по умолчанию: /r<radix>
w	0	Определяет уровень диагностических сообщений в файле листинга /w<level>, где <level> может быть: 0 - сообщать все, 1 - сообщать о предупреждениях и ошибках, 2 - сообщать только об ошибках.
x	Off	Разрешить/запретить перекрестные ссылки в файле листинга.

Табл. 1. Режимы работы ассемблера по умолчанию.

Здесь и далее используются следующие соглашения по использованию символов:

[ ] — для аргументов по выбору;

< > - для выделения специальных ключей <TAB>, <ESC> или дополнительного выбора;

| - для взаимоисключающих аргументов (выбор ИЛИ);

строчные символы - для обозначения типа данных.

Выбор по умолчанию, приведенный в табл. 1, может быть изменен командной строкой:

/<option> разрешает выбор;

/<option>+ разрешает выбор;

/<option>- запрещает выбор.

Исходный ассемблерный файл создается с использованием любого ASCII текстового редактора. Каждая линия исходного файла может содержать до четырех типов информации:

- метки (labels)
- мнемоника (mnemonics)
- операнды (operands)
- комментарий (comments)

Порядок и положение каждого типа имеет значение. Метка должна начинаться в колонке номер один. Мнемоника может начинаться в колонке два или далее. Операнды идут за мнемоникой. Комментарий может следовать за операндом, мнемоникой или меткой или может начинаться в любом столбце, если в качестве первого не пустого символа используется \* или ;.

Максимальная длина строки 255 символов.

Один или несколько пробелов должны отделять метку и мнемонику или мнемонику и операнд(ы). Операнды могут отделяться запятой. Например:

List p=16C54, r=HEX

```
      ORG      0x1FF    ;Вектор сброса
      GOTO     START    ;Возврат на начало
      ORG      0x000    ;Адрес начала исполнения программы
START
      MOVLW    0x0A      ;выполнение программы PIC МК
      MOVLW    0x0B      ;
      GOTO     START    ;выполнять всегда
      END
```

### Метки

В поле метки размещается символическое имя ячейки памяти, в которой хранится отмеченный операнд. Все метки должны начинаться в колонке 1. За ними может следовать двоеточие (:), пробел, табуляция или конец строки. Комментарий может также начинаться в колонке 1, если использует



ся одно из обозначений комментария.

Метка может начинаться с символа или нижнего тире ( \_ ) и содержать буквенные символы, числа, нижние тире и знак вопроса. Длина метки может быть до 32 символов.

### **Мнемоники**

Мнемоники представляют собой мнемонические обозначения команды, которые непосредственно транслируются в машинный код. Мнемоники ассемблерных инструкций, директивы ассемблера и макровыводы должны начинаться, по крайней мере, в колонке 2. Если есть метка на той же линии, она должна быть отделена от этой метки двоеточием или одним или более пробелами или табуляцией.

### **Операнды**

В этом поле определяются операнды (или операнд), участвующие в операции. Операнды должны быть отделены от мнемоники одним или более пробелами или табуляцией. Операнды отделяются друг от друга запятыми. Если операция требует фиксированного номера (числа) или операндов, то все на линии после операндов игнорируется. Комментарии разрешаются в конце линии. Если мнемоники позволяют использовать различное число операндов, конец списка операндов определяется концом строки или комментарием.

Выражения используются в поле операнда и могут содержать константы, символы или любые комбинации констант и символов, разделенных арифметическими операторами. Перед каждой константой или символом может стоять + или - , что указывает на положительное или отрицательное выражение.

В ассемблере MPASM используются следующие форматы выражений:

- текстовая строка;
- числовые константы и Radix;
- арифметические операторы и приоритеты;
- High / Low операторы.

Текстовая строка - это последовательность любых допустимых ASCII символов (в десятичном диапазоне от 0 до 127), заключенная в двойные кавычки. Строка может иметь любую длину в пределах 132 колонок. При отсутствии ограничения строки она считается до конца линии. Если строка используется как буквенный операнд, она должна иметь длину в один символ, иначе будет ошибка.

Числовая константа представляет собой число, выраженное в некоторой системе счисления. Перед константой может стоять + или -. Промежуточные величины в константах рассматриваются как 32-разрядные целые без знака.

MPASM поддерживает следующие системы счисления (представления значений или Radix): шестнадцатеричную, десятичную, восьмиричную, двоичную и символьную. По умолчанию принимается шестнадцатеричная система. Табл. 2 представляет различные системы счисления.

Операторы - это арифметические символы, подобные + и -, которые используются при формировании выражений. Каждый оператор имеет свой приоритет. В общем случае приоритет устанавливается слева направо, а выражения в скобках оцениваются первыми. В табл. 3 приведены обозначения, описания и примеры применения основных операторов MPASM.

Тип	Синтаксис	Пример
Десятичная	D'<цифры>'или .<цифры>	D'100' или .100
16-ричная	H'<цифры>' или 0х<цифры>	H'9f' или 0x9f
Восьмиричная	O'<цифры>'	O'777'
Двоичная	B'<цифры>'	B'00111001'
Символьная	'<символ>' или A'<символ>'	"C" или A'C'

**Табл .2.** Системы счисления (Radix).

Оператор	Описание	Пример
\$	Текущий счетчик команд	goto \$ + 3
(	левая скобка	1 + (d * 4)
)	правая скобка	(lenght + 1 ) * 255
!	операция «НЕ» (логическая инверсия)	if! ( a - b )
~	дополнение	flags = ~ flags
-	инверсия (двоичное дополнение)	- 1 * lenght
High	выделить старший байт слова	movlw high llasid
Low	выделить младший байт слова	movlw low (llasid + .251)
upper	выделить наибольший байт	movlw upper (llasid + слова
*	Умножение	a = c * b
/	Деление	a = b/c
%	Модуль	lenght = totall % 16
+	Сложение	Tot_len = lenght * 8 + 1
-	Вычитание	EntrySon = ( Tot - 1 ) / 8
<<	сдвиг влево	Val = flags << 1
>>	сдвиг вправо	Val = flags >> 1
>=	больше либо равно	if ent >= num
>	больше	if ent > num
<	меньше	if ent < num
<=	меньше либо равно	ifent<=num

**Табл. 3.** Основные арифметические операторы MPASM.

==	равно	ifent==num
!=	не равно	if ent != num
&	поразрядное «И»	flags = flags & err_bit
^	поразрядное «ИСКЛЮЧАЮЩЕЕ ИЛИ»	flags = flags ^ err_bit
	поразрядное «ИЛИ»	flags = flags   err_bit
&&	логическое «И»	if (len == 512)&&( b == c)
	логическое «ИЛИ»	if (len == 512 )    ( b == c )
=	установить равным...	entry index = 0
++	увеличить на 1 (инкремент)	i ++
—	уменьшить на 1 (декремент)	i —

**Табл. 3.** Основные арифметические операторы MPASM (продолжение).

Операторы high, low и crrer используются для получения одного байта из многобайтного значения, соответствующего метке. Применяются для управления расчетом точек динамического перехода при чтении таблиц и записи программ.

Операторы инкремента и декремента могут применяться к переменной только в качестве единственного оператора в строке. Они не могут быть встроенным фрагментом более сложного выражения.

### Комментарии

Поле комментария может использоваться программистом для текстового или символьного пояснения логической организации программы. Поле комментария полностью игнорируется ассемблером, поэтому в нем можно применять любые символы. Комментарии, которые используются в строке сами по себе, должны начинаться с символа комментария (\* или ;). Комментарии в конце строки должны быть отделены от остатка строки одним или более пробелами или табуляцией.

### Расширения файлов, используемые MPASM и утилитами

Существует ряд расширений файлов, применяемых по умолчанию MPASM и связанными утилитами. Назначения таких расширений приведены в табл.4.

Расширение	Назначение
.ASM	Входной файл ассемблера для MPASM <source_name>.ASM
.OBJ	Выходной файл перемещаемого объектного кода из MPASM <source_name>.OBJ
.LST	Выходной файл листинга, генерируемый ассемблером MPASM или MPLINK: <source_name>.LST

**Табл. 4.** Используемые по умолчанию назначения расширений файлов

Расширение	Назначение
.ERR	Выходной файл ошибок из MPASM: <source_name>.ERR
.MAP	Выходной файл распределения памяти из MPASM: <source_name>.MAP
.HEX	Выходной файл объектного кода в шестнадцатичном представлении из MPASM: <source_name>.HEX
.HXL/.HXH	Выходной файл объектного кода в шестнадцатичном представлении с отдельным представлением младших и старших байт: <source_name>.HXL, <source_name>.HXH
.LIB	Библиотечный файл, созданный MPLIB и привязанный компоновщиком MPLINK: <source_name>.LIB
.LNK	Выходной файл компоновщика: <source_name>.LNK
.COD	Выходной символьный файл или файл отладчика. Формируются MPASM или MPLINK: <source_name>.COD

**Табл. 4.** Используемые по умолчанию назначения расширений файлов (продолжение).

Листинг представляет собой текстовый файл в формате ASCII, который содержит машинные коды, сгенерированные в соответствии с каждой ассемблерной командой, директивой ассемблера или макрокомандой исходного файла. Файл листинга содержит: имя продукта и версии, дату и время, номер страницы вверху каждой страницы.

В состав листинга входят также таблица символов и карта использования памяти. В таблице символов перечисляются все символы, которые есть в программе, и где они определены. Карта использования памяти дает графическое представление о расходовании памяти МК.

### Директивы языка

Директивы языка — это ассемблерные команды, которые встречаются в исходном коде, но не транслируются прямо в исполняемые коды. Они используются ассемблером при трактовке мнемоники входного файла, размещении данных и формировании файла листинга.

Существует четыре основных типа директив в MPASM:

- директивы данных;
- директивы листинга;
- управляющие директивы;
- макро-директивы.

Директивы данных управляют распределением памяти и обеспечивают доступ к символическим обозначениям данных.

Директивы листинга управляют листингом файла MPASM и форматом. Они определяют спецификацию заголовков, генерацию страниц и другие функции управления листингом.

Директивы управления позволяют произвести секционирование обычного ассемблерного кода.

Макро-директивы управляют исполнением и распределением данных в пределах определений макротела.

Ниже приводится описание некоторых директив ассемблера MPASM, используемых в данном учебном пособии.

**CODE** - начало секции объектного кода

Синтаксис:

```
[<label>] code [ROM address>]
```

Используется при генерации объектных модулей. Объявляет начало секции программного кода. Если <label> не указана, секция будет названа .code Стартовый адрес устанавливается равным указанному значению или нулю, если адрес не был указан.

Пример:

```
RESET code H'OIFF'  
    goto START
```

**#DEFINE** - определить метку замены текста

Синтаксис:

```
#define <name> [<string>]
```

Директива задает строку <string>, замещающую метку <name> всякий раз, когда та будет встречаться в исходном тексте.

Символы, которые определены директивой #DEFINE, не могут быть просмотрены симулятором. Используйте вместо этой директивы EQU.

Пример:

```
#define length 20  
#define control 0x19,7  
#define position (X,YZ) (y-(2 * Z +X)).  
testjabel dwposition(1, length, 512)  
bsf control          ; установить в 1 бит 7 в f19
```

**END** - конец программного блока

Синтаксис:

```
end
```

Определяет конец программы. После остановки программы таблица символов сбрасывается в файл листинга.

Пример:

```
start  
;исполняемый код
```

```

;
end                ; конец программы
EQU - определить ассемблерную константу
Синтаксис:

```

```

<label> equ <expr>

```

Здесь <expr> - это правильное MPASM выражение. Значение выражения присваивается метке <label>.

Пример:

```

four equ 4          ; присваивает численное значение метке four

```

**INCLUDE** - включить дополнительный файл источника  
Синтаксис:

```

include <<include_file>>
include "<include_file>"

```

Определяемый файл считывается как источник кода. По окончании включаемого файла будет продолжаться ассемблирование исходника. Допускается до шести уровней вложенности. <include\_file> может быть заключен в кавычки или угловые скобки. Если указан полный путь к файлу, то поиск будет происходить только по этому пути. В противном случае порядок поиска следующий: текущий рабочий каталог, каталог, в котором находится исходник, каталог MPASM.

Пример:

```

include "c:\sys\sysdefs.inc"    ; system defs
include <addmain.asm>           ; register defs

```

**LIST** - установить параметры листинга  
Синтаксис:

```

list [<list_option>, , <list_option>]

```

Директива <list> разрешает вывод листинга, если он до этого был запрещен. Кроме того, один из параметров листинга может быть изменен для управления процессом ассемблирования в соответствии с табл. 5.

Параметр	Значение по умолчанию	Описание
C=nnn	80	Количество символов в строке
n=nnn	59	Количество строк на странице
t=ON OFF	OFF	Укорачивать строки листинга

**Табл. 5.** Параметры, используемые директивой list

Параметр	Значение по умолчанию	Описание
p=<type>	None	Установить тип процессора: PIC16C54, PIC16C84, PIC16F84, PIC17C42 и др.
r=<radix>	HEX	Установить систему счисления по умолчанию: hex, dec, oct.
w=<level>	0	Установить уровень сообщений диагностики в файле листинга: 0 - выводить все сообщения; 1 - выводить предупреждения и ошибки; 2 - выводить только ошибки.
x=ON OFF	OFF	Включить или выключить макрорасширения.

**Табл. 5.** Параметры, используемые директивой list (продолжение)

**NOLIST** - выключить выход листинга

Синтаксис:

NOLIST

**ORG** — установить начальный адрес программы

Синтаксис:

<label> org <expr>

Устанавливает начальный адрес программы для последующего кода в соответствии с адресом в <expr>. MPASM выводит перемещаемый объектный код, а MPLINK разместит код по определенному адресу. Если метка <label> определена, то ей будет присвоена величина <expr>. По умолчанию начальный адрес имеет нулевое значение. Директива может не использоваться, если создается объектный модуль.

Пример:

```
int_1 org 0x20          ;Переход по вектору 20
int_2 org int_1+0x10    ; Переход по вектору 30
```

**PROCESSOR** - установить тип процессора

Синтаксис:

processor <processor\_type>

Устанавливает тип используемого процессора <processor\_type>: [16C54 | 16C55 | 16C56 | 16C57 | 16C71 | 16C84 | 16F84 | 17C42]. Общие процессорные семейства могут быть выбраны как:[16C5X | 16CXX | 17CXX] Для поддержания совместимости с новыми изделиями выбирается максимум доступной памяти.

**SET** - определить ассемблерную переменную

Синтаксис:

<label> set <expr>

Директива SET функционально эквивалентна директиве EQU, за исключением того, что величина, определяемая SET, может быть изменена директивой SET.

Пример:

```
area set 0
widthset 0x12
length set 0x14
area set    length * width
length     set length + 1
```

**TITLE** - Определить программный заголовок

Синтаксис:

title "<title\_text>"

Эта директива устанавливает текст, который используется в верхней линии страницы листинга. < title\_text > - это печатная ASCII последовательность, заключенная в двойные скобки. Она может быть до 60 символов длиной.

Пример

title "operational code, rev 5.0"

## 2. компоновщик MPLINK

Абсолютный (неперемещаемый) код программы генерируется непосредственно при ассемблировании и располагается в программной памяти в порядке следования операторов программы. Операторы перехода на метку сразу же заменяются соответствующим кодом перехода на адрес метки.

При генерации перемещаемого кода каждая секция программного кода должна предваряться директивой CODE. Окончательное размещение программных кодов, расстановку физических адресов переходов выполняет компоновщик MPLINK.

Компоновщик MPLINK выполняет следующие задачи:

- распределяет коды и данные, т.е. определяет, в какой части программной памяти будут размещены коды и в какую область ОЗУ будут помещены переменные;
- распределяет адреса, т.е. присваивает ссылкам на внешние объекты в объектном файле конкретные физические адреса;
- генерирует исполняемый код, т.е. выдает файл в формате .hex, который может быть записан в память МК;
- отслеживает конфликты адресов, т.е. гарантирует, что программа



или данные не будут размещаться в пространстве адресов, которое уже занято;

- предоставляет символьную информацию для отладки.

Для более подробного изучения работы компоновщика следует обратиться к специальной литературе.

### **3. Менеджер библиотек MPLIB**

Менеджер библиотек позволяет создавать и модифицировать файлы библиотек. Библиотечный файл является коллекцией объектных модулей, которые размещены в одном файле. MPLIB использует объектные модули с именем типа «filename.o» формата COFF (Common Object File Format).

Использование библиотечных файлов упрощает компоновку программы, делает ее более структурированной и облегчает ее модификацию.

### **4. Симулятор MPSIM**

Симулятор MPSIM представляет собой симулятор событий, предназначенный для отладки программного обеспечения PIC-контроллеров. MPSIM моделирует все функции контроллера, включая все режимы сброса, функции таймера/счетчика, работу сторожевого таймера, режимы SLEEP и Power-down, работу портов ввода/вывода.

MPSIM запускается из командной строки DOS, конфигурируется пользователем и непосредственно применяет выходные данные ассемблера MPASM.

Перед использованием симулятора необходимо отассемблировать исходный файл <file\_name>.asm и получить файл объектного кода в формате INHX8M, создаваемый MPASM по умолчанию:

```
MPASM <file_name>.asm <RETURN>
```

Чтобы запустить симулятор, необходимо набрать в командной строке

```
MPSIM<RETURN>.
```

Вид экрана, получаемого при запуске MPSIM, показан на рис. 1. Экран разделен на три части, или окна. В верхнем окне показано текущее состояние моделирования, включая моделируемую программу, тип МК, число выполненных командных циклов и затраченное на них время. Среднее окно используется для вывода содержимого регистров пользователя. Набор регистров и формат выводимых на экран данных определяются файлом MPSIM.INI, который далее будет описан подробнее. Нижнее окно содержит приглашение на ввод команд, а также текущие операции и результат их выполнения.

При запуске симулятор MPSIM начинает искать командный файл MPSIM.INI. Этот текстовый файл создается пользователем и используется для задания всех задействованных в программе параметров.

User4 RADIX=X MPSIM 5.20 16c84 TIME=0.0u 0 ?=Help	
W: 00 F1: 00 F2: 1FF F3: 0001111 IOA: OF F5: OF	
%P84	;Choose Microcontroller number = 84
%SR X	;Set Input/Output radix to hexadecimal
%ZR	;Set all registers to 0
%ZT	;Zero elapsed time counter to 0
%RE	;Reset elapsed time and step count
%V W,X,2	;register W
%AD F1,X,2	;register TMR0
%AD F2,X,3	;register PCL
%AD F3,B,8	;register STATUS
%AD IOA,X,2	;Port "A" TRIS register
%AD F5,X,2	;Port "A" register
%RS	;Reset
%SC 1	;Set the clock 1MHz
%LO user4	
Hex code loaded	
Listing file	loaded
Symbol table	loaded
218960 bytes	memory free
%	

**Рис. 1** Вид рабочего окна симулятора MPSIM.

Один из примеров файла MPSIM.INI приведен ниже.

	; MPSIM file for user4
P84	;использование МК семейства PIC16C84
SR X	;представление данных в 16-ричном формате
ZR	;сброс регистров МК в нуль
ZT	;сброс таймера в нуль
RE	;сброс времени выполнения команды и счетчика циклов
V W,X,2	;вывод регистра W в hex формате на два знакоместа
AD F1 ,X,2	;вывод на экран регистра TMR0 в hex формате на два знакоместа
AD F2,X.3	;вывод на экран регистра PCL в hex формате на три знакоместа
AD F3,B,8	;вывод на экран регистра STATUS в bin формате на восемь знакомест
AD IOA,X,2	;вывод на экран регистра TRISA в hex формате на два знакоместа
AD F5,X,2	;вывод на экран регистра порта A в hex формате на два знакоместа
SC 1	установка тактовой частоты 1 МГц
RS	;сброс МК
LO user4	

В представленном файле указаны: тип микроконтроллера, система счисления данных по умолчанию, регистры, содержимое которых выводится на экран, способ представления данных, рабочие параметры. Любая команда, которая выполняется MPSIM, может быть задана в файле MPSIM.INI, который определяет начальное состояние программы. При работе MPSIM создает файл MPSIM.JRN, в котором сохраняются все сведения о нажатии клавиш в процессе работы.

В файле MPSIM.INI допускается вводить комментарии, которые даются после знака «;», но не допускается использование пустых строк.

Основные команды, применяемые в симуляторе MPSIM, приведены в табл. 6. Когда эти команды вводятся в сеансе работы с MPSIM, они заносятся в файл MPSIM.JRN, который используется при создании расширенного файла MPSIM.INI. Данный файл можно задействовать для выявления ошибок и обеспечения нормального выполнения программы после исправления кода.

Команда	Параметр	Комментарии
AB	-	Прерывание текущей сессии
AD	Reg[, Radix[, Digits]]	Вывод содержимого регистра на экран в указанном формате и заданной системе счисления X, B или D
B	[addr]	Установка точки останова по текущему или указанному адресу
C	[#break]	Продолжение выполнения программы с пропуском указанного количества следующих точек останова
DB	-	Вывод на экран всех активных точек останова
DI	[addr1[,addr2]]	Вывод на экран фрагмента памяти программ
DR	-	Вывод содержимого всех регистров
DW	[E D]	Разрешение/запрещение функционирования сторожевого таймера
E	[addr]	Выполнение программы с текущего или указанного адреса
F	Reg	Вывод на экран содержимого регистра и возможность его редактирования пользователем
GE	Filename	Получение и выполнение командного файла. Это способ загрузки командного файла .INI
GO	—	Запуск МК и начало выполнения программы
IP	[time step]	Ввод входных воздействий в соответствии со значением параметра step в файле Stimulus
LO	Filename	Загрузка в MPSIM файлов .HEX и .COD
M	Addr	Вывод на экран содержимого памяти программ, начиная с адреса «addr» и возможность его редактирования. Ввод «Q» завершает команду.

**Табл. 6.** Основные команды симулятора MPSIM.

Команда	Параметр	Комментарии
P	device	Выбор типа моделируемого МК
Q	-	Выход из MPSIM и запись команд в файл JRN
RE	-	Сброс времени выполнения и счетчика циклов
RS	-	Сброс моделируемого МК
SE	pin port	Вывод на экран состояния указанного вывода или порта и возможность его изменения
SR	O X D	Установка системы счисления по умолчанию
SS	[addr]	Пошаговое исполнение, начиная с указанного адреса. При отсутствии адреса - исполнение идет с текущего места
ST	Filename	Загрузка файла стимуляции
W		Отображение состояния регистра W с возможностью его модификации
ZM	addr1,addr2	Очистка памяти программ с адреса addr1 по addr2
ZR	-	Сброс всех регистров МК
ZT	-	Сброс таймера/счетчика МК

**Табл. 6.** Основные команды симулятора MPSIM (продолжение).

Для моделирования внешних тестовых событий (воздействий) на моделируемый МК используются файлы стимуляции с расширением .STI. Эти файлы используются MPSIM для того, чтобы обеспечить подачу однократных и повторяющихся входных сигналов в процессе выполнения программы. При этом можно наблюдать на экране, как МК реагирует на сигналы.

В качестве примера ниже приведен файл для тестирования программы, выполняющей опрос состояния линии 1 порта A.

```
! test1 .STI
STEP      RA1
1          ! Установка на входе RA1 состояния "1"
200        0 !Поступление на вход RA1 сигнала "0"
1000       1 !Переход сигнала на входе RA1 в "1"
1200       0 !Повторная подача нулевого сигнала
```

Файл воздействия состоит из множества состояний, для которых задается параметр STEP, определяющий число циклов, в течение которых поддерживается указанное состояние. Он позволяет одновременно подавать сигналы на различные выходы МК. В файле воздействия можно указать любой вывод МК, в том числе и вывод сброса (\_MCLR). Для обозначения комментариев используется знак !.

**Министерство Образования Республики Беларусь**

**УО “БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ”**

**Кафедра РЭС**

## **Практическая работа №3**

по курсу: микропроцессорные системы и их применение;

на тему:

**“ Практика программирования PIC-микроконтроллеров.”**

**Минск 2006**

# Практика программирования PIC-микроконтроллеров

## 1. Описание лабораторного макета

Для того чтобы написать первые учебные программы и проверить их функционирование, желательно иметь относительно несложный макет, содержащий самые распространенные периферийные устройства. Схема подобного макета, используемого при выполнении лабораторных работ студентами, приведена на рисунке 1.

Макет питается от источника стабилизированного напряжения +5В. Тактовая частота МК задается RC-цепью и составляет около 2 МГц. К линии RA0 порта А подключен биполярный транзистор в ключевом режиме, нагруженный на динамик BA1. Звучание динамика обеспечивается подачей на выход RA0 изменяющегося сигнала в звуковом диапазоне. К линии RA1 порта А подключен светодиод VD2, светящийся при высоком напряжении на выходе. Тумблеры SA1 и SA2, а также кнопки SB1 и SB2 подключены, соответственно, к линиям RA2 и RA3 порта А, а также к линии RA4 порта А и линии RB0 порта В. Исходное состояние кнопок - разомкнутое, что обеспечивает подачу на соответствующие входы МК высокого уровня сигнала.

Линии RB1 - RB7 порта В обслуживают семисегментный индикатор HL1 с общим анодом. Поэтому свечение сегмента индикатора обеспечивается при низком уровне сигнала на соответствующем выходе порта В. Макет также содержит средства программирования и связи с компьютером, которые на схеме не показаны.

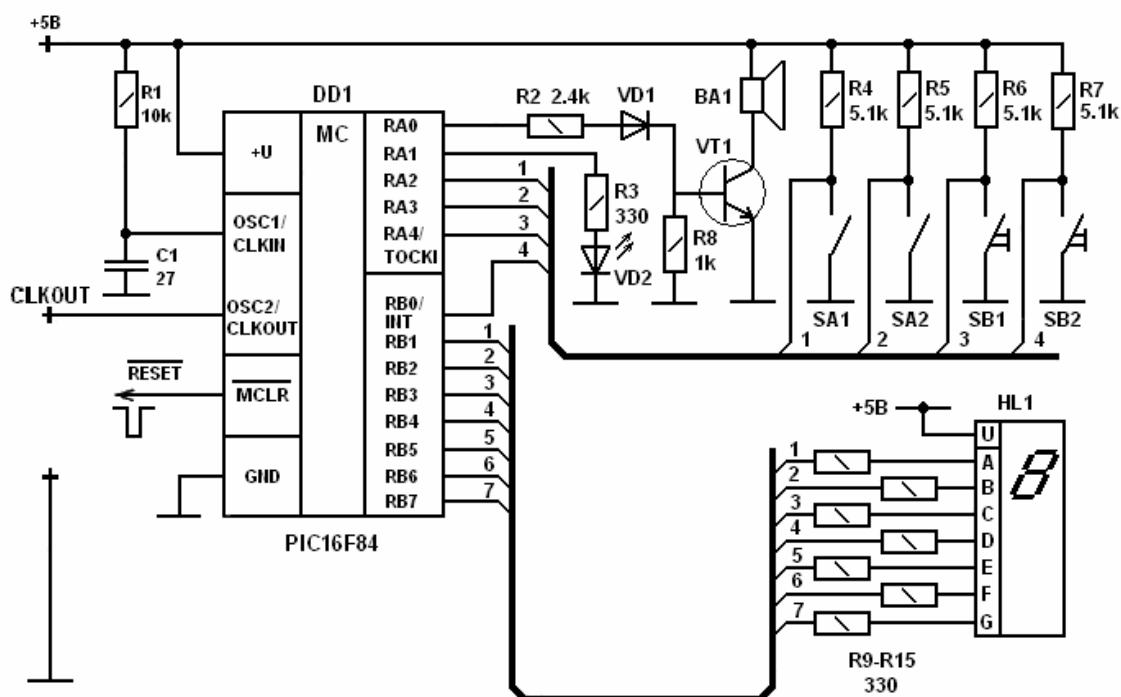


Рис.1 Схема лабораторного макета.

## 2. Инициализация микроконтроллера макета

Прежде чем переходить к созданию простейших пользовательских программ, необходимо описать используемые в дальнейшем переменные и настроить МК на работу с выбранным макетом. С этой целью мы напишем и подробно рассмотрим листинг исходной программы `init.asm`, в состав которой будут включаться все остальные программы пользователя.

```

*****
;
;* листинг исходной программы
*****

LIST P=16C84, R=HEX          ;директива, определяющая тип
                              ;процессора и систему счисления
                              ;по умолчанию
*****

;* описание используемых переменных и назначения адресов
;* ячеек для хранения переменных пользователя
*****

;      INTCON      EQU      0x0B
;      OPTION      EQU      0x81
;      TMRO        EQU      0x01
;      INTF        EQU      1
;      TOIF        EQU      5
;      PCL         EQU      0x02
;      STATUS      EQU      0x03
;      RP0         EQU      5
;      PORTA       EQU      0x05
;      PORTB       EQU      0x06
;      TRISA       EQU      0x05
;      TRISB       EQU      0x06
;      W           EQU      0
;      F           EQU      1
;      TEMP        EQU      0x0C
;      TEMPB       EQU      0x0D
;      COUNT1      EQU      0x0E
;      COUNT2      EQU      0x0F
;      COUNT3      EQU      0x10
*****

;* определение меток замены текста
*****
;      #DEFINE     Z      STATUS,2      ;бит нулевого результата
;      #DEFINE     BA1    PORTA,0       ;динамик BA1
;      #DEFINE     VD2    PORTA,1       ;светодиод VD2
;      #DEFINE     SA1    PORTA,2       ;тумблер SA1
;      #DEFINE     SA2    PORTA,3       ;тумблер SA2

```

```

#DEFINE      SB1      PORTA,4      ;кнопкаSB1
#DEFINE      SB2      PORTB,0      ;кнопка SB2
#DEFINE      HL1_A    PORTB,1      ;индикатор-сегмент А
#DEFINE      HL1_B    PORTB,2      ;индикатор-сегмент В
#DEFINE      HL1_C    PORTB,3      ;индикатор-сегмент С
#DEFINE      HL1_D    PORTB,4      ;индикатор-сегмент D
#DEFINE      HL1_E    PORTB,5      ;индикатор-сегмент Е
#DEFINE      HL1_F    PORTB,6      ;индикатор-сегмент F
#DEFINE      HL1_G    PORTB,7      ;индикатор-сегмент G
.*****
;
;*исполняемая программа
.*****
;
      ORG          0x000            ;установка начального адреса по
                                   ;сбросу
      GOTO         BEGIN           ;переход на начало программы
      ORG          0x005            ;установка начального адреса
                                   ;размещения программы

      BEGIN
      CALL         INIT_PORTS      ;вызов подпрограммы
                                   ;инициализации портов МК
.*****
;
;*программа пользователя
.*****
;
;
INIT_PORTS                                ;подпрограмма инициализации портов
      MOVLW        0xFF             ;установка линий портов
      MOVWF        PORTA            ;А и В в единичное
      MOVWF        PORTB            ;состояние
      BSF          STATUS,RP0       ;переход на банк 1
      MOVLW        0x1C             ;настройка линий RA0 и
      MOVWF        TRISA            ;RA1 порта А на вывод –
                                   ;остальных - на ввод
      MOVLW        0x01             ;настройка линии RB0
      MOVWF        TRISB            ;портаВ на ввод-
                                   ;остальных - на вывод
      BCF          STATUS.RP0       ;возврат в банк 0
      RETURN                    ;возврат из подпрограммы
;
      END                          ;конец программы

```

Рассмотрим работу этой программы. Вначале она указывает ассемблеру тип используемого МК и систему счисления по умолчанию. Идущие далее ассемблерные директивы EQU определяют ассемблерные константы, используемые в этой и последующих программах. Они позволяют использовать в тексте программы более удобные мнемонические метки, связанные к структуре конкретного МК, вместо корректных, но более



сложных ассемблерных выражений. Указатели `TEMPA`, `TEMPB`, `COUNT1` и `COUNT2` назначают адреса ячеек памяти для хранения промежуточных данных (текущих состояний, переменных циклов и т.п.).

Ассемблерные директивы `#DEFINE` задают строку, замещающую соответствующую метку, каждый раз, когда та будет встречаться в исходном тексте. В нашем случае эти директивы позволяют использовать символические имена, привязанные к схеме макета, вместо физических адресов соответствующих разрядов портов и регистров. При этом необходимо иметь в виду, что символы, которые определены директивой `#DEFINE`, не могут быть просмотрены симулятором. Поэтому для просмотра необходимо использовать физические адреса портов и регистров.

Директива `ORG 0x00` устанавливает стартовый адрес программного кода равным 0, т.е. соответствующим начальному состоянию счетчика команд МК после сброса. Команда `GOTO BEGIN` вместе с ассемблерной директивой `ORG 0x005` и меткой `BEGIN` обеспечивают переход на адрес памяти программ 0x005, начиная с которого и размещается основная часть программы. Это необходимо для того, чтобы обойти адрес 0x004, используемый в качестве вектора прерывания, и тем самым зарезервировать его для возможных будущих применений.

Затем с помощью команды `CALL IN IT PORTS` производится вызов подпрограммы инициализации портов. Вначале подпрограмма инициализации устанавливает в высокое (единичное) состояние выходные триггеры данных. Эта операция рекомендуется разработчиком МК для того, чтобы исключить неопределенность в состояниях регистров портов. Затем командой `BSF STATUS,RP0` производится переключение на банк 1 памяти данных, где расположены регистры управления направлением передачи информации `TRISA` и `TRISB`. С помощью команд `MOVLW 0x1C` и `MOVWF TRISA` линии `RA0` и `RA1` порта A настраиваются на вывод, а остальные - на ввод. Команды `MOVLW 0x01` и `MOVWF TRISB` настраивают линию `RB0` порта B на ввод, а остальные — на вывод. С помощью команды `BCF STATUS,RP0` производится возврат в банк 0, где располагаются необходимые для работы программы регистры и порты.

Поскольку в процессе работы с макетом перенастройка портов не производится, и введенных переменных достаточно для работы всех рассматриваемых учебных задач, они будут далее рассматриваться включенными по умолчанию в состав исходной программы `init.asm`. При написании учебных задач будет по возможности использоваться метод структурного программирования, при котором прикладная программа строится из некоторого набора программных модулей, каждый из которых реализует определенную процедуру обработки данных. При этом каждый из программных модулей имеет только одну точку входа и одну точку выхода. Введенные однажды программные модули могут использоваться под своим именем в других прикладных программах.

### 3. Программирование учебных задач

Начнем программирование учебных задач с написания программы, которая

считывает состояние кнопки SB1 и выводит его на светодиодный индикатор VD2 так, что не нажатому состоянию кнопки (высокому уровню сигнала на входе RA4) соответствует светящееся состояние светодиода, и наоборот.

;основная программа

```

LOOP
    CLRWDT                ;сброс сторожевого таймера
    CALL      GET_RA       ;вызов подпрограммы GET_RA
    CALL      SB1_VD2      ;вызов подпрограммы SB1_VD2
    GOTO      LOOP         ;переход к метке LOOP для
                           ;повторения процесса
;
GET_RA                    ;подпрограмма чтения
                           ;состояния порта А
    MOVF      PORTA,W      ;чтение состояния порта А в W
    MOVWF     TEMPА        ;пересылка W в TEMPА
    RETURN                    ;возврат из подпрограммы
;
SB1_VD2                  ;подпрограмма вывода на
                           ;светодиод VD2 состояния
                           ;кнопки SB1 (разряда 4 регистра
                           ;TEMPА)
    BTFSS     TEMPА.4      ;пропустить команду, если
                           ;TEMPА 4=1 (кнопка не нажата)
    GOTO      P0           ;перейти на P0
    BSF       VD2          ;зажечь светодиод VD2
P0
    BTFSC     TEMPА,4      ;пропустить команду, если
                           ;TEMPА,4=0 (кнопка нажата)
    GOTO      P1           ;перейти на P1
    BCF       VD2          ;погасить светодиод
P1
    RETURN
;

```

Основная программа содержит замкнутый цикл LOOP - GOTO LOOP, необходимый для периодического повторения цикла контроля состояния кнопки и вывода его на индикатор. Команда CLRWDT исключает влияние возможного сброса по переполнению сторожевого таймера на работу программы. Две следующие команды осуществляют вызов подпрограмм GET\_RA и SB1\_VD2. Первая из них (GET\_RA) вначале считывает текущее состояние порта А, которое помещается в рабочий регистр W. Поскольку рабочий регистр может потребоваться при исполнении других команд, его состояние записывается в регистр TEMPА, используемый здесь для временного хранения состояния порта А. Таким образом, после возврата из подпрограммы GET\_RA в разряде 4 регистра TEMPА содержится

информация о состоянии кнопки SB1: «1» - не нажата, «0» - нажата.

Подпрограмма SB1\_VD2 анализирует состояние разряда 4 регистра TEMPА и, в зависимости от него, зажигает или гасит светодиод. В системе команд МК PIC16F84 нет команд условного перехода, поэтому для организации проверки того или иного условия используются команды, позволяющие пропустить выполнение следующей команды программы, в зависимости от состояния определенного бита в заданном регистре (BTFSS и BTFSC). В частности, команда BTFSS TEMP,4 пропускает исполнение команды GOTO P0, если TEMP,4 = 1 (кнопка не нажата). Тем самым реализуется команда BSF VD2, которая зажигает светодиод VD2. Затем анализируется условие TEMP,4 = 0 (кнопка нажата) и, если оно имеет место, светодиод гасится.

Возможна более простая реализация заданного алгоритма, поскольку нажатое состояние кнопки исключает не нажатое (и наоборот), но представленный вариант более нагляден.

Рассмотрим более сложный вариант программы, предусматривающий зажигание светодиода VD2 только при следующем состоянии тумблеров и кнопок макета: SA1 = 1, SA2 = 1, SB1 = 1 и SB2 = 0.

;основная программа

LOOP

CLRWDT		;сброс сторожевого таймера
CALL	GET_RA	;вызов подпрограммы GET_RA
CALL	GET_RB	;вызов подпрограммы GET_RB
CALL	ZAG_1110	;вызов подпрограммы ZAG_1110
GOTO	LOOP	;переход к метке LOOP для
		;повторения процесса

;

GET\_RB

;подпрограмма чтения состояния  
;порта В

MOVF	PORTB,W	;чтение состояния порта В в W
MOVWF	TEMPB	;пересылка W в TEMPB
RETURN		

;

ZAG\_1110

;зажигает светодиод VD2 только  
;при следующем состоянии  
;тумблеров и кнопок макета:  
;SA1 = SA2 = SB1 = 1 и SB2 = 0

BTFSS	TEMPA,2	;пропустить команду, если
GOTO	P0	;TEMPA,2=1
BTFSS	TEMPA,3	;пропустить команду, если
GOTO	P0	;TEMPA,3=1
BTFSS	TEMPA,4	;пропустить команду, если
GOTO	P0	;TEMPA,4=1
BTFSC	TEMPB,0	;пропустить команду, если
GOTO	P0	;TEMPB,0=0
BSF	VD2	;зажечь светодиод VD2

```

        GOTO      P1
P0      BCF        VD2          ;погасить светодиод VD2
P1      RETURN
;
INCLUDE      GET_RA.ASM
;

```

Подпрограммы GET\_RA и GET\_RB помещают в регистры TEMPА и TEMPВ текущие состояния портов А и В, соответственно. Подпрограмма ZAG\_1110 анализирует состояния разрядов 2,3 и 4 регистра TEMPА и разряда 0 регистра TEMPВ, и при условии TEMPА,2,3,4 = 1,1,1 и TEMPВ,0 = 0, зажигает светодиод VD2. При невыполнении хотя бы одного из этих условий светодиод гасится.

Использование директивы INCLUDE GET\_PORTA.ASM позволяет включать уже отлаженные модули подпрограмм в текущую программу. Для того чтобы этой возможностью можно было воспользоваться, необходимо сохранять отлаженные модули в виде отдельных ассемблерных файлов.

Попробуем теперь использовать семисегментный индикатор для контроля состояния тумблеров макета. Вначале напишем программу, которая выводит на индикатор HL семисегментное изображение любого двоичного числа от 0b до 1111b в шестнадцатиричном представлении.

;основная программа

```



LOOP
    CLRWDT          ;сброс сторожевого таймера
    MOVLW0x0A       ;пересылка константы 0A в W
    CALL SEV_SEG     ;вызов подпрограммы SEVEN_SEG
    MOVWF PORTB      ;пересылка W в PORTB
    GOTO LOOP        ;переход к метке LOOP для
                    ;повторения процесса
;
SEV_SEG              ;подпрограмма обслуживания
                    ;семисегментного индикатора
    ANDLW0x0F        ;маскирование 4-х младших
                    ;разрядов W и обнуление 4-х
                    ;старших
    ADDWFPCL,F        ;сложение W с PCL и пересылка
                    ;результата в PCL
    RETLW 0x80        ;возврат из подпрограммы с 80 в W
    RETLW 0xF2        ;возврат из подпрограммы с F2 в W
    RETLW 0x48        ;возврат из подпрограммы с 48 в W
    RETLW 0x60        ;возврат из подпрограммы с 60 в W
    RETLW 0x32        ;возврат из подпрограммы с 32 в W

```

RETLW 0x25	;возврат из подпрограммы с 25 в W
RETLW 0x04	;возврат из подпрограммы с 04 в W
RETLW 0xF0	;возврат из подпрограммы с F0 в W
RETLW 0x00	;возврат из подпрограммы с 00 в W
RETLW 0x20	;возврат из подпрограммы с 20 в W
RETLW 0x10	;возврат из подпрограммы с 10 в W
RETLW 0x06	;возврат из подпрограммы с 06 в W
RETLW 0x8C	;возврат из подпрограммы с 8C в W
RETLW 0x42	;возврат из подпрограммы с 42 в W
RETLW 0x0C	;возврат из подпрограммы с 0C в W
RETLW 0x1C	;возврат из подпрограммы с 1C в W

;

Программа начинает свою работу с пересылки константы 0x0A в рабочий регистр W. Затем производится вызов подпрограммы обслуживания семисегментного индикатора SEV\_SEG. Работа подпрограммы SEV\_SEG начинается с маскирования 4-х младших разрядов W и обнуления 4-х старших. Тем самым из анализа исключаются старшие разряды передаваемого из рабочего регистра W числа. Затем маскированное содержимое регистра W добавляется к текущему состоянию младшего байта счетчика команд PCL, и результат помещается в PCL. Таким образом, производится дополнительное смещение счетчика команд на величину, которая была передана в рабочем регистре. Например, если было W=0, то содержимое счетчика команд не изменится, и будет выполнена следующая команда RETLW 0x80, которая вызовет возврат из подпрограммы с записью 0x80 = B'1000000' в регистр W. Если, как было в приведенной программе, W=0A, то к содержимому PCL будет добавлено число 0x0A, и произойдет дополнительное смещение на 10 шагов. В результате будет выполнена команда RETLW 0x10, которая вызовет возврат из подпрограммы с записью 0x10 = B'0001000' в регистр W.

После возврата из подпрограммы производится пересылка W в PORTB и отображение его состояния на семисегментном индикаторе HL. В частности, если W = 0, то при выводе 1000000b на порт B семисегментный индикатор покажет , а при W = A покажет . Таким образом, может быть отображено любое 4-разрядное двоичное число.

Метод прямого управления счетчиком команд, использованный в подпрограмме SEV\_SEG, может применяться для реализации табличной конвертации чисел. При этом необходимо иметь в виду, что данный метод не позволяет конвертировать более 256 значений в одной таблице. Кроме того, программа табличной конвертации должна целиком располагаться внутри 256-байтного блока во избежание переполнения младшего байта счетчика команд.

Используя подпрограмму SEV\_SEG, напишем теперь программу, которая читает состояния тумблеров SA1 и SA2 и выводит на индикатор соответствующее число.

;основная программа  
LOOP

```

CLRWDT                                ;сброс сторожевого таймера
CALL      GET_RA                       ;вызов подпрограммы GET_RA
RRF                                ;сдвиг вправо на один разряд
                                ;через перенос
RRF                                ;сдвиг вправо на один разряд
                                ;через перенос
ANDLW0x03                            ;маска на два младших разряда
CALL      SEV_SEG                     ;вызов подпрограммы SEVEN_SEG
MOVWF     PORTB                       ;пересылка W в PORTB
GOTO      LOOP                       ;переход к метке LOOP для
                                ;повторения процесса
;
INCLUDE   GET_RA.ASM
INCLUDE   SEV_SEG.ASM
;

```

Подпрограмма GET\_RA помещает в регистр TEMPА текущее состояние порта А. Таким образом, в разрядах 2 и 3 регистра TEMPА хранится текущее состояние тумблеров SA1 и SA2. Для того чтобы биты состояния тумблеров заняли позиции 0 и 1 регистра TEMPА, производится два сдвига вправо через перенос, причем результат второго сдвига помещается в регистр W. Затем накладывается маска на два младших разряда рабочего регистра и производится вызов подпрограммы SEV\_SEG. После выхода из подпрограммы результат подается на порт В и отображается на индикаторе.

Рассмотрим теперь программы, работающие в реальном масштабе времени, т.е. выдающие сигналы определенной длительности и частоты следования, либо учитывающие временные параметры входных сигналов. Основным элементом таких программ является подпрограмма формирования временной задержки. Рассмотрим один из возможных вариантов такой подпрограммы с использованием программных методов формирования задержки, т.е. без применения встроенного таймера.

```

;основная программа
    MOVLW0xL                           ;пересылка константы H'L' в W
    CALL      DELAY                     ;вызов подпрограммы DELAY
;
    DELAY                                ;подпрограмма формирования
                                ;задержки времени
    MOVWF     COUNT1                   ;загрузка W в регистр COUNT1
LOOPD
    DECFSZ    COUNT1,F                 ;декремент COUNT1
    GOTO      LOOPD                   ;повторение цикла H'L' раз
    RETURN                                ;возврат из подпрограммы
;

```

Основная программа производит вызов подпрограммы DELAY с неко-

торой константой L в рабочем регистре W, определяющей число внутренних циклов подпрограммы. Подпрограмма DELAY начинает свою работу с загрузки содержимого рабочего регистра в регистр пользователя COUNT1. Команда DECFSZ COUNT1,F уменьшает на единицу содержимое регистра COUNT1 и проверяет его на равенство нулю. Нулевое состояние регистра COUNT1 приводит к выходу из цикла и возврату из подпрограммы. Для исполнения каждого внутреннего цикла требуется три машинных цикла МК (1 цикл на исполнение команды DECFSZ при ненулевом результате и 2 цикла на каждую команду GOTO). Выход из подпрограммы DELAY потребует 4-х циклов (2 цикла на исполнение команды DECFSZ при нулевом результате и 2 цикла на RETURN). Если добавить к этому еще 4 цикла, необходимых для загрузки константы в рабочий регистр, вызова подпрограммы и загрузки регистра пользователя COUNT1, то общее время исполнения подпрограммы DELAY (задержка) составит

$$T_D = 4 + 3*(L - 1) + 4 = 5 + 3*L \text{ циклов,}$$

где L — константа, переданная через рабочий регистр в подпрограмму DELAY.

При тактовой частоте  $f_{osc} = 2 \text{ МГц}$  время цикла равно  $t_{ц} = 2 \text{ мкс}$ , поэтому при загрузке  $L = H'00' = .0$  максимальный формируемый интервал времени составит 1,55 мс. Такой результат связан с тем, что команда DECFSZ сначала декрементирует содержимое регистра ( $H'00' - 1 = H'FF$ ), а затем уже анализирует результат.

Минимальный формируемый интервал времени составит при тех же условиях 5 циклов или 10 мкс. Для получения такого интервала необходимо перед вызовом подпрограммы DELAY загрузить в рабочий регистр число 0x01.

Для расширения верхней границы формируемых временных интервалов, а также с целью повышения удобства работы с подпрограммой, можно добавить в цикл LOOPD одну или несколько дополнительных команд, в качестве которых чаще всего используется команда NOP. Для примера рассмотрим подпрограмму формирования задержки времени DELAY\_C

```

;
    DELAY_C                                ;подпрограмма формирования
                                           ;задержки времени (вариант C)
    MOVWF      COUNT1                     ;загрузка W в регистр COUNT1
LOOPD
    NOP                                           ;пустая команда
    DECFSZ     COUNT1,F                     ;декремент COUNT1
    GOTO       LOOPD                         ;повторение цикла H'L' раз
    RETURN                                         ;возврат из подпрограммы
;

```

Общее время исполнения подпрограммы DELAY\_C, включая ее вызов, составит

$$T_D = 4 + 4*(L - 1) + 4 = 4 + 4*L \text{ циклов.}$$

При тактовой частоте  $f_{osc} = 2\text{МГц}$  и загрузке константы  $L = \text{H}'\text{F9}' = .249$  формируемый интервал времени составит ровно 2 мс. Уменьшение константы на единицу уменьшает формируемый временной интервал на 8 мкс. В частности, при  $L = .124$  образуется задержка в 1 мс.

Для формирования больших задержек времени, лежащих в диапазоне долей и единиц секунд, такой подход неудобен. В этом случае используются вложенные циклы, как показано в следующем примере.

```

;основная программа
    MOVLW0xL          ;пересылка константы H'L' в W
    CALL              DELAY_D      ;вызов подпрограммы DELAY_D
;
DELAY_D                  ;подпрограмма формирования
                        ;большой задержки времени
(вариант D)
    MOVWF             COUNT2      ;загрузка W в регистр COUNT2
    CLRF              COUNT1      ;сброс содержимого регистра
                                ;COUNT1
LOOPD
    DECFSZ             COUNT1,F    ;декремент COUNT1
    GOTO               LOOPD      ;повторение цикла 256 раз
    CLRWDT             ;сброс сторожевого таймера
    DECFSZ             COUNT2,F    ;декремент COUNT2
    GOTO               LOOPD      ;повторение цикла H'L' раз
    RETURN             ;возврат из подпрограммы
;

```

Время исполнения внутреннего цикла подпрограммы DELAY\_D составляет  $3 \cdot 256 + 4$  машинных циклов МК, поэтому общая задержка составит

$$T_D = 5 + (3 \cdot 256 + 4) \cdot L \text{ циклов.}$$

При тактовой частоте  $f_{osc} = 2\text{МГц}$  время цикла равно  $t_{\text{ц}} = 2 \text{ мкс}$ , поэтому при загрузке  $L = \text{H}'00' = .0$  максимальный формируемый интервал времени составит около 0,4 с.

Поскольку формируемый интервал времени достаточно велик, во внешний цикл включена команда сброса сторожевого таймера.

Интервал времени 0,4 с не совсем удобен для получения задержек времени, кратных секунде, поэтому рассмотрим еще один вариант подпрограммы формирования больших задержек времени с дополнительной командой NOP во внутреннем цикле.

```

;
DELAY_E                  ;подпрограмма формирования
                        ;большой задержки времени
(вариант E)

```



MOVWF	COUNT2	;загрузка W в регистр COUNT2
CLRF	COUNT1	;сброс содержимого регистра
		;COUNT1
LOOPD		
NOP		;пустая команда
DECFSZ	COUNT1,F	;декремент COUNT1
GOTO	LOOPD	;повторение цикла 256 раз
CLRWDT		;сброс сторожевого таймера
DECFSZ	COUNT2,F	;декремент COUNT2
GOTO	LOOPD	;повторение цикла H'L' раз
RETURN		;возврат из подпрограммы
;		

Время исполнения внутреннего цикла подпрограммы DELAYE составляет  $4*256 + 4$  машинных циклов МК, поэтому общая задержка составит

$$T_D = 5 + (4*256 + 4)*L \text{ циклов.}$$

При тактовой частоте  $f_{osc} = 2\text{МГц}$  и при загрузке  $L = H'F3' = .243$  формируемый интервал времени составит около 0,5 с при погрешности не более 0,2%. Если необходима более высокая точность, можно вставить необходимое количество пустых операций во внешний цикл формирования задержки.

Рассмотрим далее несколько программ с использованием подпрограмм формирования задержки времени. Начнем с написания программы, которая подает звуковой сигнал на динамик BA1 при нажатии на кнопку SB1. Динамик будет звучать только в том случае, если на выход RA0 будет подан периодически изменяющийся сигнал. Для того чтобы звук был хорошо слышен, его частота должна находиться вблизи максимума слышимости человеческого уха. Выберем частоту звучания равной 1 КГц, что соответствует периоду следования импульсов сигнала 1мс.

;основная программа

LOOP		
CLRWDT		;сброс сторожевого таймера
CALL	GET_RA	;вызов подпрограммы
GET_ORTA		
CALL	SB1_BA1	;вызов подпрограммы SB1_BA1
GOTO	LOOP	;переход к метке LOOP для
		;повторения процесса
;		
SB1_BA1		;подпрограмма подачи звука на
		;динамик BA1 при нажатии на
		;кнопку SB1
BTFSC	TEMPA,4	;пропустить команду, если
		;TEMPA,4=0 (кнопка нажата)
GOTO	B0	;перейти на B0

```

BSF          BA1          ;подача высокого уровня на RA0
MOVLW0x3E
CALL         DELAY_C      ;вызов подпрограммы DELAY_C
BCF          BA1          ;подача низкого уровня на RA0
MOVLW0x3E
CALL         DELAY_C      ;вызов подпрограммы DELAY_C
B0
RETURN
;
INCLUDE      GET_RA.ASM
INCLUDE      DELAY_C.ASM
;

```

Как и раньше, подпрограмма GET\_RA считывает текущее состояние порта А, которое затем передается в регистр TEMPA. Подпрограмма SB1\_BA1 анализирует состояние разряда 4 регистра TEMPA и, в зависимости от результата, озвучивает динамик BA1 или нет. Необходимая выдержка линии RA0 в единичном и нулевом состояниях обеспечивается подпрограммой DELAY\_C с параметром  $L=H'3E' = .62$ . Это соответствует времени задержки около 0,5 мс, что и дает в результате необходимую частоту следования сигнала 1 КГц.

Рассмотрим далее программу, которая заставляет мигать светодиод VD2 при нажатии на кнопку SB1. Для того чтобы мигания были хорошо видны, выберем их частоту равной 1 Гц.

;основная программа  
LOOP

```

CLRWDT      ;сброс сторожевого таймера
CALL        GET_RA      ;вызов подпрограммы GET_RA
CALL        SB1_VD2M     ;вызов подпрограммы
                        ;SB1_VD2M
GOTO        LOOP        ;переход к метке LOOP для
                        ;повторения процесса
;
SB1_VD2M     ;подпрограмма мигания
            ;светодиода VD2 при нажатии
            ;на кнопку SB1
BTFSC      TEMPA,4      ;пропустить команду, если
                        ;TEMPA,4=0 (кнопка нажата)
GOTO       V0           ;перейти на V0
BSF        VD2          ;зажечь светодиод VD2
MOVLW0xF3   ;пересылка константы
            ;H'F3' = .243 в W
CALL       DELAY_E      ;вызов подпрограммы DELAY_E
BCF        VD2          ;погасить светодиод

```

```

                MOVLW0xF3                ;пересылка константы
                                           ;H'F3' = .243 в W
V0    CALL      DELAY_E                ;вызов подпрограммы DELAY_E

        BTFSS    TEMPA,4                ;пропустить команду, если
                                           ;ТЕМРА,4=1 (кнопка не нажата)
        GOTO     V1                    ;перейти на V1
V1    BCF        VD2                    ;погасить светодиод
        RETURN
;
        INCLUDE  GET_RA.ASM
        INCLUDE  DELAY_E.ASM
;

```

Программа работает почти так же, как и предыдущая. Первое отличие заключается в том, что светодиод принудительно гасится при не нажатой кнопке. Второе отличие заключается в величине интервала времени, который составляет здесь 0,5 с и формируется подпрограммой DELAY\_E.

Подпрограммы формирования задержки времени могут быть также полезны при работе с такими внешними источниками сигналов, как тумблеры, кнопки, переключатели и т.п. Дело в том, что все механические коммутаторы имеют одно негативное свойство, известное как «дребезг» контактов, которое обусловлено механическими колебаниями контактов при их замыкании и размыкании. Длительность колебаний составляет обычно несколько миллисекунд, в течение которых на вход МК может поступать пачка импульсов вместо идеального перепада.

Аппаратные способы борьбы с «дребезгом» контактов основаны на использовании RS-триггеров, одновибраторов или триггеров Шмитта. В устройствах на основе МК подавление «дребезга» контактов обычно осуществляется программными способами, которые основаны на повторном считывании состояния линии порта через определенное время.

В качестве примера рассмотрим «бездребезговый» вариант подпрограммы чтения состояния порта А.

```

;
GET_RAD                ;подпрограмма чтения состояния
                        ;порта А в регистр ТЕМРА
                        ;с подавлением "дребезжания"
DD
        MOVF     PORTA,W                ;чтение состояния порта А в W
        ANDLW0x1C                        ;наложение маски b'00011100'
                                           ;на неиспользуемые биты W
        MOVWF    TEMPA                  ;пересылка W вТЕМРА
        CLRWDT                        ;сброс сторожевого таймера WDT
        MOVLW0x0A                        ;пересылка константы

```

CALL	DELAY_E	;H'0A' = .10 в W
MOVF	PORTA,W	;вызов подпрограммы DELAY_E
ANDLW0x1C		;чтение состояния порта A в W
		;наложение на W
		;маски b'00011100'
SUBWF	TEMPA,W	;вычитание W из TEMPA
BTFSS	Z	;пропустить команду, если
		;результат нулевой
GOTO	DD	;перейти на метку DD
RETURN		
;		
INCLUDE	DELAY_E.ASM	
;		

Суть работы подпрограммы заключается в повторном чтении состояния порта A спустя некоторое время после предыдущего и сравнении его с прежним значением. Константа H'0A' = .10, пересылаемая в регистр W перед вызовом подпрограммы DELAY\_E, обеспечивает значение задержки времени около 20 мс - этого, как правило, достаточно для завершения переходных процессов при переключении механических коммутаторов. Маскирование неиспользуемых разрядов порта повышает надежность работы подпрограммы. Сброс сторожевого таймера перед вызовом подпрограммы задержки нужен для исключения сброса МК между двумя процедурами опроса порта A.

Рассмотрим теперь работу программы, которая использует некоторые из разработанных ранее подпрограмм. Пусть целью работы программы является подсчет числа нажатий на кнопку SB1 с выводом результата на семисегментный индикатор в шестнадцатиричном коде.

;основная программа

CLRF	COUNT3	;сброс счетчика нажатий
LOOP		
CLRWDT		;сброс сторожевого таймера
CALL	GET_RAD	;вызов подпрограммы GET_RAD
BTFSC	TEMPA,4	;проверка нажатия SB1
GOTO	LOOP	;если не нажата – возврат
		;на метку LOOP
INCF	COUNT3,F	;инкремент счетчика
MOVF	COUNT3,W	;пересылка содержимого
		;счетчика в рабочий регистр
CALL	SEV_SEG	;вызов подпрограммы
		;SEVEN_SEG
MOVWF	PORTB	;пересылка W в PORTB
TEST		
CALL	GET_RAD	;вызов подпрограммы GET_RAD
BTFSS	TEMPA,4	;проверка нажатия SB1
GOTO	TEST	;если еще нажата – возврат

```

;на метку TEST
GOTO      LOOP      ;возврат на метку LOOP
;
INCLUDE    GET_RAD.ASM
INCLUDE    SEV_SEG.ASM
;

```

Приведенные в главе программы не охватывают и малой доли возможностей, которые предоставляет даже такой простой макет, как изображенный на рис.1. Однако их освоение, надеюсь, будет полезным для начинающих пользователей PIC-контроллеров.

Министерство образования Республики Беларусь

*Учреждение образования*

Белорусский государственный университет информатики и  
радиоэлектроники

кафедра радиоэлектронных средств

И.Н. Цырельчук

Практическое занятие №4

**Программное обеспечение микропроцессорных систем**

по курсу «Микропроцессорные системы и их применение»

Минск 2006

## **Разработка программного обеспечения микропроцессорных систем.**

Создание программного обеспечения трудоемкая и дорогостоящая процедура. Для уменьшения стоимости ПО его надо строить так, чтобы иметь возможность многократно использовать в МПС разных видов. Системы в которых применено общее программное обеспечение называются программно совместимыми и для них характерно :

1. единые наборы команд
2. единая форма представления данных
3. единые способы адресации

Разработка программ решения задач, возникших при работе, включает следующие этапы :

1. постановка задачи
2. выбор и разработка алгоритма решения
3. представление алгоритма в виде структурной схемы
4. составление программы

Постановка задачи включает в себя составление набора характеристик задачи, определение входных параметров, методов их обработки, выходных параметров и форм представления результатов.

Сущность разработки алгоритма вытекает из его определения.

Алгоритм - это предписание по выполнению в определенном порядке совокупности элементарных операций с целью решения задачи.

Алгоритмизация задачи - это процесс создания алгоритма.

Свойства алгоритма:

1. определенность (однозначность толкование элементов, предписывающих выполнение операции);
2. массовость (возможность использования для решения широкого круга задач с разными исходными данными);
3. результативность (возможность получения результата при допустимых исходных данных, за конечное число шагов).

Существует три типа структурных схем алгоритмов:

1. системная схема
2. основная схема
3. детальная схема

Системная схема показывает, какие устройства ввода - вывода нужны для решения задачи. Само решение в схеме представлено одним блоком.

Основная схема описывает в общих чертах, но достаточно подробно решение поставленной задачи.

Детальная схема - подробная схема, содержащая в блоке одну или две команды данного МП.

Составление программ осуществляется на машинном языке ассемблера или на языке высокого уровня.

Машинный язык - это представление команд в двоичном коде.

Символическое обозначение команд, адресов и операторов называется ассемблером.

В зависимости от этапа разработки различают следующие виды программ:

1. исходную
2. объектную
3. рабочую

Исходная - это программа на ассемблере или на языке высоко высокого уровня.

Объектная - это программа, преобразованная из исходной при помощи транслятора.

Перевод ассемблера на машинный язык осуществляется в ПЗУ. Перевод программы написанной на языке высокого уровня в машинный код осуществляется при помощи программы компилятора. После проверки и корректировки получают рабочую программу. Эти программы составляют ПО пользователя. Кроме этого в МП существует системное ПО, к которому относятся:

- программы, осуществляющие перевод с ассемблера
- программа компилятор
- загрузчик
- отладчик
- программа редактирования текста

## **Правила составления программ на ассемблере.**

Программа представляется в виде последовательных выражений, каждое из которых имеет до четырех полей:

- метка
- оператор
- операнд
- комментарий

Метки используются в выражениях, на которые могут быть ссылки в программе (команды управления)

Оператор - характеризует действия предписанные командой.

Операнд - это данные, над которыми выполняются операции. В поле операнда может быть помещен операнд или его адрес. Если в поле занято два операнда, то их отделяют запятой.

Комментарий вводится программистом для удобства.



Пример:

Метка	Оператор	Операнд	Комментарий
Начало :	Переслать	В	; Контроль кода
	Логически умножить Условный переход	С Начало	

Начало - символический адрес. Конкретное значение адреса зависит от размещения программы в памяти.

Переслать В - операция передачи содержимого регистра в аккумулятор.

Контроль кода - комментарий указывающий на начало данного участка программы определенного содержания.

При записи программы соблюдаются следующие требования:

1. После метки ставится двоеточие
2. Перед текстом комментария ставится точка с запятой
3. Поле команд разделяют пробелом
4. В командах с непосредственной адресацией в поле оператора после номера регистра или ячейки памяти ставится константа, которая должна быть загружена в этот регистр или ячейку памяти. Пред константой ставится запятая

Пример:

загрузить непосредственно в регистр константу -5

MVI B ,-5

5. После чисел, представленных в двоичной форме исчисления в скобках должна стоять буква В. Если в шестнадцатеричной системе, то буква Н. В десятичной ничего не пишется.

6. Признаком индексной адресации является буква Х, которая ставится в поле операнда после адреса и отделяется от адреса запятой.

Пример:

ADD a , X

к содержимому аккумулятора прибавить содержимое ячейки памяти с адресом Х. Результат помещается в аккумулятор.

## **Подпрограммы.**

Подпрограмма - это часть программы, выполняющая определенное логическое завершённое действие.

Существуют два типа подпрограмм: - открытая -закрытая

Открытая - это часть программы, которая допускает возможность внесения изменений. Совокупность команд этой подпрограммы включается в нужные места основной программы. Использование открытых подпрограмм не требует включения в основную программу специальных команд обращения к подпрограмме, так как команды подпрограммы последовательно вписываются в нужные участки основной программы.

Недостаток открытых подпрограмм - это высокий объем памяти, занимаемый записанными подпрограммами.

Преимущество - высокое быстродействие, так как ПК не производит дополнительных действий.

Для уменьшения записи программ применяются макрокоманды.

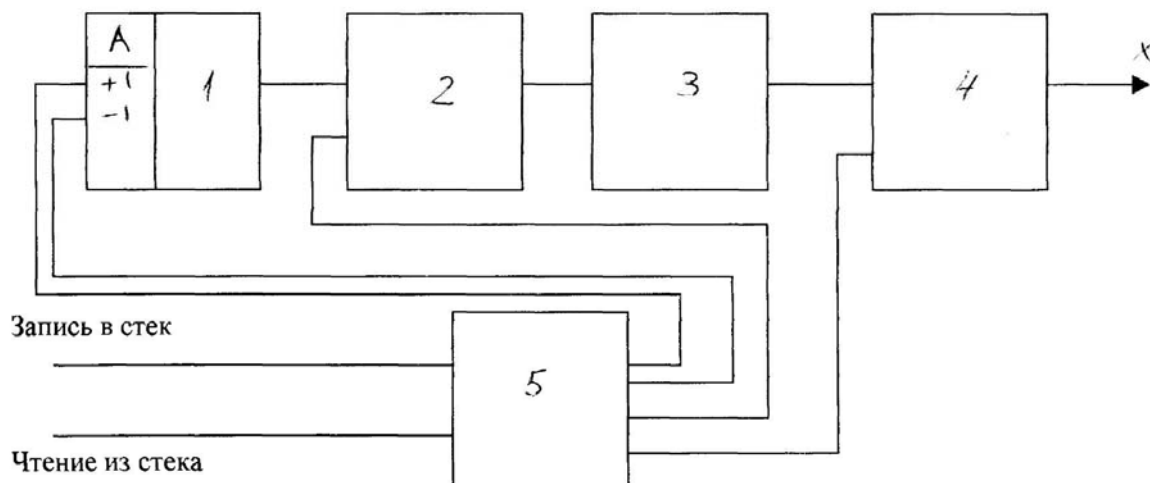
Закрытая подпрограмма - используется для уменьшения объема памяти, занимаемого программой в которой многократно повторяется последовательность команд. Они хранятся отдельно от основной программы и составляются один раз.

Когда в основной программе, возникает необходимость обращения к подпрограмме, происходит обращение к ней и вызов. В результате управление передается подпрограмме и начинается ее выполнение, после чего происходит возврат в ту точку программы, из которой был сделан вызов и возобновляется выполнение основной программы.

Запоминание номера ячейки возврата в основную программу осуществляется в стеке, при использовании которого обращение к подпрограмме обеспечивает автоматическую работу с вложенными подпрограммами. При этом в стеке происходит запоминание для каждого уровня обращения к подпрограмме.

Средства разработки подпрограмм.

Стек - динамический последовательный список данных доступ к которому возможен с одного конца. Запоминание в стеке называется проталкиванием в стек, а выборка - выталкиванием из стека.



- 1-указатель стека;
- 2-дешифратор адреса (ДшА);
- 3-матрица памяти;
- 4-регистр слов;
- 5-машинный блок управления (МБУ).

Выборка одной из ячеек матрицы памяти осуществляется через ДшА по адресу находящемуся на реверсивном счетчике адреса - указателе стека. Начальное значение адреса поступает в указатель стека на вход А. В процессе работы состояние указателя стека при каждой записи уменьшается, а при чтении увеличивается на единицу.

Машинный блок управления осуществляет управление режимами записи - чтения путем выработки соответствующих управляющих сигналов.

При записи информации входное слово поступает на регистр слов и записывается затем в память по адресу, который в данный момент времени был установлен в указателе стека.

С небольшой задержкой после записи информации содержимое указателя стека уменьшается на единицу, подготавливаясь к следующей записи. Указатель стека постоянно указывает на свободную ячейку в которой должно быть записано очередное слово.

При чтении машинный блок управления сначала вырабатывает сигнал, увеличивающий указатель стека на единицу, а затем сигнал чтения информации из памяти. В результате на выходе стека появляется слово X которое было записано последним.

Широкое использование стека объясняется следующим:

1. Система команд МП не содержит сложных операций, поэтому многие стандартные процедуры реализуются в МП программно. При каждом обращении к подпрограмме значительное время уходит на запоминание и восстановление состояний внутренних регистров МП. Так как обращение к подпрограмме встречается очень часто, то используя для сохранения внутреннего состояния МП магазинную память можно значительно уменьшить время решения задач. Это объясняется отсутствием в ходе команд записи - чтения стека - адресного поля, а так же уменьшения разрядности и времени выполнения команд.

2. Одним из основных применений МП является управление разными процессами и объектами в реальном времени. Характерное свойство работы в этом режиме - постоянное взаимодействие системы управления и ВУ через систему прерываний, что так же требует временного запоминания состояния МП. Уменьшение времени записи - чтения из стека обеспечивает более быструю реакцию МП на сигналы от ВУ.

3. Сокращение разрядности команд. Обращение к памяти при частых переходах к подпрограмме приводит к значительной экономии памяти.

## **Основные понятия в области автоматизации и программирования.**

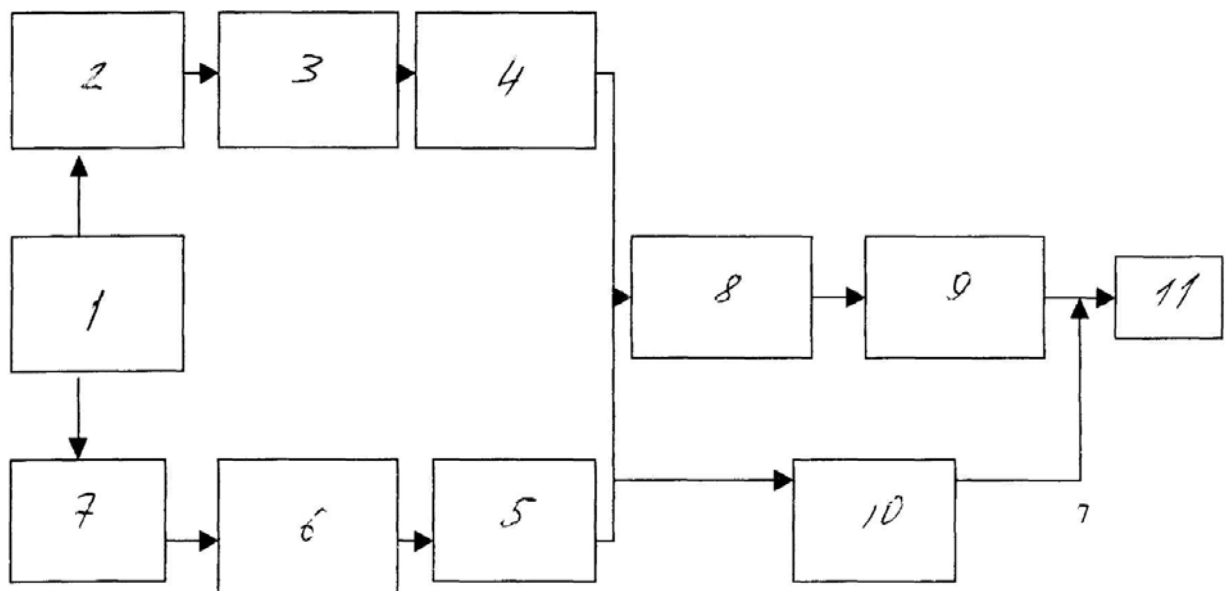
Для ускорения обеспечения разработки и отладки ПО микропроцессорных систем широко используются специальные программы называемые системными. Если системные программы реализованы на той же МПС для которой они вырабатывают ПО они называются резидентными, в противном случае - кросс программами. Рассмотрим основные системные программы.

### **Загрузчик.**

Загрузчик - это системная программа с помощью которой объектная программа вводится в память МПС. Если загруженный объектный код является абсолютным, то он записывается по адресу определенному при ассемблировании . Эта операция выполняется абсолютным загрузчиком. Он может использоваться как самостоятельная единица, но часто входит в отладочный монитор. Если объектный код является перемещаемым, т.е. перемещается из одной области памяти в другую, то загрузчик должен осуществить преобразование всех относительных адресных ссылок в абсолютные и осуществить загрузку полученной абсолютной объектной программы в требуемую область памяти.

При разработке больших программ обычно удобно разбить программу на части или модули, которые компилируются отдельно. Полученные перемещаемые объектные модули затем объединяются в общую программу. Объединение объектных модулей осуществляется программой- редактор связей, результат которой заносится в память с помощью загрузчика.

Последовательность исполнения системных программ при разработке МПС имеет вид:



- 1-редактор текста,
- 2-исходная программа на языке высокого уровня;
- 3-компилятор с языка высокого уровня;
- 4,5-перемещаемый объектный модуль;
- 6-ассемблер;
- 7-исходная программа на ассемблере;
- 8-редактор связей;
- 9-загрузчик;
- 10-связывающий загрузчик;
- 11-МПС.

Связывающий загрузчик одновременно выполняет функции редактора связей и заносит результирующую программу в память.

### **Отладчик и редактор текста.**

При пробном выполнении программы могут быть обнаружены ошибки. Назначение отладчика или отладочного монитора состоит в ускорении и облегчении поиска и исправления ошибок в объектной программе. Монитор обеспечивает возможность останова выполнения отлаживаемой объектной программы в любой заданной точке. После останова в программе, монитор позволяет выводить на экран или печать содержание регистров и памяти МП, модифицировать их содержимое, в чем и заключается отладка. Отладочные мониторы позволяют выполнять программу и в пошаговом режиме, т.е. с остановом после выполнения каждой команды.

Отладочный монитор должен находиться в памяти МПС вместе с отлаженной программой.

Редактор текста - это системная программа, которая в соответствии с командами программиста, которые вводятся с терминала, вносит исправления в текст исходной программы, находящейся в памяти ПК.

Для хранения исходной программы в редакторе предусматривается специальный буфер. Если вся исходная программа не помещается в буфер одновременно, то ее редактирование осуществляется по частям. Исправленная исходная программа выдается на магнитный носитель для дальнейшего исполнения

Обычно редактор позволяет удалять строку или строки, изменять порядок строк, стирать информацию в буфере, вводить сегмент программы в буфер, выводить его, добавлять текст в пустой или не полностью заполненный буфер, находить в тексте строки совпадающие с заданной и выдавать их номера на печать или заменять их на другие, изменять знаки в строке.

Программный отладчик работает с объектными программами. Программа редактор - с исходными.

### **Ассемблер.**

Ассемблер - это машинно-ориентированный язык программирования, в котором используется мнемонические обозначения команд, отражающие их функции и символические имена переменных.

Например в МП КР580ИК80А команда занесения ячейки ЗУ с адресом 15 в накопитель имеет код

0011 1010 0000 1111 0000 0000

В программе на ассемблере эта команда записывается

LDA15

Или

LDA ADR,

если ранее символическое имя ADR было присвоено ячейке с адресом 15. Мнемоника LDA расшифровывается как Load to Accumulator (загрузить в накопитель).

Системную программу трансляции исходной программы на языке ассемблера в коды команд МП (объектную программу) называют программой ассемблера. При трансляции в объектную программу каждая команда ассемблера преобразовывается в одну команду процессора. Но для хранения исходной программы на ассемблере требуется на порядок больше места, чем для хранения соответствующей собственной программы.

Современные ассемблеры содержат макропроцессор, т.е. такую системную программу, которая позволяет включать в разработанную программу группу команд, с помощью присвоения ей мнемоники.

Параметризованные макрокоманды макроассемблера увеличивают эффективность программирования и читабельность программ.

## **Трансляторы с языков высокого уровня.**

Языки высокого уровня упрощают написание программы. В качестве операторов используются специальные ключевые слова английского алфавита. Одному оператору на языке высокого уровня соответствует до нескольких десятков команд объектного кода в результате чего уменьшается трудоемкость написания программы.

Трансляторы делят на компиляторы и интерпретаторы.

Компилятор - это система программного транслирования исходных программ на языке высокого уровня в объектный код. Компиляторы являются многопроходными, т.е. требуют для работы несколько просмотров исходного текста программы. При большом объеме программы ее текст на языке высокого уровня занимает в ЗУ меньше места, чем объектный код. При специальном кодировании операторов языка высокого уровня программу можно не транслировать в объектный код, а вводить построчно и одновременно выполнять. При этом отпадает необходимость хранения в памяти объектного кода всей программы.

Интерпретатор - это системная программа, обеспечивающая выполнение наборов команд в соответствии с операторами исходной программы.

Компилятор транслирует всю программу, а затем выполняет ее. Интерпретатор это делает построчно.

Для хранения программы используют ЗУ. Компиляторы и ассемблеры могут генерировать код программы абсолютном и перемещаемом форматах.

В первом случае в коды команд заносятся абсолютные адреса операндов и переходов. Во втором - по отношению к базовым адресам.

Перемещаемая объектная программа может быть загружена для исполнения в любую часть памяти. Для этого программой загрузчиком выполняется относительная коррекция адресов.

## **Эмулятор.**

Это системная программа для ПК, моделирующая выполнение команд МПС, даже при отсутствии самих систем. Моделирующие программы часто предоставляют некоторые виды диагностической информации недоступной при исполнении отладочной программы, такие как указание о переполнении стека или попытке записи в ячейку ПЗУ.

Эмулятор позволяет оперировать и выводить на дисплей содержимое памяти и регистров МП, устанавливать контрольные точки, в которых программа может быть приостановлена при достижении некоторого адреса и задавать листинг с печатью каждой программы.



Моделирующая программа часто дает информацию о времени выполнения программы, например, число команд и машинных тактов выполненных от начала работы программы до её останова.

Моделирующие программы различных производителей отличаются своими возможностями, но не могут заменить отладку самой программы. Это объясняется тем, что специфические временные соотношения, возникающие при взаимодействии технических средств МПС управления полностью смоделировать невозможно.

Министерство образования Республики Беларусь

*Учреждение образования*

Белорусский государственный университет информатики  
и радиоэлектроники

кафедра радиоэлектронных средств

И.Н. Цырельчук

Практическое занятие №5

**Микропроцессорная система управления роботами**

по курсу «Микропроцессорные системы и их применение»

Минск 2006

## **МП система управления роботами**

Сердце работа МП система, она перерабатывает всю поступающую информацию и выдает управляющие воздействия на приводы. Переработка информации и выдача управляющих воздействий осуществляется с помощью программ, которые легко модифицируются.

Различают три вида роботов:

программные;

адаптивные;

интеллектуальные.

Программные имеют СУ которая может переналаживаться на различные ручные операции. После переналадки они повторяют одну и ту же программу, в строго определенной обстановке, с определенно расположенными предметами.

Адаптивные роботы могут самостоятельно в большей или меньшей мере ориентироваться вне строю определенной обстановки, приспосабливаясь к ней.

Интеллектуальные могут воспринимать и распознавать обстановку, строить модель среды, автоматически принимать решения о дальнейших действиях и выполнять его, могут изменять свое поведение и самообучаться.

В качестве примеров автоматических манипуляторов используемых при изготовлении РЭС можно привести автоматическую установку контроля параметров ИС, автоматическую установку установки радио элементов на печатную плату.

МПСУ такими манипуляторами весьма подобны друг на друга. Они ориентированны на выдачу управляющих воздействий по перемещению рабочего органа к точке с определенными координатами, подъем или опускание исполнительного органа, перемещение рабочего органа по определенной траектории в точку с определенными координатами.

Рассмотрим общие принципы проектирования МПСУ роботом на примере МПСУ чертежным графическим автоматом.

Чертежный графический автомат предназначен для вычерчивания графиков, схем, чертежей и другой графической документации в соответствии с программой, вводимой в устройство управления.

Вычерчиваемое на чертежном автомате изображение разбивается на простейшие элементы: отрезки прямых и дуги окружности. Более сложные кривые аппроксимируются с помощью указанных элементов. Каждому отрезку прямой и каждой дуге окружности соответствует один кадр. Такой кадр включает код, определяющий вид вычерчиваемой линии, т.е. команду (ПРЯМАЯ или ДУГА), номер одного из двух имеющихся перьев, которые могут иметь разную толщину линии, положение пера: (поднято, опущено), тип линии: (непрерывная, штриховая, штрихпунктирная) и числовую информацию.

Кадр информации замыкает строка КОНЕЦ КАДРА. После считывания кода КОНЕЦ КАДРА чертежный автомат останавливает

ввод и отрабатывает введенный кадр. Если следующий кадр не содержит первых двух строк, то предполагается, что код команды и тип линии не изменились, при необходимости изменить предыдущую команду на новую вслед за признаком КОНЕЦ КАДРА должна следовать строка с кодовой комбинацией КОНЕЦ КОМАНДЫ. Младший разряд этой строки служит для контроля правильности ввода информации. Его значение формируется так, чтобы суммарное число, начиная со строки, задающей код команды, было четным. Если при считывании информации обнаруживается ошибка по четности, дальнейшая работа прекращается. Поперечное и продольное перемещение чертежной головки обеспечивают два шаговых двигателя ШДХ и ШДУ. Каждый поступивший на ШД импульс обеспечивает перемещение головки на 0,1 мм. Максимальная частота импульсов не должна превышать 300 Гц. Управление каждым из двух имеющихся перьев осуществляется с помощью соленоида и усилителя мощности. При подаче на усилитель мощности высокого уровня напряжения, через обмотку соленоида протекает ток и перо опускается. После подачи на соленоид управляющего воздействия должна быть предусмотрена задержка для срабатывания соленоида. В данном случае задержка равна 0,4 сек.

Аппроксимация вычерчиваемых линий осуществляется по оценочным функциям. Рассмотрим для примера аппроксимацию отрезка прямой при перемещении по одной оси

Начальное положение пера примем за начало координат, уравнение имеет<sup>1</sup> вид:

$$y = (Dy / Dx) x$$

Оценочная функция имеет вид:

$$\Gamma = y - (Dy / Dx) x$$

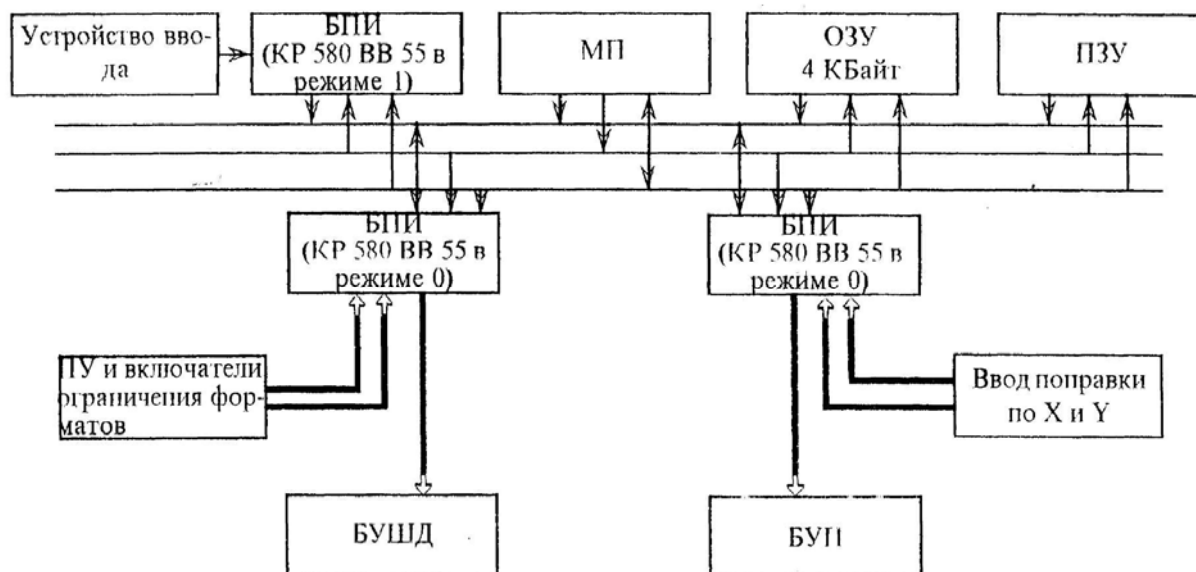
На интерполируемом отрезке прямой функция  $\Gamma = 0$ , выше этой прямой  $\Gamma > 0$ , ниже  $\Gamma < 0$ .

АЛУ ритм, основанный на вычислении оценочной функции, сгонг в следующем:

1. Для текущих значений  $x_i$  и  $y_i$  вычисляется оценочная функция  $\Gamma(x_i, y_i) = y_i - (Dy/Dx)x_i$ ,
2. Если функция  $\Gamma(x_i, y_i) < 0$ , то шаг /делается по координате У.
3. Если  $\Gamma(x_i, y_i) \geq 0$ , то шаг делается по координате Х.
4. Вычисляется новое значение оценочной функции, и процесс повторяется, вычисление этих операций накапливается при попадании в точку  $Dy, Dx$ .

При интерполяции других элементов чертежа используются оценочные функции другого вида.

Структурная схема МПСУ чертежным роботом и упрощенная схема алгоритма:



Режим "1" подтверждение готовности;

Режим "0" программно-управляемая подача данных;

БП.И блок параллельного интерфейса;

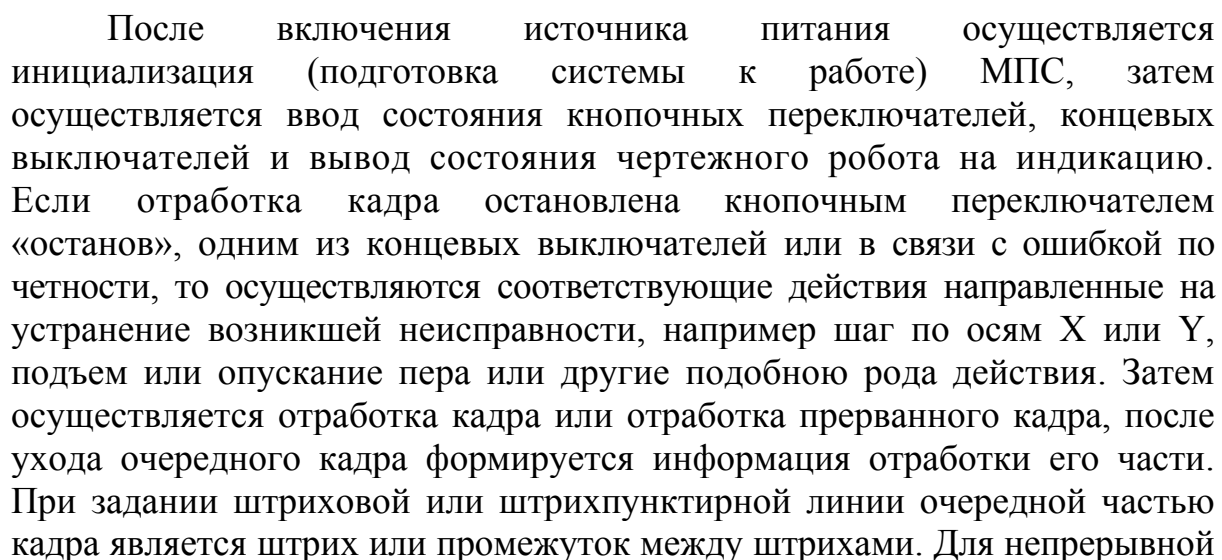
ПУ - пульт управления;

БУШД блок управления шаговым двигателем;

БУП блок управления перьями.

Ввод длимых производится в ОЗУ через интерфейс ввода, работающий в режиме "1", через этот интерфейс производителя так же сопряжение в устройстве ввода с блока МП. Управление перьями и шаговыми двигателями, а также ввод концевых выключателей и кнопок осуществляется с помощью 2БИС параллельного интерфейса, работающего в режиме "0".

**Алгоритм работы, чертежного робота имеет вид:**



линии обрабатывается сразу весь кадр. При выполнении каждого шага по X и Y проверяется состояние концевых выключателей, а после каждого шага состояние кнопочного переключателя "останов". При нажатом кнопочном переключателе "останов" или замкнутых концевых выключателях отработка кадра прерывается. При поступлении команды "конец чертежа" МПС устанавливается в исходное состояние.

## Разработка управляющей команды чертежного робота

При составлении управляющей команды необходимо решить три основные задачи: обеспечить установку меток осуществить проверку выполнения условий организовать передачу управления

Расстановка меток имеет тот же смысл, что и использование символических адресов, т.е. метка присваивает каждой ветви программы определенное имя, к которому надо обратиться при передаче управления. Метка является адресом с которого начинается определенная ветвь программы.

Проверка условий и операций передачи управления реализуется с помощью команд условного перехода: JZ; JNZ; JP; JM; JC; JNC, т.о. используя указанный тип команд удастся совместить проверку условий и передачу управления в одной команде.

Приведем дополнительную информацию о системах и механических средствах обеспечивающих работу чертежного робота.

Структура кадров вводимых в ОЗУ и предназначенных для отработки отрезка прямой или дуги окружности имеет<sup>1</sup> следующий вид:

Для команды ПРЯМАЯ:

		5	4	3	2	1	
		Номер пера	Положе- ние пера	Команда			
		Тип линии		Не используется			
Признаки $\Delta X$	{	0	0				} Значение $\Delta X$
		0	0				
		0	0				
Признаки $\Delta Y$	{	0	1				} Значение $\Delta Y$
		0	1				
		0	1				
		Конец кадра					— Знаки X Y
		0	1				} Значение $\Delta Y$
		0	1				
		0	1				
		Конец кадра					— Знаки X Y
		Конец команды				Контроль точности	

Структура кадра для команды : ДУГА НЦ (эта дуга занимает не  
 целое число квадрантов) име  
 ет вид: 5 4 3 2 1

		Номер пера	Положе- ние пера	Команда			Признак (характеризует удаление или приближение к оси Y)
		Типичный		Не используется	P		
Признак $X_0$	{	0	0				} Значение $X_0$
		0	0				
Признак $Y_0$	{	0	1				} Значение $Y_0$
		0	1				
Признак $Dx + Dy$	{	1	0				} Значение $Dx + Dy$
		1	1				
		1	0				
		1	0				
Конец кадра			X	Y	— Знаки		
Конец команды				Контроль на четность			



Для более понятного восприятия рассмотрим таблицу:

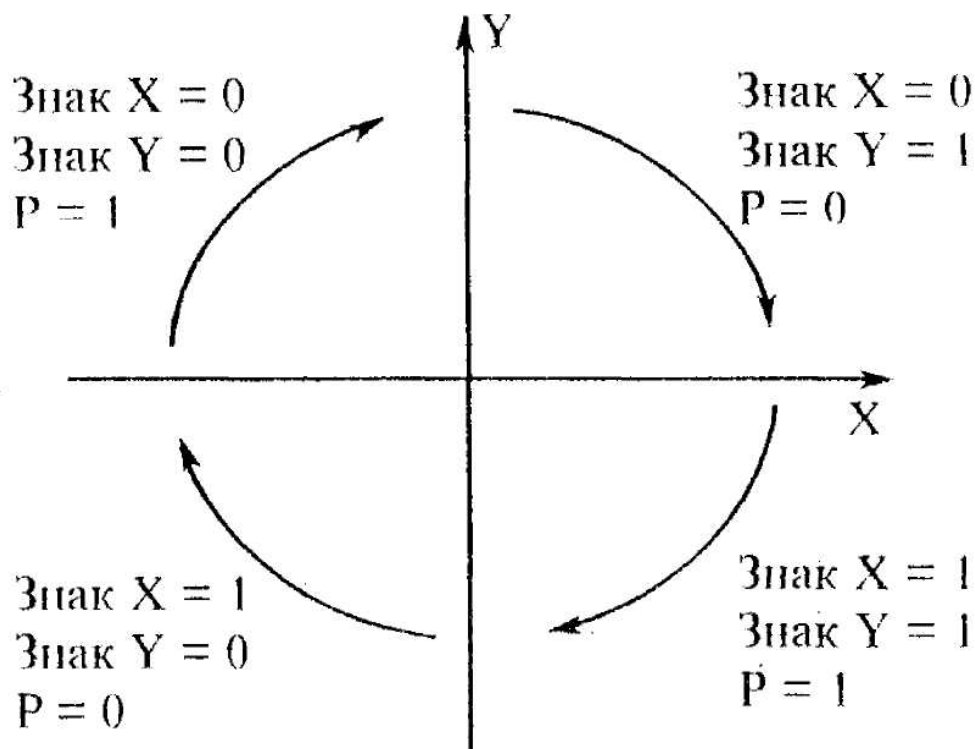
### **Кодирование входной информации для чертежного автомата.**

Тип информации	Информация	Код
Команда	ПРЯМАЯ	001
	ДУГА Ц	010
	ДУГАНЦ	011
	КОНЕЦ ЧЕРТЕЖА	111
Положение перп	Поднято	0
	Опущено	1
Номер пера	Перо 1	0
	Перо 2	1
Тип линии	Непрерывная	00
	Штриховая	01
	Штрихпунктирная	10
Число квадрантов	1 – квадрант	00
	2 - квадранта	01
	3 – квадранта	10
	4 - квадранта	1
		1
Признак Р приближения к оси У	Удаление от оси У	0
	Приближение к оси У	1
Признак числовой информации	Проекция на ось Х-Л Х,	00
	радиус R или абсциса	01
	центра дуги X <sub>0</sub> Проекция	10
	на ось У-А У, ордината	
	дуги У <sub>0</sub> Сумма	
	перемещений вдоль осей	
	Dx - Dy	

Кодовая комбинация	КОНЕЦ КАДРА	110
Знак X - знак проекции па ось X	Движение вправо Движение влево	0 1
Знак Y - знак проекции па ось Y	Движение вверх Движение вниз	0 1
Конец команды	Конец команды	1111

Для отрезка прямой (команда ПРЯМАЯ) указываются длины проекций отрезка на оси координат X и Y и значение этих проекций (ЗнX и ЗнY).

Команда ДУГА Ц обеспечивает вычерчивание дуги, содержащей целое число квадрантов. Эта дуга начинается и заканчивается на осях координат, проходящих через центр окружности. Для команды ДУГА Ц указываются число квадратов и радиус окружности, За начальную точку дуги принимается текущее положение пера. Направление вычерчивания дуги и ориентация задаются с помощью знаков приращения координат ЗнX и ЗнY для начального квадрата вычерчиваемой дуги и признака P, равного 0 при удалении пера от оси Y и 1 при приближении к оси Y.



При движении в сторону увеличения  $X$  или  $Y$  знак  $X$  и  $Y = 0$ , при уменьшении  $X$  или  $Y$  знак  $X$  или  $Y = 1$ .  $P = 0$  при удалении от оси  $Y$ ,  $P = 1$  при приближении к оси  $Y$ .

Команда ДУГА НЦ обеспечивает вычерчивание дуги общего вида. В этом случае числовая команда содержит координаты начальной точки  $X_0$ ,  $Y_0$  относительно центра окружности и сумму перемещений вдоль осей координат  $Dx + Dy$  до конца дуги. Направление вычерчивания и ориентация дуги задаются величинами  $ЗнX$ ,  $ЗнY$  и  $P$ , так же, как в команде ДУГА Ц.

Числовая информация располагается, начиная с третьей строки кадра, и занимает три младших разряда. В двух старших разрядах строк располагаются признаки числовой информации, указывающей ее смысл ( $DX$ ,  $DY$ ,  $X_0$ ,  $Y_0$ ,  $R$ ,  $Dx + Dy$ ). Числовая информация вводится, начиная со старших разрядов, и может занимать различное число строк. Для прямой, параллельной оси координат, указывается длина проекции на одну ось. Единица младшего разряда соответствует перемещению пера на  $0,1$  мм.

Распределение разрядов портов интерфейсных БИС для подключения органов управления чертежного робота проиллюстрируем таблицей.

### Источник или приемник сигнала, направление передачи

Порт/ (разряд)	БИС с адресом 00-03	БИС с адресом 04-07
Порт А, Порт А, Порт А, Порт А, Порт А, Порт А, Порт А, Порт А, Порт В, Порт В, Порт В, Порт В, Порт В, Порт В. Порт В, Порт В, Порт <sup>1</sup> С,Порт С, Порт С, Порт С, Разряд 0 Разряд 1 Разряд 2 Разряд 3 Разряд 4 Разряд 5 Разряд 6 Разряд 7 Разряд 0 Разряд 1 Разряд 2 Разряд 3 Разряд 4 Разряд 5 Разряд 6 Разряд 7 Разряд 0 Разряд 1 Разряд 2 Разряд 3	Кнопочный переключатель "f ",ввод Кнопочный переключатель "А",ввод Кнопочный переключатель "^"ввод Кнопочный переключатель "4-",ввод Кнопочный переключатель "- -",ввод Кнопочный переключатель "П1", ввод Кнопочный переключатель "П2",ввод Кнопочный переключатель "Ввод",ввод Кнопочный переключатель "Останов",ввод Кнопочный переключатель "Пуск",ввод Концевой выключатель "-Х", ввод Концевой выключатель "+Х ", ввод Концевой выключатель "-У", ввод Концевой выключатель "+У", ввод Не используется Не используется Лампочка "ввод", вывод Лампочка "останов ", вывод  Лампочка "Ошибка по четности ", вывод Лампочка "Формат ", вывод	Поправка по Х, разряд I, ввод Поправка по Х, разряд 2, ввод Поправка по Х, разряд 3, ввод Поправка по Х, разряд 4, ввод Поправка по Х, разряд 5, ввод Поправка по Х, разряд 6, ввод Поправка по Х, разряд 7, ввод Поправка по Х, разряд 8, ввод Поправка по У, разряд I, ввод Поправка по У, разряд 2, ввод Поправка по У, разряд 3, ввод Поправка по У, разряд 4, ввод Поправка по У, разряд 5, ввод Поправка по У, разряд 6, ввод Поправка по У, разряд 7, ввод Поправка по У, разряд 8, ввод Поправка по Х, разряд 9, ввод Знак поправки по Х, разряд 9, ввод Поправка по У, разряд 9, ввод Знак поправки по У, разряд 9, <b>ВВОД</b>

Порт/ (разряд)	БИС с адресом 00-03	БИС с адресом 04-07
Порт С, Разряд 4	Вход "--Х " шагового двигателя, вывод	Вход блока управления пером 1, ввод
Порт С, Разряд 5	Вход "+Х " шагового двигателя, вывод	Вход блока управления пером 2, ввод
Порт С, Разряд 6	Вход " Y " шагового двигателя, вывод	Не используется
Порт С, Разряд 7	Вход "+Y " шагового двигателя, вывод	Не используется

## **Переключатели "Правка по Х", "Правка по Y"**

Для замены одного пера на другое во время вычерчивания чертежа необходимо перо поднять и переместить чертежную головку т.о. чтобы новое перо оказалось в точности над той точкой чертежа, над которой ранее находилось старое перо, т.е. переместить на расстояние между перьями. В связи с этим в чертежном автомате могут использоваться различные головки для задания расстояния между перьями предусматриваются два десятиразрядных тумблерных регистра: "Правка по Х", и "Правка по Y". Старшие разряды этих регистров указывают знак поправки. На оставшихся девяти разрядах набирается код абсолютной величины расстояний между перьями на данной оси. Единица младшего разряда соответствует элементарному перемещению головки на 0,1 мм. При нажатом кнопочном переключателе, замкнутом концевом выключателе или установленном в единицу разряде регистра "Правка по Х" или "Правка по Y" на соответствующую линию каналов подается нулевой потенциал, в противном случае на соответствующую линию каналов подается нулевой потенциал высокого уровня. Выход на сигнальные лампочки осуществляется через инверторы с открытым коллектором, поэтому загорание лампочки происходит при наличии единицы на соответствующей линии. На основании имеющихся сведений можно приступить к разработке управляющей программы чертежного робота. Будем считать, что системная программа уже произвела преобразование введенной информации в форму, удобную для работы МПС. Новые значения переменных и соответствующие им коды представлены в таблице:

Информация	Идентификатор	Число байтов.	Значения
Команда	COM	1	01 - наклонная прямая 02 – дуга 03 - прямая, параллельная оси Х 04 - прямая, параллельная оси Y

Признак Р	PFLG	1	00 - Р = 0 . 80 - Р = 1 '      При отработке прямых Р - 0
Направление перемещени е по оси Х	DIR X	I	10 - перемещение в сторону "- х" 20 перемещение в сторону "+ х"
Направление перемещени е по оси Y	DIR Y	I	40 перемещение в сторону " у" 80 перемещение в сторону "+ у"
Количество оставшихся шагов по	DSTN	2	AX + A Y
Признак запомненног о шага	PST	1	00 - нет шага 01 - шаг по X 02 - шаг по Y 03 - шаг по X и Y

Пусть при выполнении программного блока соответствующего отработке части кадра (отрезка сплошной прямой, штриха штриховой линии, или дуги окружности) установленной в нужные значения переменных в таблице: COM; PFLG; DTRX; DIRY; DSIN. Значение PST = 0. Кроме того в стеке размещены значения оценочных функций (верхушка стека), далее проекции обрабатываемого отрезка DX и DY или текущие координаты пера относительно центра дуги X и Y. Величины F; DX; DY; X; Y представляют собой 16 - разрядные числа в дополнительном коде. Начальные значения F = 0.

При составлении программы нам потребуются не рассматриваемые ранее команды, рассмотрим их:

ANI      данные Операция "ИЛИ" над содержимыми аккумулятора и данными содержащимися в поле команды;

LHLD      загрузка регистровой пары HL содержимым ячейки памяти  
адрес которой указан во втором и третьем байте команды;

SHLD      записать в память содержимое регистровой пары HL по адресу указанному во втором и третьем байте команды ;

POP      загрузка регистровой пары из стека;

PUSH                    запись содержимого регистровой пары в стек;

DAD                    к содержимому регистровой пары HL прибавить  
содержимое другой регистровой пары;

SUB                    данные из содержимого аккумулятора вычесть второй байт  
команды (т.е. данные);

SB1                    данные из содержимого аккумулятора вычесть второй байт  
команды с учетом состояния триггера переноса;

ADI                    данные к содержимому аккумулятора прибавить второй  
байт команды;

XRI                    данные исключающее "ИЛИ" над содержимым  
аккумулятора и вторым байтом команды;

ORI                    данные операция "ИЛИ" над содержимым A и вторым  
байтом команды.

Теперь перейдем к составлению программы:

Название: "Отработка части кадра". Присвоен этой программе имя  
"PTFRM".

PTFRM

Отработка части кадра (ввод данных из указанного канала в  
накопитель)

IN 01            ввод состояния органов управления

CM A            поразрядное инвертирование содержимого  
накопителя (поразрядное "И" над содержимым накопителя и  
байтом)

ANI 3C            выделение сигналов с концевых  
выключателей

JNZLDCNT прерывание отработки кадра и переход на  
ввод управляющих сигналов, (переход по отсутствию нули)

CALL PROC (вызов программы ) подпрограмма  
определения оси перемещения и его выполнения

IN 01            ввод состояния органов управления

ANI 01                      выделение состояния кнопочного переключателя "ОСТАНОВ"

JZ LDCNT    прерывание отработки кадра и переход на ввод управляющих сигналов, выполнение манипуляций по устранению неисправностей.

LMLD DS TN    (загрузить ) занести в регистры ML число оставшихся шагов  
(пересылка данных из A в регистр H)

M.OV A,H    проверка содержимого DSTN (регистровой пары HL ) на ноль

OR A L    (поразрядное ИЛИ над содержимым регистра L и накопителем (A))

JNZPTFRM    переход на начало подпрограммы, если отработка части кадра не закончена

HALT    останов

подпрограмма определения оси перемещения и его выполнение

PROC

LDA COM    загрузка ) занесение в A кода команды  
СП 03    сравнение того, что было в ячейке COM с кодом 03

JZ M2    переход к метке M2 если код соответствует прямой, параллельной оси X

LDA COM    занесение в A кода команды

CPI 04    переход к метке M3, если код соответствует прямой, параллель JZM3    ной оси Y

наклонная прямая или дуга окружности

POP H    выборка из стека в регистровую пару HL  
значения оценочной функции F (значение содержимого пары регистров в стек)

PUSH H    восстановление стековой памяти

LDA PFLG    загрузка в A признака P (поразрядное исключаяющее "ИЛИ"

над содержимым регистра P)

XRA H    определение направления перемещения

JM M1    переход к метке M1 если  $f_0$  и  $P=1$  или  $f<()$  и  $P=0$  на шаг по оси

X (переход к минусу) шаг по оси Y

CALL FCHY    подпрограмма коррекции F при шаге по Y

M 3 :

CALL STPY    подпрограмма подготовки шага по оси

Y



RET        возврат

шаг по оси X

M 2 :            CALLSTPX   подготовка шага по оси X

RET   возврат

M 1 :            CALL FCHX    подпрограмма коррекции функции F  
при шаге по оси X

RET

Структура подпрограмм коррекции оценочной функции F при шаге функции по X при шаге функции по Y одинакова. Рассмотрим подпрограмму коррекции оценочной функции по оси Y .

Оценочная функция при вычерчивании дуги имеет значение :

т.е. нам нужно выполнить это действие.

коррекция оценочной функции при шаге по Y

FCHY        POP H        выборка из стека значений оценочной функции в  
регистровую пару HL

POP D        выборка из стека в регистровую пару DE значение  
Лу (прямое) или Y    (дуга)

POP B        выборка из стека в регистровую пару BC значений  
AX или X (луга)

LDA COM     загрузка в A кода команды

CPI 01        (сравнение байта с содержимым накопи геля),  
прямая

JZ LINE        переход на коррекцию функции при отработке  
прямой (переход по нулю)

коррекция оценочной функции при интерполяции дуги

DAD D        содержимое регистровой пары HL сложить с  
содержимым регистровой пары DE, т.е.  $G+y$

DAD D        т.о. мы получим  $G+2y$

MOV A,L      (пересылка данных из регистра L в A)

SUI 01        вычесть единицу из A (регистра L)

MOV L,A      пересылаем в A значение регистра H  
MOVA,H  
SBI 0

MOV H,A      вычитаем из A 0, с учетом состояния триггера  
C регистра  
состояний

коррекция абсолютной величины Y при интерполяции дуги

LDA PFLG      (загрузка в A признака P) анализ признака P и  
увеличение (при P=1) или уменьшение (при P=0) Y

RAL            левый сдвиг (циклический сдвиг содержимого  
накопителя влево через перенос)

JC M4            (переход при наличии переноса ) переход при  
P=1 к метке M4

DCX D            уменьшить            содержимое  
регистровой пары DE на единицу, т.е. уменьшить Y на I ,т.к.  
в DF у нас у. Это вес при P=0

переключение знака X(DIRX) и признака P ( PFI ..... G)  
при переходе в другой квадрант

MOV A, D

ORA E            проверка содержимого регистровой пары DE на  
ноль

CZ SWY            (вызов подпрограммы при нуле), переход к  
подпрограмме переключения знака X (DIR X) и признака  
P(PFLG)

M4 :            JMP M5            (безусловный переход по  
указанному адресу) к метке M5

INX D            увеличить на единицу содержимое  
регистровой пары DE  
JMP M5

коррекция оценочной функции при интерполяции прямой

LINE :

DAD B            к содержимому регистровой пары  
HL прибавить содержимое регистровой пары  
BC

MS :

X(дуга)

PUSH B            занесение в стек OX(прямая) или

PUSH D            занесение в стек DY(прямая) или  
у(дуга)

PUSH H            занесение в стек оценочной  
функции Г(занесение в стек содержимого

	регистровой пары BC; DE; HL соответственно)
	RET возврат
программа переключения D1RX и RF.LG	
SWY	LDA PFLG загрузка в аккумулятор значение признака P
	ADI 80 переключение PFLG (занесение в ячейку с указанным адресом)
	STA PFLG запоминание нового признака P (занесение содержимого A в ячейку с указанным адресом)
	LDA DIRX занесение в A знака перемещения по оси X
	XRI 30 (поразрядное <u>исключающее ИЛИ</u> над содержимым A и байтом), инвертирование 1-го разряда
	STA DIRX запоминание повою значения DIRX
	RET
программа подготовки шага по оси X	
STP X:	LDA PST занесение в A запомненного шага
	CPI 01 (сравнение байта с содержимым A) не был ли заполнен шаг по оси X
	JNZ Mб (переход при отсутствии нуля)
	CALL X вызов подпрограммы шага по X

	RET	возвращение к основной
	программе	
M6:	LDA PST	занести в А признак
запомненного шага		

	CPI 03	запомнены ли шаги по X и Y
	JNZ M7	(переход при отсутствии
нуля)		

	CALL XY	вызов подпрограммы шага по X и
Y		

	RET	
M7 :	LHLD DSTN	загрузить в регистровую пару LFI
содержимое ячейки DSTN (количество оставшихся шагов)		

DCX H	уменьшение на единицу содержимого
регистровой пары HL	
(количества оставшихся шагов)	

SHLD DSTN	сохранить содержимое регистровой пары
LH в ячейках DSTN	

RET

Аналогично подпрограмма подготовки шага по оси Y  
подпрограмма подготовки шага по оси Y

	LDA PST
	CPI 02
	JNZM8
	CALL Y
	RET
M 8:	LDA PST

CPI 03
JNZM9
CALL XY
RET

M9:	LHLD DSTN
	DCX H
	SHLD DSTN
	RET

программа шага по X:

CALL DELAY	вызов подпрограммы задержки
обеспечивающей срабатывание шагового двигателя	
или заданную частоту импульсов	

LDA DIRX    загрузить в А содержимое ячейки DIRX ,  
т.е. управляющее  
слово для шага пол X или код перемещения по оси X

OUT 02      вывод управляющего воздействия в порт С  
интерфейсом БИС или установка в единицу разряда 4  
или 5 порта С

AN1 00      установка в пуль младшего разряда  
управляющего слова (поразрядное И над  
содержимым А и байтом)

OUT 02      сброс разряда 4 или 5 порта С  
RET

программа и шага по Y:

```
CALL DELAY
LDA DIRY
OUT 02
ANI 00
OUT 02
RET
```

программа шага одновременно по X и Y:

```
CALL DELAY LDADIRX
OUT 02 LDADIRY
OUT 02 установка в единицу разряда 6 или 7 порта C LDADIRX
ANI 00
OUT 02 LDADIRY
ANI 00
OUT 02
```

Министерство образования Республики Беларусь

*Учреждение образования*

**Белорусский государственный университет  
информатики и радиоэлектроники**

кафедра радиоэлектронных средств

И.Н. Цырельчук

Практическое занятие №6

**Комплекс классификации транзисторов**

по курсу «Микропроцессорные системы и их применение»

Минск 2006

# Комплекс классификации транзисторов

## 1. Разработка системы информационного обеспечения комплекса классификации транзисторов

Определим структуры входных и выходных данных на примере комплекса классификации транзисторов.

Технические требования которым должен удовлетворять проектируемый комплекс:

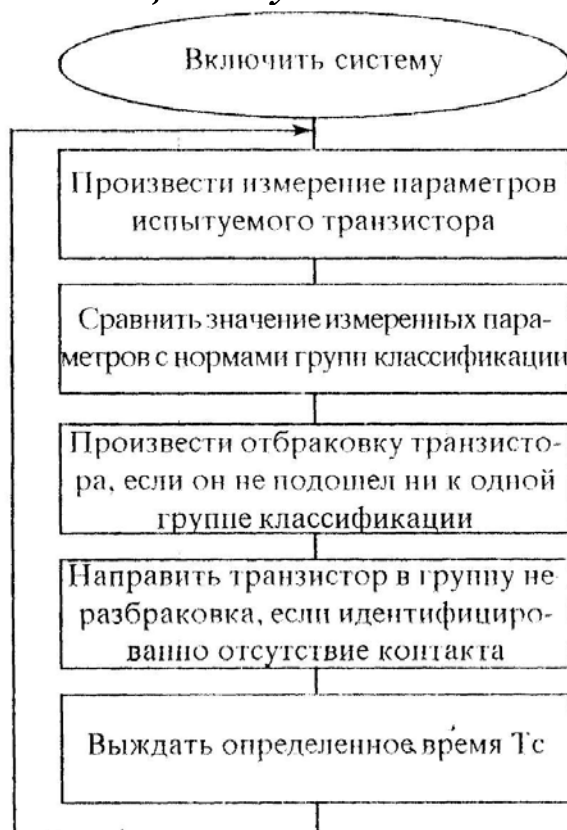
- комплекс обеспечивает автоматическое ориентирование и подачу классифицируемых транзисторов на позицию измерения, автоматическое измерение электрофизических параметров и выгрузку классифицируемых транзисторов в приемные бункеры соответствующие результату классификации;

- производительность комплекса не менее 1000 штук транзисторов в час; количество групп классификаций транзисторов до 14. Из них группы 1-12 годные,

- одна группа брак и одна группа неразбраковка (транзистор установлен, но отсутствует контакт):

комплекс обеспечивает контроль обрыва эмиттер-база, эмиттер-коллектор, и контроль каждой замыкания эмиттер-база, коллектор-база.

### *Концептуальная схема*



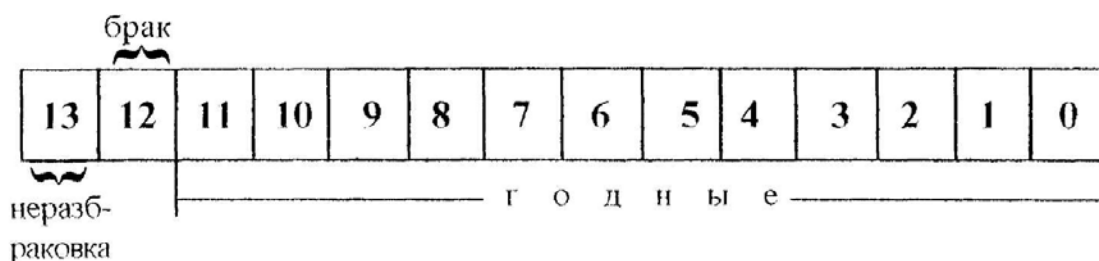


Применительно к комплексу классификации транзисторов необходимо определить по каким правилам будет производиться отнесение испытуемого транзистора к той или иной группе классификации.

В данном случае группы классификации, соответствующие им режимы измерения и нормы классификации определяются таблицами норм электропараметров и представляются закодированными в памяти ЭВМ массивами данных на испытываемый транзистор.

Эти таблицы содержат все данные о режимах измерения и нормах электропараметров *по* которым классифицируется транзистор.

Измерительный массив содержит первоначальное слово классификации. Группы классификации задаются первоначальной установкой разряда слова классификации.



Если результаты измерений параметров соответствуют гномам классификации предусмотренных в таблицах норм электропараметров, то слово классификации не изменяется, т.е. во всех его разрядах сохраняются предварительно записанные единицы, в противном случае в слове классификации переводятся в ноль разряды соответствующие группам в которых транзистор не может быть классифицирован.

Полученное после поведения всех сравнений слово классификации является основой для формирования слова управления выгрузкой.

Испытуемый массив состоит из двух частей:

- указатель полей и кодов наименований;
- информационная часть.

Первая часть состоит из следующих полей:

- указатель полей и кодов наименований;
- ноля наименований параметров;
- поля адресов интервалов;
- ноля наименований групп классификации,

Указатель полей используется для задания начальных адресов полей составляющих испытательный массив, каждая ячейка содержит начальный адрес поля.

Поле наименований параметров служит для задания назначений параметров и тестов по которым производится классификация транзисторов. Символы кодируются в соответствии с ГОСТ.

Поле адресов интервалов это поле которое располагается в ОЗУ ЭВМ за ограничителем поля наименований параметров. Одна запись

представляет собой ячейку в которой находится начальный адрес первого интервала норм разбраковки данного параметра.

Поле наименований групп классификации размещается за полем адресов интервалов в ОЗУ ЭВМ. В этом поле задается наименование групп транзисторов. Одна запись занимает четыре ячейки.

— Информационная часть испытательного массива состоит из поля заданий режимов измерения и поля интервалов.

Поле заданий режимов измерения располагается в ОЗУ ЭВМ за ограничителем, полем наименований групп классификаций. В поле задаются значения режимов измерения параметров транзисторов и записываются в последовательности соответствующей последовательности измерений при классификации транзисторов. Каждая запись поля занимает 12 ячеек. Первые 10 предназначены для программирования значений режимов, две последние для программирования резисторов управляющих схемами коммутации обеспечивающими построение соответствующих схем измерения,

Поле интервалов используется для задания информации предназначенной для окончательной классификации транзистора. Количество записей равно числу классификационных норм на все режимы измерения параметров. Длина одной записи равна 5-ти ячейкам,

В этих ячейках содержится следующая информация:

- первая ячейка - адрес режима измерения;
- вторая ячейка - нижняя граница классификационной нормы;
- третья ячейка - верхняя граница классификационной нормы;
- четвертая ячейка - маска классификации;
- пятая ячейка адрес счетчика брака.

Адрес режима измерения указывает на первую ячейку режима из поля задания режимов измерения. Этот режим задается при измерении данного параметра, а результат измерения классифицируется в соответствии с классификационными нормами данной записи.

Нижняя и верхняя граница классификационной нормы берутся из таблиц ном электропараметров на данный параметр транзистора.

Маска классификации обозначается в соответствии с номером группы.

Адрес счетчика брака указывает на адрес первой ячейки записи в массиве счетчика брака.

К выходной информации формируемой счетчиком классификации транзисторов относятся:

массив счетчиков брака;

- массив счетчиков транзисторов по группам классификации;
- массив счетчиков измеренных транзисторов.

## 2. Разработка схем алгоритмов

Даже для относительно простой системы управления трудно разработать алгоритм, охватывающий сразу все детали проектируемой системы. Поэтому рекомендуется использовать 3 последовательных уровня детализации алгоритма: концептуальную блок-схему;

- функциональную схему;
- структурную схему машинных команд.

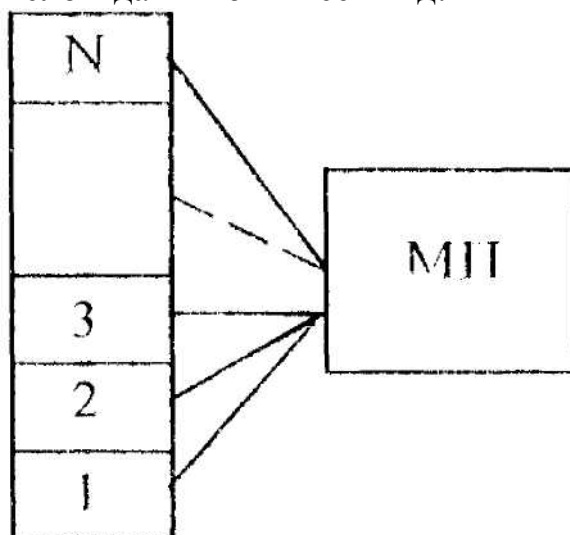
Концептуальная блок-схема алгоритма содержит общие выражения, показывающие, что должно быть сделано. Функциональная схема алгоритма устанавливает, как это может быть сделано. Структурная схема команд - это детальная схема, представляющая собой указатель для кодирования программы.

Две первые схемы могут быть сделаны безотносительно к какому-либо конкретному МП. А третья разрабатывается только для определенного МП, т.к. должны быть известны архитектура и набор команд конкретного устройства.

На стадии разработки концептуальной блок-схемы требуется иметь не очень много функциональных блоков, т.к. записанная в них информация определяет основные действия системы управления.

В качестве примера разработки прочного алгоритма управления рассмотрим задачу из области создания автоматических информационных систем (АИС).

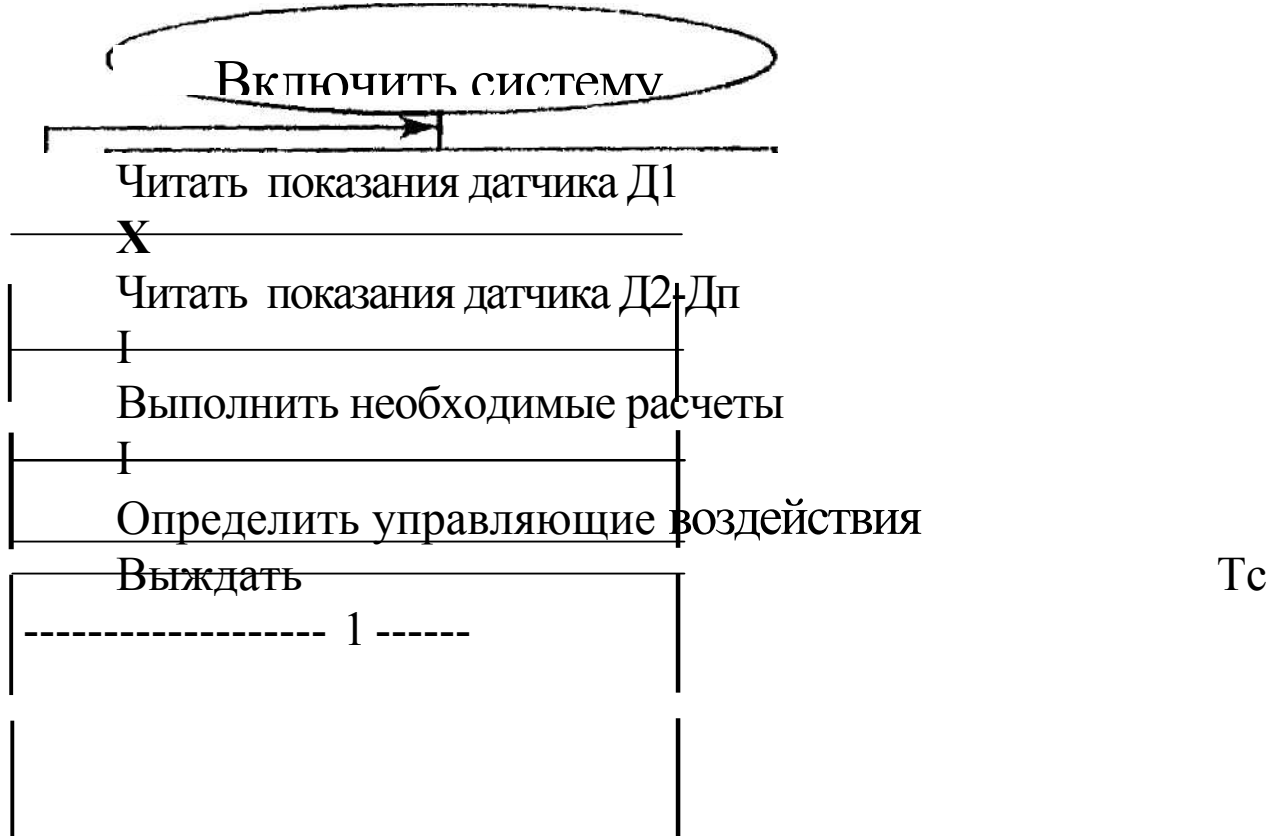
Схема сбора информации о параметрах системы, непрерывно измеряемых некоторым числом датчиков имеет вид:



В концептуальном отношении прежде всего надо решить вопрос, будет ли сбор информации проводиться непрерывно, путем исследовательского опроса всех датчиков (Д) через какие-то интервалы времени, или же показания будут считываться только тогда, когда какая-либо величина изменила свое значение и сообщает об этом МП.

Первый случай называют "программным опросом", а второй "запросом устройства ввода" или "прерыванием от источника информации".

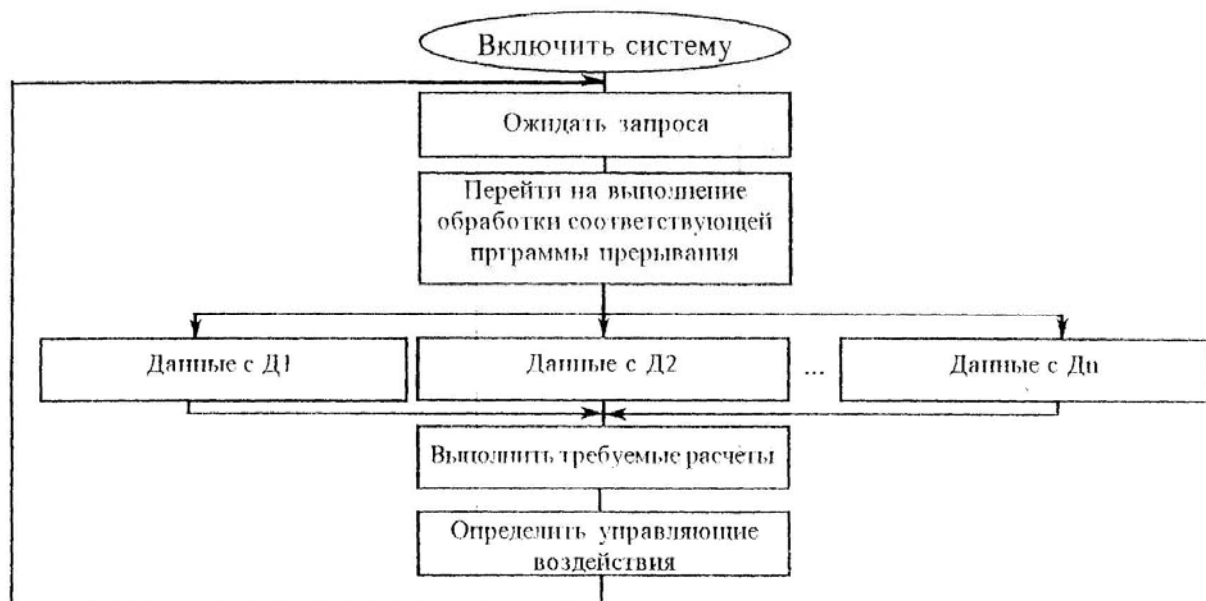
Концептуальная блок-схема алгоритма управления с программным опросом имеет следующий вид:



Последняя часть алгоритма для данного примера представляет собой схему задержки, вырабатывающую временные интервалы длительностью  $T_c$ , чтобы определять моменты времени, в которые надо снимать фиксированные отчеты текущих значений параметров. Эти интервалы могут быть одинаковыми для всех  $D$ , или же периодичность опроса каждой группы  $D$  задается программно. Для формирования временной задержки должен быть разработан специальный алгоритм (подпрограмма).

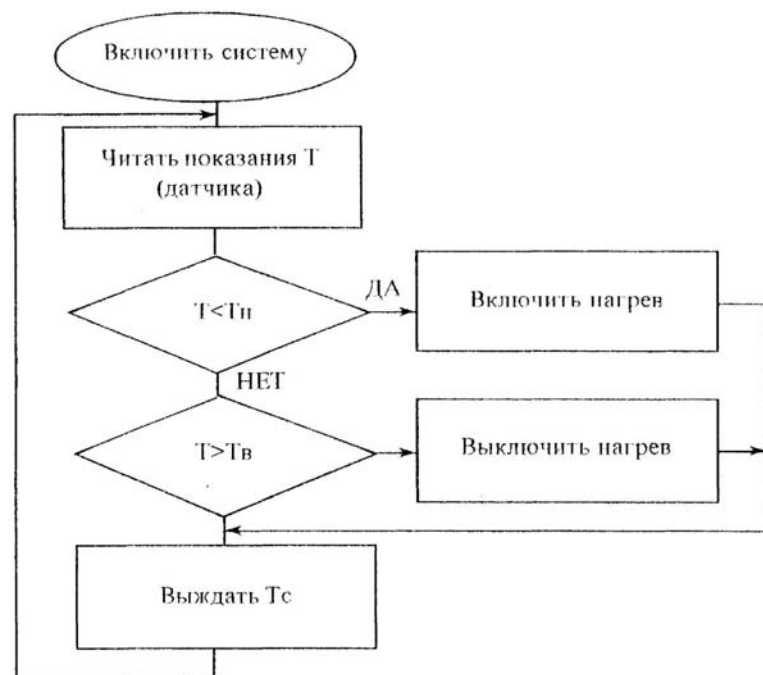
Концептуальная блок-схема алгоритма системы с запросом датчиков (на следующем рис.). Для этой системы необходимо разработать соответствующие программы обслуживания прерываний.

Здесь нет программно определяемой задержки времени. Такая система просто путем сигнала запроса от устройства ввода, чтобы начать работу.



Первая схема более удобна для программиста, но применяется при простых системах, вторая лучше используется в МП.

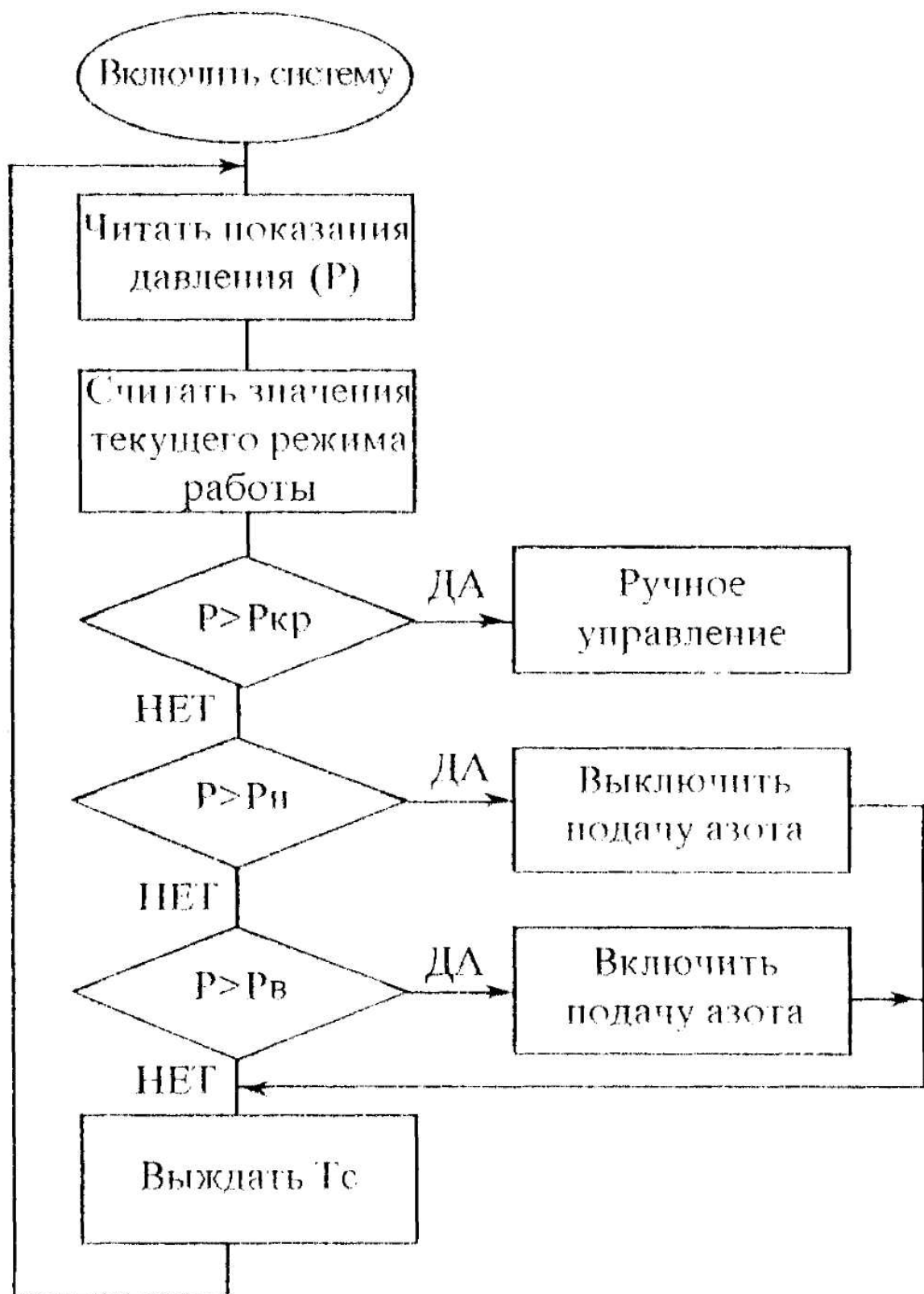
Разработаем алгоритм управления системы у которой реализуемый параметр должен иметь значение в рамках от нижнего предела  $T_n$  до верхнего  $T_v$ . В каждый фиксированный момент времени значение температуры считывается и сравнивается с нижним пределом. Когда это значение меньше  $T_n$  вырабатывается выходной сигнал включающий нагревательное устройство, затем система продолжает делать отсчеты входного сигнала и если он больше  $T_n$ , то система сравнивает эти значения с верхним пределом  $T_v$ , когда входной сигнал больше  $T_v$  посылается управляющий сигнал включающий наг рев.

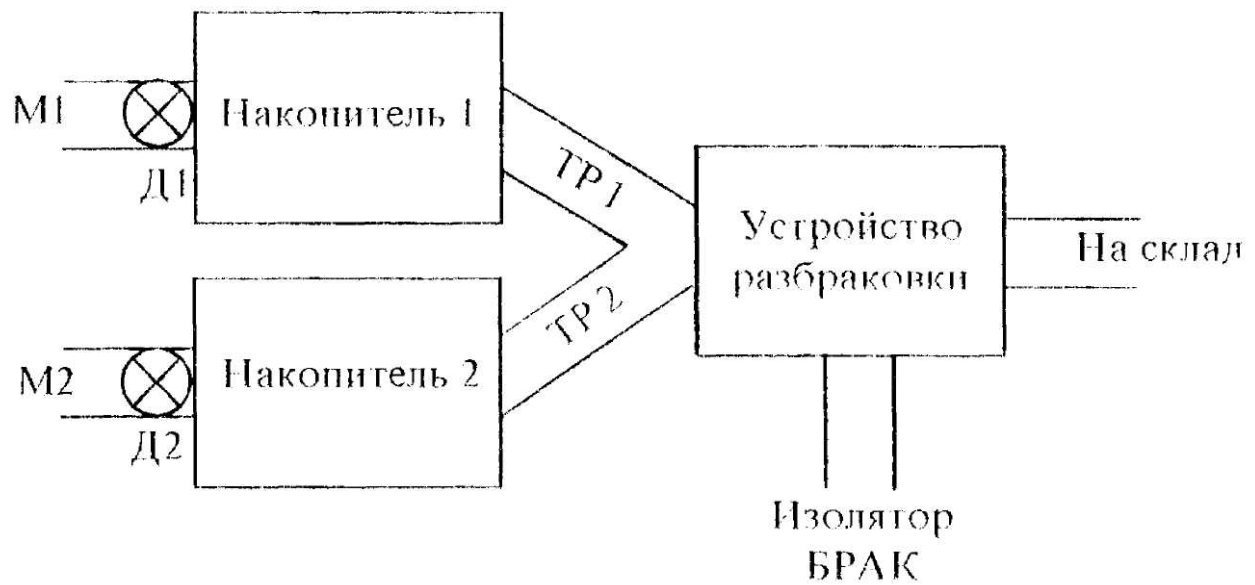


**Пример** Разработать концептуальную блок-схему алгоритма системы регулирования уровня азота в установке плазмо-химического травления. В аварийных ситуациях система переходит в режим "ручного управления".



Блок-схема алгоритма будет иметь следующий вид:

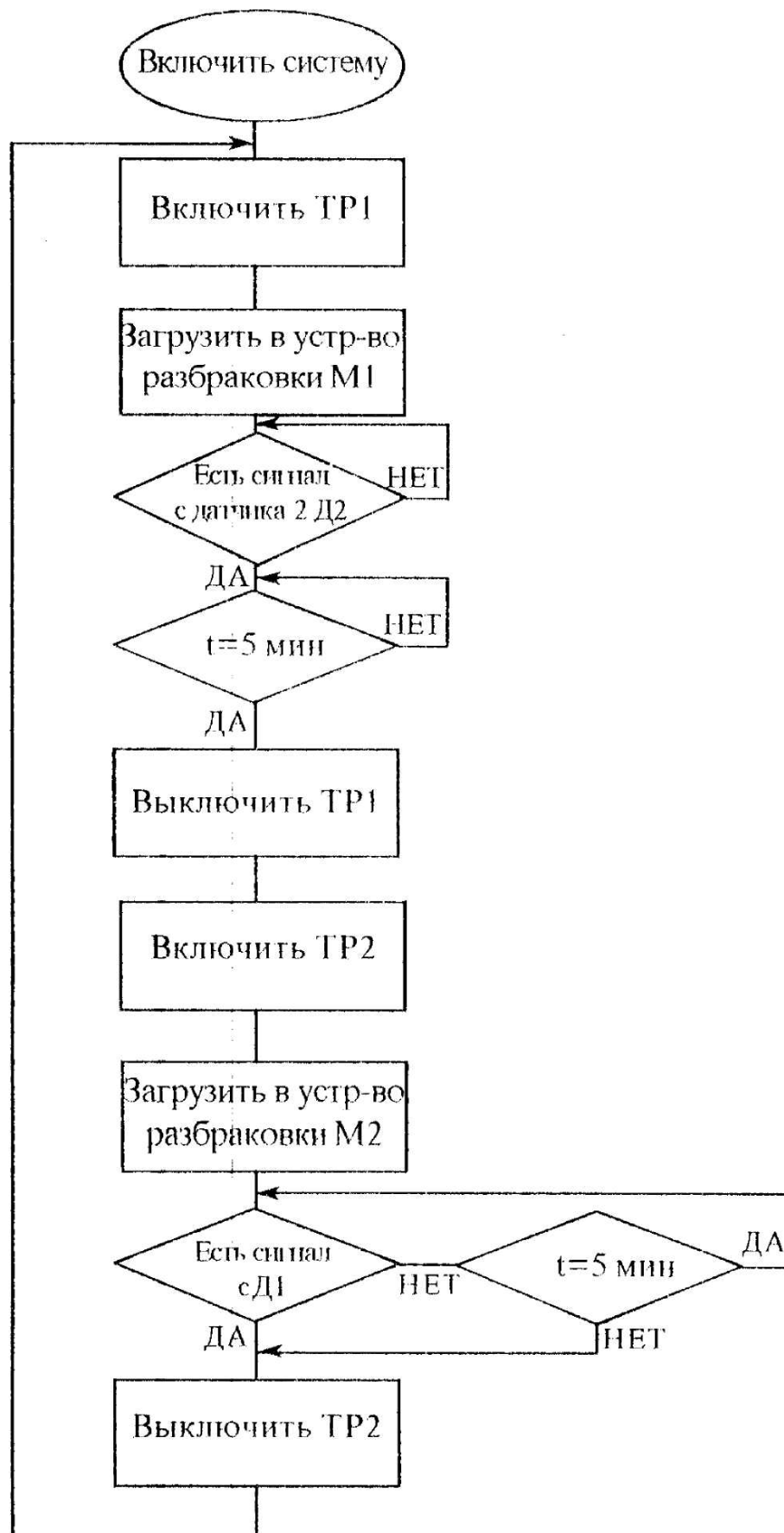




**Пример:** Разработать концептуальную блок-схему алгоритма МП системы разбраковки микросхем.

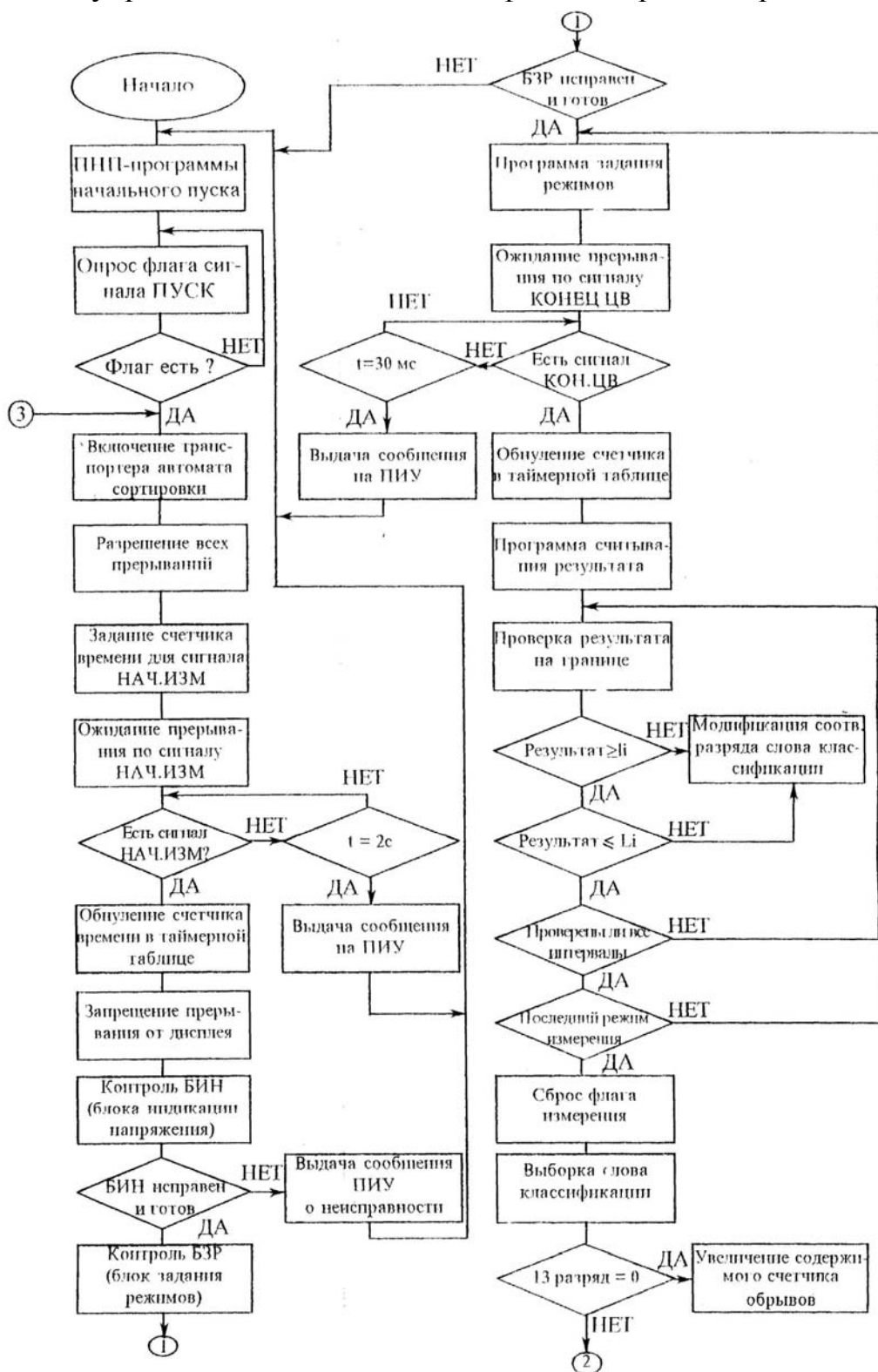
В накопитель 1 поступают микросхемы M1, в накопитель 2 микросхемы M2. Сигналы о наличии ИС в накопителях поступают от электромагнитных датчиков (Д1, Д2) находящихся перед входом соответствующих накопителей. Система управления должна обеспечить следующий режим работы: транспортер 1 (ТР1) должен быть включен, если в течение последних 5 минут от датчика накопителя 2 не поступало сигнала о наличии ИС в накопителе 2. Мин времени включения ТР1 - 5 минут. Если от датчика накопителя 2 поступает сигнал о наличии ИС в накопителе 2, то ТР1 должен быть выключен (если он уже работал больше 5 минут) и включен ТР2. Макс время включения ТР2 - 5 минут. ТР2 должен быть немедленно выключен если от датчика Д1 поступает сигнал о наличии в накопителе 1 микросхем M1.

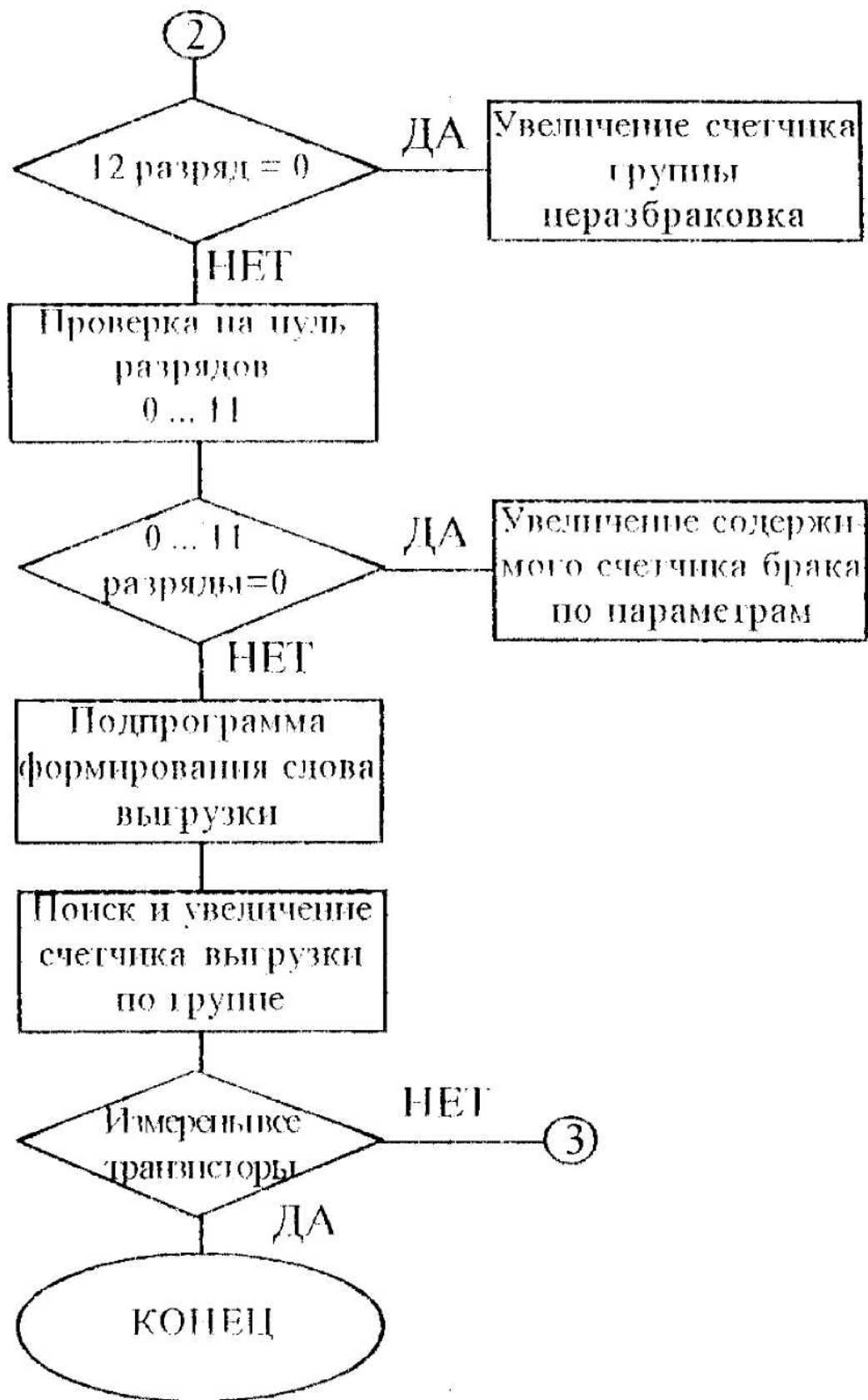




При составлении функциональной схемы алгоритма каждый блок концептуальной блок-схемы "расширяется" до такой степени, чтобы показать отдельные шаги, которые требуется совершить для достижения желаемого результата. Данная схема алгоритма должна быть как можно подробнее.

Разработаем функциональную схему алгоритма работы системы управления комплекса классификации транзисторов.





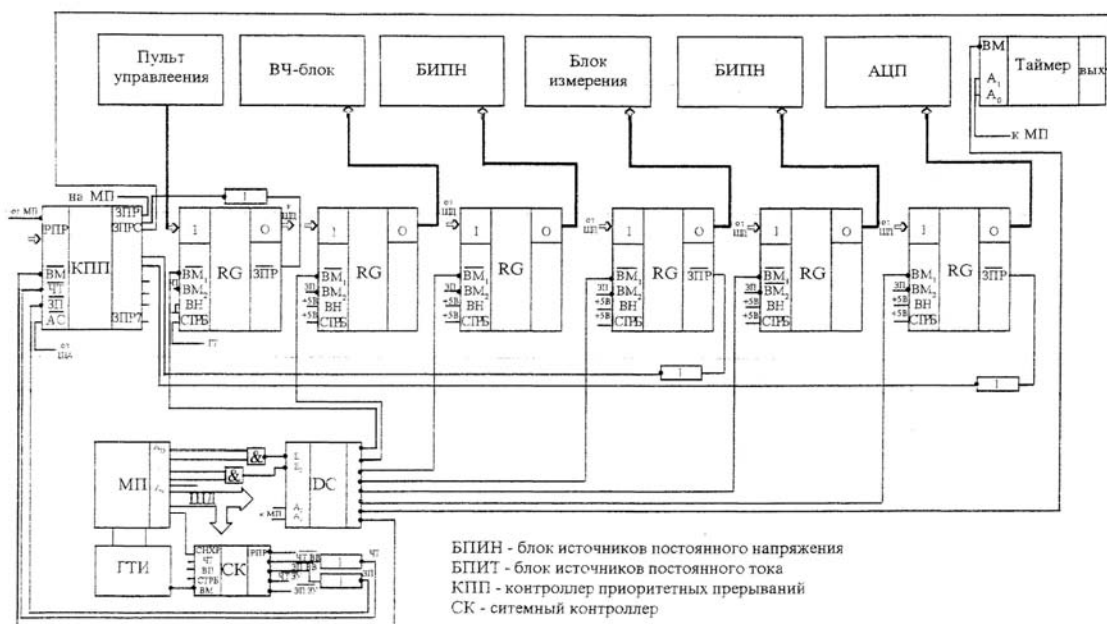
Функциональная схема алгоритма содержит ряд блоков осуществляющих вывод комплекса на режим измерения параметров транзистора. В соответствие с программой начального пуска (ПНИ)

производится вывод на дисплей справочных данных: типов испытываемых транзисторов, перечень команд рабочего режима, исходное слово классификации, производится т а к же установка счетчиков брака и рабочих ячеек в исходное состояние.

Затем система ожидает сигнал ПУСК от автомата сортировки, который появляется при нажатии кнопки ПУСК, после получения этого сигнала включается электродвигатель транспортера сортировки, перемещающий испытываемый транзистор к позиции измерения. Разрешается поступление сигналов прерываний, происходит задание таймером времени ожидания сигнала НАЧАТЬ ИЗМЕРЕНИЯ, поступающего от автомата сортировки при условии установки испытываемого транзистора на позицию измерения. Если в течение 2 сек. сигнала НАЧ.ИЗМ, нет на ПИУ (прибор индикаторный универсальный) выводится сообщение "НЕТ НАЧ.ИЗМ", останавливается работа комплекса и программа переходит в начало.

Если есть сигнал НАЧ.ИЗМ выполняется ряд операций по проверке корректности задания режимов БИМ и БЗР. Если режимы заданы неверно, например установлен автоматический режим вместо ручного подается сообщение на ПИУ и происходит выход в начало. Если БЗР готов ЭВМ выдает сигнал НАЧ.ИЗМ.

Измерение параметров транзистора начинается с задания режимов измерения, которые осуществляются программой задания режимов, загружающей в регистры БЗР в соответствии со значением режимов в испытательном массиве. Эта программа осуществляет также коммутацию схемы измерения и запуск измерителя. Производится задание времени измерения и ожидание сигнала КОНЕЦ 1ДВ, который указывает на завершение аналого-цифрового преобразования в блоке индикации напряжения (ВИН) и на возможность объема результата измерения.



Если в течение 30 мс прерывание, не происходит контроль по таймеру, происходит вывод на ПИУ сообщения НЕТ ИЗМЕРЕНИЯ, останавливается работа комплекса и выход в начало. При получении сигнала прерывания считывается значение кода АЦП. Затем производится выполнение процедур его обработки с целью классификации значения измеренного параметра. Классификация измеренного параметра осуществляется путем проверок полученного кода на нижнюю и верхнюю границы классификационных норм I и L представленных в поле задания режимов испытательного массива. Если результат не попал в границы модифицируется соответствующий разряд слова классификации, если попал, то проверяется по границам классификационных норм по остальным записям поля интервалов (если они есть). Затем происходит проверка на последний режим измерения, если транзистор необходимо испытать в других режимах, задается следующий режим измерения подпрограммой задания режимов и выполняются все необходимые действия для измерения параметров транзистора в этом режиме. Если режим измерения последний, измерение параметров окончено, происходит сброс флага измерения.

После этого производится классификация измеренного транзистора т.е. проверка на ноль разрядов в слове классификации.

### **3. Разработка структурной схемы аппаратной части МП системы управления комплексом классификации транзисторов**

Необходимыми аппаратными средствами комплекса классификации транзисторов, управление которыми осуществляет МП система являются:

-блок программируемых источников постоянного тока и постоянного напряжения, служащие для задания режимов измерения статических параметров испытываемых транзисторов. Значения режимов задаются МП системой путем пересылки данных из расположенных в ячейках памяти таблиц норм электропараметров в регистры соответствующих источников в циклах вывода.

ВЧ блок предназначен для задания ВЧ сигналов, используемых для измерения ВЧ параметров испытываемых транзисторов.

Блок измерения, осуществляющий измерение параметров испытываемых транзисторов. Построение соответствующих схем измерения осуществляется МП системой путем пересылки данных

из расположенных в ячейках памяти таблиц в регистры блока измерения.

- АЦП, осуществляющий преобразование сигналов напряжений, поступающих с выхода блока измерения и пропорциональных величине измеренного параметра в цифровой код.

- Таймер для задания времени работы и порядка включения блоков аппаратной части комплекса.

- Пульт управления для задания режимов и построения соответствующих схем измерения при переходе на ручное управление.

Каждое внешнее устройство комплекса (аппаратное ср-во) комплекса) подсоединяется к ШД МП системы с использованием порта ввода-вывода (через порт ввода/вывода). Активизация работы соответствующего порта происходит в момент появления на управляющих входах порта разрешающих сигналов ВМ I, ВМ2, поступающих от дешифратора адреса и от МП.

Проведем назначение приоритетов внешних устройств. Наивысший приоритет присвоим пульта управления это необходимо для оперативного вмешательства оператора при возникновении сбоев в работе комплекса и аварийных ситуациях, а также непредусмотренного программой перехода с одного режима измерений на другой.

Второй приоритет присвоим таймеру, т.к. в соответствии с алгоритмом работы СУ таймер используется при контроле исправности основных блоков аппаратной части, а также последовательность выполнения обменных и вычислительных операций.

Третий по важности приоритет присвоим блоку измерения т.к. в нем отражается результат работы блока источников постоянного тока и ВЧ-блока.

Четвертый приоритет присвоим АЦП.

Сформируем начально управляющие слова НУС1 и НУС2, задающие структуру КПП и адресный интервал между начальными командами подпрограмм обслуживания прерывания.

Выберем адресный интервал между начальными командами подпрограмм обслуживания прерываний равный 8.

Переведем в двойную систему 4096  
40%-H 000000000000

40%-H 000000000000

0000 1000 0000 0000

0	0	0	1	0	0	1	0
---	---	---	---	---	---	---	---

HYC12

0000	1000
0	8

 $AO \text{ для } EUC2 = 1$ 
$$A_5 - A_{15} - \underbrace{1111111111111111}_{A_{15}} \underbrace{1111111111111111}_{A_5} \underbrace{11}_{A_1 A_0}$$

HYCI  $\rightarrow$  11111111111111100  $\rightarrow$  FFFC

```
HYC2 → 1111111111111100 → FFFD
```

MVIA/I2

OUTFFFC

MVIA,08

OUTFFFD

Для задания простого приоритетного режима не требуется никаких дополнительных команд. После приема МП запроса на прерывание (сигнал ЗПР) и выдачи сигнала разрешения прерывания (1TIP) триггер разрешения прерываний МП сбрасывается и запрещает тем самым прием других прерываний. Поэтому при окончании обслуживания подпрограммы текущего прерывания МП должен выполнить команду EI (разрешить прерывание), кроме этого после выполнения запроса на обслуживание прерывания необходимо осуществить сброс соответствующего разряда регистра запросов путем введения в КПП текущего установочного слова 2.

Структура подпрограммы обслуживания прерывания для простого приоритетного режима имеет вид (рис 1):

При организации многоуровневой системы прерываний после входа МП в прерывающую программу необходимо выполнить программную установку триггера разрешения прерывания МП для получения возможности реагировать на запросы устройств, имеющих более высокий приоритет по сравнению с приоритетом обслуживаемого внешнего устройства. Это осуществляется также командой EI. По окончании подпрограммы обслуживания прерываний МП может производить изменение приоритетов внешних устройств с помощью ввода в КПП ТУС2. На время ввода в КПП ТУС2 необходимо запретить прерывание командой DI, а после ввода ТУС2 снова разрешить прерывание командой EI.

Структура подпрограммы при многоуровневой системе (рис 2):

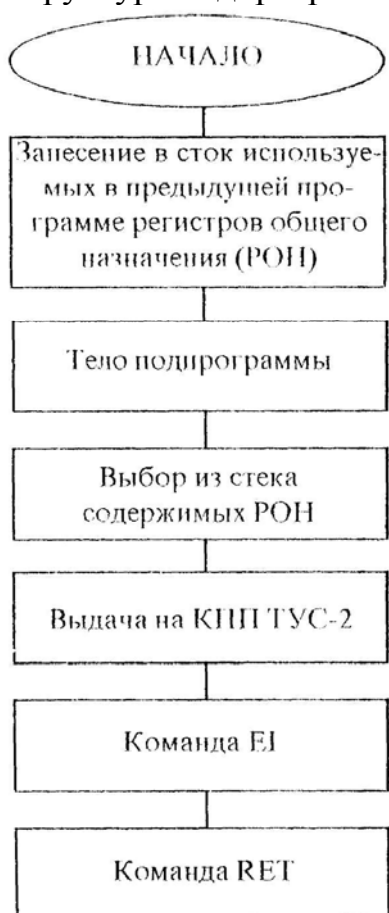


РИС. 1



РИС. 2



Сформируем ТУС2 при котором осуществляется сброс разряда регистра запросов, соответствующего обслуживаемому внешнему устройству и при этом не выполняется циклический сдвиг приоритетов.

ТУ С 2 0010/0000 \*\* 20

Если необходимо осуществить сброс соответствующего разряда регистра запроса с присвоением входу соответствующему этапу разряду наинизшего приоритета с циклическим сдвигом приоритетов ТУС

К) 10/0000\*-А О