К. Г. ФИНОГЕНОВ

ПРОГРАММИРОВАНИЕ ИЗМЕРИТЕЛЬНЫХ СИСТЕМ РЕАЛЬНОГО ВРЕМЕНИ

МОСКВА ЭНЕРГОАТОМИЗДАТ 1990

ББК 32.973–01
Ф60
УДК 681.518.3:681.3.06

Рецензент канд. техн. наук В. В. Бурляев
Редактор Н. А. Медведева

Финогенов К. Г.

Ф60 Программирование измерительных систем реального времени. – М.: Энергоатомиздат, 1990. – 256 с.: ил.
ISBN 5-283-01469-X

Описаны принципы подключения к ЭВМ типа "Электроника 60", СМ ЭВМ измерительного и управляющего оборудования с использованием параллельного и последовательного интерфейсов, аппаратуры прямого доступа в память, интерфейса КАМАК. Рассмотрено программное управление этим оборудованием. Описаны возможности операционных систем реального времени и использование системных средств для работы с расширенной памятью, обработки прерываний и т. д.

Для инженеров, разрабатывающих и эксплуатирующих автоматизированные измерительные установки.

Ф $\frac{2404090000-066}{051(01)-90}$ 217-90 ББК 32.973–01

ISBN 5-283-01469-X

ПРЕДИСЛОВИЕ

Настоящая книга задумана как практическое руководство для инженеров, разрабатывающих программное обеспечение (ПО) автоматизированных измерительно-вычислительных систем (ИВС). В ней рассказано о принципах подключения к ЭВМ измерительной и управляющей аппаратуры, особенностях ее программирования, методике построения управляющих вычислительных комплексов. Особое внимание уделено системным средствам. Инженеры, приступающие к разработке ПО систем реального времени, в первую очередь сталкиваются с трудностями организации правильного взаимодействия прикладной программы с операционной системой (ОС). Какие возможности представляют ОС прикладному программисту, где и с какой целью разумно использовать системные средства – на эти вопросы, как правило, не дают ответа технические описания ОС. В книге рассказывается, и как именно следует использовать системные средства, какие преимущества и недостатки несет в себе такой подход, на многочисленных примерах продемонстрирована техника системного программирования в задачах реального времени.

Программные комплексы реального времени являются, к сожалению, машинно- и системно-зависимыми. Настоящее пособие ориентировано на широко используемые в автоматизированных измерительных установках малые ЭВМ типа СМ-4, СМ-1300, СМ-1600, СМ-1420, "Электроника 60", "Электроника 81", "Электроника 100-25", ДВК, MERA-60, MERA-125 и другие ЭВМ, принадлежащие тем же семействам. В качестве программной базы выбрана ОС реального времени РАФОС-II, а в качестве языка программирования – язык ассемблера СМ ЭВМ. Такой подход позволил изложить вопросы разработки программных комплексов реального времени на конкретном программном материале. Чтение книги требует предварительного знакомства с языком ассемблера и с основными понятиями ОС реального времени в объеме пособий [28] и [29]. или [21]. Для первоначального ознакомления со структурами и функционированием средств связи малых ЭВМ с измерительным оборудованием можно рекомендовать пособия [19] и [30].

Книга посвящена, главным образом, программированию. Однако методика и техника программирования измерительной аппаратуры в значительной степени определяются структурой и особенностями функционирования средств сопряжения аппаратуры с ЭВМ, а те, в свою очередь, – системным интерфейсом используемой машины. Поэтому

начинается книга с рассмотрения системных интерфейсов малых ЭВМ. В гл. 1 дана общая характеристика интерфейсов ОШ и МПИ, используемых в машинах серий СМ и "Электроника", рассказано о назначении и взаимодействии сигналов системной магистрали в процессе выполнения операций передачи данных и машинных команд.

В гл. 2 рассматриваются структура и принципы программирования средств сопряжения измерительного оборудования с ЭВМ (интерфейсов). Приводятся примеры программирования интерфейсов, работающих в режиме ожидания готовности, в режиме прерываний, а также с прямым доступом в память.

В гл. 3 описаны интерфейсы общего назначения, широко используемые при организации измерительно-вычислительных комплексов: параллельного 16-разрядного интерфейса для подключения измерительной аппаратуры, а также радиальных интерфейсов ИРПР и ИРПС.

В гл. 4 описаны принципы программирования аппаратуры КАМАК, приведены примеры программ управления отдельными модулями и небольшими измерительными системами в различных режимах.

Глава 5 представляет собой введение в ОС реального времени. Здесь уделено внимание аппарату системных макрокоманд, организации параллельных процессов, использованию расширенной памяти, оверлейному режиму.

Измерительно-вычислительные комплексы реального времени, как правило, широко используют режим прерываний для управления установкой или процессом измерений, приема измерительной или контрольной информации, взаимодействия с оператором. Разнообразные системные средства организации режима прерываний рассмотрены в гл. 6.

В гл. 7 рассмотрена системная организация ввода-вывода. Здесь выделены два важных вопроса: системные средства обращения к стандартному периферийному оборудованию и методика разработки системных драйверов для измерительной аппаратуры.

В последней гл. 8 речь идет о специфических вопросах разработки программных комплексов реального времени: интерактивном управлении ходом измерительно-вычислительного процесса, выводе контрольной информации, интеллектуальных программах.

Предлагаемая книга является, по существу, первой попыткой систематического изложения конкретных вопросов разработки программного обеспечения измерительных систем и, естественно, не свободна от недостатков. Их было бы гораздо больше, если бы не заинтересованное участие рецензента книги канд. техн. наук, доцента В.В. Бурляева, которому автор выражает искреннюю признательность. Автор будет благодарен читателям за любые замечания и предложения по содержанию и стилю книги. Отзывы и замечания направлять по адресу: 113114, Москва, Шлюзовая наб., 10, Энергоатомиздат.

Автор

ВВЕДЕНИЕ

Резкий рост промышленного производства разнообразных средств вычислительной техники привел к интенсивному внедрению ЭВМ в процесс измерений. Использование мини- и микроЭВМ (или *малых* ЭВМ, как мы их будем в дальнейшем называть) в качестве элемента измерительной установки позволяет повысить точность и информативность измерений, увеличить их производительность, автоматизировать процесс получения, накопления и обработки информации. Несмотря на огромное разнообразие конкретных измерительных установок, их структура часто оказывается схожей.

Физическое воздействие, подлежащее регистрации и измерению (температура, давление, световой поток, магнитное поле, поток элементарных частиц и т. д.), преобразуется соответствующими датчиками в электрический сигнал. В качестве датчиков могут использоваться термопары, пьезоэлектрические и фотоэлектрические приборы, детекторы элементарных частиц и проч. Электрические сигналы, снимаемые с выхода датчика, несут в себе информацию о характеристиках регистрируемого физического воздействия, причем эта информация может заключаться в различных параметрах сигналов: величине непрерывно изменяющегося электрического напряжения или тока, амплитуде импульсов, временно́м интервале между отдельными импульсами или средней частоте их следования и т. д. Для того чтобы ЭВМ могла воспринимать эту информацию, ее надо преобразовать в цифровой код, и поэтому непременным элементом измерительной установки является один или несколько кодировщиков (аналого-цифровых преобразователей, (АЦП)). Помимо кодировщиков электронная аппаратура измерительной установки может содержать также различные устройства, выполняющие усиление, преобразование, сортировку, предварительное накопление и отбор поступающих сигналов или кодов. С выхода электронной аппаратуры закодированная информация поступает в ЭВМ.

Функции ЭВМ в измерительной установке весьма многообразны. Одной из таких функций является просто накопление поступающей информации. Во многих случаях объем измерительной информации столь велик, что ее накопление и хранение возможно только на машинных носителях, таких, как накопители на магнитных дисках (НМД) или лентах (НМЛ). Автоматизированные системы сбора и накопления информации, в которых окончательная обработка информации выпол-

няется спустя значительное время после завершения измерений, широко используются в космических исследованиях, в физике элементарных частиц, в исследованиях океана и атмосферы, в биологии, медицине и других отраслях науки.

Возможности ЭВМ используются полнее, если в ее функции входит анализ поступающей информации. Быстродействие современных ЭВМ позволяет выполнять предварительный анализ регистрируемой измерительной информации в темпе ее поступления, или, как говорят, в *реальном времени*. Целью предварительного анализа может быть отбор полезных событий и отбрасывание информации о различного рода фоновых явлениях и помехах, что часто приводит к существенному снижению объема накапливаемых данных и повышению их информативности. Широко используется также методика "подправки" каждого регистрируемого кода по результатам градуировочных измерений. Это позволяет компенсировать нелинейность и другие погрешности измерительного тракта и обойтись без относительно сложных систем стабилизации. Часто в процессе измерений регистрируются и изучаются сразу несколько видов физических воздействий (например, частицы различной природы, регистрируемые одним детектором). Анализ в реальном времени позволяет выполнить раздельное накопление информации об этих объектах.

В автоматизированных измерительных установках в функции ЭВМ обычно входит управление ходом измерительного процесса. В программу управления измерительной установкой вводится информация о требуемых режимах и последовательности измерений, после чего ЭВМ в заданные моменты времени блокирует и разблокирует входы кодировщиков, переключает измерительные каналы, изменяет (при необходимости) параметры электронной аппаратуры, входящей в установку. При этом ЭВМ может сама вносить поправки в процесс измерений, например вычислять время экспозиции в соответствии с интенсивностью регистрируемого процесса, либо определять число измерений в зависимости от скорости изменения исследуемой величины.

Использование ЭВМ предоставляет широкие возможности по управлению измерительной установкой и автоматизации процесса измерений. Смена образцов, выведение их на позиции подготовки (облучения, откачки, нагрева и т. д.), измерений и хранения, изменение физических условий в измерительном объеме, воздействие на какие-либо характеристики установки с целью их стабилизации – все это нетрудно выполнять по командам от ЭВМ в соответствии с заложенной в нее программой. Программное управление расширяет возможности установки, упрощает работу с ней и повышает точность измерений.

Важным элементом автоматизированных ИВС является диалог ЭВМ с оператором. В состав ПО помимо программ сбора и обработки информации включаются вспомогательные программы, позволяющие оператору наблюдать за ходом измерений, получать информацию о состоянии установки, изменять режим измерений или обработки и т. д. Некоторые

из этих программ активизируются автоматически и выводят на экран дисплея контрольную информацию через определенные интервалы времени либо при выполнении некоторого условия (создания аварийной ситуации, завершения очередной серии измерений, регистрации требуемого события); другие программы активизируются оператором подачей соответствующих команд через клавиатуру терминала. Таким образом, управление автоматизированной измерительной установкой осуществляется в основном программным образом через терминал ЭВМ.

Разработка измерительно-вычислительных комплексов (ИВК) реального времени требует основательного знакомства с целым рядом вопросов: языками программирования, аппаратными средствами связи электронного измерительного или управляющего оборудования с ЭВМ, а также принципами и техникой программирования этого оборудования, операционными системами (ОС) реального времени и методикой использования системных средств в прикладных программах, особенностями организации программных комплексов реального времени.

Для программирования систем реального времени используются как универсальные (ФОРТРАН, ПАСКАЛЬ, языки ассемблеров), так и специализированные (IML, QUASIC, КОК) языки программирования. Пока наиболее распространенными являются ФОРТРАН и языки ассемблеров, обычно на ФОРТРАНе пишутся программы обработки данных, а на языках ассемблеров – программы управления электронным оборудованием измерительной установки. Все шире при разработке программных комплексов реального времени используются языки ПАСКАЛЬ, МОДУЛА-2 и некоторые другие.

Структура и особенности средств сопряжения электронного оборудования с ЭВМ (так называемых *интерфейсов*) в значительной степени определяют характеристики всего ИВК, прежде всего его быстродействие, возможности параллельной обработки нескольких процессов, простоту перестройки. Обычно эти характеристики носят взаимно противоречивый характер. Так, наибольшую скорость приема информации обеспечивает интерфейс прямого доступа в память; он же наилучшим образом позволяет вести параллельную обработку. Однако этот интерфейс слишком сложен и дорог и, главное, для него характерно отсутствие универсальности: интерфейс конструируется под конкретную измерительную аппаратуру и, более того, под конкретный режим измерений. В тех случаях, когда специфика измерений требует периодической перестройки аппаратной части комплекса, удобно использовать аппаратуру КАМАК, обладающую большой универсальностью. Однако аппаратура КАМАК работает относительно медленно, а программирование ее является довольно сложным делом. Таким образом, выбор средств сопряжения измерительной аппаратуры и ЭВМ является ответственным этапом разработки ИВК.

Существует два основных способа управления любыми внешними по отношению к ЭВМ устройствами: режим программного управления

и режим прерываний. Правильный выбор и сочетание этих режимов важны для эффективной работы измерительного комплекса. Кроме того, при использовании режима прерываний возникает необходимость решения ряда методических вопросов: назначения устройствам приоритетов, разделения функций обработки по уровням прерываний и т. д. Значительную специфику в этом отношении имеет программирование аппаратуры КАМАК.

Использование вычислительных машин в измерительных системах кардинально отличается от их применения для решения вычислительных задач. В последнем случае основным инструментом инженера является язык программирования высокого уровня (ФОРТРАН, ПЛ/1, ПАСКАЛЬ и др.). Знаний по архитектуре ЭВМ и тем более системному ПО в этом случае практически не требуется; программы и даже сложные программные комплексы отличаются детерминированностью и, будучи однажды отлажены, работают в дальнейшем всегда одинаково хорошо. Программы реального времени, напротив, широко используют средства ОС. Такие системные функции, как обработка прерываний, работа с системным таймером, организация ввода-вывода, временная и событийная синхронизация задач, построение мультипрограммных комплексов и многие другие являются важнейшими средствами организации программ и программных комплексов реального времени. Далее, программы реального времени, в отличие от вычислительных, работают в условиях изменяющейся внешней среды. Ход выполнения программы обычно зависит от таких непредсказуемых условий, как скорость накопления и характер регистрируемых событий, нестабильность измерительной аппаратуры, возникновение аварийных ситуаций и т. д. Поэтому в программах реального времени обычно содержатся элементы интеллектуальности. Программа сама следит за внешними условиями и автоматически изменяет свое выполнение в зависимости от складывающихся обстоятельств. Отладка таких программ включает в себя моделирование возможных ситуаций и тщательную проверку работы разрабатываемого комплекса в различных условиях.

Все это вместе взятое позволяет рассматривать разработку программного обеспечения измерительных систем реального времени как самостоятельную область программирования с развитым алгоритмическим аппаратом, специфическими методами отладки и тестирования, своеобразными условиями применения. Овладение методами и техникой разработки программных комплексов реального времени является необходимым условием успешного применения средств автоматизации в любой отрасли народного хозяйства.

Глава 1

СИСТЕМНЫЕ ИНТЕРФЕЙСЫ МАЛЫХ ЭВМ

1.1. ОБЩАЯ ХАРАКТЕРИСТИКА ИНТЕРФЕЙСОВ

Конфигурация измерительно-вычислительного комплекса (ИВК), средства связи измерительной и управляющей аппаратуры с ЭВМ, а также техника программного управления этой аппаратурой в значительной степени определяются тем, какой системный интерфейс используется в примененной ЭВМ. Под системным интерфейсом ЭВМ понимают весь комплекс средств сопряжения центрального процессора (ЦП), оперативной памяти (ОП) и внешних устройств (ВУ), входящих в состав ИВК. Системный интерфейс представляет собой совокупность унифицированной магистрали для передачи информации, электронных схем, служащих для согласования, преобразования и управления сигналами на магистрали, а также унифицированных алгоритмов (протоколов) обмена информацией между отдельными устройствами ЭВМ.

Системный интерфейс семейства машин СМ ЭВМ носит название "Общая шина" (ОШ). Конкретные технические характеристики интерфейса различных машин этого семейства могут несколько различаться (например, магистраль ЭВМ СМ-1300 содержит 16 линий адреса, ЭВМ СМ-4 18 линий, а ЭВМ СМ-1420 22 линии адреса), но состав и назначение сигналов интерфейса, так же как и протоколов обмена информацией, для всех этих машин совпадают. Прообразом ОШ является системный интерфейс UNIBUS вычислительных машин семейства PDP-11 американской корпорации DEC.

Системный интерфейс семейства машин "Электроника" называется каналом обмена информацией или просто каналом ЭВМ. При этом у некоторых машин ("Электроника 100-25", "Электроника 79") канал практически тождествен ОШ и отличается от последнего только конструктивными особенностями. Другая группа машин ("Электроника 60" и ее разновидности, "Электроника 81") имеет системный интерфейс, идеологически схожий с ОШ, но отличающийся от него составом линий и сигналов и порядком их взаимодействия. Этот тип канала аналогичен системному интерфейсу QBUS машин LSI-11 той же корпорации DEC. Схожий системный интерфейс используется в диалоговых вычислительных комплексах ДВК ("Электроника НЦ-80-20"), где он носит название МПИ (микропроцессорный интерфейс). В результате микро-ЭВМ семейства "Электроника" оказываются аппаратно несовместимыми как с мини-ЭВМ того же семейства ("Электроника 100-25", "Электроника 79"), так и с машинами типа СМ (СМ-4, СМ-1420 и др.). Это озна-

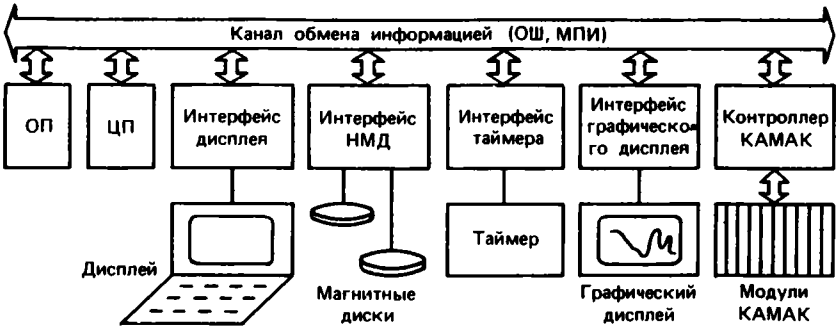


Рис. 1.1. Структура ИВК

чает, что внешнее устройство, например накопитель на магнитном диске (НМД), рассчитанное для использования с ЭВМ СМ-4, нельзя непосредственно подключить к ЭВМ "Электроника 60". С другой стороны, все упомянутые машины имеют единую систему машинных команд, единый язык программирования низкого уровня – язык АССЕМБЛЕРа, одинаковые принципы подсоединения и программного управления как стандартными ВУ, так и измерительной аппаратурой, входящей в состав ИВК, одни и те же ОС. Все это определяет программную совместимость рассматриваемых машин не только на уровне машинных кодов, но и на уровне структурной организации программных комплексов реального времени.

Практическое использование ЭВМ для автоматизации измерений или построения автоматизированных приборов и установок не требует детального изучения структуры и функционирования системного интерфейса. Однако представление об общих принципах и основных характеристиках системного интерфейса облегчает рассмотрение средств связи измерительной аппаратуры с ЭВМ, режимов работы и особенностей программирования этой аппаратуры, возможностей оптимизации управляющих программ.

Схематически канал обмена информацией – магистральный системный интерфейс (рис. 1.1) можно представить в виде длинного многопроводного кабеля, к которому единообразно подключаются все устройства ЭВМ: оперативная память (ОП), центральный процессор (ЦП), а также устройства сопряжения с различным периферийным оборудованием – алфавитно-цифровыми дисплеями, накопителями на магнитных дисках и лентах (НМД и НМЛ), печатающими устройствами и т. д. Устройства сопряжения, называемые интерфейсами, служат для преобразования унифицированных сигналов магистрали в сигналы, управляющие работой периферийного оборудования. Помимо стандартных ВУ, служащих главным образом для ввода в ЭВМ или вывода из нее текстов

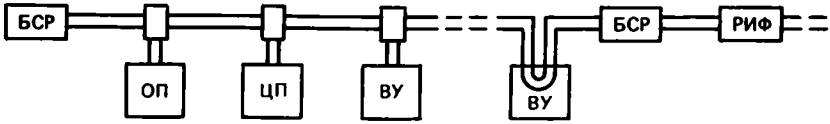


Рис. 1.2. Подключение дополнительных устройств к каналу ЭВМ

программ, а также исходных данных и результатов обработки, в состав ИВК входит, естественно, то или иное измерительное и управляющее оборудование: аналого-цифровые и цифро-аналоговые преобразователи (АЦП и ЦАП), таймеры, входные и выходные регистры, средства управления электромеханическим оборудованием и т. д. Все эти устройства могут подсоединяться к магистрали ЭВМ индивидуально, через соответствующие интерфейсы, либо входить в систему программно-управляемых электронных модулей, например систему КАМАК, и подсоединяться к ЭВМ через общий интерфейс – контроллер КАМАК.

Конструктивно канал представляет собой печатную плату, обеспечивающую электрическое соединение контактов разъемов, с помощью которых к каналу подключаются различные устройства, либо совокупность отрезков многопроводного кабеля, соединяющих разъемы для подключения устройств (рис. 1.2). В последнем случае одноименные контакты входного и выходного разъемов каждого устройства соединены перемычками. Так или иначе сигналы магистрали независимо от места их возникновения (ЦП, ВУ, ОП) поступают последовательно на все устройства комплекса. В задачу каждого устройства входит расшифровка этих сигналов и отбор из них тех, которые адресованы данному устройству. Для предотвращения отражений конечные отрезки кабеля подсоединяются к блокам согласующих резисторов (БСР). Такая конструкция канала позволяет легко расширять аппаратный состав вычислительного комплекса. Для этого достаточно, связав между собой отрезками кабеля добавляемые устройства, подключить их к конечному отрезку магистрали действующего комплекса вместо БСР, перенеся его на выходной разъем последнего устройства. При подключении дополнительных устройств следует иметь в виду, что общая длина магистрали не должна превышать 15 м; кроме того, все подключенные устройства должны создавать не более 20 единиц нагрузки (каждое устройство, в зависимости от его сложности, эквивалентно одной или нескольким единицам нагрузки, обычно не более 4). Указанные ограничения можно снять, воспользовавшись расширителями интерфейса (РИФ). Каждое такое устройство, нагружая расширяемый отрезок магистрали одной единицей нагрузки, обеспечивает возможность дополнительного подключения до 19 единиц нагрузки при суммарной длине дополнительного кабеля до 15 м. Таким образом, системный интерфейс допускает практически неограниченное развитие аппаратных средств комплекса.

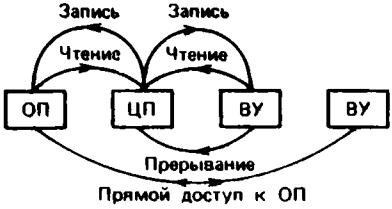


Рис. 1.3. Основные действия на системной магистрали ЭВМ

Несмотря на разнообразие задач, решаемых ЭВМ, процессы, происходящие на системной магистрали, ограничены небольшим числом основных действий. К таким действиям можно отнести операции записи, чтения, прерывания и прямого доступа в память (рис. 1.3).

Операция чтения позволяет процессору получить необходимую для выполнения программы информацию: из ОП – код очередной команды или данные (операнды выполняемой команды); из ВУ – слово состояния ВУ или очередную порцию данных.

В процессе операции записи процессор передает в ОП результат вычислений, а в ВУ – новые значения управляющего слова или очередную порцию данных (заметим, что направление передачи данных определяется относительно процессора: чтение в процессор, запись из процессора).

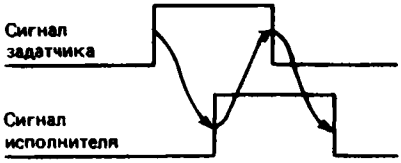
С помощью операции прерывания ВУ оповещает процессор о своей готовности к передаче очередной порции данных. Эта операция позволяет ВУ выполнять активную роль и создает предпосылки для оперативной обработки регистрируемых комплексом внешних событий.

Операция прямого доступа в память служит для быстрой передачи в ОП или из нее отдельных порций или массивов информации под управлением не процессора, как обычно, а контроллера ВУ (контроллера прямого доступа). Использование прямого доступа приводит к заметному усложнению аппаратуры ВУ, но существенно повышает скорость обмена информацией (по сравнению с режимом прерываний в десятки и даже сотни раз). Прямой доступ используется с такими стандартными ВУ, как НМД и НМЛ, а также с разнообразной измерительной и регистрирующей аппаратурой (АЦП, регистрами, запоминающими устройствами, служащими для предварительного накопления регистрируемых данных, и т. д.).

С помощью перечисленных операций реализуется все многообразие действий, выполняемых ЭВМ. Редактирование текста создаваемой программы, трансляция, выполнение вычислительной задачи, прием информации из измерительной аппаратуры, управление автоматизированной установкой – все это раскладывается на простейшие операции, прежде всего чтения и записи, а также в случае необходимости прерываний и прямого доступа. Быстродействие ЭВМ в конечном счете определяется скоростью выполнения этих элементарных операций.

Для магистральных системных интерфейсов характерны некоторые общие принципы построения.

Рис. 1.4. Реализация квитирования



В процессе взаимодействия любых двух устройств ЭВМ одно из них обязательно выполняет активную, управляющую роль и является *задатчиком*, второе же оказывается управляемым, *исполнителем*. Чаще всего функцию задатчика выполняет процессор. Например, операции считывания из ОП очередной команды и ее операндов или записи в управляющий регистр ВУ управляющей информации, а в регистр данных ВУ очередного данного выполняются по инициативе и под управлением ЦП, который выступает здесь в качестве задатчика. Исполнителем в первом случае является ОП, во втором – ВУ. Внешнее устройство может стать задатчиком на ОШ в процессе выполнения операции прерывания, когда из ВУ в ЦП поступает адрес вектора прерывания. Центральный процессор здесь играет пассивную роль исполнителя. Наконец, при операциях прямого доступа задатчиком является ВУ (конкретнее, контроллер прямого доступа), а исполнителем – ОП.

Другим важным принципом, заложенным в структуру магистрального интерфейса, является принцип запроса-ответа (квитирования): каждый управляющий сигнал, посланный задатчиком, подтверждается ответным сигналом исполнителя. При отсутствии ответного сигнала исполнителя в течение заданного интервала времени (обычно 10–20 мкс, так называемый *тайм-аут*) задатчик фиксирует ошибку обмена (исполнитель отсутствует, неисправен, выключен) и прекращает данную операцию. Практически квитирование обычно реализуется так, как показано на рис. 1.4. Сигнал, установленный задатчиком на какой-либо линии магистрали, распространяется по ней и через некоторое время доходит до исполнителя. Последний, получив сигнал задатчика, устанавливает на какой-то другой линии магистрали ответный сигнал, который также начинает распространяться по магистрали. Через некоторое время он доходит до задатчика, который, получив этот сигнал, и удостоверившись тем самым в том, что исполнитель присутствует и нормально функционирует, снимает свой сигнал. Исполнитель, зафиксировав прекращение действия сигнала задатчика, снимает свой сигнал, и процесс обмена сигналами заканчивается. Такой принцип обмена сигналами позволяет выполнять операции на магистрали с максимально возможной для каждой пары задатчик-исполнитель скоростью при высокой надежности обмена. Максимальная скорость пересылки информации по магистрали составляет $2{,}5 \cdot 10^6$ слов/с.

Третья важная особенность архитектуры рассматриваемых ЭВМ заключается в идентичности подключения к системному интерфейсу всех устройств ЭВМ, включая ОП и ЦП. В составе магистрали отсутст-

вуют специальные линии или сигналы управления ВУ; основным средством обмена информацией с ВУ являются упомянутые выше операции чтения и записи. Идентичность подключения к магистрали ОП и ВУ определяет возможность использовать в процессе управления ВУ весь набор команд процессора: пересылки, анализа содержимого, логических и арифметических операций и проч. Это существенно расширяет возможности программирования ВУ и в какой-то степени унифицирует процесс составления программ реального времени.

Схемы связи магистрали ЭВМ с ВУ (интерфейсы ВУ) имеют в своем составе регистры, через которые и происходит передача информации. Каждому такому регистру (а число их в зависимости от сложности интерфейса может колебаться от 2 до 15–20) присваивается определенный адрес, точно так же, как и ячейкам ОП. При этом адреса регистров ВУ и ячеек ОП не перекрываются: под регистры ВУ всегда отводятся старшие 4К[1] адресов, используемых на данной ЭВМ. Так, для машин, имеющих в составе магистрали 16 адресных линий ("Ээлектроника 60", СМ-1300), адреса регистров ВУ занимают область от 160000 до $177\,7776_8$ (28К...32К)[2]; для машин с 18 линиями адреса (СМ-4, "Электроника 100-25") эта область лежит в диапазоне адресов от 760000 до $77\,7776_8$ (124К...128К); наконец, в машинах СМ-1420, ДВК-3, ДВК-4 (22 адресные линии) под адреса ВУ отведена область от 17 760000 до $1\,777\,7776_8$ (2044К...2048К).

То, что в машинах разных типов регистры ВУ имеют различающиеся физические адреса, не влияет на программирование. Действительно, все эти машины принадлежат к классу 16-разрядных, т. е. имеющих 16-разрядные регистры процессора, 16-разрядные ячейки ОП и 16 линий данных в составе магистрали. Максимальный адрес, который можно указать в программе, составляет $177\,776_8$-$65\,534_{70}$, и, следовательно, адреса регистров ВУ, занимая старшие 4К адресного пространства программы (так называемого *виртуального адресного пространства*), всегда попадают в область 160000...177776_8. Для ЭВМ СМ-1300 или "Электроника 60" этим адресам в программе соответствуют те же самые физические адреса, действительно закрепленные за регистрами ВУ. Для ЭВМ типа СМ-4 и других, работающих с расширенной памятью (более 28К), старшим 4К адресного пространства программы могут соответствовать, вообще говоря, любые физические адреса, в том числе и адреса ОП, а не ВУ. Преобразование виртуальных адресов в физические выполняет *диспетчер памяти*, который ко всем виртуальным адресам программы прибавляет некоторое смещение (задаваемое программами ОС), образуя физические адреса, соответствующие данным виртуальным. Этот

---

[1] Сокращение К здесь и всюду в дальнейшем обозначает 1024 слова (не байта!).

[2] В ЭВМ с системой команд СМ-4 адреса регистров ВУ и ячеек ОП принято указывать в восьмеричной форме.

процесс носит название *отображения*. Для того чтобы работать с регистрами ВУ, требуется сообщить об этом установленным образом ОС. В этом случае диспетчер памяти будет прибавлять к старшим 4К виртуального адресного пространства такое смещение (отличное от смещения для остальных адресов), чтобы эти адреса соответствовали реальным физическим адресам регистров ВУ данной ЭВМ. Например, для ЭВМ СМ-4 смещение составит $600\,000_8$, в результате виртуальному адресу $160\,000_8$ будет соответствовать физический адрес $760\,000_8$, т. е., действительно, адрес регистра ВУ.

1.2. ОПЕРАЦИИ ПЕРЕДАЧИ ДАННЫХ

В ЭВМ семейства "Электроника", как и в СМ ЭВМ, предусмотрены четыре *операции передачи данных:* ввод (чтение слова), ввод–пауза–вывод (чтение слова с паузой), вывод (запись слова) и вывод байта (запись байта). Рассмотрим сначала процесс передачи данных на примере операций чтения и записи на магистрали ОШ. Как уже отмечалось выше, задатчиком при этих операциях в большинстве случаев выступает процессор, а исполнителем ОП или ВУ. На рис. 1.5 показаны линии ОШ, участвующие в операциях чтения-записи, а также направление передачи сигналов по этим линиям между задатчиком и исполнителем. В качестве исполнителя на рис. 1.5 условно показан интерфейс ВУ; для определенности число адресных линий принято равным 18 (ЭВМ СМ-4).

Линии адреса *А00...17* служат для задания адреса регистра ВУ (или ячейки ОП), по которому выполняется обращение в данной операции. Адрес устанавливает задатчик, в данном случае ЦП.

Линии данных *Д00...15* служат для передачи данных из адресуемого регистра в ЦП (в случае операции чтения) или в обратном направлении (в случае операции записи).

Линии управления операциями *У0* и *У1* служат для передачи исполнителю кода выполняемой операции. Сочетание сигналов *У0*-0, *У1*-0

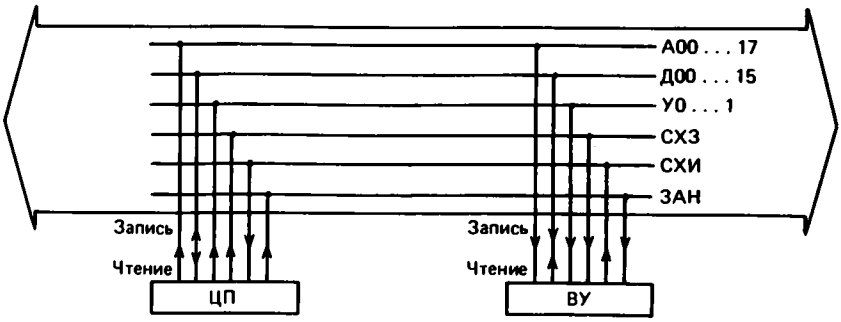


Рис. 1.5. Линии ОШ

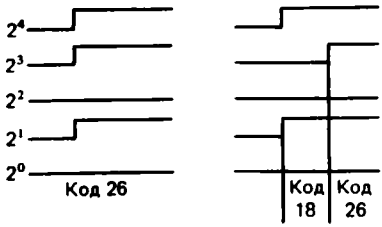


Рис. 1.6. Перекос сигналов на магистрали

соответствует операции чтения слова; *У0*-1, *У0*-0 – операции чтения слова с паузой; *У0*-0, *У1*-1 – операции записи слова и *У0*-1, *У1*-1 – операции записи байта.

Сигнал синхронизации задатчика *СХЗ* информирует исполнителя о том, что задатчик выполнил свою часть операции (установил на ОШ необходимые сигналы) и требует от исполнителя выполнения своей части той же операции.

Сигнал синхронизации исполнителя *СХИ* информирует задатчик о том, что исполнитель завершил свою часть операции.

Сигнал занятости шины *ЗАН* означает, что магистраль занята выполнением операции.

При рассмотрении взаимодействия сигналов ОШ необходимо учитывать так называемый *перекос сигналов*, который возникает из-за неполной идентичности электрических характеристик физических линий ОШ и приемопередающих устройств. Пусть в некоторый момент времени задатчик установил на группе линий заданный код (например, код 26 на рис. 1.6). Через время, определяемое скоростью распространения сигналов по магистрали и временем срабатывания приемных устройств исполнителя, этот код будет принят исполнителем, однако в силу разброса электрических характеристик линий и приемных устройств, подключенных к этим линиям, разряды кода дойдут до исполнителя не одновременно. Если, например (как показано на рис. 1.6), задержка в линии, по которой передается разряд 3, окажется больше, чем в остальных линиях, то в течение некоторого интервала времени на входе исполнителя будет действовать код 18, а не 26. Это и есть перекос сигналов. С целью его компенсации предусматривается, чтобы задатчик, установив на линиях магистрали какой-либо многоразрядный код (адреса или данных) выжидал по меньшей мере 75 нс и только после этого вырабатывал сигнал синхронизации. В результате исполнитель приступит к выполнению своей части операции не раньше того момента, когда на его входе установится правильный код.

Рассмотрим ход выполнения операции записи (рис. 1.7).

Процессор, приступив к выполнению этой операции, устанавливает на линии занятости шины сигнал *ЗАН*, на линиях адреса – адрес того *ВУ*, которому пересылаются данные, на линиях управления – комбинацию сигналов *У0*-0 и *У1*-1 (операция записи слова), а на линиях дан-

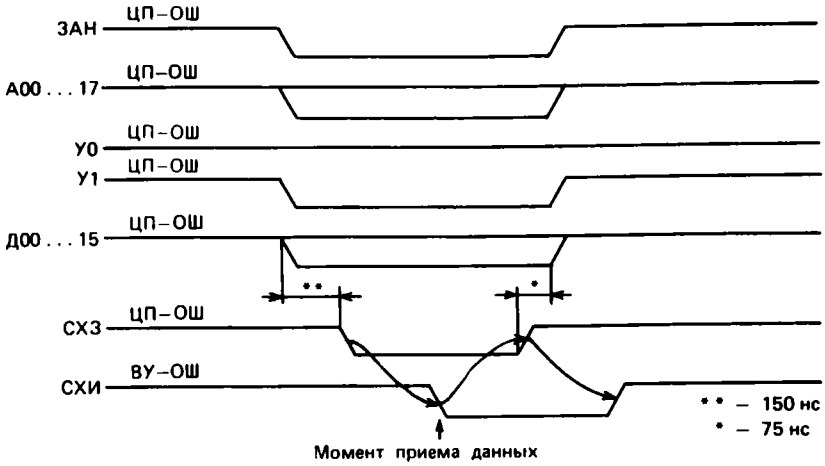


Рис. 1.7. Операция записи

ных – пересылаемые данные (на магистрали ЭВМ эти сигналы отрицательны, хотя по мере прохождения по цепям ЭВМ они могут неоднократно изменять свою полярность). Через 150 нс после этого ЦП устанавливает сигнал *СХЗ*. Код адреса распространяется по магистрали и поступает последовательно во все устройства, подключенные к ОШ. Однако воспринимает этот код только то устройство (в нашем случае конкретный интерфейс ВУ), которому принадлежит установленный на линиях *А* адрес. Интерфейс считывает с линий *А* адрес, с линий *У* – код операции, и по сигналу *СХЗ* выполняет свою часть обмена – считывает с линий *Д* данные и помещает их в свой внутренний регистр. Задержка задатчиком сигнала *СХЗ* на 150 нс, во-первых, позволяет скомпенсировать перекос сигналов адреса и данных (75 нс) и, во-вторых, предоставляет исполнителю время для декодирования адреса (еще 75 нс).

Одновременно с приемом данных интерфейс ВУ вырабатывает сигнал *СХИ*, говорящий о том, что данные приняты. Центральный процессор, получив этот сигнал, снимает сигнал *СХЗ* и через 75 нс все остальные сигналы (*ЗАН, А, У* и *Д*). Задержка компенсирует перекос на линиях адреса и управления и предотвращает случайное срабатывание других устройств, подключенных к магистрали, которое могло бы произойти, если бы во время действия на входах устройств неправильного (вследствие перекоса) адреса сигнал *СХЗ* еще оставался бы установленным.

В приведенном выше описании операции чтения были опущены некоторые детали взаимодействия сигналов. Так, сигнал *СХЗ* может быть установлен только спустя некоторое время после сброса сигнала *СХИ* от предыдущей операции. Однако при рассмотрении принципов построения

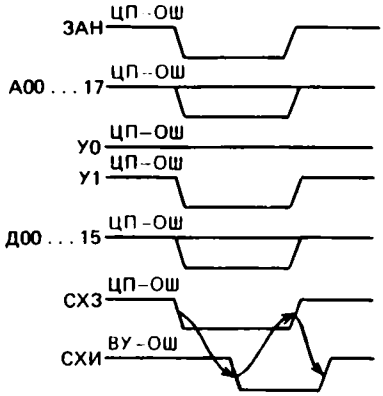


Рис. 1.8. Упрощенная диаграмма операции записи

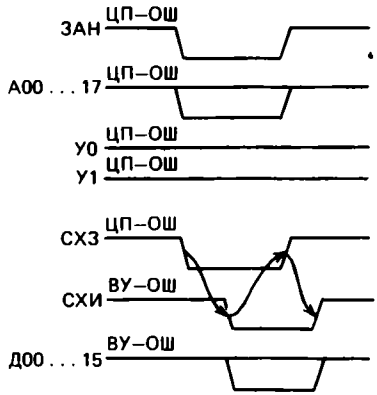


Рис. 1.9. Упрощенная диаграмма операции чтения

и функционирования интерфейсов ВУ этими деталями, так же как и временными задержками, вводимыми для компенсации перекоса, можно пренебрегать. В этом случае диаграммы обмена упрощаются и становятся более наглядными (рис. 1.8).

Рассмотрим теперь операцию чтения, в которой процессор читает данные из некоторого ВУ (рис. 1.9). ЦП устанавливает сигналы *ЗАН, А00...17, У0...1* и *СХЗ*. Исполнитель, получив сигнал СХЗ, подключает к линиям *Д* выходы регистра, в котором хранятся данные, и устанавливает сигнал *СХИ*. Центральный процессор, получив сигнал *СХИ*, принимает данные с линией *Д* в свой внутренний регистр и сбрасывает все установленные им сигналы, а исполнитель, зафиксировав прекращение действия сигнала *СХЗ*, сбрасывает сигнал *СХИ*, на чем операция обмена и прекращается.

Цикл магистрали, т. е. операция чтения или записи, длится на ЭВМ СМ-4 приблизительно 1,2 мкс. Приведенные выше диаграммы сигналов полностью относятся и к тому случаю, когда ЦП обменивается данными не с ВУ, а с ОП.

Рассмотренный протокол, как уже говорилось, характерен для мини-ЭВМ с системным интерфейсом типа ОШ. Интерфейсы, используемые в микроЭВМ "Электроника 60" или ДВК, отличаются от ОШ составом линий и последовательностью взаимодействия сигналов. Важнейшее отличие заключается в том, что в микропроцессорном интерфейсе для сигналов адреса и данных используются одни и те же линии (*мультиплексированные линии* адреса–данных). Для того чтобы сигналы адреса и данных не накладывались друг на друга, они передаются по магистрали не одновременно, как в ОШ, а последовательно: сначала адрес, затем данные.

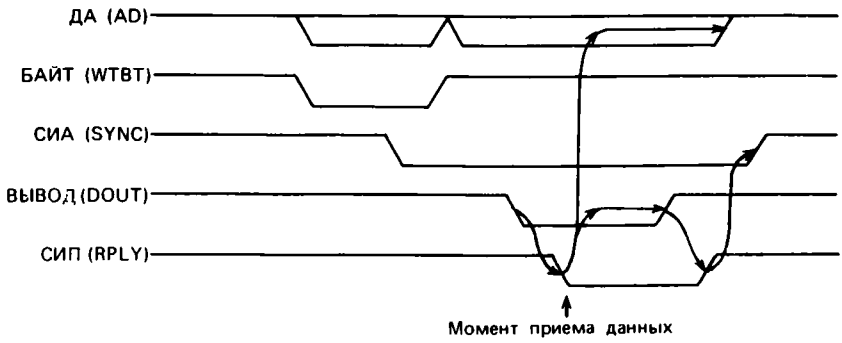


Рис. 1.10. Операция записи на магистрали МПИ

Рассмотрим протокол магистрали микроЭВМ с системным интерфейсом МПИ. Следует заметить, что внутреннее устройство машин, использующих этот протокол, может различаться. Так, ЦП микроЭВМ "Электроника 60" собран из трех БИС серии К581, а в микроЭВМ ДВК используются однокристальные микропроцессоры КМ1801ВМ1, КМ1801ВМ2, КМ1801ВМ3. Различаются эти ЭВМ также составом микропрограммных и аппаратных средств (наличие расширенных наборов команд арифметики с фиксированной и плавающей точкой, процессора с плавающей точкой, расширенной памяти и пр.). В то же время единство системного интерфейса приводит к единой структуре и, более того, к аппаратной совместимости периферийных устройств, подключаемых к ЭВМ.

На рис. 1.10 изображен несколько упрощенный протокол операции *ВЫВОД* (запись) на магистрали МПИ. Поскольку пассивное состояние всех сигналов на магистрали составляет приблизительно +2,5 В, а активное – около нуля, кратковременные сигналы имеют отрицательную полярность. В справочной литературе по микропроцессорам для выводов и сигналов микропроцессоров используются международные обозначения (*AD, RPLY*). В описаниях ЭВМ тем же сигналам обычно даются русские наименования (*ДА, СИП*). На рис. 1.10 и 1.11 приведены оба варианта обозначений.

Начиная операцию, ЦП устанавливает на мультиплексированных линиях адреса – данных *ДА (AD)* адрес ВУ (или ОП при обращении к оперативной памяти). Одновременно с адресом устанавливается сигнал *БАЙТ (WTBT)*, который заменяет сигнал *У1* и характеризует тип операции (запись). После небольшой задержки ЦП устанавливает сигнал *СИА (SYNC)*, приблизительно эквивалентный сигналу *СХЗ* на ОШ и говорящий о том, что адрес установлен на магистрали. Интерфейс ВУ, к которому происходит обращение, получив сигнал *СИА*, запоминает ("защел-

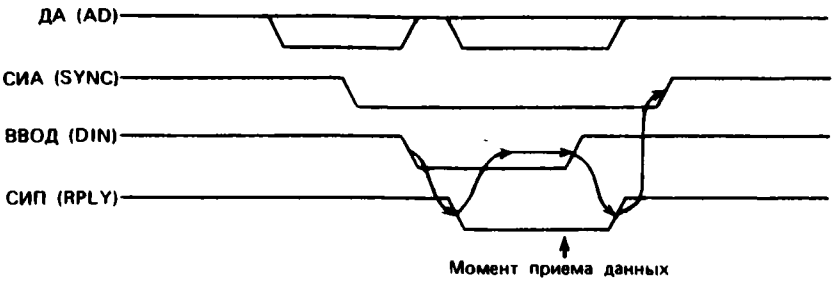


Рис. 1.11. Операция чтения на магистрали МПИ

кивает") снятый с линий *ДА* адрес в своем внутреннем регистре. Это позволяет ему в следующей части цикла магистрали направить принимаемые данные по требуемому адресу (внутри интерфейса), хотя к тому времени код адреса с линий магистрали будет уже снят.

Процессор, выдержав сигналы адреса на магистрали заданное время (около 300 нс для КМ1801ВМ1), снимает их, и сигнал *БАЙТ* сразу же устанавливает на тех же линиях *ДА* код передаваемых в ВУ данных, сопровождая их (с небольшой задержкой) сигналом *ВЫВОД(DOUT)*. Интерфейс ВУ, получив сигнал *ВЫВОД*, отзывается сигналом *СИП* и принимает данные с линий *ДА*. Центральный процессор, получив сигнал *СИП* и выждав около 150 нс для уверенного приема данных пассивным устройством, снимает сигнал *ВЫВОД* и еще через 100 нс – данные с линий *ДА*. Далее ВУ, зафиксировав прекращение действия сигнала *ВЫВОД*, снимает сигнал *СИП*, а ЦП, дождавшись съема сигнала *СИП*, снимает сигнал *СИА*, чем и заканчивается операция на магистрали. Таким образом, сигнал *ВЫВОД*, как и *СИА*, выполняет функцию сигнала *СХЗ* на ОШ, а сигнал *СИП* эквивалентен сигналу *СХИ*. При этом сигнал *СИА* синхронизует передачу процессором адреса, а сигнал *ВЫВОД* – передачу данных; ВУ не подтверждает получение адреса, а на получение данных отзывается сигналом *СИП*.

На рис. 1.11 приведен протокол операции *ВВОД* (чтение) на магистрали МПИ.

Так же как и в случае записи, ЦП устанавливает на линиях *ДА* адрес, синхронизуя его сигналом *СИА*. Отсутствие сигнала *БАЙТ* говорит ВУ о том, что имеет место операция чтения. Интерфейс ВУ по сигналу *СИА* защелкивает адрес. ЦП, сняв с линий *ДА* адрес, устанавливает сигнал *ВВОД(DIN)*, сигнализируя им о своей готовности к приему данных. Интерфейс ВУ, приняв сигнал *ВВОД*, выставляет данные на линии *ДА* и подтверждает их установку сигналом *СИП*. ЦП, получив сигнал *СИП* и выждав некоторое время, принимает данные с магистрали в свой внутренний регистр и снимает сигнал *ВВОД*. Интерфейс ВУ, зафиксировав прекращение действия сигнала *ВВОД*, снимает сигнал *СИП*, а ЦП, обна-

ружив это, снимает сигнал *СИА*, чем и заканчиваются процессы на магистрали.

Из сравнения рис. 1.10 и 1.8, а также 1.11 и 1.19 видно, что, хотя интерфейсы ОШ и МПИ идеологически схожи, их логическая реализация сильно различается, что и обусловливает аппаратную несовместимость машин типа СМ-4 и ДВК (или "Электроника 60"). В то же время не составляет большого труда преобразовать сигналы одного интерфейса в сигналы другого. Эту операцию выполняют устройства сопряжения (*адаптеры магистралей*), позволяющие подключить, например, к ЭВМ "Электроника 60" устройства (НМД, контроллер КАМАК и др.), предназначенные для ЭВМ СМ-4. Возможно и обратное – расширение состава ВУ ЭВМ СМ-4 за счет устройств, предназначенных для работы с ЭВМ "Электроника 60" или ДВК. В качестве примера магистральных адаптеров можно указать модули MQU-*60* и МСМ-60, поставляемые в составе ИВК MERA-*60*. Различаются эти адаптеры тем, что модуль MQU-*60* поддерживает все протоколы магистрали, а МСМ-60 не обеспечивает прямого доступа в память. Для той же цели служит БИС сопряжения К1801ВП1-054, осуществляющая основные операции по преобразованию сигналов системных магистралей ОШ и МПИ.

Следует подчеркнуть, что адаптеры магистралей представляют собой пассивные преобразующие устройства, не допускающие программного управления. Их можно использовать лишь для расширения состава ВУ однопроцессорной ЭВМ, но не для организации двухпроцессорных комплексов. Последнее требует использования программно управляемых межмашинных интерфейсов.

1.3. ПРОЦЕСС ВЫПОЛНЕНИЯ МАШИННЫХ КОМАНД

Рассмотрим участие операций чтения-записи в выполнении машинных команд на примере команды записи числа 5 в регистр ВУ, расположенный по адресу 160010_8: MOV#5, @#160010. Эта команда занимает три слова памяти (рис. 1.12): в первом располагается код команды, во втором – первый операнд или *операнд-источник* (в данном случае число 5), а в третьем слове – абсолютный адрес второго операнда или *операнда-приемника* (восьмеричное число 160010). Команда MOV, у которой в поле адресации одного из операндов указан адрес регистра ВУ, является, можно сказать, основной командой программирования ВУ.

Выполнение любой машинной команды состоит из нескольких *фаз*, число которых зависит от количества операндов в данной команде и от используемых способов адресации. Всего различают четыре фазы; в рассматриваемой команде их три.

Фаза выборки команды. Выполняется операция чтения содержимого ячейки ОП, адрес которой берется из счетчика команд (СК). Как известно, в СК всегда находится адрес очередной команды

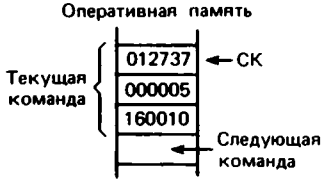


Рис. 1.12. Расположение команды в памяти

выполняемой программы. Чтение содержимого ОП по этому адресу приводит к передаче в один из внутренних регистров ЦП (конкретно – регистр команд) кода очередной команды. Содержимое СК увеличивается на 2, тем самым подготавливая его к чтению следующей команды программы, либо, как в данном случае, к чтению первого операнда.

Фаза выборки операндов. ЦП декодирует команду и по режимам адресации определяет, что во втором и третьем словах команды находятся сам первый операнд и адрес второго. Рассматриваемая фаза в общем случае состоит из двух этапов: выборки операнда-источника и выборки операнда-приемника. Для выполнения первого этапа *исполнительный адрес*, т. е. действительный адрес операнда, берется из СК и выполняется операция чтения из ОП по этому адресу. Операнд-источник считывается в один из внутренних регистров процессора. Содержимое СК увеличивается на 2 и подготавливает СК к выборке следующего операнда. Затем выполняется второй этап: инициируется операция чтения из ОП по адресу, находящемуся в СК, и исполнительный адрес операнда-приемника считывается в другой внутренний регистр процессора. Сам операнд, т. е. содержимое регистра ВУ, в данном случае не извлекается; это было бы необходимо при выполнении команд сложения, сравнения и др.

Фаза выполнения. ЦП выполняет под операндами требуемое командой арифметическое или логическое действие (сложение, сдвиг и т. д.). В случае команды MOV никаких действий над операндами не предусматривается и данная фаза отсутствует.

Фаза записи результата. Выполняется запись результата (в данном случае просто первого операнда) по адресу операнда-приемника. Необходимая для этого информация, т. е. записываемое число и адрес, по которому оно пересылается, содержится во внутренних регистрах процессора. Инициируется цикл магистрали, соответствующий операции записи в интерфейс ВУ.

Таким образом, выполнение команды MOV#5, @#160010 потребовало четырех циклов магистрали (3 операции чтения из ОП и 1 операция записи в ВУ), при этом одно обращение к памяти использовано для считывания кода команды. Легко видеть, что команда MOV R0, @#160010 потребует на одну операцию меньше, так как операнд-источник уже находится в регистре процессора, команда же MOV R0, (R1) выполнится с использованием лишь двух циклов магистрали – одного для считыва-

ния кода команды и второго для записи числа в регистр интерфейса (если адрес этого регистра был предварительно занесен в R1).

Часто фазы выборки команды, выполнения операции и записи результата объединяют, получая фазу выборки-выполнения или основную фазу; фазу выборки операндов, наоборот, разделяют на фазы выборки операнда-источника и операнда-приемника. Соответственно говорят о времени выборки-выполнения или основном времени $T_{осн}$, времени выборки операнда-источника $T_{ист}$ и времени выборки операнда-приемника $T_{пр}$. Полное время выполнения команды $T_к$ можно найти, сложив все три компонента: $T_к = T_{осн} + T_{ист} + T_{пр}$.

В технической документации, выпускаемой изготовителями вычислительных средств, обычно содержатся те или иные сведения о времени выполнения машинных команд. Так, в описании процессора М2 для ЭВМ "Электроника 60М" приведены таблицы $T_{осн}$, $T_{ист}$ и $T_{пр}$ для различных групп команд и разных способов адресации, что дает возможность вычислить $T_к$ для любой реальной команды. Описание микропроцессора К1801ВМ1, используемого в микроЭВМ ДВК-1 и ДВК-2, содержит данные о количестве микропроцессорных тактов и циклов обращения к памяти для каждой команды (с учетом способа адресации), откуда также можно найти $T_к$. Такие оценки бывают полезны при сравнении различных вариантов программ с целью нахождения наиболее оптимального по времени выполнения. Следует только иметь в виду, что полное время выполнения некоторого фрагмента программы не всегда равно сумме времен выполнения команд, составляющих этот фрагмент. Так, в микропроцессоре КМ1801ВМ3 предусмотрено совмещение во времени работы основных блоков процессора: операционного, микропрограммного управления, диспетчера памяти и системной магистрали. В результате одновременно могут идти четыре процесса: выполнение i-й команды в операционном блоке, формирование первой микрокоманды для следующей $(i + 1)$-й команды в блоке микропрограммного управления, чтение из ОП $(i + 2)$-й команды в блоке системной магистрали и формирование физического адреса $(i + 3)$-й команды в диспетчере памяти. Такое совмещение операций заметно повышает быстродействие ЭВМ и соответственно сокращает время выполнения программы по отношению к суммарному времени выполнения входящих в нее команд.

В табл. 1.1 приведены для сравнения ориентировочные значения времен выполнения некоторых команд для разных способов адресации. Видно, что для каждой конкретной ЭВМ времена выполнения различных команд (при одном способе адресации) различаются не более чем в 1,5 раза. С другой стороны, способ адресации влияет на время выполнения очень сильно, изменяя его в несколько раз. Это и понятно, так как чем больший уровень "косвенности" используется при адресации операндов, тем больше циклов обращения к ОП требуется для получения процессо-

Таблица 1.1. **Полное время выполнения некоторых команд для ЭВМ разных типов, мкс**

Команда		Тип ЭВМ			
		"Электроника 60М"	ДВК-1,2 (КМ1801ВМ1)	СМ-4	ДВК-4 (КМ1801ВМ3)
INC	R1	4,8	2,0	1,5	0,5
INC	(R1)	8,0	5,4	3,8	1,2
INC	A	11,2	7,0	4,9	1,3
INC	@A	13,2	8,8	6,2	2,7
ROR	R1	6,0	2,0	1,9	–
ROR	(R1)	9,2	5,4	4,2	—
ROR	A	12,4	7,0	5,3	–
ROR	@A	14,4	8,8	6,6	–
MOV	R1, R2	4,0	2,0	1,3	0,5
MOV	(R1),(R2)	6,8	6,8	4,8	–
MOV	A, B	13,2	10,0	7,0	–
MOV	@A, @B	17,6	14,0	9,6	–
ADD	R1, R2	4,0	2,0	1,5	0,5
ADD	(R1), (R2)	8,8	6,8	5,0	–
ADD	A, B	15,2	10,0	7,0	–
ADD	@A, @B	19,6	14,0	9,7	–
JMP	A	6,8	–	3,6	–
JSR	R5, SUB	12,4	–	5,3	–

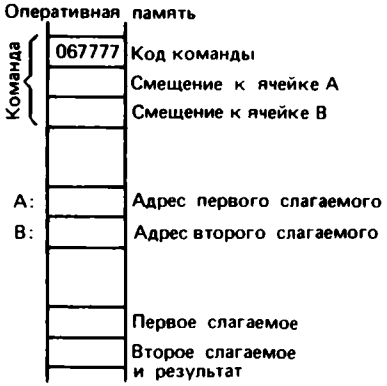


Рис. 1.13. Расположение в памяти операндов команды с косвенной относительной адресацией

ром операнда. Например, команда ADD R1, R2 требует только одного цикла ОП для выполнения фазы выборки команды. Оба операнда уже находятся в регистрах общего назначения процессора, и результат сложения также остается в регистре R2. Время выполнения такой команды незначительно. Команда же ADD @A, @B требует 8 циклов ОП. Действительно, как это следует из рис. 1.13, информация, относящаяся к этой

команде, занимает в общей сложности 7 ячеек памяти. При выполнении команды ЦП последовательно считывает эту информацию (код самой команды, два смещения к адресам операндов, два адреса операндов и, наконец, сами операнды) и при восьмом обращении к ОП записывает в память вычисленную сумму. В результате время выполнения команды увеличивается в зависимости от типа ЭВМ в 5–7 раз.

Таким образом, при разработке оптимальных по времени выполнения программ следует как можно шире использовать регистровые способы адресации и избегать, насколько это возможно, команд с косвенной адресацией через оперативную память.

Глава 2

СТРУКТУРА И ПРОГРАММИРОВАНИЕ ПАРАЛЛЕЛЬНЫХ ИНТЕРФЕЙСОВ

2.1. ПРОСТЕЙШИЙ ИНТЕРФЕЙС

Как уже отмечалось выше, подключение к ЭВМ измерительной аппаратуры выполняется с помощью специальных согласующих схем – интерфейсов. Интерфейс, очевидно, должен, с одной стороны, поддерживать протокол магистрали ЭВМ, т. е. обеспечивать прием и передачу данных через магистраль с помощью стандартных операций записи и чтения, а с другой – управлять подключенным к нему электронным (например, измерительным) оборудованием. При этом управление оборудованием должно осуществляться программным образом, с помощью команд ЭВМ.

В простейшем случае интерфейс (рис. 2.1) содержит узел выбора адреса и синхронизации, а также 16-разрядный регистр данных (РД). Оба узла связаны с каналом ЭВМ через специальные согласующие элементы – приемники *Прм* и передатчики *Прд*. Информация из внешнего электронного оборудования может поступать в регистр данных по 16 входным линиям либо, наоборот, пересылаться из РД во внешнее оборудование по 16 выходным линиям.

Если интерфейс предназначен для передачи кодов из ЭВМ в электронное оборудование (например, ЦАП или цифровой индикатор), то используются только приемники, через которые данные из канала поступают в регистр, и выходные линии регистра. Если же интерфейс планируется использовать для приема кодов в ЭВМ из электронного оборудования (например, АЦП или набора ключей для ручного задания кода), то используются только входные линии регистра и передатчики для передачи данных из регистра в канал.

Рассмотрим функционирование интерфейса. В процессе выполнения операций передачи данных (чтения или записи) ЦП устанавливает на

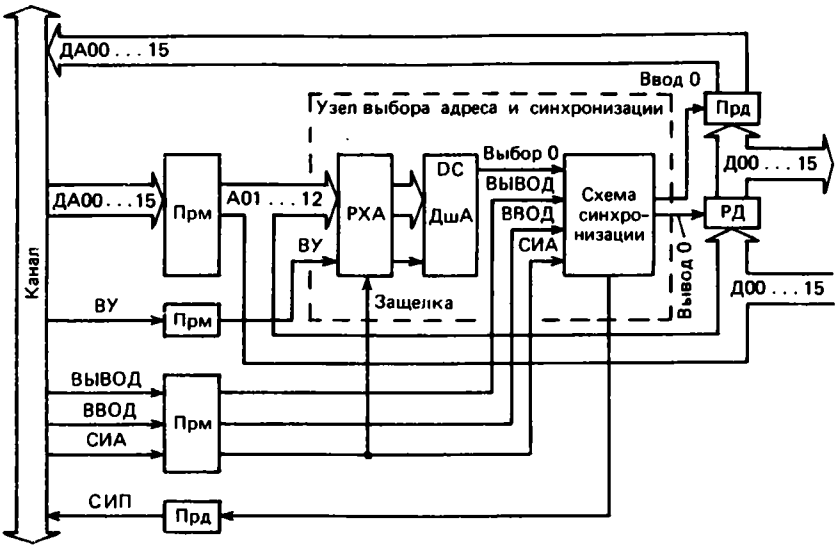


Рис. 2.1 Структурная схема простейшего интерфейса

соответствующих линиях магистрали адрес ВУ и сигнал *СИА*. Поступивший в интерфейс адрес защелкивается по сигналу *СИА* в регистре хранения адреса *РХА*. Если интерфейс предназначен для обмена только целыми словами, состояние сигнала *ДА00* в адресной фазе операции значения не имеет и *РХА* не воспринимается. Из рис. 2.1 видно, что в РХА не поступают также и три старших разряда адреса *13...15*. Вместо них используется один сигнал ВУ, генерируемый процессором или вспомогательными схемами при обращении к любому ВУ. Как известно, в ЭВМ с 16-разрядной адресной шиной адреса ВУ могут принимать значения от $160\,000_8$ до $177\,776_8$. Двоичные представления этих чисел содержат логические 1 в разрядах с *13* по *15*. Поэтому обращение по адресу любого ВУ сопровождается установкой 1 на линиях *ДА13...15*. Сигнал ВУ и образуется процессором как логическое произведение этих трех сигналов. Использование сигнала ВУ вместо трех адресных несколько упрощает схему интерфейса.

Адресные сигналы *А01...12* и ВУ поступают на дешифратор адреса *ДшА*, который служит для декодирования и выделения адреса, принадлежащего данному интерфейсу. Для того чтобы пользователь мог настроить интерфейс на требуемый адрес, в схему *ДшА* вводят съемные перемычки или переключатели, например так, как это показано на рис. 2.2, а. Легко видеть, что если все перемычки установлены в верхнем положении, элемент И сработает при установке на магистрали любого из четырех адресов 177 770, 177 772, 177 774 и $177\,776_8$ (поскольку

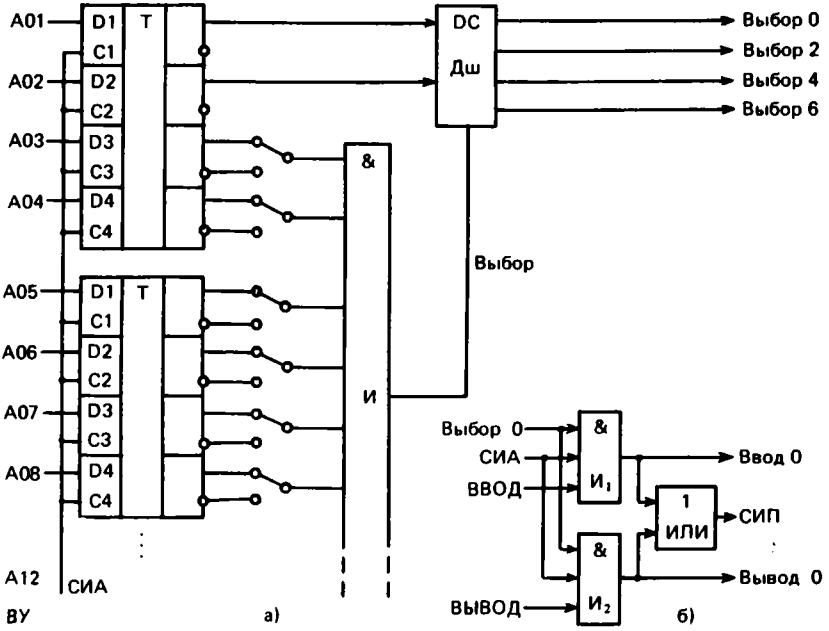


Рис. 2.2. Структурная схема дешифратора адреса

адресные линии *A01* и *A02* не подсоединены к элементу И). Первый из этих адресов можно назвать базовым и присвоить ему индекс 0; тогда остальные адреса интерфейса получат индексы 2, 4 и 6. Простейший интерфейс, изображенный на рис. 2.1, содержит лишь один регистр и требует, следовательно, только один адрес. Однако чаще в интерфейсе имеются два, три или четыре регистра, поэтому стандартная схема дешифратора адреса рассчитана на декодирование четырех смежных адресов, начиная с базового, который устанавливается с помощью съемных перемычек. Если перенести перемычку на линии *A03* с прямого на инверсный выход триггера *РХА*, базовый адрес интерфейса изменится на 177760_8, при переносе перемычки на линии *A04* адрес будет равен 177750_8 и т. д. При переносе всех перемычек (*A03...12*) интерфейс окажется настроен на минимально возможный базовый адрес $160\,000_8$.

Сигнал *Выбор* с выхода элемента И поступает на дешифратор *Дш*, который вырабатывает при наличии сигнала *Выбор* один из сигналов *Выбор 0, Выбор 2, Выбор 4* или *Выбор 6* в зависимости от состояния адресных линий *A01* и *A02*. В схеме, приведенной на рис. 2.1, используется лишь один сигнал *Выбор 0*.

Схема синхронизации (рис. 2.2, *б*) служит для преобразования сигнала *Выбор 0* в один из двух сигналов *Ввод 0* или *Вывод 0* в зависимости

от того, какая операция (чтение или запись) выполняется на магистрали. В случае операции чтения схема синхронизации вырабатывает сигнал *Ввод 0*, который открывает передатчики *Прд*, связывающие РД с магистралью (передатчики в простейшем случае представляют собой просто элементы И, установленные в каждом разряде линий данных), и данные из РД поступают на линии *ДА* канала. Одновременно в схеме синхронизации вырабатывается сигнал *СИП*, поддерживающий протокол магистрали. Таким образом, ЦП читает слово, содержащееся в РД. В случае операции записи возникает сигнал *Вывод 0*, который поступает на тактовые входы всех триггеров РД. Поскольку в соответствии с протоколом магистрали в этот момент на линиях *ДА* процессор установил пересылаемые в ВУ данные, эти данные, пройдя через приемники (представляющие собой постоянно открытые вентили), записываются в РД. Одновременно вырабатывается подтверждающий сигнал *СИП*.

Рассмотрим программирование описанного интерфейса. Пусть базовый адрес, устанавливаемый с помощью перемычек, равен 164000_8. Тогда команда MOV R1, 164000 приведет к записи в регистр интерфейса содержимого регистра R1 процессора, а команда MOV 164000, ABC выполнит считывание данных из регистра интерфейса и пересылки их в ячейку ОП с меткой ABC.

Как уже отмечалось, для управления ВУ можно использовать весь набор команд ЭВМ. Так, команда CLR 164000 должна очистить регистр интерфейса, а команда INC 164000 – увеличить содержимое регистра интерфейса на 1. Однако не всегда это будет так. Действительно, в процессе выполнения команды INC процессор считывает содержимое адресуемого регистра в свой внутренний регистр, прибавляет там единицу и переносит результат назад в регистр интерфейса. Для правильного выполнения этой команды в интерфейсе должны быть предусмотрены возможности как чтения из регистра, так и запись в него. Однако, как отмечалось выше, если интерфейс предназначен только для вывода информации, в нем может не быть передатчиков для чтения данных из регистра на магистраль, если же интерфейс предназначается только для ввода, в нем может не быть приемников. В таком случае команды INC, DEC, ADD, BIS, BIC и др. будут выполняться неправильно. Перед тем как использовать команды изменения содержимого, необходимо удостовериться в том, что все разряды адресуемого регистра допускают как чтение, так и запись.

В приведенных выше программных строках для обращения к регистру интерфейса использовалась относительная адресация. Тот же результат был бы получен при использовании абсолютной адресации: MOV R1, @#164000. Оба варианта команд занимают два слова памяти, однако в первом случае во втором слове команды записывается смещение к адресу 164000_8, а во втором случае – сам адрес. Но и выполняются команды по-разному: в первом варианте исполнительный адрес нахо-

дится прибавлением смещения к содержимому СК (в сумме получится 164 000), а во втором в СК просто переносится адрес (164 000) из второго слова команды. Вариант с относительной адресацией относится к числу *позиционно-зависимых кодов:* такая команда правильно выполняется лишь в одном единственном месте виртуального адресного пространства программы, именно в том, для которого было определено смещение к исполнительному адресу, записанное в коде команды. Однако в большинстве случаев прикладная программа после трансляции и компоновки получает фиксированные виртуальные адреса и никогда не перемещается по виртуальному адресному пространству. Это, между прочим, справедливо как для однопользовательских, так и для многопользовательских многозадачных систем. Только в некоторых специальных случаях при разработке программы надо обеспечить ее позиционную независимость. Так, в *позиционно-независимом коде* пишутся системные драйверы, поскольку при загрузке они попадают в произвольные области виртуального пространства программы; иногда позиционной независимостью должны обладать программы обработки прерываний.

Интерфейс, подобный рассмотренному, применен, например, в многоканальном цифро-аналоговом преобразователе (ЦАП) для ЭВМ "Электроника 100-25". Устройство служит для преобразования цифровых данных, поступающих от ЭВМ, в напряжение постоянного тока в диапазоне от –10 до +9,9976 В с погрешностью менее 0,025%. ЦАП имеет четыре независимых канала, каждому из которых соответствует свой регистр данных (РД). Адреса регистров: 176 760, 176 762, 176 764 и $176\,766_8$; код выходного напряжения занимает в каждом регистре разряды 0...12, причем разряд 12 является знаковым. Всего, таким образом, можно получить $2^{12} = 4096$ значений выходного напряжения, из которых половина – положительные, а половина – отрицательные, как это видно из следующей таблицы, где значения кодов указаны в восьмеричной форме:

Код в РД	Выходное напряжение, В
000000	+0,0000
000001	+0,0024
000002	+0,0049
000003	+0,0073
.	.
.	.
.	.
007776	+9,9952
007777	+9,0076
010000	+10,0000
010001	–9,9976
.	.
.	.
.	.

017775	−0,0073
017776	−0,0049
017777	−0,0024

Пусть нужно установить на выходах ЦАП напряжения −10, −5, 0 и +5 В. Это делается следующей группой команд:

```
MOV  #10000, 176760
MOV  #14000, 176762
CLR  176764
MOV  #4000, 176766
```

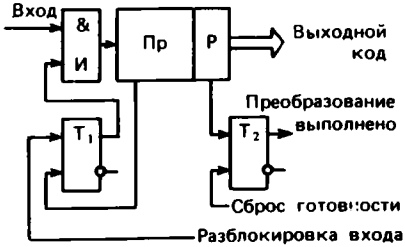
Существенным недостатком рассмотренного интерфейса является невозможность контролирования его состояния. Пусть мы используем интерфейс для ввода в ЭВМ данных, которые поступают в регистр интерфейса из измерительной установки. Для считывания содержимого регистра используется команда вида MOV 164000, ABC. Однако ниоткуда не следует, что к моменту выполнения команды новые данные поступили в регистр интерфейса. Выполняя эту команду повторно без анализа состояния интерфейса, мы рискуем несколько раз считать одни и те же данные. Точно так же, если интерфейс используется для пересылки данных в электронное устройство, перед выполнением команды MOV R1, 164000 следует убедиться, что это устройство готово принять данные, в противном случае команда выполнится впустую и данные будут потеряны. Для выполнения указанных проверок интерфейс снабжается средствами контроля готовности внешнего устройства принять или передать данные. Поскольку эти проверки осуществляются программно, такой интерфейс можно назвать интерфейсом с программным управлением.

2.2. ИНТЕРФЕЙС С ПРОГРАММНЫМ УПРАВЛЕНИЕМ

Рассмотрим в качестве примера измерительного электронного устройства АЦП, упрощенная структурная схема которого приведена на рис. 2.3. АЦП служит для преобразования амплитуд входных импульсов (или значений непрерывно действующего на входе напряжения) в двоичные коды, направляемые далее в ЭВМ для накопления и обработки и состоит из собственно преобразователя *Пр* с линейным элементом на входе, выходного регистра *Р*, куда записывается код амплитуды входного импульса, триггера T_1 блокировки входа и триггера T_2 указания готовности АЦП к передаче в ЭВМ очередного кода. Разрядность выходного кода определяет точность измерения амплитуды импульса; показанный на рисунке 12-разрядный АЦП обеспечивает $2^{12} = 4096$ уровней квантования входного сигнала.

Алгоритм программного управления работой АЦП следующий. По команде от ЭВМ вход АЦП разблокируется. При поступлении входного импульса преобразователь фиксирует его амплитуду, блокирует с помощью триггера T_1 свой вход, чтобы исключить искажения из-за наложения

Рис. 2.3. Структурная схема АЦП



входных импульсов, и начинает преобразование амплитуды импульса в цифровой код. После того как в выходном регистре установится код амплитуды, взводится триггер T_2, вырабатывая сигнал готовности. В таком состоянии АЦП находится до тех пор, пока ЭВМ, обнаружив тем или иным способом сигнал готовности, не приступит к обслуживанию АЦП. Обслуживание заключается в приеме выходного кода, сбросе триггера готовности и разблокировке входа АЦП. После этого АЦП снова переходит в состояние ожидания входного импульса.

Если АЦП рассчитан на измерение непрерывно действующего на его входе напряжения, то тот же, по существу, алгоритм обслуживания приобретает несколько иной смысл. Получив сигнал разблокировки входа, АЦП фиксирует действующее в настоящий момент значение входного напряжения, блокирует свой вход, преобразует напряжение в код и устанавливает триггер готовности. ЭВМ, обнаружив сигнал готовности, считывает код из регистра *Р* с одновременным сбросом триггера готовности. Если сигналы разблокировки входа вырабатываются периодически, период их следования определяет период дискретизации измеряемого напряжения.

Таким образом, для управления АЦП интерфейс должен вырабатывать сигналы разблокировки входа и сброса готовности и принимать из АЦП сигналы готовности и выходного кода. Упрощенная структурная схема интерфейса приведена на рис. 2.4.

От рассмотренного ранее простейшего интерфейса интерфейс с программным управлением отличается наличием дополнительного передатчика, связанного с линией *Д07*, а также элемента И, образующего логическое произведение сигналов *Вывод 0* и *Д00*. Первый используется для считывания в ЭВМ состояния готовности АЦП (обращением по базовому адресу 0 интерфейса); второй – для генерации сигнала разблокировки входа (обращением по тому же адресу 0). Поскольку выходной код АЦП содержит только 12 разрядов, число передатчиков здесь уменьшено до 12; считывание кода выполняется командой чтения по адресу 2 интерфейса. Сигнал *Ввод 2* можно использовать и для сброса триггера готовности. В этом случае триггер (и сигнал) готовности будет сбрасываться одновременно с чтением выходного кода АЦП, что соответствует описанному выше алгоритму.

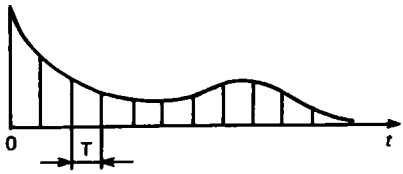


Рис. 2.7. Измерение переходного процесса

равен T и всего надо выполнить 100 замеров. Базовый адрес интерфейса будем по-прежнему считать равным 160020_8:

```
CSR = 160020                ; Символические
DBR = 160022                ; обозначения РКС и РД
DATA:   .BLKW 100.          ; Область для данных
START:  MOV #DATA, R1       ; Регистр для косвенной
                            ; адресации
        MOV   #100., R2     ; Счетчик цикла
; Запуск процесса
1 $ :   INC   CSR           ; Разблокировка АЦП
2 $ :   TSTB  CSR           ; Ожидание
        BPL   2 $           ; готовности
        MOV   DBR,(R1)+     ; Прием кода в буфер ОП
; Задержка на время T
        SOB   R2, 1 $       ; Повторить 100 раз
```

Метка START в приведенной программе относится к первой программной строке и отделяет, таким образом, поля данных от выполнимых программных строк. Эта условность будет использована и в последующих примерах.

Приведем теперь пример программы, выполняющей распределение кодов от АЦП по каналам. Для 12-разрядного АЦП понадобится 4096 каналов. Будем считать, что требуется накопить 256 000 кодов (в среднем 64 кода в канале):

```
CSR = 160020                ; Символические
DBR = 160022                ; обозначения РКС и РД
DATA:   .BLKW 4096.         ; Буфер под спектр
START:  CLR   R0            ; Счетчик внутреннего цикла
        MOV   #1000., R1    ; Счетчик внешнего цикла
1 $ :   INC   CSR           ; Запуск АЦП
2 $ :   TSTB  CSR           ; Ожидание
        BPL   2 $           ; готовности
        MOV   DBR, R2       ; В R2 код входного импульса
        ASL   R2            ; В R2 код × 2
        INC   DATA(R2)      ; Инкремент в канале
        DECB  R0            ; Организация
        BNE   1 $           ; внутреннего цикла из 256 шагов
        SOB   R1, 1 $       ; Внешний цикл из 1000 шагов
```

Примером промышленного АЦП с программным управлением является модуль аналогового ввода для ЭВМ "Электроника 60". Модуль преобразует напряжение постоянного тока (как положительное, так и отрицательное), поступающее по любому из 16 входных каналов, в цифровой код. Для выбора требуемого входного канала и подключения его к АЦП служит программно-управляемый мультиплексор входных аналоговых сигналов. Модуль воспринимает входной сигнал в диапазоне –9,9...+10,1 В, имеет время преобразования около 100 мкс и погрешность преобразования 0,5% полной шкалы.

Управление мультиплексором осуществляется с помощью регистра адреса данных (РАД), расположенного по адресу 160070_8. В регистре используются разряды 0...3, в которые записывается номер входного канала. Каждое программное обращение к РАД с целью записи нового номера канала приводит к стробированию входного аналогового сигнала и соответственно к пуску преобразователя. Выходной код АЦП поступает в разряды 0...9 РД с адресом 160072_8. Готовность АЦП (окончание преобразования) фиксируется установкой разряда готовности, в качестве которого используется разряд 10 РКС с адресом 160074.

Приведем программу последовательного опроса всех 16 входов и записи кодов входных напряжений в 16-словный массив в ОП.

```
DAR = 160070                  ; Регистр адреса данных
DBR = 160022                  ; Регистр данных
CSR-160074                    ; Регистр команд и состояний
DATA:   .BLKW 16.             ; Буфер под данные
START:  MOV    #16., R1       ; Счетчик опросов
        MOV    #DATA, R2      ; Адрес массива в R2
        CLR    DAR            ; Канал 0, пуск АЦП
1$ :    BIT    #2000, CSR     ; Анализ разряда 10,
        BEQ    1$             ; ожидание готовности
        MOV    DBR, (R2)+     ; Код в ОП
        INC    DAR            ; Следующий канал, пуск АЦП
        SOB    R1, 1$         ; Цикл из 16 шагов
```

Режим программного управления (программирования по готовности) широко используется в измерительных установках, поскольку обеспечивает максимальную скорость приема информации. Действительно, рассмотрим фрагмент программы приема данных из измерительной аппаратуры с минимально возможной обработкой (накопление кодов в ОП в порядке их поступления):

```
1$  : INC    CSR            ; Пуск ВУ
2$    TSTB   CSR            ; Ожидание
      BPL    2$             ; готовности
      MOV    DBR, (R1)+     ; Прием данных
      BR     1$             ; Возврат на пуск
```

Приведенный фрагмент выполняется на ЭВМ СМ-4 за 18 мкс, на ЭВМ "Электроника 60" за 38 мкс. Таким образом, максимальная скорость накопления информации составляет соответственно $5{,}5 \cdot 10^4$ и $2{,}6 \cdot 10^4$ событий/с. Это очень высокие скорости для аппаратуры, управляемой от ЭВМ.

С другой стороны, в режиме программного управления ВУ пассивно; момент обращения к ВУ определяется программой, а не реальными внешними условиями (например, приходом сигнала на вход измерительной аппаратуры). Если регистрируемые события возникают относительно редко, режим программного управления оказывается чрезвычайно неэффективным в вычислительном плане: в цикле ожидания готовности ЦП занят опросом разряда готовности и не может выполнять другой, более полезной работы, например обработки или визуализации поступающей информации. Если в установку входят несколько источников измерительной информации, то ожидание (возможно, бесплодное) готовности одного из них лишает возможности опрашивать остальные. Конечно, в этом случае можно организовать последовательный, кольцевой опрос источников, но это увеличит программные издержки и снизит скорость накопления информации:

```
1$  : TSTB    CSR1    ; ВУ1 готово?
      BPL     2$      ; Нет, опрос следующего
; Строки приема информации из ВУ1
2$  : TSTB    CSR2    ; ВУ2 готово?
      BPL     3$      ; Нет, опрос следующего
; Строки приема информации из ВУ2
3$  : TSTB    CSR3    ; ВУ3 готово?
      BPL     4$      ; Нет, опрос следующего
       .
       .
       .
      JMP     1$      ; Возврат на опрос ВУ1
```

Неэффективное использование ЦП, невозможность организации параллельных процессов (сбор данных – обработка) ограничивает использование режима программного управления. В этом смысле гораздо более эффективным является режим прерываний.

2.3. ИНТЕРФЕЙС С ОБСЛУЖИВАНИЕМ ПРЕРЫВАНИЙ

Режим прерываний существенно отличается от рассмотренного выше режима программного управления. Если ВУ работает в режиме прерываний, то в оперативной памяти ЭВМ находятся две программы: *основная* (или фоновая), которая может и не иметь отношения к внешним устройствам, и *программа обработки прерываний* (ПОП), основной задачей которой обычно является передача очередной порции информации между

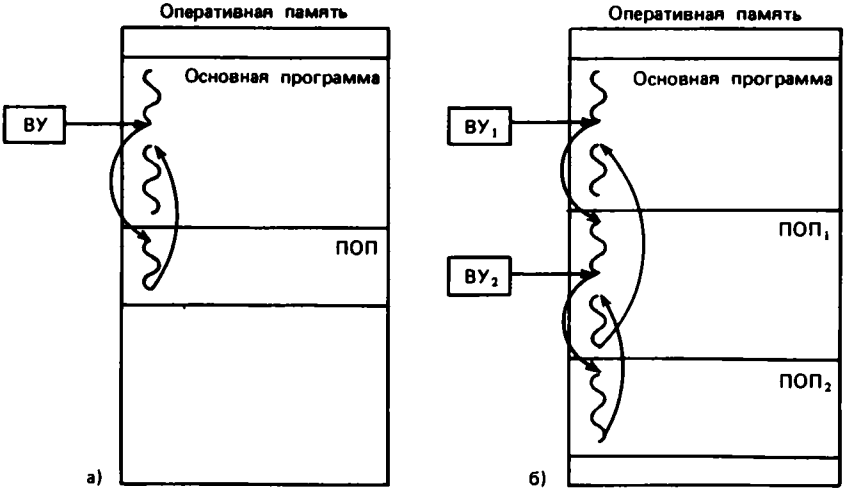


Рис. 2.8. Процедура прерываний

ВУ и ЦП или ОП (рис. 2.8, *а*). Процессор постоянно занят выполнением основной программы. Если ВУ готово к передаче или приему информации, оно посылает в ЭВМ по соответствующим электрическим линиям сигнал запроса прерывания. Процессор прерывает основную программу и переходит на выполнение ПОП. После завершения ПОП управление передается назад в основную программу, которая продолжается с того места, на котором была прервана сигналом от ВУ.

Режим прерываний обеспечивает высокую эффективность использования времени ЦП: если ВУ не готово к обмену данными, процессор выполняет основную программу, т. е. занят полезной работой. В режиме прерываний ЭВМ может обслуживать параллельно любое количество ВУ. Каждому ВУ соответствует своя программа обработки прерываний (рис. 2.8, *б*). Если запрос на прерывание приходит от $ВУ_2$, управление передается на $ПОП_2$, а после ее завершения выполнение прерванной программы продолжается. При этом в некоторых ЭВМ возможны вложенные прерывания: если запрос на прерывание от $ВУ_2$ придет в тот момент, когда выполняется $ПОП_1$, то $ПОП_1$ прерывается (при условии, что приоритет $ВУ_2$ выше приоритета $ВУ_1$) и управление передается $ПОП_2$. После завершения обработки прерывания от $ВУ_2$ продолжается обработка прерывания от $ВУ_1$. По завершении $ПОП_1$ управление передается в основную программу. Использование в системе прерываний приоритетов позволяет организовать обслуживание ВУ в порядке важности. В первую очередь обслуживаются ВУ с максимальным приоритетом; если запросов от этих ВУ не поступило или если они уже обслужены, обрабатываются прерывания от менее приоритетных ВУ. Наконец, в про-

межутках между обработкой прерываний выполняется основная программа.

Рассмотрим процесс передачи управления от программы к программе. Для того чтобы осуществить переход на ПОП, достаточно загрузить в СК адрес первой выполняемой строки ПОП, т. е. значение метки, относящейся к этой строке, а в регистр состояния процессора (РСП) – значение приоритета ЦП при выполнении ПОП. Однако при этом надо обеспечить возможность возврата в прерванную программу и продолжения ее выполнения так, как будто никакого прерывания не было. С этой целью надо запомнить в какой-то области памяти следующую информацию:

адрес очередной команды прерываемой программы, т. е. содержимое СК в момент прерывания;

содержимое регистра РСП, т. е. слово состояния процессора (ССП) на момент прерывания. В разрядах кодов условий РСП может храниться информация, необходимая для выполнения следующей команды, если она представляет собой, например, команду условного перехода;

содержимое регистров общего назначения (РОН), если они будут использоваться в ПОП.

Для возвращения в прерванную программу надо восстановить содержимое СК, РСП и РОН.

Система прерываний обеспечивает при получении сигнала прерывания автоматическое сохранение в стеке содержимого СК и РСП, (*вектора текущего процесса*), загрузку новых значений СК и РСП, относящихся к ПОП, а после завершения ПОП – выгрузку из стека содержимого СК и РСП, относящегося к прерванной программе. Содержимое РОН автоматически не сохраняется. Если в ПОП предусматривается использование РОН, сохранение их содержимого в начале ПОП и восстановление в конце должен предусмотреть программист.

Значения СК и РСП, относящиеся к каждой программе обработки прерываний, а их может быть столько, сколько ВУ подсоединено к ЭВМ, должны быть заранее помещены в какую-то область памяти, доступную процессору. Для этого в схеме распределения адресов предусматривается область памяти для *векторов прерываний*. Вектор прерываний – это два слова оперативной памяти, закрепленные за определенным ВУ. В слове с меньшим адресом хранится адрес ПОП для данного ВУ, в слове с большим адресом – ССП для ПОП. Фактически в ССП заполняются только разряды 5–7, где указывается приоритет процессора при выполнении ПОП. Под адресом вектора понимается адрес младшего слова пары; таким образом, адреса векторов кратны 4 и могут оканчиваться только на 0 или 4.

Для векторов прерываний выделено самое начало ОП с адресами от 000 до 776_8 (всего 256 ячеек), где можно разместить 128 векторов. Область векторов прерываний делится на зоны. Зона с адресами $000...276_8$ отведена под векторы стандартных ВУ, таких, как консольный терминал,

сетевой и программируемый таймеры, печатающее устройство, НМД и НМЛ, перфоленточные и перфокарточные устройства ввода-вывода и др. В этой же зоне часть адресов ($040...056_8$) отводится под область коммуникации ОС с прикладными программами, а часть ($000...036_8$) – под векторы внутренних прерываний для перехода на программы обработки таких ситуаций, как попытка выполнения неправильной команды либо обращения к недоступной памяти, сбоя питания и т. д. Несколько адресов этой зоны зарезервированы для устройств пользователя. Это адреса 170, 174, 270 и 274_8. По этим адресам можно располагать векторы прерываний нестандартной, в частности, измерительной аппаратуры.

Зона с адресами $330...776_8$, в которой может поместиться 80 векторов, называется зоной *изменяемых* или *плавающих векторов*. Она отводится под нестандартные устройства пользователя. Следует, однако, иметь в виду, что операционная система РАФОС, загружая программу в ОП, использует ячейки с 776 по 500_8 (96 слов) под стек. В этом случае в зоне плавающих векторов остается место под 32 вектора, что, впрочем, вполне достаточно для большинства применений.

Рассмотрим теперь аппаратные средства организации прерываний в ЭВМ типа СМ-4 (рис. 2.9). В составе ОШ имеются четыре пары линий *запроса передачи ЗП4–ЗП7* и *разрешения передачи РП4–РП7*, различающиеся уровнем приоритета и, кроме того, пара аналогичных линий для запроса прямого доступа *ЗПД* и *РПД* (запрос на прямой доступ не приводит к прерыванию программы, но аппаратная реализация прямого доступа схожа с реализацией системы прерываний). В системе прерываний имеется 4 уровня приоритетов ВУ со значениями от 4 до 7. Устройство приобретает тот или иной приоритет в зависимости от того, к какому уровню прерываний оно подсоединено. На одном уровне может находиться любое количество устройств. При этом если к линиям *ЗП* все ВУ подсоединяются параллельно, то каждая линия *РП* проходит последовательно через все устройства, находящиеся на этом уровне. Это нужно для того, чтобы реализовать приоритетность устройств, принадлежащих к одному уровню. Чем дальше от процессора подсоединено

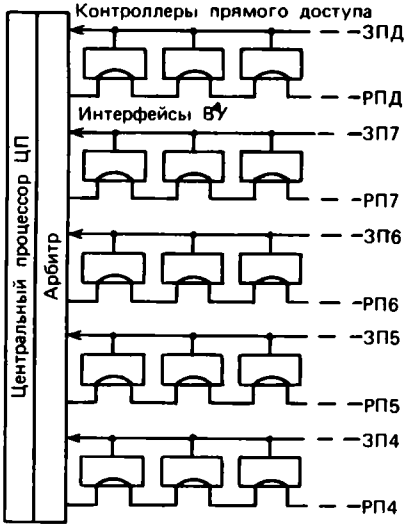


Рис. 2.9. Аппаратная реализация системы прерываний

к линии *РП* устройство, тем меньшим относительным приоритетом среди устройств этого уровня оно обладает.

Как только в ВУ возникает состояние готовности, интерфейс ВУ выставляет запрос на соответствующую линию *ЗП*, т. е. устанавливает на этой линии уровень логической 1. Специальный узел процессора, носящий название *арбитра*, сравнивает номер возбужденной линии *ЗП* с приоритетом процессора при выполнении текущей программы, хранящимся в битах 5–7 ССП. Если приоритет линии запроса ниже или равен приоритету выполняемой программы, прерывания не происходит. Линия запроса при этом остается возбужденной, т. е. прерывание как бы ждет своей очереди. Когда приоритет процессора станет ниже приоритета возбужденной линии (в результате перехода на программу с меньшим приоритетом или появления команды, устанавливающей более низкий приоритет в выполняемой программе), арбитр немедленно инициирует процедуру прерывания.

Интерфейс, получив по линии *РП* сигнал разрешения передачи, выставляет на линии данных ОШ адрес своего вектора прерывания, а на специальную линию прерывания – сигнал, информирующий процессор о начале процедуры прерывания. Получив этот сигнал, процессор проталкивает в стек текущее ССП и содержимое СК (т. е. адрес возврата в прерванную программу) и загружает СК и РСП из вектора прерываний, чем и реализуется переход на ПОП.

Программа обработки прерываний заканчивается специальной командой RTI (Return from interrupt – возврат из прерывания), которая выталкивает из стека ССП и адрес возврата в прерванную программу и загружает их соответственно в РСП и СК. На этом обработка прерывания заканчивается, и процессор переходит на продолжение выполнения прерванной программы.

Рассмотрим временные характеристики системы прерываний. На рис. 2.10 изображена последовательность сигналов ОШ, участвующих в процедуре прерываний. На каждой диаграмме указано название линии ОШ, по которой передается данный сигнал, а также его источник и приемник.

Интерфейс, готовый к обмену, выставляет запрос на линию *ЗПi*. Этот запрос принимается арбитром, который начинает процедуру арбитража, т. е. сравнение номера линии запроса прерывания i с текущим приоритетом процессора p. Если $i>p$, арбитр после окончания исполнения текущей команды устанавливает сигнал разрешения передачи *РПi*, одновременно приостанавливая прием других запросов по линиям *ЗП* до окончания обработки текущего запроса. Если $i \leqslant p$, ничего не происходит, запрос ждет снижения приоритета процессора.

Интерфейс, приняв сигнал *РПi*, подтверждает его получение установкой сигнала на линии *ПВБ* (подтверждение выборки). Арбитр, получив сигнал *ПВБ*, блокируется и сбрасывает сигнал *РПi*, на чем заканчивается процедура арбитража.

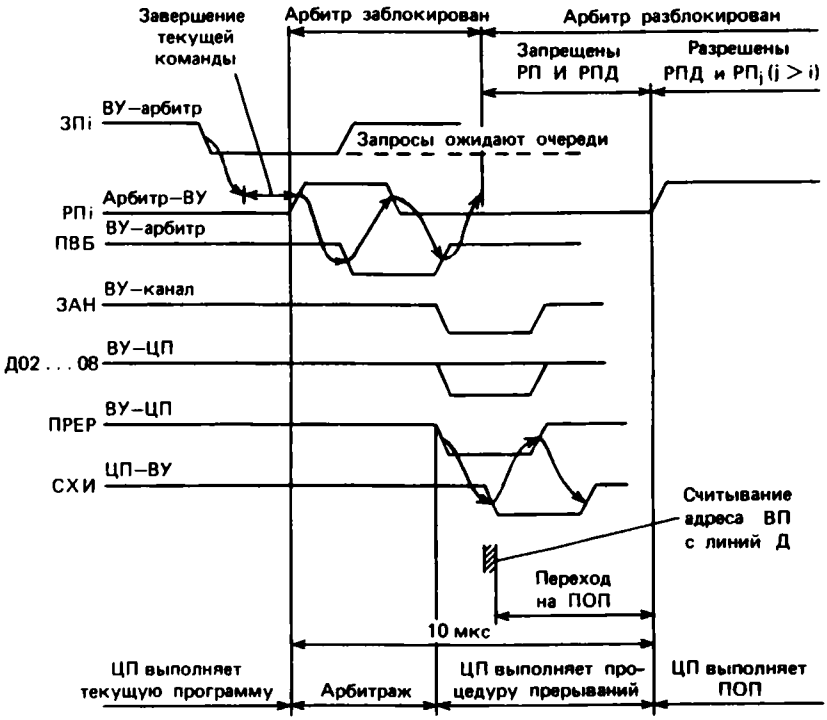


Рис. 2.10. Протокол прерываний

Интерфейс, приняв сброс *РПi*, начинает цикл передачи в процессор адреса своего вектора прерываний. Для этого интерфейс устанавливает сигнал на линии *ЗАН*, оповещая все другие устройства ЭВМ о том, что ОШ занята данным устройством. Начиная с этого момента интерфейс становится задатчиком на ОШ; исполнителем в данном случае является процессор. Интерфейс выставляет на линии данных *Д02...Д08* адрес вектора прерываний, посылает в процессор сигнал прерывания по линии *ПРЕР* и сбрасывает сигнал *ПВБ*. Сброс *ПВБ* разблокирует арбитр, и он может приступить к обработке очередного сигнала *ЗП*. Однако сигнал *ПРЕР* запрещает арбитру выдачу сигнала *РП* вплоть до окончания процедуры прерывания (т. е. до передачи управления ПОП).

Процессор, получив сигнал *ПРЕР*, задерживает его на 75 нс для компенсации перекоса, затем стробирует (принимает) адрес вектора прерывания и в подтверждение этой операции выдает сигнал *СХИ*.

Интерфейс, приняв сигнал *СХИ*, снимает адрес вектора с линий *Д*, сбрасывает сигналы *ПРЕР* и *ЗАН*, освобождая шину.

Процессор, приняв сброс сигнала *ПРЕР*, подтверждает этот факт сбросом сигнала *СХИ* и приступает к процедуре перехода на ПОП, заключающейся в проталкивании в стек текущего ССП и содержимого СК и загрузке СК и РСП из вектора прерываний. В результате выполнения этой процедуры управление передается ПОП.

Арбитраж приоритетов и процедура прерывания занимают в ЭВМ СМ-4 около 10 мкс. Для перехода из ПОП в основную программу (выполнение команды RTI) требуется еще 4,5 мкс. В результате общие потери времени на каждое прерывание (аппаратные издержки системы прерываний) составляют для ЭВМ СМ-4 около 15 мкс. Таким образом, скорость накопления информации в режиме прерываний будет всегда ниже, чем в режиме ожидания готовности. При этом надо иметь в виду, что указанное значение издержек относится лишь к тому случаю, когда обработка прерываний выполняется без участия ОС. Реально, однако, процедура перехода на ПОП включает в себя некоторые программные действия, состав которых определяется используемой ОС. Выполнение процедуры прерывания по правилам ОС гарантирует правильное взаимодействие программ реального времени в случае наложения прерываний от нескольких источников, в том числе и от тех, которые обслуживаются автоматически средствами ОС (например, прерывания от системного таймера или от контроллера НМД). Программная составляющая потерь (системные издержки) на каждое прерывание в несколько раз превышает аппаратную составляющую и для ОС РАФОС составляет около 100 мкс на ЭВМ СМ-4 и около 200 мкс на ЭВМ "Электроника 60".

Система прерываний допускает обработку вложенных прерываний. Действительно, если в процессе выполнения ПОП придет запрос на прерывание по линии с более высоким приоритетом, вся процедура прерывания повторится: ССП и СК текущей программы (в данном случае ПОП) протолкнутся в стек, из соответствующего вектора прерываний будет получена информация о ПОП более приоритетного устройства, и начнется обслуживание этого ВУ. По команде RTI произойдет возврат в ПОП ВУ с более низким приоритетом, а по такой же команде RTI, заканчивающей эту программу, процессор перейдет на продолжение выполнения основной прерванной программы.

Устройства, подключенные к одному уровню, не могут прерывать друг друга, поскольку они обладают одинаковым приоритетом по отношению к процессору. Их относительный приоритет, определяемый порядком подключения к линии *РП*, влияет на последовательность их обслуживания в том случае, когда несколько устройств, находящихся на одном уровне, одновременно выставили запросы на прерывание. Ближайшее к процессору ВУ, приняв сигнал разрешения передачи, блокирует его дальнейшее прохождение (см. рис. 2.9), т. е. ВУ, находящиеся на одном уровне, обслуживаются в порядке их расположения на линии *РПi*.

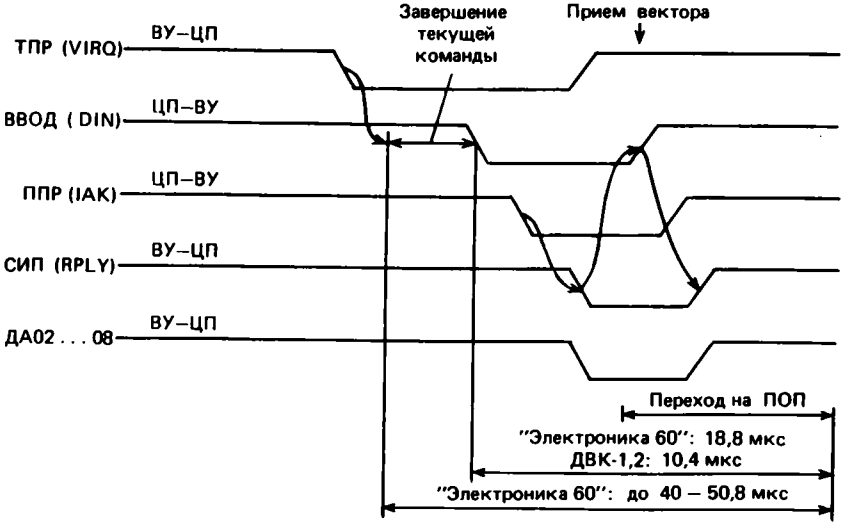


Рис. 2.11. Протокол прерываний в микропроцессоре КМ1801ВМ1

Кроме четырех уровней программных прерываний с приоритетом от 4 до 7 существует еще уровень *внепроцессорного прерывания (прямого доступа в память)*. К этому уровню подсоединяются интерфейсы (контроллеры) прямого доступа, которые могут осуществлять обмен данными непосредственно с оперативной памятью без участия процессора. Устройства, подсоединенные к уровню внепроцессорного прерывания, имеют наибольший приоритет. Арбитр, получив запрос на прямой доступ в память, разрешает передачу данных независимо от текущего приоритета процессора. Работа канала прямого доступа будет рассмотрена в § 2.5.

Система прерываний в микроЭВМ семейств "Электроника" и ДВК отличается от рассмотренной некоторыми деталями. Прежде всего, как уже отмечалось, различаются протоколы системных магистралей. Системный протокол прерываний интерфейса МПИ для микропроцессора КМ1801ВМ1 приведен на рис. 2.11. Устройство, требующее обслуживания, устанавливает на линии *ТПР(VIRQ)* сигнал требования прерывания. Центральный процессор, получив этот сигнал и дождавшись освобождения магистрали, устанавливает сигнал *ВВОД(DIN)* и вслед за ним сигнал подтверждения прерывания *ППР(IAK)*. Внешнее устройство, приняв сигнал *ППР*, блокирует его дальнейшее прохождение, отвечает процессору сигналом *СИП(RPLY)*, устанавливает на линиях *ДА02...08* адрес своего вектора прерываний и снимает сигнал *ТПР*. Центральный процессор, получив сигнал *СИП*, снимает сигнал *ВВОД* и принимает адрес вектора с линий *ДА*. ВУ, зафиксировав прекращение действия сигнала

ВВОД, снимает сигнал *СИП* и адрес вектора с линий *ДА*. На этом процедура обмена сигналами заканчивается. Центральный процессор, получив адрес вектора, осуществляет переход на *ПОП* точно таким же образом, как и в рассмотренном выше протоколе ОШ.

Микропроцессор КМ1801ВМ3 имеет четырехуровневую систему прерываний (хотя только одну линию разрешения прерывания *IAK*, проходящую последовательно через все подключенные к ЭВМ интерфейсы). Таким образом, система прерываний ЭВМ ДВК-4 принципиально не отличается от системы прерываний ЭВМ СМ-4. То же можно сказать о миниЭВМ семейства "Электроника" и микроЭВМ "Электроника 60-1". Однако ЭВМ "Электроника 60" и "Электроника 60М" имеют одноуровневую систему прерываний, в которой все ВУ обладают одинаковым приоритетом. В РСП этих машин для указания приоритета процессора используется один разряд 7, установка которого соответствует высокому приоритету (все прерывания от ВУ запрещены), а сброс – низкому (прерывания от ВУ разрешены). Таким образом, программам обработки прерываний следует назначать приоритет 4 (или 5, 6, 7, что приведет к тем же результатам); при этом вложенные прерывания будут запрещены. Если же приоритет процессора при выполнении ПОП будет равен 0, запрос на прерывание от ВУ (в том числе того же самого), поступивший во время выполнения ПОП, приведет к вложенному прерыванию.

Аппаратные издержки системы прерываний в ЭВМ ДВК-1, 2 приблизительно такие же, как и в ЭВМ СМ-4. Для ЭВМ "Электроника 60" длительность процедуры прерывания составляет 18,8 мкс, а время выполнения команды RTI 10 мкс. Суммарное время составляет 28,8 мкс, что в 2 раза больше, чем для ЭВМ СМ-4.

Для того, чтобы ВУ могло работать в режиме прерываний, в его интерфейсе должны быть предусмотрены соответствующие элементы. Структурная схема такого интерфейса (для рассмотренного выше АЦП) приведена на рис. 2.12. В интерфейс введен ряд дополнительных элементов. Триггер *Т* служит для разрешения или запрещения прерываний по командам ЭВМ. Для установки триггера (и разрешения прерываний) надо установить разряд 6 по адресу 0 интерфейса. Поскольку выход триггера связан через передатчик с линией *Д06*, чтение слова по адресу 0 приводит к считыванию состояния триггера *Т*. Узел прерываний осуществляет весь протокол прерываний, изображенный на рис. 2.11. Для этого на его вход подается сигнал запроса прерывания, который при наличии сигнала (или, лучше сказать, уровня) разрешения прерываний преобразуется в сигнал *ТПР*. С каналом узел прерываний связан через приемопередатчики *Прм–Прд* всеми сигналами, необходимыми для реализации протокола прерываний: *ТПР, ППР, ВВОД, СИП*. Кроме того, модуль имеет выход на линии данных *Д02...08*, по которым передается адрес вектора прерываний. Этот адрес устанавливается в узле прерываний пользователем с помощью съемных перемычек.

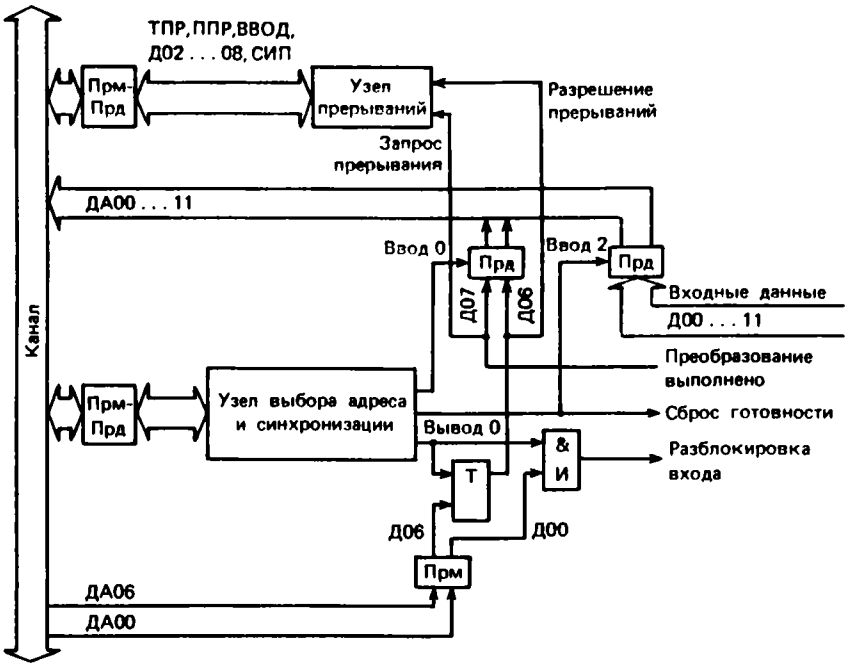


Рис. 2.12. Структурная схема интерфейса, работающего в режиме прерываний

Структурная схема программного комплекса по управлению АЦП в режиме прерываний приведена на рис. 2.13. Программирование ВУ в режиме прерываний требует проведения некоторых подготовительных операций: заполнения вектора прерываний обслуживаемого устройства, разрешения прерываний и запуска ВУ (в рассматриваемом примере – разблокировки входа АЦП). Если предполагается обслуживать одновременно несколько устройств, указанные операции (инициализация прерываний) выполняются для каждого ВУ в отдельности. Прерывания инициализируются обычно в начале основной программы, которая затем может перейти к выполнению запланированных для нее действий.

Программа обработки прерываний принимает из АЦП очередной код, разблокирует вход АЦП, подготавливая его к регистрации следующего входного импульса и завершается с помощью команды RTI возврата из прерывания. При необходимости в ПОП можно включить команды первичной обработки принятого кода с целью, например, отбора кодов, лежащих в заданном диапазоне (цифровое окно). Следует только иметь в виду, что удлинение ПОП снижает скорость регистрации входных событий. Поэтому в ПОП желательно включать лишь

Рис. 2.13. Структурная схема программного комплекса с обработкой прерываний

ту обработку, которую необходимо выполнять в реальном времени поступления событий.

Простейшая программа приема кодов из АЦП в режиме прерываний и распределения их по каналам приведена ниже. Предполагается, что базовый адрес интерфейса АЦП по-прежнему равен 160020, а вектор прерываний расположен по адресу 300_8:

```
CSR = 160020                ; Символические
DBR = 160022                ; обозначения РКС и РД
DATA:   .BLKW 4096.         ; Буфер под спектр
START:  MOV  #INT, 300      ; Адрес ПОП в ВП
        MOV  #240, 302      ; Приоритет 5 в ВП
        MOV  #101, CSR      ; Разрешение прерываний и
                            ; разблокировка входа

        ;; Продолжение основной программы
        .
        .
INT:    MOV  DBR, R4        ; Код из АЦП в R4
        ASL  R4             ; Код × 2 в R4
        INC  DATA (R4)      ; Инкремент канала
        INC  CSR            ; Разблокировка входа
        RTI                 ; Выход из прерывания
```

Разнообразная измерительная и управляющая аппаратура, подключаемая к ЭВМ, часто сочетает возможности работы как в режиме программного управления, так и в режиме прерываний. Рассмотрим несколько примеров промышленной аппаратуры такого рода.

Для ЭВМ "Электроника 100-25" выпускается 64-канальный АЦП, который преобразует входное напряжение в пределах от –10 до +9,9976 В с погрешностью не более 0,1% полной шкалы; время преобразования не более 30 мкс. Формат регистров АЦП приведен на рис. 2.14. Разряд 0 РКС управляет пуском АЦП. Разряд носит импульсный характер: засылка в него 1 приводит к стробированию напряжения, действующего

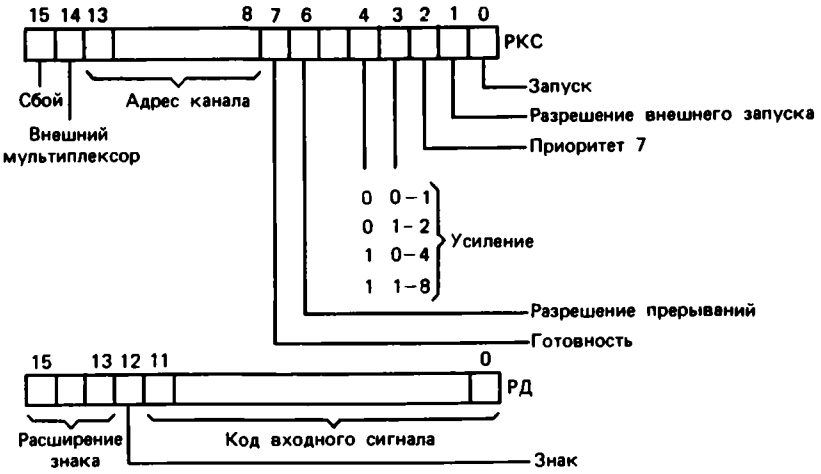


Рис. 2.14. Регистры АЦП

на входе выбранного канала, и запуску схемы, преобразующей это напряжение в выходной код. По окончании преобразования устанавливается разряд 7 РКС и, если разряд 6 установлен, возникает прерывание. С помощью разрядов 3 и 4 РКС устанавливается коэффициент усиления внутреннего усилителя. При смене усиления происходит запуск преобразователя. Адрес канала устанавливается в разрядах 8...13 РКС, изменение адреса канала также запускает преобразователь. Фактически АЦП запускается при любом программном обращении к РКС. Разряд 15 служит для фиксации сбоя, он устанавливается, если новое преобразование инициируется при незавершенном предыдущем. Если разряд 6 РКС установлен, сбой инициирует прерывание. В АЦП предусмотрена возможность запуска внешним сигналом (а не программно). Разрешение внешнего запуска дается установкой разряда 1 РКС, при этом программный запуск блокируется. Имеется также возможность подключения к АЦП внешнего мультиплексора (например, на другое число каналов), для чего надо установить разряд 14 РКС. Работа внутреннего мультиплексора при этом блокируется.

Разряд 2 РКС служит для программной установки приоритета АЦП. Если разряд 2 установлен, запрос на прерывание идет по линии 7 и АЦП имеет наибольший приоритет. При сброшенном разряде 2 уровень запроса (4, 5 или 6) устанавливается пользователем с помощью перемычек на плате интерфейса.

Пусть надо последовательно опросить все 64 канала АЦП и занести результаты измерений в ОП. Предположим, что входные напряжения не превышают 1 В, тогда для повышения точности преобразования

целесообразно установить коэффициент усиления, равный 8. При такой постановке задачи нет необходимости использовать режим прерываний; напротив, режим программного управления обеспечит большую скорость измерений. Возможный вариант программы:

```
CSR = 176770                    ; Обозначение РКС
DBR = 176772                    ; Обозначение РД
DATA:   .BLKW 64.               ; Место под результаты измерений
START:  MOV     #64., R1        ; Счетчик шагов цикла
        MOV     #DATA, R2       ; Адрес массива результатов
        MOV     #31, CSR        ; Канал 0, усиление 8, запуск
1$  :   TSTB    CSR             ; Ожидание конца
        BPL     1$              ; преобразования
        MOV     DBR, (R2)+      ; Код в ОП
        ADD     #400, CSR       ; Инкремент номера канала и
                                ; запуск
        SOB     R1, 1$          ; Повторить 64 раза
```

Если моменты измерений определяются какими-то внешними событиями, относительно редкими по машинным масштабам, целесообразно использовать режим прерываний. Вектор прерываний АЦП располагается по адресу 130_8 (АЦП принадлежит к числу устройств, для которых место под векторы прерываний зарезервировано в зоне стандартных устройств). Предположим, что результаты измерений по каналу 63 надо накапливать в ОП и число измерений не превысит 1000. Программные строки, относящиеся к измерениям, могут выглядеть следующим образом:

```
CSR = 176770                    ; Обозначение РКС
DBR = 176772                    ; Обозначение РД
VEC = 130                       ; Адрес вектора прерываний
DATA:   .BLKW 1000.             ; Место под результаты измерений
ADDR:   .WORD DATA              ; Указатель текущего адреса
START:  MOV  #INT, VEC          ; Адрес ПОП в вектор
        MOV  #240, VEC+ 2       ; Приоритет 5 в вектор
        MOV  #37502, CSR        ; Канал 63, внешний запуск,
                                ; усиление 1, режим прерываний

          ;; Продолжение основной программы
          .

INT:    MOV  DBR, @ADDR         ; Код из РД в ОП
        ADD  #2, ADDR           ; Смещение текущего адреса
        RTI                     ; Возврат в основную программу
```

Переход на ПОП в приведенном примере осуществляется после того, как АЦП, запущенный внешним сигналом, выполнил преобразование входного напряжения в код. В ПОП этот код переносится в ОП по адре-

су, хранящемуся в ячейке ADDR, после чего указатель в ADDR смещается к следующей (свободной) ячейке в массиве DATA.

Для временно́й синхронизации измерительного процесса в состав ИВК часто включается *программируемый таймер*, с помощью которого можно точно задавать длительность или период измерений. Такие таймеры, подключаемые к системной магистрали, разработаны для многих малых ЭВМ: СМ-4, "Электроника 60", "Электроника 100-25" и др. Следует заметить, что в современных ЭВМ имеется так называемый *системный*, или *сетевой, таймер*, работающий с частотой сети 50 Гц. Обращение к этому таймеру осуществляется с помощью директив ОС, в составе которой имеются специальные программы управления таймером.Точность измерения интервалов времени системным таймером невелика из-за нестабильности частоты сети. В тех случаях, когда требуется измерение интервалов времени с высокой точностью, можно использовать программируемый таймер, частота которого задается кварцевым генератором и отличается высокой стабильностью. Программируемый таймер подключается к системной магистрали через интерфейс, подобный описанному, и содержит три программно доступных регистра: регистр команд и состояний РКС с адресом $172\,540_8$, регистр буферный РБ с адресом $172\,542_8$ и регистр-счетчик РСч с адресом $172\,544_8$.

Буферный регистр РБ служит для хранения задаваемого программно интервала времени. Единицы измерения интервала определяются рабочей частотой таймера, которая устанавливается с помощью разрядов 1 и 2 РКС (формат РКС приведен на рис. 2.15).

Регистр-счетчик (РСч) предназначен для отсчета заданного интервала времени, значение которого переносится в него автоматически из РБ. Направление счета задается с помощью разряда 4 РКС; обычно используется обратный счет. Как только (в процессе обратного счета) содер-

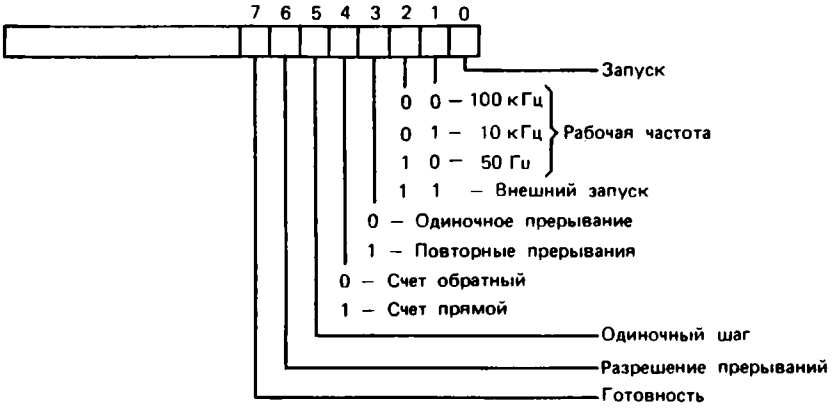


Рис. 2.15. Регистр команд и состояний программируемого таймера

жимое РСч окажется равным нулю, устанавливается разряд 7 РКС и инициируется прерывание (если разряд 6 РКС установлен). При этом возможны два режима работы таймера: одиночных и повторных прерываний (устанавливаются с помощью разряда 3 РКС). В первом режиме после отсчета заданного интервала времени и инициирования прерывания работа таймера заканчивается. В режиме повторных прерываний в момент окончания отсчитываемого интервала РСч перезагружается из РБ, и начинается повторный отсчет заданного интервала. Каждый раз при окончании очередного интервала инициируется прерывание. Этот режим дает возможность отсчитывать большие интервалы времени, организовав в программе цикл счета прерываний. В режиме одиночных прерываний максимальный интервал времени при рабочей частоте 10 кГц равен

$$T_{\text{макс}} = N/f = 65\,536/10^4 \approx 6{,}5 \text{ с},$$

где $N = 2^{16}$; f – частота кварцевого генератора таймера.

Для отсчета более продолжительных интервалов времени приходится либо переходить на менее стабильную частоту сети, либо работать в режиме повторных прерываний.

Режим внешнего запуска, устанавливаемый засылкой в РКС числа 5, позволяет либо работать с внешним генератором, настроенным на требуемую частоту, либо использовать таймер как счетчик внешних импульсов (и в том, и в другом случае импульсы подаются на специальный вход таймера).

Таким образом, для запуска таймера надо настроить его на требуемый режим, записав в РКС соответствующее управляющее слово, а в РБ – значение отсчитываемого интервала времени и установить разряд 0 РКС. Обычно таймер используется в режиме прерываний; тогда в ПОП таймера предусматриваются команды управления элементами измерительной установки.

Рассмотрим несколько примеров программирования установки, включающей в себя рассмотренный выше АЦП и программируемый таймер. Пусть требуется получить распределение (спектр) значений непрерывно изменяющегося напряжения постоянного тока, отображающего некоторый стационарный процесс. Для квантования напряжения и преобразования его в код служит АЦП, а частота дискретизации задается внешним генератором. Зададим время измерения 5 мин, после чего таймер должен выключить АЦП и передать управление на программу обработки. Поскольку в процессе накопления спектра не предполагается выполнять какую-либо вычислительную работу, прием кодов от АЦП целесообразно осуществлять в режиме ожидания готовности. Программа управления установкой выглядит следующим образом:

```
CSRTMR = 172540          ; РКС таймера
BRTMR = 172542           ; Буферный регистр таймера
CNTTMR = 172544          ; Регистр-счетчик таймера
```

```
VECTMR = 104                          ; Вектор таймера
CSRADS = 176 770                      ; РКС АЦП
DBRADC = 176 772                      ; РД АЦП
SPECTR:  .BLKW  4096.                 ; Место под спектр
COUNT:   300.                         ; Время измерения 300 с
START:   MOV    #CLOCK, VECTMR        ; Адрес ПОП таймера
         MOV    #300, VECTMR + 2      ; Приоритет 6
         MOV    #10000., BRTMR        ; Интервал 10 000 тактов = 1 с
         MOV    #2, CSRADC            ; Канал 0, усиление 1,
                                      ; внешний запуск
         MOV    #113, CSRTMR          ; Повторение прерывания, 10 кГц,
                                      ; разрешение прерываний, запуск
1$ :     TSTB   CSRADC                ; Ожидание
         BPL    1$                    ; готовности АЦП
         MOV    DBRADC, R1            ; Прием кода
         ASL    R1                    ; Умножение на 2
         INC    SPECTR (R1)           ; Инкремент в канале
         BR     1$                    ; На новый цикл ожидания
CLOCK:   BIC    #200, CSRTMR          ; Сброс разряда готовности
         DEC    COUNT                 ; Счет числа прерываний
         BEQ    1 $                   ; Переход после 300 прерываний
         RTI                          ; Выход из ПОП
1 $ :    CLR    CSRTMR                ; Остановка таймера
         MOV    #PROCES, (SP)         ; В стек адрес перехода
         RTI                          ; Выход из ПОП по новому адресу
PROCES:                               ; Начало программы обработки
```

В приведенном примере частота таймера и интервал отсчета выбраны так, чтобы прерывания от таймера инициировались 1 раз/с. В программе организован счетчик прерываний, настроенный на подсчет 300 прерываний. Засылка в РКС управляющего слова (число 113_8) приводит к заданию рабочей частоты 10 кГц и режима повторных прерываний, а также разрешению прерываний и одновременно запуску таймера. После этого программа переходит к бесконечному циклу накопления спектра в режиме ожидания готовности.

Прерывания от таймера (приходящие 1 раз/с) приостанавливают цикл накопления кодов на время работы ПОП таймера. В ПОП отсчитывается число прерываний (в ячейке COUNT); после завершения ПОП командой RTI управление возвращается в цикл накопления кодов от АЦП. После прихода 300-го прерывания, т. е. по истечении 300 с, происходит переход на участок ПОП, начинающийся с локальной метки 1 $. Таймер останавливается очисткой РКС, в ячейку стека, предназначенную для хранения адреса возврата, засылается адрес участка программы, содержащего строки обработки спектра, и по команде RTI выполняется выход из прерывания с передачей управления на метку PROCES. Следует иметь в виду, что такой способ передачи управления по внешнему событию, несмотря на внешнюю простоту и эффективность, имеет ограни-

ченную область применения и годится лишь для тех программ, которые не используют (в программе обработки прерываний) системных средств.

Время выполнения цикла приема информации в приведенном примере составляет приблизительно 50 мкс (с учетом времени преобразования АЦП). Таким образом, максимальная скорость приема информации составляет $2 \cdot 10^4$ измерений/с. Если измеряемое напряжение мало отклоняется от среднего уровня, результаты измерений будут попадать в ограниченное число каналов массива SPECTR и возникнет вероятность переполнения и соответственно потери информации в отдельных каналах. Существуют разные методы борьбы с этим явлением. Одним из простейших является прекращение измерений, если хотя бы один из каналов переполнился. При этом необходимо определить истекшее время, которое, очевидно, будет меньше запланированного (5 мин). Посмотрим, как будет выглядеть цикл приема кодов от АЦП в этом случае:

```
REMNS:  300.                        ; Ячейка для остатка секунд
REMNT:  10000.                      ; Ячейка для остатка тактов
1$ :    TSTB   CSRADC               ; Ожидание
        BPL    1 $                  ; готовности АЦП
        MOV    DBRADC, R1           ; Прием кода
        ASL    R1                   ; Умножение на 2
        INC    SPECTR (R1)          ; Инкремент в канале
        CMP  SPECTR (R1), #65535.   ; Канал заполнен?
        BEQ    2 $                  ; Да, переход на останов
        BR     1 $                  ; Нет, возврат
2 $:    CLR    CSRTMR               ; Останов таймера
        SUB    COUNT, REMNS         ; Число прошедших секунд
        SUB    CNTTMR, REMNT        ; Число прошедших тактов
PROCES:                             ; Начало программы обработки
```

В программе резервируются две дополнительные ячейки: REMNS для сохранения числа прошедших секунд и REMNT для сохранения числа пришедших тактов после последнего прерывания (длительность такта 10^{-4} с). В цикле накопления спектра после очередного инкремента в канале осуществляется проверка содержимого канала. Если число в канале меньше 65535, накопление продолжается. Если же канал заполнился, происходит переход на участок отработки "аварийного" останова, прекращается работа таймера, в ячейку REMNS заносится разность между полным временем измерения в секундах и числом оставшихся секунд, находящимся в ячейке COUNT, а в ячейку REMNT – разность между полным числом тактов в секунде и числом оставшихся тактов, которое берется из РСч таймера. После этого программа может приступить к отработке спектра.

Рассмотрим пример, в котором таймер используется для периодического запуска измерительной аппаратуры. Пусть требуется периодически, скажем, 1 раз в 5 с, опрашивать все 64 канала АЦП, собирать информацию в ОП и обрабатывать ее по некоторому алгоритму. Очевидно, что

АЦП должен работать в режиме ожидания готовности, таймер – в режиме повторных прерываний. Текст программы может выглядеть следующим образом:

```
CSRTMR = 712 540                    ; РКС таймера
BRTMR = 172 542                     ; РБ таймера
CSRADC = 176 770                    ; РКС АЦП
DBRADC = 176 772                    ; РД АЦП
DATA:   .BLKW 64.                   ; Место под данные
ADDR:   .WORD DATA                  ; Указатель текущего адреса
FLAG:   0                           ; Флаг
START:  MOV   #CLOCK, 104           ; Заполнение вектора
        MOV   #300, 106             ; прерываний таймера
        MOV   #50000, BRTMR         ; Интервал 50000 тактов, равный 5 с
        MOV   #113, CSRTMR          ; Пуск таймера
ACQ:    TST   FLAG                  ; Ожидание
        BEQ   ACQ                   ; установки флага
        CLR   FLAG                  ; Сброс флага
        MOV   #64., R1              ; Счетчик
        MOV   #1, CSRADC            ; Канал 0, запуск АЦП
1$ :    TSTB  CSRADC                ; Ожидание
        BPL   1$                    ; готовности
        MOV   DBRADC, @ADDR         ; Код в ОП
        ADD   #2, ADDR              ; Смещение указателя
        ADD   #400, CSRADC          ; Инкремент номера канала и запуск
        SOB   R1, 1$                ; Повторить 64 раза
        MOV   #DATA, ADDR           ; Восстановление указателя

        ;; Обработка накопленных данных
        JMP   ACQ                   ; Переход на ожидание флага
CLOCK:  BIC   #200, CSRTMR          ; Сброс разряда готовности
        INC   FLAG                  ; Установка флага
        RTI                         ; Выход из прерывания
```

После инициализации прерываний и запуска таймера программа входит в цикл ожидания установки флага, в качестве которого используется содержимое ячейки FLAG. В исходном состоянии флаг сброшен. Единственное действие, выполняемое в ПОП таймера, – это установка флага. Сразу же после возврата из прерывания в основную программу цикл ожидания установки флага разрывается и программа, прежде всего, сбросив флаг, переходит на опрос всех 64 каналов АЦП (в режиме ожидания готовности) и обработки полученных данных. Перед началом обработки восстанавливается исходное положение указателя в ячейке ADDR для того, чтобы в следующем цикле измерений (по прошествии 5 с) накопление данных из АЦП началось снова с адреса DATA. По окончании обработки программа возвращается на ожидание установки флага.

Недостатком описанной программы является невозможность выполнять какую-либо вычислительную работу параллельно с измерениями, даже если цикл измерений и обработки занимает менее 5 с. Однако организация параллельных процессов обычно требует системных средств и будет рассмотрена в последующих главах.

2.4. ИНТЕРФЕЙС ПРЯМОГО ДОСТУПА В ПАМЯТЬ

Как было показано выше, режим прерываний удобен при обработке случайных и относительно редких событий и, кроме того, позволяет обслуживать одновременно много ВУ. Однако прием каждой порции информации из измерительной аппаратуры в ОП требует в режиме прерываний значительного времени. Рассмотрим простейший случай, когда коды, поступающие из измерительной аппаратуры, просто накапливаются в выделенной для этого области оперативной памяти в порядке поступления, без всякого анализа или обработки. Программа обработки прерываний, обеспечивающая этот процесс, будет состоять по меньшей мере из четырех команд: пересылки слова данных из РД в ОП, инкремента указателя адреса в ОП, инкремента счетчика накопленных кодов и, наконец, разблокировки (запуска) измерительной аппаратуры. Чистое время выполнения ПОП составит на ЭВМ СМ-4 около 25 мкс. К этому времени надо добавить аппаратные издержки системы прерываний – около 15 мкс, а также системные издержки, составляющие минимум 100 мкс. В результате получается, что для пересылки в ОП одного слова данных затрачивается около 150 мкс процессорного времени. Если скорость поступления регистрируемых событий высока, значительная часть процессорного времени (в приведенном примере – до 85%) будет тратиться на непроизводительную работу переходов на ПОП и обратно. В таких условиях сама идея прерываний теряет смысл, так как фактически у ЦП не остается времени на выполнение основной программы. Более эффективным может оказаться режим ожидания готовности, если в конкретной ситуации можно смириться с его недостатками – обслуживанием только одного ВУ и невозможностью организовать параллельное выполнение основной программы.

В тех случаях, когда требуется регистрировать измерительную информацию, поступающую с высокой средней скоростью, существенного повышения эффективности работы вычислительного комплекса можно добиться, использовав прямой доступ в память (ПДП). Идея ПДП заключается в том, что программа обслуживания ВУ выполняется не программно, как обычно, а аппаратно, с помощью схем, содержащихся в интерфейсе ВУ (интерфейсе ПДП). В этом случае время пересылки порции информации в ОП может быть снижено с 25 до 1,2 мкс (цикл магистрали). Что же касается аппаратных и программных издержек системы прерываний, то они здесь фактически отсутствуют, так как передачу данных в ОП организует интерфейс ПДП помимо процессора, прерывать работу

процессора не требуется и замедление работы основной программы будет происходить лишь за счет того, что интерфейс ПДП будет при каждом поступлении информации занимать один цикл магистрали. Таким образом, ПДП является чрезвычайно эффективным средством приема измерительной информации, но, как будет показано ниже, он оказывается довольно сложным и дорогим устройством и, как и все аппаратные средства, не обладает достаточной гибкостью. Поскольку программа обслуживания ВУ выполняется на аппаратном уровне, программист не может произвольно менять ее содержание и интерфейсы ПДП оказываются, таким образом, узко специализированными на определенную измерительную аппаратуру и определенный способ накопления информации, хотя в интерфейсе ПДП может быть предусмотрено несколько режимов приема и накопления данных (за счет дальнейшего усложнения его конструкции).

Аппаратные средства организации ПДП (см. рис. 2.10) аналогичны средствам организации прерываний. В составе ОШ наряду с четырьмя парами линий запроса передачи *ЗПi* и разрешения передачи *РПi* ($i = 4...7$) имеется пара линий запроса прямого доступа ЗПД и разрешения прямого доступа РПД, к которым подключаются интерфейсы ПДП. Приоритет линии ЗПД выше приоритета линий *ЗПi*, поэтому арбитр предоставляет право интерфейсам ПДП занимать магистраль для передачи данных в первую очередь.

В микроЭВМ наряду с парой линий требования и подтверждения прерываний *ТПР* и *ППР* имеются еще линии требования прямого доступа *ТПД* и предоставления прямого доступа *ППД*. Кроме того, в обмене участвует специальный сигнал подтверждения выборки *ПВ*. На рис. 2.16 изображена последовательность сигналов канала микроЭВМ, участвующих в процедуре *ПДП* при выполнении операции передачи одного слова из ВУ, оснащенного контроллером *ПДП*, в ОП.

Центральный процессор, получив требование прямого доступа и дождавшись окончания текущего цикла магистрали (но не окончания всей выполняемой команды), устанавливает сигнал *ППД*. Интерфейс, приняв *ППД*, подтверждает его получение установкой сигнала на линии *ПВ*. Центральный процессор, получив сигнал *ПВ*, сбрасывает сигнал *ППД*, чем и заканчивается процедура предоставления прямого доступа (или, другими словами, предоставления устройству канала).

Интерфейс *ПДП*, захватив канал, инициирует операцию записи в ОП точно так же, как это обычно делает процессор, для чего в интерфейсе должна иметься соответствующая аппаратура. Эта часть обмена сигналами повторяет протокол операции записи (могла бы быть и операция чтения). Сигнал *ПВ*, "держащий" магистраль, остается в активном состоянии до окончания обмена, который в зависимости от характеристик ВУ может быть пословным или групповым. Если ВУ передает в ОП сразу группу слов, сигнал *ПВ* поддерживается активным до полного окончания передачи, что показано на рис. 2.16 пунктиром.

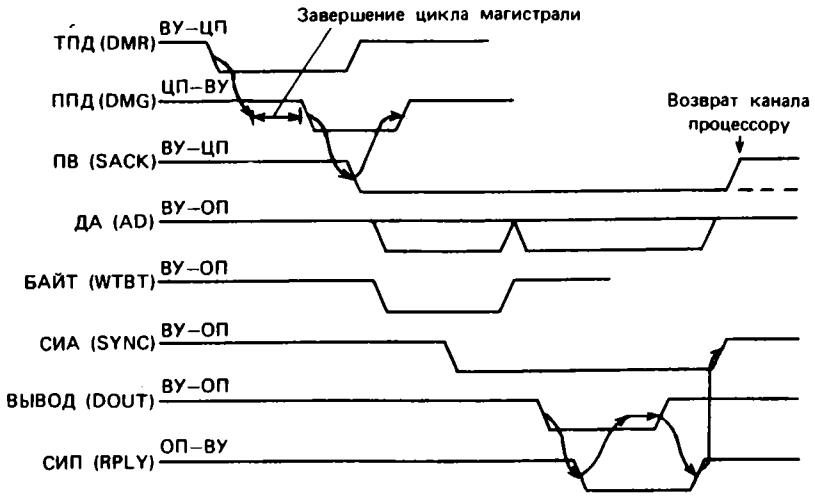


Рис. 2.16. Протокол магистрали в режиме прямого доступа в память

Следует подчеркнуть, что в протоколе ПДП отсутствует сигнал прерывания работы ЦП. Действительно, в процессе ПДП процессор не прерывает выполнение текущей программы и вообще не участвует в процедуре ПДП. Если в момент получения запроса на прямой доступ ЦП выполняет какие-то действия, не затрагивающие ОШ (например, команды с прямой адресацией), работа ЦП вообще не замедляется. Если же процессору в течение цикла ПДП потребуется ОШ (например, для считывания из ОП очередной команды), его работа приостанавливается до завершения очередного цикла ПДП.

Рассмотрим в качестве примера интерфейс ПДП, обслуживающий АЦП. Как уже отмечалось выше, конструкция интерфейса ПДП определяется как функциями измерительной аппаратуры, так и принятым способом накопления информации. Предположим, что коды от АЦП требуется заносить в память в порядке их поступления, без какого-либо анализа или сортировки. Упрощенная структурная схема интерфейса ПДП, выполняющего эту задачу, приведена на рис. 2.17. В состав интерфейса входит ряд регистров, оснащенных приемниками и передатчиками для связи с каналом, узел выбора адреса и синхронизации, а также узлы прямого доступа (ПД), прерываний и управления интерфейсом. В функции узла ПД входит реализация протокола операции ПДП, приведенного на рис. 2.16. Для этого узел ПД имеет доступ ко всем сигналам канала, участвующим в протоколе захвата магистрали и передачи данных в ОП: *ТПД, ППД, ПВ, БАЙТ, СИА, ВЫВОД* и *СИП*. Входным сигналом для узла ПД является сигнал запроса прямого доступа, генерируемый узлом управления интерфейсом; получив от ЦП разрешение на ПДП, узел ПД

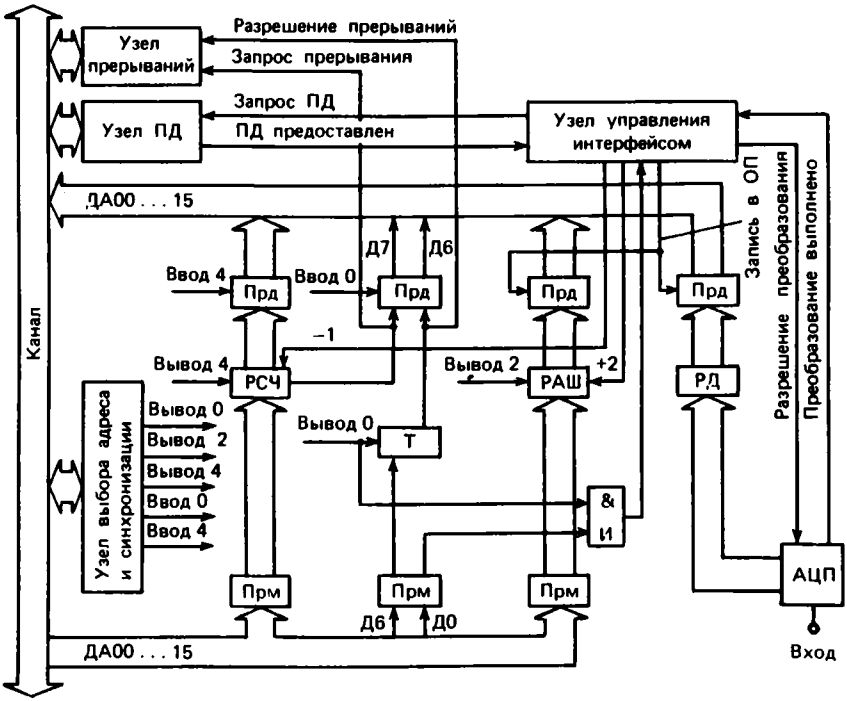


Рис. 2.17. Структурная схема интерфейса прямого доступа в память для обслуживания АЦП

информирует об этом узел управления интерфейсом сигналом *ПД предоставлен*, после чего оба эти узла вместе организуют передачу данных в ОП.

Узел прерываний позволяет интерфейсу работать не только в режиме ПДП, но и в режиме прерываний. Обычно интерфейс ПДП генерирует сигнал запроса прерывания при окончании передачи по прямому доступу всей запланированной информации. В рассматриваемом примере это происходит после приема с выхода АЦП и накопления в ОП заданного числа кодов входных событий.

В отличие от программно-управляемых интерфейсов в интерфейсе ПДП АЦП не связан с программно-доступными регистрами. Все управление АЦП (запуск, фиксация готовности, прием выходного кода) выполняется аппаратно узлом управления интерфейсом. Аппаратно же осуществляется счет числа зарегистрированных кодов и инкремент адреса ОП перед пересылкой очередного кода.

Инициализация интерфейса ПДП заключается в загрузке регистра адреса шины (РАШ) (адрес 2) начальным адресом буфера ОП, куда

предполагается пересылать регистрируемые коды, задании числа накапливаемых событий в регистре счетчике РСч (адрес 4) и разрешении прерываний установкой разряда 6 в РКС интерфейса, который расположен по базовому адресу 0 и содержит разряды: 0 – запуск интерфейса, 6 – разрешение прерываний и 7 – фиксация приема интерфейсом заданного числа кодов. Если обозначить базовый адрес интерфейса CSR, адрес 2 – ADDBUS и адрес 4 – CNTE, описанные операции потребуют следующих программных строк:

```
MEMBUF:  .BLKW 10000.              ; Буфер ОП для накопления кодов
START:   MOV #MEMBUF, ADDBUS       ; Начальный адрес ОП
         MOV #10000., CNTR         ; Требуемое число событий
         MOV #100, CSR             ; Разрешение прерываний
```

Запуск интерфейса, т. е. запуск процесса накопления информации, выполняется как обычно, установкой 1 в рязряде 0 РКС: INC CSR.

Все последующие действия по приему и регистрации поступающей информации выполняются интерфейсом аппаратно, без участия ЦП и независимо от текущей программы. Сюда входят:

генерация узлом управления интерфейсом сигнала разрешения преобразования в АЦП;

ожидание сигнала *Преобразование выполнено*, говорящего о том, что АЦП зарегистрировал очередное событие и его код находится в регистре данных РД;

по получении указанного сигнала передача в ЦП требования прямого доступа;

по получении разрешения на прямой доступ реализация собственно ПДП – генерация цикла записи в ОП содержимого РД по адресу, находящемуся в РАШ;

инкремент РАШ (при побайтовой записи в память – добавление в РАШ 1, при пословной 2);

декремент РСч;

генерация сигнала разрешения преобразования в АЦП и повторение всех описанных операций.

После накопления заданного числа кодов в РСч устанавливается 0, что является сигналом для установки разряда 7 РКС и генерации сигнала прерывания. Происходит прерывание текущей программы и переход на ПОП, в которой, скорее всего, выполняется какая-то обработка накопленных событий (например, перезапись на магнитную ленту с целью последующего анализа уже не в реальном времени эксперимента).

Нетрудно увидеть, что описанный интерфейс при всей его сложности страдает слишком узкой функциональной направленностью. Он не только рассчитан на работу с единственным типом измерительной аппаратуры (АЦП), но к тому же обеспечивает лишь один режим накопления информации. Однако сравнительно небольшие усовершенствования интерфейса позволят использовать его и в другом типичном режиме – анали-

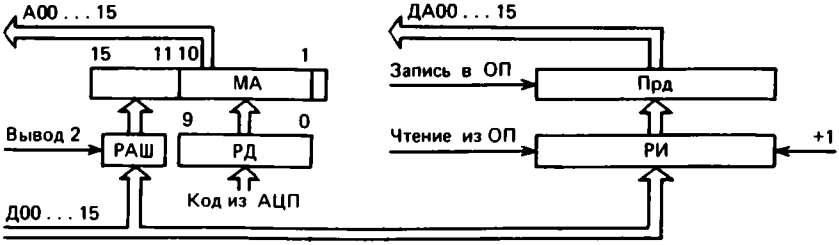


Рис. 2.18. Развитие структурной схемы интерфейса ПДП

заторном, для чего в интерфейс вводятся два дополнительных узла – регистр инкрементный (РИ) и мультиплексор адреса (МА) и изменяется схема подключения РАШ и РД (рис. 2.18).

В анализаторном (инкрементном) режиме код из АЦП не записывается в ОП, а определяет номер канала, в которм надо выполнить инкремент. Адрес канала в ОП формируется аппаратно мультиплексором адреса МА, в старшие разряды которого (с 11 по 15) из РАШ поступает фиксированное смещение, характеризующее начало области ОП, отводимой под спектр, а в младшие (с 1 по 10) – код из АЦП. Поскольку при передаче из РД в МА код сдвигается на 1 разряд влево, что равносильно его умножению на 2, каждый канал занимает 2 байта памяти, т. е. одно слово.

По адресу, сформированному МА, происходит чтение содержимого ОП в РИ, добавление к нему единицы и запись этого нового содержимого канала назад в ОП по тому же адресу. Такая схема формирования адреса не позволяет размещать спектр в любом месте ОП. Действительно, поскольку в программно-адресуемом РАШ под смещение отводятся разряды с 11 по 15, область под спектр должна начинаться на границе 1024 слов, т. е. по адресу 1К, 2К, 3К и т. д. При этом если бы интерфейс проектрировали под АЦП большей точности (с большим числом разрядов), то и начало спектра пришлось бы позиционировать на более крупные участки ОП (для 11-разрядного АЦП на 2К, для 12-разрядного на 4 К и т. д.).

При регистрации спектра полное число накапливаемых кодов обычно превышает объем одного слова памяти (65535). Поэтому в рассматриваемом интерфейсе целесообразно увеличить длину РСч до 32 разрядов.

В реальном интерфейсе ПДП могут быть предусмотрены дополнительные возможности: переменный объем канала (1 или 2 байта), программное завершение накопления (например, при истечении заданного времени), работа с АЦП различной разрядности и пр. Настройка интерфейса на требуемый режим работы осуществляется установкой соответствующих разрядов РКС.

Интерфейсы ПДП целесообразно использовать в тех случаях, когда при высоком темпе поступления измерительной информации требуется параллельно со сбором данных выполнять какую-либо вычислительную работу (например, обработку предыдущей серии измерений).

Глава 3

ИНТЕРФЕЙСЫ ОБЩЕГО НАЗНАЧЕНИЯ

3.1. ПАРАЛЛЕЛЬНЫЙ ИНТЕРФЕЙС ОБЩЕГО НАЗНАЧЕНИЯ

Для подключения к ЭВМ нестандартного электронного оборудования удобно использовать параллельный интерфейс общего назначения (другие его названия: устройство параллельного обмена, адаптер ввода-вывода, модуль двоичных входов-выходов). Интерфейс позволяет организовать двустороннюю передачу данных 16-разрядным параллельным кодом между ЭВМ и внешним электронным оборудованием и имеет средства программного управления ВУ и анализа его состояния. Передача данных может вестись как в режиме ожидания готовности, так и в режиме прерываний.

Упрощенная структурная схема интерфейса приведена на рис. 3.1. Узел выбора адреса и синхронизации декодирует адрес, поступающий по магистрали, и вырабатывает сигналы ввода и вывода, управляющие работой узлов интерфейса. Двухсекционный узел прерываний позволяет обслуживать два источника прерываний в установке: *Требование А* и *Требование В*, конкретное назначение которых определяется структурой электронного оборудования, подключаемого к ЭВМ.

Для передачи данных из ЭВМ в ВУ в интерфейсе имеется 16-разрядный выходной регистр данных, доступный как для записи, так и для чтения по адресу 2. Данные, записанные в выходной регистр, появляются на 16 выходных линиях данных и могут быть приняты внешним электронным оборудованием. Входные данные из внешнего оборудования поступают по 16 входным линиям данных на передатчики, открываемые сигналом *Ввод 4*. Предполагается, что буферный регистр входных данных входит в состав внешнего оборудования. Как видно из рис. 3.1, интерфейс имеет дополнительные цепи синхронизации обмена. Сигнал *Вывод 2*, который осуществляет передачу новых данных с магистрали в выходной регистр, поступает на линию *Вывод данных* и может служить для внешнего оборудования сигналом готовности новых данных. Сигнал *Ввод 4*, осуществляющий прием входных данных на магистраль, поступает на линию *Ввод данных* и может служить для внешнего оборудования сигналом получения ЭВМ очередных данных. Оба этих сигнала могут использоваться во внешнем электронном оборудовании для синхронизации его работы, однако надо иметь в виду, что сигналы имеют импульсный характер и не могут служить признаками состояния.

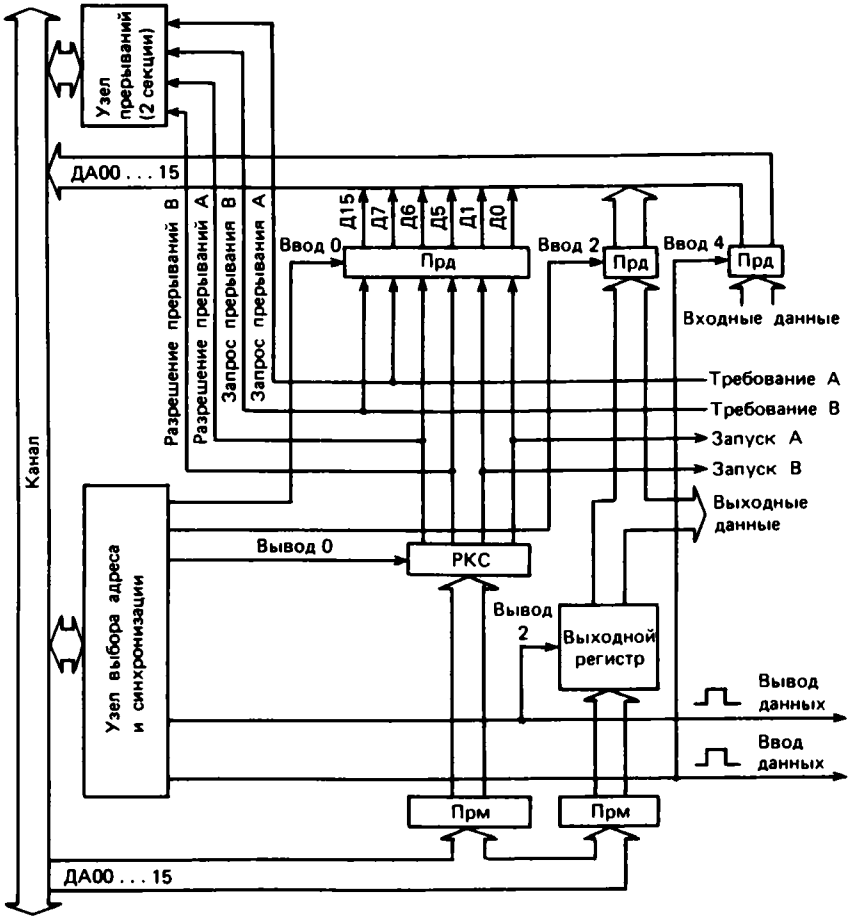


Рис. 3.1. Структурная схема параллельного интерфейса общего назначения

Регистр команд и состояний с адресом 0 содержит четыре триггера (разряды 0, 1, 5 и 6), программно доступных как для записи, так и для чтения. Разряды 0 и 1 используются для запуска внешнего оборудования (разблокировка входа, разрешение преобразования и т. д.). Разряды 5 и 6 с внешним оборудованием не связаны, они используются для программного разрешения и запрещения прерываний. Запросы на прерывания (*Требование А* и *Требование В*) поступают из внешнего оборудования на передатчики, связанные с разрядами 7 и 15, при этом прерывание по разряду 7 РКС разрешается и запрещается, как и обычно, разрядом 6, а прерывание по разряду 15 контролируется разрядом 5. Как и

в описанных ранее интерфейсах, триггеры, хранящие состояние требования прерывания, должны входить в состав внешнего оборудования.

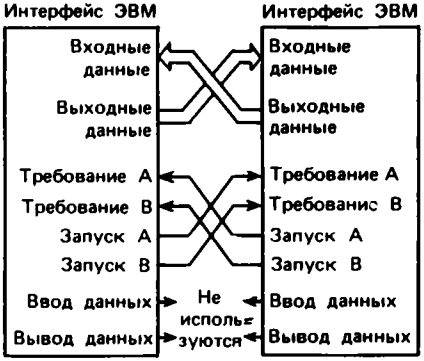
Использование триггерных *(Запуск А, Запуск В, Требование А, Требование В)* и импульсных *(Ввод данных, Вывод данных)* сигналов имеет определенную специфику. Так, сигналы *Запуск А (В)*, устанавливаемые программно командами BIS # 1, CSR (BIS # 2, CSR) и сбрасываемые командами BIS # 1, CSR (BIC # 2, CSR) соответственно, могут служить для программного задания длительности функционирования ВУ, а также для передачи в ВУ двухразрядного двоичного кода, характеризующего требуемую операцию или требуемые характеристики ВУ (коэффициент усиления, разрядность преобразования и т. п.).

Если внешнее оборудование должно и принимать, и передавать данные (представляя, например, комбинацию АЦП и ЦАП), сигналы *Требование А* и *Требование В* служат, как обычно, идентификаторами готовности ВУ к обмену данными с ЭВМ. При этом ЭВМ может обмениваться данными с ВУ по каждому каналу как в режиме ожидания готовности, так и в режиме прерываний. В тех случаях, когда нет необходимости в двух источниках прерываний от ВУ, связанных с передачей данных, разряд 15 РКС можно использовать для фиксации сбоев в работе внешнего оборудования, как это обычно предусматривается в стандартных периферийных устройствах ЭВМ. Для сброса триггеров готовности (входящих в состав ВУ) естественно воспользоваться импульсными сигналами *Ввод данных* и *Вывод данных*, возникающими в моменты приема и передачи данных. С помощью этих сигналов можно также при необходимости организовать импульсный запуск аппаратуры. Если подключенное через интерфейс ВУ только передает данные в ЭВМ (АЦП), то выходной регистр интерфейса не используется. В этом случае команда CLR DBROUT "фиктивной" загрузки нулем выходного регистра приведет к генерации импульсного сигнала *Вывод данных*, которым можно осуществить запуск АЦП. Для генерации сигнала *Ввод данных* нужно выполнить команду, инициирующую операцию чтения данных по адресу 4 интерфейса, например TST DBRIN.

Параллельные интерфейсы, аналогичные описанному, разработаны для многих машин: "Электроника 60" и ДВК (модуль И2), MERA-60 (модуль MWW-60) и др.

Параллельный интерфейс общего назначения можно использовать для объединения двух ЭВМ с целью пересылки данных из одной ЭВМ в другую. В этом случае каждая ЭВМ оснащается модулем интерфейса, причем входы и выходы каждого модуля соединяются друг с другом так, как это показано на рис. 3.2. Для передачи данных из одной ЭВМ в другую на передающей ЭВМ должна функционировать программа передачи данных, а на принимающей ЭВМ – программа приема. Не вдаваясь в подробности организации верхних уровней этих программ (задание объема передаваемых данных и их адресов, контроль правильности

Рис. 3.2. Соединение двух ЭВМ через параллельный интерфейс общего назначения



передачи и повторение передачи в случае одиночного сбоя и т. д.), рассмотрим лишь особенности собственно передачи слова данных (16 двоичных разрядов) через параллельный интерфейс общего назначения.

Основная трудность заключается в необходимости программной организации квитирования (обычно квитирование осуществляется аппаратно). Для программного квитирования можно использовать по одному сигналу требования с каждой стороны, программно анализируя как установку, так и сброс этого сигнала. При использовании сигналов *Запуск А* и *Требование А* программы пересылки одного слова будут иметь следующий вид:

```
; Программа на передающей ЭВМ
        MOV   DATA, DBROUT ; Подготовка данных для пересылки
        INC   CSR          ; Установка флажка "Данные готовы"
1$ :    TSTB  CSR          ; Ожидание приема данных
        BPL   1$           ; другой ЭВМ
        CLR   CSR          ; Сброс флажка "Данные готовы"
2$ :    TSTB  CSR          ; Ожидание сброса флажка "Данные
        BMI   2$           ; приняты" из принимающей ЭВМ
; Программа на принимающей ЭВМ
1$ :    TSTB  CSR          ; Ожидание флажка "Данные готовы"
        BPL   1$           ; из передающей ЭВМ
        MOV   DBRIN, DATA  ; Прием данных
        INC   CSR          ; Установка флажка "Данные приняты"
2$ :    TSTB  CSR          ; Ожидание сброса флажка "Данные
        BMI   2$           ; готовы" из передающей ЭВМ
        CLR   CSR          ; Сброс флажка "Данные приняты"
```

Таким образом, пересылка каждого слова данных должна сопровождаться шестью строками программного квитирования на каждой ЭВМ.

3.2. ПАРАЛЛЕЛЬНЫЙ ИНТЕРФЕЙС ИРПР

Для подключения к ЭВМ стандартного периферийного оборудования (алфавитно-цифровых терминалов, перфоленточных устройств ввода-вывода и др.) используется радиальный параллельный интерфейс (ИРПР). В зависимости от типа подключаемого оборудования конкретная реали-

Здесь символические обозначения CSROUT и DBROUT присвоены адресам РКС вывода и РД вывода соответственно, а DATA – условное обозначение ячейки памяти, в которой хранится передаваемая информация.

Заполнение РД вывода байтом данных приводит к установке триггера *Т*, который блокирует прохождение сигнала *ЗП* в разряд готовности. Таким образом, заполнение РД приводит к сбросу бита готовности. Одновременно устанавливается строб-сигнал *СТР*, который в приемной части интерфейса устанавливает разряд 7 РКС ввода. Сигнал *СТР* и передаваемые данные поддерживаются на входах приемника до тех пор, пока байт данных не будет прочитан в ЭВМ. На время действия сигнала *Ввод 2* прекращается действие сигнала ЗП. Изменение состояния этого сигнала приводит в передающей части интерфейса к сбросу триггера *Т*. В результате, во-первых, сбрасывается сигнал *СТР* и соответственно разряд 7 РКС ввода, а во-вторых, разблокируется прохождение сигнала *ЗП* в разряд 7 РКС вывода, чем и восстанавливается исходное состояние интерфейса (очередной байт передан; интерфейс готов передавать следующий байт). Из сказанного ясно, что программирование ввода осуществляется так же просто, как и программирование вывода:

```
1$ :  TSTB   CSRIN        ; Ожидание поступления
      BPL    1$           ; данных на вход
      MOVB   DBRIN, DATA  ; Прием данных в ОП
```

Символические обозначения CSRIN и DBRIN присвоены адресам РКС ввода и РД ввода соответственно. Следует заметить, что в разрядах 7 РКС ввода и вывода, как и в разрядах *Д0... 7* РД ввода отсутствуют триггеры; состояния разрядов в любой момент времени определяются уровнями соответствующих сигналов *(ЗП, СТР* и *Д0... 7)*.

Разряды 7 РКС ввода и вывода подключены к системам прерываний, что дает возможность работать в режиме прерываний как от передатчика, так и от приемника. Для инициализации режима следует установить разряды 6 РКС, разрешающие прерывания.

Рассмотрим пример использования ИРПР для построения двухмашинного комплекса, в котором роль центральной ЭВМ выполняет СМ-4, а в качестве периферийной выступает "Электроника 60" или ДВК. Следует заметить, что в системе РАФОС-II имеется программное обеспечение локальной многомашинной системы, состоящей из одной центральной и нескольких периферийных ЭВМ. Приводимый ниже пример иллюстрирует принципы построения такого рода программ.

Структура комплекса изображена на рис. 3.6. Машины связаны через устройства параллельного обмена, работающие в соответствии с протоколом ИРПР. Каждое устройство включает в себя передающую и приемные части (источник *И* и приемник *П*). Передающая часть каждого устройства связана с приемной частью устройства на другой ЭВМ линиями:

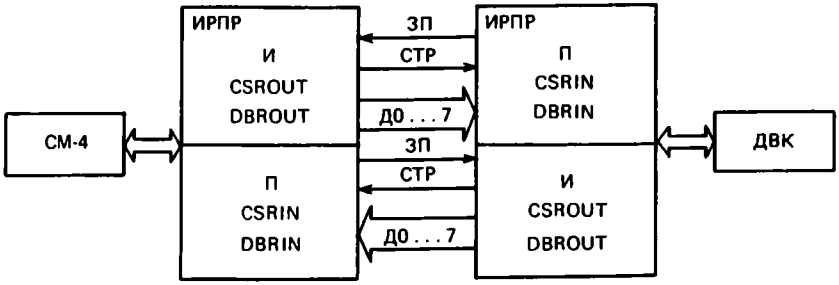


Рис. 3.6. Схема двухмашинного комплекса со связью через ИРПР

запрос приемника *ЗП*, строб источника *СТР*, данные *Д*. Всего, таким образом, в кабель связи входит 20 линий.

Разработка и использование программного обеспечения упрощается, если на центральной ЭВМ установлена многопользовательная ОС (TS – монитор РАФОС или система NTS). В этом случае физическая линия связи с периферийной ЭВМ объявляется (при генерации системы) одним из терминалов комплекса, что как бы включает ее в сферу действия ОС и дает возможность использовать при организации межмашинного обмена стандартные алгоритмы ОС. Данные, поступающие в центральную ЭВМ по линии связи, рассматриваются ОС как сообщения, набираемые оператором на клавиатуре соответствующего терминала; данные, предназначенные для вывода на экран этого терминала, передаются по линии связи на периферийную ЭВМ. Если на периферийной ЭВМ выполняется программа, которая передает символы, поступающие с линии связи, на экран, а символы, набираемые на клавиатуре, – в линию связи, то система обеспечивает *эмуляцию терминала* центральной ЭВМ на терминале периферийной. Оператор периферийной ЭВМ как бы работает на центральной ЭВМ, имея доступ ко всем ее ресурсам (но не ресурсам своей ЭВМ!).

После загрузки системы NTS первым действием оператора, работающего за каким-либо терминалом, является нажатие клавиши *Возврат каретки* (⟨ВК⟩). Операционная система, получив этот символ, пересылает на экран терминала текст, включающий, в частности, требование на ввод шифра пользователя. Если оператор вводит удовлетворяющий систему шифр, ОС открывает доступ к файлам пользователя и приступает к выполнению команд оператора, вводимых с клавиатуры терминала.

Программа эмуляции терминала (выполняемая на периферийной ЭВМ) в самом простейшем виде выглядит следующим образом:

```
1$ :  TSTB   CSROUT        ; Ожидание готовности
      BPL    1$            ; линии к передаче
      MOVB   #15, DBROUT   ; Пересылка ⟨ВК⟩
      CLR    177560        ; Запрет прерываний от клавиатуры
```

```
2$ :  TSTB   CSRIN          ; Получен символ из СМ-4?
      BPL    4$             ; Нет еще, на проверку клавиатуры
      MOVB   DBRIN, R1      ; Да, в R1 его
3$ :  TSTB   177564         ; Ожидание
      BPL    3$             ; готовности экрана
      MOVB   R1, 177566     ; Вывод полученного символа на экран
4$ :  TSTB   177560         ; Какая-либо клавиша нажата?
      BPL    2$             ; Нет еще, на проверку линии
      MOVB   177562, R2     ; Да, в R2 этот символ
      CMPB   R2, #3         ; Не нажали ли <CTRL/C>?
      BEQ    6$             ; Да, на завершение программы
5$ :  TSTB   CSROUT         ; Ожидание готовности
      BPL    5$             ; линии к передаче
      MOVB   R2, DBROUT     ; Вывод набранного символа в линию
      BR     2$             ; Бесконечный цикл
6$ :  BIS    #100, 177560   ; Разрешение прерываний от клавиатуры
      .EXIT                 ; Завершение программы
```

Программа начинается с пересылки в СМ-4 кода <ВК> (предполагается, что к моменту запуска на ДВК программы эмуляции ЭВМ СМ-4 включена и система NTS загружена). Затем запрещаются перрывания от клавиатуры терминала, чтобы нажатия клавиш не "перехватывались" программой обработки прерываний от клавиатуры, входящей в состав ОС, функционирующей на ДВК. Далее организуется цикл последовательного опроса приемника ИРПР и клавиатуры терминала, обозначенный в программе стрелками. Как только в приемник ИРПР приходит символ, он отправляется на экран терминала; как только появляется символ в РД клавиатуры, он пересылается в РД передатчика для отправки в СМ-4. Проверка кода нажатой клавиши на управляющий символ <CTRL/C>, имеющий код 3, позволяет в любой момент завершить программу эмуляции одновременным нажатием клавиш <CTRL> и <C>.

Приведенная программа позволяет оператору, работающему за терминалом ДВК, запускать на СМ-4 любые системные или прикладные программы. При этом программы могут обмениваться информацией со всеми ВУ СМ-4, если же в программе встречается строка вывода сообщения на терминал, то это сообщение появляется на экране терминала ДВК. Серьезным ограничением этого режима является недоступность внешних устройств ДВК и как следствие этого невозможность пересылки файлов (с текстами программ или экспериментальными данными) с СМ-4 на ДВК или в обратном направлении.

Для организации пересылки файлов на СМ-4 должна быть запущена специальная программа поддержки этого режима (что, между прочим, можно выполнить с терминала ДВК); программа эмуляции, выполняемая на ДВК, также должна выйти из цикла приема-передачи символов, и приступить к действиям, поддерживающим пересылку файлов. Алгоритм ее работы в случае передачи файла с ДВК на СМ-4 может быть следующим:

ввод с клавиатуры терминала команды оператора, описывающий входной и выходной файлы, а также направление передачи;

анализ этой строки на правильность и пересылка имени выходного файла в программу, выполняемую на СМ-4;

открытие канала связи со "своим" файлом;

чтение этого файла в буфер ОП;

пересылка содержимого буфера по линии связи.

Алгоритм программы, выполняемой на СМ-4 следующий:

прием по линии связи имени выходного файла;

открытие канала связи с этим файлом и резервирование на НМД места для него;

ожидание поступления по линии связи массива с содержимым файла;

прием по линии связи содержимого файла в буфер ОП;

запись содержимого буфера в файл на НМД.

Наличие на центральной и периферийной ЭВМ указанных программ позволяет выполнять целый ряд операций: накапливать на периферийных ЭВМ данные и пересылать их затем в центральную ЭВМ для обработки, выполнять на центральной ЭВМ разработку программ реального времени и пересылать их загрузочные модули (после трансляции и компоновки) на периферийную ЭВМ для выполнения, получать доступ из каждой ЭВМ к периферийному оборудованию другой ЭВМ. Последнее может оказаться полезным в тех случаях, когда периферийная (или, наоборот, центральная) ЭВМ оснащена каким-либо уникальным для данной ВС оборудованием: графическими средствами, накопителями большой емкости и т. д.

3.3. ПОСЛЕДОВАТЕЛЬНЫЙ ИНТЕРФЕЙС

Все рассмотренные выше интерфейсы отличались тем, что передача данных между интерфейсом и измерительным оборудованием осуществлялась по 8- или 16-разрядной шине данных сразу целым байтом или словом. Такая параллельная передача данных обеспечивает высокие скорости обмена информацией, но требует многопроводных шин связи и относительно громоздкого оборудования. Поэтому интерфейсы с параллельной передачей данных, или параллельные интерфейсы, используются лишь в тех случаях, когда программно-управляемое электронное оборудование можно расположить недалеко от ЭВМ (не далее нескольких десятков метров). В тех же случаях, когда источник или приемник информации удален от ЭВМ на значительное расстояние, применяется последовательная передача, при которой данные проходят по единственной линии связи последовательно бит за битом. Это позволяет существенно снизить стоимость линий связи, но, с другой стороны, заметно (по меньшей мере на порядок) уменьшает скорость передачи данных. Все же экономические соображения в сочетании с простотой реализации часто берут верх и последовательная передача измерительной информа-

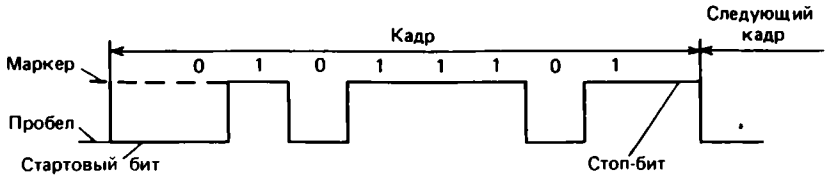


Рис. 3.7. Структура кадра при последовательной передаче данных

ции широко используется в тех случаях, когда исходя из безопасности работы персонала вычислительный комплекс нельзя расположить рядом с измерительным оборудованием либо когда датчики измерительной информации, связанные с единой ЭВМ, разбросаны на большой площади.

Различают два способа последовательной передачи данных: *синхронный* и *асинхронный*. В первом случае данные передаются сплошным потоком, без разделения на слова или байты. Для идентификации отдельных битов источник и приемник должны управляться синхронно работающими генераторами. Во втором случае начало и конец каждой порции информации (*кадра*) отмечаются специальными метками; требования к синхронизации передатчика и приемника заметно снижаются, хотя падает и скорость передачи (за счет меток начала и конца кадра). В измерительной технике чаще используется асинхронная передача.

Стандартный формат последовательной асинхронной передачи данных изображен на рис. 3.7. Уровень логической 1 в линии называется *маркером*, уровень логического нуля – *пробелом*. При отсутствии данных в линии действует сигнал маркера. Передача порции данных начинается с посылки *стартового бита* пробела. После этого передаются биты данных, число которых в кадре может устанавливаться от 5 до 8. За битами данных может следовать бит *паритета*, который также называется битом контроля четности (или нечетности). Этот бит выбирается в каждой порции данных таким образом, чтобы общее число единиц в битах данных и бите паритета было четным (или нечетным). Кадр заканчивается *стоп-битом*, имеющим уровень маркера. После этого в линии может поддерживаться состояние отсутствия данных (уровень маркера) либо начинаться следующий кадр (стартовым битом пробела).

Сигналы в линии могут иметь различное представление. При передаче на небольшие расстояния в линии действуют уровни напряжения 3...5 В. При бо́льших расстояниях (до 1,5 км) и для управления электромеханическим оборудованием используют *токовую петлю* – импульсы постоянного тока значением 20 или 40 мА, передаваемые по двухпроводной линии (*витой* или *скрученной паре*) либо по кабелю. В случае *дуплексной связи*, т. е. передачи информации как в прямом, так и в обратном направлении, используют четырехпроводную линию. Асинхронная связь постоянным током (токовая петля) по четырехпроводной дуплексной ли-

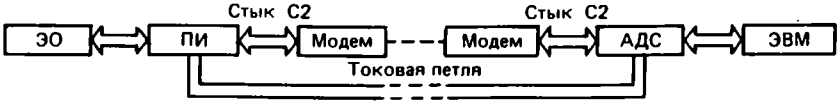


Рис. 3.8. Структура линии связи при последовательной передаче данных

нии носит название радиального последовательного интерфейса (ИРПС). Наконец, при передаче информации по телефонным линиям уровни напряжения преобразуют в посылки (пачки) синусоидальных сигналов. Сигналу маркера соответствует частота (тон) 1270 Гц, сигналу пробела – 1070 Гц. Для дуплексной связи по одной и той же телефонной линии используют две пары частот, например 1270...1070 Гц и 2225...2025 Гц. Частота 2225 Гц служит для передачи в обратном направлении сигнала маркера, частота 2025 Гц – сигнал пробела.

Структура линии связи электронного оборудования с удаленной ЭВМ изображена на рис. 3.8. Электронное оборудование (ЭО) должно быть снабжено последовательным интерфейсом (ПИ), преобразующим параллельный код, поступающий из ЭО, в последовательный в соответствии с описанным выше стандартным форматом. Преобразование сигналов напряжения, действующих на выходе ПИ, в сигналы телефонного тона осуществляется специальным устройством – *модемом.* Для управления модемом (подключения модема к линии, включения тона, передачи данных, контроля состояния модема) предусматриваются соответствующие стандартные сигналы. Совокупность линий для передачи этих сигналов образует стык С2, широко используемый в вычислительных системах.

Подключение телефонной линии к ЭВМ осуществляется через второй модем, преобразующий телефонные посылки в уровни напряжения стыка С2, и последовательный интерфейс, называемый часто *адаптером дистанционной связи* (АДС) или *устройством последовательного обмена* (УПО), который является связующим звеном между стыком С2 и системным интерфейсом ЭВМ. Следует отметить, что из трех изображенных на рис. 3.8 элементов линии связи – АДС, модема и ПИ – два первых относятся к числу стандартных устройств, включаемых в состав вычислительных систем (АДС) либо выпускаемых промышленностью как самостоятельные изделия (модемы). Что же касается последовательного интерфейса, то, поскольку его конструкция в какой-то степени определяется назначением и характеристиками подключаемого к нему электронного оборудования, его надо разрабатывать для каждого конкретного применения. Разработка ПИ облегчается тем, что основные функциональные узлы интерфейса выпускаются в виде интегральных микросхем.

Описанная структура связи (с преобразованием в частотный телефонный сигнал) применяется только в тех случаях, когда расстояние между ЭО и ЭВМ весьма велико – более 1,5...2 км. При меньших расстояниях

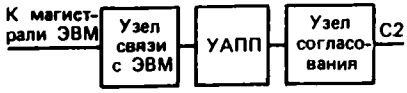


Рис. 3.9. Структура АДС

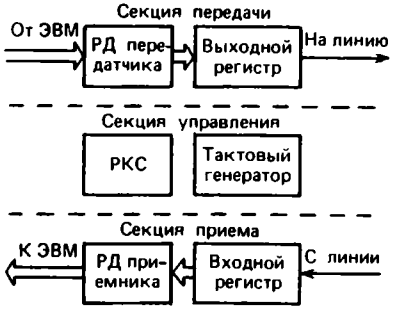


Рис. 3.10. Структура УАПП

необходимость в использовании модемов отпадает и связь ПИ и АДС осуществляется по двухпроводной линии с помощью токовой петли. Стандартные АДС часто имеют два выхода: на стык С2 и токовую петлю.

Рассмотрим структуру адаптера дистанционной связи (рис. 3.9). Он включает в себя узлы связи с системным интерфейсом ЭВМ, универсальный асинхронный приемопередатчик (УАПП) и узлы согласования, осуществляющие, в частности, преобразование ТТЛ-уровней в уровни стыка С2. Узлы связи с системным интерфейсом (дешифратор адреса, модуль управления прерываниями и пр.) практически не отличаются от рассмотренных в предыдущих параграфах. Эти узлы обеспечивают протокол системной магистрали при передаче данных между ЦП и интерфейсом.

Наиболее специфической частью АПД является УАПП, упрощенная структурная схема которого приведена на рис. 3.10.

Он состоит из трех секций: передачи, управления и приема. Секция передачи служит для преобразования данных из параллельной формы в последовательную. Байт данных поступает из ЭВМ (по команде программы) в параллельной форме в регистр данных РД передатчика. После завершения передачи в линию предыдущего байта и освобождения выходного регистра байт данных переносится (также параллельно) в выходной сдвиговый регистр. Здесь к нему добавляются служебные биты: стартовый, стоповый и паритета. Полученное таким образом содержимое кадра многократно сдвигается в сторону младших битов, в результате чего на выходе концевого триггера регистра, связанного с передающей линией, последовательно появляются значения всех битов кадра. Пока байты данных передаются в линию, в РД передатчика может загружаться из ЭВМ следующая порция информации.

Секция приема работает аналогично. Биты, поступающие из линии, вдвигаются во входной сдвиговый регистр. После получения всего кадра из него убираются служебные биты и оставшаяся информационная часть переносится параллельно в РД приемника, откуда по команде программы данные принимаются в ЭВМ. Пока происходит пересылка данных в ЭВМ, входной сдвиговый регистр может принимать следующую порцию (кадр) данных.

В секции управления имеются регистры команд и состояний (обычно два), с помощью которых программно устанавливаются характеристики УАПП: скорость передачи, число информационных битов, наличие и вид паритета и т. д. Кроме того, отдельные разряды РКС (или иногда дополнительные разряды РД) фиксируют ошибки приема данных, например получения в РД приемника следующего байта данных до считывания в ЭВМ предыдущего (*ошибка наложения* или *переполнения*).

Тактовый генератор, входящий в состав УАПП, определяет частоту сдвига в сдвиговых регистрах и тем самым скорость передачи и приема данных. Очевидно, что передача и прием в конкретной линии должны вестись с одной скоростью.

Если АДС работает на токовую петлю, функции узла согласования ограничиваются преобразованием уровней напряжения, действующих на выходе УАПП, в токовые посылки. Если же АДС предназначен для подсоединения к модему либо другой аппаратуре с выходом на стык С2, узел согласования должен вырабатывать и воспринимать ряд управляющих сигналов. Число этих сигналов определяется функциональными возможностями АДС. В простейшем случае из нескольких десятков сигналов, предусмотренных стандартом на стык С2, используются следующие:

цепь 108: подключить АПД к линии (сигнал вырабатывается по команде ЭВМ, говорит о готовности ЭВМ передавать данные и требует подключения модема к линии);

цепь 107: АПД готова (сигнал вырабатывается модемом в ответ на сигнал 108 и говорит о том, что модем подключен к линии связи и готов взаимодействовать с ЭВМ);

цепь 105: запрос передачи (модем, получив от ЭВМ этот сигнал, включает сигнал маркера и информирует тем самым удаленный модем о начале сеанса связи);

цепь 106: готов к передаче (сигнал вырабатывается модемом в ответ на сигнал 105 и говорит о готовности модема принимать данные);

цепь 103: передаваемые данные;

цепь 104: принимаемые данные;

цепь 109: детектор принимаемого сигнала (этим сигналом приемный модем сообщает АДС об обнаружении на линии несущей частоты — сигнала маркера).

Адаптер дистанционной связи с модемом или без него (в зависимости от длины линии связи) может использоваться для подключения к ЭВМ терминального оборудования, а также связи двух ЭВМ с целью организации двухпроцессорной вычислительной системы. При этом, если обе ЭВМ имеют выход на стык С2, их можно связать через *нуль-модем*, представляющий собой два разъема стыков С2 с перемычками, соединяющими ответные цепи: цепь 103 с цепью 104, цепь 105 с цепью 106 и т. д. Естественно, на обеих ЭВМ должны быть активизированы программы приема-передачи данных. Часто электронное оборудование, пред-

назначенное для использования в автоматизированных системах, имеет выход на стык С2. Это дает возможность подключения такого оборудования к ЭВМ через стандартный (а не специализированный) последовательный интерфейс – АДС.

Рассмотрим передачу данных через АДС с модемом по некоммутируемой телефонной связи. Адаптер дистанционной связи содержит четыре программно-адресуемых регистра: РД и РКС передатчика и приемника. Форматы регистров приведены на рис. 3.11. Данные, подлежащие передаче, помещаются в младший байт РД передатчика командой пересылки байта. Готовность передатчика принять очередной байт данных из ЭВМ можно определить как опросом разряда 7 РКС передатчика, так и по прерыванию от передатчика (вектор прерываний передатчика располагается по адресу 314). Для работы в режиме прерываний необходимо разрешить прерывания установкой разряда 6 РКС передатчика. Разряд 0 РКС передатчика позволяет программно включить или выключить состояние *сплошного пробела* (отсутствие информации). Этот сигнал может использоваться, например, для передачи сообщения об окончании работы программы.

При установленном разряде 2 РКС передатчика устройство переводится в контрольный режим, когда выход передатчика отключается от линии и подключается ко входу приемника.

Данные, принимаемые с линии, поступают в младший байт РД приемника. При этом, если в процессе приема данных возникает ошибка, устанавливается разряд 15 и один из разрядов 12...14, в которых фиксируется тип ошибки. Наличие в РД приемника очередного принятого байта данных можно определить как опросом разряда 7 РКС приемника, так и по прерыванию от приемника (вектор прерываний расположен по адресу 310), если установлен разряд 6 РКС приемника. При этом надо иметь в виду, что приемник может создавать сигнал прерывания не только при приеме очередного байта (в этом случае устанавливается разряд 7 РКС приемника), но и при изменении состояния сигналов на стыке С2, в частности при установке сигнала 125 (вызов). Состояния цепей 109, 106 и 125 отражаются в разрядах 12...14, что дает возможность, получив сигнал прерывания, определить его причину. Прерывания по изменению состояния АПД разрешаются установкой разряда 5 и сопровождаются установкой разряда 15 РКС приемника, сбрасывается разряд 15 любым обращением к РД приемнику.

Разряды 1 и 2 РКС приемника управляют сигналами 108 и 105 стыка С2. Разряд 11 устанавливается в середине стартового бита принимаемой последовательности и сбрасывается по окончании кадра. Разряды 3 и 10 используются при работе с некоторыми модемами для повышения надежности связи.

Простейшая программа приема информации через модем и АДС выглядит следующим образом:

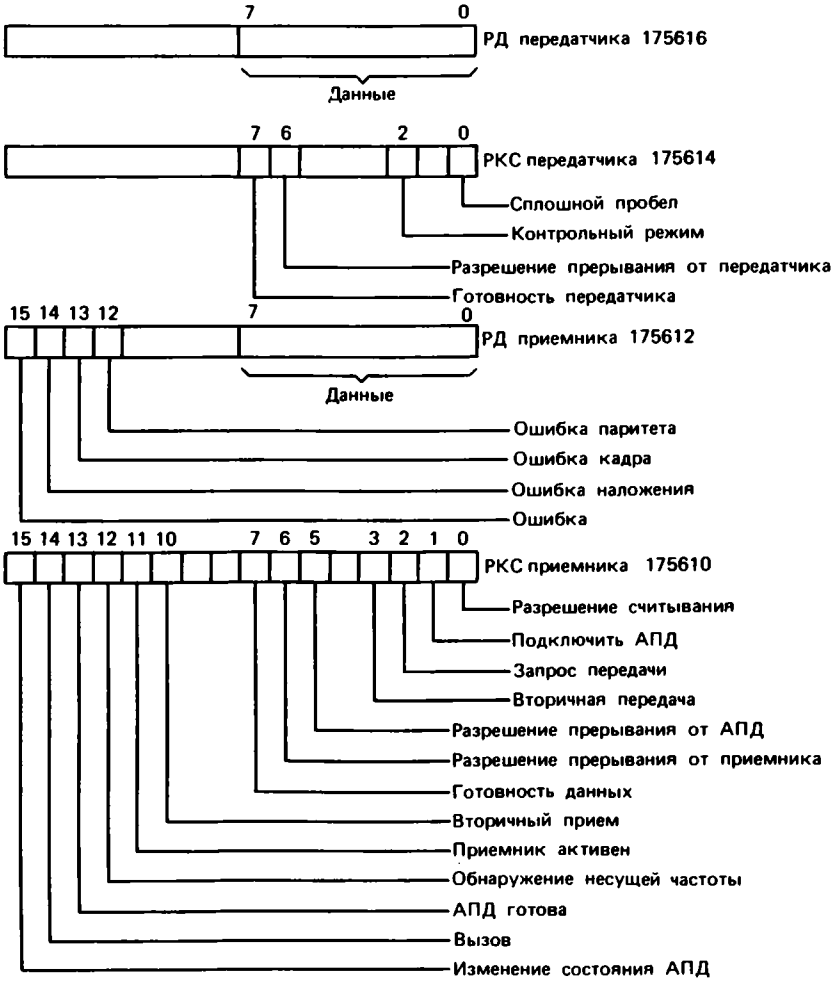


Рис. 3.11. Регистры АДС

```
CSRRCV = 175610                    ; РКС приемника
DBRRCV = 175612                    ; РД приемника
CSRSND = 175614                    ; РКС приемника
DBRSND = 175616                    ; РД приемника
START:  MOV   #DATA, R1            ; Адрес буфера для данных
        MOV   #6, CSRRCV           ; Подключить модем к линии
1 $:    BIT   #20000, CSRRCV       ; Ожидание готовности
        BEQ   1 $                  ; модема к передаче
2 $:    TSTB  CSRSND               ; Ожидание готовности
        BPL   2 $                  ; передатчика
        MOVB  #1, DBRSND           ; Пересылка кода 1 (пуск) в ЭО
3 $:    BIT   #10000, CSRRCV       ; Ожидание несущей
        BEQ   3 $                  ; частоты
        TST   DBRRCV               ; Сброс разряда 15 РКС приемника
4 $:    BIT   #100200, CSRRCV      ; Ожидание готовности данных
        BEQ   4 $                  ; или прекращения приема
        BMI   FIN                  ; Несущая снята, данных больше нет
        MOVB  DBRRCV, (R1)+        ; Байт данных в ОП
        BR    4                    ; За новым байтом
FIN: ; Строки обработки полученных данных
```

Если последовательный интерфейс предназначен для связи ЭВМ с удаленной на не слишком большое расстояние измерительной или управляющей аппаратурой, необходимость в модеме отпадает и для передачи данных используется токовая петля. В этом случае программирование упрощается. Рассмотрим программное управление работой некоторого исполняющего оборудования. Поскольку команды оборудованию можно передавать только в виде байтов данных, нужно предусмотреть в приемном последовательном интерфейсе дешифрацию принимаемых данных и формирование в зависимости от поступившего кода тех или иных управляющих сигналов. Коды управляющих сигналов определяют формат регистра данных передатчика. Пусть, например, разряд 0 РД передатчика задает пуск двигателя установки, разряды 1...4 – скорость движения (возможны 15 значений скорости, которым соответствуют двоичные коды от 0001 до 1111), разряды 5 и 6 – направление движения (11 – вперед, 10 – назад). Останавливается двигатель автоматически. Фрагмент программы управления, задающий движение вперед с максимальной скоростью, выглядит следующим образом (РКС передатчика обозначен CSR, РД передатчика – DBR):

```
1 $:    TSTB  CSR                  ; Ожидание готовности
        BPL   1 $                  ; передатчика
        MOVB  #176, DBR            ; Задание скорости и направления
2 $:    TSTB  CSR                  ; Ожидание готовности
        BPL   2 $                  ; передатчика
        BIS   #1, DBR              ; Пуск установки
```

Как уже отмечалось, для последовательного интерфейса характерна невысокая скорость передачи данных. Даже при отсутствии модема

и частотных ограничений телефонной линии скорость последовательной передачи редко превышает 9600 –19200 бит/с, что соответствует 960 – 1920 байт/с (1–05 мс на передачу 1 байта). За это время аппаратура, в которую поступают данные, вполне успевает принять очередной байт, и передачу данных можно вести без подтверждения с приемной стороны (см. предыдущий пример). Темп передачи определяется здесь лишь скоростью (невысокой) выдачи передатчиком данных в линию. Обратная связь приемника с передатчиком отсутствует (ср. рис. 3.5).

В настоящее время все большее распространение получают методы скоростной последовательной передачи по радиочастотным кабелям и волоконно-оптическим линиям связи (ВОЛС). Так, цифровая система передачи данных "Электроника МС-4101" реализует передачу со скоростью до 8 Мбит/с. В этих условиях вести передачу без подтверждения приема каждого информационного слова нельзя, так как нет уверенности в том, что аппаратура или ЭВМ на приемном конце будет успевать принимать данные. Проблема подтверждения (квитирования) может решаться как аппаратно, так и программно. Так, например, выпускаемая промышленностью система передачи информации СПИ-15А (СПИ-15А.01) предусматривает дуплексный асинхронный обмен по двум радиочастотным кабелям длиной до 1000 м с аппаратным подтверждением на каждое слово. Как и в интерфейсе ИРПр, описанном в п. 3.2, готовность передатчика устанавливается не после завершения передачи предыдущего байта, а после приема его принимающей ЭВМ. В этом случае программирование передачи в режиме ожидания готовности сводится к ожиданию установки бита 7 РКС передатчика и засылки затем в РД передатчика очередного передаваемого байта, а программирование приема – в ожидании установки бита 7 РКС приемника (новый байт пришел) и приеме очередного байта из РД приемника. Выпускаемая серийно система передачи информации включает в себя последовательные интерфейсы ПИ-1 для подключения к ЭВМ СМ-4, ПИ-1М для ЭВМ СМ-1420 и ПИ-2 для подключения к ЭВМ "Электроника 60". Таким образом, с помощью этой аппаратуры можно организовывать двухпроцессорные и многопроцессорные комплексы, состоящие из указанных машин в любом сочетании.

С другой стороны, цифровая система передачи данных по ВОЛС МС-4101 не обеспечивает аппаратного подтверждения приема. Для организации квитирования приходится вместе с информационным словом передавать по линии бит подтверждение передачи, который на приемном конце должен преобразовываться (после приема очередного слова) в бит подтверждения приема и пересылаться в передающую ЭВМ. В зависимости от интерфейса, связывающего аппаратуру МС-4101 с ЭВМ, процедура квитирования может осуществляться аппаратно или программно, как это было описано в п. 3.1. Поскольку в системе МС-4101 информация передается 19-разрядными словами, разряды 16 ... 18 можно использовать в качестве квитирующих.

Глава 4

ИНТЕРФЕЙС КАМАК

4.1. СТРУКТУРА СИСТЕМЫ КАМАК

КАМАК представляет собой систему электронных модулей, предназначенную для построения цифровых измерительных установок, управляемых от ЭВМ. КАМАК удачно объединяет в себе, с одной стороны, богатый набор электронных функциональных модулей самого разнообразного назначения (усилители, счетчики, таймеры, аналого-цифровые и цифро-аналоговые преобразователи, запоминающие устройства и т. д.), а с другой стороны, – средства связи всей этой аппаратуры с ЭВМ, для чего предусмотрен специальный управляющий модуль–контроллер КАМАК. Характерным для системы КАМАК является наличие унифицированного канала передачи данных (магистрали) между отдельными модулями и контроллером. И модули, и контроллер имеют выход на магистраль, по линиям которой происходит обмен рабочей и служебной информацией, а также питание модулей; контроллер, кроме того, связан с ЭВМ (см. рис. 1.1). Всеми процессами на магистрали управляет (по командам от ЭВМ) контроллер, однако если в модуле возникла ситуация, требующая вмешательства ЭВМ, модуль может послать в контроллер запрос на обслуживание и инициировать тем самым конкретную программу обработки.

Стандартизация модулей по конструкции, способу подсоединения к магистрали, характеристикам электрического питания, параметрам входных и выходных сигналов позволяет быстро собирать и модернизировать экспериментальные установки, комплектуя их требуемыми модулями, а единая система команд существенно облегчает разработку алгоритмов управления системой. При этом компоновка любой измерительной системы сводится по существу к составлению программы взаимодействия ЭВМ с модулями (порядок опроса состояния модулей, записи и съема информации и т. д.), технические же вопросы согласования модулей друг с другом или с контроллером отпадают ввиду всесторонней стандартизации системы.

Конструктивной основой системы КАМАК является специальный каркас – *крейт*, содержащий 25 станций – направляющих, по которым в крейт вдвигаются модули. В зависимости от сложности модуль может иметь единичную ширину $l = 17{,}2$ мм и занимать одно место в крейте либо ширину, кратную l. Контроллер обычно занимает два крайних правых места. Таким образом, в крейте может размещаться до 23 различных модулей. Каждый модуль имеет стандартный 86-контактный разъем для подсоединения к магистрали. Разводка линий магистрали по контактам разъемов всех станций (кроме крайней правой, принадлежащей контроллеру) выполнена единообразно, что позволяет устанавливать любые модули на любые места крейта.

Рис. 4.1. Магистраль крейта КАМАК

На рис. 4.1 приведено схематическое изображение магистрали крейта. Бо́льшая часть линий магистрали – параллельные линии, соединяющие одноименные контакты всех разъемов; сигналы, передаваемые по этим линиям, доступны всем модулям. Рабочая информация в системе КАМАК передается 24-разрядным двоичным параллельным кодом, для чего служат 24 линии чтения *R* (передача из модулей в контроллер) и 24 линии записи *W* (передача данных из контроллера в модули). Поскольку в каждом модуле могут размещаться несколько функциональных узлов (например, несколько счетчиков) и, кроме того, еще имеются многочисленные обслуживающие схемы, для адресации к элементам модуля служат 4 линии субадреса *A*, по которым номер узла в модуле или его субадрес передается также двоичным параллельным кодом. Всего, таким образом, в каждом модуле может использоваться до $2^4 = 16$ субадресов.

В процессе обращения контроллера к модулю может быть задано выполнение различных операций – чтение или запись информации, опрос состояния регистра и т. д. Для передачи кода операции предусмотрены 5 линий функций *F*, что дает возможность использовать до 32 различных функций. Значения функций стандартизированы (см. табл. 4.1), например, функция *F(2)* никогда не используется для записи информации в регистр, а только для чтения его содержимого с последующим сбросом. Назначение же регистров и характер содержащейся в них информации зависят от функционального назначения и конкретной схемы модуля.

Группа параллельных линий отводится для управления и передачи служебных сигналов. Сюда относятся линии (и соответственно сигналы) *Z, C, I, B, Q, X, S1, S2*. Некоторые из этих сигналов (*B, S1, S2*) генерируются контроллером или модулями автоматически в процессе обмена информацией по магистрали, на них нельзя воздействовать программным образом; другие сигналы устанавливаются, снимаются либо контролируются программно, и их назначение необходимо понимать для правильного составления программ управления.

Рассмотрим кратко последнюю группу сигналов.

Сигнал *Z* (*Начальная установка*) служит для приведения всей системы в исходное состояние. По этому сигналу сбрасываются (очищаются)

Таблица 4.1. **Функции системы КАМАК**

Номер функции	Наименование	Двоичный код
F(0)	Чтение регистра первой группы	00000
F(1)	Чтение регистра второй группы	00001
F(2)	Чтение и сброс регистра первой группы	00010
F(3)	Чтение обратного кода регистра первой группы	00011
F(8)	Проверка запроса	01000
F(9)	Сброс регистра первой группы	01001
F(10)	Сброс запроса	01010
F(11)	Сброс регистра второй группы	01011
F(16)	Запись в регистр первой группы	10000
F(17)	Запись в регистр второй группы	10001
F(18)	Селективная установка регистра первой группы	10010
F(19)	Селективная установка регистра второй группы	10011
F(21)	Селективный сброс регистра первой группы	10101
F(23)	Селективный сброс регистра второй группы	10111
F(24)	Запрещение	11000
F(25)	Исполнение	11001
F(26)	Разрешение	11010
F(27)	Проверка статуса	11011

Применение. Функции *F* с номерами *4, 6, 12, 14, 20, 22, 28* и *30* нестандартизированы; функции с номерами *5, 7, 13, 15, 29, 31* – резервные.

регистры всех модулей крейта, блокируются входы и выходы модулей и т. д.

Сигнал *C* (*Сброс*) вызывает сброс регистров модулей крейта.

Сигнал *I* (*Запрет*) запрещает любые действия в модулях. В отличие от сигналов Z и C, имеющих импульсный характер, сигнал *I* может быть установлен на магистрали на заданное время. После сброса сигнала *I* модули снова становятся работоспособными.

Сигнал *Q* (*Ответ*) генерируется модулем в ответ на адресуемые ему вопросы о состоянии тех или иных узлов. Значение $Q = 1$ рассматривается как утвердительный ответ, $Q = 0$ – как отрицательный. Послав, например, в модуль команду *Проверка состояния входа*, можно по состоянию сигнала *Q* установить, открыт вход модуля (в этом случае $Q = 1$) или закрыт ($Q = 0$). Если модуль имеет несколько входов, можно опросить их последовательно и, анализируя состояние сигнала *Q* после каждого вопроса, выяснить, какие входы открыты и какие закрыты.

Сигнал *X* (*Команда принята*) вырабатывается модулем всякий раз при получении им "законной" команды, которую данный модуль в состоянии выполнить. Нулевое значение сигнала X ($X = 0$) указывает на наличие неисправности (например, отсутствие адресуемого модуля)

или серьезной ошибки в программе обслуживания (в модуль послана команда, которую он не может выполнить).

Две группы линий (N и L) служат для установления связи контроллера с определенными модулями. В отличие от остальных линий магистрали линии N и L имеют радиальный характер; каждый модуль связан с контроллером индивидуальной парой линий N и L. Когда контроллер генерирует команду обращения к какому-то модулю, он устанавливает соответствующую функцию КАМАК на линиях F, требуемый субадрес – на линиях A и возбуждает линию N, соответствующую адресуемому модулю. Сигналы F и A поступают во все модули. Однако воспринимает их только тот модуль, который подсоединен к возбужденной линии N, т. е. модуль, установленный на станции с номером N.

Если в модуле создалась ситуация, требующая вмешательства ЭВМ (АЦП преобразовал входной сигнал в код, счетчик зарегистрировал заданное число импульсов и т. д.), модуль может послать в контроллер запрос на обслуживание, установив логическую 1 на линии L. Обычно возбуждение линии L (L-запрос) приводит к прерыванию текущей программы и переходу на программу обработки прерывания от данного модуля. Поскольку от каждого модуля в контроллер идет индивидуальная линия L, контроллер, получив запрос, может определить, из какого именно модуля он пришел.

Как уже отмечалось, каждая команда, с которой контроллер обращается к какому-либо модулю, состоит из трех элементов: функции F, субадреса A и номера адресуемого модуля N. Управление аппаратурой КАМАК и заключается в выполнении последовательности команд NAF (команд КАМАК), соответствующей заданному алгоритму функционирования установки. Требуемая последовательность команд NAF записывается в виде машинной программы. Выполнение ЭВМ группы машинных команд, описывающих некоторую команду КАМАК, приводит к передаче в контроллер всех трех элементов этой команды: N, A и F. Контроллер, получив эту информацию, возбуждает соответствующие линии магистрали крейта, чем и достигается выполнение команды.

Как и в любой программе реального времени, последовательность выполняемых операций обычно не является фиксированной, а определяется ходом процесса измерений. Например, получив от модуля АЦП запрос на обслуживание, ЭВМ выполняет программу приема из модуля и записи в оперативную память подготовленного модулем кода входного сигнала. Если, однако, при этом выясняется, что общий объем накопленной информации достиг заданного значения, ЭВМ выполняет программу выключения модуля АЦП – блокировки его входа, запрещения генерации им L-запросов и т. д. Все рассмотренные в предыдущих главах программные средства управления измерительной аппаратурой в полной мере приложимы и к аппаратуре КАМАК, однако техника программирования аппаратуры КАМАК имеет значительную специфику, отражающую сложность и универсальность системы.

4.2. ПРИНЦИПЫ ПРОГРАММИРОВАНИЯ АППАРАТУРЫ КАМАК

Хотя общий принцип программирования аппаратуры КАМАК – создание требуемой последовательности команд *NAF* – во всех случаях остается справедливым, техника программирования зависит от типа используемого контроллера крейта. Описываемые ниже правила программирования относятся к широко распространенному контроллеру, входящему в состав ИВК на базе ЭВМ СМ-4, "Электроника 60", САМАС–MERA и др.

Для обеспечения программного обращения к функциональным узлам модулей КАМАК в контроллере крейта предусматривается выделение для каждого модуля 16 адресов на ОШ, по числу возможных субадресов в модуле; 16 адресов выделено и для регистров самого контроллера (хотя в действительности в контроллере всего 3 программно-адресуемых регистра). Всего, таким образом, для каждого крейта в адресном поле ЭВМ отводится 16×24 = 384 адреса, при этом в установку может входить до четырех крейтов КАМАК. Начальный (базовый) адрес каждого крейта устанавливается в его контроллере с помощью перемычек; для крейта 1 базовый адрес должен быть равен $164\,000_8$, для крейта 2 – 166000_8, для крейта 3 – 162000_8 и для крейта 4 – $160\,000_8$. В табл. 4.2 указаны адреса функциональных узлов модулей КАМАК для крейта 1 с базовым адресом 164 000. Видно, что для обращения по субадресу *A* (*0*) модуля, установленного на станции 1, следует указывать адрес 164 040; для обращения по субадресу *A* (*1*) того же модуля – адрес 164 042 и т. д. в порядке возрастания адресов. Адреса функциональных узлов модуля, установленного на станции 2, начинаются с 164 100; модуля, установленного на станции 3, – с 164 140 и т. д.

Контроллер крейта имеет три программно-адресуемых регистра: регистр управления и состояний (РУС) с адресом 164 000, регистр запросов и маски запросов (РЗМ) с адресом 164 002 и регистр старшего байта (РСБ) с адресом 164004.

Формат РУС приведен на рис. 4.2. Младшие 5 разрядов используются для указания контроллеру крейта номера выполняемой функции. Если, например, требуется выполнить функцию *F*(*8*), то в разряды 0...4 РУС следует записать число 8: MOV # 8., 164 000

С помощью разряда 5 РУС можно устанавливать и сбрасывать сигнал запрета *I* на магистрали крейта. Первое делается командой BIS # 40, 164000, второе – командой BIC # 40, 164 000.

Разряд 6 РУС, как и обычно, служит для разрешения прерывания ЭВМ по запросу из контроллера крейта, а разряд 7, устанавливаемый контроллером аппаратно, индицирует наличие такого запроса. Этот разряд используется при управлении аппаратурой КАМАК в режиме ожидания готовности, или, лучше сказать, в режиме ожидания запроса на обслуживание.

Таблица 4.2. **Адреса функциональных узлов модулей КАМАК для крейта 1**

Обозначение	Адрес на ОШ	Обозначение	Адрес на ОШ
N(1)A(0)	164040	*N(2)A(2)*	164104
N(1)A(1)	164042	.	
N(1)A(2)	164044	.	
⋮		.	
N(1)A(15)	164076	*N(2)A(15)*	164136
N(2)A(0)	164100	*N(3)A(0)*	164140
N(2)A(1)	164102	.	
		.	
		.	
		N(23)A(15)	165376

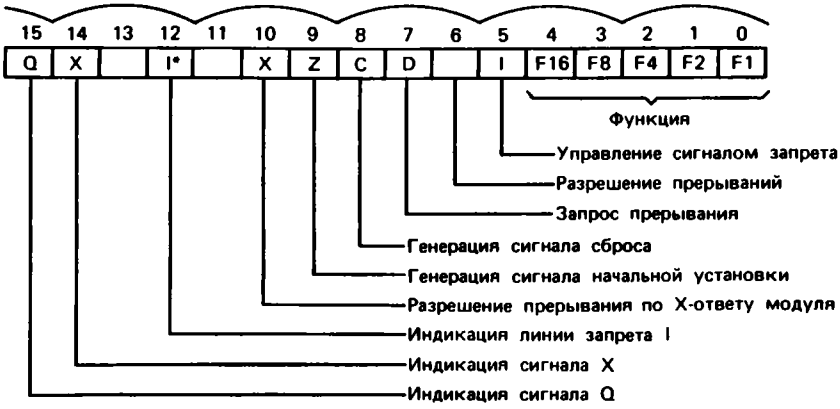


Рис. 4.2. Формат РУС контроллера КАМАК

С помощью разрядов 8 и 9 генерируются сигналы *C* и *Z* соответственно. Программное возбуждение этих сигналов выполняется командами

```
MOV #400, 164000
MOV #1000, 164000
```

Следует заметить, что после генерации соответствующих сигналов разряды 8 и 9 сбрасываются автоматически аппаратурой контроллера; сбрасывать программно их не требуется.

Установка разряда 10 РУС разрешает прерывания ЭВМ в том случае, когда из адресуемого модуля получен ответ $X = 0$. Это дает возможность программно обрабатывать аппаратные ошибки (например, неисправность адресуемого модуля).

Разряд 12 РУС индицирует состояние линии запрета *I* магистрали крейта и позволяет в случае необходимости программно определить наличие или отсутствие запрета.

Наконец, в разрядах 14 и 15 РУС аппаратно устанавливается значение ответов модуля *X* и *Q* на последнюю команду КАМАК. Анализировать ответы *Q* можно с помощью команды TST:

```
TST  164000
BPL  Q0        ; Переход на Q0, если Q = 0
BMI  Q1        ; Переход на Q1, если Q = 1
```

Команды КАМАК, с помощью которых генерируются сигналы *I*, *C* и *Z*, носят название *безадресных*. Рассмотрим теперь выполнение *адресных* команд КАМАК, с помощью которых происходит программное обращение к функциональным узлам конкретных модулей.

Программное выполнение *адресных* команд КАМАК осуществляется в два этапа с помощью двух или даже трех машинных команд. Сначала в РУС записывается код требуемой функции, затем выполняется необходимая операция по адресу конкретного функционального узла модуля, для чего чаще всего используются команды MOV (если функция КАМАК предусматривает передачу данных) и TST (для управляющих функций, выполняемых без передачи данных по линиям *R* и *W*).

Пусть на станции 1 установлен некоторый модуль, в котором используются следующие команды КАМАК:

F(0)A(0) – чтение содержимого регистра модуля;

F(16)A(0) – запись информации в регистр модуля;

F(9)A(0) – сброс регистра модуля

Введем символические обозначения:

```
CSR = 164 000  ; Адрес РУС
AQ = 164 040   ; Адрес регистра на ОШ
```

Чтение содержимого регистра модуля в регистр *R0* процессора осуществляется следующей парой команд:

```
CLR  CSR       ; Занесение в РУС кода функции F(0)
MOV  A0, R0    ; Передача информации
```

Запись некоторого числа в регистр модуля выполняется так:

```
MOV  #16., CSR   ; Занесение в РУС кода функции F(16)
MOV  #100., A0   ; Запись в регистр модуля числа 100
```

Сброс регистра не сопровождается передачей информации, поэтому в качестве второй, исполняющей команды используется команда TST:

```
MOV  #9., CSR    ; Занесение в РУС кода функции F(9)
TST  A0          ; Выполнение команды (сброс регистра)
```

Как уже отмечалось выше, в составе магистрали крейта имеются 24 линии чтения и 24 линии записи. Это дает возможность передавать

24-разрядные данные. Однако длина слова ЭВМ составляет всего 16 разрядов. Для согласования этих величин в составе контроллера имеется специальный 8-разрядный регистр старшего байта (РСБ) с адресом 164004, служащий для временного хранения старшего байта 24-разрядного слова КАМАК.

Для записи 24-разрядного слова в модуль КАМАК необходимо перед обращением к модулю загрузить старший байт данных в РСБ.

При чтении 24-разрядного слова из модуля КАМАК в ЭВМ старший байт слова попадает в РСБ, откуда его можно извлечь соответствующей машинной командой.

Таким образом, для чтения 24-разрядного слова данных в условиях предыдущего примера надо выполнить следующие команды:

```
CLR  CSR          ; В РУС F(0)
MOV  A0, R0       ; Младшие 16 разрядов в R0
MOVB CSR+4, R1    ; Старшие 8 разрядов в R1
```

Запись 24-разрядного числа также потребует трех машинных команд:

```
MOVB  #377, CSR+4  ; Старший байт в РСБ
MOV   #16., CSR    ; Функция F(16)
MOV   #177777, A0  ; Передача всего слова
```

Последней командой выполняется собственно передача в регистр модуля 24-рязрядного слова данных. Легко видеть, что в приведенном примере все 24 разряда регистра заполнились логическими 1.

4.3. ОБСЛУЖИВАНИЕ ЗАПРОСОВ ОТ МОДУЛЕЙ КАМАК

В простейшем варианте работа с запросами от модулей КАМАК мало отличается от обслуживания любого электронного оборудования, подключенного к ЭВМ через стандартный интерфейс. Если модуль КАМАК готов к передаче данных, он устанавливает *L*-запрос на подсоединенной к нему линии *L* магистрали крейта, который, поступив в контроллер, устанавливает разряд 7 РУС, что свидетельствует о готовности модуля и, если разряд 6 РУС установлен, возникает сигнал прерывания. Программа обработки прерывания выполняет необходимые действия по обслуживанию модуля, например считывает из него информацию. Для повышения скорости приема данных может оказаться целесообразным отказаться от режима прерывания. В этом случае программа непрерывно опрашивает разряд 7 РУС (командой TSTB CSR) в ожидании его установки. Зафиксировав установление логической 1 в разряде 7 РУС, программа переходит к считыванию данных из модуля, после чего опять возвращается к циклу ожидания готовности модуля.

Однако реально дело обстоит значительно сложнее, так как программа обычно обслуживает не один, а несколько модулей одновременно, при этом в каждом модуле может быть несколько источников запросов

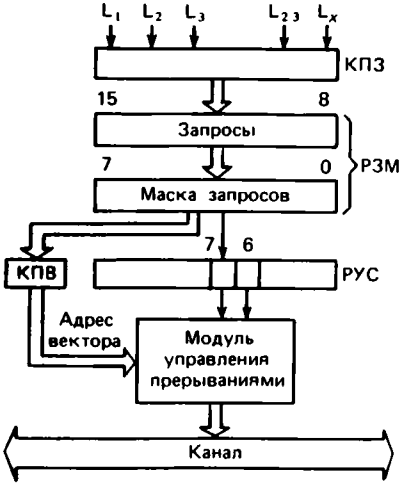


Рис. 4.3. Схема прохождения запросов от модулей в контроллере КАМАК

на обслуживание. Далее по ходу измерений может оказаться необходимым изменять условия работы отдельных модулей или их узлов: блокировать или разблокировать входы модулей, разрешать или запрещать прерывания от модулей в целом или от отдельных узлов модулей и т.д. Для выполнения всех этих действий в системе КАМАК предусмотрены средства управления взаимодействием модулей и контроллера. Частично эти средства реализованы в контроллере крейта, частично в модулях. Рассмотрим сначала средства управления запросами, имеющиеся в контроллере крейта (рис. 4.3).

L-запросы от модулей, а также специальный запрос L_X на прерывание при $X = 0$ поступают на *коммутационное поле запросов* (КПЗ), имеющее 8 выходов. Запрос L_X скоммутирован на выход 8, остальные запросы коммутируются пользователем в зависимости от конфигурации измерительной установки. Если в установке имеется не более 8 модулей, вырабатывающих L-запросы, то каждому из них можно поставить в соответствие один из выходов КПЗ. При этом надо иметь в виду, что получившиеся 8 запросов будут иметь разные приоритеты: чем больше номер запроса, тем выше его приоритет. Естетственно, приоритетность запросов будет иметь значение только в режиме прерываний, да и то лишь в том случае, когда несколько запросов случайно совпадут во времени. Если число модулей в установке больше 8, придется к каждому выходу КПЗ подсоединять линии запросов от нескольких модулей. Запросы внутри каждой такой группы будут связаны с одним вектором прерывания и вызывать одну программу обработки прерывания. Определение конкретного источника запроса в этом случае должна выполнять ПОП путем последовательного опроса модулей данной группы.

С выхода КПЗ 8 сигналов запросов поступают на соответствующие разряды старшего байта РЗМ. Этот байт образует *регистр L-запросов*. При необходимости его содержимое можно прочитать из программы и выяснить, какие именно модули послали запросы на обслуживание (точнее говоря, какая из 8 групп модулей послала запрос).

В контроллере предусмотрена возможность маскирования (блокирования) отдельных запросов, для чего используется младший байт РЗМ, выполняющий функции *регистра маски L-запросов*. Для того

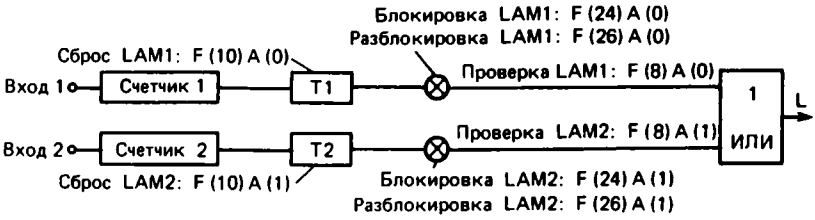


Рис. 4.4. Схема прохождения *LAM*-требований в модуле КАМАК

чтобы пропустить конкретный запрос через регистр маски на схему прерывания, нужно в соответствующем разряде этого регистра установить 1.

Если хотя бы один запрос прошел через маску, он устанавливает разряд 7 РУС, и, если разряд 6 РУС установлен (прерывания разрешены), модуль управления прерываниями начинает процедуру прерывания. В контроллере предусмотрена возможность прерывания от 8 различных источников и каждому из них соответствует свой вектор прерывания. Значения адресов векторов прерывания для каждого из 8 запросов устанавливаются пользователем с помощью *коммутационного поля векторов* (КПВ). Обычно используются соседние векторы с адресами, например, 300, 304, 310, 314_8 и т. д.

Рассмотрим теперь формирование запроса на обслуживание (*L*-запроса) в самом модуле на примере сдвоенного счетчика с предустановкой, содержащего типичные средства формирования запроса. Модуль включает два двоичных 24-разрядных счетчика, работающих независимо друг от друга. Счетчики считают импульсы, поступающие на их входы, расположенные на передней панели модуля. Переполнение любого счетчика вызывает так называемое *LAM-требование*, которое в конечном итоге поступает на линию *L* магистрали крейта, образуя *L*-запрос модуля. Таким образом, в модуле имеется два источника запросов на обслуживание, и для того, чтобы между ними не возникало путаницы, предусмотрена целая система блокировок и проверок (рис. 4.4).

Сигналом переполнения каждого счетчика устанавливается триггер переполнения (*T1* или *T2*), называемый обычно *триггером состояния источника LAM-требования*. Оба триггера образуют *регистр состояния*. Сигналы с выходов обоих триггеров состояния (сигналы *LAM*-требований) через программно-управляемые вентили (маски *LAM*-требований) поступают на линию *L*-запроса магистрали крейта. Любое из *LAM*-требований модуля можно заблокировать (в рассматриваемом примере для этого используется функция $F(24)A(i)$, $i = 0,1$) или, наоборот, разблокировать (функцией $F(26)A(i)$) и тем самым блокировать или разблокировать прерывания от каждого счетчика. Для того чтобы выяснить, какой именно счетчик требует обслуживания, предусмотрена функция

проверки *LAM*-требования *F(8)A(i)*, которая проверяет состояние выхода вентиля. Если данное *LAM*-требование присутствует, модуль в ответ на команду *F(8)A(i)* вырабатывает сигнал $Q = 1$; если данное *LAM*-требование отсутствует, то сигнал $Q = 0$.

Триггер состояния может хранить запрос на обслуживание неограниченно долго. После того как программа определила источник *L*-запроса, соответствующий триггер состояния должен быть сброшен, для чего предусмотрена команда *F(10)A(i)*.

Приведенная схема является типичной, но не единственно возможной. Так, в рассмотренном модуле не предусмотрена проверка *L*-запроса всего модуля, являющегося логической суммой *LAM*-требований; нет также возможности заблокировать *L*-запрос одной командой. Иногда такие возможности предусматриваются конструкцией модуля. При большом количестве потенциальных источников запроса в модуле их максировка может осуществляться не по отдельности в каждом функциональном узле модуля, а одним обращением к регистру маски, куда с линий *W* магистрали соответствующей командой записывается маскирующее (размаскирующее) слово. Могут быть и другие отличия.

Рассмотренная система прохождения запроса на обслуживание предоставляет гибкие возможности программного (в том числе динамического) управления взаимодействием модулей и контроллера. Рассмотрим кратко эти возможности.

1. В измерительную установку входит один модуль с единственным источником запроса обслуживания. Для приема запроса модуля достаточно разблокировать его *LAM*-требование и размаскировать *L*-запрос в контроллере крейта (рис. 4.3 и 4.4). После этого можно либо ожидать готовности модуля путем опроса разряд 7 РУС (при сброшенном разряде 6), либо установить разряд 6 и работать в режиме прерываний.

Для иллюстрации приемов программирования рассмотрим установку, содержащую сдвоенные счетчики с предустановкой. Кроме команд, показанных на рис. 4.4, в модуле счетчиков используются следующие команды:

F(0)A(0), A(1) – чтение содержимого счетчиков *1* и *2* на линии *R* (считываются 24 разряда);

F(2)A(0), A(1) – чтение со сбросом соответствующего счетчика;

F(9)A(0), A(1) – сброс счетчика *1* или *2* вместе с их триггерами переполнения *T1* или *T2*, т. е. вместе с источниками *LAM*-требований;

F(16)A(0), A(1) – сброс счетчика *1* или *2* и запись в них информации (24 разряда), поступающей с линий *W*;

F(25)A(0), A(1) – добавление 1 к содержимому счетчика *1* или *2*;

F(25)A(2) – добавление 1 к содержимому обоих счетчиков.

Предположим, что счетчики используются для задания фиксированной программно-управляемой задержки. В этом случае в счетчик заносится некоторое число, на его вход подаются периодические импульсы от внешнего генератора и фиксируется момент переполнения счетчика.

Если частота генератора равна f, а в счетчик занесено число M, то задержка составит $(2^{24} - M)/f$. Таким образом, если требуется создать задержку в N периодов генератора, в счетчик следует записать число $M = 2^{24} - N$. Для случая, когда в установке используется только один счетчик, программа обслуживания L-запроса модуля в режиме ожидания готовности выглядит следующим образом (модуль установлен на станции 1):

```
A0 = 164 040                    ; Субадрес счетчика 1
CSR = 164 000                   ; Адрес РУС
DHR = 164 004                   ; Регистр старшего байта
N:      .BLKW 1                 ; Ячейка с числом N (≤65535)
START:  MOV    #1000, CSR       ; Генерация сигнала Z
        CLR    R1               ; В R1 младшие 16 разрядов числа 2^16
        SUB    N, R1            ; В R1 2^16 - N
        MOVB   #377, DHR        ; В РСБ старшие 8 разрядов числа M
        MOV    #16., CSR        ; Передача числа 2^24 - N
        MOV    R1, A0           ; в счетчик
        MOV    #26., CSR        ; Разблокировка
        TST    A0               : LAM 1
1$ :    TSTB   CSR              ; Ожидание
        BPL    1$               ; L-запроса от счетчика

        .
        ;; Продолжение программы
```

Командой MOV R1, A0 в счетчик модуля заносится число $M = 2^{24} - N$ и начинается отсчет задержки. Программа останавливается в ожидании L-запроса от модуля. Как только по истечении заданного времени установится разряд 7 РУС, выполнение программы будет продолжено.

В приведенном примере аппаратная задержка использовалась для блокирования выполнения программы на заданное время. Часто требуется, не прекращая выполнения программы, активизировать через заданное время некоторый процесс, например съем показаний измерительной аппаратуры. В этом случае надо воспользоваться режимом прерываний:

```
A0 = 164 040                    ; Субадрес счетчика 1
CSR = 164 000                   ; Адрес РУС
DMR = 164 002                   ; Адрес РЗМ
DHR = 164 004                   ; Адрес РСБ
N:      .BLKW 1                 ; Ячейка с числом N
START:  MOV    #1120, CSR       ; Сигнал Z + разрешение прерываний
                                ; + F(16)
        MOV    #ISR, 300        ; Адрес ПОП в вектор прерываний
        MOV    #240, 302        ; Приоритет 5
        MOV    #1, DMR          ; Маска L-запросов
        CLR    R1               ; Формирование
        SUB    N, R1            ; числового значения
        MOVB   #377, DHR        ; задержки
        MOV    R1, A0           ; Исполнение функции F(16)
```

```
        MOV   #26., CSR     ; Код F(26) в РУС
        TST   A0            ; Разблокировка LAM-требования

        ⋮ ; Продолжение программы

ISR:    MOV   #10., CSR     ; Сброс LAM-требования
        TST   A0            ; и L-запроса
; Съем показаний измерительной аппаратуры
        RTI                 ; Возврат в основную программу
```

2. В установке имеются модули, содержащие несколько источников запросов, как это показано, например, на рис. 4.4. Тогда в ПОП надо прежде всего выяснить (с помощью команд *F(8)A(0)* и *F(8)A(1)*, какой именно узел модуля запросил обслуживание, и перейти на строки обработки прерывания от этого узла. Поскольку может оказаться, что обслуживания требует не один, а несколько узлов модуля, в необходимых случаях следует, не удовлетворившись обслуживанием первого обнаруженного запроса, проверить запросы от остальных узлов модуля.

Усложним предыдущий пример организацией не одной, а двух временных задержек, для чего используем оба счетчика модуля:

```
A0 = 164 040                ; Субадрес счетчика 1
A1 = 164 042                ; Субадрес счетчика 2
CSR = 164 000               ; Адрес РУС
DMR = 164 002               ; Адрес РЗМ
DHR = 164 004               ; Адрес РСБ
N1:     .BLKW 1             ; Задержка 1
N2:     .BLKW 1             ; Задержка 2
START:  MOV   #1120, CSR    ; Z, разрешение прерываний, F(16)
        MOV   #ISR, 300     ; Одна ПОП на оба LAM
        MOV   #240, 302     ; Приоритет 5
        MOV   #1, DMR       ; Маска L-запросов
        CLR   R1            ; Образование
        SUB   N1, R1        ; и занесение
        MOVB  #377, DHR     ; числа M1 = 2^24 - N1
        MOV   R1, A0        ; в счетчик 1
        CLR   R1            ; Образование и занесение
        SUB   N2, R1        ; числа M2 = 2^24 - N2
        MOV   R1, A1        ; в счетчик 2
        MOV   #26., CSR     ; Разблокировка
        TST   A0            ; обоих
        TST   A1            ; LAM-требований

        ⋮ ; Продолжение основной программы

ISR:    MOV   #8., CSR      ; Проверка
        TST   A0            ; LAM1
        BPL   TEST2         ; LAM1 нет
        MOV   #10., CSR     ; LAM1 есть,
        TST   A0            ; сбросить его
```

```
        ⋮; Обработка запроса от счетчика 1
TEST2:  TST   A1          ; Проверка LAM2
        BPL   FIN         ; LAM2 нет
        MOV   #10., CSR   ; LAM2 есть,
        TST   A1          ; сбросить его

        ⋮; Обработка запроса от счетчика 2
FIN:    RTI               ; Возврат в основную программу
```

3. Измерительная установка содержит несколько (не более 8) модулей, в каждом из которых имеется один или несколько источников запросов обслуживания. Поскольку за каждым модулем закреплен свой вектор прерывания, программный комплекс может содержать столько ПОП, сколько имеется модулей. В этом случае содержание каждой ПОП будет таким же, как в предыдущем примере.

4. Установка содержит более 8 модулей – источников запросов. В этом случае, как отмечалось выше, модули объединяются в группы, так что запросы от нескольких модулей приводят к прерыванию через один вектор прерываний. В каждой ПОП следует, очевидно, сначала опросить все модули, объединенные в данную группу, с целью определения конкретного источника запроса, а затем перейти к обслуживанию соответствующего модуля (или функционального узла в нем). Опрос модулей в рассматриваемом примере выполняется с помощью команд *NF(8)A(0)* и *NF(8)A(1)*.

4.4. ПРОГРАММИРОВАНИЕ ПРОСТЫХ ИЗМЕРИТЕЛЬНЫХ СИСТЕМ

В состав измерительных установок часто входит один или несколько АЦП. В стандарте КАМАК разработаны модули АЦП различного назначения, различающиеся скоростью преобразования, разрядностью и другими характеристиками. Структура распространенного модуля АЦП и его основные команды приведены на рис. 4.5. Входные импульсы, пройдя через вентиль блокировки входа, поступают на схему АЦП. Полученный код заносится в выходной буферный регистр БР, откуда по команде чтения *F(0)A(0)* или *F(2)A(0)* поступает на линии *R* магистрали КАМАК. Одновременно с заполнением БР устанавливается триггер *LAM*-состояния *T*, сигнал с которого через вентиль блокировки *LAM*-требования проходит на линию *L* магистрали. Как видно из рис. 4.5, проверку готовности модуля можно осуществлять разными способами. Если *LAM*-требование модуля заблокировано в самом модуле командой *F(24)A(0)*, модуль не будет посылать запросы в контроллер крейта и проверку готовности следует выполнять с помощью команды *F(27)A(0)*. Если же *LAM*-требование разблокировано командой *F(26)A(0)*, то заполнение БР очередным кодом приведет к появлению *L*-запроса, который можно обнаружить, опрашивая разряд 7 РУС или инициируя систему прерываний. Максимальное быстродействие обеспечивается в режиме

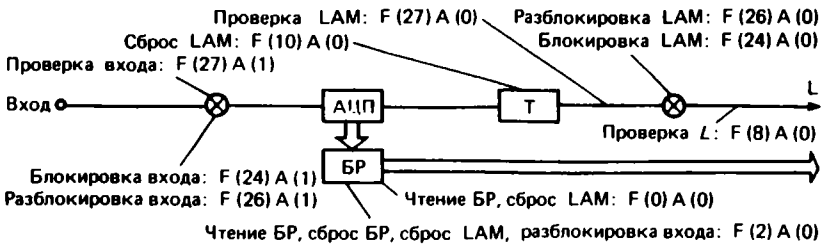


Рис. 4.5. Структура модуля АЦП КАМАК

ожидания готовности (установки разряда 7 РУС), поэтому, если в установку входит только один модуль (источник запросов), этот способ является предпочтительным.

Рассмотрим простейшую измерительную систему, в которую входят АЦП и счетчик с предустановкой, используемый в качестве таймера. Окончание заданного интервала времени естественно отработать по прерыванию от счетчика, накопление же кодов от АЦП целесообразно проводить в режиме ожидания готовности.

Будем считать, что АЦП установлен на станции 2, а счетчик с предустановкой – на станции 3. L-запрос от этой станции скоммутирован на разряд 0 РЗМ, и адрес вектора прерывания, закрепленного за запросом от счетчика, равен 300. АЦП настроен на преобразование входных амплитуд в 8-разрядные коды, и в процессе измерений требуется получить распределение регистрируемых импульсов по каналам.

Пусть на вход счетчика поступают периодические импульсы с частотой 1 кГц, а требуемое время экспозиции составляет 100 с. Поскольку используемый нами счетчик не может считать в обратном направлении (вычитая поступающие импульсы), в него надо заранее занести такое число, чтобы, складывая его с числом импульсов, поступивших на его вход, через 100 с он переполнился. Это число составляет

$$2^{24} - 100 \cdot 10^3 = 2^{24} - 10^5 = 77\,777\,777_8 + 1 - 303\,240_8 = 77\,474\,540_8.$$

Младшие 16 разрядов этого числа образуют 074 540, старшие 8 разрядов (старший байт слова КАМАК) – 376. Программа управления установкой может иметь следующий вид:

```
CSR = 164 000                    ; Адрес РУС
ADC0 = CSR + (2 * 32).           ; A0 модуля АЦП
ADC1 = ADCO + 2                  ; A1 модуля АЦП
TIMER = CSR + (3 * 32).          ; A0 счетчика
SPECTR:  .BLKW 256.              ; Область под спектр
FLAG:    0                       ; Флаг окончания измерений
START:   MOV  #CSR, R1           ; Адрес РУС в R1
         MOV  # 1000, (R1)       ; Сигнал Z
         MOV  #ISRCNT, 300       ; Адрес ПОП счетчика
```

```
        MOV  #240, 302        ; Приоритет 5
        MOV  #1, 2 (R1)       ; Маска L-запросов в РЗМ
        MOV  #376, 4 (R1)     ; Заполнение РСБ
        MOV  #16., (R1)       ; Предустановка
        MOV  #74540, TIMER    ; счетчика
        MOV  #26., (R1)       ; Функция F(26) в РУС
        TST  ADC1             ; Разблокировка АЦП
        TST  TIMER            ; Разблокировка LAM счетчика
        MOV  #100+27., (R1)   ; Разрешение прерываний + F(27)
1$ :    TST  FLAG             ; Если флаг установлен,
        BNE  2 $              ; перейти к обработке спектра
        TST  ADC0             ; Проверка LAM АЦП
        BPL  1 $              ; и ожидание Q = 1
        BIC  #25., (R1)       ; Теперь в РУС код функции F(2)
        MOV  ADC0, R2         ; Код входной амплитуды
        ASL  R2               ; Код x 2
        INC  SPECTR (R2)      ; Инкремент в канале
        BIS  #27., (R1)       ; Восстановление F(27)
        BR   1 $              ; Возврат в цикл ожидания
2$ :
        ⋮; Строки обработки спектра

ISRCNT: MOV  #24., CSR        ; Сброс LAM
        TST  TIMER            ; счетчика и
        TST  ADC1             ; блокировка входа АЦП
        INC  FLAG             ; Установка флага
        RTI
```

В начале основной программы генерируется сигнал Z, заполняется вектор прерываний от счетчика, разблокируется в РЗМ прохождение запроса на прерывание от счетчика, выполняется предустановка счетчика. Занесение в R1 базового адреса контроллера крейта и использование затем адресации через этот регистр несколько сокращают размер загрузочного модуля и ускоряет его выполнение. После разблокировки входа АЦП и *LAM*-требования счетчика и разрешения прерываний в РУС контроллера начинается накопление информации и одновременно отсчет времени. *L*-запрос от АЦП остается заблокированным как в самом модуле, так и в РЗМ. В результате прерывание может возникнуть по единственной причине – из-за переполнения счетчика. С другой стороны, пропадает возможность ожидания готовности АЦП проверкой разряда 7 РУС; эту проверку необходимо теперь выполнять непосредственно в модуле (проверка *LAM*-требования командой *F(27)A(0)*).

Для того чтобы по прерыванию от счетчика перейти на обработку спектра, в программе предусмотрен флаг FLAG, состояние которого проверяется в цикле ожидания *LAM*-требования от АЦП. Если флаг оказывается установлен, программа немедленно переходит на обработку спектра.

При появлении в БР АЦП кода входной амплитуды устанавливается *LAM*-требование и программный цикл ожидания готовности разрывается. Для чтения кода из БР в РУС следует записать код функции *F(2)*, в то время как до этого там находился код функции *F(27)*. Смена кода функции осуществляется командой BIC #25., (R1), которая сбрасывает ненужные разряды кода функции, оставляя установленным разряд 6 разрешения прерываний. После выполнения процедуры поиска и инкремента нужного канала в РУС восстанавливается код функции *F(27)* и программа возвращается в цикл ожидания готовности.

В ПОП от счетчика блокируются вход АЦП и *LAM*-требование счетчика и устанавливается флаг окончания измерений. Если прерывание пришло во время выполнения цикла ожидания готовности, управление будет немедленно передано на строки обработки спектра. Если же прервалась регистрация кода в спектре, после возврата из ПОП этот участок программы выполнится до конца и переход на строки обработки произойдет после выполнения команды BR 1$.

Рассмотрим теперь более сложную измерительную систему, в которую входят пять АЦП, фиксирующих пять параметров входного события. Например, в эксперименте по исследованию потока заряженных частиц – продуктов некоторой реакции это могут быть две координаты места образования частицы, две координаты ее пролета на выходе измерительной установки и пролетное время либо энергия частицы. В ядерно-физических исследованиях координата места регистрации частицы в детекторе часто определяется по значению разности моментов возникновения импульсов на двух выходах детектора. Полученный интервал времени преобразуется в амплитуду импульса, после чего для получения цифрового кода можно использовать обычный АЦП. Пролетное время также сначала преобразуется в амплитуду, и лишь после этого – в цифровой код.

При невысокой частоте поступления событий (потоке частиц невысокой плотности) регистрацию кодов целесообразно осуществлять в режиме прерываний. Это освободит ЦП для выполнения другой полезной работы, например предварительной обработки поступающей информации с целью вывода ее на экран графического дисплея. В то же время, поскольку все пять кодов возникают практически одновременно и данные от всех АЦП надо обрабатывать совместно, в качестве единственного источника прерываний можно использовать любой АЦП. Однако в ходе измерений возникают неполноценные события, когда частица, зарегистрированная в первом детекторе, не попадает во второй. Такие события, образованные только тремя или двумя кодами из пяти, следует отбрасывать. Операция отбраковки неполноценных событий часто выполняется с помощью аппаратных средств, однако может осуществляться и программно.

Рассмотрим вариант программы, в которой при регистрации каждого полноценного события в ОП записываются пять кодов, полученных из

АЦП, неполноценные же события отбрасываются. Всего требуется накопить 1000 событий. Пусть АЦП расположены на станциях 2, 4, 6, 8 и 10, их *L*-запросы скоммутированы на разряды 0,1...4 РЗМ, а адреса векторов прерывания равны 300, 304, 310, 314 и 320_8:

```
CSR = 164000                    ; Адрес РУС
ADC10 = CSR + (2 * 32.)         ; Субадрес 0 АЦП 1
ADC11 = ADC10 + 2               ; Субадрес 1 АЦП 1
ADC20 = CSR + (4 * 32.)         ; Субадреса
ADC21 = ADC20 + 2               ; АЦП 2
ADC30 = CSR + (6 * 32.)         ; Субадреса
ADC31 = ADC30 + 2               ; АЦП 3
ADC40 = CSR + (8. * 32.)        ; Субадреса
ADC41 = ADC40 + 2               ; АЦП 4
ADC50 = CSR + (10. * 32.)       ; Субадреса
ADC51 = ADC50 + 2               ; АЦП 5
DATA:   .BLKW 5 * 1000.         ; Область ОП для данных
ADDR:   .WORD DATA              ; Указатель текущего адреса
CNTR:   1000.                   ; Счетчик событий
START:  MOV     #CSR, R1        ; Адрес РУС в R1
        MOV     #1000, (R1)     ; Сигнал Z
        MOV     #ISR, 300       ; Адрес ПОП и приоритет
        MOV     #240, 302       ; в вектор прерываний от АЦП 1
        MOV     #ISR, 304       ; То же для
        MOV     #240, 306       ; вектора АЦП 2
        MOV     #ISR, 310       ; То же для
        MOV     #240, 312       ; вектора АЦП 3
        MOV     #ISR, 314       ; То же
        MOV     #240, 316       ; для вектора АЦП 4
        MOV     #ISR, 320       ; То же для
        MOV     #240, 322       ; вектора АЦП 5
        MOV     #37, 2 (R1)     ; Маска L-запросов в РЗМ
        MOV     #26., (R1)      ; Функция разблокировки
        TST     ADC10           ; Разблокировка LAM в АЦП 1
        TST     ADC11           ; Разблокировка входа в АЦП 1
        TST     ADC20           ; То же
        TST     ADC21           ; для АЦП 2
        TST     ADC30           ; То же
        TST     ADC31           ; для АЦП 3
        TST     ADC40           ; То же
        TST     ADC41           ; для АЦП 4
        TST     ADC50           ; То же
        TST     ADC51           ; для АЦП 5
        MOV     #102, (R1)      ; Разрешение прерываний + F(2)

        ⋮ ; Продолжение основной программы

ISR:    CMPB    CSR + 3, #37    ; Все АЦП сработали?
        BNE     EXIT            ; Нет, на выход из ПОП
```

ными. Однако эту терминологию не всегда удается выдержать. Например, компоненты ОС естественнее называть программами: управляющая программа (монитор), системные обслуживающие программы, программа работы с файлами и т. д. Строго говоря, все эти программные модули, участвуя в вычислительном процессе, представляют собой задания, а не программы. Даже по отношению к пользовательским программным модулям соблюдение указанной терминологии вызывает затруднения. Так, при описании хода выполнения некоторого задания приходится ссылаться на строки соответствующего исходного текста, т. е. на строки программы. Многие участки выполняемого задания традиционно называют программами или подпрограммами: программа обработки прерываний, подпрограмма завершения. Все же в дальнейшем под программой будет в основном пониматься исходный текст некоторого программного модуля, а под заданием (или иногда задачей) – тот же программный модуль, участвующий в вычислительном процессе.

Операционные системы различаются и по другим признакам: скорости реакции на внешнее прерывание, порядку распределения ресурсов ЭВМ между выполняемыми заданиями, возможностям временнóй синхронизации заданий, средствам взаимодействия оператора с запущенным им заданием и т. д. Таким образом, ОС надо выбирать с учетом конкретных условий использования ВС. Анализ этих условий и выбор ОС являются важным этапом при проектировании ИВК. Рассмотрим типичную структуру ОС реального времени (рис. 5.1).

Оператор взаимодействует с ОС, вводя через клавиатуру терминала команды оператора. Эти сообщения поступают в *программу связи с оператором*, которая анализирует введенные команды, проверяет их правильность и либо выполняет затребованные действия самостоятельно, либо запускает соответствующие системные программы. Функции команд оператора весьма разнообразны. С их помощью осуществля-

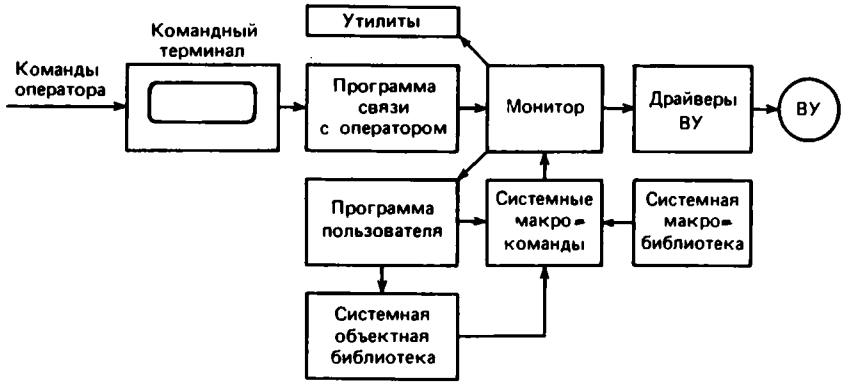


Рис. 5.1. Структура ОС реального времени

ется запуск программ пользователя, вызов системных обслуживающих программ, вывод сообщения о состоянии ВС, перераспределение памяти и внешних устройств, загрузка требуемых системных компонентов, переход на другую ОС и т. д.

Важнейшим элементом ОС является монитор, представляющий набор управляющих программ и системных таблиц и выполняющий основные действия по управлению вычислительным процессом. Именно в мониторе заложены ориентация ОС и ее возможности по организации вычислительного процесса. В функции монитора входит организация реакции ВС на прерывания, обслуживание системного таймера, управление вводом-выводом, наблюдение за очередностью выполнения заданий с учетом их текущих состояний и приоритетов, обработка сбоев и целый ряд других, важных для систем реального времени задач. Доступ к функциям монитора из прикладной программы осуществляется с помощью *системных макрокоманд* (*системных директив*), включение которых в программу обеспечивает динамическое по ходу выполнения задания обращение к нужным функциям монитора.

Указание в тексте программы на языке АССЕМБЛЕРа имени системной макрокоманды (так называемый *программный запрос* к монитору) приводит на этапе трансляции к включению в текст программы вместо имени макрокоманды нескольких строк, называемых макрорасширением и хранящихся в *системной макробиблиотеке*. На этапе выполнения эти строки осуществляют подготовку аргументов запроса к монитору и передачу управления программе монитора, соответствующей конкретному запросу. Следует иметь в виду, что возможности монитора по реализации программных запросов частично определяются в процессе генерации ОС, поэтому перед разработкой программ реального времени надо убедиться в соответствии конфигурации используемого монитора поставленной задаче.

Можно выделить следующие важнейшие действия, выполняемые посредством системных макрокоманд:

ввод-вывод;

обслуживание нестандартных ВУ (измерительной аппаратуры) в режиме прерываний;

временная синхронизация вычислительного, а иногда и измерительного процесса с помощью системного таймера;

автоматическая настройка программ на конкретную аппаратную конфигурацию ИВС;

организация комплексов взаимодействующих программ реального времени;

взаимодействие оператора с выполняемым заданием и управление ходом его выполнения.

приведенный список отнюдь не исчерпывает возможности системных макрокоманд, которые являются важнейшим элементом программ реального времени.

Обращение к большинству макрокоманд допускается как из программ, написанных на языке ассемблера, так и из программ на ФОРТРАНе. Это позволяет в случае необходимости значительную часть программного обеспечения систем реального времени создавать на ФОРТРАНе, не прибегая к использованию относительно трудоемкого программирования на языке ассемблера. Следует, однако, иметь в виду, что программы реального времени, написанные на ФОРТРАНе, требуют большего места в памяти, неоптимальны по времени выполнения и к тому же во многих случаях не отличаются наглядностью.

Подпрограммы на ФОРТРАНе, с помощью которых происходит вызов системных макрокоманд, содержатся в *системной объектной библиотеке*. Состав этой библиотеки определяет возможности использования языка ФОРТРАН при создании программ реального времени в данной ОС. Кроме того, в системной объектной библиотеке могут содержаться различные вспомогательные подпрограммы (преобразования чисел и кодов, обработки строк и т. д.). Подсоединение к основной программе подпрограмм из системной библиотеки производится на этапе компоновки программы автоматически, если комопоновщик (редактор связей) встретил их имена в объектном модуле.

Для организации связи системных и прикладных программ со стандартными ВУ, т. е. для осуществления ввода-вывода в состав ОС включаются *драйверы* ВУ – программы управления внешними устройствами. В функции драйвера входит выполнение операций ввода-вывода на физическом уровне путем обращения к регистрам ВУ, отсчитывание количества переданных слов или байтов, смещение указателя в буфере ОП, откуда (или куда) передаются данные, и т. д. Все эти действия обычно выполняются в режиме прерываний. Монитор следит за этим процессом, инициируя работу драйвера, если в выполняемой программе встретился запрос на ввод-вывод, организуя очередь к драйверу, если запросы на ввод-вывод поступают из программы быстрее, чем драйвер и ВУ успевают их отрабатывать, и выполняя ряд других операций по взаимодействию работы драйвера и основной программы.

Описанные компоненты ОС принадлежат к управляющим программам. Отдельную группу составляют *обслуживающие программы* (*утилиты*). Сюда входят программы работы с файлами на НМД и НМЛ, программы, позволяющие подготовить и отладить прикладную программу (редактор текста, ассемблер, компоновщик, отладчик), программа-библиотекарь и ряд других.

Рассмотрим основные характеристики ОС реального времени на примере широко распространенной ОС с разделением функций РАФОС-II.

Операционная система РАФОС-II (в дальнейшем тексте – просто РАФОС) включает в себя пять мониторов, различающихся возможностями, объемом и быстродействием.

Исполняющий бездисковый RM-монитор предназначен для работы в локальных многомашинных комплексах и на микроЭВМ без внешней

памяти. Монитор загружается в ОП ЭВМ, например, по линии связи вместе с прикладной программой и другими требуемыми компонентами ОС. После загрузки управление передается прикладной программе, которая может содержать программные запросы к монитору. RM-монитор имеет минимальный размер 1,8К, обслуживает ОП объемом до 28К и обеспечивает эффективное управление ВС, носящей чисто исполняющий характер, т. е. являющейся элементом некоторой установки и предназначенной исключительно для ее обслуживания. На такой ВС нельзя выполнять подготовку и отладку новых программ из-за отсутствия утилит; нельзя также оперативно запускать требуемые по ходу измерения задания, так как при работе с RM-монитором выполняться может только то единственное задание, которое было загружено вместе с монитором. Таким образом, RM-монитор является однозадачной системой с минимальными возможностями.

Однозначный SJ-монитор требует для своей работы НМД, на котором хранятся выгружаемые компоненты ОС. Резидентная часть SJ-монитора, всегда находящаяся в ОП, занимает объем немногим менее 2К, хотя часто для нормального выполнения прикладной программы объем резидентной части ОС надо увеличивать до 4К. В отличие от RM-монитора SJ-монитор позволяет единственному пользователю выполнять весь комплекс работ по подготовке задания: создание исходного текста программы на языках АССЕМБЛЕР (МАКРОАССЕМБЛЕР), ФОРТРАН, БЕЙСИК и ПАСКАЛЬ; трансляцию, компоновку и запуск программы; интерактивную отладку программы с помощью специально предусмотренных программ-отладчиков. Кроме того, имеется возможность, используя системные программы, работать с библиотеками и другими файлами на НМД и НМЛ (создавать, уничтожать, копировать и т. д.). При этом все запускаемые пользователем задания выполняются только последовательно, друг за другом, в каждый момент в ОП может находиться только одно задание. В рассмотренном варианте SJ-монитор является однозначным, *однотерминальным,* однопользовательским монитором, обеспечивающим пользователя всеми необходимыми услугами.

Иногда измерительный комплекс требует наличия нескольких операторов, находящихся, например, в разных помещениях и обслуживающих отдельные узлы сложной установки. Однозадачная ОС, естественно, не позволит каждому оператору запускать свои задания – единственное выполняемое задание должно быть запущено с командного терминала. Однако SJ-монитор представляет возможность выполняемому заданию общаться с несколькими терминалами (до 16), передавая на них сообщения и получая от них требуемые данные. Если это средство (многотерминальная поддержка) указано при генерации ОС, система, оставаясь однозначной, становится *многотерминальной.*

SJ-монитор часто используется в управляющих и измерительных комплексах, если требуется обеспечить минимальное время реакции на пре-

рывание, а функции программного обеспечения не слишком разнообразны (например, только сбор данных).

Другой, столь же широко распространенный монитор системы РАФОС – FB-монитор отличается от SJ-монитора практически лишь тем, что позволяет организовать *двухзадачный фоново-оперативный режим.* В этом случае в ОП одновременно находятся два задания – оперативное и фоновое. Задания могут входить в единый программный комплекс, но могут быть и вполне независимыми. Оперативное задание обладает бо́льшим приоритетом и выполняется в первую очередь. Если по каким-либо причинам оперативное задание блокируется (например, оно инициировало вывод накопленных данных на магнитный диск и ждет его завершения или ожидает установочных данных от оператора), начинает выполняться фоновое задание. Такой режим позволяет заметно повысить эффективность использования ресурсов ЭВМ, в первую очередь центрального процессора, и, кроме того, придает бо́льшую гибкость программному комплексу, обслуживающему установку. Действительно, если разделить функции программного обеспечения между двумя заданиями так, чтобы оперативное задание выполняло сбор и накопление регистрируемых данных, а также запись массивов накопленной информации на магнитный диск, а фоновое – анализ и обработку этих массивов, то организация программ обработки заметно упрощается. Оператор, взаимодействуя через терминал с фоновым заданием, может обращаться к тем или иным массивам данных, задавать различные алгоритмы обработки и т.д., не затрагивая при этом выполнение оперативного задания. Более того, при необходимости кардинального изменения алгоритма обработки можно, сняв с выполнения текущее фоновое задание, загрузить и запустить новое, которое будет обращаться к тем же данным, но обрабатывать их по-иному. Все эти манипуляции никак не затрагивают оперативное задание и не нарушают хода измерений.

FB-монитор имеет размер разидентной части около 4К, а включая выгружаемые компоненты 10К. Таким образом, из обслуживаемого монитором объема ОП 28К на прикладные программы остается 18–24К. Так как оперативная и фоновая программы одновременно находятся в ОП, сумма их размеров не должна превышать указанного значения.

Так же как и SJ-, FB-монитор позволяет организовать многотерминальный режим, причем за каждым заданием (как оперативным, так и фоновым) могут быть закреплены по несколько терминалов (в сумме не более 16). Поскольку во всех случаях общение с системой в целом может производиться только с одного командного терминала, FB-монитор следует отнести к двухзадачным однопользовательским системам.

Используемые в настоящее время версии FB-монитора позволяют запускать кроме оперативного и фонового еще до шести *системных заданий*, служащих в основном для расширения функций монитора. Приоритеты этих заданий, определяемые в процессе их запуска, лежат между наивысшим приоритетом оперативного задания и низшим прио-

ритетом фонового. Вычислительная система, в которой запущены системные задания, становится многозадачной (но по-прежнему однопользовательской).

В состав РАФОС включены две подсистемы, запускаемые в качестве системных заданий: подсистема EL регистрации ошибок аппаратных средств ВС и подсистема QUEUE для буферизации вывода на медленные ВУ с целью организации к ним очередей запросов. Поскольку пользователю не рекомендуется разрабатывать системные задания, носящие прикладной характер, в дальнейшем это средство рассматриваться не будет, а FB-монитор будет считаться двухзадачным.

Более сложный и совершенный XM-монитор, также являющийся двухзадачным и однопользовательским, обеспечивает все возможности FB-монитора и сверх того содержит системные программы управления *расширенной памятью*, обращение к которым из прикладной программы осуществляется с помощью специального набора системных макрокоманд.

Как уже отмечалось в гл. 1, особенности архитектуры рассматриваемых машин не позволяют программе непосредственно обращаться за пределы 32К слов. Поскольку 4К адресного пространства резервируется под адреса регистров ВУ (так называемая *страница ввода-вывода*), на ОП остается только 28К. В машинах типа "Электроника 60" объем физической ОП действительно составляет 28К. Такую память, как уже отмечалось, обслуживают RM-, SJ- и FB-мониторы, причем из-за того, что часть ОП занимает сам монитор, программа пользователя не может иметь размер, превышающий 24–26К. С целью расширения возможностей ВС многие ЭВМ оснащаются физической ОП объемом 124К и больше. В этом случае вся ОП, выходящая за пределы 28К, называется расширенной и доступ к ней осуществляется с помощью аппаратуры диспетчера памяти, входящей в состав ЦП. Набор макрокоманд для работы с расширенной памятью как раз и предоставляет возможность программного управления диспетчером памяти.

Наличие в ВС расширенной ОП отнюдь не означает, что программы теперь могут иметь любой размер. По-прежнему максимальный адрес, используемый в программе, не может превышать 2^{16}, т. е. 32К (строго говоря, $2^{16} - 2 = 177776_8$). Однако теперь возникает возможность, расположив часть программы в расширенной памяти (рис. 5.2), полностью использовать адресное пространство задания, "сэкономив" ту его часть, которую ранее занимали монитор и страница ввода-вывода. Обычно в расширенной памяти располагают буфера с данными. Поскольку выделение расширенной памяти для оперативного и фонового задания происходит независимо, увеличение размера одной программы (например, фоновой) не приводит к необходимости сокращать размер другой (оперативной), так как это имеет место в ВС с 28К.

Программа может использовать все 32К своего адресного пространства, только если в ней нет непосредственных обращений к регистрам

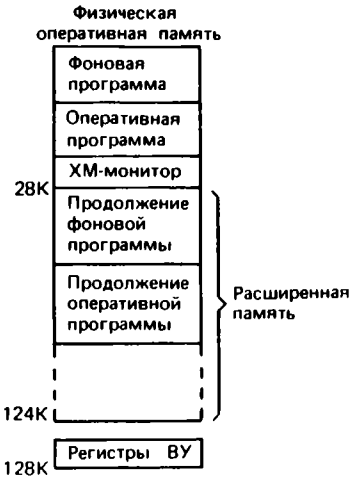


Рис. 5.2. Использование расширенной памяти

ВУ. Если такие обращения имеются, доступное адресное пространство сокращается до 28К. Если же программа, кроме того, работает с таблицами монитора (что может понадобиться, например, для автоматического анализа состояния ВС), то максимальный размер программы уменьшается до 20–21К. Однако и в этом случае использование расширенной памяти целесообразно, так как позволяет заметно увеличить суммарный объем оперативной и фоновой программ.

В состав второй версии системы РАФОС включен многозадачный TS-монитор разделения времени, обеспечивающий одновременную работу с ВС многих (до 20) пользователей, каждый из которых, работая за индивидуальным терминалом, может составлять, отлаживать и выполнять свои задания (в том числе и несколько заданий одновременно) независимо от других пользователей. Как и любая многопользовательная система, TS-монитор защищает файлы и задания пользователей от несанкционированного доступа, что исключает случайные разрушения чужих файлов или заданий. С другой стороны, система предоставляет возможности нескольким пользователям работать с одним файлом, например общим банком данных, а также пересылать сообщения из задания в задание. Реализация всех этих функций потребовала заметного увеличения размера монитора, который в зависимости от конкретной конфигурации занимает объем 20–24К и предназначен для обслуживания ВС с ОП от 48 до 124К.

TS-монитор удобно использовать для организации многопользовательских ВС общего назначения, однако его возможности по управлению измерительными системами ограничены. Хотя в состав TS-монитора включены средства реального времени (обращение к регистрам ВУ, обработка прерываний), однако он не обеспечивает необходимого для задач реального времени быстродействия. Так, если скорость приема информации из измерительной установки в режиме прерываний для FB-монитора может доходить до нескольких тысяч событий в секунду, то TS-монитор обеспечивает прием лишь около 100 событий/с. С другой стороны, все мониторы системы РАФОС программно совместимы друг с другом. Это дает возможность составлять и отлаживать программы под управлением TS-монитора в эффективном многопользовательском

режиме на достаточно мощной ЭВМ, имея в виду последующее использование этих программ в небольших ВС, обслуживаемых SJ- или FB-мониторами. Такая методика, однако, мало пригодна для задач реального времени, так как средства реального времени TS-монитора весьма специфичны и программа реального времени, отлаженная с TS-монитором, с другими мониторами работать не будет (в этом отношении SJ- и FB-мониторы обеспечивают гораздо лучшую совместимость, хотя тоже не полную).

В настоящее время широкое распространение получила операционная система NTS. Идеологически эта система схожа с TS-монитором РАФОС, но отличается от него более эффективной организацией вычислительного процесса. Все сказанное выше о TS-мониторе в полной степени относится и к системе NTS, которую удобно использовать для отладки вычислительных программ, предназначенных для работы под управлением монитором РАФОС, а также для организации ИВК, от которых не требуется высокого быстродействия.

Следует заметить, что непрерывное усовершенствование малых ЭВМ, выпускаемых промышленностью, в частности повышение их быстродействия и увеличение объема оперативной памяти, остро ставит вопрос о выборе ОС для их обслуживания в системах реального времени. Действительно, SJ- и FB-мониторы РАФОС не обслуживают расширенную память, а средства XM-монитора в этом отношении довольно ограничены. Повышение быстродействия процессора вместе с развитием аппаратных средств ЭВМ делают заманчивой идею многопользовательского режима при решении задач реального времени. Как раз эту возможность и предоставляют TS-монитор и система TNS. Невысокое быстродействие этих систем при обработке прерываний может быть в определенной степени скомпенсировано, если программирование измерительной аппаратуры выполнять пользуясь специально разработанными для этой цели драйверами. Написание драйверов для нестандартного оборудования представляет собой довольно трудоемкую операцию, требующую высокой квалификации, однако эти единовременные затраты в дальнейшем упрощают прикладное программирование и позволяют использовать преимущества многозадачных многопользовательских систем TS и NTS в задачах реального времени.

Помимо системы РАФОС в вычислительных комплексах, оснащенных достаточно мощными машинами, используется другая ОС реального времени – ОС РВ. Эта система, являясь многозадачной (в данной системе выполняемая программа называется задачей), обеспечивает управление процессами реального времени с одновременным обслуживанием многих пользователей. Пользователи, работая за индивидуальными терминалами, могут одновременно и независимо выполнять любую доступную системе работу: создавать исходные тексты программ, транслировать, отлаживать или выполнять эти программы, обрабатывать файлы данных на НМД и НМЛ, работать с графическими средствами и т. д. При этом

каждый пользователь работает так, как будто все ресурсы машины находятся в его единоличном распоряжении. Для обеспечения такого режима работы система должна обеспечивать четкое взаимодействие между программами и аппаратными средствами самой ЭВМ.

Организация мультизадачной работы ВС, особенно в условиях выполнения задач реального времени, требует решения целого ряда разноплановых вопросов, которые условно можно разбить на три группы:

планирование ОП, т. е. порядок предоставления ОП отдельным задачам. Поскольку готовые к выполнению задачи могут находиться как в ОП, так и на магнитных дисках (если им не хватило места в памяти), то сюда входят также вопросы взаимодействия оперативной и внешней памяти;

распределение между задачами времени ЦП. Фактически ЦП в любой момент времени выполняет какую-то одну задачу, и мультизадачный режим предполагает регулярное (но совсем не обязательно периодическое) переключение ЦП с одной задачи на другую;

организация взаимодействия задач друг с другом, что может понадобиться в условиях работы единого программного комплекса. Сюда входят вопросы временно́й синхронизации задач, передачи из задачи в задачу данных или служебной информации и пр.

Планирование ОП и распределение между задачами времени ЦП происходят на основе системы приоритетов. Приоритет может быть назначен задаче на разных этапах ее создания, в том числе он может задаваться динамически во время выполнения задачи. Всего может быть 250 значений приоритетов от 1 до 250. В первую очередь всегда выполняется задача с наивысшим приоритетом, но на время, когда она освобождает процессор (ожидание завершения ввода-вывода, ожидание внешнего события и т. д.), автоматически запускается задача с более низким приоритетом. Если в ОП находятся несколько задач с одинаковыми приоритетами, они регулярно сменяют друг друга.

Если активизированных задач больше, чем их помещается в памяти (максимальный размер задачи в ОС РВ составляет 32К), то задачи с более низкими приоритетами находятся на диске. Система следит за ними, и как только в ОП освобождается место, задачи с диска загружаются в память. Выполняемая задача может перейти в заблокированное состояние, если она ждет наступления определенного события, например ввода данных с терминала или завершения вывода на магнитный диск. В этом случае она временнно выгружается на диск, чтобы дать место в ОП ожидающей на диске задаче с более низким приоритетом. Таким образом, система наблюдает за состоянием всех активизированных задач, обеспечивая наиболее эффективное (с точки зрения заложенных в нее алгоритмов) их выполнение.

Измерительно-вычислительные системы относительно редко требуют участия нескольких операторов и соответственно организации многотерминального режима. Однако мультизадачные ОС широко использу-

ются для управления измерительными установками и процессами реального времени. Мультизадачный комплекс реального времени в силу его отчетливо выраженной модульности обладает заметными преимуществами перед однозадачным. При разработке такого комплекса удобно распределить функции программного обеспечения между отдельными задачами, включив, например, в комплекс задачи сбора и накопления данных, индикации, обработки данных по тем или иным алгоритмам, управления ходом измерений, контроля работы установки, аварийные и т. д. В этом случае легко, скажем, изменить алгоритм обработки: для этого надо просто активизировать задачу с требуемым алгоритмом, причем такая замена не затрагивает работу остальных задач. Если на какое-то время отпала необходимость в индикации поступающей измерительной информации, задачу индикации можно снять с выполнения, исключив ее из конкуренции за вычислительные ресурсы и улучшив условия выполнения остальных задач комплекса. Изменение состава комплекса и замену его компонентов можно производить в соответствии с реально складывающейся ситуацией, не прерывая измерений.

Очевидно, что отдельные задачи мультизадачного комплекса должны быть связаны друг с другом. Например, задача сбора и накопления данных должна передавать накопленные данные задачам обработки и индикации; задача, контролирующая работу установки, в случае аварийной ситуации может активизировать обычно бездействующую аварийную задачу; темп работы задачи индикации может определяться темпом поступления измерительной информации и т. д. Система ОС РВ предоставляет богатый набор средств для синхронизации задач по времени и по событиям, передачи данных из задачи в задачу, управления ходом вычислительного процесса как вручную с клавиатуры терминала, так и автоматически с помощью специальной управляющей задачи и т. д. Все это делает ОС РВ очень удобным инструментом для создания программных комплексов реального времени. Подробно в книге система ОС РВ не рассматривается; желающие познакомиться с ней адресуются к пособиям [21] и [28] либо справочнику [22].

5.2. АППАРАТ СИСТЕМНЫХ МАКРОКОМАНД

Как уже отмечалось, программные запросы к монитору, реализуемые с помощью системных макрокоманд, относятся к числу важнейших элементов программных комплексов реального времени. С одной стороны, системные макрокоманды заметно облегчают труд программиста, давая ему как бы язык более высокого по сравнению с языками АССЕМБЛЕРов уровня для эффективного программирования работы многочисленных устройств ввода-вывода и других элементов ВС; с другой стороны, программные запросы к монитору позволяют использовать алгоритмы ОС для решения таких вопросов, как организация па-

раллельных процессов, событийная и временна́я синхронизация заданий, динамическое управление программным комплексом и т. д. Свободное владение аппаратом системных макрокоманд так же необходимо при разработке программного обеспечения ИВК, как и знание языков программирования различных уровней.

Обращение к средствам ОС осуществляется путем включения в текст программы программного запроса (или *марковызова*), представляющего собой имя системной макрокоманды со списком фактических макроаргументов. Для каждой макрокоманды в системной макробиблиотеке хранится текст *макроопределения*, в котором фигурируют *формальные макроаргументы*. Программа АССЕМБЛЕР, встретив в программе макровызов, подставляет на его место текст макроопределения из системной макробиблиотеки, заменив в нем формальные аргументы на фактические и выполнив некоторые дополнительные преобразования, также связанные с формой и количеством фактических аргументов. Таким образом, текст *макрорасширения* макрокоманды, включаемый в программу, сохраняя неизменную структуру, может заметно варьироваться в зависимости от вида фактических аргументов конкретного программного запроса.

Программные строки макрорасширения носят вспомогательный, служебный характер. С их помощью список аргументов макровызова преобразуется в некоторую стандартную форму, удобную для использования программами монитора. Собственно обращение к монитору осуществляется с помощью специальной команды EMT (*эмуляция прерывания*), которой завершаются макрорасширения большей части системных макрокоманд. Действие этой команды будет описано ниже.

Рассмотрим в качестве типичного примера макрорасширение макровызова .ENTER, с помощью которого резервируется место для файла на магнитном диске и открывается канал связи с ним. Для выполнения этого программного запроса в полях данных программ должны быть предусмотрены две области. В первую, размером в четыре слова, директивой RAD50 записывается полная спецификация создаваемого файла. Вторая, размером в три слова, остается пустой; в дальнейшем она будет использована для хранения аргументов макрокоманды .ENTER:

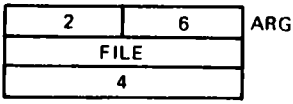
```
FILE:  .RAD50 /DK FILE01DAT/  ; Имя DK:FILE01.DAT
ARG:   .BLKW 3                ; Область для аргументов
```

Для того чтобы создать на диске DK: файл с заданным именем размером в четыре блока (1K) и открыть для связи с ним канал с номером 6, следует включить в программу макровызов .ENTER в следующей форме:

```
FOPN: .ENTER #ARG, #6, #FILE, #4
```

В результате ассемблирования в программу, начиная с метки FOPN, будут вставлены машинные коды, эквивалентные следующим строкам:

Рис. 5.3. Область для аргументов, заполненная макровызовом.ENTER




```
MOV   #ARG, R0                ; Адрес области для аргументов
MOV   #6+(2*400), @R0         ; Номер канала и код .ENTER
MOV   #FILE, 2 (R0)           ; Адрес имени файла
MOV   #4, 4 (R0)              ; Длина файла
EMT   375                     ; Команда перехода в монитор
```

Приведенный текст и является макрорасширением макрокоманды .ENTER с конкретным набором фактических аргументов. После занесения в регистр R0 адреса ARG области для аргументов, отведенной в прикладной программе, начинается заполнение этой области. В младший байт первого слова заносится номер канала (6), в старший байт – код запроса .ENTER (каждый запрос имеет свой код). В последующие слова заносятся адрес спецификации файла и число блоков в файле (рис. 5.3). Заканчивается макрорасширение командой EMT 375, которая передает управление монитору. При этом в R0 находится адрес области, заполненной аргументами, которые, таким образом, становятся доступны программам монитора, обрабатывающим данный запрос.

Монитор, выполнив затребованные действия, возвращает управление в прикладную программу на строку, стоящую за макрокомандой ENTER. После этого область ARG можно использовать повторно, так как заполняющая ее информация больше не нужна. Указание того же аргумента ARG в следующей макрокоманде (не обязательно .ENTER) приведет на этапе выполнения строк макрорасширения к заполнению этой области новыми аргументами, которые, естественно, сотрут старые. Обычно в программах независимо от количества используемых программных запросов предусматривается одна общая для всех запросов область для аргументов размером в пять слов (некоторые макрокоманды требуют область такого размера).

Как видно из текста приведенного макрорасширения, его программные строки используют регистр R0, разрушая его прежнее содержимое. Это свойственно практически всем программным запросам. Таким образом, если содержимое R0 необходимо передать "через" макрокоманду, оно должно быть сохранено перед программным запросом и восстановлено после него:

```
MOV    R0, -(SP)
.ENTER #ARG, #6, #FILE, #4
MOV    (SP)+, R0
```

Макроопределения, содержащиеся в системной макробиблиотеке, составлены таким образом, что допускают значительную свободу в выборе форматов аргументов макрокоманд. Прежде всего, вместо не-

посредственной адресации можно использовать любую другую, подходящую по смыслу, что значительно расширяет возможности программных запросов. Пусть в программе создается несколько файлов, и программа ведет учет использованных каналов (каждый файл требует своего канала), храня в ячейке CHANL номер первого свободного канала. Пусть далее размер файла заранее неизвестен и определяется программно в процессе приема измерительной информации, причем найденное число блоков оказывается в регистре R1. В этом случае макровызов .ENTER может быть записан следующим образом: .ENTER #ARG, CHANL, #FILE, R1

Это приведет к такому макрорасширению:

```
MOV   #ARG, R0          ; Адрес области для аргументов
MOV   #2*400, @R0       ; Код .ENTER в старший байт
MOVB  CHANL, @R0        ; Номер канала в младший
MOV   #FILE, 2 (R0)     ; Адрес имени файла
MOV   R1, 4 (R0)        ; Число блоков
EMT   375               ; Переход в монитор
```

Интересно отметить, что здесь не только изменилась форма аргументов (фактически аргументы переносятся ассемблером в текст макрорасширения в том виде, в котором они фигурируют в исходной строке с макровызовом), но и сам текст макрорасширения стал иным: в нем уже не пять программных строк, а шесть. Это изменение произошло в результате использования аргумента CHANL в форме, соответствующей относительной адресации.

При определении формы записи аргументов в макровызове следует иметь в виду, что в процессе макрорасширения аргументы становятся операндами – источниками команд MOV и MOVB (или иногда BIS, BISB). Поэтому использование, например, в качестве номера канала числа 6 без значка непосредственной адресации приведет к появлению в макрорасширении строки MOV 6, @R0, что, очевидно, лишено смысла (обращение к ОП по адресу 6). В то же время такая ошибка не выявится на этапах трансляции и компоновки, да и при выполнении программы неправильность ее работы может быть обнаружена не сразу.

Не все программные запросы создают блок аргументов в ОП. Например, макровызов .PRINT #MES, служащий для вывода на экран терминала текста, хранящегося по адресу MES, расширяется в две строки:

```
MOV   #MES, R0
EMT   351
```

Малое количество аргументов (всего 1) не требует создания в ОП блока аргументов. Однако все сказанное выше о формах аргументов справедливо и в этом случае. Например, программный запрос .PRINT R5 выведет на экран строку текста, адрес которой помещен в R5, а программный запрос .PRINT ADRMES будет работать со строкой, адрес которой находится в ячейке с меткой ADRMES.

Некоторые макровызовы используют R0 по-иному, помещая в старший байт R0 код данного запроса, а в младший – числовой аргумент, если он имеется. Например, макровызов .CLOSE # 3, закрывающий канал 3 связи с файлом либо ВУ, расширяется в следующие строки:

```
MOV   #3 + (6 * 400), R0
EMT   374
```

Если номер закрываемого канала находится в ячейке ОП по адресу CHANL, то следует использовать запрос .CLOSE CHANL, макрорасширение которого будет иметь вид

```
MOV   #6 * 400, R0
BISB  CHANL, R0
EMT   374
```

В обоих случаях к моменту перехода в монитор в старшем байте R0 будет находиться число 6 – код запроса .CLOSE, а в младшем – номер канала (который, таким образом, не может превышать значение 255).

Некоторые макровызовы используют для размещения аргументов стек задания пользователя. Таков, например, программный запрос .CSISPC, который позволяет ввести в прикладную программу (с клавиатуры терминала или из ОП) стандартную *командную строку* в виде, например, MX1: EXP01.DAT = DK:X.DAT. Такая процедура используется в программах, организующих пересылку файлов с одного ВУ на другое. В приведенной строке файл X.DAT, находящийся на устройстве пользователя DK:, пересылается на гибкий диск MX1, где ему назначается имя EXP01 .DAT.

Программный запрос .CSISPC активизирует интерпретатор командной строки, который проверяет правильность введенной команды, освобождая прикладную программу от необходимости выполнения этой относительно трудоемкой работы. В качестве первого аргумента макровызова указывается адрес области памяти размером 39 слов, куда будут помещены элементы имен файлов, указанных в командной строке, в качестве второго аргумента – адрес четырехсловной области, содержащей типы файлов, используемые по умолчанию. Третий аргумент указывает адрес командной строки. Программный запрос .CSISPC # OUTSPC, # DEFTYP, # 0 приведет к следующему макрорасширению:

```
MOV  #OUTSPC, -(SP)      ; Первый аргумент в стек
MOV  #DEFTYP, -(SP)      ; Второй аргумент в стек
CLR  -(SP)               ; Третий аргумент в стек
EMT  345                 ; Переход в монитор
```

Нуль на месте третьего аргумента является указанием на то, что командная строка будет вводиться с клавиатуры терминала. Очевидно,

что и в этом случае допустимо использование различных способов адресации аргументов в строке с макровызовом.

Макровызовы, которые перед переходом в монитор заполняют блок аргументов в ОП (например, макровызов .ENTER), предоставляют программисту весьма полезную возможность повторных запросов с динамическим изменением значений отдельных аргументов в этом блоке.

Пусть, например, требуется создать на диске DK: шесть файлов для последующего накопления экспериментальных данных с именами FILE01...FILE06 и закрепить за ними каналы 1...6. Это можно выполнить следующим образом:

```
FILE: .RAD50 /DK FILE01DAT/    ; Имя первого файла
ARG: .BLKW 3                   ; Область для аргументов

START:  .ENTER #ARG, #1, #FILE, #4
        MOV     #5, R1         ; Осталось создать 5 файлов
1$:     INCB    ARG            ; Инкремент номера канала
        INC     FILE+4         ; Инкремент номера файла
        MOV     #ARG, R0       ; Настройка R0
        .ENTER                 ; Программный запрос
        SOB     R1, 1$         ; Повторить 5 раз
```

Первый запрос .ENTER с полным списком аргументов создает файл DK:FILE01 .DAT размером в четыре блока и открывает канал 1 связи с ним. Далее в цикле выполняется инкремент номера канала прямо в блоке аргументов (см. рис. 5.3) и инкремент номера файла в третьем слове области FILE со спецификацией файла. Четыре слова этой области в результате выполнения на этапе ассемблирования директивы .RAD50 заполняются кодами RADIX-50 следующих сочетаний символов: DK_, FIL, E01, DAT. Команда INC FILE+4, выполняемая в цикле, записывает в третье слово коды RADIX-50 сочетаний E02...E06. Макровызов .ENTER без аргументов расширяется в единственную строку EMT 375, которая передает управление монитору в предположении, что область аргументов уже заполнена, а ее адрес помещен в R0. Одно из достоинств такого приема программирования заключается в заметной экономии памяти: приведенный фрагмент занимает (в загрузочном модуле) в общей сложности, вместе с областью для аргументов и спецификацией файла, 28 слов, в то время как повторенный 6 раз макровызов .ENTER с полным набором аргументов – 66 слов, а с учетом шести спецификаций файлов и области для аргументов – 93 слова.

Рассмотрим теперь процесс выполнения монитором программного запроса. Команда EMT с тем или иным аргументом инициирует *внутреннее* или *программное прерывание* через вектор прерываний с адресом 30. Такое прерывание называют также *синхронным*, потому что оно возникает всегда в одной и той же точке программы, синхронно с ее выпол-

нением. В ячейке ОП с адресом 30 записан адрес EMT-процессора или EMT-диспетчера, входящего в состав монитора. Команда EMT вызывает такую же реакцию ЦП, как и внешнее прерывание от ВУ: текущее ССП и адрес следующей за EMT команды проталкиваются в стек, а в счетчик команд загружается адрес EMT-процессора из вектора прерывания. Действия EMT-процессора вкратце сводятся к следующим. Прежде всего в стек проталкивается содержимое всех шести РОН с R5 по R0. Следует, однако, иметь в виду, что фактически сохраняются лишь РОН R1–R5, так как содержимое R0 было разрушено в процессе выполнения макрорасширения. Далее монитор читает код команды EMT, вызвавшей прерывание. Адрес этой команды монитору известен: он на 2 меньше адреса возврата, хранящегося в стеке. Как можно заметить из приведенных примеров, в макрорасширениях разных программных запросов фигурируют команды EMT с различными значениями аргумента: EMT 375, EMT 351, EMT 374 и т. д. В действительности дело обстоит сложнее, так как, например, команда EMT 375 характерна не только для макрокоманды .ENTER, но и для макрокоманд .CDFN, .CMKT, .DELETE и целого ряда других. То же справедливо и для команды EMT 374. Помимо макрокоманды .CLOSE эта команда "обслуживает" еще и макрокоманды .CHAIN, .DATE, .HERR и многие другие. Макрокоманды, имеющие одинаковые EMT, различаются кодами, записанными в тексте их макроопределений. Так, макрокомандам .ENTER, .CDFN, .CMKT и .DELETE присвоены коды 2, 15, 23 и 0 соответственно; для макрокоманд .GLOSE, .CHAIN, .DATE и .HERR характерны коды 6, 10, 12 и 5.

Машинный код команды EMT может иметь значение от 104000_8 (команда EMT 0) до 104377_8 (команда EMT 377). EMT-процессор, получив из прикладной программы содержимое слова с командой EMT, анализирует значение его младшего байта и в зависимости от этого значения переходит на ту или иную ветвь, которые выполняются схожим образом. По таблицам адресов, находящимся в мониторе, и по значению кода макрокоманды EMT-процессор находит адрес программы монитора, соответствующей выполняемому программному запросу и передает ей управление вместе с адресом таблицы аргументов в программе пользователя (или с самими аргументами, если их немного). Активизированная программа монитора, выполнив затребованные действия (закрытие канала, вывод текста на терминал и т. д.), возвращает управление в EMT-процессор, где происходит восстановление содержимого всех РОН и выполняется команда RTI возврата из прерывания в прикладную программу.

Все описанные действия требуют заметного времени. Заполнение таблицы аргументов, сохранение и восстановление регистров, поиск адреса требуемой программы монитора, некоторые проверки, выполняемые монитором, занимают в зависимости от кода команды EMT и кода макрокоманды 50...80 машинных команд, на выполнение которых на ЭВМ СМ-4 уходит 150...250 мкс. Это время представляет собой си-

стемные издержки выполнения программных запросов. Следует подчеркнуть, что речь идет не о времени выполнения программ монитора, реализующих затребованные действия, а лишь о процессе перехода на эти программы и возврата из них.

Использование системных макрокоманд требует умения диагностировать их выполнение и анализировать возможные ошибки. Ошибка в макровызове (например, неправильно написанные аргументы) приведет к тому, что ОС не выполнит требуемые действия, однако система при этом обычно не выводит никаких диагностических сообщений, а программа продолжает выполняться, хотя ее дальнейшее выполнение, возможно, уже лишено смысла. Поэтому важно уметь определять, правильно ли выполнен каждый программный запрос.

Если при выполнении любого программного запроса обнаруживается ошибка, монитор устанавливает разряд *C* в РСП. Кроме того, код ошибки (обычно числа 0, 1 или 2) заносятся в специально выделенный для этого байт с абсолютным адресом 52_8. Диагностика выполнения программных запросов заключается, таким образом, в проверке состояния разряда *C* РСП и анализе байта 52, если разряд *C* оказался установлен.

Например, для программного запроса .ENTER, резервирующего на НМД место для файла и открывающего канал связи с ним, предусмотрено два кода ошибки: 0 – канал занят; 1 – на устройстве нет места для файла.

Программа с анализом правильности выполнения программного запроса .ENTER может иметь следующий вид:

```
FILE:     .RAD50 /DK FILE01DAT/
ARG:      .BLKW 3
MSG 1:    .ASCIZ " <16> "КАНАЛ ЗАНЯТ" <17>
MSG2:     .ASCIZ " <16> "УСТРОЙСТВО ЗАПОЛНЕНО" <17>
          .EVEN
START:    .ENTER # ARG,#0, #FILE, #8.
          BCS ERR

            ;Продолжение программы

ERR:      TSTB 52          ; Код ошибки 0?
          BEQ CHBUSY       ; Да, на вывод сообщения
          .PRINT #MSG2     ; Нет, значит, нет места для файла
EXIT:     .EXIT            ; Аварийный выход
CHBUSY:   .PRINT #MSG1     ; Канал занят
          BR EXIT          ; На аварийный выход
```

После макровызова .ENTER анализируется разряд *C* ССП. Если разряд *C* сброшен, программа продолжает выполняться. Если разряд *C* установлен, происходит переход на метку ERR и анализируется содержимое байта ошибки. В зависимости от кода ошибки (0 или не 0) выводится соответствующее аварийное сообщение и программа завершается

с помощью программного запроса .EXIT. Код 16, направляемый в регистр данных экрана терминала в начале строки с сообщением, переключает терминал в режим вывода русского текста. Код 17 возвращает терминал в режим латинского текста.

Если имеется возможность программного устранения ошибки, программа может попытаться это сделать и затем повторить запрос .ENTER. В частности, при нехватке места на диске можно удалить ненужные файлы (программным запросом .DELETE), а при занятости канала повторить запрос .ENTER для другого канала либо освободить занятый канал (запросами .CLOSE или .SAVESTATUS).

5.3. ПОДПРОГРАММЫ ЗАВЕРШЕНИЯ

Одной из наиболее характерных особенностей программ реального времени является насыщенность их параллельными процессами. Действительно, лишь в редких случаях программа реального времени выполняет в каждый конкретный момент только одну задачу, например прием данных из измерительной установки или обработку накопленной информации. Более типична ситуация, когда параллельно с накоплением данных идут такие процессы, как первичная обработка и визуализация поступающей информации; буферизация данных и вывод заполненных буферов на внешние ЗУ; отсчет времени экспозиции или периода измерений; ожидание команд оператора и их программная отработка. Существуют три основные концепции, лежащие в основе организации параллельных процессов: прерывания от ВУ, многозадачный режим и *подпрограммы завершения* (*подпрограммы обработки асинхронных системных прерываний*). Ниже описывается третье из перечисленных средств.

Подпрограмма завершения (ППЗ) – это программный фрагмент, написанный пользователем и служащий для программной отработки некоторого асинхронного внешнего события: окончания заданного интервала времени, завершения операции ввода-вывода, пересылки данных из задания в задание. Эти подпрограммы являются чрезвычайно мощным вычислительным средством, позволяющим выполнять временную синхронизацию измерительно-вычислительного процесса, управление программами реального времени, вывод контрольных сообщений и многое другое. Рассмотрим механизм активизации ППЗ на примере программного запроса .MRKT, служащего для отсчета заданного интервала времени. Отсчет ведется системным таймером, работающим на частоте сети (50 Гц). Структура программы, содержащей макровызов .MRKT, имеет следующий вид:

```
  ARG:    .BLKW 4    ; Область для аргументов
  TIME:   0,50.      ; Значение временного интервала
START:    .MRKT #ARG, #TIME, #COMP, #1

            ;; Продолжение основной программы
            :
```

Смещение	Содержимое
0	Время (старшие разряды)
2	Время (младшие разряды)
4	Указатель следующего элемента
6	Номер задания
10	Идентификатор запроса
12	0
14	Адрес ППЗ

Рис. 5.4. Элемент очереди временнóго запроса

```
COMP:
        .
        .; Подпрограмма завершения
        .
        RTS PC
```

Макровызов .MRKT содержит четыре аргумента. Первый, обычный для большинства макрокоманд является адресом области для аргументов. В данном случае эта область состоит из четырех слов. В качестве второго аргумента указывается адрес двухсловного блока, куда заносится требуемый интервал времени в единицах тактов системного таймера (1 такт = 1/50 с). В первом слове размещаются старшие разряды кода интервала, во втором слове – младшие. В приведенном выше фрагменте указан интервал 50 тактов (или 1 с). Третий аргумент представляет собой адрес ППЗ, а четвертый (он может быть опущен) является идентификатором данного запроса.

В организации обработки временнýх запросов большую роль играет *элемент очереди*, представляющий собой участок оперативной памяти длиной в семь слов, в котором в определенном порядке хранится вся необходимая для обработки временнóго запроса информация (рис. 5.4): отсчитываемый интервал времени, адрес следующего элемента, если запросов было несколько, или 0, если данный запрос единственный или последний, номер задания, выдавшего зарпос, идентификатор запроса и адрес ППЗ. Место для элемента очереди зарезервировано в системных полях данных, а заполняют и используют его соответствующие программы монитора. В процессе выполнения временнóго запроса монитор, получив управление посредством команды EMT, входящей в состав макрорасширения, заполняет элемент очереди данными, взятыми из аргументов макровызова (предварительно несколько видоизменив для удобства последующего анализа форму записи интервала времени), и передает адрес элемента программе обработки временнýх запросов, ставя тем самым этот элемент в *очередь временнýх запросов* (состоящую, впрочем, пока из одного элемента).

Каждый такт работы системного таймера эта программа сравнивает время, записанное в элементе очереди, с текущим временем, хранящимся (и наращиваемым) в специальных ячейках монитора. Когда сравне-

ние показывает, что заданный интервал времени истек, дальнейший анализ времени в этом элементе прекращается (он изымается из очереди временных запросов), а управление передается ППЗ, адрес которой берется из элемента очереди.

В результате взаимодействие монитора и прикладной программы протекает следующим образом. После выполнения программного запроса .MRKT основная программа продолжается, а монитор как бы начинает отсчитывать время (в действительности отсчет времени ведется непрерывно, но монитор начинает сравнивать текущее время с временем, записанным в данном элементе). По истечении заданного интервала времени монитор приостанавливает основную программу и передает управление ППЗ. Действия, выполняемые в ППЗ и длительность ее работы никак не ограничиваются. Завершается ППЗ командой RTS PC, по которой управление передается монитору, возобновляющему выполнение приостановленной основной программы.

Таким образом, концепция подпрограмм завершения дает возможность организовать параллельно с выполнением основной программы ожидание внешнего события (в данном случае – окончания временнóго интервала) и при наступлении этого события обработать его произвольным образом, прервав на это время выполнение основной программы. Полезно заметить, что прерывание основной программы происходит в неизвестной заранее точке, отчего такое прерывание и называется асинхронным.

Если основная программа выдает несколько временных запросов так, что отсчитываемые интервалы времени накладываются друг на друга, возникает очередь временных запросов, состоящая из нескольких элементов. Отдельные элементы связываются монитором в очередь путем указания в соответствующем слове каждого элемента (смещение 4) адреса следующего элемента. Последний элемент содержит в этом слове 0. Элементы располагаются монитором в очереди в определенном порядке: ближе к голове очереди стоят элементы, "срок действия" которых истекает раньше. По мере окончания отсчитываемых интервалов времени элементы выбывают из очереди и длина очереди сокращается.

Как уже отмечалось, монитор предоставляет заданию только один элемент очереди. Количество доступных элементов можно увеличить программным запросом .QSET, среди аргументов которого указывается адрес поля памяти в прикладной программе, отведенного под элементы очереди (по семь слов на каждый элемент):

```
QUE:    .BLKW 3*7          ; Место под три элемента
        .QSET #QUE, #3     ; Запрос к монитору
```

Монитор, выполняя программный запрос .QSET, образует связный список свободных элементов, в котором элементы связаны друг с другом, как и в очереди временных запросов, с помощью указателей следующих элементов.

Рассмотрим простой пример использования временны́х запросов и ППЗ. Пусть программа, запустив процесс измерений длительностью, скажем, 5 мин, должна за 30 с до окончания измерений выдать на терминал предупреждающее сообщение, сопровождаемое звуковым сигналом, а за 10 с до окончания – второе предупреждающее сообщение:

```
ARG:     .BLKW 4              ; Область для аргументов
TIME1:   0,15000.             ; 15000 тактов = 5 мин
TIME2:   0,15000.-1500.       ; 5 мин – 30 с
TIME3:   0,15000.-500.        ; 5 мин – 10 с
MSG2:    .ASCIZ <7><16>"ОСТАЛОСЬ 30 СЕКУНД!"<17>
MSG3:    .ASCIZ <7><16>"ОСТАЛОСЬ 10 СЕКУНД!"<17>
         .EVEN
QUE:     .BLKW 7*2            ; Место для двух элементов
START:   .QSET #QUE, #2       ; Образование элементов
         .MRKT #ARG, #TIME1, #COMP1
         .MRKT #ARG, #TIME2, #COMP2
         .MRKT #ARG, #TIME3, #COMP3
           .
           .; Продолжение основной программы
           .
COMP1:     .
           .; Отработка завершения измерений
           .
         RTS  PC              ; Возврат в основную программу
COMP2:   .PRINT #MSG2         ; Гудок, "ОСТАЛОСЬ 30 СЕКУНД!"
         RTS  PC              ; Возврат в основную программу
COMP3:   .PRINT #MSG3         ; Гудок, "ОСТАЛОСЬ 10 СЕКУНД!"
         RTS  PC              ; Возврат в основную программу
```

Поскольку в программе предполагается одновременная обработка трех временны́х запросов, требуется иметь три элемента очереди. Один элемент предоставляется монитором, два дополнительных создаются программным запросом .QSET. Далее в очередь к системному таймеру ставятся три временны́х запроса: на 5 мин, 5 мин – 30 с и 5 мин – 10 с. В аргументах запросов указаны адреса трех ППЗ. В последующих строках основной программы запускается процесс измерений (например, разблокируются входы измерительной аппаратуры); дальнейшие действия основной программы зависят от конкретной задачи.

Через 13500 с после запуска измерений завершается отсчет временно́го интервала, указанного в строке с меткой TIME2. Выполнение основной программы приостанавливается, и управление передается на ППЗ COMP2, в которой с помощью программного запроса .PRINT на экран терминала выводится текст "ОСТАЛОСЬ 30 СЕКУНД!". Первый символ строки текста с меткой MSG2 – число 7, является кодом звукового сигнала терминала. После выполнения команды RTS PC управление возвращается в прерванную программу. Точно так же еще через 20 с дается второй предупреждающий сигнал и выводится второе сообщение. На-

конец, через 5 мин после начала измерений активизируется ППЗ COMP1, которая, видимо, блокирует измерительную аппаратуру и выполняет другие завершающие действия, например переписывает накопленные данные на НМД.

В приведенном примере ППЗ выполнялись раздельно во времени, не накладываясь друг на друга. Часто, однако, возникает ситуация, когда несколько ППЗ должны быть активизированы почти одновременно. В таких случаях монитор организует очередь ППЗ, которая строится из тех же элементов, что и очередь временных запросов. После истечения заданного в программе интервала времени монитор преобразует освободившийся элемент очереди временных запросов в элемент очереди ППЗ, в котором заполнены только два слова: адрес следующего элемента, в данном случае 0, и адрес ППЗ. Последняя имеет более высокий приоритет для монитора, чем выполняемое задание, поэтому сразу же после окончания работы программы обработки временных запросов монитор запускает ППЗ, а элемент очереди ППЗ освобождается и возвращается в список свободных элементов. Если же во время выполнения ППЗ завершится следующий заданный в программе интервал времени, образованный монитором элемент очереди следующей ППЗ будет находиться в очереди ППЗ данного задания до тех пор, пока не завершится ППЗ, выполняемая в настоящий момент. После этого монитор сможет активизировать следующую ППЗ и освободить ее элемент очереди. Таким образом, при поступлении нескольких запросов на выполнение ППЗ они ставятся в очередь в порядке появления запросов и выполняются друг за другом. При этом важно понимать, что элемент очереди ППЗ освобождается сразу же после запуска данной ППЗ, в то время как элемент очереди временных запросов освобождается лишь после истечения заданного интервала времени.

Описанный механизм характерен для FB-монитора. В SJ-мониторе ППЗ не ставятся в очередь, а прерывают друг друга. Это отличие может существенно изменить функционирование программы реального времени при переносе ее из одной конфигурации ОС в другую.

Примеры использования ППЗ в программах реального времени будут приведены в последующих параграфах и главах.

5.4. ФОНОВО-ОПЕРАТИВНЫЙ РЕЖИМ

Для организации фоново-оперативного режима ВС должна работать под управлением FB-монитора РАФОС, который осуществляет загрузку фонового и оперативного заданий в отведенные для них области ОП и распределение между ними времени ЦП, а также предоставляет программисту набор средств для управления ходом выполнения заданий.

FB-монитор состоит из трех программных компонентов, расположение которых в ОП после загрузки ОС показано на рис. 5.5[1]

В старших адресах ОП непосредственно перед страницей ввода-вывода размещается драйвер *системного устройства*, с которого загружается система. Для мини-ЭВМ типа СМ-4 системным устройством обычно является накопитель на магнитном диске кассетного типа (НМД). Его драйвер занимает всего 120...140 слов. В микроЭВМ в качестве системного устройства используется накопитель на гибком магнитном диске (НГМД). Размер драйвера НГМД в зависимости от типа накопителя колеблется от 250 до 2000 слов. Поскольку функционирование ОС предполагает частое обращение к системному устройству, его драйвер в отличие от драйверов остальных ВУ является резидентным. Над драйвером системного устройства располагаются компоненты монитора.

Управление работой заданий пользователя осуществляется *резидентным монитором* RMON. Программа RMON является наиболее важной частью монитора. Она всегда находится в ОП в фиксированной области (является резидентной), и с ее помощью реализуется значительная часть функций ОС.

Для управления файлами предусмотрена *программа обслуживания пользователя* USR. В отличие от RMON, USR в процессе работы ВС может выгружаться из ОП и снова загружаться с диска по мере необходимости. Этим обеспечивается более эффективное использование ОП прикладными программами.

Взаимодействие оператора и системы РАФОС выполняется с помощью *интерактивного монитора* KMON, который так же, как и USR, является выгружаемой программой. Если в ОП нет ни фонового, ни оперативного заданий, то активной является программа KMON, которая ожидает команд с клавиатуры терминала. Одной из фукнций KMON является запуск заданий. При работе в однозадачном режиме запуск задания (фонового) осуществляется командой монитора RUN. Программа KMON, получив эту команду, загружает с диска в ОП затребованную программу и передает ей управление, после чего сама переходит в пассивное состоя-

---

[1] В технической документации корпорации DEC принято изображать память ЭВМ в перевернутом виде: начало памяти (меньшие, младшие адреса) располагаются внизу рисунка, конец памяти (большие, старшие адреса) – вверху. Видимо, причина этого носит этимологический характер: low address по-английски обозначает и "меньший адрес", и "низкий адрес". Перевернутое изображение ОП существенно затрудняет рассмотрение функционирования ЭВМ. Действительно, все программы располагаются на таких рисунках "вверх ногами" и выполняются снизу вверх, что совершенно неестественно и противоречит порядку программных строк в текстах тех же программ. Вообще более привычным является нумерация объектов (в данном случае слов ОП) сверху вниз, а не наоборот. Следует заметить, что многие американские авторы пользуются в своих книгах (даже посвященных тем же ЭВМ корпорации DEC) изображением памяти, начинающимся с меньших адресов. В настоящей книге принято именно такое обозначение.

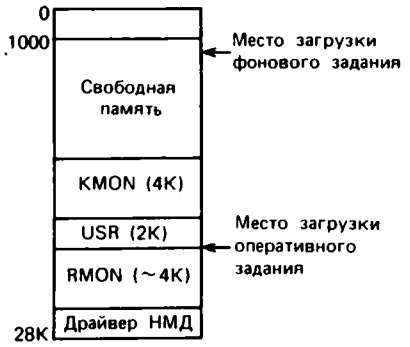


Рис. 5.5. Состояние оперативной памяти после загрузки FB-монитора

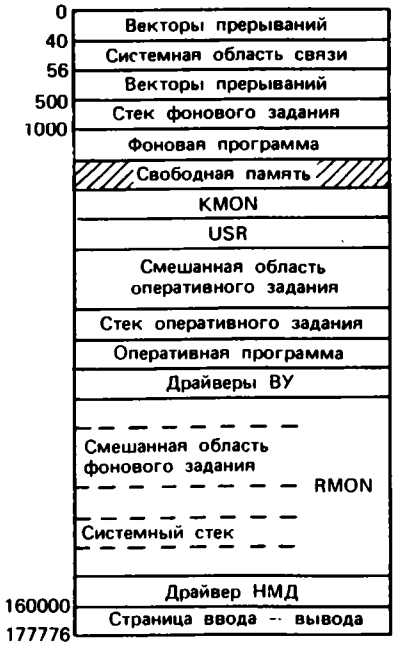


Рис. 5.6. Состояние оперативной памяти после загрузки заданий пользователя

ние, в вычислительном процессе не участвует и при нехватке памяти может быть удалена из ОП. Для активизации двухзадачного комплекса следует сначала командой FRUN запустить оперативное задание. Программа KMON, загрузив с диска оперативную программу и передав ей управление, остается в ОП в активном состоянии, что позволяет оператору продолжать взаимодействие с KMON, в частности, потребовать запуска фонового задания. Загрузив и запустив фоновое задание, программа KMON, как и ранее, переходит в пассивное состояние и может быть удалена из памяти. Однако после завершения фонового задания (естественным образом или специальной командой оператора) программа RMON сразу же загружает в ОП KMON, что дает возможность оператору запустить новое фоновое задание. Таким образом, работая в двухзадачном режиме, оператор может, не прерывая выполнения оперативного задания, активизировать по мере необходимости те или иные программы в качестве фонового задания.

Рассмотрим расположение заданий в ОП после их загрузки (рис. 5.6).

Фоновая программа в процессе ее компоновки системной программой LINK получает базовый абсолютный адрес 1000_8. При этом с адреса 1000 начинается собственно текст программы, т. е. ее программные строки и поля данных. Область адресов с 0 по 776_8 входит в абсолютную программную секцию с именем ABS. В ней располагаются *системная область связи* (ячейки $40–56_8$), *блок управления загрузкой*, начинающийся в

ячейке 360_8, и *стек фонового задания* (ячейки $500–776_8$, 96_{10} слов). Загрузочный файл X.SAV фонового задания (здесь X – произвольное имя файла, а SAV – его тип, называемый иначе *расширением имени файла*), созданный программой-компоновщиком LINK, хранится на диске. При активизации задания командой монитора RUN файл X.SAV загружается в ОП с самого ее начала, так что адреса, присвоенные строкам программы компоновщиком, совпадают с физическими адресами ячеек памяти, куда эти строки загружаются.

Нулевой блок файла X.SAV, который соответствует упомянутой выше абсолютной программной секции, содержит полезную информацию обычно лишь в словах $40–52_8$, куда компоновщик записывает характеристики задания, и $360–376_8$, где каждому блоку загрузочного файла, действительно содержащему данные, отвечает установленный бит. Остальные слова нулевого блока, в том числе соответствующие стеку задания, заполнены нулями. Между тем начало ОП отведено под векторы прерываний ВУ. Эта область частично заполняется монитором после загрузки его в ОП (векторы прерываний стандартных устройств), частично же она могла быть заполнена оперативной программой, если та содержала строки инициализации прерываний. Чтобы не разрушить векторы прерываний, монитор, загружая в ОП начало нулевого блока файла с программой, просматривает карту защиты памяти, расположенную в RMON, и не загружает те ячейки, которые защищены в связи с их использованием системой или оперативным заданием. Отсюда следует, что оперативная программа перед заполнением используемых ею векторов прерываний должна защитить их (с помощью программного запроса .PROTECT), иначе векторы будут разрушены при разгрузке фонового задания.

Остальные блоки файла X.SAV монитор загружает в ОП, руководствуясь блоком управления загрузкой (ячейки $360–376_8$).

После загрузки программы ее системная область связи содержит следующую информацию (адреса слов восьмеричные):

слово 40 – стартовый адрес программы;

слово 42 – начальное значение указателя стека (1000_8 для фонового задания);

слово 44 – слово состояния задания;

слово 46 – адрес загрузки USR;

слово 50 – максимальный адрес, принадлежащий заданию;

байт 52 – код ошибки при выполнении программных запросов;

байт 53 – статус завершения задания;

слово 54 – начальный адрес RMON;

слово 56 – информация для специальных терминалов.

Содержимое части ячеек системной области связи устанавливается системой и из программы может только читаться. В некоторые ячейки возможна и запись по ходу выполнения программы.

Описанный порядок загрузки в ОП фонового задания приводит к необходимости соблюдать определенные правила при обращении к начальным ячейкам ОП. Пусть, например, мы хотим обрабатывать прерывания от клавиатуры системного (командного) терминала, используя для этого вместо системного драйвера собственную ПОП, начинающуюся с метки DISP. Попытка заполнить вектор прерывания на стадиях компоновки и загрузки путем включения в текст программы строк

```
.ASECT
. = 60
.WORD DISP
.WORD 200
.PSECT
```

не приведет к желаемому результату, так как хотя значение DISP и число 200_8 будут содержаться в ячейках 60 и 62_8 нулевого блока файла X.SAV, монитор не загрузит эти ячейки ОП, так как они защищены системой. Инициализировать прерывания следует на стадии выполнения программы строками

```
MOV     #DISP, 60
MOV     #200, 62
```

Напротив, заполнять слово состояния задания удобно на стадии компоновки:

```
.ASECT
. = 44
.WORD 40000
.PSECT
```

В этом примере в слове состояния задания устанавливается разряд 14, что разрешает использование как прописных, так и строчных букв.

Загрузочный файл X.REL оперативного задания создается компоновщиком LINK при указании в команде его вызова ключа /FOREGROUND.

В этом случае компоновщик образует перемещаемый загрузочный модуль, который можно настроить на любой базовый адрес. Для этого в файл X.REL включен список строк программы, содержимое которых зависит от ее расположения в ОП. Нулевой блок файла содержит в словах $40–62_8$ характеристики задания.

При загрузке оперативного задания командой FRUN монитор извлекает из нулевого блока файла информацию о длине программы, смещает соответствующим образом KMON и USR и загружает программу в память так, чтобы ее старший адрес разместился вплотную к RMON (или драйверам ВУ, если они загружены). После этого монитор просматривает список слов, требующих настройки на конкретный базовый адрес и, определив константу настройки, изменяет содержимое этих слов в загруженной программе. Только после этого задание запускается.

Из рис. 5.6 видно, что непосредственно над оперативной программой располагается ее стек, размер которого составляет 64 слова. Таким образом, каждое задание имеет свой стек. Кроме стеков прикладных программ в составе RMON имеется еще область системного стека, который используется, когда монитор находится в системном состоянии.

Выше стека оперативного задания расположена его *смешанная область* – таблица размером около 140 слов, в которой содержатся различные данные, необходимые монитору для взаимодействия с заданием в ходе его выполнения. При загрузке оперативного задания система специально выделяет память под эту таблицу (смешанная область фонового задания расположена в RMON и дополнительной памяти при загрузке задания не требует). Что же касается системной области связи, то для нее всегда используются ячейки $40–56_8$. В зависимости от того, какое задание выполняется в настоящий момент, эти ячейки автоматически заполняются содержимым, относящимся к данному заданию.

Оперативное задание отличается от фонового способом загрузки в ОП и режимом выполнения. Однако в исходном тексте программы не заключено никаких указаний на предполагаемый режим ее выполнения. Соответствующие атрибуты придаются программе на этапе компоновки, когда создается либо файл X.SAV (для фонового задания), либо файл X.REL (для оперативного), а также в процессе активизации заданий командами оператора RUN или FRUN. Поэтому любую программу можно в принципе выполнять как в одном, так и в другом режимах, если подготовить для нее два загрузочных файла типов SAV и REL. Надо только иметь в виду, что возможности оперативного и фонового заданий несколько различаются. Так, в программе, предназначенной для выполнения в оперативном режиме, нельзя использовать макрокоманды .FETCH, .RELEASE, .CHAIN; оперативная программа не может накладываться в ОП на программы USR и KMON, и поэтому ее максимальный размер на 6К меньше, чем у фоновой программы; фоновая программа может расширяться в процессе выполнения в сторону больших адресов (динамическое распределение памяти), а оперативная "зажата" между RMON и USR и место для ее расширения нужно предусматривать заранее.

И оперативное, и фоновое задания, как правило, взаимодействуют с оператором через один и тот же терминал. Монитор обеспечивает идентификацию передаваемых сообщений, в результате чего оператор знает, какое задание к нему обращается и сам может передавать сообщение любому заданию по выбору. Делается это следующим образом.

Вывод сообщения из задания на экран терминала обычно осуществляется с помощью программного запроса .PRINT с указанием адреса выводимого сообщения. Если система работает в фоново-оперативном режиме и сообщение поступает из оперативного задания, оно предваряется выводом на экран терминала символов F>. Сообщение из фонового задания начинается символами B>. Монитор, выведя на экран F>, пере-

ключает и экран, и клавиатуру на работу с кольцевыми буферами ввода-вывода, находящимися в смешанной области оперативного задания (подробнее о буферах ввода-вывода см. гл. 8). Поэтому любое сообщение, введенное оператором с клавиатуры, попадает в кольцевой буфер оперативного задания и соответствующими программными запросами (.TTYIN или .TTINR) может быть изъято оттуда и обработано программой. Если же сообщение передало фоновое задание, монитор, выведя на экран B ⟩, переключает экран и клавиатуру на работу с кольцевыми буферами фонового задания, в результате чего сообщение, вводимое оператором, будет доступно фоновой программе.

Оператор имеет возможность переключать буфера ввода и вывода по своему усмотрению. Для связи с оперативным заданием надо на клавиатуре терминала набрать комбинацию ⟨CTRL/F ⟩ (т. е. одновременно нажать управляющую клавишу ⟨ CTRL ⟩ и клавишу ⟨ F ⟩). После этого сообщения, вводимые с клавиатуры, будут поступать в кольцевой буфер ввода оперативного задания. Для связи с фоновым заданием следует набрать ⟨ CTRL/B ⟩.

Как уже отмечалось, при работе в фоново-оперативном режиме в качестве фонового задания может выполняться любая программа, в том числе и не имеющая отношения к оперативной (например, редактор текста или транслятор). Однако больший интерес представляет случай, когда обе программы являются частью двухпрограммного комплекса реального времени. При разработке такого комплекса возникает необходимость организации взаимодействия оперативного и фонового заданий – передачи из программы в программу данных или служебной информации, временной синхронизации их работы и т. д. Рассмотрим некоторые примеры решения этих вопросов.

Пусть оперативное задание выполняет сбор, накопление и запись на диск некоторой измерительной информации. Предположим, что проводимый эксперимент требует непрерывного контроля за ходом накопления данных, для чего параллельно с оперативным заданием выполняется фоновое, в функции которого входит периодическая обработка накапливаемых данных по некоторому алгоритму, построение графического образа данных и вывод изображения на графический дисплей. Очевидно, что фоновое задание должно иметь доступ к данным, накапливаемым оперативным заданием.

Для пересылки данных из одного задания в другое в системе РАФОС предусмотрены три макрокоманды передачи данных .SDAT, .SDATC и .SDATW и три макрокоманды приема данных .RCVD, RCVDC и .RCVDW. Пересылка данных осуществляется специальной программой пересылки данных, входящей в состав RMON. Для пересылки данных передающее задание выполняет указанную в программе разновидность программного запроса .SDAT. Монитор, получив этот запрос, заполняет элемент очереди и ставит его в очередь к программе пересылки данных. При этом используется тот же элемент, с помощью которого организуются временные

запросы. Принимающее задание для получения данных должно выполнить какую-либо из разновидностей программного запроса .RCVD, что тоже приводит к заполнению элемента очереди и постановке его в очередь к той же программе (для каждого задания монитор предоставляет по одному элементу очереди). После этого программа пересылки данных осуществляет физическую передачу данных из буфера передающей программы в буфер принимающей. Элементы очереди заняты до окончания физической передачи данных, после чего они возвращаются в списки свободных элементов.

При использовании программных запросов .SDAT и .RCVD управление возвращается в программу сразу же после постановки элемента в очередь к драйверу. Программа продолжает свое выполнение (хотя данные еще не переданы); передача данных произойдет после получения монитором парного запроса из другого задания.

Запросы .SDATW и .RCVDW блокируют выполнение задания до окончания передачи данных; задание будет продолжено только после завершения передачи.

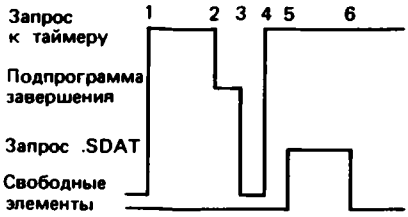
Наконец, запросы .SDATC и .RCVDC позволяют программно отработать завершение передачи: управление возвращается в программу сразу же после постановки элемента в очередь, но в момент завершения передачи основная программа прерывается и управление передается ППЗ, адрес которой указывается в числе аргументов программного запроса. После выполнения ППЗ (в которой, например, может быть проконтролирована правильность передачи или поставлен запрос на передачу следующей порции данных) управление возвращается в основную программу.

Вернемся к описанному выше примеру двухзадачного комплекса. Пусть объем данных, передаваемых из оперативной программы в фоновую, составляет 1024 слова, а темп накопления данных требует повторной передачи каждые 10 с.

Текст передающей (оперативной) программы будет иметь следующий вид:

```
DATAF: .BLKW   1024.                    ; Буфер с данными
ARG:   .BLKW   5                        ; Поле для аргументов
TIME:  0,500.                           ; 500 тактов = 10 с
Q:     .BLKW   7                        ; Дополнительный элемент очереди
START: .QSET   #Q, #1                   ; Образование элемента очереди
                                        ; ввода-вывода
       .MRKT   #ARG, #TIME, #COMP       ; Запрос на 10 с
         .
         .; Сбор и накопление данных
         .
       .EXIT
COMP:  .MRKT   #ARG, #TIME, #COMP       ; Запрос на 10 с
       .SDAT   #ARG, #DATAF, #1024.;    Запрос на пересылку
       RTS     PC
```

Рис. 5.7. Диаграмма использования элементов очереди в конкретном примере



Программа начинается с образования с помощью программного запроса .QSET одного дополнительного элемента очереди. Далее ставится запрос к системному таймеру на отсчет интервала времени, равного 10 с, после истечения которого управление будет передано ППЗ COMP. После этого программа приступает к сбору и накоплению данных. ППЗ COMP прежде всего устанавливает повторный запрос к таймеру с указанием самой себя в качестве ППЗ, после чего выполняет запрос .SDAT с указанием адреса буфера с данными и количества передаваемых слов. Таким образом, каждые 10 с сбор данных прерывается, накопленные к этому моменту данные из буфера DATAF пересылаются в буфер приемной задачи (точнее говоря, ставятся в очередь на пересылку) и сбор продолжается.

Рассмотрим диаграмму использования элементов очереди (рис. 5.7). Первый запрос к таймеру (момент *1*) использует один элемент очереди. По истечение 10 с этот элемент преобразуется в элемент очереди ППЗ (момент *2*), который после активизации ППЗ освобождается (момент *3*). ППЗ использует тот же элемент для постановки повторного запроса к таймеру (момент *4*), а запрос на передачу данных .SDAT требует еще одного элемента (момент *5*). Таким образом, начиная с момента *5*, используются два элемента очереди. Поскольку в мониторе предусмотрен лишь один элемент, недостающий элемент был создан в начале программы запросом .QSET. Заметим, что через некоторое время после выдачи запроса на пересылку данных произойдет их физическая передача (если фоновое задание выдаст запрос .RCVD), что приведет к освобождению одного элемента (момент 6).

Структура фоновой программы будет иметь следующий вид:

```
WCNT:   .BLKW   1
DATAB:  .BLKW   1024.
ARG:    .BLKW   5
START:  .RCVDW #ARG, #WCNT, #1024.
        BCS     END
          .
          .; Формирование и вывод изображения из буфера
          .
        BR      START
END:    .EXIT
```

Буфер данных принимающей программы начинается с метки DATAB. Однако в процессе пересылки данных первое слово приемного буфера заполняется числом переданных слов. Поэтому в аргументе запроса .RCVDW указан адрес буфера WCNT общим размером 1025 слов (1 + + 1024), а программа формирования изображения работает с массивом, начинающимся со второго слова этого буфера, т. е. с метки DATAB. Использование в программе синхронной формы запроса .RCVDW гарантирует, что формирование нового изображения начинается только после получения данных.

Программный запрос .RCVDW (как и другие разновидности этой макрокоманды) устанавливает бит С в ССП, если в системе отсутствует другое задание. В рассматриваемом примере это может случиться, если программа накопления, выполняемая в качестве оперативного задания, по какой-либо причине завершилась, а фоновая программа вывода изображения продолжает выполняться. Для отработки такой ситуации в программе после запроса .RCVDW включена строка BCS END, которая аварийно завершает фоновое задание, если перед этим завершилось оперативное.

Приведенные программы составлены в предположении, что цикл формирования данных и вывода изображения длится менее 10 с. Можно, однако, предположить, что время обработки принятого массива и формирования графического образа не является постоянным, а зависит от реальной структуры данных (количества пиков, числа отсчетов и т. д.). Если окажется, что к моменту выдачи следующего запроса предыдущий запрос не будет снят из-за того, что фоновая программа еще обрабатывает предыдущую порцию данных и не дошла до выполнения запроса .RCVDW, произойдет нарушение работы программы сбора, так как она прекратит выполнение в ожидании освобождения элемента очереди. Чтобы избежать этого, можно перед выдачей запроса .SDAT анализировать состояние очереди запросов, и если эта очередь не пуста, пропускать один цикл передачи данных.

Анализ состояния очереди запросов можно выполнять разными способами. Один из них — проверка содержимого заголовка списка свободных элементов очереди, находящегося во втором слове смешанной области задания. Если в списке есть свободные элементы, в заголовке списка содержится адрес первого элемента; если все элементы очереди заняты, в заголовок списка записывается (монитором) 0.

Как уже отмечалось, смешанная область фонового задания входит в RMON. Адрес ее заранее неизвестен, тем более что конкретные адреса тех или иных составляющих RMON, как и его начальный адрес, зависят от условий, поставленных при генерации ОС, а также от размеров драйвера системного устройства, который, будучи загружен в ОП, смещает в большей или меньшей степени места загрузки остальных компонентов ОС в сторону меньших адресов. Для того чтобы пользователь мог работать с системными таблицами, в состав RMON включена область, ячейки

которой, содержащие информацию о системе, всегда имеют фиксированное положение относительно начала RMON. С другой стороны, начальный адрес RMON хранится в ячейке 54_8 системной области связи. Таким образом, получив из ячейки 54_8 адрес RMON и прибавив к нему требуемое смещение, можно обратиться к любой ячейке с фиксированным смещением. Содержимое этих ячеек описано в документации ОС; список наиболее важных ячеек приведен в табл. 5.1. В частности, в ячейку со смещением 320 монитор заносит адрес смешанной области выполняющегося в настоящий момент задания.

Смешанная область задания содержит данные, используемые системой для организации вычислительного процесса. В некоторых случаях, однако, к ним приходится обращаться из прикладной программы. Список наиболее интересных для прикладного программиста ячеек приведен в табл. 5.2.

Отсутствие конкретных смещений последних трех областей говорит о том, что местоположение их относительно начала смешанной области может изменяться в зависимости от условий генерации монитора.

Хотя, как указывалось выше, к ячейкам с фиксированными смещениями и к смешанной области можно обратиться "на физическом уровне", получив адрес RMON из ячейки 54_8, более корректный способ заключается в использовании специально для этого предназначенных системных макрокоманд .GVAL и .PVAL. Первая служит для чтения содержимого ячеек с фиксированными смещениями, вторая – для их модификации. В качестве аргумента макровызова .GVAL указывается фиксированное смещение ячейки, содержимое которой требуется получить в программу; искомое содержимое монитор возвращает в R0. Следует заметить, что система РАФОС не защищает от прикладных программ системные области (отражением чего и является наличие макрокоманды .PVAL). Хотя обращение к таблицам монитора является распространенной практикой, пользоваться этим средством надо с большой осторожностью, так как неправильное вмешательство в работу монитора в лучшем случае разрушит систему, а в худшем – приведет к продолжению выполнения задания с непредсказуемыми результатами.

Вернемся к нашему примеру двухзадачного комплекса с передачей данных и рассмотрим вариант оперативной программы, в котором перед выдачей очередного программного запроса .SDAT анализируется заголовок списка свободных элементов и, если этот список пуст (что приведет при попытке выполнить запрос .SDAT к блокировке программы), выдача очередного запроса .SDAT откладывается до следующего цикла передачи, т. е. на 10 с:

```
; Оперативная программа
        .QSET     #Q, #1
        .MRKT     #ARG, #TIME, #COMP
          :
          :; Сбор и накопление данных
```

Таблица 5.1. **Содержимое ячеек с фиксированными смещениями**

Смещение (восьмеричное)	Символическое обозначение	Описание
0	$RMON	Точка входа в монитор в случае прерывания. Содержит команду JMP $ INTEN
4	$ CSW	Область каналов фонового задания (16 каналов, всего 80 слов)
256	BLKEY	Номер сегмента каталога НМД, находящегося в настоящий момент в ОП
260	CHKEY	Смещение в таблицах монитора к информации о драйвере устройства и номер устройства, для которого в ОП находится сегмент каталога
262	$DATE	Текущая дата
264	DFLAG	Флаг "Идет обращение к каталогу". Запрещает завершение задания вводом ⟨CTRL/C⟩ в процессе работы с каталогом
266	$ USRLC	Адрес обычной загрузки USR, определяемый монитором, если содержимое ячейки 46_8 равно 0
270	QCOMP	Адрес подпрограммы завершения операции ввода-вывода для всех системных драйверов. Делается известным драйверу с помощью макрокоманды .DRFIN
274	SYUNIT	Номер устройства (SY:), с которого была загружена система
300	CONFIG	Слово конфигурации системы
304	TTKS	Адрес РКС клавиатуры консольного терминала (обычно 177560_8)
306	TTKB	Адрес РД клавиатуры консольного терминала (обычно 177562_8)
310	TTPS	Адрес РКС экрана консольного терминала (обычно 177564_8)
312	TTPB	Адрес РД экрана консольного терминала (обычно 177566_8)
320	CNTXT	Адрес смещанной области выполняемого задания
322	JOBNUM	Номер выполняемого задания, указывающий, является оно фоновым, оперативным или системным
324	SYNCH	Адрес программы монитора, обрабатывающей программный запрос .SYNCH
326	LOWMAP	Карта защиты векторов прерываний, защищает ячейки $0 - 476_8$
360	$MTPS	Точка входа в монитор при выполнении программного запроса .MTPS
362	$MFPS	Точка входа в монитор при выполнении программного запроса .MFPS
364	SYINDX	Смещение в таблицах монитора к информации о драйвере системного устройства
370	CONFG2	Расширение слова конфигурации системы
374	USRARE	Размер программы USR в байтах
404	PNPTR	Смещение к таблице $PNAME с именами драйверов от начала RMON
420	$ MEMSZ	Объем доступной оперативной памяти в блоках по 32 слова

Таблица 5.2. **Содержимое ячеек смешанной области**

Смещение	Символическое обозначение	Описание
0	L.STATE	Слово состояния задания (отличное по содержанию от слова состояния в ячейке 44)
2	L.QHDR	Заголовок списка свободных элементов
4	L.CMPE	Адрес последнего элемента очереди ППЗ
6	L.CMPL	Заголовок очереди ППЗ
10	L. CHWT	Адрес канала ввода-вывода, когда задание находится в ожидании завершения операции ввода-вывода по этому каналу
12	L.PCHW	Ячейка для хранения L.CHWT во время выполнения ППЗ
14	L.PERR	Ячейка для хранения байтов 52 и 53 системной области связи во время выполнения ППЗ
24	L.JNUM	Номер данного задания
26	L.CNUM	Число каналов ввода-вывода (по умолчанию 16)
30	L.CSW	Адрес области каналов
32	L.IOCT	Число выполняемых операций ввода-вывода
34	L.SCTR	Счетчик блокированя. Отрицательное значение указывает на то, что задание находится в заблокированном состоянии
–	L.NAME	Имя и тип файла выполняемого задания
–	L.SP	Ячейка для хранения указателя стека
–	L.QUE	Единственный элемент очереди

```
        .EXIT
; Подпрограмма завершения
COMP:.MRKT   #ARG, #TIME, #COMP ; Запрос на 10 с
     .GVAL   #ARG, #320         ; Получение адреса смешанной
                                ; области
     TST     2 (R0)             ; Список свободных элементов
                                ; пуст?
     BEQ     FIN                ; Да, пропустить выдачу
     .SDAT   #ARG, #DATAF, #1024,; Запрос на пересылку
     RTS     PC                 ; Нормальное завершение
FIN: .PRINT  #MSG1              ; Выдача сообщения о пропуске
     RTS     PC                 ; цикла
MSG1:.ASCIZ  <16> "ОБРАБОТКА ЗАТЯНУЛАСЬ, ПРОПУЩЕН ЦИКЛ" <17>
```

После выполнения программного запроса .GVAL в начале ППЗ в R0 оказывается адрес смешанной области задания (в данном случае – оперативного). Команда TST 2(R0) анализирует второе слово смешанной области – заголовок списка свободных элементов. Если список пуст (в заголовке записан 0), один цикл выдачи запроса .SDAT пропускается, а на экран терминала выводится соответствующее сообщение.

Следует заметить, что для успешной работы описанных программ оперативное задание не должно занимать все время процессора и, следовательно, сбор данных должен выполняться в режиме прерываний. Если сбор данных осуществлять в режиме ожидания готовности, цикл

ожидания готовности целиком займет время ЦП, и фоновое задание попросту не будет выполняться.

В программных комплексах, обладающих высокой степенью интеллектуальности, программа сама назначает имена создаваемым ею файлам (имя может образовываться из текущей даты, например MAY15.DAT, или из фамилии пациента, в результате осмотра или исследования которого создан файл с данными, например IVANOV.001, и т. д.). В этом случае при работе в фоново-оперативном режиме может возникнуть необходимость передачи из оперативного задания в фоновое атрибутов файла. Это можно сделать с помощью программного запроса .CHCOPY, который создает канал, копируя в него информацию из указанного канала другого задания, открывая тем самым связь принимающего задания с тем же файлом.

Пусть оперативное задание программно определяет не только имя файла (включая, возможно, и номер диска, на котором есть свободное место), но и номер канала связи с этим файлом. Имя файла в коде RADIX-50 записывается в ячейку FILNAM, а номер канала в виде двоичного числа – в ячейку CHNUMB. Для копирования канала фоновым заданием в него надо передать номер канала. Это можно сделать программным запросом .SDAT. Соответствующие фрагменты программ будут выглядеть следующим образом:

```
                                          ; Оперативная
                                          ; программа
FILNAM:  .BLKW    4                       ; Блок описания
CHNUMB:  .BLKW    1                       ; файла
                                          ; Номер канала
ARG:     .BLKW    5                       ; Область для аргумен-
                                          ; тов
BUF:     .BLKW    256.                    ; Буфер с данными
          .                               ; Определение номера
          .                               ; канала и имени файла
          .
         .ENTER   #ARG, CHNUMB, #FILNAM, #1;  ; Создание файла
         .WRITW   #ARG, CHNUMB, #BUF, #256, #0 ; Запись в файл
         .SDATW   #ARG, #CHNUMB, #1       ; Передача номера канала
                                          ; Фоновая программа
CH:      .BLKW    2                       ; Приемный буфер для
                                          ; номера канала
ARG:     .BLKW    5                       ; Область для
                                          ; аргументов
BUF:     .BLKW    256.                    ; Буфер для
                                          ; содержимого файла
          .
          .
          .
         .RCVDW   #ARG, #CH, #1           ; Прием номера канала
         .CHCOPY  #ARG, #0, CH+2          ; Создание канала 0
         .READW   #ARG, #0, #BUF, #256., #0 ; Чтение файла
```

Следует обратить внимание на способы адресации, использованные в макрокомандах .ENTER и .SDATW. Если бы на стадии составления программы номер канала был известен, второй аргумент макрокоманды .ENTER имел бы непосредственную адресацию (например, #0). В данном случае номер канала хранится в ячейке с меткой CHNUMB, поэтому используется относительная адресация (CHNUMB). Во втором аргументе макрокоманды .SDATW должен указываться адрес передаваемого сообщения; этот адрес известен (ячейка CHNUMB), и аргумент записан с непосредственной адресацией (# CHNUMB).

Приведенный пример одновременно иллюстрирует возможности передачи данных из задания в задание через файл на НМД; оба задания могут использовать этот файл для записи, модификации или чтения.

5.5. ИСПОЛЬЗОВАНИЕ РАСШИРЕННОЙ ПАМЯТИ

Оснащение ЭВМ расширенной памятью требует, во-первых, включения в состав процессора специальной аппаратуры – диспетчера памяти, осуществляющего физический доступ к расширенной памяти, а во-вторых, – наличия программ управления диспетчером памяти. Хотя в принципе прикладной программист может написать эти программы сам, чаще всего для работы с расширенной памятью используются средства ОС. Состав и возможности этих средств в различных ОС неодинаковы. Так, в системе ОС РВ директивы (макрокоманды) управления памятью дают возможность:

расширения адресного пространства задачи за пределы 32К слов;

передачи массивов данных из задачи в задачу, если задачи образуют связный программный комплекс;

использования несколькими задачами общих областей памяти, в которые помещаются, например, библиотеки универсальных подпрограмм.

Возможности системы РАФОС скромнее. Фактически средства ХМ-монитора позволяют лишь довести объем адресного пространства каждого из заданий до 32К, а логического – до 96К. Рассмотрим использование макрокоманд управления расширенной памятью ХМ-монитора РАФОС.

Для работы с расширенной памятью существенным являются понятия *виртуальных* и *физических адресов*. Виртуальные адреса – это некоторые условные адреса, назначаемые строкам загрузочного модуля компоновщиком. Виртуальный адрес может иметь значение от 0 до $177\,776_8$ (32К), хотя виртуальное пространство задания обычно занимает только часть этого диапазона. При этом часть виртуального адресного пространства может принадлежать собственно программным строкам и данным, а часть представлять собой адреса служебных областей или системных программ.

Рассмотрим для примера виртуальное адресное пространство фонового задания, работающего под управлением FB-монитора (рис. 5.8).

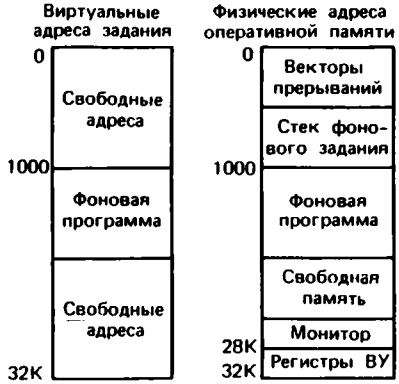


Рис. 5.8. Соотношение виртуальных и физических адресов фонового задания при работе с FB-монитором

Компоновщик всегда назначает строкам оттранслированной фоновой программы адреса начиная с 1000_8. Если, например, программа начиналась с однословного поля данных, обозначенного меткой START, то после трансляции это поле получит относительный адрес 0, а после компоновки – адрес 1000_8. Все обращения к метке START после трансляции преобразуются в ссылки на адрес 0, а после компоновки – на адрес 1000_8. Например, команда MOV #START, R0 будет преобразовываться следующим образом:

После трансляции	После компоновки
012 700	012 700
000 000	001 000

Во второй строке кода команды размещается сначала относительный (000000), а затем виртуальный (001000) адрес.

Таким образом, виртуальное пространство текста программы начинается с адреса 1000_8 и простирается до конца программы, за которым располагаются свободные, неиспользуемые виртуальные адреса. Однако программа может обращаться и к виртуальным адресам, выходящим за рамки собственно программных строк или строк данных. Так, строки программы MOV #ISR, @#300 (или просто MOV #ISR, 300) и TSTB @#172540 (или TSTB 172540) содержат обращения к виртуальным адресам 300_8 и 172540_8, не относящимся непосредственно к тексту программы. Однако эти адреса включаются в виртуальное адресное пространство задания. Точно так же программа может обратиться и к адресам, помеченным на рис. 5.8 как свободные.

При загрузке фоновой программы в память она занимает физические адреса, совпадающие с виртуальными (см. рис. 5.8). При этом по физическим адресам от 1000_8 и далее располагается текст программы, а часть физических адресов отводится под монитор, векторы прерываний и другие системные области. Поскольку программе доступно все виртуальное адресное пространство от 0 до 32К, программа может работать не только с собственными полями данных, но и таблицами монитора,

векторами прерываний, регистрами ВУ, стеком. Для обращения ко всем этим компонентам (кроме стека) надо знать их физические адреса, которые и указываются в программных строках.

Загрузка оперативной программы отличается тем, что ее начальный виртуальный адрес заранее не известен и определяется только в процессе загрузки исходя из размеров программы. Однако так же, как и для фоновой программы, виртуальные адреса загруженной оперативной программы совпадают с физическими адресами, и программе доступны все 32К виртуального пространства, а следовательно, и вся ОП плюс регистры ВУ. Совпадение виртуальных и физических адресов в ВС с 28К памяти является причиной того, что при описании таких систем понятие виртуальных адресов обычно не используется.

В системах с расширенной памятью ситуация отличается. Программа – вся целиком или по крайней мере ее часть – размещается в ОП так, что виртуальные адреса, назначенные строками программы компоновщиком, не совпадают с физическими адресами, по которым эти строки располагаются в ОП. Получается, что в ОП загружается как бы неправильная программа – записанные в ней (виртуальные) адреса не соответствуют тем физическим, по которым надо обращаться в процессе ее выполнения. В функции диспетчера памяти как раз и входит преобразование виртуальных адресов каждой программы в физические адреса ОП. Практически это сводится к формированию определенного смещения, добавляемого ко всем виртуальным адресам данной программы в процессе выполнения программных строк. При этом разные участки программы могут характеризоваться разными смещениями. Это приводит к загрузке программы в несмежные, разрывные участки ОП. Процесс преобразования виртуальных адресов в физические носит название *отображения*.

Отображение виртуальных адресов на физические производится с помощью специального набора 32-разрядных регистров диспетчера памяти, носящих название *регистров активных страниц* (РАС). Каждый РАС состоит из *регистра адреса страницы*, куда, собственно, и заносится смещение, определяющее отображение данной страницы, и *регистра описания страницы*, содержащего характеристики страниц (вид доступа, длину страницы и т. д.).

Виртуальной страницей называется область виртуального адресного пространства размером 4К слов или менее. Страницы состоят из блоков длиной 32 слова каждый, поэтому минимальный размер страницы составляет 32 слова. Виртуальные страницы начинаются с адресов, кратных 4К: 0, 4К, 8К, 12К и т. д. до 28К. Полное виртуальное пространство программы, таким образом, состоит из 8 страниц по 4К слов каждая. За каждой виртуальной страницей однозначно закреплен один из РАС: РАС 0 всегда отображает нулевую виртуальную страницу (с виртуальным адресом 0), РАС 1 – первую страницу (с виртуальным адресом 4К, или 20000_8) и т. д. (рис. 5.9). Отображение всегда происходит целыми страницами. Если размер страницы меньше 4К, то часть виртуального про-

Рис. 5.9. Отображение виртуального пространства на физическое

странства до начала следующей страницы не может быть отображена на физическую память, и обращение из программы по этим виртуальным адресам не имеет смысла: диспетчер памяти "не знает", что делать с неотображенными адресами и не ставит им в соответствие каких-либо физических адресов. Программа, содержащая укороченные виртуальные страницы, теряет часть своих виртуальных адресов, и, следовательно, ее реальный максимальный размер уменьшается.

Процесс формирования физического адреса из виртуального изображен на рис. 5.10. Для каждой виртуальной страницы диспетчер памяти определяет требуемое значение смещения и записывает его в соответствующий РАС (конкретно в регистр адреса страницы) в виде числа блоков по 32 слова. В 16-разрядном виртуальном адресе выделяются три поля. Старше три разряда (13 ... 15) адреса определяют номер адресуемой виртуальной страницы и соответственно номер РАС. В следующих семи разрядах (с 6 по 12) записан номер блока относительно начала страницы, в котором находится конкретный адрес. Наконец, младшие разряды адреса (0...5) указывают номер адресуемого слова (в последнем блоке). Младшие шесть разрядов виртуального адреса переходят в физический без изменения. Номер блока складывается со смещением, взятым из РАС, образуя 12-разрядный номер физи-

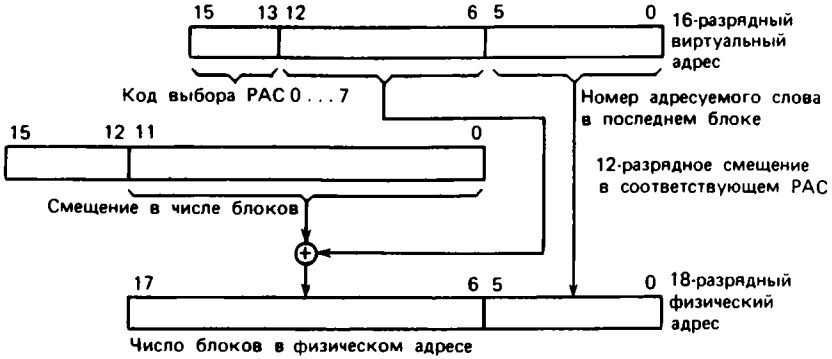


Рис. 5.10. Процесс формирования физического адреса в диспетчере памяти

ческого блока ОП. Таким образом, каждая виртуальная страница позиционируется в физической памяти на границе 32 слов, что обеспечивает минимальные потери физической памяти.

В процессе выполнения любой программы, и особенно программы реального времени, обычно насыщенной программными запросами к монитору, происходит частое переключение на выполнение внутренних программ монитора. Для защиты монитора от неправомерных действий программы пользователя в системах с расширенной памятью предусматриваются два режима работы процессора: *режим ядра* и *режим пользователя.* Каждый из этих режимов характеризуется использованием своего указателя стека и своего набора регистров активных страниц. При переключении режима происходит и изменение отображения; одни и те же виртуальные адреса в режиме ядра и режиме пользователя могут отображаться на разные участки физической памяти: в первом случае на программы монитора, во втором – на программу пользователя.

Текущий режим работы процессора фиксируется в разрядах 14 и 15 РСП (00 – режим ядра, 11 – режим пользователя); в разрядах 12 и 13 РСП фиксируется режим, существовавший до последнего прерывания.

В режиме ядра выполняются программы RMON и USR, а также программы обработки прерываний и программы драйверов ВУ. Программы пользователя, а также KMON и системные утилиты выполняются в режиме пользователя.

Действующее по умолчанию отображение виртуальных адресов (одинаковое для обоих режимов процессора) соответствует рис. 5.9. Первые семь виртуальных страниц отображаются на начало ОП; для этой части программы виртуальные и физические адреса совпадают. Последняя виртуальная страница (виртуальные адреса $160000...177776_8$) отображается на страницу ввода-вывода (адреса регистров внешних устройств). Таким образом, программа может использовать 28К памяти (за вычетом пространства, занятого монитором и другими системными компонентами) и имеет доступ к регистрам ВУ. Доступа к расширенной памяти программа не имеет. Если не считать изменения физических адресов ВУ (760000_8 вместо 160000_8 и т. д.) такое отображение не отличается от отображения в системе без диспетчера памяти (ср. с рис. 5.8).

Для того чтобы изменить отображение, действующее по умолчанию, программа должна сообщить монитору, сколько физической памяти ей требуется, описать области виртуальных адресов, требующих отображения, и дать указание монитору выполнить отображение виртуальных адресов на физические. Все это выполняется с помощью программных запросов к монитору. Заметим, что выделение конкретных физических адресов ОП осуществляет монитор. В программе указывается только объем требуемой памяти, но не ее местоположение.

Для работы с макрокомандами управления памятью необходимо освоить понятия *окон* и *районов* (*регионов*).

Виртуальное адресное окно представляет собой связную область виртуальных адресов, отображаемую на связную область памяти. Окно может включать одну или несколько виртуальных страниц. Минимальный размер окна соответствует минимальному размеру страницы и составляет 32 слова, максимальный соответствует восьми полным страницам и составляет 32К слов. Так же, как и виртуальные страницы, виртуальные окна должны начинаться на границе 4К слов. Таким образом, каждая программа содержит от 1 до 8 окон, причем количество окон определяется количеством раздельных областей памяти, на которые отображается программа. При создании окон монитор присваивает им идентификаторы.

Области, на которые отображаются окна, называются районами. Каждый район имеет идентификатор, который определяется при образовании района. Район включает в себя целое число блоков по 32 слова; максимальный размер района составляет 96К. Программа может использовать не более четырех районов одновременно. Сумма всех районов, с которыми работает данная программа, образует ее *логическое адресное пространство.* К сожалению, ХМ-монитор не допускает использование одного и того же района двумя заданиями одновременно. Это исключает передачу данных из задания в задание через расширенную память, как это делается в системе ОС РВ.

В ХМ-мониторе предусмотрено два способа отображения прикладных программ: *виртуальное* и *привилегированное.* При любом способе отображения программа может быть как оперативной, так и фоновой.

Фоновая виртуальная программа загружается в ОП начиная с адреса 500_8 и может занимать всю свободную память вплоть до начала USR. Таким образом, максимальный размер программы составляет приблизительно 21К. Однако программа может использовать и оставшиеся виртуальные адреса (до 32К). Для этого программа должна создать в расширенной памяти один или несколько районов, а в своем пространстве виртуальных адресов – одно или несколько окон. Выполнив затем отображение окон на районы, программа получает доступ к расширенной памяти и тем самым может увеличить свой размер (рис. 5.11). Однако ни при каких обстоятельствах программа не имеет непосредственного доступа к векторам прерывания, монитору и странице ввода-вывода. Это значит, что обращение к внешним устройствам как стандартным, так и нестандартным, например измерительной аппаратуре, осуществляется исключительно через драйверы с помощью системных макрокоманд ввода-вывода. Поэтому использование виртуальных программ требует обязательного написания драйверов для всего программно-управляемого оборудования.

Программа объявляется виртуальной установкой разряда 10 слова состояния задания (ячейка 44_8). Поскольку, как это видно из рис. 5.11, программа в процессе выполнения не может обратиться к этой ячейке (ей не соответствует никакой виртуальный адрес программы), запол-

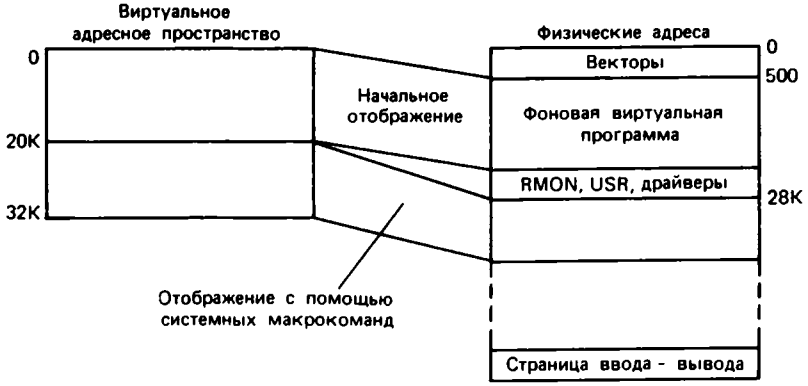


Рис. 5.11. Отображение фоновой виртуальной программы

нение слова состояния задания следует выполнять на стадии загрузки программы в ОП, для чего в программе предусматриваются строки:

```
.ASECT      ; Абсолютная секция, физический адрес 0
.= 44       ; Смещение к физическому адресу 44
.WORD 2000  ; Установка разряда 10
.PSECT      ; Переход к виртуальному пространству
```

Оперативная виртуальная программа загружается, как и обычная оперативная программа, между RMON и USR, причем USR и KMON смещаются в сторону меньших адресов на размер программы. Максимальный размер программы – около 17К, и она, как и фоновая, не имеет доступа к векторам прерывания, монитору и регистрам ВУ. Однако с помощью макрокоманд управления расширенной памятью программа может использовать оставшиеся виртуальные адреса (вплоть до 32К), отобразив эти адреса на расширенную память.

Отображение привилегированных программ (как фоновой, так и оперативной) в точности соответствует отображению при работе с SJ- или FB-мониторами (на рис. 5.12 показано отображение фоновой привилегированной программы). Программа имеет доступ к векторам прерываний, таблицам монитора и регистрам ВУ, поэтому из такой программы возможно программирование ВУ на физическом уровне путем непосредственного обращения к регистрам ВУ. С другой стороны, значительная часть виртуального пространства программы (около 11К) используется для отображения на системные области, что уменьшает максимальный размер программы. Однако при необходимости эту часть виртуального пространства можно с помощью системных макрокоманд отобразить на расширенную память, сняв отображение на системные области. Естественно, что пока действует отображение на расширенную память, программа не может выполнять программирование ВУ на физическом уровне.

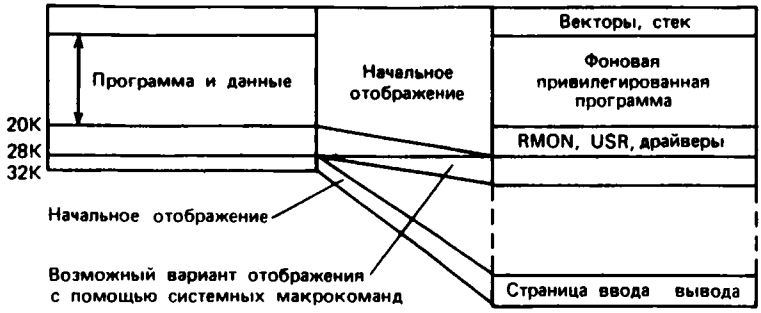


Рис. 5.12. Отображение фоновой привилегированной программы

Кроме того, следует иметь в виду, что такое изменение отображения недопустимо, если активизирован механизм прерываний от ВУ. Другая возможность использования расширенной памяти привилегированной программой – отображение на расширенную память какой-то части виртуальных адресов, принадлежащих самой программе и отображаемых по умолчанию не на системные области, а на физическую ОП. В этом случае используемое программой виртуальное пространство не увеличивается, но появляется возможность, образовав в расширенной памяти несколько районов, последовательно отображать на них один и тот же участок виртуального пространства, увеличив объем используемой физической памяти (такая методика возможна и для виртуальных программ). Пример этого будет приведен ниже.

Характерным для привилигированных программ является автоматическое возвращение к начальному отображению после снятия отображения на расширенную память. В виртуальных программах этого не происходит.

Монитор воспринимает задание как привилегированное, если разряд 10 его слова состояния содержит 0. Поскольку компоновщик заполняет этот разряд нулем, задание, запускаемое под управлением ХМ-монитора, по умолчанию объявляется привилегированным.

Для работы с макрокомандами управления памятью в программе необходимо создать специальные структуры данных: *блок описания района* (БОР) и *блок описания окна* (БОО). Эти блоки располагаются в поле данных программы и снабжаются метками. Указание соответствующей метки в поле параметров используемого программного запроса управления памятью позволяет передавать ему всю информацию, содержащуюся в блоке описания. Поскольку БОР и БОО находятся в поле данных прикладной программы, их содержимое может по мере необходимости уточняться и модифицироваться.

Формат БОР приведен на рис. 5.13. Блок состоит из трех полей, которым присвоены определенные символические смещения. Смещения

R.GID	Идентификатор района
R.GSIZ	Размер района
R.GSTS	Статус района

Рис. 5.13. Формат блока описания района

W.NAPR	Базовый РАС	Идентификатор окна	W.NID
W.NBAS	Виртуальный базовый адрес окна		
W.NSIZ	Размер окна		
W.NRID	Идентификатор района		
W.NOFF	Смещение		
W.NLEN	Размер области отображения		
W.NSTS	Статус окна		

Рис. 5.14. Формат блока описания окна

используются в тех случаях, когда надо работать не целиком с блоком, а с отдельными его полями. Пример этого будет приведен далее.

Блок описания района создается обычно с помощью макрокоманды .RDBBK, в поле параметров которой указывается требуемый размер создаваемого района в количестве 32-словных блоков:

REGION: .RDBBK 128. ; Размер района 4К

Указанной макрокомандой по метке REGION резервируются три слова для БОР и во второе слово записывается число 128. Остальные поля БОР будут заполняться монитором. В слово R.GID монитор поместит идентификатор созданного района, а в слово статуса района (R.GSTS) – диагностическую информацию. Установка разряда 15 этого слова говорит об успешном создании района; разряда 14 – о снятии отображения одного или нескольких окон при удалении данного района программным запросом .ELRG. Состояние диагностических разрядов может считываться программой для контроля правильности ее выполнения.

Формат БОО приведен на рис. 5.14. Блок создается с помощью макрокоманды .WDBBK, которая резервирует место для блока и частично заполняет его информацией. Эта макрокоманда имеет следующий формат:

.WDBBK apr, siz, rid, off, len, sts,

где apr – номер базового РАС, через который происходит отображение начальной виртуальной страницы окна (в окно могут входить и несколько страниц);

siz – размер окна в блоках по 32 слова;

rid – идентификатор района, на который будет отображаться данное окно. Этот идентификатор обычно определяет монитор, причем он станет известен лишь после создания монитором района. Значение идентификатора надо будет программно перенести в поле W.NRID из поля R.GID БОР;

off – смещение от начала района до места, с которого начинается отображение на район данного окна. Район может быть больше окна, и окно можно последовательно отображать на различные участки района;

len – размер области отображения в блоках по 32 слова (отображать можно и часть окна). Если значение аргумента len равно нулю или он отсутствует, отображается все окно (или часть окна, соответствующая размеру района, если окно больше района);

sts – слово статуса окна, содержащее диагностическую информацию в разрядах 15, 14 и 13 и управляющую в разряде 8. Установка разряда 8 (мнемоническое обозначение WS.MAP)требует от монитора автоматического отображения нового окна, созданного с помощью программного запроса .CRAW. Если разряд WS.MAP не установлен, созданное окно надо отображать с помощью программного запроса .MAP.

Разряды 15, 14 и 13 слова статуса окна контролируются монитором. Установка разряда 15 говорит об успешном создании окна; разряда 14 – о снятии отображения одного или нескольких существующих окон в результате выполнения данной операции (создания или отображения нового окна); разряда 13 – об удалении одного или нескольких окон.

Для управления памятью используется специальная группа системных макрокоманд.

Макрокоманда .CRRG служит для создания района; размер района берется из БОР, а его местоположение в физической ОП определяет монитор.

Для создания окна служит макрокоманда .CRAW. Характеристики окна (размер, положение в виртуальном пространстве, район для отображения и др.) берутся из БОО.

Отображение окна осуществляется с помощью макрокоманды .MAP. После выполнения этого запроса виртуальные адреса, входящие в окно, отождествляются с физическими адресами ОП и программа получает доступ к району.

Для динамического управления памятью по ходу выполнения программы предусмотрены макрокоманды .UNMAP (снять отображение), .ELRG (уничтожить район) и .ELAW (уничтожить окно). Получить текущий статус окна можно с помощью макрокоманды .GMCK.

Рассмотрим примеры использования программных запросов управления памятью. Пусть для накопления результатов измерений в ОП требуется 8К слов и еще 4К слов занимают константы компенсации аппаратурных погрешностей. Компенсация выполняется в реальном времени, в процессе накопления данных, поэтому оба буфера – и данных, и констант – должны быть доступны программе в ходе измерений. Предположим, что для управления измерительной установкой разработан и включен в систему специальный драйвер. Это избавляет от необходимости программировать работу установки на физическом уровне и позволяет более полно использовать виртуальное адресное пространство, так как программа не должна отображаться ни на регистры ВУ, ни на монитор ОС. В этом случае естественно программу сделать виртуальной.

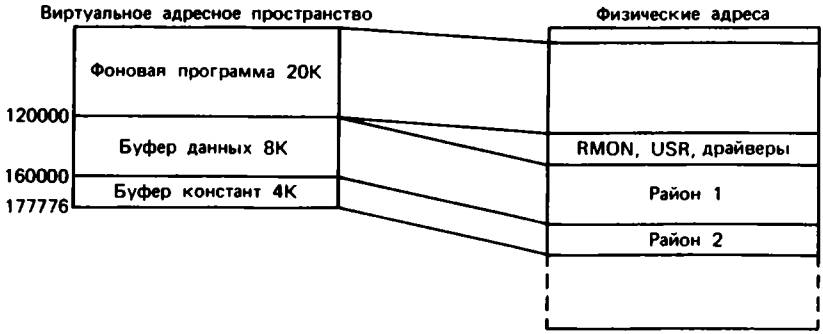


Рис. 5.15. Пример расширения адресного пространства задания

Как уже отмечалось, максимальный размер виртуальной программы составляет 17...21К, а если программный комплекс работает в двухзадачном режиме, то еще меньше. Вместе с тем старшие виртуальные адреса программы свободны, и их можно использовать для работы с расширенной памятью, где в этом случае следует предусмотреть районы для размещения буферов данных (8К) и констант (4К). Отведем под буфер данных область виртуального пространства с адресами $120000...157776_8$, а под буфер констант – область $160000...177\,776_8$ (рис. 5.15). Сама программа может при этом занимать младшие 20К адресного пространства. Будем для определенности считать, что в буфере данных накапливается распределение поступающих кодов событий по каналам (всего 8192 канала), при этом компенсация аппаратурных погрешностей заключается в прибавлении к каждому коду соответствующей константы из буфера констант. Номер используемой константы определяется старшими 12 битами 13-разрядного кода события. Участки программы выделения расширенной памяти и накопления кодов событий приведены ниже:

```
ARG:  .BLKW 5                                  ; Область для аргументов
REG1: .RDBBK 8192./32.                         ; Буфер 8К – БОР 1
REG2: .RDBBK 4096./32.                         ; Буфер 4К – БОР 2
WDB1: .WDBBK 5, 8192./32., 0, 0, 0, WS .MAP    ; БОО 1
WDB2: .WDBBK 7, 4096./32., 0, 0, 0, WS .MAP    ; БОО 2
      .ASECT                                   ; Объявление
      . = 44                                   ; программы
      2000                                     ; виртуальной
      .PSECT
      .CRRG     #ARG, #REG1                    ; Создание района 1
      .CRRG     #ARG, #REG2                    ; Создание района 2
      MOV     REG1 + R.GID, WDB1 + W.NRID      ; Пересылка идентификаторов
      MOV     REG2 + R.GID, WDB2 + W.NRID      ; районов
```

```
.CRAW   #ARG, #WDB1          ; Создание и отображение окна1
.CRAW   #ARG, #WDB2          ; Создание и отображение окна2
  .
  .; Участок накопления данных
  .
  .; В R0 поступил код события
  .
MOV     R0, R3               ; Передача кода в рабочий
                             ; регистр
BIC     #160001, R3          ; В R3 смещение в массиве
                             ; констант
ADD     #160000, R3          ; В R3 виртуальный адрес
                             ; константы
ADD     (R3), R0             ; В R0 скорректированный код
ASL     R0                   ; В R0 смещение в массиве
                             ; данных
ADD     #120000, R0          ; В R0 виртуальный адрес
                             ; канала
INC     (R0)                 ; Инкремент в канале
```

В области данных с помощью макрокоманд .RDBBK и .WDBBK создаются два БОР по числу районов и два БОО по числу окон. В БОО 1 указывается РАС 5 (виртуальный адрес канала окна 120000_8), а в БОО 2 – РАС 7 (виртуальный адрес 160000_8). Параметр WS.MAP требует автоматического отображения окна после его создания, что избавляет от необходимости использовать программный запрос .MAP.

В начале программы устанавливается разряд 8 слова состояния задания (фактически это выполняется компоновщиком при создании файла с загрузочным модулем). Далее создаются районы, а их идентификаторы пересылаются в соответствующие БОО. Программными запросами .CRAW создаются и сразу же отображаются окна. Начиная с этого момента программа может обращаться по виртуальным адресам 120000... 157776_8 (район 1) и 160000...177776_8 (район 2). Следует обратить внимание на то, что последняя виртуальная страница (РАС 7) отображается не на страницу ввода-вывода, как обычно, а на расширенную память.

Прием измерительных данных осуществляется драйвером измерительной аппаратуры, который, как и все драйверы, работает в режиме прерываний. Предполагается, что принятый код событий поступает в R0. Этот код копируется в регистр R3, где наложением маски из него выделяются старшие 12 бит (биты 1...12). Эти биты образуют код номера константы. Поскольку он начинается в разряде 1 (а не 0), в R3 оказывается записанным смещение к требуемой константе компенсации относительно начала массива констант. Виртуальный адрес начала массива известен – это 160000_8, он же – базовый адрес для РАС 7. Прибавление этого адреса к смещению позволяет получить абсолютный виртуальный адрес константы. Далее в R0 выполняется коррекция кода события – прибавление к нему константы компенсации: скорректированный код

умножается на 2 для получения смещения к соответствующему каналу в буфере данных, и прибавлением виртуального адреса начала массива данных (числа 120000_8) формируется абсолютный виртуальный адрес канала, в который попал скорректированный код события. Командой INC выполняется инкремент числа отсчетов в данном канале.

Рассмотрим теперь пример динамического управления расширенной памятью из привилегированной программы, имеющей отображение на монитор и регистры ВУ. Привилегированный режим нужен в тех случаях, когда программирование измерительной аппаратуры выполняется на физическом уровне, без использования драйвера.

В ряде случаев, например при исследовании переходных процессов, накопление измерительной информации заключается не в распределении регистрируемых кодов по каналам, а в записи их в выделенный буфер в порядке поступления, что дает возможность построить зависимость измеряемой величины от времени. При исследовании быстропротекающих процессов требуется обеспечить минимальное время записи каждого кода, что исключает использование на стадии измерений магнитных дисков. В то же время доступной памяти может не хватить для накопления заданного объема информации. Решением вопроса является динамическое переотображение виртуального буфера на различные участки физической памяти с целью последовательного их использования сначала для накопления информации, а затем (после окончания измерений) для считывания и обработки.

Диаграмма отображения программы приведена на рис. 5.16. Предположим, что сама программа занимает не более 8К. Тогда под буфер данных можно выделить три виртуальные страницы общим размером 12К начиная с адреса 40000_8. В физической памяти создается район размером 36К, куда данные записываются сначала в первую треть (область *А*), затем, после переотображения виртуального окна, во вторую (область *Б*) и, наконец, в третью (область *В*). Считывание и обработка данных осуществляется также по областям.

Максимальную скорость приема измерительной информации обеспечивает режим ожидания готовности. Для получения непосредственного доступа к регистрам ВУ задача объявляется привилегированной (бит 10 слова состояния задания содержит 0). Строки программы, имеющие отношение к накоплению данных, могут выглядеть следующим образом:

```
CSR-160100                                  ; Регистр управления
DBR-160102                                  ; Регистр данных
ARG:    .BLKW 5                             ; Область для аргументов
REG:    .RDBBK 1024.*12.*3/32.              ; Район 12К×3 = 36К
WDB:    .WDBBK 2, 1024.*12./32.             ; Окно 12К
START:.CRRG #ARG, #REG                      ; Создание района
        MOV     REG+R.GID, WDB+W.NRID       ; Пересылка идентификатора
        .CRAW   #ARG, #WDB                  ; Создание окна
        .MAP    #ARG, #WDB                  ; Отображение на область А
```

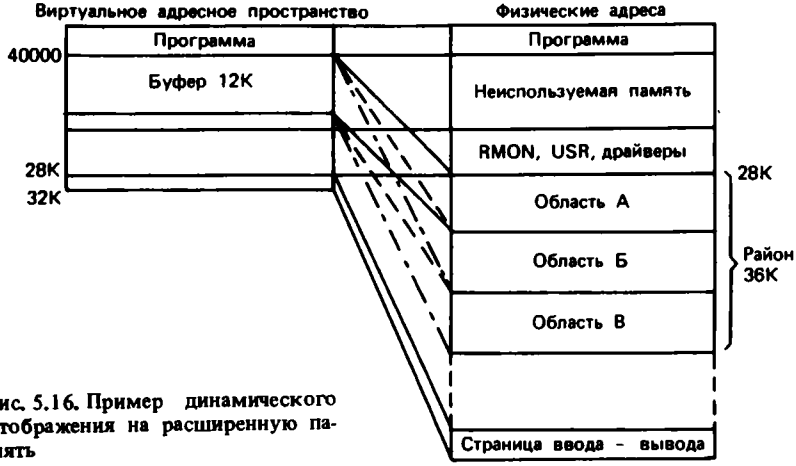


Рис. 5.16. Пример динамического отображения на расширенную память

```
        MOV     #1024.*12., R2                  ; Число слов в области
        MOV     #3, R3                          ; Число областей
        MOV     #40000, R1                      ; Начальный виртуальный адрес
; Синхронизация с началом исследуемого процесса
WAIT:   TSTB    CSR                             ; Ожидание готовности
        BPL     WAIT                            ; очередного кода
        MOV     DBR, (R1)+                      ; Прием кода
        SOB     R2, WAIT                        ; Повторить 12К раз
        ADD     #1024.*12./32., WDB+W.NRID      ; К следующей области
        .MAP    #ARG, #WDB                      ; Отображение на следующую
                                                ; область
        SOB     R3, WAIT                        ; Повторить для всех областей
; Весь район заполнен
        CLR     WDB+W.NOFF                      ; Смещение к области А
        .MAP    #ARG, #WDB                      ; Отображение на область А
        MOV     #40000, R1                      ; Базовый виртуальный адрес
        MOV     (R1)+, R4                       ; Начало обработки первого
                                                ; кода
```

В области данных программы создаются блоки описания единственного района размером 36К и единственного окна размером 12К с базовым РАС 2, которое будет последовательно отображаться на три участка района (области *А, Б, В*). В БОО подразумеваются нулевые значения аргументов W.NOFF (смещение относительно начала района) и W.NLEN (размер области отображения; при нулевом значении аргумента окно отображается целиком). В начале программы создается район, его идентификатор пересылается в БОО, затем создается и отображается окно (на область *А*). Определяется число кодов, накапливаемых в каждой области

(в регистре R2), и число областей (в регистре R3). В регистр R1 заносится известный нам начальный адрес окна (виртуальная страница 3, адрес 40000_8). На этом инициализация программы заканчивается, программа готова к приему данных.

Прием данных из измерительной установки должен начаться одновременно с началом исследуемого процесса. Поэтому программа либо должна сама запускать исследуемый процесс, либо ожидать его начала.

Прием каждого кода выполняется с помощью четырех программных строк – ожидания готовности измерительной аппаратуры, передачи кода из регистра данных аппаратуры в ОП с помощью автоинкрементной адресации и перехода на новый такт ожидания. После накопления 12К кодов выполняется переотображение виртуального окна на следующий участок района. Для этого в поле W.NOFF БОО заносится требуемое значение смещения и выдается программный запрос .MAP.

Приведенные примеры не исчерпывают возможности использования расширенной памяти в программах реального времени.

5.6. РАЗРАБОТКА ПРОГРАММ УВЕЛИЧЕННОГО РАЗМЕРА

Как уже отмечалось, адресное пространство 16-разрядных ЭВМ составляет всего 32К. В зависимости от свойств ОС программе пользователя разрешается занимать большую или меньшую часть этого пространства. Так, в системе ОС РВ прикладные задачи (правда, только те, которые не работают с регистрами ВУ на физическом уровне) могут использовать все виртуальное адресное пространство и иметь размер, приближающийся к 32К (небольшая часть адресного пространства отводится под стек задачи и ее заголовок – системную область, близкую по назначению системной области связи). В системе NTS прикладная программа может иметь размер до 28К; наконец, при работе под управлением FB-монитора РАФОС 4К отводится под страницу ввода-вывода, по меньшей мере 4К занимает резидентный монитор, 1...3К уходит на размещение драйвера системного устройства, векторов прерываний и стека, и в результате максимальный размер программы уменьшается до 23...21К. Существует несколько возможностей увеличения размера программы. Одна из них – работа с расширенной памятью – рассматривалась в предыдущем параграфе. В этом параграфе рассмотрены две другие методики – оверлейный режим и запуск программ "цепочкой".

Организация *оверлейных программ* (*программ с перекрытиями*) является наиболее распространенным приемом выхода за рамки виртуального адресного пространства, используемым во многих ВС. Максимальный размер оверлейной программы ограничивается только наличным *дисковым* пространством и на малых ЭВМ может доходить до нескольких мегаслов. Сущность оверлейного режима заключается в том, что весь текст программы делится на участки (*сегменты*), из которых толь-

ко начальный, *корневой* сегмент является резидентным и постоянно находится в ОП. Остальные сегменты хранятся на НМД и загружаются в память поочередно по мере их вызова из корневого или какого-либо загруженного ранее сегмента, накладываясь при этом на уже отработавшие и ненужные в настоящий момент участки программы. Очевидно, что программа с перекрытиями должна быть тщательно распланирована, чтобы исключить, например, переходы из какого-то сегмента в тот, который был им же только что вытеснен из ОП. Однако программы, составленные с учетом современных взглядов на методологию программирования и имеющие хорошо структурированный модульный характер, обычно легко поддаются такому планированию.

Рассмотрим пример программного комплекса реального времени, работающего в оверлейном режиме. Пусть процедура измерений состоит из трех этапов:

инициализации установки и градуировки измерительной аппаратуры;

собственно измерений и накопления измерительной информации;

обработки накопленных данных.

Будем считать, что из измерительной аппаратуры поступают 13-разрядные коды, которые в зависимости от поставленной задачи должны накапливаться в ОП либо последовательным списком, либо в виде распределения по каналам прямоугольной матрицы, состоящей из 128×64 = 8К каналов. На планирование оверлейного программного комплекса оказывают влияние три обстоятельства:

количество относительно независимых программных участков;

размеры этих участков;

логические связи между ними.

Рассматриваемый программный комплекс целесообразно составить из следующих сегментов:

корневой резидентный сегмент ROOT, управляющий последовательностью вызова остальных сегментов, т. е. порядком выполнения этапов измерений;

сегмент INIT, в котором осуществляется инициализация установки, а также градуировка аппаратуры и вычисление градуировочных коэффициентов:

сегмент GRAD, представляющий собой массив градуировочных коэффициентов;

сегмент LIST, в который входят программа приема измерительной информации в режиме "список", а также одномерный массив, заполняемый последовательно приходящими кодами;

сегмент MATRIX с программой распределения поступающих кодов по каналам и матрица для их накопления;

сегмента PROCES, включающего в себя программу обработки.

Оценим приблизительно размеры описанных участков программы. В резидентный корневой сегмент кроме относительно недлинной программы задания порядка выполнения этапов измерений войдет абсолютная

программная секция .ABS., занимающая первые 256 слов ОП, а также оверлейный драйвер, в функции которого входит физическая загрузка затребованных в программе сегментов в ОП и передача им управления в соответствии с командами передачи управления JSR или JMP, записанными в прикладной программе. Размер оверлейного драйвера составляет 120...150 слов или несколько больше, в зависимости от количества точек входа в перекрываемые участки программы. Будем считать, что корневой сегмент ROOT займет 0,5К слов.

Подпрограмма инициализации и градуировки может содержать большое количество символьных таблиц, используемых в процессе начального диалога с оператором, а также таблиц с адресами регистров измерительной и прочей аппаратуры, исходных данных и др. Пусть размер этого сегмента составляет 4,5К слов.

Если считать, что каждому значению измеренного кода соответствует свой градуировочный коэффициент, массив коэффициентов в сегменте GRAD займет объем 8К слов.

Размер сегмента LIST определяется главным образом количеством кодов, накапливаемых в ОП за одну серию измерений. Если условия измерений требуют накопления 10^4 кодов, то с учетом самой программы накопления размер сегмента может составить 11К слов.

В сегменте MATRIX 8К слов займет матрица для накопления распределения регистрируемых кодов, а с учетом программы распределения по каналам размер сегмента может составить, например, 9К слов.

Длина сегмента PROCES определяется принятым алгоритмом обработки. Пусть для конкретного алгоритма размер сегмента PROCES составляет 7К слов. Тогда весь программный комплекс будет иметь объем 40К слов, что в 2 раза превышает допустимый размер программы.

Распланируем память под участки программы так, как это показано на рис. 5.17. При этом примем во внимание следующие соображения:

сегмент ROOT, являющийся резидентным, должен всегда находиться в ОП;

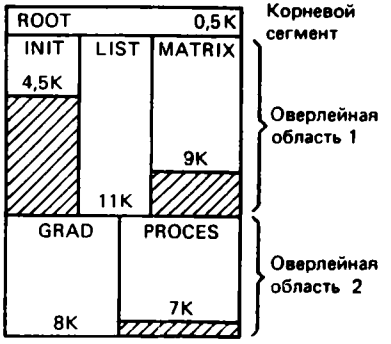


Рис. 5.17. План оверлейной программы (заштрихованы временно неиспользуемые области ОП)

процедуры градуировки и измерений списком или матрицей взаимно исключают друг друга, поэтому для сегментов INIT, LIST и MATRIX можно отвести одну и ту же *оверлейную область* памяти. Размер оверлейной области определяется длиной наибольшего входящего в нее сегмента и в данном случае составляет 11К;

массив градуировочных коэффициентов будет использоваться и при градуировке, и при накоплении информации. Поэтому сегмент GRAD должен находиться в ОП одновременно с сегментами INIT, LIST или MATRIX и, следовательно, входить в отдельную оверлейную область. С другой стороны, процедура обработки, выполняемая после завершения измерений, не использует градуировочные коэффициенты (если предположить, что поправка на результаты градуировки выполняется в процессе накопления принимаемых кодов), и сегмент PROCES может разделить с сегментом GRAD одну оверлейную область, размер которой составит 8К.

На рис. 5.17 видно, что объем ОП, требуемый для размещения оверлейной программы, не превышает 19,5К, что позволяет выполнять эту программу под управлением FB-монитора РАФОС на машинах без расширенной памяти.

Порядок загрузки и выполнения отдельных подпрограмм будет, очевидно, следующий.

Из резидентной части программы вызывается сегмент INIT с подпрограммой установки начальных условий и градуировки. Поскольку для выполнения градуировки требуется наличие в ОП сегмента GRAD с областью для записи градуировочных коэффициентов, подпрограмма в сегменте INIT должна инициировать загрузку во вторую оверлейную область сегмента GRAD и приступить к процедуре градуировки. После завершения градуировочных измерений, вычисления градуировочных коэффициентов и занесения их в массив сегмента GRAD подпрограмма сегмента INIT завершается и возвращает управление в резидентную часть программы. В зависимости от заданных условий на место INIT загружается один из двух сегментов накопления (LIST или MATRIX), и начинаются собственно измерения. При этом сегмент GRAD остается в ОП и используется подпрограммами накопления для внесения оперативных поправок в получаемые из измерительной аппаратуры коды. По завершении измерений управление возвращается в сегмент ROOT, откуда инициируется загрузка (на место уже ненужных градуировочных коэффициентов) сегмента PROCES с подпрограммой обработки. При этом, хотя программные строки накопления, входящие в подпрограммы сегментов LIST или MATRIX, уже отработаны, однако массивы накопленных данных будут использоваться подпрограммой обработки и должны находиться в ОП. Обработка завершается записью полученных результатов на НМД, возвратом в корневой сегмент ROOT и завершением задания.

Рассмотренный пример показывает, что при выполнении оверлейной программы возникает необходимость в организации различного рода связей отдельных сегментов программы:

вызов подпрограммы, входящей в оверлейный сегмент с предварительной загрузкой этого сегмента или без загрузки, если он уже находится в ОП;

загрузка в ОП сегмента с полями данных (пустыми или содержащими какую-либо исходную информацию) с целью последующей работы с этими данными;

обращение к данным, находящимся в другом сегменте.

Для пояснения способов реализации указанных связей рассмотрим возможную структуру программы (или подпрограммы), написанной на языке АССЕМБЛЕРа. Типичная программа включает в себя программные строки и поля данных, имеет одну или несколько *точек входа (стартовых точек)* и состоит из одной или нескольких *программных секций* либо входит наряду с другими подпрограммами в общую программную секцию. Поскольку текст программы содержится в каком-то файле, для ее идентификации часто используется имя этого файла. Так, в предшествующем тексте под именами сегментов подразумевались имена соответствующих объектных файлов. Наконец, файл может состоять из нескольких *программных модулей*, которым назначаются имена с помощью директивы .PSECT. В ряде случаев, например при работе с объектными библиотеками, имя программного модуля приобретает особое значение.

Пусть в файле ACQUIS.MAC (который в результате трансляции преобразуется в объектный файл ACQUIS.OBJ) содержатся следующие строки:

```
            .TITLE  ACQLST
            .PSECT  EXEC
START::     MOV     #8., R0
START1::    MOV     R0, -(SP)

            ⋮; Программные строки
END:        RTS     PC
            .PSECT  DAT
            .GLOBL  MAS1, MAS2, LEN1, LEN2
LEN1:       .WORD   4096.
MAS1:       .BLKW   4096.
LEN2:       .WORD   1024.
MAS2:       .BLKW   1024.

            ⋮; Поля данных

            .END
```

В приведенном примере программные строки с помощью директивы PSECT выделены в программную секцию с именем EXEC; поля данных входят в другую программную секцию с именем DAT. Такое разделение не является необходимым, однако в программах на языке ассемблера, использующих подпрограммы на ФОРТРАНе, а также в оверлейных программных комплексах оно оказывается полезным или даже необходимым. Подпрограмма имеет две точки входа START и START1 и может вызываться, например, командами JSR PC, START или JSR PC, STSRT1 с различием в исходных условиях: в первом случае подпрограмма сама определяет значение некоторого параметра (число 8), занося его в R0, а во втором предполагает, что значение этого параметра передается ей через регистр R0 извне. Входные метки START и START1, которые можно рассматривать как альтернативные имена подпрограммы, объявлены с помощью двойного двоеточия глобальными. Это значит, что вызов нашей подпрограммы может содержаться не только в этом, но и в других программных модулях. Завершающая метка END не носит глобального характера: обращение к ней возможно только из этого же программного модуля.

Метки MAS1 и MAS2, характеризующие массивы данных, также объявлены глобальными с помощью директивы .GLOBL, эквивалентной двойному двоеточию. Таким образом, становится возможным запись в эти массивы (или чтение из них) из других программных модулей. Метки LEN1 и LEN2, указывающие на слова, в которых записаны длины массивов, объявлены глобальными для того, чтобы к ним можно было обращаться из другой программной секции (EXEC) того же модуля.

Вернемся к примеру оверлейного программного комплекса. Его структура может иметь следующий вид:

```
            ; Файл ROOT.MAC
START:      JSR  PC, INI          ; Загрузка сегмента INIT
              .
              .; Подготовка к измерениям
              .
            JSR  PC, LISTAC       ; Загрузка сегмента LIST
              .
              .; Подготовка к обработке
              .
            JSR  PC, PROCES       ; Загрузка сегмента PROCES
            .PRINT #MSG
            .EXIT
MSG:        .ASCIZ <16> "ПРОГРАММА ЗАВЕРШИЛАСЬ" <17>
            .EVEN
            .END  START
            ; Файл INIT.MAC
            .PSECT INISCT
INI: :      JSR  PC, GRADU        ; Загрузка сегмента GRAD
```

```
            .
            .; Процедура градуировки
            .
            .PRINT #MSG
            RTS  PC              ; Возврат в ROOT
MSG:        .ASCIZ <16> "ГРАДУИРОВОЧНЫЕ КОЭФФИЦИЕНТЫ ПОЛУЧЕНЫ"
                        <17>
            .EVEN
            .END
            ; Файл GRAD.MAC
            .PSECT GRDSCT
GRADU::     RTS  PC              ; Возврат в INIT после загрузки сегмента
            .PSECT COFSCT D
COEFF::     .BLKW 8192.
            .END
            ; Файл LIST.MAC
            .PSECT LSTSCT
LISTAC::    .PRINT #MSG1
            .
            .; Процедура накопления списком
            .
            .PRINT #MSG2
            RTS  PC              ; Возврат в ROOT
MSG1;       .ASCIZ  <16> "НАЧАЛОСЬ НАКОПЛЕНИЕ СПИСКОМ"<17>
MSG2:       .ASCIZ <16> "НАКОПЛЕНИЕ СПИСКОМ ЗАКОНЧИЛОСЬ" <17>
            .EVEN
            .PSECT LSTDAT D
LISTDT::    .BLKW  10000.
            .END
            ; Файл MATRIX.MAC
            .PSECT MTRSCT
MTRAC::
            .; Процедура накопления спектра
            .
            .
            RTS  PC              ; Возврат в ROOT
            .PSECT MTRDAT D
MTRXDT::    .BLKW  8192.
            .END
            ; Файл PROCES.MAC
            .PSECT PRCSCT
            .PRINT #MSG
PROCES::
            .
            .; Обработка данных
            .
            RTS  PC
MSG:        .ASCIZ <16> "НАЧАЛАСЬ ОБРАБОТКА" <17>
            .EVEN
            .END
```

В резидентной программе корневого сегмента осуществляется последовательный запуск подпрограмм инициализации и градуировки INI, собственно измерений LISTAC (или, если требуется, MTRAC) и, наконец, обработки PROCES. Естественно, строки загрузки соответствующих сегментов и вызова подпрограмм могут разделяться программными блоками подготовки к измерениям (например, выбор алгоритма и условий измерений) и обработки (например, выбор алгоритма обработки или ввод установочных констант). После окончания обработки резидентная программа выводит сообщение "ПРОГРАММА ЗАВЕРШИЛАСЬ" и завершается выполнением программного запроса .EXIT.

Сегмент INIT загружается в ОП оверлейным драйвером при выполнении команды JSR PC, INI (в действительности, компоновщик, обрабатывая модули оверлейной программы, заменяет указанные в ней адреса переходов на адреса "фиктивных" подпрограмм, находящихся в самом драйвере и обеспечивающих его запуск с целью загрузки нового сегмента). Первой же командой подпрограммы INI выполняется загрузка сегмента GRAD и передача управления на его входную точку GRADU. Команда RTS PC – единственная строка программной секции GRDSCT – возвращает управление назад в подпрограмму INI. Таким образом сегмент GRAD оказывается загруженным и подпрограмма INI получает доступ к массиву COEFF.

Подпрограмма INI включена в программную секцию INISCT. Это сделано для наглядности и не является необходимым. Если в файле INIT опустить директиву .PSECT с явным указанием имени, компоновщик создаст непоименованную программную секцию и, поскольку ей по умолчанию присваивается атрибут LCL (local – локальная), о ее существовании будет известно только в данном программном модуле. В другом модуле (например, GRAD) тоже может иметься непоименованная локальная программная секция, но о ней тоже не будет известно за пределами модуля, и совпадение их имен (точнее, совпадение пустых мест вместо имен) не повлияет на процесс компоновки. Не так обстоит дело в модуле GRAD. Здесь секцию GRADSCT также можно было бы не объявлять, однако введение специальной программной секции для массива данных COFSCT с атрибутом D (data – данные) является обязательным. Дело в том, что по умолчанию программной секции присваивается атрибут I (instruction – команда). Этот атрибут не означает, что в такой секции не может быть полей данных. Однако компоновщик, создавая оверлейную программу и встретив в такой секции глобальную метку, рассматривает ее как точку входа в подпрограмму Все ссылки на нее из других модулей компоновщик, как уже отмечалось, заменяет ссылками на сгенерированные им фиктивные подпрограммы в оверлейном драйвере. Поэтому глобальными метками в программных секциях с атрибутом I можно пользоваться только для передачи на них управления, но не для обращения к ним как к адресам массивов с данными. Явное назначение секции GRDSCT атрибута D выключает меха-

низм образования фиктивных ссылок, и глобальную метку COEFF теперь можно использовать как имя массива с данными. По той же причине введены программные секции с атрибутом D в модули LIST и MATRIX. Это дает возможность обращаться к массивам LISTDT и MTRXDT из подпрограммы обработки PROCES, входящей в другой сегмент.

Подпрограмма INI, выполнив градуировку, выводит соответствующее сообщение и возвращает управление в ROOT, где после проведения подготовки к измерениям вызывается подпрограмма LISTAC (или MTRAC). Программа оповещает оператора о запуске, а затем об окончании измерений и завершается возвратом в ROOT. Однако текст программы и, главное, массив с накопленными данными остаются в ОП, и программа обработки после загрузки ее (в другую оверлейную область!) может обращаться к данным с помощью, например, команд вида MOV LISTDT, R1.

Поскольку оверлейный программный комплекс создает компоновщик, в командной строке его вызова должна содержаться информация о расположении в ОП отдельных сегментов программы. Применительно к рассматриваемому примеру это делается вводом с клавиатуры терминала следующей команды, состоящей из нескольких строк:

```
LINK/PROMPT/MAP:DK: ROOT
INIT/O:1
LIST/O;1
MATRIX/O:1
GRAD/O:2
PROCES/O:2//
```

Ключ PROMPT позволяет вводить команду, состоящую из нескольких строк (в первой строке указывается только имя резидентного модуля, но не оверлейных). Ключ MAP приводит к созданию карты загрузочного модуля, в котором указаны фактические адреса программных секций и их размеры, а также значения глобальных меток. Эти сведения необходимы для отладки программного комплекса. В последующих строках указано, что сегменты INIT, LIST и MATRIX принадлежат первой оверлейной области и, следовательно, загружаются в память на одно и то же место. Сегменты GRAD и PROCES принадлежат второй оверлейной области. Две косых черты признаком конца команды.

Оверлейный режим является простым и эффективным средством практически неограниченного увеличения размера программы. Другим способом, имеющим меньшие возможности, является организация цепочки программ, при которой каждая программа, входящая в программный комплекс, перед своим завершением запускает следующую программу, та, отработав, запускает в свою очередь следующую и т. д. Последовательный запуск программ осуществляется программным запросом .CHAIN, заменяющим обычную макрокоманду завершения .EXIT.

Завершающаяся программа может передать своей "сменщице" произвольную информацию. Для этого отводятся 102 слова с адресами 510 – 776_8. Слова с адресами 500 – 506_8 используются для передачи системе спецификации файла с запускаемой программой. Дополнительную возможность передачи информации программе, запускаемой по цепочке, предоставляют каналы, которые не закрываются при смене программ. Таким образом, запускаемая программа, используя открытый запускающей программой канал, может продолжать работать с файлом, даже не зная его имени.

Структура программы, использующей программный запрос .CHAIN, выглядит следующим образом:

```
NXTPGM:  .RAD50  'DK PROGR2SAV'
MESSG:   .BLKW   102.              ; Буфер для пересылки
START:
            .
            .; Текст программы
            .
         MOV     #106., R0         ; Всего 106 слов
         MOV     #NXTPGM, R1       ; начиная с NXTPGM
         MOV     #500, R2          ; По адресу 500 и далее
1 $      MOV     (R1) +, (R2) +    ; переслать
         SOB     R0, 1$            ; в цикле
         .CHAIN                    ; Вызов следующей программы
         .END    START
```

Глава 6

ПРОГРАММНЫЕ КОМПЛЕКСЫ С ОБРАБОТКОЙ ПРЕРЫВАНИЙ

6.1. ОБРАБОТКА ПРЕРЫВАНИЙ ПОД УПРАВЛЕНИЕМ ОС

Простая ВС, выполняющая заданный заранее и ограниченный набор функций по приему данных и управлению установкой может работать и без ОС. В этом случае программист лишается возможности использовать системные программы, обеспечивающие ввод-вывод через стандартные ВУ, временну́ю синхронизацию измерительного процесса с помощью системного таймера и т. д. Все эти действия должны программироваться на физическом уровне путем обращения к регистрам соответствующих устройств. Такая методика характерна для автономных ИВК, в которых отсутствие дисковой памяти не позволяет использовать современные дисковые ОС.

Чаще, однако, ВС работает под управлением ОС. В этом случае ОС, предоставляя программисту широкий набор системных средств организации вычислительного процесса, накладывает и определенные ограничения на функционирование прикладных программ. Одним из таких

ограничений является требование, чтобы при выполнении программы обработки прерываний монитор находился в *системном состоянии.*

FB- и XM-мониторы могут находиться в одном из двух состояний: *системном* или *пользователя.* В состоянии пользователя выполняются любые задания (т. е. прикладные программы и утилиты), а также интерактивный монитор KMON и ряд программ резидентного монитора KMON. Это состояние характеризуется возможностью переключения заданий (с фонового на оперативное и наоборот) в соответствии с алгоритмом работы *планировщика заданий*, причем каждое задание использует свой стек, свою смешанную область и свое содержимое системной области связи. Монитор переходит в системное состояние для выполнения действий, затрагивающих важные для его работы структуры данных. В системном состоянии задания выполняться не могут, работа планировщика запрещена и используется системный стек (находящийся в RMON). Находясь в этом состоянии, монитор может модифицировать системные таблицы, не боясь вмешательства в этот процесс активных заданий. В системном состоянии выполняются, в частности, такие действия, как блокировка задания, инициация и завершение ввода-вывода, обслуживание прерываний от системного таймера и ВУ. В FB-мониторе в системном состоянии выполняются программные запросы .PROTECT и .CHCOPY, а в XM-мониторе сверх того еще все программные запросы, связанные с отображением. Необходимость перевода монитора в системное состояние при обработке прерываний от ВУ, в частности от измерительной аппаратуры, а также определенные ограничения, связанные с этим состоянием, приводят к специфической методике написания программ обработки прерываний (ПОП), работающих в рамках ОС. Для организации режима прерываний в системе РАФОС используются программные запросы .PROTECT, .DEVICE, .INTEN и .SYNCH.

Программный запрос .PROTECT служит для защиты используемого заданием вектора прерываний перед загрузкой в него адреса ПОП и значения приоритета процессора. Защита вектора закрепляет его за данным заданием и к тому же позволяет удостовериться в том, что вектор не используется уже другим заданием (прикладным или системным). Это особенно важно при работе в фоново-оперативном режиме. Получив от оператора команду запуска оперативного задания, монитор загружает программу с диска в ОП на отведенное для нее место между RMON и USR. При этом данные из нулевого блока файла загружаются в начало ОП, в ячейки $0...476_8$ (рис. 6.1). Обычно нулевой блок содержит информацию только в ячейках $40...56_8$, соответствующих системной области связи. Если в оперативном задании предусмотрена обработка прерываний, соответствующие векторы прерываний заполняются требуемыми значениями по ходу выполнения программы, на участке инициализации прерываний. Однако при запуске оператором фонового задания в начало ОП (ячейки $0...776_8$) будет загружен его нулевой блок, что приведет к уничтожению содержимого векторов прерываний оперативного задания.

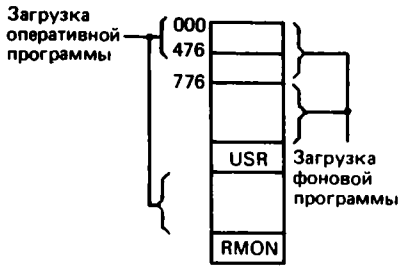


Рис. 6.1. Использование оперативной памяти заданиями пользователя

Для защиты используемых векторов предусмотрена специальная таблица – *карта защиты памяти*. Эта таблица длиной 20 байт расположена в RMON. Каждый из 160 двоичных разрядов, входящих в таблицу, сопоставлен с определенным словом памяти в области векторов прерываний (ячейки $0...476_8$). Установка разряда защищает соответствующий вектор прерываний, предохраняя его содержимое от разрушения при загрузке нулевого блока файла программы. Некоторые ячейки автоматически защищаются монитором РАФОС (другими словами, соответствующие этим ячейкам разряды карты защиты устанавливаются самим монитором). Сюда относятся, например, векторы прерываний консольного терминала, системного устройства (НМД), системного таймера и др. При загрузке в ОП программ содержимое этих ячеек не изменяется. Для того же, чтобы предохранить от разрушения векторы прерываний, используемые в заданиях, их надо защитить с помощью программного запроса .PROTECT. Этот запрос корректирует карту защиты памяти, закрепляя указанный в макровызове вектор прерываний за данным заданием, и гарантирует его сохранность. Если после выполнения запроса бит С РСП оказался установлен, это значит, что монитор не смог защитить данный вектор и программе с ним работать нельзя. Такая ситуация может возникнуть, если заданный адрес вектора превышает 474_8 или не кратен четырем (в этом случае байт 52_8 системной области связи содержит код завершения 0) или данный вектор уже закреплен за другой программой (код завершения 1). Таким образом, строки заполнения вектора прерываний выглядят следующим образом:

```
        .PROTECT #ARG, #300  ; Защита вектора 300
        BCS      ERROR       ; Вектор занят
        MOV      #ISR, 300   ; Адрес ПОП в вектор
        MOV      #340, 302   ; Приоритет 7 в вектор
          .
          .
          .
ERROR:  .PRINT   #MSG        ; Вывод аварийного сообщения
        .EXIT                ; Аварийное завершение
MSG:    .ASCIZ  <16>"ВЕКТОР ЗАНЯТ"<17>
```

Рассмотрим теперь назначение программного запроса .DEVICE. В процессе инициализации прерываний программа устанавливает бит 6 управляющего регистра ВУ, чтобы разрешить прохождение прерываний от этого устройства. После завершения работы с устройством или во всяком случае перед окончанием всей программы, бит 6 должен быть программно сброшен. Если, однако, оператор снял программу с выполнения вводом с клавиатуры команды ⟨CTRL/C⟩⟨CTRL/C⟩, бит 6 может остаться установленным и устройство будет посылать в ЭВМ сигналы прерываний, хотя в ОП уже нет программы их обработки. Для устранения такой ситуации предусмотрена макрокоманда .DEVICE, аргументом которой служит адрес списка, содержащего адреса регистров устройств и числа, которые надо занести в эти регистры после завершения программы (нормального или по двойному ⟨CTRL/C⟩). Обычно в регистры заносится 0, что и приводит к сбросу бита 6. Идентификатором конца списка служит 0 на месте следующего адреса:

```
        .DEVICE #ARG, #LIST
          .
          .
          .
ARG:    .BLKW   2          ; Область для аргументов
LIST:   .WORD   164000     ; Адрес регистра ВУ
        .WORD   0          ; Пересылаемое число
        .WORD   164002     ; Адрес другого регистра
        .WORD   0          ; Пересылаемое число
        .WORD   0          ; Конец списка
```

Программные запросы .INTEN и SYNCH определяют "статус" ПОП и влияют на ее харктеристики и возможности. При заполнении вектора прерываний во второе слово вектора заносится значение приоритета процессора, с которым начнет выполняться ПОП. Установление в векторе прерываний наивысшего значения приоритета (7) блокирует после перехода на ПОП любые прерывания и обеспечивает монитору условия для перехода в системное состояние. Таким образом, при входе в ПОП приоритет процессора максимален, чем можно воспользоваться для выполнения самой срочной части обработки прерывания. Не рекомендуется делать длительность этой части ПОП более 50 мкс.

Основная часть ПОП должна выполняться при более низком значении приоритета процессора. Снижение приоритета и одновременно перевод монитора в системное состояние осуществляются с помощью программного запроса .INTEN, в качестве аргумента которого указывается требуемое значение приоритета, обычно совпадающее с уровнем линии запроса прерывания, к которой подключено данное ВУ. Программный запрос .INTEN переводит монитор в системное состояние, снижает приоритет до указанного значения и передает управление назад в ПОП на команду, следующую за макрокомандой .INTEN.

стоянии (причем глубина его погружения в это состояние увеличится) и управление будет передано на ПОП от устройства, приславшего запрос. После завершения этой программы управление вернется в прерванную ПОП. Хотя выполнение ПОП при этом задержится, никаких сбоев в работе программного комплекса не произойдет.

Если на том же участке программы (между строками .INTEN и .SYNCH) придет запрос на прерывание от устройства, предыдущий запрос от которого сейчас обрабатывается, арбитр не пропустит этот запрос и он будет "висеть" до тех пор, пока не снизится приоритет процессора. Если текст ПОП после макрокоманды .INTEN заканчивается командой RTS PC, новый запрос вызовет повторное прерывание лишь после того, как управление перейдет из ПОП в основную программу, т. е. после полного завершения обработки предыдущего прерывания. Не так будет обстоять дело при наличии в тексте ПОП макрокоманды .SYNCH. На участке программы после программного запроса .SYNCH приоритет процессора равен нулю и разрешены все прерывания. В случае прихода повторного запроса на прерывания от того же устройства ППЗ будет прервана и управление вернется в начало той же ПОП. Когда, выполняя ПОП, процессор дойдет до программного запроса .SYNCH, дальнейший ход программы будет зависеть от того, свободен ли блок аргументов макрокоманды .SYNCH. Элемент очереди любой ППЗ освобождается в тот момент, когда данная ППЗ активизируется процессором. Запрос .SYNCH, использовав свой блок аргументов в процессе перевода ПОП на уровень ППЗ, освобождает его после передачи управления программе. Поэтому повторный запрос .SYNCH, выданный в результате повторного прерывания от того же источника, сможет поставить завершающую часть ПОП (после макрокоманды .SYNCH) в очередь ППЗ. Она будет выполнена после того, как полностью отработает "первая" ППЗ. Таким образом, однократное наложение прерываний не нарушит правильного хода работы программного комплекса. Если, однако, в процессе выполнения ППЗ поступят последовательно два прерывания, запрос на ППЗ от второго поставлен в очередь не будет, так как единственный блок аргументов запроса .SYNCH будет занят запросом от первого поступившего прерывания. Это может привести к фатальному нарушению работы программного комплекса.

В тех случаях, когда прерывания поступают случайно во времени, необходимо исключить возможность описанных выше повторных прерываний. Проще всего это сделать, запретив прерывания от данного устройства перед переходом на уровень ППЗ и разрешив их снова уже в ППЗ:

```
.INTEN    5            ; Переход в системное состояние

  .; Программа обработки прерываний
  .
```

```
BIC      #100, CSR  ; Запрет прерываний
.SYNCH   SYNBLK     ; Переход на ППЗ
BR       ERRSYN     ; Все-таки ошибка
BIS      #100, CSR  ; Разрешение прерываний
.
.; Подпрограмма завершения
.
RTS      PC         ; Выход из ППЗ
```

В такой программе первое же прерывание, наложившееся на ППЗ, приводит к запрещению дальнейших прерываний на все время, пока очередь ППЗ не опустеет, освободив блок аргументов SYNBLK. Поэтому наложение прерываний не приводит к нарушению работы программы, хотя, конечно, часть входной информации теряется.

Экономное написание ПОП требует использования регистров общего назначения (РОН). Между тем ОС накладывает здесь определенные ограничения. Аппаратный переход на ПОП не сопровождается сохранением содержимого РОН и на участке ПОП до программного запроса .INTEN пользоваться РОН можно лишь в том случае, если их содержимое (из основной программы) будет сначала сохранено, а затем обязательно восстановлено:

```
ISR:  MOV  R0, -(SP)   ; Сохранение R0 в стеке
      MOV  R1, -(SP)   ; Сохранение R1 в стеке
        .
        .; Строки ПОП, использующие регистры R0 и R1
      MOV  (SP)+, R1   ; Восстановление R1
      MOV  (SP)+, R0   ; Восстановление R0
      .INTEN 5         ; Переход в системное состояние
```

При выполнении запроса .INTEN монитор сохраняет содержимое регистров R4 и R5, и ими можно свободно пользоваться на этом участке ПОП. Надо только иметь в виду, что содержимое R4 и R5 при входе в ПОП неопределено, а после выхода из нее будет потеряно. Если содержимое регистра требуется сохранять от прерывания к прерыванию, в конце ПОП его придется перенести в ячейку ОП:

```
      .INTEN 5         ; Переход в системное состояние
      MOV  REG4, R4    ; Восстановление R4
      MOV  DBR, (R4)+  ; Данное из РД ВУ в память
      MOV  R4, REG4    ; Сохранение нового содержимого R4
        .
        .; Любые действия с R4 и R5
      RTS  PC          ; Выход из прерывания

REG4: BLKW 1           ; Ячейка ОП для хранения содержимого R4
```

Если после запроса .INTEN требуется использовать регистры R0...R3, их содержимое надо сначала сохранить, а перед выходом из ПОП восстановить.

При выполнении запроса .SYNCH монитор сохраняет содержимое регистров R0 и R1 (из основной программы), так что ими можно пользоваться до завершения ПОП. При этом регистр R0 заполняется значением аргумента из пятого слова блока аргументов для запроса .SYNCH. Этим можно воспользоваться для передачи информации из приоритетной части ПОП в ППЗ:

```
        .INTEN 6                ; Переход в системное состояние

        ; Работа с R4
        MOV R4, SYNBLK+8.       ; Передача содержимого R4 в ППЗ
        .SYNCH  SYNBLK          ; Переход на ППЗ
        BR ERRSYN               ; Ошибка

        ; В R0 содержимое R4 из ПОП
        . Любые действия с R1
        RTS PC                  ; Выход из ППЗ и ПОП
SYNBLK: .WORD 0,0,0,0,0,-1,0    ; Блок аргументов для .SYNCH
```

Заметим, что регистр R0 используют практически все макрокоманды РАФОС. Поэтому автоматическое сохранение его содержимого имеет большой смысл, так как позволяет включать системные макрокоманды в текст ППЗ, не нарушая выполнения основной программы. Остальные регистры (R2...R5) можно использовать, лишь предварительно сохранив их содержимое (в стеке или ячейках ОП) и восстановив его перед окончательным выходом из ПОП.

6.2. ОБРАБОТКА ПРЕРЫВАНИЙ В TS-МОНИТОРЕ И СИСТЕМЕ NTS

Выполнение программ реального времени под управлением TS-монитора или в системе NTS (NTS-монитор) имеет значительную специфику, связанную с отображением виртуального адресного пространства программы на физическую ОП. Дело в том, что и в том и другом случае старшие 4К виртуального адресного пространства программы отображаются не на адреса регистров ВУ, а на область монитора, выполняющую функции программы RMON. Поэтому прикладная программа не имеет непосредственного доступа ни к регистрам ВУ, ни к векторам прерываний.

Программа может получить доступ к регистрам ВУ одним из двух способов: изменить начальное отображение так, чтобы виртуальные адреса $160000...177776_8$ были отображены на физические адреса ВУ, либо использовать специальный набор макрокоманд для работы с реги-

страми ВУ. Второй способ может заметно замедлить работу программы, так как любое обращение к ВУ, обычно выполняемое с помощью одной команды, теперь требует программного запроса к монитору, что сопряжено с существенными системными издержками. Поэтому обычно используют первый способ – изменение начального отображения.

Существенным недостатком обоих случаев, ограничивающим их использование в задачах реального времени, является запрет на включение в состав прикладной программы программ обработки прерываний. Прерывания могут обрабатываться только на уровне ППЗ, т. е. как бы после выполнения программных запросов .INTEN и .SYNCH. В результате скорость обработки внешних событий в TS-мониторе и в системе NTS заметно ниже, чем в других мониторах РАФОС. При этом исключается удобная возможность организации иерархической обработки прерываний.

ППЗ, обрабатывающие прерывания, выполняются, естественно, с нулевым приоритетом процессора. Несмотря на это, механизм вложенных прерываний не работает: прерывание (даже более высокого уровня), пришедшее до окончания обработки предыдущего, не прерывает выполняемую ППЗ, а ставит запрос на ее повторное выполнение в очередь ППЗ. Отсутствие в этот момент свободного элемента очереди приводит к аварийному отказу ОС с необходимостью ее последующей перегрузки. Для избежания таких ситуаций следует блокировать прерывания от ВУ на все время обработки поступившего прерывания.

ППЗ, обрабатывающей прерывания, назначается приоритет в диапазоне от 0 до 7. Это чисто системный, "программный" приоритет. "Аппаратный" приоритет процессора, хранящийся в РСП, при выполнении любых программ равен 0. Назначение ППЗ приоритета от 1 до 7 делает ее приоритетной. Это выражается в том, что для такой программы не действует механизм разделения времени, и она выполняется без помех до конца (до завершающей команды RTS PC). Исключение составляют два случая: активизация (внешним прерыванием) ППЗ с более высоким приоритетом в другом задании и блокирование программы в ожидании завершения заданного интервала времени или начатого ввода-вывода. В первом случае ППЗ прерывается, а после обработки более приоритетной ППЗ из другого задания продолжается с той же точки в прежнем режиме. Во втором случае ППЗ теряет свой приоритет и начинает участвовать в разделении времени наряду с остальными запущенными заданиями.

ППЗ, которой назначен приоритет 0, не является приоритетной. Она в процессе своей активизации прерывает остальные задания, но после истечения заданного при генерации кванта времени переходит в разряд обычных программ и участвует вместе с ними в разделении времени. Это, впрочем, имеет значение только для относительно длинных программ, поскольку указанный квант времени обычно достаточно велик (0,2 с).

Еще один источник снижения скорости обработки прерываний в рассматриваемых мониторах связан с механизмом подкачки.

Если активизировано больше программ, чем их может поместиться в ОП, часть из них находится на диске и ждет своей очереди для выполнения, причем происходит периодическая выгрузка отработавших свой квант времени программ из ОП на диск с загрузкой на их место программ, ожидающих на диске. Программы реального времени подчиняются общим правилам, поэтому вполне может получиться, что в момент прихода перерывания соответствующая программа выгружена на диск, и для ее загрузки в ОП потребуется значительное время. Для того чтобы избежать потерь времени и ускорить реакцию на прерывание, программу можно зафиксировать ("запереть") в памяти с помощью соответствующих программных запросов. При этом такое фиксирование имеет два варианта. В первом варианте программа перед фиксированием пересылается в начало ОП, что требует заметного времени, но устраняет фрагментацию памяти. Во втором варианте программа запирается прямо там, где она находится к моменту получения программного запроса на фиксирование. Эта процедура значительно быстрее предыдущей, но приводит к разделению ОП на несвязанные участки, что может снизить эффективность использования ОП другими заданиями.

Рассмотрим программные запросы TS- и NTS-мониторов, используемые в программах реального времени. Для работы с регистрами ВУ без отображения на них виртуальных адресов программы в TS-мониторе служат команды .IOGET, .IOPUT, .IOBTS и .IOBTC, а в системе NTS – соответствующие им макрокоманды .IOPEEK, .IOPOKE, .IOBIS и .IOBIC.

Макровызов .IOPEEK arg, addr позволяет считать содержимое регистра ВУ с адресом addr. Считываемое слово пересылается монитором в R0.

Макровызов .IOPOKE arg, addr, value пересылает значение value в регистр ВУ с адресом addr.

Макровызов .IOBIS arg, addr, value выполняет действия, аналогичные команде процессора BIS, т. е. устанавливает отдельные разряды в регистре ВУ с адресом addr. Значение value определяет, какие именно разряды подлежат установке.

Макровызов .IOBIC arg, addr, value выполняет действия, аналогичные команде процессора BIC, т. е. очищает разряды в регистре ВУ с адресом addr. Значение value определяет, какие именно разряды подлежат сбросу.

Рассмотрим простой пример. Пусть АЦП подключен к системной магистрали и имеет адреса 167000 и 167002_8. Требуется принимать коды из АЦП в режиме ожидания готовности:

```
CSR = 167 000                    ; РКС АЦП
DBR = 167 002                    ; РД АЦП
ARG:    .BLKW 3                  ; Область для аргументов
START:  .IOBIS #ARG, #CSR, #1    ; Разблокировка АЦП
W:      .IOPEEK #ARG, #CSR       ; Считывание РКС в R0
        TSTB  R0                 ; Анализ бита 7
        BPL  W                   ; и ожидание его установки
        .IOPEEK #ARG, #DBR       ; Чтение кода из РД
```

```
.  ; Распределение по каналам
.  ; (код находится в R0)
BR W                    ; Повторить
```

Программирование ВУ упрощается, если предварительно отобразить последнюю виртуальную страницу программы с адресами 160000... 177776_8 на страницу ввода-вывода. Это делается с помощью программного запроса .IOMAP. Рассмотренный выше пример с АЦП преобразуется тогда следующим образом:

```
    .IOMAP          ; Изменение отображения
    INC  CSR        ; Разблокировка АЦП
W:  TSTB CSR        ; Ожидание
    BPL  W          ; готовности
    MOV  DBR, R0    ; Прием кода

    .  ; Распределение по каналам
    .

    BR   W          ; Повторить
```

Если необходимость в обращении к регистрам ВУ отпала, исходное отображение последней виртуальной страницы программы на RMON можно восстановить программным запросом .IOUNMAP (в TS-мониторе) или .RMMAP (в NTS).

Обработка прерываний от ВУ требует подключения ППЗ к вектору прерываний устройства. Для этого в TS-мониторе предусмотрена макрокоманда .INTCON, а в NTS – макрокоманда .INTCNT. Формат макрокоманды:

.INTCNT arg, vec, compl, prty

где arg – адрес четырехсловной области для аргументов; vec – адрес вектора прерываний; compl – входная метка ППЗ, предназначенной для обработки прерываний от данного ВУ; prty – приоритет ППЗ, который может иметь значение от 0 до 7.

Для ускорения обработки прерываний целесообразно зафиксировать программу в памяти, для чего в TS-мониторе используются макрокоманды .POLOCK и .IMLOCK. Первая фиксирует программу с предварительным перемещением ее в начало ОП, а вторая предназначена для немедленного фиксирования без перемещения. В NTS немедленное фиксирование осуществляет макрокоманда .JOBFIX, а фиксирование с перемещением – та же макрокоманда с параметром DOWN.

Прием кодов из АЦП в режиме прерываний в системе NTS может быть выполнен с помощью следующей программы:

```
.JOBFIX DOWN      ; Фиксирование с перемещением
.IOMAP            ; Отображение на страницу ввода-вывода
.INTCNT #ARG, #300, #COMPL, #1
MOV  #101, CSR    ; Разрешение прерываний и разблокировка входа
```

```
        ⋮ ; Продолжение основной программы
COMPL:  MOV  DBR, R0
        ⋮ ; Обработка кода
        RTS  PC
```

6.3. СТРУКТУРЫ ПРОГРАММНЫХ КОМПЛЕКСОВ С ОБРАБОТКОЙ ПРЕРЫВАНИЙ

Режим прерываний используется в измерительных комплексах в основном для управления ходом измерительно-вычислительного процесса, а также для приема данных из измерительной аппаратуры, если эти данные поступают относительно редко, случайно во времени или от нескольких источников. В первом случае источником прерываний служат сигналы от какой-то управляющей аппаратуры – таймера, контролирующих или управляющих элементов установки, а ПОП используется для изменения порядка выполнения программы. Если при этом отсчет времени ведется системным таймером, то обработка прерываний от него ведется на уровне ППЗ. Во втором случае прерывания поступают от измерительной аппаратуры, а в функции ПОП входят первичная обработка и накопление поступающих данных.

Рассмотрим примеры организации программных комплексов с обработкой прерываний управляющего характера. Пусть на некоторой установке выполняются длительные непрерывные измерения. Обычно аппаратура, входящая в установку, подвержена дрейфу: с течением времени изменяются напряжения источников питания, коэффициенты усиления усилителей и т. д. С целью учета дрейфа измерения периодически прерываются и выполняется процедура градуировки аппаратуры. Градуировочные коэффициенты сохраняются в области ОП, доступной программе, ведущей измерения. Каждый раз после приема очередной порции измерительных данных программа считывает градуировочные коэффициенты, выполняет с их помощью коррекцию поступивших данных и накапливает (или обрабатывает) их уже в таком подправленном виде.

Для периодического прерывания программы удобно воспользоваться системным таймером и аппаратом ППЗ. Пусть градуировку требуется осуществлять 1 раз/ч. Программный комплекс будет иметь следующую структуру:

```
ARG:    .BLKW 4              ; Область для аргументов
TIME:   .WORD 2, 137440      ; T = 1 ч
        .MRKT #ARG, #TIME, #COMP
          ⋮
```

```
COMP:   .MRKT #ARG, #TIME, #COMP
        .  ; Запуск процедуры градуировки
        .  ; Ожидание окончания градуировки
        .  ; Определение и сохранение градуировочных коэффициентов
        RTS     PC              ; Возврат к измерениям
```

Системный макровызов .MRKT в качестве одного из своих аргументов использует адрес двухсловного блока, содержащего значение времени в тактах (старшие разряды в первом слове, младшие – во втором). Интервал времени 1 ч состоит из $180000_{10}...537440_8$ тактов. Это число и записано в блоке с меткой TIME.

Основная программа, приступив к измерениям, выдает программный запрос .MRKT с указанием адреса ППЗ COMP. После этого основная программа продолжается. Через время, указанное в поле TIME, в данном случае через 1 ч, программа будет прервана и управление передано в ППЗ на метку COMP. Эта подпрограмма прежде всего снова выдает запрос .MRKT с теми же аргументами, а затем приступает к процедуре градуировки. После завершения градуировки управление возвращается в основную программу. Каждый час процесс градуировки повторяется.

Важно отметить, что в приведенном примере не требуется никаких специальных действий по организации взаимодействия основной программы в ППЗ. Действительно, все поля данных доступны и той, и другой программам. ППЗ периодически обновляет значения градуировочных коэффициентов, а основная программа считывает их в темпе поступления измерительных данных. Основной программе не требуется знать, произошло ли обновление градуировочных коэффициентов и когда именно: в любом случае ею будут прочитаны последние, наиболее новые значения. Таким образом, в данном случае основная программа и ППЗ работают параллельно и независимо.

Часто, однако, сигнал прерывания должен изменить ход основной программы. Пусть, например, основная программа осуществляет непрерывный вывод накапливаемых данных на графический дисплей. Сигнал от таймера, характеризующий окончание времени экспозиции, должен переключить основную программу на участок обработки накопленных данных. В тех случаях, когда основная программа носит циклический характер с небольшим периодом повторения, такое переключение можно осуществлять с помощью *флага* (*семафора*). Основная программа в каждом шаге цикла анализирует состояние флага, и если он оказывается установленным, переходит на выполнение соответствующего участка:

```
; Основная программа
FLAG:   .WORD 0         ; Программный флаг
START:  TST   FLAG      ; Проверка флага
        BNE   1 $       ; Флаг установлен!
        BR    2 $       ; Флаг не установлен, продолжить
```

```
1$:         JMP    PROCES ; Перейти на участок обработки
2$:
                 ; Продолжение основной программы
            JMP    START  ; Циклическое выполнение
PROCES:
                 ;Участок обработки данных
```

Двухступенчатый переход (сначала командой BNE на метку 1$, а затем командой JMP на метку PROCES) является обычным приемом, позволяющим передать управление на сколь угодно удаленную точку программы.

Переключение (установка) флага осуществляется в программе обработки управляющего прерывания, либо в ППЗ, если в качестве источника прерываний используется системный таймер:

```
; ПОП или ППЗ
COMP: MOV #1, FLAG  ; Установка флага
      RTS  PC       ; Возврат в основную программу
```

В тех случаях, когда основная программа не носит циклического характера, переключить ход ее выполнения можно с помощью программного запроса .SPCPS. Этот макровызов используется исключительно в ППЗ и позволяет принудительно изменить содержимое счетчика команд (и слова состояния процессора) при выходе из ППЗ, т. е. изменить точку возврата в основную программу. Формат макровызова: .SPCPS arg, addr, где arg – адрес двухсловной служебной области для размещения аргументов макрокоманды, а addr – адрес таблицы из трех слов, в первом и третьем словах которой указываются требуемый адрес перехода и новое ССП. После выполнения программного запроса второе слово таблицы будет содержать старый адрес возврата из ППЗ, а третье слово – старое значение ССП.

Пусть по сигналу прерывания от системного таймера требуется, прервав выполнение основной программы в произвольной точке, передать управление на участок обработки, начинающийся меткой PROCES. ППЗ будет иметь следующий вид:

```
COMP: .SPCPS #ARG, #SPCTBL
      RTS PC
```

В полях данных программы должна быть предусмотрена таблица для запроса .SPCPS, содержащая адрес перехода PROCES:

```
SPCTBL: .WORD PROCES  ; Новый адрес возврата
        .WORD 0       ; Здесь будет старый адрес возврата
        .WORD 0       ; Здесь будет старое ССП
```

Рассмотрим теперь примеры работы с прерываниями управляющего характера, поступающими от аппаратных средств установки. В простейшем случае управляющий сигнал является признаком того, что на выходе измерительной части установки имеются данные, которые надо

программно прочитать и обработать по заданному алгоритму. Если в ИВК входит аппаратура КАМАК, прием внешнего сигнала и преобразование его в сигнал прерывания можно осуществить с помощью регистра прерываний (регистра запросов), который вырабатывает сигнал L при поступлении на любой из его входов, число которых может достигать 24, внешнего сигнала. Для работы с несколькими управляющими сигналами в модуле предусмотрена возможность программного определения номеров входов, на которые поступили внешние сигналы.

Пусть по внешнему сигналу должна выполняться запись в ОП числа импульсов, поступивших к этому моменту в счетчик, причем результаты последовательных измерений накапливаются в ОП в виде массива. В регистрирующую часть установки входят наряду с прочим оборудованием регистр прерываний и модуль счетчиков, содержащий четыре независимых двоичных 16-разрядных счетчика. В конкретном эксперименте используются два счетчика (из четырех) для счета импульсов от двух датчиков и соответственно два входа регистра прерываний (из 24) для управления приемом данных. Данные от двух счетчиков должны накапливаться раздельно.

Для работы со счетчиками используются следующие команды КАМАК:

F (2) A (0) – чтение со сбросом счетчика 1;

F (2) A (1) – чтение со сбросом счетчика 2.

Управление регистром прерываний осуществляется с помощью следующих команд КАМАК:

F (1) A (14) – чтение состояния запросов;

F (11) A (12) – сброс регистра запросов;

F (19) A (13) – селективная установка разрядов маски запросов.

Будем считать, что модуль счетчиков установлен на станции 1, регистр прерываний – на станции 2, базовый адрес контроллера крейта составляет 164000_8, L – запрос от регистра прерываний маскируется разрядом 0 РЗМ и используется вектор прерываний 340_8.

Программа управления рассматриваемой частью установки имеет следующий вид:

```
CSR =164000                      ; РУС контроллера
N2A12=CSR+ ⟨ 2*40 ⟩ +⟨12.*2 ⟩    ; A (12) регистра прерываний
N2A13=N2A12+2                    ; A (13) – "–    – "–
N2A14=N2A13+2                    ; A (14) – "–    – "–
N1A0=CSR+ ⟨ 1*40 ⟩               ; A (0) модуля счетчиков
N1A1=N2A0+2                      ; A (1) – "–    – "–
DATA1:  .BLKW 1000.              ; Массив данных от счетчика 1
ADDR1:  .WORD DATA1              ; Указатель в нем
DATA2:  .BLKW 1000.              ; Массив данных от счетчика 2
ADDR2:  .WORD DATA2              ; Указатель в нем
START:  MOV #ISR, 340            ; Заполнение
        MOV #340, 342            ; вектора прерываний
        MOV #1000, CSR           ; Сигнал Z
```

```
        MOV  #19., CSR          ; Разблокировка запросов 1 и 2
        MOV  #3, N2A13          ; в регистре прерываний
        MOV  #1, CSR+2          ; Разблокировка L-запроса в РУС
        MOV  #101, CSR          ; Разрешение прерываний + F(1)
        .
        .; Продолжение основной программы
        .
; Программа обработки прерываний
ISR:    .INTEN 5                ; Переход в системное состояние
        MOV N2A14, R4           ; Чтение состояния запросов
        ROR R4                  ; Бит 0 состояния запросов в C
        BCS  3$                 ; Запрос 1 есть
1$:     ROR  R4                 ; Бит 1 состояния запросов в C
        BCS  4$                 ; Запрос 2 есть
2$:     MOV  #11., CSR          ; Сброс запросов
        TST  N2A12              ; в регистре запросов
        MOV  #101, CSR          ; Разрешение прерываний
        RTS  PC                 ; Выход из прерывания
3$:     MOV  ADDR1, R5          ; Указатель в массиве
        MOV  N1A0, (R5)+        ; Данное из счетчика 1 в ОП
        MOV  R5, ADDR1          ; Восстановление указателя
        BR   1$                 ; На проверку запроса 2
4$:     MOV  ADDR2, R5          ; Данное
        MOV  N1A1, (R5)+        ; из счетчика 2
        MOV  R5, ADDR2          ; в память
        BR   2$                 ; На выход
```

Основная программа выполняет инициализацию установки и затем непосредственного участия в процессе измерений не принимает. Приход сигнала от измерительной установки вызывает прерывание основной программы и переход на ПОП, которая выполняется в системном состоянии монитора. Прежде всего программа должна выяснить, какой сигнал вызвал прерывание. Для этого читается содержимое регистра состояния запросов модуля регистра прерываний, в котором по условию задачи могут быть установлены только два бита – 0 и 1. Командой циклического сдвига вправо бит 0 выдвигается в разряд *C* РСП, где и анализируется командой BCS. Если бит 0 установлен, происходит чтение данного из регистра 1 в соответствующий массив в ОП. В любом случае затем проверяется бит 1 слова состояния запросов, поскольку оба запроса могли прийти одновременно. После чтения данного из счетчика 2 выполняется сброс запросов в регистре прерываний, восстановление бита разрешения прерываний в РУС и выход из ПОП в основную программу командой RTS PC. То же происходит и при отсутствии запроса 2.

Размер массивов для приема данных (по 1000 слов) выбран условно; в приведенной программе опущен также счет принятых кодов и завершение измерений после накопления достаточного объема информации (или каким-либо другим способом).

Программный комплекс составлен таким образом, что все действия по приему и накоплению информации выполняются на уровне прерываний. В результате в течение измерений ЦП практически свободен и основную программу можно использовать для любой вычислительной работы. С другой стороны, завершение измерений никак не отразится на ходе основной программы, хотя в действительности должно, видимо, привести к каким-то программным действиям. Здесь возможны два случая.

Часто основная программа выполняет в процессе измерений некоторую вспомогательную работу: динамическое формирование и вывод графического кадра по частичным результатам измерений либо периодический контроль и индикацию состояния установки. При завершении измерений эти действия, очевидно, должны быть прекращены, а программа должна перейти к обработке (в широком смысле этого слова) всей накопленной информации. Для реализации такого переключения основной программы по внешнему событию можно воспользоваться программным запросом .SPCPS, позволяющим произвольно изменять адрес возврата из ППЗ. Однако запрос .SPCPS может быть выдан только в ППЗ, поэтому предварительно следует перевести ПОП на уровень ППЗ программным запросом .SYNCH.

Пусть в приведенной выше ПОП ведется счет числа поступающих кодов и при накоплении заданного объема данных происходит переход на метку 5$. Завершающая часть ПОП будет выглядеть следующим образом:

```
5$ : MOV     #11., CSR        ; Сброс
     TST     N2A12            ; регистра запросов
     .SYNCH  #SYNBLK          ; Переход на уровень ППЗ

     .SPCPS  #ARG, #SPCAD     ; Изменение адреса возврата
     RTS     PC               ; Выход из ППЗ по новому адресу
```

При этом в программе должны быть предусмотрены следующие поля данных:

```
ARG:     .BLKW  2
SYNBLK:  .WORD  0, 0, 0, 0, 0,-1,0
SPCAD:   .WORD  FINISH, 0, 0
```

Метка FINISH в блоке для запроса .SPCPS характеризует требуемую точку входа в основной программе.

Проще обстоит дело в том случае, когда завершение измерений не должно необратимо изменить ход основной программы, а только требует выполнения некоторых относительно продолжительных действий (передача накопленных данных на внешние носители, очистка массивов и т. д.). Все эти действия можно выполнить на уровне ППЗ, где, как известно, разрешено обращение к любым функциям монитора (ввода-

вывода, запросов к системному таймеру и проч.). В этом случае заключительные строки ПОП будут выглядеть следующим образом:

```
5$ : MOV     #11., CSR
     TST     N2A12
     .SYNCH  #SYNBLK
       .
       .; Обработка накопленных данных
       .
     RTS     PC
```

Последняя команда RTS PC возвращает управление в ту точку основной программы, где она была прервана последним внешним сигналом, поступившим на вход регистра прерываний.

Другой типичный алгоритм измерительного процесса заключается в том, что сигнал от установки немедленно или спустя заданный промежуток времени разрешает накопление измерительной информации (например, с помощью АЦП), причем это разрешение действует также в течение определенного интервала времени. Такие *временные окна* часто используются при исследовании нестационарных процессов различной природы.

Предположим, что внешний сигнал периодически запускает аппаратный таймер (выполненный в стандарте КАМАК или подключенный непосредственно к системной магистрали), причем время выдержки таймера определяет длительность измерений (экспозиции). Если время экспозиции невелико, а частота поступления регистрируемых событий значительна, принимать коды от АЦП целесообразно в режиме ожидания готовности. По окончании измерений требуется обработать накопленные данные. Программную реализацию описанного алгоритма можно выполнить различными способами, различающимися степенью занятости ЦП.

Первый способ заключается в том, что и накопление информации, и ее обработка выполняются в основной программе, на *уровне задания* (в отличие от уровня прерываний). В этом случае (рис. 6.2, *а*) основная программа после инициализации установки входит в бесконечный цикл ожидания флага измерений. Внешний сигнал, поступивший, например, на вход регистра прерываний, инициирует переход на ПОП запуска, в которой запускается аппаратный таймер и устанавливается флаг измерений. Выход из ПОП возвращает программу в цикл ожидания флага измерений, но поскольку флаг теперь установлен, программа выходит из цикла ожидания и приступает к приему данных из АЦП. При этом в каждом цикле опроса разряда готовности РКС АЦП выполняется также опрос флага обработки, пока находящегося в сброшенном состоянии. Этот процесс длится до завершения заданного интервала времени, когда прерыванием от таймера инициируется ПОП таймера, устанавливающая флаг обработки и возвращающая управление в основную программу в цикл ожидания готовности АЦП и флага обработки. Поскольку флаг

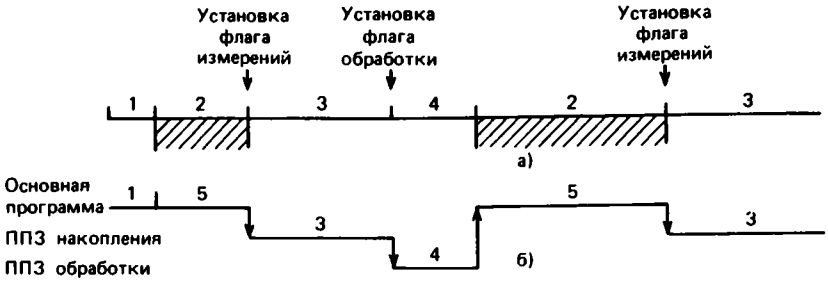


Рис. 6.2. Диаграммы работы программного комплекса с прерываниями управляющего характера:

а – работа на уровне задачи; *б* – работа на уровне ППЗ: *1* – инициализация установки; *2* – ожидание флага измерений (ЦП занят в цикле ожидания); *3* – прием данных от АЦП с опросом флага обработки; *4* – обработка накопленных данных; *5* – ЦП свободен и может выполнять любую работу

теперь установлен, программа переходит на блок обработки накопленных данных. В конце этого блока сбрасываются оба флага, и программа возвращается в цикл ожидания флага измерений. Легко видеть, что в том случае, когда период внешних сигналов запуска заметно превышает сумму времен измерений и обработки, значительная доля процессорного времени (заштрихованная на рис. 6.2, *а*), будет затрачена на ожидание флага измерений. Освободить время ЦП можно переведя процессы измерения и обработки на уровень ППЗ. В этом варианте программный комплекс будет работать следующим образом (рис. 6.2, *б*).

Основная программа после инициализации установки начинает выполнять запланированную для нее работу. Внешний сигнал инициирует переход на ПОП запуска, в которой, как и раньше, запускается таймер. Однако после этого ПОП не завершается, а с помощью программного запроса .SYNCH переходит на уровень ППЗ (ППЗ накопления) и приступает к опросу АЦП и приему измерительной информации. Как и в предыдущем примере, в цикл ожидания готовности АЦП включена проверка состояния флага обработки (пока сброшенного). Завершение заданного временного интервала инициирует ПОП таймера, которая прерывает выполнение ППЗ накопления. В ПОП таймера устанавливается флаг обработки, а затем эта ПОП в свою очередь программным запросом .SYNCH (со своим блоком аргументов) переводится на уровень ППЗ. Планировщик заданий ставит новую ППЗ (обработки) в очередь за только что выполнявшейся и еще не завершенной ППЗ накопления. Поскольку флаг измерений теперь установлен, в первом же цикле опроса готовности АЦП ППЗ накопления выходит из этого цикла и завершается командой RTS PC. Это приводит к запуску ППЗ обработки, в которой выполняется вся необходимая обработка и сброс обоих флагов. После

окончания обработки управление передается основной программе, которая будет выполняться вплоть до прихода следующего сигнала запуска.

Таким образом, использование ППЗ может быть весьма эффективным с точки зрения повышения производительности ВС.

Рассмотрим теперь некоторые особенности приема измерительной информации в режиме прерываний. Структура программного комплекса в этом случае зависит от того, требуется ли обрабатывать каждое регистрируемое событие в отдельности или обработке подвергаются пачки событий. В первом случае часто используется *кольцевой буфер*, во втором – *накопительный*.

Кольцевой буфер представляет собой область ОП, куда информация заносится программой обработки прерываний в темпе ее поступления из измерительной установки, а считывается основной программой в темпе, диктуемом скоростью обработки этой информации. Средняя скорость поступления данных в буфер не должна превышать средней скорости их обработки, иначе буфер быстро заполнится и данные начнут теряться. Однако наличие промежуточного буфера устраняет потери данных при кратковременных изменениях скорости их поступления, связанных, например, со случайным характером исследуемого процесса. Кольцевой буфер полезен также в тех случаях, когда время выполнения программы обработки не постоянно, а зависит, например, от конкретных характеристик обрабатываемого события.

Рассмотрим влияние емкости буфера k на характер регистрации событий, составляющих пуассоновский поток. Если время обработки каждого события (длительность выполнения программы обработки) равно T, то очевидно, что максимальная скорость обработки $m_{\text{макс}} = 1/T$ событий/с. При этом в случае регистрации периодических событий для $m \leqslant m_{\text{макс}}$ потери отсутствуют и скорости поступления и обработки событий совпадают: $n = m$. Однако для событий, распределенных во времени случайным образом, часть событий теряется и $m < n$. На рис. 6.3 приведены зависимости безразмерной величины mT от nT при разных значениях емкости буфера k. Без буфера ($k = 0$) n и m связаны известной зависимостью

$$m = \frac{n}{(1 + nT)}$$

и при $nT = 1$ потери составляют 50%. Введение буфера изменяет зависимость $m(n)$ и заметно увеличивает долю обрабатываемых событий. Так, при $k = 5$ потери составляют 1,03% для $nT = 0{,}8$ и 4,26% для $nT = 0{,}9$. Дальнейшее увеличение k снижает потери только в узком диапазоне $0{,}8 < nT < 1$. Таким образом, при регистрации пуассоновского потока событий достаточно использовать буфер емкостью 4–6 событий. При нестационарном входном потоке или переменном времени обработки может понадобиться буфер большей емкости.

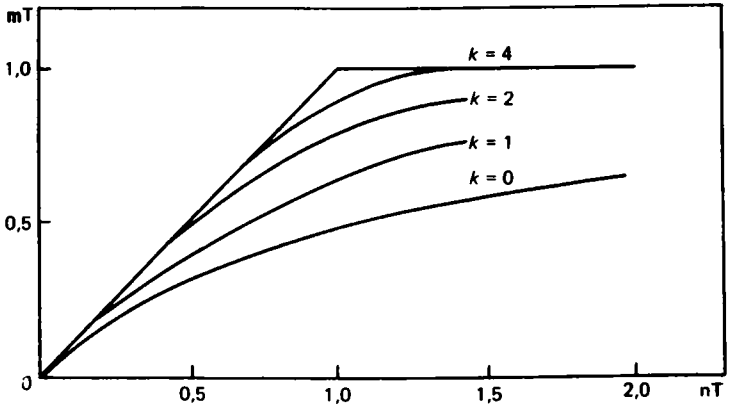


Рис. 6.3. Влияние кольцевого буфера на скорость накопления событий

Кольцевой буфер содержит собственно область буфера, систему указателей и счетчик заполнения буфера (рис. 6.4). Область буфера состоит из полей. Размер каждого поля (байт, слово или несколько слов) определяется объемом информации, принимаемой при каждом прерывании. Указатели – это ячейки ОП, в которых хранятся адреса полей буфера. В указателе PUT хранится адрес первого свободного поля, в указателе GET – адрес того поля, из которого надо считывать информацию. В начале работы, когда буфер пуст, оба указателя – и PUT, и GET, указывают на первое поле буфера.

При поступлении в буфер первого события оно записывается по адресу, находящемуся в PUT, после чего этот адрес увеличивается на число байтов в поле, указывая опять на первое свободное поле. В счетчик заполнения буфера записывается 1. Каждое последующее событие записывается в очередное поле, смещает указатель PUT и увеличивает на 1 содержимое счетчика заполнения.

Программа, считывающая и обрабатывающая информацию из буфера, следит за состоянием счетчика заполнения. Если содержимое счетчика отличается от нуля, это означает, что в буфере есть информация. В этом случае считывается поле, адрес которого находится в указателе GET, этот адрес увеличивается на размер поля, а число в счетчике заполнения уменьшается на 1. После обработки полученной порции информации считывается следующее поле и так далее до тех пор, пока содержимое счетчика заполнения не сделается равным 0. После этого программа обработки продолжает опрашивать счетчик заполнения, ожидая поступления информации в буфер. На рис. 6.5 изображено состояние буфера после того, как в него пришли четыре события, два из которых считались программой обработки.

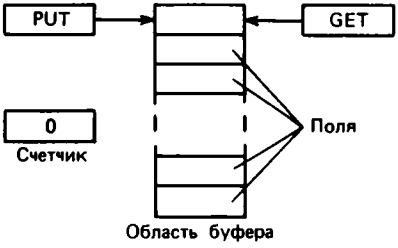


Рис. 6.4. Структура кольцевого буфера

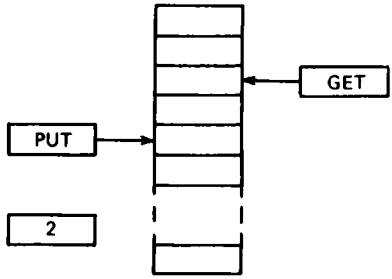


Рис. 6.5. Состояние кольцевого буфера в некоторый момент времени

Указатель PUT, перемещаясь в процессе записи информации по буферу, доходит наконец до конца буфера. В этом случае при регистрации очередного события адрес в указателе должен не увеличиться, а, наоборот, уменьшиться на длину буфера. Тем самым указатель возвращается в начало буфера, после чего продолжает перемещаться по буферу до его конца, опять возвращается в начало и так далее по кольцу. Аналогичные преобразования выполняются и с указателем GET.

При кратковременном повышении скорости поступления событий буфер начинает заполняться быстрее, чем очищаться. Если указатель PUT, перемещаясь по буферу, дошел до указателя GET с обратной стороны, возникает ситуация переполнения буфера, при которой дальнейшая запись информации в буфер должна быть заблокирована до освобождения хотя бы одного поля в буфере. Естественно, вся поступающая в это время информация теряется.

Рассмотрим пример программного комплекса с накоплением событий в кольцевом буфере. Пусть данные, подлежащие обработке, поступают в ЭВМ из модуля входных регистров КАМАК, куда они записываются из установки при регистрации ею внешнего события. Модуль содержит два 24-разрядных регистра. Запись в любой из них приводит к возникновению соответствующего *LAM*-требования, инициирующего *L*-запрос и прерывание в ЭВМ. В рассматриваемом примере в регистры из установки поступают 16-разрядные коды, причем оба регистра заполняются данными одновременно, поэтому для инициирования прерывания можно использовать любое из *LAM*-требований. Программный комплекс состоит из основной программы, которая после инициализации прерываний начинает выбирать данные из кольцевого буфера и обрабатывать их по заданному алгоритму, и ПОП, переносящей данные из модуля КАМАК в кольцевой буфер.

Программные блоки комплекса, имеющие отношение к кольцевому буферу, приведены ниже.

```
;Поля данных программ
RING:    .BLKW 8.*2          ; Кольцевой буфер, 8 полей, 16 слов
PUT:     .WORD RING          ; указатель записи в RING
GET:     .WORD RING          ; Указатель чтения из RING
CNTR:    .WORD 0             ; Счетчик заполнения RING
BUF:     .BLKW 2             ; Промежуточный буфер
ARG:     .BLKW 5             ; Область для аргументов макрокоманд
; Символические обозначения
CSR = 164000                 ; РУС контроллера КАМАК
N = 5                        ; Модуль установлен на станции 5
A0 = 0                       ; Субадрес 0
A1 = 1                       ; Субадрес 1
A0M305 = CSR + ⟨ N * 40 ⟩ + ⟨A0 * 2⟩
A1M305 = CSR + ⟨ N * 40 ⟩ + ⟨A1 * 2⟩
; Инициализация аппаратуры и прерываний
START:   MOV   #1000,CSR     ; Сигнал Z
         MOV   #INTR, 300    ; Заполнение
         MOV   #340, 302     ; вектора прерываний
         MOV   #2, CSR+2     ; Установка маски L-запросов
         MOV   #26., CSR     ; F (26)
         TST   A0M305        ; Разблокировка L-запросав модуле
         MOV   #100, CSR     ; Разрешение прерываний + F (0)
; Чтение данных из кольцевого буфера
START1:  TST   CNTR          ; Опрос счетчика заполнения RING
         BEQ   START1        ; Буфер пуст
         CMP   GET, #RING+32.; GET вышел за пределы буфера?
         BNE   GETEV         ; Нет
         MOV   #RING, GET    ; Возврат GET на начало буфера
GETEV:   MOV   GET, R2       ; Адрес поля в R2
         MOV   #BUF, R0      ; Адрес промежуточного буфера в R0
         MOV   (R2)+, (R0)+  ; Передача данных из RING
         MOV   (R2)+, (R0)+  ; в BUF
         MOV   R2, GET       ; Новое значение указателя GET
         DEC   CNTR          ; Одно поле из RING прочитано
; Обработка данных в BUF
         JMP   START1        ; Одно событие обработано
; Программа обработки прерываний
INTER:   .INTEN 4            ; Переход в системное состояние
         CMP   CNTR, #8.     ; Проверка на переполнение буфера
         BNE   NOTFUL        ; Не полон
         MOV   #500, CSR     ; Сигнал C + разрешение прерываний
         RTS   PC            ; Выход из ПОП при переполнении буфера
NOTFUL:  CMP   PUT, #RING+32 ; PUT вышел за пределы RING?
         BNE   INRING        ; Нет
         MOV   #RING, PUT    ; Возврат PUT на начало буфера
INRING:  MOV   PUT, R4       ; Адрес поля в R4
         MOV   A0M305, (R4)+ ; Чтение данных
         MOV   A1M305, (R4)+ ; из модуля в RING
         INC   CNTR          ; Еще одно событие
```

```
MOV   R4, PUT        ; Новое значение PUT
MOV   #100, CSR      ; Разрешение прерываний + F (0)
RTS   PC             ; Нормальный выход из ПОП
```

После инициализации прерываний основная программа приступает к циклическому опросу ячейки CNTR в ожидании поступления в кольцевой буфер хотя бы одного события. Как только содержимое счетчика переполнения оказывается больше нуля, программа начинает процедуру чтения очередного поля. Прежде всего проверяется, не вышел ли указатель GET при последнем чтении за пределы буфера. Если это произошло, в GET заносится адрес начала буфера. Далее данные из очередного поля кольцевого буфера переносятся в промежуточный буфер BUF, новое (увеличенное на 4) значение указателя заносится в GET и уменьшается на 1 содержимое счетчика заполнения кольцевого буфера. После завершения процедуры чтения программа приступает к обработке полученных из кольцевого буфера данных. Обработка завершается возвратом в цикл опроса ячейки CNTR.

Процедура записи в кольцевой буфер, выполняемая ПОП, схожа с процедурой чтения. После перехода в системное состояние программным запросом .INTEN, выполняется проверка буфера на переполнение. Если к моменту очередного прерывания счетчик заполнения показывает, что в кольцевом буфере уже находится 8 событий, данные из регистров КАМАК не могут быть перенесены в ОП. В этом случае генерируется сигнал *C*, сбрасывающий регистры КАМАК, разрешаются прерывания в РУС контроллера и ПОП завершается. Если же буфер не полон, выполняется процедура записи данных в кольцевой буфер. Прежде всего программа проверяет, не вышел ли указатель PUT при последней записи за пределы буфера. Если это произошло, в PUT заносится адрес начала буфера. Далее данные из регистров модуля КАМАК последовательно переносятся в кольцевой буфер, число в CNTR увеличивается на 1, а в PUT заносится новое значение указателя (на 4 больше предыдущего). ПОП завершается повторным разрешением прерывания (контроллер КАМАК сбрасывает разряд 6 РУС после выполнения процедуры прерывания).

Как уже отмечалось, кольцевой буфер полезен в тех случаях, когда события обрабатываются по отдельности. Если же обработке подвергаются пачки событий, для их накопления используются накопительные буфера. Чтобы избежать потерь событий в течение времени обработки накопленных в буфере данных, предусматривают два буфера. Пока идет обработка информации, накопленной в первом буфере, данные из аппаратуры поступают во второй. После заполнения второго буфера информация в нем начинает обрабатываться, а данные из аппаратуры опять направляются в первый буфер и т. д. Емкость буфера выбирают исходя из того, какой объем информации подлежит обработке.

Рассмотрим простой случай, когда обработка данных заключается в записи их на магнитный диск. Обычно в малых ЭВМ обмен данными

с магнитным диском осуществляется блоками по 256 слов, поэтому перед передачей на диск данные необходимо сгруппировать в пачки по 256 слов, для чего удобно использовать накопительные буфера. Программный комплекс с накопительными буферами, обслуживающий установку из предыдущего примера, может выглядеть следующим образом:

```
; Поля данных программы
B1:     .BLKW 256.                  ; Буфер 1
B2:     .BLKW 256.                  ; Буфер 2
PUT:    .WORD B1                    ; Указатель записи в буфер
CNT:    .WORD 0                     ; Счетчик заполнения буферов
FILE:   .RAD50  /MX1MYFILEDAT/      ; Имя файла для данных
; Обозначения
CSR=164000                          ; Адрес РУС
A0M305 = CSR + ( 5 * 40 )           ; Адрес регистра 0
A1M305 = A0M305 + 2                 ; Адрес регистра 1
; Инициализация аппаратуры и прерываний
START:  MOV     #1000, CSR          ; Сигнал Z
        MOV     # INTR,  300       ; Заполнение
        MOV     #340, 302           ; вектора
        MOV     #2, CSR + 2         ; Маска L-запросов
        MOV     #26., CSR           ; F (26)
        TST     A0M305              ; Разблокировка L-запроса
        .ENTER #ARG, #0, #FILE, #-1
        CLR     R2                  ; Счетчик блоков
; Чтение буфера B1
ST1     CMP     CNT, #256.          ; Ожидание
        BNE     ST1                 ; заполнения B1
        CLR     CSR                 ; Запрет прерываний в РУС
        CLR     CNT                 ; Сброс счетчика заполнения
        MOV     # B2, PUT          ; Переключение указателя на B2
        MOV     #100, CSR           ; Разрешение прерываний
        .WRITW #ARG, #0, #B1, #256., R2
        INC     R2                  ; Счетчик блоков
; Чтение буфера B2
ST2:    CMP     CNT, #256.          ; Ожидание
        BNE     ST2                 ; заполнения B2
        CLR     CSR                 ; Запрет прерываний в РУС
        CLR     CNT                 ; Сброс счетчика заполнения
        MOV     #B1, PUT            ; Переключение указателя на B2
        MOV     #100, CSR           ; Разрешение прерываний
        .WRITW #ARG, #0, #B2, #256., R2
        INC     R2                  ; Счетчик блоков
        JMP     ST1                 ; На чтение буфера B1
; Программа обработки прерываний
INTR:   .INTEN  4                   ; Переход в системное состояние
        CMP     CNT, #256.          ; Проверка на переполнение
        BNE     NOTFUL              ; Не полон
        MOV     #500, CSR           ; Буфер полон, сброс данных и
        RTS     PC                  ; завершение ПОП
```

```
NOTFUL: MOV    PUT, R4          ; Указатель записи в буфер
        MOV    A0M305, (R4) +   ; Запись данных
        MOV    A1M305, (R4) +   ; в буфер
        MOV    R4, PUT          ; Новое значение указателя
        INC    CNT              ; Еще одно событие
        MOV    #100, CSR        ; Разрешение прерываний
        RTS    PC               ; Переход в основную программу
```

В полях данных основной программы предусмотрены два буфера B1 и B2 по 256 слов каждый, указатель записи в них, а также счетчик заполнения буферов CNT.

В основной программе инициализируются прерывания и создается на диске файл с именем MX1 : MYFILE.DAT. Значение последнего аргумента макрокоманды .ENTER указывает, что для файла надо отвести свободную область максимального размера. Далее программа циклически опрашивает счетчик CNT в ожидании накопления в буфере B1 256 событий. После заполнения буфера производится сброс счетчика, смещение указателя записи на начало буфера и переключение адреса буфера в ячейке PUT. Все эти действия выполняются при запрещенных прерываниях. Далее содержимое буфера B1 переписывается в нулевой блок файла, выполняется инкремент номера блока (в регистре R2) и программа переходит на цикл ожидания заполнения буфера B2. После заполнения B2 все описанные операции повторяются, но переключение происходит опять на буфер B1. Команда JMP ST1 образует бесконечный цикл накопления, который в реальной программе должен, очевидно, разрываться либо после накопления требуемого объема информации, либо каким-то внешним воздействием (команда оператора, срабатывание таймера и пр.).

В программе обработки прерываний выполняется проверка буфера на переполнение, и если к моменту прихода данного прерывания в буфере уже записаны 256 событий, регистры КАМАК очищаются и осуществляется выход из ПОП. В противном случае значение указателя заносится в R4, выполняется пересылка данных из регистров КАМАК в буфер, смещается (увеличивается на 4) значение указателя и добавляется 1 в счетчик заполнения.

Глава 7

ОРГАНИЗАЦИЯ ВВОДА-ВЫВОДА

7.1. СИСТЕМА ВВОДА-ВЫВОДА

В системе РАФОС существуют три разновидности ввода-вывода: *синхронный* (с ожиданием), *асинхронный* (без ожидания) и *с управлением по событию* (с подпрограммой завершения). Синхронную передачу организуют программные запросы .READW и .WRITW, асинхронную –

запросы .READ и .WRITE, а передачу по событию – запросы .READC и .WRITC. Эти программные запросы можно использовать для управления любыми ВУ: алфавитно-цифровым терминалом, АЦПУ, НМД, НМЛ и т. д. Кроме того, для связи с терминалом предусмотрена особая группа макрокоманд, куда относится уже упоминавшаяся макрокоманда .PRINT, организующая вывод на экран терминала строк текста, макрокоманды TTYOUT и .TTOUTR для посимвольного вывода на экран терминала и макрокоманды .TTYIN и .TTINR для посимвольного ввода знаков с клавиатуры терминала. Действие этих программных запросов будет описано в следующей главе.

Синхронный ввод-вывод, организуемый с помощью макрокоманд .READW и .WRITW, характеризуется блокированием программы, выдавшей запрос на передачу данных, до окончания передачи. Время, на которое программа приостанавливается, зависит от объема передаваемой информации и скорости работы ВУ. Например, ввод с магнитной ленты большого массива данных может потребовать нескольких минут. Все это время программа остается блокированной средствами ОС и не занимает времени процессора. Если запрос на синхронный ввод-вывод выдало оперативное задание, все это время будет отдано фоновому заданию. Как только передача данных полностью завершится, выполнение программы будет продолжено. Синхронный ввод-вывод используется в тех случаях, когда дальнейшее выполнение программы опирается на результат ввода-вывода.

Асинхронный ввод-вывод, организуемый с помощью макрокоманд .READ и .WRITE, характерен тем, что программа, выставив запрос на передачу данных, не ожидает конца передачи, а продолжает выполняться дальше. В этом случае параллельно идут два процесса: выполнение программы и передача данных. При этом программа будет периодически прерываться на время, необходимое для обслуживания драйвером очередного прерывания от ВУ. Такой режим используется в тех случаях, когда дальнейший ход программы не зависит от результатов передачи данных. Если, например, по ходу программы сформирован массив данных, который надо сохранить на магнитном диске, целесообразно, инициировав вывод информации на ВУ, продолжить выполнение программы.

Наконец, последняя разновидность – ввод-вывод с управлением по событию, для организации которого используются макрокоманды .READC и .WRITC, является развитием асинхронного ввода-вывода. В этом случае программа, выставив запрос на передачу данных, т. е. инициировав драйвер, продолжает выполняться дальше. Параллельно осуществляется передача данных. Как только передача данных полностью завершится, управление передается в подпрограмму завершения, точно так же, как это происходит при обработке временных запросов (см. гл. 6). Входная метка ППЗ указывается в виде одного из аргумен-

Рис. 7.1. Канал ввода-вывода

Смещение	Содержимое	
0	Слово состояния канала	
2	Физический номер начального блока файла	
4	Длина файла в блоках	
6	Число записанных блоков	
10	Номер устройства	Длина очереди

тов в макровызове ввода-вывода. При передаче управления ППЗ выполнение основной программы приостанавливается до тех пор, пока не будет выполнена ППЗ, которая имеет повышенный приоритет по сравнению с основной программой. Последней строкой ППЗ должна быть команда RTS PC, по которой происходит передача управления монитору. Монитор, в свою очередь, передает управление основной программе, и та продолжает свою работу. Для организации ввода-вывода монитор использует два типа структур данных – *каналы ввода-вывода* и элементы очереди ввода-вывода.

Канал (рис. 7.1) представляет собой таблицу из пяти слов, содержащую информацию о ВУ (или, более детально, о файле). Эта таблица создается и модифицируется монитором (и частично драйвером ВУ) и используется как средство взаимодействия различных системных программ, участвующих в организации ввода-вывода. Располагаются каналы в мониторе, в области ячеек с фиксированными смещениями, где зарезервировано место под 16 каналов.

При планировании операций ввода-вывода обычно достаточно знать номера каналов, закрепленных за файлами и устройствами. Ни содержимое каналов, ни их местонахождение программиста не интересуют. Однако при разработке интеллектуальных программ, активно взаимодействующих с вычислительной средой и самостоятельно определяющих условия и порядок своего выполнения, к содержимому каналов приходится обращаться. Анализируя содержимое канала, прикладная программа может, например, оценить, достаточна ли длина файла для размещения в нем экспериментальных данных, выяснить, на каком именно устройстве находится конкретный файл, определить степень заполненности файла информацией и т. д. Определенный интерес может представить также *слово состояния канала* (рис. 7.2), хранящееся в первой из пяти ячеек канала. Здесь фиксируются аппаратные ошибки (разряд 0), занятость канала (разряд 15), характеристики открытого файла (разряд 7) и другая полезная в некоторых случаях информация.

Канал связи с уже существующим на магнитном диске файлом открывается с помощью программного запроса .LOOKUP, среди аргументов которого указываются номер канала и адрес области, содержащей имя имеющегося файла:

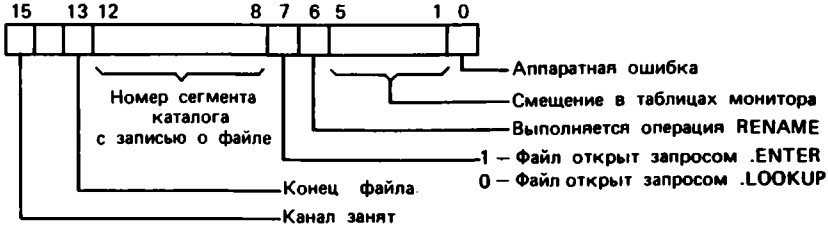


Рис. 7.2. Слово состояния канала

```
ARG:  .BLKW 3                      ; Область для аргументов
FILE: .RAD50 /DK 27JUNE005/        ; Полная спецификация файла
      .LOOKUP #ARG, #0, #FILE      ; Открыть канал 0
```

Макровызовом .LOOKUP открываются также каналы связи с устройствами нефайловой структуры, такими, как терминалы, АЦПУ, перфоленточные устройства ввода-вывода:

```
ARG:  .BLKW 3                      ; Область для аргументов
PRNT: .RAD50 /LP/                  ; Логическое имя АЦПУ
      .WORD 0                      ; Идентификатор конца имени
      .LOOKUP #ARG, #1, #PRNT      ; Открыть канал 1
```

Для создания на диске нового файла используется программный запрос .ENTER, в котором кроме упомянутых аргументов указывается размер создаваемого файла:

```
ARG: .BLKW 4                       ; Область для аргументов
F1:  .RAD50 /MX1MYFILEDAT/         ; Спецификация файла
     .ENTER #ARG, #2, #F1, #64.    ; Создать файл длиной 64 блока
```

Если операции ввода-вывода на данном канале завершены, канал можно закрыть программным запросом .PURGE с указанием номера закрываемого канала. После этого канал считается свободным и его можно снова использовать для связи с другим файлом или устройством: .PURGE #2

Следует иметь в виду, что новый файл, созданный с помощью программного запроса .ENTER, рассматривается системой как *временный.* Над ним можно выполнять любые операции ввода-вывода, но после завершения задания файл будет уничтожен. Это очень удобно при отладке программы; если же созданный файл требуется сохранить, канал связи с ним следует закрыть запросом .CLOSE (а не .PURGE). После этого файл делается *постоянным.* Удалить его (точнее, пометить несуществующим) можно программным запросом .DELETE:

```
ARG:  .BLKW 3
FILE: .RAD50 /MX1MYFILEDAT/
      .DELETE #ARG, #2, #FILE
```

С гибкого диска MX1 удаляется файл MYFILE.DAT. Второй аргумент макровызова определяет номер канала, используемого запросом .DELETE. Этот канал к моменту выполнения запроса .DELETE должен быть свободен.

Рассмотрим примеры, иллюстрирующие методику работы с описанными программными запросами.

При выводе на печать графиков и других изображений сначала формируется файл с закодированным графическим образом. Повторное использование программы формирования файла приводит к попытке создания на диске при каждом запуске программы файла с одним и тем же именем. Для того чтобы обеспечить успешное выполнение программы независимо от наличия или отсутствия на диске версии файла от предыдущего запуска, можно в начале программы уничтожать предыдущий файл:

```
.DELETE #ARG, #0, #FILE
.ENTER  #ARG, #0, #FILE, #2
```

Если на диске был файл с указанным в программе именем, он будет уничтожен запросом .DELETE и снова создан запросом .ENTER. Если же файла не было, запрос .DELETE не выполнится, а запрос .ENTER создаст файл заново.

Можно предложить другой вариант программы, в котором предыдущая версия файла не уничтожается, но программа по-прежнему успешно выполняется как при отсутствии на диске файла, так и при его наличии:

```
    .LOOKUP #ARG, #0, #FILE          ; Попытка открыть канал
    BCC     1$                       ; Файл уже имеется
    .ENTER  #ARG, #0, #FILE, #2      ; Файла не было, создать его
1$                                   ; Продолжение программы
```

Установленное состояние разряда *C* РСП после выполнения запроса .LOOKUP говорит о том, что указанный файл на диске отсутствует и канал связи с ним открыть не удалось. В этом случае выполняется запрос .ENTER, создающий файл заново. Такую методику можно рекомендовать при дефиците дискового пространства и работе с файлами большого размера. Дело в том, что программный запрос .ENTER создает новый временный файл и в том случае, если файл с таким именем уже имеется. Программный запрос .CLOSE, закрывающий канал связи с новым файлом и объявляющий этот файл постоянным, одновременно уничтожает старый файл с тем же именем (если только он не был перед этим защищен с помощью команды оператора PROTECT). Поэтому перед выдачей запроса .ENTER обычно нет необходимости проверять отсутствие файла. Если, однако, файл занимает значительную часть дискового пространства, на диске может не хватить места для одновременного существования двух версий файлов, и описанный прием оказывается полезным.

Как уже отмечалось, в мониторе выделено место под 16 каналов. Если программа использует более 16 файлов, требуется создать дополнительные каналы. Это можно выполнить различными способами.

Программный запрос .CDFN позволяет отвести место под каналы в прикладной программе, при этом каналы в мониторе перестают использоваться:

```
ARG: .BLKW 3                ; Область аргументов
CHN: .BLKW 5 * 50.          ; Область для каналов
     .CDFN #ARG, #CHN, #50; Создание 50 каналов
```

Всего таким образом можно создать до 255 каналов (с номерами от 0 до 254). Недостаток такого способа – нерациональное использование памяти.

Часто программа обращается к файлам последовательно. В этом случае одни и те же каналы можно использовать повторно, закрывая их после окончания работы с файлом программным запросом .CLOSE и открывая затем для работы с другим файлом с помощью запросов .ENTER или .LOOKUP. Однако эта методика не годится для работы с временными файлами, так как запросы .CLOSE будут преобразовывать их в постоянные, засоряя диск ненужной информацией. Для таких случаев и предусмотрен программный запрос .PURGE, который освобождает канал, не выполняя при этом никаких дополнительных операций. Временный файл, канал связи с которым закрыт запросом .PURGE, уничтожается, постоянный – остается.

Другой способ повторного использования канала заключается в сохранении его содержимого в отведенной для этого области памяти с помощью программного запроса .SAVESTATUS и восстановления, когда это потребуется, программным запросом .REOPEN. Такая методика допустима только для каналов, открытых запросом .LOOKUP:

```
F1:     .RAD50 /DK FIL001DAT/      ; Описание файла 1
F2:     .RAD50 /DK FIL002DAT/      ; Описание файла 2
CH1:    .BLKW 5                    ; Место для сохранения
CH2:    .BLKW 5                    ; двух каналов
START:  .LOOKUP #ARG, #0, #F1      ; Открыли канал 0 с файлом 1
        .SAVEST #ARG, #0, #CH1     ; Сохранили канал 0 в CH1 и закрыли
        .LOOKUP #ARG, #0, #F2      ; Открыли канал 0 с файлом 2
        .SAVEST #ARG, #0, #CH2     ; Сохранили канал 0 в CH2 и закрыли
          .
          .; Возможно использование канала 0
          .
        .REOPEN #ARG, #0, #CH1     ; Канал 0 связи с файлом 1
          .
          .; Работа с файлом 1
          .
        .PURGE #0                  ; Освободили канал 0
        .REOPEN #ARG, #0, #CH2     ; Канал 0 связи с файлом 2
```

```
        .
        .; Работа с файлом 2
        .
        . PURGE #0                    ; Освободили канал 0
```

В приведенном примере один и тот же канал 0 используется сначала для инициализации файлов 1 и 2 (программные запросы .LOOKUP), а затем для последовательной работы с этими файлами. Основное достоинство такого способа работы с большим количеством файлов заключается в том, что программные запросы .LOOKUP, выполняемые выгружаемым компонентом монитора USR, сосредоточиваются в начальной, инициализирующей части программы. Основная часть программы получает доступ к неограниченному числу файлов с помощью программных запросов .REOPEN и .PURGE, не требующих присутствия программы USR в ОП. Это дает возможность выгрузить USR из ОП, увеличив тем самым доступный объем памяти. Использование пары запросов .SAVESTATUS – .REOPEN удобно также в тех случаях, когда в процессе модификации какой-то сложной программы возникает необходимость вставить строки обращения к новому файлу, но программист не уверен, что в мониторе имеется свободный канал или во всяком случае не знает, какой именно канал свободен.

```
        MOV    R1, –(SP)              ; Сохранили R1
        MOV    #15., R1               ; 15 попыток с каналами 15...1
  1$ :.SAVEST #ARG, R1, #CH           ; Попытка сохранить канал в CH
        BCC    2 $                    ; Попытка удалась
        SOB    R1, 1 $                ; Попробовать следующий канал
        MOV    (SP) +, R1             ; Аварийный
        .EXIT                         ; выход
  2$: .LOOKUP #ARG, R1, #FILE         ; Открыли канал с новым файлом
        .
        .
        .
        .PURGE  R1                    ; Закрыли канал
        .REOPEN #ARG, R1, #CH         ; Восстановили прежний канал
        MOV    (SP) +, R1             ; Восстановили R1
        .
        .
        .
```

В этом фрагменте делаются последовательные попытки сохранить каналы с номерами 15...1. Анализ разряда *C* РСП после программного запроса .SAVESTATUS позволяет удостовериться в том, что канал (номер которого находится в R1) был сохранен. Возврат из монитора с установленным разрядом *C* свидетельствует о том, что либо данный канал был открыт запросом .ENTER (тогда .SAVESTATUS не действует), либо он не был открыт вообще. В любом из этих случаев номер канала уменьшается на 1 (командой SOB) и делается повторная попытка. После успешного сохранения какого-то канала он используется для связи с новым файлом, а после окончания работы с файлом закрывается запро-

сом .PURGE и вновь открывается уже с исходным содержимым, которое хранилось все это время в области СН. Если все каналы просмотрены, но ни одного сохранить не удалось, задание аварийно завершается.

Небольшое усовершенствование программы позволяет использовать и те каналы, которые вообще не были открыты:

```
1$: .SAVEST #ARG, R1, #CH     ; Попытка сохранить канал
    BCC     2$                ; Попытка удалась
    TSTB    52                ; Если код завершения
    BEQ     2$                ; равен 0, канал свободен
    SOB     R1, 1$            ; Канал открыт .ENTER
    .EXIT                     ; Аварийный выход
2$: .LOOKUP #ARG, R1, #FILE   ; Продолжение программы
```

Запрос .SAVESTATUS возвращает следующие коды завершения в байт 52 системной области связи (вместе с установкой разряда *C* РСП):

0 – канал не был занят, .SAVESTATUS не выполнен;

1 – канал был открыт посредством .ENTER, .SAVESTATUS не выполнен.

Анализ байта 52 в случае $C = 1$ позволяет перейти на строки работы с новым файлом, как только встретился свободный, не открытый ранее канал. Запрос .SAVESTATUS при этом не выполнится, но, поскольку канал свободен, его можно открыть для связи с новым файлом запросом .LOOKUP (или, если необходимо, .ENTER).

Второй тип структур данных, используемый монитором при организации ввода-вывода, – элемент очереди ввода-вывода.

Элемент очереди является передаточным звеном между программой пользователя и драйвером ВУ. Монитор, выполняя программный запрос на ввод-вывод (.READ, .WRITE и др.), заполняет первый свободный элемент очереди данными, взятыми из аргументов макрокоманды ввода-вывода и передает адрес этого элемента соответствующему драйверу, ставя элемент в очередь запросов ввода-вывода к данному ВУ.

Как только в очереди ввода-вывода появляется элемент очереди, монитор запускает драйвер ВУ, который непосредственно осуществляет передачу данных. Элемент ввода-вывода освобождается только после того, как драйвер полностью завершит свою работу.

Элемент очереди (рис. 7.3) представляет собой таблицу из семи слов, в которой содержится информация, необходимая драйверу для выполнения операций ввода-вывода. Каждый раз, когда приходит прерывание от ВУ, драйвер передает очередную порцию данных (обычно байт или слово). При этом драйвер модифицирует поля элемента очереди: адрес буфера пользователя и счетчик переданных слов. После того как весь запланированный объем данных передан (в счетчике слов 0) драйвер передает управление монитору, который анализирует последнее слово элемента очереди. Если выполнялся ввод-вывод с ожиданием, монитор разблокирует ожидающую программу и передает ей управление. Если

Смещение	Содержимое			
0	Адрес следующего элемента			
2	Адрес слова состояния канала			
4	Физический номер блока файла			
6		14 11 Номер задания	10 8 Номер устройства	7 0 Код специальной функции
10	Адрес буфера пользователя			
12	Счетчик слов: < 0 – операция записи > 0 – операция чтения = 0 – операция поиска			
14	0 – режим с ожиданием 1 – асинхронный режим		Четное число – адрес ППЗ	

Рис. 7.3. Элемент очереди ввода-вывода

же в последнем слове элемента содержится адрес ППЗ, монитор преобразует элемент очереди ввода-вывода в элемент очереди ППЗ и ставит его в очередь ППЗ (если, конечно, такая очередь имеется). Элемент очереди ППЗ будет занят до тех пор, пока эта программа не активизируется монитором. После активизации ППЗ элемент очереди освобождается и поступает в список свободных элементов.

Содержимое элемента очереди редко используется прикладным программистом. Необходимо только следить за тем, чтобы к моменту выдачи программой запроса на ввод-вывод или на отсчет интервала времени список свободных элементов не был исчерпан. Отсутствие свободного элемента приведет к прекращению выполнения задания до тех пор, пока элемент не появится (в результате завершения выполняемой операции ввода-вывода или истечения заданного интервала времени). Такая приостановка задания может быть фатальной, особенно если элементы заняты длительными временными запросами.

Все макровызовы ввода-вывода имеют схожий формат:

.PRREQ area, chan, buf, wcnt, crtn, blk

Здесь .PRREQ – обобщенное обозначение макрокоманд .READ, .READW, .READC, .WRITE, .WRITW и .WRITC; area – адрес 5-словной области для аргументов; chan – номер канал, используемого в данной операции ввода-вывода; buf – адрес буфера ввода-вывода в прикладной программе; wcnt – число передаваемых слов; crtn – адрес ППЗ операции ввода-вывода (используется только в запросах .READC и .WRITC; в остальных запросах на месте аргумента crtn указывается последний аргумент blk); blk – номер блока в файле, с которого надо начать запись или чтение данных.

Файлы на магнитных дисках состоят из целого числа блоков по 256 слов, нумеруемых с 0. Прочитать или записать данные на диск можно

только с начала блока. В аргументе blk может быть указан номер любого блока, входящего в файл, а в аргументе wcnt – любое количество слов (большее или меньшее одного блока).

Рассмотрим несколько примеров организации ввода-вывода.

Часто файлы с экспериментальными данными содержат наряду с числовой информацией еще и текстовую, служебную (дата и условия измерений, предварительные результаты, преобразованные в символьную форму с целью вывода на печать или на экран дисплея, и т. д.). Пусть на диске DK: имеется файл EXPDAT.DAT размером 17 блоков, причем в первых 16 блоках хранится числовая информация, а последний блок (с номером 16) является служебным. Требуется прочитать числовые данные в буфер BUF, а из служебного блока вывести на дисплей первые 80 символов:

```
ARG:    .BLKW 5                          ; Область для аргументов
BUF:    .BLKW 16.*256.                   ; Буфер ввода
TXT:    .BLKW 80.                        ; Буфер для текста
END:    0                                ; Конец текста для .PRINT
FIL:    .RAD50 /DK EXPDATDAT/            ; Спецификация файла
START:  .LOOKUP #ARG,#0,#FIL             ; Открыть канал 0
        .READW #ARG,#0,#BUF,#16.*256.,#0
        .READW #ARG,#0,#TXT,#40.,#16.
        .PRINT #TXT
```

Второй запрос на чтение (символьной информации) должен быть синхронным, в противном случае макровызов .PRINT начнет выводить содержимое буфера TXT еще до того, как буфер будет заполнен информацией с диска.

Поскольку запись на диск производится по блокам, данные, поступающие из измерительной установки, удобно накапливать в буфере ОП размером 256 слов, а после заполнения буфера переписывать на диск в очередной блок файла. При длительных измерениях возрастает вероятность машинного сбоя, который может привести к потере номера последнего записанного блока. В этом случае затруднительно организовать перезапуск задания с целью продолжения накопления информации. Для повышения надежности ИВК номер очередного блока можно хранить на диске в отдельном файле или в одном из блоков файла с данными. Строки программы, относящиеся к записи на диск очередного блока, могут иметь такой вид:

```
BUF:    .BLKW 256.                       ; Буфер с данными
BLK:    .BLKW 1                          ; Ячейка для номера блока
        .READW #ARG,#0,#BLK,#1,#0
        .WRITW #ARG,#0,#BUF,#256.,BLK
        INC    BLK
        .WRITW #ARG,#0,#BLK,#1,#0
```

Следует заметить, что номера блоков и дорожек, используемые в программе, представляют собой логические номера, определяемые относительно начала файла. Так файл размером 6144 слова содержит 24 блока с логическими номерами 0...23, размещаемыми на двух дорожках с логическими номерами 0 и 1. Физические номера блоков, определяемые относительно начала диска, как и физические номера цилиндров (дорожек), на которых они расположены, обычно программисту неизвестны.

Для реализации описанного алгоритма удобно воспользоваться аппаратом подпрограмм завершения операций ввода-вывода. Действительно, процедуру модификации–записи можно выполнить в ППЗ, активизируемую окончанием чтения, а процедуру сортировки–чтения выполнить в ППЗ, активизируемую окончанием записи. В этом случае обеспечивается автоматическая синхронизация выполнения программных процедур с вращением диска без какой-либо затраты времени ЦП. Уровень основной программы остается свободным, и при необходимости основной программе можно поручить какую-нибудь работу, не имеющую большой срочности, например вывод накапливаемого спектра на экран графического дисплея.

Для того чтобы первую же процедуру сортировки–чтения поставить на уровень ППЗ, можно использовать программный запрос на ввод-вывод к *нуль-устройству*, которое осуществляет фиктивные операции ввода-вывода без передачи информации. Нуль-устройство обслуживается программой *нуль-драйвера*, который работает по тем же правилам, что и любой другой драйвер. Запросами к нуль-драйверу удобно пользоваться, в частности, при отладке программ, содержащих ввод-вывод, если в процессе отладки нежелательно или невозможно реально осуществлять передачу данных. В нашем случае запрос .READC к нуль-драйверу позволяет перейти с уровня основной программы на уровень ППЗ.

Рассматриваемая программа регистрации спектра имеет следующую структуру:

```
BUF3K:  .BLKW 3072.                  ; Буфер модификации
FILE:   .RAD50 /DK 2DSP/             ; Спецификация
        .RAD50 /DAT/                 ; файла
BLKC:   0                            ; Текущий номер блока
BLKN:   22.*12.-12.                  ; Начальный блок последней дорожки
TRKN:   0                            ; Текущий номер дорожки
NUL:    .RAD50 /NL/                  ; Идентификатор
        0                            ; "нуль-устройства"
ARG:    .BLKW 5                      ; Область для аргументов
START:  .LOOKUP #ARG, #0, #FILE;     Канал 0 связи с файлом
        .LOOKUP #ARG, #1, #NUL;      Канал 1 связи с нуль-драйвером
        .READC #ARG, #1, #0, #0, #TDISK, #0
        .SPND                        ; Блокирование задания
```

```
TDISK:  MOV #-12., BLKC          ; Инициализация номера блока
        CLR TRKN                 ; Начинаем с дорожки 0
READ:   ADD #12., BLKC           ; Начальный блок очередной дорожки
        CMP BLKC, BLKN           ; Вышли из файла?
        BGT TDISK                ; Да, начать с начала
        .READC #ARG, #0, #BUF3K, #3072., #MODIF, BLKC
        CALL SORT                ; Вызов подпрограммы сортировки
        RTS PC                   ; Выход из ППЗ
MODIF:  CALL MOFIFC              ; Вызов подпрограммы модификации
        INC TRKN                 ; Следующая дорожка
        .WRITC #ARG, #0, #BUF3K, #3072., #READ, BLKC
        RTS PC                   ; Выход из ППЗ
```

В начале основной программы с помощью программных запросов .LOOKUP открываются каналы связи с файлом (канал 0) и с нуль-устройством (канал 1) и ставится запрос .READC к нуль-драйверу. В качестве ППЗ этого запроса фигурирует процедура сортировки–чтения TDISK. Далее основная программа блокируется программным запросом .SPND либо приступает к выполнению запланированных для нее операций.

Подпрограмма завершения TDISK устанавливает начальный номер блока файла (0) и начальный номер дорожки (0), после чего ставит запрос .READC на чтение 3К слов файла, начиная с указанного блока. Пока длится позиционирование и затем чтение (по каналу ПДП), программа продолжает работать, выполняя процедуру сортировки (подпрограмма SORT). После окончания сортировки ППЗ завершается и управление передается в основную программу.

После того как драйвер магнитного диска закончит физическую передачу данных в буфер BUF3K, монитор активизирует ППЗ MODIF. В этой ППЗ сначала вызывается подпрограмма модификации спектра в буфере BUF3K, а по окончании модификации ставится запрос на запись модифицированного буфера на ту же дорожку. После постановки запроса в очередь к драйверу ППЗ завершается с передачей управления основной программе. Поскольку диск вращается безостановочно, процедура модификации выполняется фактически одновременно с позиционированием (см. рис. 7.5, *а*).

Завершение записи на диск активизирует ППЗ READ (являющуюся частью ППЗ TDISK). В этой подпрограмме вычисляется номер начального блока следующей дорожки (каждая дорожка содержит 12 блоков) и выполняется проверка на выход из файла. Если перед этим была модифицирована последняя дорожка файла, происходит переход на метку TDISK (в рамках той же ППЗ) и программа начинает модификацию спектра сначала.

В зависимости от объема входного буфера сортировка может занимать разное время. Рассмотрим работу программы в условиях, когда

длительность сортировки превышает время позиционирования–чтения (рис. 7.5, *б*). ППЗ TDISK (или READ), поставив запрос на чтение, вызывает подпрограмму сортировки SORT. По окончании чтения монитор должен активизировать ППЗ MODIF, однако уровень ППЗ занят выполняемой подпрограммой. В результате ППЗ MODIF ставится в очередь ППЗ, и процедура модификации откладывается, как это показано на рис. 7.5, *б*, до окончания сортировки. Из-за того что подпрограмма SORT выполняется на уровне ППЗ, требование монитора на активизацию ППЗ MODIF не прерывает эту подпрограмму, а ожидает ее завершения, что и обеспечивает работоспособность алгоритма при любых соотношениях времен сортировки–чтения.

Несмотря на относительно сложный алгоритм взаимодействия ППЗ и запросов на ввод-вывод, программе не требуются дополнительные элементы очереди. Рассмотрим диаграмму использования элемента очереди по ходу выполнения программы (рис. 7.5, *в*). Запрос .READC использует элемент очереди ввода-вывода. Этот элемент занят вплоть до окончания операции ввода, после чего он преобразуется в элемент очереди ППЗ. В таком качестве элемент занят до окончания сортировки, когда активизируется ППЗ MODIF, а элемент освобождается. Запрос .WRITC опять использует тот же элемент в качестве элемента очереди ввода-вывода. По окончании записи элемент очереди ввода-вывода преобразуется в элемент очереди ППЗ и, поскольку ППЗ сразу же активизируется, элемент освобождается вплоть до запроса .READC.

Программные запросы на ввод-вывод, так же как и остальные запросы к монитору, устанавливают разряд *C* РСП в случае каких-либо ошибок, обнаруженных при их выполнении. В байт 52 системной области связи в этом случае заносится код ошибки. Следует, однако, иметь в виду, что ошибка программного запроса на ввод-вывод отнюдь не эквивалентна ошибке физической передачи данных. Действительно, успешное выполнение запроса на ввод-вывод лишь означает, что в очередь к соответствующему драйверу поставлен элемент очереди ввода-вывода. Диагностика правильности выполнения запроса на ввод-вывод относится именно к этому моменту. Сама передача данных начнется спустя какое-то (в отдельных случаях значительное) время, и анализ возможных ошибок передачи данных можно будет провести только после ее завершения. Поскольку окончание ввода-вывода – событие асинхронное, не связанное с ходом выполняемой программы, анализ ошибок ввода-вывода следует проводить в подпрограмме завершения, где можно обратиться к слову состояния канала и проверить отсутствие аппаратной ошибки либо ситуации *конец файла* (см. рис. 7.2).

При выполнении программных запросов на *ввод* монитор анализирует аргументы wcnt и blk с учетом длины файла, полученной из третьего слова канала (см. рис. 7.1). Если начальный номер блока, указанный в аргументе blk, выходит за пределы файла, устанавливается разряд *C* РСП и управление возвращается в программу с кодом завершения 0 и очи-

щенным регистром R0. Разряд *C* РСП устанавливается также в тех случаях, когда указанный в запросе на ввод канал не был открыт, либо в канале уже зафиксирована аппаратная ошибка устройства. В первом случае код завершения равен 2, во втором 1.

Если начальный номер блока не выходит за пределы файла, бит *C* не устанавливается, даже если сделан запрос на чтение большего числа слов, чем содержится в файле. В этом случае в регистр R0 заносится число слов, которые реально будут прочитаны. Сравнивая содержимое R0 с аргументом wcnt, можно установить, правильно ли был сделан запрос.

При выполнении программных запросов на *вывод* разряд *C* РСП устанавливается не только если начальный номер блока, указанный в аргументе blk, выходит за пределы файла, но и во всех случаях, когда делается попытка записи за пределами файла (wcnt превышает длину участка файла от blk до конца файла). Код завершения в этих случаях равен 0, а в регистр R0 помещается число слов, которые реально могут быть записаны в файл. Коды завершения 1 и 2 устанавливаются так же, как и при запросах на ввод.

7.2. СТРУКТУРА И ФУНКЦИОНИРОВАНИЕ ДРАЙВЕРОВ ВНЕШНИХ УСТРОЙСТВ

Драйвер – это системная программа, выполняющая передачу массивов данных между ВУ и буфером в программе пользователя. Несмотря на различия, связанные со спецификой конкретных ВУ, все драйверы сходны по структуре и принципам функционирования. Характерным для драйверов является:

передача порций данных (слов, байтов) по прерываниям от ВУ;

использование системных структур – каналов и элементов очереди ввода-вывода;

единообразное обращение из программы с помощью запросов .READ или .WRITE и их разновидностей;

постоянное взаимодействие с монитором в процессе своей работы и как следствие этого жесткие требования к структуре.

В состав ОС обычно включаются райверы для всех стандартных ВУ – терминала, магнитных дисков и лент, АЦПУ и др. Кроме того, разрабатываются драйверы для многих специальных применений ВС.

Драйвер системы виртуального доступа локальной вычислительной сети (ЛВС) позволяет работать с периферийным оборудованием удаленных ЭВМ, входящих в сеть, как со своими собственными, не вдаваясь в тонкости протокола сети и технических особенностей средств связи ЭВМ с сетью.

Драйверы графических устройств упрощают подготовку и вывод графической информации, избавляя программиста от необходимости обращаться к регистрам устройств на физическом уровне.

Драйвер *электронного диска* (*квазидиска*) дает возможность использовать в составе ВС расширенную память практически любого объ-

ема, записывая в нее информацию, как и на обычный магнитный диск, в виде файлов. Перенос на электронный диск программ и данных пользователя существенно (в 5...10 раз) повышает скорость их обработки и уменьшает нагрузку на НГМД, увеличивая срок их службы. Другой аспект применения электронного диска – использование его в качестве системного устройства для хранения в течение сеанса работы на ЭВМ монитора ОС и других системных программ. В этом случае перед началом работы необходимый состав системных средств копируется с НГМД на электронный диск, соответствующей командой монитора электронный диск назначается системным устройством и в дальнейшем работа ЭВМ осуществляется под управлением копии ОС, находящейся на электронном диске. Помимо заметного ускорения работы такой режим позволяет эксплуатировать ВС с дисковой ОС вообще без НМД, что может быть удобно, например, в полевых условиях. Электронный диск для таких применений должен быть сконструирован на базе энергонезависимого ЗУ (либо снабжен батарейным питанием). Возможна также начальная загрузка ОС на электронный диск по линии связи с удаленной ЭВМ.

Программное обращение к электронному диску при наличии драйвера ничем не отличается от обращения к НМД или НГМД – изменяется только имя устройства (например, EX: вместо DX:). При этом оснащение ВС электронным диском не затрагивает возможности параллельного использования и магнитных дисков, например, для архивации данных.

Драйверы измерительной аппаратуры разрабатываются прежде всего для проблемно-ориентированных ИВК с относительно стабильной аппаратной конфигурацией. При этом достоинства драйверов в полной мере проявляются при определенных условиях использования измерительной аппаратуры:

передача данных осуществляется в режиме прерываний;

каждый запрос на передачу данных предполагает пересылку не одного данного, а массива;

в прикладных программах используется системный аппарат ввода-вывода: синхронная и асинхронная передача, подпрограммы завершения, очереди запросов;

прикладные задачи программируются на языках высокого уровня. Последнее является немаловажным обстоятельством при оценке целесообразности разработки и использования драйвера, так как включение в систему драйвера, структура и функционирование которого соответствуют определенным системным соглашениям, позволяет программировать работу обслуживаемого им ВУ на языках высокого уровня с использованием стандартных операторов ввода-вывода.

Программа системного драйвера состоит из шести участков, называемых *секциями* (рис. 7.6): *преамбулы, заголовка, секции инициализации ввода-вывода, секции обработки прерываний, секции завершения ввода-вывода* и *заключения*.

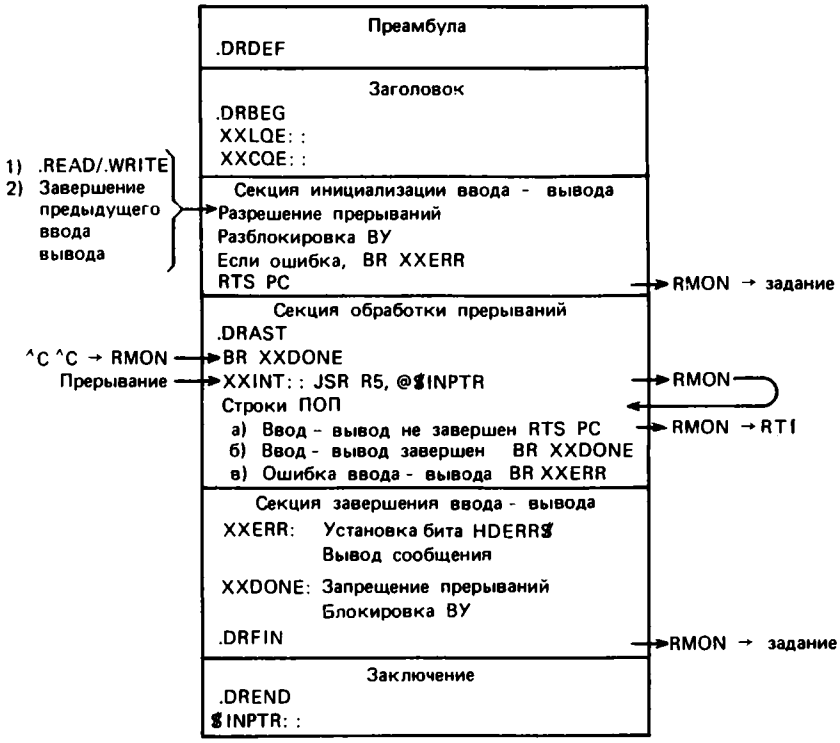


Рис. 7.6. Структура драйвера и его взаимодействие с вычислительной средой

Преамбула драйвера представляет собой список символических обозначений определенных адресов, смещений и констант, используемых далее в тексте драйвера. Указанный список генерируется системной макрокомандой .DRDEF, предназначенной для применения исключительно в текстах драйверов.

Для обращения к слову состояния канала в преамбуле определяются обозначения HDERR$ = 1 (бит 0 аппаратной ошибки) и EOF$ = 20000 (бит 13 конца файла).

Значительное место в программе драйвера занимает работа с элементом очереди ввода-вывода. Все поля элемента очереди получают в преамбуле символические обозначения в двух вариантах: в виде смещений относительно начала элемента и в виде смещений (в том числе отрицательных) относительно третьего слова элемента, в котором содержится номер начального блока файла. Последнее удобно потому, что при передаче драйверу ссылки на очередной элемент очереди монитор помещает

Таблица 7.1. **Числовые (I) и символические (II) обозначения полей элемента очереди ввода-вывода**

Смещение				Содержимое
Относительно начала элемента		Относительно третьего слова		
I	II	I	II	
0	Q.LINK	–4	Q$LINK	Адрес следующего элемента
2	Q.CSW	–2	Q$CSW	Адрес канала
4	Q.BLKN	0	Q$BLKN	Номер блока
6	Q.FUNC	2	Q$FUNC	Код специальной функции
7	Q.JNUM	3	Q$JNUM	Номер задания
7	Q.UNIT	3	Q$UNIT	Номер устройства
10	Q.BUFF	4	Q$BUFF	Адрес буфера
12	Q.WCNT	6	Q$WCNT	Счетчик слов
14	Q.COMP	10	Q$COMP	Адрес ППЗ

в заголовок драйвера адрес именно третьего слова этого элемента. В табл. 7.1 приведены условные обозначения полей элемента очереди вместе с их числовыми эквивалентами.

В загрузочном модуле драйвера присутствует *слово статуса устройства*, в котором содержатся код идентификации устройства (числа от 0 до 377_8 размещаются в младшем байте), а также характеристики устройства, определяемые записью 1 или 0 в соответствующие разряды старшего байта. В преамбуле даются символические обозначения этих разрядов (приводимые ниже пояснения относятся к состоянию 1 соответствующих разрядов):

VARSZ$ = 400 – драйвер поддерживает тома переменного размера;

ABTIO$ = 1000 – при завершении задания происходит обращение к секции завершения ввода-вывода драйвера;

SPFUN$ = 2000 – драйвер разрешает использование в программе специального программного запроса .SPFUN, позволяющего передавать в драйвер дополнительную информацию (код специальной функции), что расширяет возможности драйвера;

HNDLR$ = 4000 – при завершении задания происходит обращение к секции завершения ввода-вывода, если в задании отсутствуют активные элементы очереди;

SPECL$ = 10000 – устройство имеет специальную файловую структуру (например, магнитные ленты кассетного типа);

WONLY$ = 20000 – устройство предназначено только для вывода;

RONLY$ = 40000 – устройство предназначено только для ввода;

FILST$ = 100000 – устройство прямого доступа со стандартной файловой структурой.

Спецификация файла с загрузочным модулем драйвера состоит из двухбуквенного имени и типа SYS: RK.SYS (драйвер НМД), DX.SYS (драйвер НГМД), LP.SYS (драйвер АЦПУ) и т. д. Двухбуквенное обозначение имени драйвера фигурирует также в составе ряда символических обозначений, используемых в тексте драйвера. Например, в драйвере НМД базовый регистр управления обозначается RK$CSR, вектор прерываний – RK$VEC, слово статуса устройства – RKSTS и т. д. Все эти обозначения генерируются макрокомандой .DRDEF, среди аргументов которой указывается имя драйвера.

Разработчик драйвера имеет возможность внести в преамбулу собственные символические обозначения. Если, например, ВУ имеет несколько управляющих регистров (как это имеет место в контроллере КАМАК), то помимо обозначения CA$CSR = 164000, создаваемого автоматически, можно ввести обозначения CA$DMR = CA$CSR + 2 и CA$HBR = = CA$CSR + 4 (предполагается, что драйверу дано имя CA).

В преамбуле определяются также (на основании аргументов макрокоманды .DRDEF) числовые значения размера устройства XXDSIZ, кода идентификации XX$COD и слова состояния XXSTS. Здесь, как и в дальнейшем, XX – произвольное имя рассматриваемого драйвера.

Формат макрокоманды .DRDEF:

.DRDEF name, code, stat, size, csr, vec.

где name – имя драйвера, состоящее из двух символов; code – код идентификации устройства (число от 0 до 377_8: стандартным ВУ назначены младшие номера, нестандартным ВУ рекомендуется давать номера 377, 376_8 и т. д.); stat – слово статуса устройства (образуется комбинацией описанных выше характеристик устройства); size – размер устройства в блоках по 256 слов; csr – адрес РКС устройства; vec – адрес вектора прерываний устройства.

Заголовок драйвера создается с помощью макрокоманды .DRBEG с аргументами, которая генерирует две таблицы. Первая таблица заполняет ячейки $52...60_8$ блока 0 загрузочного модуля драйвера. Сюда записываются размер драйвера, слово статуса устройства и другая служебная информация. Вторая таблица, начинающаяся с метки XXSTRT, определяет содержимое первых пяти слов самого драйвера: адрес вектора прерывания, смещение к метке XXINT, с которой начинается программа обработки прерываний, приоритет 7 процессора (число 340_8), а также ячейки с метками XXLQE и XXCQE. В ячейку XXLQE монитор помещает адрес последнего элемента, стоящего в очереди к драйверу; в ячейку XXCQE заносится адрес текущего, обрабатываемого драйвером элемента. Работа монитора с драйвером и заключается, в частности, в динамическом изменении содержимого этих ячеек по мере постановки в очередь к драйверу новых элементов или обработки старых.

Информация, генерируемая макрокомандой .DRBEG, используется системой в процессе загрузки драйвера в ОП (командой оператора LOAD или программным запросом .FETCH).

Формат макрокоманды .DRBEG:

.DRBEG name,

где name – использованное уже в макрокоманде .DRDEF имя драйвера.

Многие ВУ используют не один вектор прерывания, а два или больше (для крейта КАМАК резервируется восемь векторов). С помощью макрокоманды .DRVTB можно в первом слове драйвера (с меткой XXSTRT) записать не адрес единственного вектора, а смещение к таблице векторов, а также заполнить эту таблицу.

Секция инициализации ввода-вывода, начинающаяся, таким образом, с шестой ячейки (адрес 12) относительно начала драйвера, содержит программные строки инициализации прерываний. Сюда обычно включаются строки разрешения прерываний в РКС ВУ, разблокировки ВУ, а также проверки работоспособности устройства и (в случае обнаружения неисправности) перехода в секцию завершения ввода-вывода. Монитор передает управление на начало секции инициализации после образования элемента очереди ввода-вывода и занесения его адреса в ячейку XXCQE в процессе выполнения программного запроса .READ или .WRITE. Таким образом, каждый программный запрос на ввод-вывод физически выполняет инициализацию прерываний. Сказанное, однако, справедливо только для одиночных запросов, не создающих очереди. Если же драйвер занят обработкой текущего запроса на ввод-вывод, последующие запросы .READ или .WRITE приведут только к заполнению элементов и организации из них очереди к драйверу. Вызывать секцию инициализации ввода-вывода для выполнения следующего запроса, стоящего в очереди, монитор будет каждый раз после полной отработки предыдущего запроса (передачи запланированного в нем количества слов).

Секция инициализации ввода-вывода вызывается монитором как подпрограмма. Поэтому завершающей строкой секции должна быть команда RTS PC, по которой происходят возврат в монитор и передача им управления заданию в ожидании прерывания от ВУ.

Секция обработки прерываний выполняет, в принципе, те же функции, что и обычная ПОП. Как правило, в ней осуществляется передача одной порции данных, модификация адреса буфера пользователя в элементе очереди, модификация источника слов (там же), проверка на завершение ввода-вывода, т. е. на передачу запланированного количества слов. Здесь же возможна проверка правильности передачи (по биту паритета или другими средствами, предоставляемыми устройством) и в случае ошибки повторение передачи, если, конечно, это допускается конструкцией ВУ. Секция начинается макрокомандой .DRAST, выполняющей те же функции, что и программный запрос .INTEN, используемый в ПОП. Формат макрокоманды .DRAST:

.DRAST name, pri, abo,

где name – имя драйвера; pri – приоритет процессора при выполнении процедуры обработки прерываний в драйвере (совпадает с уровнем запроса прерывания от данного ВУ); abo – необязательный параметр, определяющий метку в секции завершения ввода-вывода, с которой начинается процедура прекращения операции ввода-вывода.

Пример макрорасширения макрокоманды .DRAST:

```
        .DRAST XX, 4, XXDONE     ; ВУ подключено к уровню 4
        BR  XXDONE               ; три
XXINT:: JSR  R5, @$INPTR         ; строки
        .WORD ¬C( 4*40 )         ; макрорасширения
```

Значение метки XXINT заносится монитором в первое слово вектора прерываний. Во втором слове вектора находится число 340. Таким образом, каждое прерывание от ВУ, обслуживаемого драйвером, приводит (на аппаратном уровне) к передаче управления на метку XXINT с приоритетом процессора, равным 7. Командой JSR R5, @$INPTR осуществляется переход на подпрограмму, адрес которой находится в ячейке $INPTR. Эта ячейка, расположенная в конце текста драйвера, содержит адрес $ INTEN программы монитора, реализующей перевод монитора в системное состояние, сохранение содержимого регистра R4 из основной программы, снижение приоритета процессора до значения, указанного во втором аргументе макрокоманды .DRAST (число 4 * 40 = = 200 в следующем за XXINT слове), и возврат в секцию обработки прерываний драйвера. После выполнения процедуры обработки прерывания командой RTS PC управление возвращается монитору, который снимает системное состояние, восстанавливает содержимое регистров R4 и R5 (содержимое последнего было сохранено в стеке естественным образом при выполнении команды JSR R5, @$INPTR) и выполняет команду RTI, осуществляющую аппаратный возврат в прерванное задание.

ПОП драйвера имеет кроме точки входа в процедуру обработки прерываний (метка XXINT) еще одну точку входа, расположенную по адресу XXINT–2. Монитор осуществляет передачу управления в эту точку при вводе оператором с клавиатуры команды ⟨CTRL/C⟩ ⟨CTRL/C⟩, а также при фатальных ошибках. Как видно из приведенного выше текста макрорасширения, в этом случае происходит переход на метку XXDONE, находящуюся в секции завершения ввода-вывода, где перед снятием с выполнения всего задания выполняются действия, прекращающие операцию ввода-вывода: запрет прерываний и вызов программы монитора, освобождающей элемент очереди.

В секции обработки прерываний обычно предусматривается несколько выходов. После передачи очередной порции данных, как уже отмечалось выше, происходит нормальный выход из ПОП командой RTS PC.

В случае передачи последней порции данных выполняется команда BR XXDONE, передающая управление в секцию завершения ввода-вывода (см. выше). Наконец, при обнаружении ошибки ввода-вывода управление может быть передано на строки отработки этой ситуации, также находящиеся в секции завершения ввода-вывода.

Секция завершения ввода-вывода является, в сущности, продолжением секции обработки прерываний. Управление передается сюда из ПОП в следующих случаях:

операция ввода-вывода нормально завершилась (передан запланированный объем информации);

оператор ввел команду ⟨ CTRL/C ⟩⟨ CTRL/C ⟩;

в процессе передачи данных зафиксирована аппаратная ошибка.

В первых двух случаях в секции завершения выполняется участок, начинающийся с метки XXDONE (обозначение метки может быть любым): запрещаются прерывания и, если нужно, блокируется ВУ. Заканчивается этот участок макрокомандой .DRFIN, в макрорасширение которой входят строки перехода на программу монитора, управляющую очередями. В последнем случае перед блокированием устройства и выходом в монитор (с помощью той же макрокоманды .DRFIN) устанавливается бит HDERR$ в слове состояния канала и осуществляются другие действия по обработке аппаратной ошибки, например вывод на экран терминала предупреждающего сообщения.

Монитор, получив управление с помощью макрокоманды .DRFIN, освобождает текущий элемент очереди с передачей его в список свободных элементов, а адрес следующего, если он есть, передает драйверу. В этом случае перед тем, как вернуть управление в задание, монитор активизирует секцию инициализации ввода-вывода в драйвере.

Формат макрокоманды .DRFIN:

```
.DRFIN name,
```

где name – имя драйвера.

Последняя секция драйвера – заключение – образуется с помощью макрокоманды .DREND и состоит из нескольких указателей на системные области. В частности, здесь размещается ячейка $INPTR, упоминавшаяся выше.

Формат макрокоманды .DREND:

```
.DREND name,
```

где name – имя драйвера.

На рис. 7.6 обозначены основные этапы выполнения программы драйвера, а также его взаимодействия с монитором и прикладной программой.

7.3. РАЗРАБОТКА ДРАЙВЕРА ДЛЯ ИЗМЕРИТЕЛЬНОЙ АППАРАТУРЫ

Рассмотрим технику разработки и возможности использования драйвера измерительной аппаратуры на примере драйвера для управления АЦП в стандарте КАМАК. Будем считать, что базовый адрес контроллера крейта равен 164000_8, адрес вектора прерываний – 310_8, АЦП установлен на станции 11 и этой станции в РЗМ соответствует разряд 3. Поставим задачу приема кодов от АЦП и записи их в ОП в порядке поступления.

Для управления АЦП используем следующие функции КАМАК:

F(2)A(0) – чтение кода, сброс выходного регистра, сброс *LAM*-требования, разблокировка входа;

F(24)A(0) – блокировка *L*-запроса;

F(24)A(1) – блокировка входа;

F(26)A(0) – разблокировка *L*-запроса;

F(26)A(1) – разблокировка входа.

В преамбулу войдут следующие строки:

```
.MCALL .DRDEF
.DRDEF CA, 377, 0, 0, 164000, 310
CA$A0 = CA$CSR + ⟨11.*40⟩
CA$A1 = CA$A0 + 2
```

В качестве аргументов макрокоманды .DRDEF указаны имя драйвера CA, код идентификации устройства 377_8, адрес РУС контроллера крейта 164000_8 и адрес вектора прерывания 310_8. Размер устройства применительно к измерительной аппаратуре отсутствует, и никакие из перечисленных выше характеристик устройства к данному случаю не применимы. Макрокоманда генерирует символическое обозначение CA$CSR = 164000, которое используется в добавленных в преамбулу строках описания символических обозначений адресов регистров (субадресов) конкретного модуля CA$A0 и CA$A1.

Заголовок драйвера создается с помощью единственной программной строки:

```
.DRBEG CA
```

Для дальнейшего существенно, что эта макрокоманда, создав в начале загрузочного модуля драйвера таблицу из пяти слов, обозначила последние два слова глобальными метками CALQE и CACQE.

Секция инициализации ввода-вывода пишется под конкретную аппаратуру:

```
MOV  CACQE, R4      ; R4 указывает на номер блока
MOV  CA$CSR, R5     ; Адрес РУС в R5
MOV  #26., (R5)     ; Функция F(26) в РУС
TST  @#CA$A0        ; Разблокировка L-запроса
```

```
BIT   #40000,(R5)      ; Проверка X
BEQ   CAERR            ; Аппаратная ошибка, на выход
TST   @#CA$A1          ; Разблокировка входа АЦП
BIT   #40000,(R5)      ; Проверка X
BEQ   CAERR            ; Аппаратная ошибка, на выход
MOVB#10,2(R5)          ; Маска L-запросов в РЗМ
MOV   #102,(R5)        ; Разрешение прерываний в РУС
                       ; и функция F(2) "на будущее"
RTS   PC               ; Выход в монитор
```

Весь текст драйвера должен быть написан в позиционно-независимом коде, так как место загрузки драйвера в ОП заранее не определено. Кроме того, драйвер должен занимать минимальный объем. Поэтому в тексте драйвера широко применяется адресация через регистры общего назначения, а при обращении к странице ввода-вывода следует использовать абсолютную адресацию.

Следует заметить, что при входе в секцию инициализации ввода-вывода монитор переходит в системное состояние, что обеспечивает надежное выполнение инициализации. В секции можно использовать все регистры. В пятое слово драйвера, имеющее метку CACQE, монитор помещает адрес третьего слова элемента очереди, т. е. слова, в котором обычно хранится номер начального блока файла. В приведенном варианте секции инициализации не производится обращений к содержимому элемента ввода-вывода, однако в первой строке секции адрес элемента переносится в регистр R4. Это дает возможность при обнаружении аппаратной ошибки и переходе на метку CAERR обратиться (уже в секции завершения ввода-вывода) через элемент очереди к каналу и установить в слове состояния канала бит HDERR$.

В секции инициализации ввода-вывода разблокируются прерывания (в модуле и контроллере) и вход АЦП. С целью проверки работоспособности аппаратуры после обеих команд обращения к модулю *F(26)A(0)* и *F(26)A(1)* проверяется наличие сигнала *X*. При его отсутствии происходит переход на метку CAERR, находящуюся в секции завершения ввода-вывода. Если ошибок не было, команда RTS PC передает управление монитору, а тот – заданию. Поскольку прерывания теперь разрешены, каждый *L*-запрос, поступивший из АЦП, будет приводить к прерыванию программы с передачей управления на метку CAINT, генерированную макрокомандой .DRAST в секции обработки прерываний.

Текст ПОП пишется разработчиком драйвера исходя из особенностей программирования конкретной аппаратуры и принятого алгоритма обработки каждого события:

```
          .DRAST CA, 4, CADONE     ; Макрокоманда .DRAST
;         BR     CADONE            ; Строки
;CAINT::  JSR    R5,@$INPTR        ; макрорасширения
;         .WORD¬C⟨ 4*40 ⟩          ; макрокоманды .DRAST
```

```
MOV   CA$CSR, R5                ; Адрес РУС в R5
MOV   CACQE, R4                 ; R4 указывает на номер блока
MOV   @#CA$A0, @Q$BUFF (R4)     ; Код из АЦП в буфер
BIT   #40000, (R5)              ; Нет ли аппаратной ошибки?
BEQ   CAERR                     ; Есть
ADD   #2, Q$BUFF (R4)           ; Смещение указателя в буфере
DEC   Q$WCNT (R4)               ; Модификация счетчика кодов
BEQ   CADONE                    ; Принято заданное количество
MOV   #102, (R5)                ; Разрешение прерываний + F(2)
RTS   PC                        ; Выход в монитор
```

Три строки после макровызова .DRAST, помещенные в приведенном выше тексте как комментарий, вставляются транслятором в процессе макрорасширения. Вход в ПОП осуществляется в точке CAINT. После перехода в системное состояние и снижения приоритета процессора до значения, указанного в аргументе макрокоманды .DRAST, монитор возвращает управление в ПОП на строки, написанные разработчиком драйвера. В секции обработки прерываний, как и в любой ПОП, можно свободно пользоваться регистрами R4 и R5, содержимое которых сохраняет система.

Поскольку в ячейке CACQE находится адрес третьего слова элемента очереди, то после занесения этого адреса в регистр R4 в программе можно использовать символические смещения относительно слова Q$BLKN, приведенные в табл. 7.1. Так, команда MOV@# CA$A0, @Q$BUFF (R4) переносит код из буферного регистра АЦП в буфер пользователя, адрес которого находится в слове элемента очереди со смещением Q$BUFF. Далее проверяется правильность выполнения команды КАМАК и при возникновении аппаратной ошибки (сигнал $X = 0$) осуществляется переход на метку CAERR в секции завершения. В противном случае модифицируется содержимое элемента очереди: увеличивается на 2 адрес буфера и уменьшается на 1 содержимое счетчика слов. Если в счетчике оказывается 0, т. е. из АЦП принято запланированное количество кодов, происходит переход на метку CADONE для нормального завершения работы драйвера. Если же ввод еще не окончен, в РУС контроллера КАМАК восстанавливается бит 6, и по команде RTS PC происходит переход в монитор, который выполняет все необходимые операции по выходу из режима прерываний и предаче управления заданию.

Продолжением ПОП является секция завершения ввода-вывода, состоящая из двух участков: нормального завершения (метка CADONE) и завершения по ошибке (метка CAERR).

```
CAERR:  BIS   #HDERR$, @-2 (R4)  ; Аппаратная ошибка
        CLR   Q$COMP(R4)         ; Запрет вызова ППЗ
        MOV   PC, R4             ; Позиционно-независимый адрес
        ADD   #ERRMES-., R4      ; аварийного сообщения
1$:     TSTB  @#177564           ; Ожидание готовности
        BPL   1$                 ; экрана терминала
```

```
        TSTB  (R4)                  ; Конец сообщения?
        BEQ   CAFIN                 ; Да
        MOVB  (R4)+,  @#177566      ; Нет, вывод символа
        BR    1$                    ; Повторить
ERRMES: .ASCIZ <15> <12> "***ADC ERROR***" <15> <12>
        .EVEN
CADONE: MOV   #24., (R5)+           ; F (24) в РУС
        TST   @#CA$A0               ; Блокировка L-запроса
        TST   @#CA$A1               ; Блокировка входа АЦП
        BIC   #10, (R5)             ; Маска L-запросов в РЗМ
CAFIN:  .DRFIN CA                   ; Выход в монитор
```

Переход на метку CAERR может произойти как на этапе инициализации, так и при выполнении операции ввода данных, если обнаружена неисправность модуля (сигнал $X = 0$). В этом случае в слове состояния канала устанавливается бит HDERR$, засылкой нуля в последнее слово элемента очереди (имеющее смещение Q$COMP) запрещается вызов ППЗ и на экран терминала выводится аварийное сообщение. После этого выполняется участок нормального завершения, на котором блокируются *L*-запрос и вход АЦП, а также запрещается прохождение *L*-запроса от модуля в РУС. Далее с помощью макровызова .DRFIN осуществляется переход на программу монитора (адрес которой хранится в ячейке с фиксированным смещением 270_8 относительно начала монитора), освобождающую элемент очереди.

Последняя секция драйвера, заключение, генерируется с помощью макровызова .DREND:

```
.DREND CA
.END
```

После получения исходного текста драйвера CA.MAC его необходимо оттранслировать, скомпоновать и включить в систему:

MAC CA	– трансляция драйвера;
LINK/EXEC:CA.SYS	– компоновка;
COPY/SYSTEM CA.SYS MX0:CA.SYS	– копирование на системный диск;
INSTALL CA	– установка в системе;
LOAD CA	– загрузка в ОП.

Для нормального функционирования загрузочный модуль драйвера должен иметь тип SYS и находиться на системном диске. Операция установки драйвера заключается во включении информации о драйвере в соответствующие системные таблицы. Так, в таблицу $PNAME помещается имя драйвера в коде RADIX-50, в таблицу $STAT – статусное слово, в таблицу $DVREC – адрес на системном диске, в таблицу $HSIZE – размер драйвера в байтах. Последняя информация используется при загрузке драйвера в ОП командой LOAD. После загрузки драйвера и опре-

деления тем самым его местонахождения в ОП, в таблицу $ENTRY заносится адрес точки входа в драйвер, т. е. адрес четвертого слова от начала драйвера. Узнать этот адрес (что требуется, в частности, при отладке драйвера) можно с помощью команды оператора SHOW DEVICES.

После того как драйвер установлен в системе и загружен в ОП, к нему можно обращаться из прикладной программы. Делается это точно так же, как и при работе с любым стандартом ВУ:

```
CAMAC: .RAD50 /CA/              ; Описание имени
       0                        ; устройства
DATA:  .BLKW 8192.              ; Место под данные (8К слов)
ARG:   .BLKW 5                  ; Область для аргументов
FILE:  .RAD50 /DK/              ; Спецификация
       .RAD50 /DATA01CAM/       ; файла данных
START: .LOOKUP #ARG, #0, #CAMAC
       .ENTER #ARG, #1, #FILE, #32.
         .
         .
         .
       .READC #ARG, #0, #DATA, #8192., #COMP, #0
         .
         .
         .
COMP:  .WRITE #ARG, #1, #DATA, #8192., #0
       RTS PC
```

По метке CAMAC дано описание имени устройства (имени драйвера), а по метке DATA зарезервировано место под накапливаемые данные. В приведенном примере накопленные данные переносятся на НМД, поэтому на диске заводится файл DK:DATA01.CAM соответствующего размера. Программным запросом .LOOKUP открывается канал 0 связи с АЦП, а программным запросом .ENTER на НМД резервируется место для файла и открывается канал 1 связи с ним.

Программный запрос .READ формирует элемент очереди и передает его драйверу CA. После этого программа продолжается, а драйвер организует прием кодов из АЦП в режиме прерываний по *L*-запросам от модуля. После накопления заданного в макровызове .READC числа кодов драйвер завершает работу, а монитор, обнаружив в последнем слове элемента очереди адрес ППЗ COMP, активизирует эту подпрограмму. В приведенном примере в ППЗ осуществляется сброс данных на диск, причем использование асинхронного запроса .WRITE позволяет параллельно с записью на диск продолжить выполнение программы.

Таким образом, использование драйвера чрезвычайно упрощает программирование измерительной аппаратуры, однако в такой же степени уменьшает возможности модификации программы. Действительно, разработанный драйвер рассчитан на использование определенного модуля с заданной заранее системой команд; модуль должен быть устано-

влен на конкретной станции; алгоритм обработки включен в текст драйвера и не может изменяться. Универсальность драйвера можно повысить, заложив в него возможность настройки и используя для обращения к нему специальный программный запрос .SPFUN. Этот запрос позволяет передавать в драйвер код специальной функции – одно из 128 чисел в диапазоне $200_8 \ldots 377_8$. Формат макровызова .SPFUN следующий:

.SPFUN arg, chan, func, buf, wcnt, blk, crtn,

где arg – область из шести слов для аргументов макровызова; chan – номер канала связи с конкретным ВУ; func – числовой код специальной функции; buf – адрес буфера в прикладной программе; wcnt – обычно число передаваемых слов или других единиц информации (для драйвера, разрабатываемого пользователем, допускается придавать этому параметру любой смысл, соответствующий условиям функционирования конкретного ВУ); blk – обычно начальный номер блока (может иметь любой другой смысл, соответствующий специфике конкретного ВУ); crtn – адрес ППЗ (этот параметр может быть опущен).

Рассмотрим пример драйвера, настраивающегося в зависимости от кода функции, на требуемый номер станции (1...23) и заданный алгоритм обработки. Для модуля АЦП естественно предусмотреть два алгоритма-накопления в порядке поступления и распределения по каналам. Будем считать, что в коде специальной функции двоичные разряды 2...6 определяют номер станции, а разряд 0 задает алгоритм обработки. Тогда код функции можно образовать следующим образом:

200 + ⟨ N * 4 ⟩ – АЦП установлен на станции N, накопление в порядке поступления;

200 + ⟨ N * 4 ⟩ + 1 – АЦП установлен на станции N, поступающие коды распределяются по каналам.

В преамбуле драйвера следует определить возможность работы с макровызовом .SPFUN:

```
.MCALL .DRDEF
.DRDEF CA, 377, SPFUN$, 0, 164 000, 310
```

Определения CA$A0 и CA$A1 теперь отсутствуют, так как номер станции АЦП заранее неизвестен. В то же время явное указание адреса вектора прерываний 310 предполагает, что *L*-запросы от всех используемых станций скоммутированы на этот вектор; в противном случае в драйвер пришлось бы передавать не только номер станции, но и адрес вектора.

Заголовок драйвера остается без изменения: .DRBEG CA

При выполнении программного запроса на ввод-вывод (в данном случае запроса .SPFUN) управление передается в секцию инициализации. При этом код специальной функции, указываемый в числе аргументов макровызова .SPFUN, заносится в младший байт четвертого слова элемента очереди (см. рис. 7.3). В секции инициализации следует проанали-

зировать содержимое этого байта и извлечь из него номер станции, на которой установлен АЦП. Кроме того, полезно проверить отрицательность кода специальной функции, чтобы не обрабатывать ошибочно выданные запросы .READ или .WRITE. В секции инициализаци, в отличие от секции обработки прерываний, можно пользоваться всеми регистрами общего назначения:

```
MOV  CACQE, R4          ; Адрес третьего слова элемента очереди
MOV  CA$CSR, R5         ; Адрес РУС
MOV  Q$FUNC(R4), R0     ; Получили код функции
BIC  #177603, R0        ; Оставили только N
ASL  R0                 ; Теперь
ASL  R0                 ; в R0
ASL  R0                 ; N x 40
ADD  R5, R0             ; CSR + ⟨ N x 40 ⟩ , т. е. NA(0)
TSTB Q$FUNC(R4)         ; Есть специальная функция?
BPL  CAERR1             ; Нет, был запрос .READ или .WRITE
MOV  #26., (R5)         ; F(26) в РУС
TST  (R0)+              ; NA(0) F(26)
BIT  #40000, (R5)       ; Проверка
BEQ  CAERR2             ; Аппаратная ошибка
TST  (R0)               ; NA (1) F (26)
BIT  #40000             ; Проверка X
BEQ  CAERR2             ; Аппаратная ошибка
MOV  #100, (R5)         ; Разрешение прерываний в РУС
MOVB#377, 2 (R5)        ; Размаскирование всех L-запросов
RTS  PC                 ; Выход в монитор
```

В секции обработки прерываний должны быть предусмотрены две ПОП по числу алгоритмов обработки, каждой из которых передается заданное число накапливаемых событий. Однако в случае распределения приходящих кодов по каналам число накапливаемых событий обычно задается весьма большим. Действительно, если в 1024-канальном спектре желательно накопить в каждом канале в среднем по 10^3 отсчетов, то общее число регистрируемых событий составит около 10^6. Запись такого числа в ОП требует два машинных слова.

Одну половину этого двухсловного числа естественно передать в драйвер через счетчик слов; для передачи второй половины можно воспользоваться номером начального блока. Такое "нештатное" использование номера блока допускается в тех случаях, когда понятие блока не имеет смысла для конкретного устройства (сказанное справедливо также для адреса буфера и счетчика слов). Будем для определенности считать, что в счетчике слов находится младшая половина числа событий, а в ячейке для номера блока – старшая. Учтем еще, что 1 в разряде 0 кода специальной функции обозначает алгоритм распределения по каналам, а 0 в том же разряде – алгоритм записи кодов в ОП в порядке поступления. Секция обработки прерываний может иметь следующий вид:

```
        .DRAST  CA, 4, CAFIN        ; Перевод в системное состояние
        MOV     CA$CSR, R5          ; Адрес РУС
        MOV     R0, -(SP)           ; Понадобятся
        MOV     R1, -(SP)           ; регистры R0 и R1
        MOV     (R5), -(SP)         ; Сохраним также РУС
        MOV     CACQE, R4           ; Адрес элемента очереди
        MOV     Q$FUNC(R4), R0      ; Код специальной функции
        BIC     #177603, R0         ; Оставили только N
        ASL     R0                  ; Теперь
        ASL     R0                  ; в R0
        ASL     R0                  ; N x 40
        ADD     R5, R0              ; NA (0) сохраним в R0
        MOV     #2, (R5)            ; Функция F(2) в РУС
        MOV     (R0), R1            ; Код из АЦП в R1
        BIT     #40000, (R5)        ; Нет ли аппаратной ошибки?
        BEQ     CAERR3              ; Есть
        BIT     #1, Q$FUNC(R4)      ; Код алгоритма 1?
        BEQ     CAINT1              ; Нет, запись код за кодом
; Строки распределения по каналам
        ASL     R1                  ; Код x 2
        ADD     Q$BUFF(R4), R1      ; Адрес канала в R1
        INC     (R1)                ; Инкремент в канале
        TST     Q$WCNT(R4)          ; Младшая половина счетчика = 0?
        BNE     1$                  ; Нет, на ее декремент
        DEC     Q$WCNT(R4)          ; Да, все равно декремент
        BNE     1$                  ; В младшей половине 0?
        DEC     Q$WCNT(R4)          ; Да, тогда декременты младшей
        DEC     Q$BLKN(R4)          ; и старшей половин и
        BR      INTOUT              ; на выход из прерывания
1$:     DEC     Q$WCNT(R4)          ; Нет, декремент только младшей
        BNE     INTOUT              ; Еще не 0, на выход из ПОП
        TST     Q$BLKN(R4)          ; Младшая половина = 0, а старшая?
        BEQ     CADONE              ; Старшая = 0, накопление окончено.
        BR      INTOUT              ; На выход из прерывания
; Строки накопления в порядке поступления
CAINT1: MOV     R1, @Q$BUFF(R4)     ; Код в буфер
        ADD     #2, Q$BUFF(R4)      ; Смещение указателя
        DEC     Q$WCNT(R4)          ; Декремент счетчика
        BEQ     CADONE              ; Накопление окончено
INTOUT: JSR     PC, RESTOR          ; Восстановить регистры
        BIS     #100, (R5)          ; Разрешение прерываний в РУС
        RTS     PC                  ; Выход из прерывания
RESTOR: MOV     (SP)+, R0           ; Подпрограмма
        MOV     (SP)+, (R5)         ; восстановления
        MOV     (SP)+, R1           ; регистров
        RTS     R0                  ; Выход из подпрограммы
```

В ПОП потребовалось использование двух регистров общего назначения, поэтому в начале ПОП сохраняется в стеке содержимое R0 и R1, а в конце – восстанавливается. Кроме того, в стек проталкивается со-

держимое РУС. Это дает возможность пользоваться драйвером в программах, обслуживающих кроме АЦП и другие модули КАМАК (правда, не в режиме прерывания). Получив в R0 код специальной функции, программа выделяет в нем номер станции (биты 2...6) и сдвигом влево на три разряда образует число $N \times 40$, которое фигурирует в определении физического адреса модуля. Прибавлением к нему базового адреса РУС формируется адрес *N A(0)*. После получения кода из АЦП осуществляется анализ бита 0 кода специальной функции и разветвление программы в зависимости от заданного алгоритма. Если происходило накопление спектра,, то после регистрации очередного события требуется уменьшить на 1 содержимое двухсловного счетчика событий. В случае регистрации кодов в порядке поступления выполняется декремент однословного счетчика и смещение указателя в буфере. В любом случае ПОП заканчивается восстановлением содержимого использованных в программе регистров, включая РУС, и выходом в монитор.

Секция завершения ввода-вывода также несколько усложняется, так как в ней появляются три входа отработки ошибок: CAERR1, CAERR2 и CAERR3. Вход CAERR1 соответствует реакции на программную ошибку (выдача запроса .READ или .WRITE вместо .SPFUN). В этом случае следует запретить вызов ППЗ и вывести на экран сообщение "***WRONG REQUEST***". Поскольку в программе драйвера предусматривается вывод различных сообщений, для сокращения размеров драйвера это делается с помощью подпрограммы REPORT. При входе в подпрограмму адрес выводимого сообщения должен находиться в регистре R4. Необходимость писать текст драйвера в позиционно-независимом коде усложняет процедуру заполнения R4 (нельзя воспользоваться обычной конструкцией MOV ERRMS1, R4, которая является позиционно-зависимой). Переход на метку CAERR1 осуществляется в секции инициализации еще до сохранения R0, R1 и РУС, поэтому восстанавливать их не требуется.

Вход CAERR2 соответствует аппаратной ошибке, зарегистрированной на этапе инициализации. В этом случае надо установить бит 0 в слове состояния канала, вывести сообщение "***ADC ERROR***" и завершить работу драйвера. Восстанавливать регистры не требуется.

Вход CAERR3 соответствует аппаратной ошибке, зарегистрированной в ПОП. Обработка этой ошибки отличается от обработки предыдущей только необходимостью восстановить содержимое регистров.

Обработка ошибок завершается сбросом разряда разрешения прерывания в РУС (метка CAFIN) и выходом из драйвера в монитор. Для нормального завершения работы драйвера после приема запланированного количества кодов предусмотрена метка CADONE. В этой части программы дополнительно блокируется АЦП и восстанавливаются регистры. В результате секция завершения ввода-вывода выглядит следующим образом:

```
CAERR1: CLR     Q$COMP(R4)              ; Запрет ППЗ
        MOV     PC, R4                  ; Формирование адреса
        ADD     #ERRMS1-., R4           ; сообщения по ошибке
        JSR     PC, REPORT              ; Вывести сообщение 1
        BR      CAFIN                   ; На завершение
CAERR3: JSR     PC, RESTOR              ; Восстановить регистры
CAERR2: BIS     #HDERR$, @Q$CSW(R4)     ; Аппаратная ошибка
        MOV     PC, R4                  ; Формирование адреса
        ADD     #ERRMS2-., R4           ; сообщения об ошибке
        JSR     PC, REPORT              ; Вывести сообщение 2
        BR      CAFIN                   ; На завершение
ERRMS1: .ASCIZ <15><12>"***WRONG REQUEST***"<15><12>
ERRMS2: .ASCIZ <15><12>"***ADC ERROR***"<15><12>
        .EVEN
REPORT: TSTB    @#177564                ; Подпрограмма
        BPL     REPORT                  ; вывода на
        TSTB    (R4)                    ; экран терминала
        BEQ     RTSPC                   ; аварийного сообщения
        MOVB    (R4)+, @#177566         ; При входе в подпрограмму
        BR      REPORT                  ; в R4 должен быть адрес
RTSPC:  RTS     PC                      ; выводимой строки
CADONE: MOV     #24., (R5)              ; F(24) в РУС
        TST     (R0)+                   ; NA(0)F(24)
        TST     (R0)                    ; NA(1)F(24)
        JSR     PC, RESTOR              ; Восстановление регистров
CAFIN:  BIC     #100, @#CA$CSR          ; Запрещение прерываний в РУС
        .DRFIN                          ; Завершение работы драйвера
```

Приведем два примера программирования АЦП с использованием описанного выше драйвера. Пусть, как и раньше, требуется накапливать данные в порядке их поступления:

```
DATA:   .BLKW 1000.  ; Место под данные
ARG:    .BLKW 6      ; Область для аргументов (6 слов!)
ADC:    .RAD50 /CA/  ; Описание
        .WORD 0      ; имени драйвера
START:  .LOOKUP #ARG, #0, #ADC
        .SPFUN #ARG, #0, #200+<11.*40>, #DATA, #1000., #0,#CRTN
          .
          .; Продолжение программы
          .
CRTN:     .
          .; Подпрограмма завершения
          .
        RTS PC
```

В приведенном фрагменте предполагается, что АЦП установлен на станции 11 и требуется накопить 1000 кодов. Накопление осуществляется параллельно с выполнением программы; после регистрации последнего события управление передается в ППЗ (метка CRTN), где, например, данные переписываются на диск.

Рассмотрим теперь случай накопления спектра. Пусть АЦП настроен на выдачу 9-разрядного кода и требуется зарегистрировать 20000 событий. В функции ППЗ входит установка программного флага, с помощью которого основной программе передается информация о завершении измерений (АЦП установлен на станции 2):

```
DATA:   .BLKW 512.   ; Место под 2^9 = 512 каналов
ARG:    .BLKW 6      ; Область для аргументов
FLAG:   .WORD 0      ; Флаг
ADC     .RAD50 /CA/  ; Описание
        .WORD 0      ; имени драйвера
START:  .LOOKUP #ARG, #0, #ADC
        .SPFUN #ARG, #0, #200 + ⟨2 * 40⟩ +1, #DATA, #3392., #3, #CRTN
         .
         .; Продолжение программы
         .
CRTN:   INC FLAG     ; Подпрограмма
        RTS PC       ; завершения
```

Заданное число событий требует для записи двух слов. Поскольку $200000 = 3 \cdot 65536 + 3392$, в старшем слове (предпоследний аргумент макровызова .SPFUN, предназначенный для номера блока) записывается 3, в младшем (пятый аргумент макровызова, предназначенный для счетчика слов) – 3392. Форма записи третьего аргумента показывает, что задается алгоритм накопления спектра.

Хотя описанный драйвер и отличается некоторой универсальностью, позволяя выбирать алгоритм регистрации и номер станции (что, между прочим, привело к заметному усложнению программы), его основной недостаток – жесткая функциональность – сохранился. С другой стороны, использование драйвера существенно упрощает программирование измерительной аппаратуры и позволяет вести его на языке высокого уровня. В тех случаях, когда алгоритм работы аппаратуры достаточно жестко определен, но программа в целом часто изменяется (каждый пользователь составляет свою программу) применение подобного драйвера может оказаться оправданным. В особенности это относится к промышленным приборам и экспериментальной аппаратуре жесткой конфигурации. Что же касается экспериментальных установок, выполняемых на базе аппаратуры КАМАК, одним из основных достоинств которой является простота применения аппаратной конфигурации системы, то применение драйвера описанного типа слишком сужает их возможности.

Существует альтернативный подход к разработке драйвера для аппаратуры КАМАК. Если программный запрос на ввод-вывод допускает передачу значительного числа аргументов, можно разработать драйвер, выполняющий любые операции КАМАК с произвольными модулями. Например, в ОС РВ программный запрос на ввод-вывод QIO$ допускает передачу до 12 аргументов, из которых семь используются произ-

вольно. Через эти аргументы драйверу можно передать такую информацию, как код операции (ввод, вывод, отмена запроса, подсоединение к вектору и отсоединение от него и т. д.); адрес буфера и его размер; все три составляющие операции КАМАК (*N, A* и *F*); адрес вектора или номер *L*-запроса и т. д. Например, программный запрос

QIO$C IO.WLB, 1, 5, STATBL, ASTPR, ⟨ BUF, 4, 0, ⟨ A*100+N*2 ⟩,16

позволяет записать в модуль, установленный на станции *N*, по субадресу *A* 24-разрядное слово из буфера BUF с помощью функции *F(16)*. По окончании передачи установится локальный программный флаг 5 и активизируется ППЗ ASTPR. Диагностическая информация о завершении вывода будет записана в статусный блок STATBL.

С использованием аналогичных по структуре программных запросов можно зарезервировать за программой конкретный *L*-запрос и определить точку входа в соответствующую ему ПОП, реализовать функции КАМАК, не требующие передачи данных (блокировка, маскирование), выполнить заданную группу операций КАМАК (своего рода макрокоманда) и т. д. Драйвер приобретает высокую степень универсальности, однако программирование теряет простоту и наглядность и, в сущности, мало отличается от программирования на физическом уровне.

Глава 8

УПРАВЛЕНИЕ ПРОГРАММАМИ РЕАЛЬНОГО ВРЕМЕНИ

8.1. СРЕДСТВА ВВОДА-ВЫВОДА ЧЕРЕЗ ТЕРМИНАЛ

Важным средством управления программами реального времени является ввод параллельно с выполнением программы управляющих символов и команд через клавиатуру терминала. В системе РАФОС предусмотрены две возможности передачи данных через терминал: с помощью драйвера терминала и программных запросов ввода-вывода .READ и .WRITE (с модификациями .READW, .READC, .WRITW, .WRITC), а также с помощью специальных *программ обслуживания терминала* (входящих в состав RMON) и программных запросов .PRINT, .TTYOUT, .TTOUTR, .TTYIN и .TTINR.

Для вывода на экран терминала некоторого сообщения необходимо сформировать его в выходном буфере, являющемся частью прикладной программы. Буфер должен быть заполнен кодами ASCII воспроизводимых на экране символов (букв, цифр, специальных знаков). Допускается помещение в буфер управляющих кодов, специфичных для конкретного терминала, осуществляющих, например, перемещение курсора по полю экрана, стирание знаков и строк, переключение алфавитов и т. д.

Ввод с клавиатуры при помощи драйвера имеет некоторые особенности. Объем вводимых данных указывается в словах, что исключает возможность ввода нечетного числа байтов, в частности одного. Работа

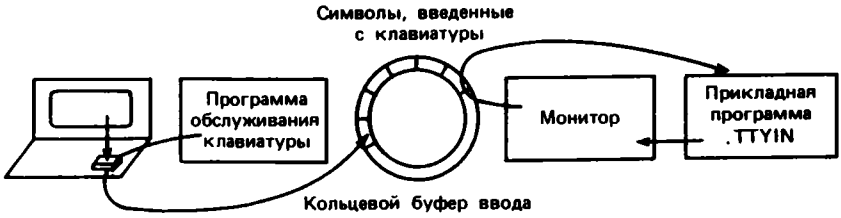


Рис. 8.1. Процесс ввода в программу символов с клавиатуры

драйвера прекращается только после выполнения двух условий: с клавиатуры введено не менее затребованного в макрокоманде ввода числа символов; введенная строка заканчивается кодами ⟨ВК⟩, ⟨ПС⟩ или ⟨CTRL/Z⟩.

Таким образом, запросы .READ, .READW и .READC требуют набора на клавиатуре по меньшей мере двух символов: командного и завершающего строку (например, ⟨ВК⟩). Если же команда должна состоять из нескольких символов, за ними приходится вводить завершающий ВК. Это не всегда удобно. С другой стороны, запрос .READC позволяет организовать параллельные процессы: выполнение программы и одновременно ожидание команды. Ввод с клавиатуры заданного числа командных символов приведет к прерыванию программы и переходу на ППЗ, в которой можно выполнить управляющие действия – включить или выключить аппаратуру, изменить режим ее работы, организовать переход к требуемому программному модулю и т. д.

Рассмотрим вторую возможность ввода-вывода – с помощью включенных в состав монитора программ обслуживания терминала и соответствующих программных запросов. Этот способ взаимодействия оператора с прикладной программой основан на использовании *кольцевых буферов ввода-вывода*, находящихся в смешанной области каждого задания.

Когда оператор нажимает на какую-либо клавишу, программа обслуживания клавиатуры активизируется прерыванием от клавиатуры и передает код нажатой клавиши в кольцевой буфер ввода (рис. 8.1). По мере набора последующих символов на клавиатуре их коды накапливаются в кольцевом буфере, который является, таким образом, промежуточной станцией между оператором и прикладной программой. Для передачи символа из кольцевого буфера в программу последняя должна выдать соответствующий запрос к монитору (.TTYIN или TTINR). Монитор, получив запрос, забирает из кольцевого буфера первый введенный в него символ и передает его в программу. Последующие запросы .TTYIN или .TTINR будут передавать в программу очередные символы из кольцевого буфера ввода. В результате кольцевой буфер ввода заполняется в темпе работы оператора за клавиатурой, а освобождается в темпе работы прикладной программы.

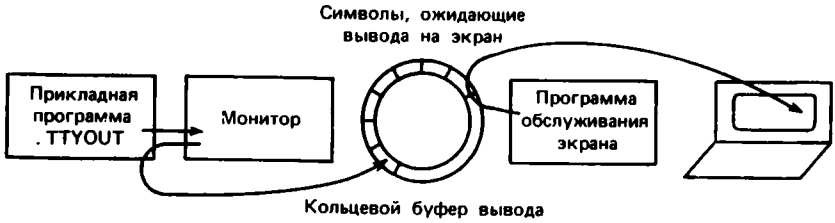


Рис. 8.2. Процесс вывода из программы символов на экран

Кольцевой буфер вывода (рис. 8.2) заполняется монитором по мере выполнения им программных запросов .TTYOUT, .TTOUTR или .PRINT, выдаваемых прикладной программой. Символы накапливаются в кольцевом буфере, ожидая своей очереди для вывода на экран. Программа обслуживания экрана, входящая в состав монитора, активизируется прерываниями от экрана и после выполнения последним предыдущей операции вывода передает в регистр данных экрана код очередного символа из кольцевого буфера. Таким образом, кольцевой буфер вывода заполняется в темпе работы прикладной программы, а освобождается в темпе работы экрана терминала.

Ввод данных с клавиатуры терминала может осуществляться в двух режимах: *обычном* и *специальном*. Вид режима определяется разрядом 12 слова состояния задания, располагаемого в ячейке с абсолютным адресом 44_8, входящей в системную область связи. Значение разряда 12, равное 0, соответствует обычному режиму, 1 – специальному. Компоновщик, создавая файл с загрузочным модулем, заносит 0 в рязряд 12, так что по умолчанию назначается обычный режим. Специальный режим можно установить либо на этапе подготовки загрузочного модуля, включив в программу следующие строки:

```
.ASECT
.=44
.WORD 10000
.PSECT
```

либо на этапе выполнения программы строкой MOV # 10 000, 44.

Обычный режим ввода характеризуется тем, что ввод символов из кольцевого буфера в прикладную программу может начаться только после нажатия клавиши с каким-либо ограничителем строки (⟨ВК⟩, ⟨ПС⟩, ⟨CTRL/Z⟩, ⟨CTRL/C⟩). Пока в буфере нет признака конца строки, монитор как бы считает, что буфер пуст. Программный запрос на ввод символа (.TTYIN или .TTINR) ничего не введет в программу. Если же был набран ограничитель строки, монитор, выполняя программный запрос на ввод, перешлет один (первый) символ в прикладную программу. Выполняя программные запросы .TTYIN (.TTINR) в цикле, можно ввести в программу всю набранную на клавиатуре строку. По-

скольку строка, пока она еще не закончилась ограничителем, хранится в кольцевом буфере, возможно редактирование строки: удаление неправильно набранных символов и замена их новыми, а также стирание строки. Кроме того, в обычном режиме монитор обеспечивает высвечивание на экране набранных символов (*эхо*, или *эхоконтроль*).

Специальный режим ввода характеризуется прежде всего тем, что передача символа из кольцевого буфера в прикладную программу происходит в ответ на каждый запрос на ввод независимо от того, имеется в буфере целая строка или отдельные символы. Такой режим удобен прежде всего тем, что позволяет вводить одиночные символы, не сопровождаемые нажатием ⟨ВК⟩. Естественно, что в специальном режиме нельзя редактировать строку; кроме того, монитор выключает эхоконтроль, и отображение набираемых символов на экране должна осуществлять прикладная программа. Следует подчеркнуть, что специальный режим относится только к вводу с клавиатуры. Вывод на экран всегда осуществляется одинаково.

Рассмотрим особенности программных запросов на ввод-вывод через терминал. Эти запросы выполняются по-разному в зависимости от состояния разряда 6 слова состояния задания (ячейка 44_8). Этот разряд носит название *Запрет ожидания терминала*, но правильнее было бы его назвать *запрет блокировки программы средствами ОС*. Если разряд установлен, блокировка запрещена, если сброшен – разрешена. Рассмотрим оба случая.

1. Разряд 6 слова состояния задания сброшен. Запросы .TTYIN и TTINR выполняются одинаково. Если к моменту выдачи запроса кольцевой буфер не пуст, очередной символ вводится в программу. Если же буфер пуст, программа блокируется средствами ОС до тех пор, пока символ не будет набран на клавиатуре. Поскольку блокирование программы средствами ОС не занимает процессор, в это время может выполняться менее приоритетное (фоновое) задание. Запрос .TTINR пересылает символ из кольцевого буфера в R0; запрос .TTYIN делает то же самое, но сверх того может переслать символ и в ячейку ОП, если ее адрес указан в виде аргумента макрокоманды:

```
.TTYIN  MEM                ; Символ в R0 и в MEM
```

Запросы .TTYOUT и .TTOUTR также выполняются одинаково. Если к моменту выдачи запроса в кольцевом буфере есть свободное место, символ записывается в кольцевой буфер и с течением времени выводится на экран. Если же буфер полон (что может произойти в том случае, если символы поступают из программы в кольцевой буфер быстрее, чем они успевают высвечиваться на экране), программа блокируется средствами ОС до освобождения места в буфере.

2. Разряд 6 слова состояния задания установлен (блокировка программы запрещена). Запросы .TTYIN и .TTINR выполняются одинаково, лишь если к моменту выдачи запроса в буфере имеется символ.

Если же буфер пуст, запрос .TTYIN останавливает выполнение программы до появления символа. В отличие от п. 1 программа переходит в состояние ожидания не с помощью средств ОС, а входом в программный цикл:

```
EMT  340   ; Макрорасширение
BCS  .-2   ; макрокоманды .TTYIN
```

В случае оперативно-фонового режима такой программный цикл, выполняемый в оперативной программе, не даст выполняться фоновой.

Иначе выполняется запрос .TTINR. Если к моменту выдачи запроса буфер пуст, монитор устанавливает разряд *C* РСП и передает управление программе, которая продолжается так и не введя символа. Таким образом, с помощью запроса .TTINR при установленном разряде 6 слова состояния задания символы можно вводить только с упреждением. При этом по состоянию разряда *C* программа может определить, был введен символ или нет и в зависимости от этого перейти на то или иное продолжение.

Запросы на вывод функционируют аналогично запросам на ввод. Если к моменту выдачи запроса кольцевой буфер вывода полон и для поступившего из программы символа нет места, запрос .TTYOUT организует программный цикл ожидания освобождения буфера:

```
EMT  341   ; Макрорасширение
BCS  .-2   ; макрокоманды .TTYOUT
```

Запрос .TTOUTR в тех же условиях передает управление программе (символ теряется), но при этом устанавливает разряд *C* РСП. Программа может проанализировать состояние разряда *C* и в зависимости от результата анализа выполнить те или иные действия.

Рассмотрим пример ввода-вывода через терминал. Пусть программа в некоторой точке должна затребовать от оператора командный символ, чтобы перейти на одно из возможных продолжений. Казалось бы, для этого требуется три строки:

```
MOV #10000, 44  ; Спецрежим
.TTOUTR #'?     ; Вывод подсказки (знака вопроса)
.TTINR          ; Ожидание нажатия клавиши
```

Однако если программа работает уже давно, может случиться, что в кольцевом буфере ввода есть какие-то коды. Они могли попасть туда, например, в результате случайного нажатия на клавиши. В этом случае программный запрос .TTINR введет в качестве командного какой-то случайный код и ход программы окажется непредсказуемым. Поэтому перед вводом командного символа следует очистить кольцевой буфер ввода от "мусора". Программа очистки буфера и ввода командного символа для работы в фоновом режиме выглядит следующим образом:

```
      MOV #10100, 44   ; Спецрежим и запрет блокировки
1$:   .TTINR           ; Получить символ "мусор"
```

```
BCC 1 $          ; Символ есть, вернуться за следующим
.TTYOUT #'?      ; Буфер очищен, вывод подсказки
.TTYIN           ; Ожидание командного символа
.TTYOUT          ; Эхо
.
.; Обработка командного символа в R0
.
```

Первый запрос .TTINR в этом примере делает попытку получить символ из кольцевого буфера ввода. Если в буфере что-то есть (а это ненужные нам символы), первый символ передается в R0 и управление возвращается в программу со сброшенным битом *C* ССП. Команда BCC осуществляет переход, вводится следующий ненужный символ (также в R0) и т. д. Когда буфер опустеет и запрос .TTINR не сможет ввести символ, управление вернется в программу с установленным битом *C* и цикл очистки буфера закончится. Выведя на экран терминала подсказку-требование, программа входит в цикл ожидания ввода командного символа (запрос .TTYIN). После получения символа с клавиатуры и эхоконтроля происходит анализ кода в R0 и переход на нужную ветвь программы.

Ожидание командного символа программным циклом, как это предусмотрено в рассмотренном примере, может использоваться в фоновой программе. Если же такие строки включить в оперативную программу, то в течение всего времени ожидания командного символа фоновая программа выполняться не сможет (так как процессор, находясь в программном цикле, выполняет оперативную программу). Повысить эффективность работы программного комплекса можно следующим образом:

```
     MOV #10100, 44   ; Установка спецрежима
1$:  .TTINR           ; Очистка
     BCC 1 $          ; от "мусора"
     .TTYOUT #'?      ; Вывод подсказки
     BIC #100, 44     ; Сброс бита 6 в ячейке 44
     .TTYIN           ; Ожидание команды
     .TTYOUT          ; Эхо
```

Здесь в ожидании командного символа оперативная программа блокируется средствами ОС, процессор остается свободен и фоновая программа может выполняться.

8.2. ПРОГРАММНОЕ УПРАВЛЕНИЕ ХОДОМ ИЗМЕРИТЕЛЬНО-ВЫЧИСЛИТЕЛЬНОГО ПРОЦЕССА

Для программных комплексов реального времени характерна неоднозначность хода выполнения, зависимость конкретного порядка выполнения тех или иных программных блоков от внешних условий. Так, например, распространенной методикой является переход на программу обработки экспериментальных данных после того, как эти данные нако-

плены. Если задана длительность измерений, то переход на программу обработки осуществляется по сигналу прерывания от таймера – системного или аппаратного. Если же задано требуемое число зарегистрированных событий, то момент перехода на программу обработки должен определяться программно, с помощью счетчика числа событий. Как уже отмечалось выше, при одном источнике измерительной информации прием данных целесообразно вести в режиме ожидания готовности, обеспечивающем заметно большее быстродействие. Если же источников экспериментальных данных несколько и данные поступают несинхронно, придется использовать режим прерываний. Возможная структура такого программного комплекса приведена на рис. 8.3.

Основная программа, выполнив инициализацию установки, блокируется программным запросом .SPND. Программы обработки прерываний принимают данные, каждая от своего источника измерительной информации. В одной из ПОП (или, возможно, в нескольких) ведется счет регистрируемых данных. Пока число событий меньше заданного, ПОП заканчивается командой RTS PC выхода в основную программу. Как только накопится требуемый объем информации, в ПОП запрещаются дальнейшие прерывания и выполняется программный запрос .RSUM снятия блокировки основной программы, которая переходит на строки обработки накопленной информации.

Недостатком приведенной структуры является слишком жесткий характер. Действительно, в программном комплексе не предусмотрено никаких средств управления. Будучи запущен, программный комплекс выполняется единообразно: накапливает заданный объем информации, а затем переходит к ее обработке. Такое построение программы целесообразно лишь в тех случаях, когда измерения носят рутинный единообразный характер. В условиях экспериментальных исследований в программный комплекс необходимо ввести средства управления его работой по командам оператора. Проще всего это сделать с помощью макрокоманд посимвольного ввода. Структура такого программного комплекса приведена на рис. 8.4. Основная программа, выполнив инициализацию аппаратуры и прерываний, устанавливает спецрежим ввода с терминала и выдает программный запрос .TTINR (или .TTYIN). В результате программа блокируется средствами ОС до ввода с клавиатуры терминала управляющего символа. Оператор может в любой момент вмешаться в работу комплекса, нажав на ту или иную клавишу. После анализа введенного символа осуществляется переход на требуемую подпрограмму, а затем возврат на строку с макровызовом .TTINR. Типичный перечень подпрограмм управления выглядит следующим образом:

пуск измерений;

останов измерений;

вывод накопленной информации на экран в виде таблицы или графика;

сброс накопленной информации;

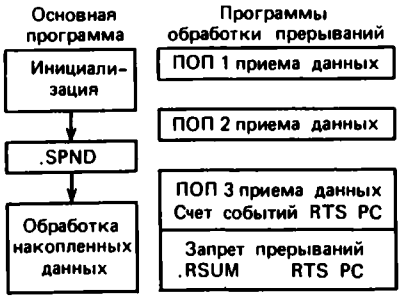


Рис. 8.3. Простейшая структура программного комплекса с обработкой прерываний

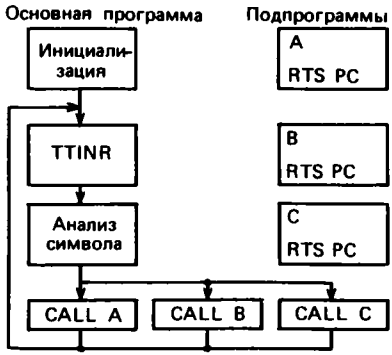


Рис. 8.4. Структура программного комплекса с управлением через клавиатуру

ввод новых условий измерений (например, нового значения времени экспозиции);

вывод по ходу измерений частичных результатов, например истекшего времени или числа зарегистрированных событий;

завершение всей программы с предварительной передачей накопленной информации на НМД;

аварийное завершение программы.

В приведенных примерах предполагалось, что в течение времени накопления измерительной информации основной программе было "нечего делать", что и позволило организовать ожидание команд оператора. Сложнее обстоит дело в тех случаях, когда основная программа, инициализировав аппаратуру, не блокируется, а продолжает выполняться, например, оперативно анализируя поступающую информацию или осуществляя упаковку этой информации и передачу ее на какие-либо ВУ. Возможности управления такой программой зависят от ее организации. Часто алгоритм основной программы представляет собой бесконечный цикл. Если время выполнения одного шага цикла невелико (не превышает 1–2 с), программный запрос на ввод с терминала команды оператора можно включить в цикл в качестве, например, его последней строки. Поскольку в данном случае нельзя допустить блокирование программы в ожидании команды оператора (которая, весьма вероятно, и не поступит), необходимо запретить ожидание терминала в слове состояния задания. Структура программы будет выглядеть следующим образом:

```
        .
        .; Строки инициализации аппаратуры
        .
        MOV  #10100, 44  ; Специальный режим и запрет ожидания
START:  .
        .; Цикл основной программы
        .
```

```
        .TTINR          ; Был введен символ?
        BCC  COMPR      ; Да, переход на его обработку
        JMP  START      ; Нет, на новый шаг цикла
COMPR:  .
        .; Анализ символа и переходы на подпрограммы
        .
        JMP  START      ; Возврат в основной цикл
```

Программный запрос .TTINR в приведенной программе делает попытку забрать символ из кольцевого буфера ввода, находящегося в мониторе. Если в течение предыдущего шага цикла оператор нажал на какую-либо клавишу и переслал таким образом код символа в кольцевой буфер ввода, запрос .TTINR заберет этот код и завершится со сброшенным разрядом *C* в РСП. В результате произойдет переход на строки анализа кода, а затем на требуемую подпрограмму. Если к моменту выдачи запроса .TTINR кольцевой буфер ввода был пуст, запрос завершится с установленным разрядом *C*, что приведет к продолжению цикла.

Рассмотренная методика годится лишь для циклических программ с относительно коротким шагом цикла. Если длительность шага велика или программа вообще не является циклической, для управления программой можно воспользоваться программным запросом .READC. Первый запрос .READC должен входить в строки инициализации, а повторный стоять на конце ППЗ:

```
DISP:   .RAD50 /TT/     ; Спецификация
        .WORD 0         ; устройства
CMD:    .BLKB 2         ; Место для команды
START:  .LOOKUP #ARG, #0, #DISP
        MOV #10100, 44  ; Спецрежим + запрет ожидания
1$:     .TTINR          ; Очистка
        BCC 1$          ; "мусора"
        .PRINT #MES     ; Вывод подсказки
        .READC #ARG, #0, #CMD, #1, #COMP
        .
        .; Продолжение программы
        .
COMP:   .
        .; Анализ символа в CMD, переход на требуемую подпрограмму и
        .; возврат из нее
2 $:    .TTINR          ; Очистка
        BCC 2 $         ; "мусора"
        .PRINT #MES     ; вывод подсказки
        .READC #ARG, #0, #CMD, #1, #COMP
        RTS PC          ; Возврат в программу
MES:    .ASCIZ ⟨16⟩ "ВВОДИТЕ КОМАНДУ:" ⟨17⟩
```

Программный запрос .READC заполняет элемент очереди ввода-вывода, ставит его в очередь к драйверу терминала и возвращает управление в программу. Продолжается выполнение программы, но когда опе-

ратор наберет на клавиатуре какой-либо символ и ВК (в аргументах макрокоманды .READC указан ввод одного слова, т. е. двух символов), произойдет прерывание программы с передачей управления подпрограмме завершения COMP. Введенные символы будут к этому моменту находиться в буфере CMD. В ППЗ следует проанализировать первый символ в CMD и осуществить переход на требуемую подпрограмму. После выполнения ППЗ управление возвращается в прерванную программу. Если механизм управления требуется сохранить, ППЗ должна заканчиваться повторной макрокомандой .READC с теми же аргументами.

Существенным элементом программы управления является очистка кольцевого буфера ввода перед выводом подсказки *Вводите команду:* и запросом на ввод с клавиатуры. Дело в том, что программный запрос .READC вводит символы в буфер прикладной программы не непосредственно из регистра данных клавиатуры, а из кольцевого буфера ввода. Если оператор, вводя команду, по ошибке набрал больше символов, чем указано в макрокоманде .READC, то "лишние" символы после выполнения запроса .READC останутся в кольцевом буфере и будут храниться там до очередного программного запроса на ввод. В этом случае очередной запрос заберет из буфера сначала этот "мусор", и лишь затем – символы очередной команды. В результате как эта, так, скорее всего, и все последующие команды будут восприниматься неправильно. Описанная ситуация может возникнуть и при отсутствии ошибок оператора. Действительно, терминалы ЭВМ при нажатии клавиши ⟨ВК⟩ обычно аппаратно генерируют и код ⟨ПС⟩, а при нажатии ⟨ПС⟩ – и код ⟨ВК⟩. Таким образом, ввод с клавиатуры командного символа и ⟨ВК⟩ заполнит кольцевой буфер тремя кодами, а не двумя, и последний из этих кодов (⟨ПС⟩ или ⟨ВК⟩) будет воспринят следующим запросом на ввод, как первый символ команды. Предусмотренная в программе очистка кольцевого буфера ввода перед каждым программным запросом на ввод с клавиатуры ликвидирует отмеченный недостаток (кстати говоря, отсутствующий в ОС NTS).

Обычно ППЗ, обработав инициировавшее ее асинхронное событие (завершение ввода-вывода, окончание временного интервала), передает управление монитору, который, в свою очередь, осуществляет возврат в прерванную программу. Однако в ряде случаев желательно изменить ход основной программы так, чтобы после отработки ППЗ управление вернулось не в точку прерывания, а в какое-то другое место программы. Для этого можно воспользоваться описанным ранее программным запросом .SPCPS. Поскольку таблица с требуемым адресом перехода, используемая этим макровызовом, находится в прикладной программе, ее можно программно модифицировать, изменяя при необходимости адрес перехода. В частности, этим можно воспользоваться для прерывания хода программы и перехода в требуемую точку по команде оператора. Структура такой программы с управлением может выглядеть следующим образом:

```
ARG:     .BLKW 5                  ; Область для аргументов
SPCTBL:  .WORD 0, 0, 0            ; Таблица для .SPCTBL
CMD:     .BLKB 2                  ; Поле для команды
DISP:    .RAD50 /TT/              ; Спецификация
         .WORD 0                  ; устройства
         .LOOKUP #ARG, #0, #DISP
         MOV #10100,44            ; Спецрежим + запрет ожидания
1$ :     .TTINR                   ; Очистка
         BCC 1$                   ; "мусора"
         .READC #ARG, #0, #CMD, #1, #CMPL
          .
          .; Продолжение программы
          .
SECT1:    .
          .; Участок 1
          .

SECT2:    .
          .; Участок 2
          .

SECT3:    .
          .; Участок 3
          .
CMPL:    CMPB CMD, #'1            ; Нажата клавиша ⟨ 1 ⟩ ?
         BNE 1$                   ; Нет
         MOV #SECT1, SPCTBL       ; Да, установить адрес возврата
         BR 10$                   ; И на выход
1$ :     CMPB CMD, #'2            ; Нажата клавиша ⟨ 2 ⟩ ?
         BNE 2$                   ; Нет
         MOV #SECT2, SPCTBL       ; Да, установить адрес возврата
         BR 10 $                  ; И на выход
2$ :     CMPB CMD, #'3            ; Нажата клавиша ⟨ 3 ⟩?
         BNE 3$                   ; Незапланированный символ
         MOV #SECT3, SPCTBL       ; Установить адрес возврата
10$ :    .SPCPS #ARG, #SPCTBL; Собственно замена адреса
20$ :    .TTINR                   ; Очистка
         BCC 20 $                 ; "мусора"
3$ :     .READC #ARG, #0, #CMD, #1, #CMPL
         RTS PC                   ; Выход из ППЗ.
```

При вводе с клавиатуры команды происходит прерывание программы с переходом на подпрограмму завершения CMPL. Если была введена команда ⟨1⟩⟨ВК⟩, в таблице SPCTBL устанавливается адрес возврата SECT1 на участок 1 основной программы. При вводе команды ⟨2⟩ ⟨ВК⟩ устанавливается адрес возврата на участок 2 и т. д. Программный запрос .SPCPS выполняет фактическую замену адреса возврата, и после окончания ППЗ по команде RTS PC происходит переход на требуемый участок основной программы.

До сих пор предполагалось, что после ввода команды оператора программа продолжается, хотя и изменяется ход ее выполнения. Частным случаем управления программой является ее аварийное завершение,

для чего в ОС предусмотрено специальное средство – управляющий символ ⟨ CTRL/C ⟩. Ввод с клавиатуры двух ⟨ CTRL/C ⟩ (или одного ⟨ CTRL/C ⟩, если программа к этому моменту установила запрос на ввод с клавиатуры) завершает программу средствами ОС. Этим удобно пользоваться при отладке программ, а также для аварийного завершения неправильно начатых измерений. Однако при таком завершении программы все результаты ее работы будут потеряны. В тех случаях, когда в процессе аварийного завершения по команде оператора требуется выполнить какие-либо запланированные заранее действия (передача на диск накопленной информации, закрытие файлов и т. д.), можно воспользоваться программным запросом .SCCA, который "перехватывает" двойное ⟨ CTRL/C ⟩ и сообщает об этом программе установкой 1 в старшем бите предусмотренного для этого слова состояния терминала. Поскольку ввод двойного ⟨ CTRL/C ⟩ в этом случае не вызывает прерывания, программа должна непрерывно опрашивать слово состояния терминала в ожидании ввода команды завершения. Поэтому данный способ управления можно использовать лишь в тех случаях, когда прием измерительной информации осуществляется в режиме прерываний. Структура программы выглядит следующим образом:

```
ARG:    .BLKW 2              ; Область для аргументов
SCCA:   .WORD 0              ; Слово состояния терминала
        .SCCA  #SRG, #SCCA
          .
          .; Инициализация прерываний
          .
        JMP    END           ; На ожидание команды завершения

          .; Программы обработки прерываний
END:    TST    SCCA          ; Цикл ожидания
        BEQ    END           ; команды завершения
          .
          .; Выполнение запланированных перед
          .; завершением программы действий
        .EXIT
```

8.3. ВЫВОД КОНТРОЛЬНОЙ ИНФОРМАЦИИ

Возможность вывода контрольной информации является непременной чертой программных комплексов реального времени. Действительно, в измерительных установках, управляемых от ЭВМ, как правило, отсутствуют средства непосредственного наблюдения за ходом измерений. Судить о протекании процесса измерения или управления можно только по информации, выводимой ЭВМ на внешние устройства – алфавитно-цифровые или графические дисплеи, графопостроители и т. д. в соответствии с заложенными в ЭВМ программами. Поэтому оснащение программного комплекса реального времени такими программами является обя-

зательным условием успешного функционирования измерительно-вычислительной системы.

Программы вывода контрольной информации имеют свои особенности в зависимости от характера выводимой информации (тексты, отдельные числа, таблицы), а также в зависимости от того, чем инициируется вывод – командами оператора, сигналами таймера либо какими-либо событиями реального времени.

Вывод текстов не представляет особого труда. Обычно текстовые сообщения выводятся на экран терминала с помощью программмного запроса .PRINT. Если разных сообщений не очень много, они записываются в тексте программы целиком, с помощью директивы .ASCIZ. Если же программный комплекс имеет существенно диалоговый характер, выводимые сообщения удобно формировать программно из отдельных слов или обрывков фраз, комбинируемых в нужном порядке. В этом случае состав выводимого текста может определяться программой в зависимости от результатов измерений и других внешних условий.

Рассмотрим простой пример программного составления сообщений. Пусть в некоторой точке программы требуется вывести сообщение о состоянии процесса измерений. Состояние процесса описывается переменной FLAG: если FLAG = – 1, идет начальная фаза измерений, если FLAG = 1 – основная фаза, наконец, если FLAG = 0 – завершающая фаза. Занесем в программу элементы сообщений, а также таблицу адресов этих элементов:

```
T1:     .ASCII  <16>"НАЧАЛЬНАЯ "<0>
T2:     .ASCII   "ОСНОВНАЯ "<0>
T3:     .ASCII   "ЗАВЕРШАЮЩАЯ"<0>
T4:     .ASCII   "ФАЗА ИЗМЕРЕНИЙ"<17><0>
        .EVEN
ADRT1:  .WORD T1
ADRT2:  .WORD T2
ADRT3:  .WORD T3
ADRT4:  .WORD T4
```

Составим макроопределение, компонующее законченное сообщение из двух произвольных элементов. Сообщение компонуется в выходном буфере BUFPRN:

```
.MACRO  PRNT   PRNTA, PRNTB
        MOV    #BUFPRN, R2
        MOV    PRNTA, R1
        MOVB   (R1)+, (R2)+
        TSTB   (R1)
        BNE    .-4
        MOV    PRNTB, R1
        MOVB   (R1)+, (R2) +
        TSTB   (R1)
        BNE    .-4
```

```
        CLRB  (R2)
        .PRINT #BUFPRN
.ENDM   PRNT
```

Приведенное выше макроопределение рассчитано на работу с двумя аргументами, однако нетрудно преобразовать его так, чтобы можно было составлять текст из большего и, если нужно, различного числа элементов.

Строки программ, выводящие требуемое сообщение в зависимости от содержимого ячейки с меткой FLAG, могут выглядеть следующим образом:

```
     TST  FLAG              ; Анализ флага
     BMI  1$                ; FLAG = -1
     BEQ  2$                ; FLAG = 0
     PRNT ADRT2, ADRT4      ; FLAG = 1
     BR   5$                ; На продолжение
1$   PRNT ADRT1, ADRT4      ; FLAG = -1
     BR   5$                ; На продолжение
2$:  PRNT ADRT3, ADRT4      ; FLAG = 0
5$:                         ; Продолжение программы
```

Для объединения символьных строк можно было воспользоваться подпрограммой CONCAT из системной объектной библиотеки.

Сложнее обстоит дело с выводом чисел. Как правило, числа хранятся в ОП в двоичной форме; выводить же их на экран терминала (или на печать) следует в десятичной или иногда восьмеричной форме. Поэтому перед выводом числа преобразуют в двоично-десятичную форму. Далее устройства вывода алфавитно-цифровой информации работают в коде ASCII. Это требует дополнительного преобразования каждой десятичной (или восьмеричной) цифры в соответствующий ей код ASCII. В составе объектной библиотеки системы ОС РВ имеются стандартные подпрограммы, выполняющие указанные преобразования, что заметно облегчает вывод числовой информации. В объектной библиотеке системы РАФОС таких подпрограмм, к сожалению, нет, и программисту приходится составлять их самостоятельно.

Рассмотрим несколько примеров вывода контрольной информации. Пусть требуется периодически (например, каждые 5 с) выводить на экран число зарегистрированных событий. Для того чтобы протокол измерений был более наглядным, целесообразно вместе с числом событий выводить текущее время, которое можно получить с помощью программного запроса .GTIM. Однако этот запрос возвращает значение времени (в тактах) в виде двоичного числа, занимающего два слова памяти. Для вывода на экран двоичное представление времени следует преобразовать в число часов, минут и секунд в кодах ASCII, для чего можно использовать подпрограмму TIMASC из системной объектной библиотеки. Будем считать, что число зарегистрированных событий накапливается в однословной ячейке COUNT. Для того чтобы освобо-

дить процессор для выполнения более важной работы, вынесем вывод контрольной информации на уровень ППЗ. В основной программе надо только инициировать процесс вывода, выдав программный запрос к системному таймеру:

```
ARG:     .BLKW  2                ; Область для аргументов
TIME:    .WORD  0, 250.          ; 250 тактов = 5 с
CRNTIM:  .BLKW  2                ; Поле для текущего времени
TBLTIM:  .WORD  2                ; Таблица
         .WORD  CRNTIM           ; аргуметов для
         .WORD  TIMAS            ; подпрограммы TIMASC
TIMAS:   .BLKB  8.               ; Время в ASCII
         .ASCII  "  "            ; 2 пробела
CNTASC:  .BLKB  5                ; Число событий в ASCII
         .BYTE  0                ; Ограничитель строки
COUNT:   .WORD  0                ; Ячейка для числа событий
         .MRKT  #ARG, #TIME, #CONTRL
            .
            .; Продолжение основной программы
            .
; Подпрограмма завершения
CONTRL:  .MRKT  #ARG, #TIME, #CONTRL
         .GTIM  #ARG, #CRNTIM    ; Получить текущее время
         MOV    #TBLTIM, R5      ; Настройка регистра связи
         JSR    PC, TIMSAC       ; Время в коды ASCII
         JSR    PC, CODE         ; Вызов подпрограммы CODE
         .PRINT #TIMAS           ; Вывод сформированного сообщения
         RTS    PC               ; Выход из ППЗ
CODE: ; Подпрограмма преобразования двоичного числа в ячейке COUNT
            .; в 5-разрядное десятичное в кодах ASCII (поле CNTASS)
            .
            .
         RTS    PC
```

Программа устанавливает запрос к системному таймеру (макрокоманда .MRKT) и приступает к сбору и накоплению измерительной информации. При этом предполагается, что программа регулярно модифицирует содержимое ячейки COUNT, засылая в нее текущее значение числа зарегистрированных событий. Через 5 с программа прерывается с передачей управления на ППЗ CONTRL. В этой подпрограмме прежде всего выдается повторный запрос .MRKT с теми же аргументами для организации периодических прерываний основной программы, после чего выполняются действия по перекодировке информации и выводу контрольного сообщения. Программный запрос .GTIM позволяет получить значение текущего времени, выраженное в тактах; это число поступает в двухсловное поле CRNTIM. Для преобразования этого числа в коды ASCII используется подпрограмма TIMASC из системной объектной библиотеки. Эта подпрограмма возвращает значение времени в форме ЧЧ:ММ:СС, где ЧЧ – число часов, ММ – число минут, а СС –

число секунд. Вся строка занимает 8 байт и помещается в поле TIMAS, образуя первую часть выводимого сообщения. Далее с помощью прикладной подпрограммы CODE число событий, находящееся в ячейке COUNT, преобразуется в 5-разрядное десятичное число, представленное в виде кодов ASCII. Эта информация записывается в 5 байтов поля CNTASC, образуя вторую часть выводимого сообщения. Все сообщение заканчивается нулевым байтом, служащим для программного запроса .PRINT идентификатором конца выводимой строки. Наконец, вся строка выводится на экран, образуя, например, сообщение 12:03:46 11679

Воспользовавшись механизмом ППЗ, можно инициировать вывод контрольных сообщений какими-либо событиями реального времени. Предположим, что измерительная информация, поступающая из установки, накапливается в буфере объемом 256 слов (1 блок на диске). После заполнения буфера его содержимое пересылается на диск, а информация продолжает накапливаться во втором буфере такого же объема. Оператору желательно следить за ходом накопления и записи на диск экспериментальных данных. В этом случае для записи на диск используется программный запрос вида

.WRITC # ARG, CHAN, BUF, # 256., # CONTRL, R2

где ячейки CHAN и BUF предназначены для хранения номера канала и адреса текущего буфера соответственно, аргумент 256. определяет число пересылаемых слов, CONTRL – входная метка ППЗ, а в R2 хранится и наращивается номер блока, в который записывается данная порция информации. ППЗ имеет следующую структуру:

```
CONTRL:     .
            .; Перекодировка номера блока в коды ASCII
            .
            .PRINT #STRING
            RTS PC
```

Выводимая строка STRING составляется следующим образом:

```
STRING:   .ASCII   <16>"НА ДИСК ВЫВЕДЕН БЛОК НОМЕР "<17>
NUMBER:   .BLKB    3
          .BYTE    0
          .EVEN
```

В поле NUMBER помещается номер блока в кодах ASCII (предполагается, что в эксперименте будет записано не более 999 блоков). Завершающий нулевой байт, как обычно, является ограничителем выводимой строки.

В приведенных примерах программы работают по заданному алгоритму, не допуская перестройки. Часто желательно вывод контрольной информации осуществлять по командам оператора. Для этого удобно использовать программные запросы .READC или .TTYIN (.TTINR) с последующим анализом введенной команды и переходом на подпрограмму вывода того или иного сообщения. Примеры такого рода были приведены в предыдущем параграфе.

При выводе контрольной информации на экран терминала часто возникает проблема специфического расположения различных сообщений на поле экрана, если программа параллельно выводит сообщения различного характера. Например, по прерываниям от установки могут формироваться сообщения о текущем состоянии процесса: *Найдена кассета №5, Кассета №5 доставлена на позицию измерений, Идут измерения активности, Измерения активности закончены, Идет поиск новой кассеты* и т. д. Эти сообщения желательно выводить на какое-то фиксированное место экрана, называемое *экранным окном*, например на верхнюю строку, так, чтобы они сменяли друг друга. Помимо этого процесса вывода, идущего автоматически, оператор может запрашивать у программы ту или иную информацию о результатах выполненных уже измерений, о наличии в установке готовых к измерениям образцов, о временно́м плане измерений, составляемом программой, и т. д. Всю эту информацию в виде текстов и таблиц желательно выводить также на фиксированное место экрана, например, начиная со второй строки, не заграгивая при этом верхнюю строку с оперативным сообщением о состоянии установки. Наконец, одну или несколько нижних строк экрана полезно отвести под вывод подсказок и ввод команд оператора либо для краткого перечня команд оператора, присущих данной программе, – своего рода *меню* управления измерительно-вычислительным процессом.

Для организации экранных окон на физическом уровне при отсутствии соответствующего системного программного обеспечения следует использовать команды терминала, позволяющие перемещать курсор по экрану, стирать участки текста, передвигать отдельные строки или весь текст и т. д. Рассмотрим упрощенный пример организации экранных окон на экране распространенного дисплея 15ИЭ-00-013.

Дисплей имеет две системы команд, различающиеся своими возможностями (например, стереть экран можно только в системе команд №1, а организовать прямую адресацию курсора – в системе команд №2). Каждая команда представляет собой код или определенную последовательность кодов, посылаемых в РД экрана. Например, код 14_8 в системе команд №1 очищает экран и переводит курсор в его начало; код 27_8 переводит экран в систему команд №2, последовательность кодов 33_8 и 105_8 возвращает экран в систему команд №1 и т. д. Познакомиться с полным перечнем команд можно с помощью комплекта эксплуатационных документов дисплея.

Поскольку прерывания от экрана обрабатывает система и "перехват" этих прерываний приведет к нарушению системных функций, засылку управляющих кодов в РД экрана целесообразно выполнять в режиме ожидания готовности. Для упрощения программирования напишем текст макрокоманды, пересылающей в РД экрана один символ – код:

```
.MACRO D SYM, ?L
L:      TSTB    177564          ; Ожидание
        BPL     L               ; готовности экрана
        MOVB    #SYM, 177566    ; Пересылка символа
.ENDM   D
```

Будем считать для простоты, что требуется независимо выводить всего две строки текста: одну – на первой строке экрана, начиная с позиции 10; вторую – на третьей строке экрана, начиная с позиции 5. Для вывода текста воспользуемся драйвером терминала и программным запросом .WRITW (целесообразность этого будет обоснована ниже). Программные фрагменты организации экранных окон целесообразно оформить в виде подпрограммы:

```
 ; Подпрограмма вывода в первое окно
WNDW1: D  33          ; Установка
       D  131         ; прямой адресации,
       D  40          ; первая строка
       D  ⟨ 40+9.⟩    ; десятая позиция
       D  33          ; Стирание строки
       D  113         ; от курсора
       .WRITW #ARG, #0, MSGADD, MSGLEN, #0
       RTS  PC
 ; Подпрограмма вывода во второе окно
WNDW2: D  33          ; Установка
       D  131         ; прямой адресации,
       D  ⟨40+2⟩      ; третья строка
       D  ⟨40+4⟩      ; пятая позиция
       D  33          ; Стирание строки
       D  113         ; от курсора
       .WRITW #ARG, #0, MSGADD, MSGLEN, #0
       RTS  PC
```

Обе подпрограммы практически одинаковы. Пара кодов 33 и 131 (всюду даются восьмеричные значения кодов) переводит экран в режим прямой адресации. Следующие два кода воспринимаются как координаты курсора: первое число – номер строки, второе – номер позиции в этой строке. Число 40 обозначает первую строку или первую позицию, число 41 – вторую строку или позицию и т. д. Следующая пара кодов 33 и 113 обеспечивают стирание строки от курсора до ее конца. Стирание необходимо для удаления "хвоста" предыдущей фразы, если новая фраза оказалась короче предыдущей. Собственно вывод текста осуществляется программным запросом .WRITW, в аргументах которого указан номер канала (0), адрес ячейки с адресом сообщения (MSGADD) и адрес ячейки с длиной сообщения (MSGLEN).

В начале программы целесообразно очистить экран:

```
D  27    ; Переход в систему команд №2
D  33    ; Переход в систему
D  105   ; команд №1
D  14    ; Очистка экрана
D  27    ; Переход в систему команд №2
```

Весь остальной вывод осуществляется в системе команд №2. Для вывода сообщения в требуемое окно необходимо занести адрес сообщения

и его длину в ячейки MSGADD и MSGLEN, соответственно и вызвать соответствующую подпрограмму:

```
MSG1:    .ASCII <16> "ИДЕТ ПОИСК КАССЕТЫ"  <17>
MSG1LG:  10.        ; Число слов (не байт!) в сообщении
         MOV        #MSG1, MSGADD
         MOV        MSG1LG, MSGLEN
         JSR        PC, WNDW1
```

Необходимость в программном запросе с ожиданием .WRITW вызвана тем, что в одной программе используется и режим ожидания готовности экрана (макрокоманда D), и режим прерываний (вывод сообщений). Если вместо запроса .WRITW использовать более удобную макрокоманду .PRINT, не реализующую блокирование программы до завершения вывода, может получиться, что выполнение циклов ожидания готовности экрана наложится на еще не закончившийся вывод на экран по прерываниям предыдущего сообщения. Это приведет к нарушению работы программы.

Значительно более изящным оказывается вариант программы, в котором программирование экрана на физическом уровне заменено использованием системных средств, а именно программного запроса .TTYOUT, который выводит символ на экран в режиме прерываний. Приведенная выше макрокоманда D примет следующий вид:

```
.MACRO   D  SYM
         .TTYOUT  #SYM
.ENDM    D
```

Тексты подпрограмм WNDW1 и WNDW2 не изменятся, но вместо громоздкого запроса .WRITW теперь можно использовать запрос .PRINT:
.PRINT MSGADD

Фрагмент основной программы с выводом текста также упростится:

```
MOV      #MSG1, MSGADD
JSR      PC, WNDW1
```

Использование экранных окон заметно повышает наглядность вывода и облегчает работу оператора. Дальнейшего улучшения качества текста можно добиться используя прописные и строчные буквы. Для этого надо установить разряд 14 в слове состояния задания: MOV # 40000, 44

После этого символы, вводимые в поле аргумента директив .ASCII или .ASCIZ на нижнем регистре, будут восприниматься при выводе на экран как строчные английские или прописные русские буквы; вводимые на верхнем регистре – как прописные английские или строчные русские буквы. По-прежнему русский текст следует предварять кодом 16, английский – кодом 17.

8.4. ИНТЕЛЛЕКТУАЛЬНЫЕ ПРОГРАММЫ

Интеллектуальными можно назвать программы, изменяющие (самостоятельно или после консультации с оператором) ход своего выполнения в зависимости от внешних по отношению к программе условий: наличия дискового или оперативного пространства, конкретной аппаратной конфигурации, текущей даты или времени суток и т. д. Пример такого рода был приведен в § 7.1, где программа выясняла, нет ли на диске файла с заданным именем, и при наличии такого файла продолжала работать с ним, а при отсутствии – заводила его заново. Придание программе свойств интеллектуальности заметно упрощает работу оператора, избавляя его от необходимости каждый раз перед запуском задания выполнять "вручную" процедуру проверки наличия или, наоборот, отсутствия на диске файлов с определенными именами.

Методика программного анализа состояния дисковой памяти может быть полезна не только на этапе инициализации, но и в реальном времени проведения измерений. Иногда конечный размер файлов зависит от темпа и характера поступающей информации и не может быть определен заранее. В таких случаях, резервируя на диске место для нового файла программным запросом .ENTER, особенно важно убедиться в том, что затребованный объем дискового пространства предоставлен. Для этого достаточно проанализировать состояние разряда *C* РСП и код завершения в байте 52 системной области связи. Если анализ показал нехватку дискового пространства, можно уничтожить ненужные файлы программным запросом .DELETE, осовободив тем самым место для новых данных. Поскольку, однако, уничтожение файлов является весьма ответственной операцией, перед ее выполнением целесообразно, выведя на экран список наличных файлов с их размерами, получить указания от оператора, какие именно файлы можно уничтожить. Информация обо всем дисковом пространстве содержится в *каталоге* (*директории*) НМД, занимающем нулевую дорожку. Каталог может состоять из нескольких сегментов размером 512 слов (два блока) каждый. Начинается каталог с блока, имеющего физический номер 6. Таким образом, первый сегмент каталога занимает блоки 6 и 7, второй сегмент – блоки 10 и 11 (восьмеричные) и т. д. Каталог НГМД содержит только один сегмент.

Информация о файлах, а также о свободном дисковом пространстве хранится в сегменте каталога в виде семисловных записей (рис. 8.5). В первом слове записи указывается статус области, описываемой данной записью. Отдельные биты слова статуса имеют следующие значения (числа восьмеричные):

Содержимое слова статуса	*Значение*
400	Временный файл
1000	Пустая область (оставшаяся от удаленного файла)
2000	Постоянный файл
102000	Постоянный защищенный файл
4000	Конец сегмента

Слово статуса файла	
Имя файла в RADIX-50	
Расширение имени файла в RADIX-50	
Длина файла	
Номер задания	Номер канала
Дата создания	

Рис. 8.5. Запись в каталоге НМД

Прочитав в ОП сегмент каталога, можно последовательным просмотром записей получить информацию об имеющихся файлах, их размерах и расположении. Чтение сегмента осуществляется обычным программным запросом .READ, в котором указывается физический (а не относительный, как обычно) номер начального блока. Для получения доступа к физическим номерам блоков следует при открытии канала воспользоваться специальной *"нефайловой"* формой макровызова .ENTER, в составе аргументов которого вместо имени определенного файла указывается имя всего устройства (DX, RK1 и т. д.).

Спецификация файла записана в каталоге в коде RADIX-50, который широко используется системными программами. Для вывода на экран терминала эта запись должна быть преобразована в коды ASCII, для чего удобно использовать подпрограмму R50ASC из системной объектной библиотеки. Однако эта подпрограмма, так же как и другие подпрограммы ФОРТРАНа, работает в среде *исполняющей системы ФОРТРАНа* – набора системных программ и таблиц, обеспечивающих нормальное выполнение прикладных программ, написанных на ФОРТРАНе. Для того чтобы компоновщик подсоединил к прикладной программе исполняющую систему, главный программный модуль должен представлять собой программу на ФОРТРАНе. Поэтому приведенная ниже программа анализа каталога оформлена как подпрограмма, вызываемая из главного модуля оператором CALL START:

```
ARG:     .BLKW 5              ; Область для аргументов
DEV:     .RAD50 /MX1/         ; Нефайловое
         .WORD 0              ; имя устройства
BUF:     .BLKW 2000           ; Буфер для чтения сегмента
MX:      .RAD50 /MX1/         ; Имя устройства в RADIX-50
NAMRAD:  .BLKW 2              ; Поле для имени файла в RADIX-50
EXTRAD:  .BLKW 1              ; Поле для расширения в RADIX-50
N6:      6                    ; Число символов в имени
N3:      3                    ; Число символов в расширении
DEVASC:  .ASCII "MX1"         ; Имя устройства в ASCII
NAMASC:  .BLKB 6              ; Поле для имени файла в ASCII
         .ASCII "."           ; Символ точка
EXTASC:  .BLKB 3              ; Поле для расширения в ASCII
LEN:     .ASCII "    " <16> "БЛОКОВ. УДАЛИТЬ?  " <17> " Y/N :"
         .BYTE 200            ; Ограничитель строки для .PRINT
         .EVEN                ; Выравнивание
```

```
TARG6:   3, N6, NAMRAD, NAMASC   ; Таблицы для вызова
TARG3:   3, N3, EXTRAD, EXTASC   ; Подпрограммы R50ASC
MES:     .ASCII <16> "ВСЕ ФАЙЛЫ ПРОСМОТРЕНЫ" <17>
NULL:    .BYTE 0                 ; Ограничитель строки
         .EVEN                   ; Выравнивание
SPACE:   .ASCII " "              ; Пробелы
START::  .LOOKUP #ARG, #0, #DEV; Нефайловый LOOKUP
         .READW #ARG, #0, #BUF, #2000, #6
         MOV    #BUF, R1         ; Начало сегмента
         ADD    #10., R1         ; Пропуск заголовка
NEXT:    .PRINT #NULL            ; Перевод строки
         BIT    #4000, (R1)      ; Конец сегмента?
         BNE    OUT              ; Да, на выход
         BIT    #2000, (R1)+     ; Постоянный файл?
         BNE    PERM             ; Да, на обработку записи
         ADD    #12., R1         ; Нет, на следующую
         BR     NEXT             ; запись
PERM:    MOV    #3, R2           ; Переслать 3 слова
         MOV    #NAMRAD, R3      ; Куда
1$:      MOV    (R1)+, (R3)+     ; Пересылается имя файла и
         SOB    R2, 1$           ; расширение
         MOV    R1, -(SP)        ; Сохранение R1
         MOV    #TARG6, R5       ; Преобразование имени
         JSR    PC, R50ASC       ; файла в ASCII
         MOV    #TARG3, R5       ; Преобразование расширения
         JSR    PC, R50ASC       ; в ASCII
         MOV    (SP)+, R1        ; Восстановление R1
         MOV    (R1)+, R0        ; Двоичная длина файла в R0
         MOV    #LEN + 1, R2     ; Адрес поля для числа блоков ASCII
         JSR    PC, BINASC       ; Преобразовать длину в ASCII
         .PRINT #DEVASC          ; Вывод всей строки на терминал
         MOV    #10000, 44       ; Спецрежим
         .TTYIN                  ; Ввод команды
         .TTYOUT                 ; Эхо
         CLR    44               ; Обычный режим
         CMPB   R0, #'Y          ; Введено <Y> ?
         BNE    NOTDEL           ; Нет, не стирать файл
         .DELETE #ARG, #1, #MX   ; Стереть
NOTDEL:  ADD    #4, R1           ; Указатель на следующую запись
         MOV    SPACE, LEN       ; Заполнение пробелами
         MOV    SPACE, LEN + 2   ; поля для
         MOVB   SPACE, LEN + 4   ; длины файла
         JMP    NEXT             ; на следующую запись
OUT:     .PRINT #MES             ; Все файлы просмотрены
         RTS    PC               ; Возврат в главный модуль

; Подпрограмма BINASC преобразования двоичного числа в десятичное с пред-
; ставлением в кодах ASCII. При входе в подпрограмму в R0 – исходное двоич-
; ное число, в R2 – адрес поля для помещения кодов
```

```
BINASC:  MOV   R0, -(SP)        ; Двоичное число в стек
         CLR   R0               ; Очистка R0
1$:      INC   R0               ; Получаем в R0 число десятков
         SUB   #10., (SP)       ; в числе +1, а в стеке – остаток
         BGE   1$               ; после вычитания десятков – 10.
         ADD   #10., (SP)       ; Коррекция на 10 и преобразование
                                ; в ASCII
         DEC   R0               ; коррекция на 1
         BEQ   2$               ; Если от числа что-то осталось,
         JSR   PC, BINASC       ; повторить
2$:      MOVB  (SP)+, (R2)+     ; Очередную цифру в ASCII из стека
         RTS   PC               ; Либо на 2$, либо в программу
```

Программа с помощью нефайлового запроса .LOOKUP открывает канал 0 для доступа к каталогу НГМД (физическое имя MX1). Сегмент каталога, начинающийся с блока 6, считывается в буфер BUF. Регистр R1 используется далее на протяжении программы в качестве указателя анализируемых слов каталога. Первые пять слов каталога занимает заголовок, где указано, в частности, число сегментов в каталоге, длина каждой записи и другая служебная информация. Для пропуска заголовка служит команда ADD # 10., R1. Далее программа входит в цикл, начинающийся с метки NEXT. Каждый шаг цикла анализирует одну запись каталога. Работа программы завершается, когда в первом слове очередной записи (слове статуса) обнаруживается число 4000_8 – признак конца сегмента. В этом случае на экран терминала выводится сообщение о завершении просмотра и командой RTS PC осуществляется возврат в вызываемую программу.

Просмотр записи об очередном файле начинается с анализа его статуса. Удалению подлежат только постоянные файлы, поэтому, если слово статуса содержит код, отличающийся от 2000_8, происходит смещение регистра-указателя R1 и переход к анализу следующей записи. При обнаружении постоянного файла его спецификация (имя и расширение) переносятся в буфер NAMRAD и дважды вызывается подпрограмма R50ASC для преобразования в коды ASCII сначала имени файла (шесть символов), а затем – расширения (три символа). Следует обратить внимание на необходимость сохранения регистров общего назначения перед обращением к программам системной объектной библиотеки. В данном случае сохранению подлежит только регистр R1.

В памяти начиная с метки DEVASC расположена "заготовка" для образования символьной строки со спецификацией имени файла в принятом формате. В заготовку заранее включены имя устройства MX1: и точка, разделяющая имя файла и его расширение. Продолжением этой заготовки служит строка, содержащая вопрос оператору о целесообразности удаления конкретного файла. В начале строки оставлено пустое место для записи в него длины файла в кодах ASCII. Преобразование длины файла в двоичном формате в десятичную форму с представлением

в кодах ASCII осуществляется подпрограммой BINASC, занимающей всего 14 слов памяти и являющейся хорошим примером оптимальных программ общего назначения.

После вывода на экран строки со спецификацией файла, его длиной и вопросом оператору терминал переводится в специальный режим и программа блокируется запросом .TTYIN в ожидании ввода команды оператора. Если оператор вводит команду ⟨ Y ⟩ , программным запросом .DELETE осуществляется удаление файла и программа переходит на анализ следующей записи каталога. В случае нажатия любой другой клавиши строка с макровызовом .DELETE обходится. Перед возвратом на метку NEXT в поле, предназначенное для кодов ASCII длины файла, заносятся пять пробелов. Это предотвращает искажение выводимой информации в том случае, когда после, например, двухзначной длины файла выводится однозначная.

При практическом использовании описанной методики целесообразно сначала сформировать полный список имеющихся на диске файлов, чтобы оператор мог оценить общую ситуацию, и лишь затем затребовать указаний по каждому файлу. Однако, такое усовершенствование заметно усложнит программу.

Обычно результатом работы программы реального времени является один или несколько файлов на НМД, содержащих полученные в процессе измерений данные. При разработке таких программ следует уделить внимание вопросу об именах выходных файлов. Если программа запускается относительно редко и в промежутке между последовательными запусками выходные файлы полностью обрабатываются и уничтожаются, имена файлов определяются в исходном тексте программы с помощью директивы .RAD50. Однако часто файлы, получаемые в процессе измерений, сохраняются, и тогда при каждом прогоне программы следует образовывать файлы с новыми именами. Конечно, имя файла можно каждый раз задавать с клавиатуры терминала, но более удобно предусмотреть в программе средства автоматического назначения и модификации имени файла по некоторому алгоритму. Если измерения повторяются на чаще 1 раза в сутки, имя файла можно образовать из текущей даты, например, DK:25JUN9.DAT (для измерений, проведенных 25 июня 1989 г.) или DK:01FEB0.DAT (для измерений 1 февраля 1990 г.). Естественно, расширение имени может быть иным. Текущую дату в кодах ASCII можно получить с помощью системной подпрограммы DATE, а для преобразования в код RADIX-50 использовать подпрограмму IRAD50 из системной объектной библиотеки. Как и в предыдущем примере, основная программа должна быть написана на ФОРТРАНе; для передачи параметров между подпрограммами удобно использовать COMMON-блок и соответствующую ему программную секцию с атрибутами D, OVR и GBL. При осуществлении преобразования имени файла необходимо иметь в виду, что подпрограмма DATE возвращает дату в формате

DD-MMM-YY, например, 25-JUN-89. Для образования законного имени файла следует убрать дефисы и первую цифру года. Подпрограмма образования имени файла и резервирования места на диске имеет следующий вид:

```
.PSECT  NAMFIL, D, OVR, GBL          ; Секция для передачи параметра
NAMASC: .BLKW 9.                     ; Место для даты в ASCII
        .EVEN                        ; Выравнивание
.PSECT                               ; Переход в программную секцию
DEVRAD: .RAD50 /DK/                  ; Имя устройства
NAMRAD: .BLKW 2                      ; Место для имени файла
EXTRAD: .RAD50 /DAT/                 ; Расширение
ARG:    .BLKW 5                      ; Область для аргументов
TARG:   3, NNN, NAMASC, NAMRAD       ; Таблица для IRAD50
NNN:    6                            ; Получить 6 символов
START:  JSR  PC, FORSEC              ; Получить дату в ASCII
        MOV #NAMASC + 2, R0          ; Преобразование
        MOV #NAMASC + 3, R1          ; даты
        .REPT 3                      ; из
        MOVB (R1)+, (R0)+            ; формата
        .ENDR                        ; DD-MMM-YY
        ADD # 2, R1                  ; в формат
        MOVB (R1), (R0)              ; DDMMMY
        MOV  TARG, R5                ; Вызов подпрограммы
        JSR  PC, IRAD50              ; преобразования ASCII – RADIX-50
        .ENTER #ARG, #0, #DEVRAD, #4
        RTS  PC                      ; Возврат из подпрограммы
```

Программный модуль на ФОРТРАНе должен содержать следующие строки:

```
CALL START
  .
  .
  .
SUBROUTINE FORSEC
LOGICAL*1 DATEC(9)
COMMON /NAMFIL/ DATEC
CALL DATE (DATEC)
RETURN
END
```

В модуле на ФОРТРАНе предусматривается подпрограмма FORSEC, вызываемая из подпрограммы на языке ассемблера. В ней происходит обращение к подпрограмме DATE, получение текущей даты и заполнение ею девятибайтовой логической переменной в COMMON-блоке NAMFIL. Подпрограмма на языке ассемблера сжимает дату и вызывает подпрограмму IRAD50 из системной объектной библиотеки. Резуль-

татом преобразования заполняется поле NAMRAD в спецификации файла. Другие элементы спецификации (имя устройства и расширение) включены в исходный текст программы.

Часто возникает необходимость многократного запуска некоторой программы, которая, в частности, формирует на диске файл с данными. В таких случаях удобно, чтобы программа модифицировала каждый раз имя файла. В операционной системе ОС РВ это делается автоматически: при повторных запусках программы образуются файлы с одним и тем же именем, но различающиеся номером версии. В системе РАФОС такого средства нет, однако можно программно сформировать номер версии, воспользовавшись для этого расширением имени файла. Рассмотрим упрощенный пример такого рода. Пусть программа при первом запуске создает на диске файл DK:IMAGES.000. При последующих запусках создаются файлы с расширениями 001, 002 и т. д. до 009:

```
FILES:   .RAD50 /DK IMAGES/      ; Имя файлов
EXT:     .RAD50 /000/            ; Начальное расширение
ARG:     .BLKW 5                 ; Область для аргументов
START:   MOV    #10., R1         ; Не более 10 файлов
BEGIN:   .LOOKUP #ARG, #0, #FILES
         BCC    NEWFIL           ; Такой файл уже есть
         .ENTER #ARG, #0, #FILES, #16.
         BCS    ERROR            ; Нет места на диске
         BR     CONT             ; Файл создан, на продолжение
NEWFIL:  .PURGE #0                ; Освободить канал
         INC    EXT              ; Модификация расширения
         DEC    R1               ; Еще один файл
         BNE    BEGIN            ; На проверку следующего
         .PRINT #MES1            ; Уже есть 10 файлов
         .EXIT                   ; Аварийное завершение
ERROR:   .PRINT  #MES2           ; Нет места на диске
         .EXIT                   ; Аварийное завершение
CONT:      .
           .; Продолжение программы
           .
MES1:    .ASCIZ "10 FILES  ALREADY"
MES2:    .ASCIZ "DEVICE  FULL"
         .EVEN
```

Программа прежде всего проверяет с помощью запроса .LOOKUP, есть ли на диске файл с расширением 000. Если такого файла нет, монитор возвращает управление в программу с установленным разрядом с РСП, а канал остается свободным. В этом случае организуется файл с исходным расширением и программа продолжается. При обнаружении файла, о чем можно судить по значению разряда *C* РСП, канал 0 открывается, поэтому для повторного использования его надо освободить запро-

сом .PURGE. После этого модифицируется расширение файла и процедура поиска на диске файла с такой спецификацией повторяется. Модификация расширения осуществляется путем арифметического прибавления 1 к коду RADIX-50 сочетания 000. При этом используется то обстоятельство, что коды RADIX-50 сочетаний 000...009 представляют собой ряд возрастающих чисел, различающихся на 1. При необходимости не составляют труда проверка и организация большего чем 10 числа файлов, однако программа модификации расширения заметно усложнится.

При обнаружении на диске всех 10 файлов, а также в случае ошибки при выполнении программного запроса .ENTER, на экране терминала выдаются аварийные сообщения, а программа завершается.

Рассмотренную методику можно использовать и в предыдущем примере, что даст возможность запускать программу несколько раз в течение суток с образованием файлов с именами, например, DK:25JUN9.000, DK:25JUN9.001 и т. д.

В предыдущих примерах так или иначе анализировалось дисковое пространство. В программах, организующих накопление в ОП больших объемов данных, иногда используются *динамические буфера*, т. е. области ОП, динамически подсоединяемые к программе в процессе ее выполнения. Такая методика требует предварительного анализа оперативного пространства, так как использование программой большей памяти, чем есть в наличии, неминуемо приведет к наложению полей данных на монитор (в однозадачной системе) или на оперативное задание (в двухзадачной системе) с необратимым разрушением программного комплекса и системы в целом. Рассмотрим структуру фонового задания, работающего с динамическими буферами. Пусть собственно программа имеет фиксированный размер, но длина буфера для принимаемых данных заранее не определена, так как зависит, например, от конкретной настройки используемых в установке АЦП, в которых предусмотрена возможность изменения числа уровней квантования, т. е. числа каналов.

В программе используется системная область связи задания, в ячейку 50 которой компоновщик помещает самый старший адрес, принадлежащий программе. Для простой программы этот адрес можно определить заранее (из листинга компоновщика) и использовать в программе в виде константы, однако в случае сложной программы, содержащей подпрограммы и программные модули на ФОРТРАНе и компонуемой в зависимости от конкретных условий по-разному, старший адрес необходимо получить из ячейки 50. Поскольку в процессе назначения динамических буферов содержимое ячейки 50 будет изменяться, в программе следует предусмотреть ячейку (назовем ее HILIM), в которой будет сохраняться первоначальное значение старшего адреса.

Программа запрашивает у оператора требуемый размер динамического буфера (либо определяет его сама из каких-то соображений) и преобразует его в двоичную форму, помещая это число в ячейку SIZBIN. Участок анализа доступного оперативного пространства выглядит сле-

дующим образом:

```
MOV     MOV 50, HILIM        ; Старший адрес
        MOV HILIM, R1        ; То же
        ADD SIZBIN, R1       ; Требуемый старший адрес
        BCC SETTOP           ; Адрес больше 177 777
        MOV #-2, R1          ; Тогда сделаем 177 776
SETTOP: .SETTOP R1           ; Получить требуемую память
        CMP R0, R1           ; Хватит ли места?
        BEQ ACQ              ; Хватит!
        SUB HILIM, R0        ; Не хватит, есть столько места

; Преобразование числа в R0 в ASCII и вывод на экран с требованием помощи
; от оператора (уменьшить число каналов АЦП, снять оперативное задание
; и т. д.)
ACQ:    MOV HILIM, R1        ; Старший адрес программы
        ADD #2, R1           ; Адрес начала динамического буфера
; Накопление в ОП, начиная с адреса в R1
```

Сложением содержимого ячейки SIZBIN со старшим адресом программы определяется требуемый старший адрес. Выход его за пределы беззнакового представления чисел приведет к неправильной работе программы – запросу на значительно меньшую память, чем требуется в действительности. Для устранения этой ошибки после операции сложения анализируется разряд *C* РСП и в случае обнаружения переноса ($C = 1$) в R1 заносится максимально возможный адрес 177776_8 (с таким же успехом можно было занести число 157776_8 или еще меньшее, учитывая размер монитора). Программный запрос .SETTOP устанавливает программе новое значение старшего адреса, содержащееся в регистре R1. Если указанный адрес находится за пределами доступной памяти, монитор расширяет пространство задания до последнего свободного адреса. В любом случае новое значение старшего адреса (которое, между прочим, может быть и меньше старого значения) заносится монитором как в ячейку 50, так и в регистр R0. Далее программа проверяет, получила ли она требуемую память, и при ее отсутствии запрашивает у оператора дальнейшие указания. При наличии необходимого оперативного пространства программа приступает к накоплению данных, используя в качестве начального адреса буфера для данных увеличенный на 2 исходный старший адрес программы.

Рассмотренная методика может быть особенно полезна в оперативно-фоновых программных комплексах, когда объем свободной памяти зависит от размеров как фонового, так и оперативного заданий и не может быть определен заранее.

В некоторых случаях желательно предусмотреть в ИВК некоторую свободу в использовании тех или иных модификаций аппаратуры. Программа обслуживания такого ИВК должна определять, какой именно вариант аппаратуры установлен и настраиваться на его характеристики. Иногда это можно сделать анализируя слово статуса устройства. В дру-

гих случаях, когда за используемыми вариантами аппаратуры закреплены различные адреса на системной магистрали, определение аппаратной конфигурации комплекса удобно осуществлять последовательно обращаясь по альтернативным адресам и фиксируя ситуацию тайм-аута, указывающую на отсутствие ВУ по данному адресу. Обычно тайм-аут приводит к системному прерыванию через вектор 4 и аварийному завершению задания. Программный запрос .TRPSET позволяет прикладной программе "перехватить" это прерывание и обработать его программными средствами.

Рассмотрим в качестве примера программу, которая обычно использует для временной организации измерений программируемый таймер, однако при отсутствии последнего может работать с системным таймером при снижении точности временной синхронизации и некотором уменьшении возможностей:

```
ARG:     .BLKW 5                ; Область для аргументов
FLAG:    .WORD 0                ; Флаг системного таймера
MES1:    .ASCIZ "PROGRAMMABLE TIMER"
MES2:    .ASCIZ "SYSTEM TIMER"
         EVEN
START:   .TRPSET #ARG, #TRAP;   "Перехват" тайм-аута
         TST   172540           ; Проверка аппаратного таймера
         TST   FLAG             ; Какой таймер?
         BNE   SYSTIM           ; Системный
PRGTIM:  .PRINT #MES1           ; Вывод сообщения
         BR    CONT             ; На продолжение программы
SYSTIM:  .PRINT #MES2           ; Вывод сообщения
         BR    CONT             ; На продолжение программы
TRAP:    INC   FLAG             ; Был тайм-аут
         RTI                    ; Возврат в программу
CONT:                           ; Продолжение программы
```

После выполнения программного запроса .TRPSET программа обращается по "сомнительному" адресу программируемого таймера. Если таймер присутствует, команда TST 172540 выполняется нормально, системного прерывания не происходит, и программа, проанализировав состояние флага FLAG и выяснив, что он сброшен, выводит сообщение о типе таймера и переходит на продолжение (метка CONT). При отсутствии таймера команда обращения по адресу 172540 не выполнится (процессор не получит сигнал синхронизации исполнителя), возникнет системное прерывание и управление будет передано на ПОП с меткой TRAP. ПОП устанавливает флаг FLAG и командой RTI возвращает управление в программу (на строку TST FLAG). В этом случае выводится информационное сообщение об отсутствии программируемого таймера и программа продолжается с той же метки CONT. При этом предполагается, что в необходимых местах по ходу программы анализируется состояние флага FLAG и выполняются разветвления в зависимости от состояния флага и, следовательно, типа используемого таймера.

СПИСОК ИСПОЛЬЗОВАННЫХ СОКРАЩЕНИЙ

АДС – адаптер дистанционной связи
АПД – аппаратура передачи данных
АЦП – аналого-цифровой преобразователь
БОО – блок описания окна
БОР – блок описания района
БР – буферный регистр
БСР – блок согласующих резисторов
ВК – возврат каретки
ВОЛС – волоконно-оптическая линия связи
ВС – вычислительная система
ВУ – внешнее устройство
ДВК – диалоговый вычислительный комплекс
Дш – дешифратор
ИВК – измерительно-вычислительный комплекс
ИВС – измерительно-вычислительная система
ИРПС – интерфейс радиальный последовательный
ИРПР – интерфейс радиальный параллельный
КПВ – коммутационное поле векторов
КПЗ – коммутационное поле запросов
МА – мультиплексор адреса
МПИ – микропроцессорный интерфейс
НГМД – накопитель на гибких магнитных дисках
НМД – накопитель на магнитных дисках
НМЛ – накопитель на магнитной ленте
ОП – оперативная память
ОС – операционная система
ОШ – общая шина
ПДП – прямой доступ к памяти
ПИ – последовательный интерфейс
ПО – программное обеспечение
ПОП – программа обработки прерываний
ППЗ – подпрограмма завершения
Пр – преобразователь
Прд – передатчики
Прм – приемники
ПС – перевод строки
Р – регистр
РАД – регистр адреса данных
РАС – регистр активной страницы
РАШ – регистр адреса шины
РБ – регистр буферный
РД – регистр данных
РЗМ – регистр запросов и маски запросов
РИ – регистр инкрементный
РИФ – расширитель интерфейса
РКС – регистр команд и состояний
РОН – регистры общего назначения
РП – разрешение передачи
РПД – разрешение прямого доступа
РПР – регистр прерываний
РСБ – регистр старшего байта
РСП – регистр состояния процессора
РСЧ – регистр-счетчик
РУС – регистр управления и состояния
СК – счетчик команд
ССП – слово состояния процессора
Т – триггер
ТТЛ – транзисторно-транзисторная логика
УАПП – универсальный асинхронный приемо-передатчик
УПО – устройство последовательного обмена
ЦАП – цифро-аналоговый преобразователь
ЦП – центральный процессор
ЭО – электронное оборудование

КРАТКОЕ ОПИСАНИЕ НАИБОЛЕЕ УПОТРЕБИТЕЛЬНЫХ СИСТЕМНЫХ МАКРОКОМАНД РАФОС

arg – всюду поле для аргументов.

.CDFN arg, adr, num – определить область с адреса adr в программе пользователя как область num каналов ввода–вывода вместо области под 16 каналов в мониторе.

.CHAIN – передать управление другой фоновой программе.

.CHCOPY arg, chn, chn1 – скопировать открытый канал с номером chn1 в канал chn другого задания.

.CLOSE chn – закрыть канал chn.

.CRAW arg, wdb – создать (и отобразить) окно, описанное в блоке описания окна wdb.

.CRRG arg, rdb – создать район, описанный в блоке описания района rdb.

.CSISPC outfil, ext, instrg, inbuf – вызвать интерпретатор командной строки с целью проверки входной командной строки, находящейся по адресу instr (или вводимой с клавиатуры терминала), и размещения образованных спецификаций файлов по адресу outfil. По адресу ext могут размещаться команды расширения имен файлов, действующие по умолчанию, а по адресу inbuf сохраняться входная строка.

.DELETE arg, chn, fil – удалить файл, описанный в поле fil по каналу chn.

.DEVICE arg, adr – загрузить после завершения программы регистры ВУ, указанные в поле adr информацией, приведенной там же.

.DSTATUS adr, dev – получить в поле adr информацию от устройства, описанного в поле dev.

.ENTER arg, chn, fil, len – создать файл с именем, описанным в поле fil, по каналу chn размером len блоков.

.EXIT – завершить задачу.

.GTIM arg, ctim – получить в поле ctim значение текущего времени.

.GVAL arg, offs – получить в R0 содержимое ячейки монитора с фиксированным смещением offs.

.INTEN pri – переключить монитор в системное состояние с приоритетом процессора pri.

.LOCK/.UNLOCK – запереть/отпереть USR.

.LOOKUP arg, chn, fil – открыть канал связи chn с файлом (или устройством), описанном в fil.

.MAP arg, wdb – отобразить окно, указанное в блоке описания окна wdb.

.MRKT arg, tim, cmpl, req – поставить запрос с номером req системному таймеру, чтобы через время, указанное в поле tim, прервать программу и передать управление подпрограмме завершения cmpl.

.PRINT txt – вывести на экран терминала сообщение с адресом txt.

.PROTECT arg, vec – защитить вектор прерываний с адресом vec.

.PURGE chn – закрыть канал chn.

.PVAL arg, offs, vl – поместить значение vl в ячейку монитора с фиксированным смещением offs.

.QSET que, nmb – создать nmb элементов очереди по адресу que.

.RCVD arg, buf, wrd – получить wrd слов из другого задания в буфер buf; после постановки запроса на получение задание продолжается.

.RCVDC arg, buf, wrd, cmpl – получить wrd слов из другого задания в буфер buf; после постановки запроса на получение задание продолжается. По завершении получения необходимых данных задание прерывается и управление передается подпрограмме завершения cmpl.

.RCVDW arg, buf, wrd – получить wrd слов из другого задания в буфер buf; после постановки запроса задание блокируется до завершения процесса получения данных.

.RENAME arg, chn, fils – переименовать файл по каналу chn; старое и новое имена файла описаны в поле fils.

.REOPEN arg, chn, adr – восстановить канал chn, сохраненный запросом .SAVESTATUS по адресу adr.

.SAVESTATUS arg, chn, adr – сохранить канал chn по адресу adr.

.SCCA arg, adr – запретить завершение задания по CTRL/C; в случае ввода двойного CTRL/C установить бит 15 слова состояния терминала по адресу adr.

.SDAT arg, buf, wrd – передать wrd слов в другое задание из буфера buf; после постановки запроса на передачу задание продолжается.

.SDATC arg, buf, wrd, cmpl – передать wrd слов в другое задание из буфера buf; после постановки запроса на передачу задание продолжается. По завершении передачи задание прерывается и управление передается на подпрограмму завершения cmpl.

.SDATW arg, buf, wrd – передать wrd слов в другое задание из буфера buf; после постановки запроса на передачу задание блокируется до ее завершения.

.SPND/.RSUM – заблокировать/разблокировать задание.

.SYNCH sbl – перейти с уровня прерываний на уровень подпрограмм завершения; в качестве элемента очереди используется поле sbl.

.TTINR – ввести в R0 один символ из кольцевого буфера ввода.

.TTYIN chr – ввести в R0 и в поле chr один символ из кольцевого буфера ввода.

.TTOUTR – вывести из R0 один символ в кольцевой буфер вывода (фактически на экран).

.TTYOUT chr – вывести из поля chr один символ в кольцевой буфер вывода (фактически на экран).

.TWAIT arg, tim – заблокировать задание на время, указанное в поле tim.

.WRITE arg, chn, buf, wrd, blk – вывести wrd слов из буфера buf в файл (устройство) по каналу chn, начиная с блока blk; после постановки запроса на вывод задание продолжается.

.WRITC arg, chn, buf, wrd, cmpl, blk – вывести wrd слов из буфера buf в файл (устройство) по каналу chn, начиная с блока blk; после постановки запроса на вывод задание продолжается. По завершении вывода задание прерывается и управление передается на подпрограмму завершения cmpl.

.WRITW arg, chn, buf, wrd, blk – вывести wrd слов из буфера в файл (устройство) по каналу chn, начиная с блока blk; после постановки запроса на вывод задание блокируется до его завершения.

СПИСОК ЛИТЕРАТУРЫ

1. **Балашов Е.П., Григорьев В.Л., Петров Г.А.** Микро- и миниЭВМ /Учебное пособие для вузов. Л.: Энергоатомиздат, 1984.

2. **Бирюков В.В., Рыбаков А.В., Шакула Ю.П.** Введение в систему программирования ОСРВ. М.: Финансы и статистика, 1986.

3. **Бруснецов Н.П.** Микрокомпьютеры. М.: Наука, 1985.

4. **Велихов Е.П., Выставкин А.Н.** Проблемы развития работ по автоматизации научных исследований: Препринт. М.: ИРЭ АН СССР, 1983. №3 (358), 52 с.

5. **Вигдорчик Г.В., Воробьев А.Ю., Праченко В.Д.** Основы программирования на Ассемблере для СМ ЭВМ. М.: Финансы и статистика, 1983.

6. **Виноградов В.И.** Дискретные информационные системы в научных исследованиях. М.: Атомиздат, 1976.

7. **Гилл А.** Программирование на языке ассемблера для PDP-11: Пер. с англ. М.: Радио и связь, 1983.

8. **Каган Б.М., Сташин В.В.** Основы проектирования микропроцессорных устройств автоматики. М.: Энергоатомиздат, 1987.

9. **Кичев Г.Г., Некрасов Л.П.** Архитектура и аппаратные средства мини-ЭВМ СМ-1600. М.: Машиностроение, 1988.

10. **Колпаков И.Ф.** Электронная аппаратура на линии с ЭВМ в физическом эксперименте. М.: Атомиздат, 1974.

11. Курочкин С.С. Системы КАМАК – Вектор. М.: Энергоиздат, 1981.

12. **Куценко А.В., Полосьянц Б.А., Ступин Ю.В.** Мини-ЭВМ в экспериментальной физике. М.: Атомиздат, 1975.

13. **Малые ЭВМ и их применение** / Под ред. Б.Н. Наумова. М.: Статистика, 1980.

14. **Микропроцессоры.** Учебник для вузов в 3 книгах: Кн. 1: Архитектура и программирование микроЭВМ. Организация вычислительных процессов / П.В. Нестеров, В.Ф. Шаньгин, В.Л. Горбунов и др. М.: Высшая школа, 1986.

15. **МикроЭВМ.** Практическое пособие в 8 книгах / Под ред. Л.Н. Преснухина. Кн. 1: Семейство ЭВМ "Электроника 60" / И.Л. Талов, А.Н. Соловьев, В.Д. Борисенков. М.: Высшая школа, 1988. Кн. 2: Персональные ЭВМ / В.С. Кокорин, А.А. Попов, А.А. Шишкевич. М.: Высшая школа, 1988.

16. **Мини-** и микроЭВМ семейства "Электроника" / Б.Л. Толстых, И.Л. Талов, В.Г. Цывинский и др. М.: Радио и связь, 1987.

17. **Мячев А.А., Иванов В.В.** Интерфейсы вычислительных систем на базе мини- и микроЭВМ / Под ред. Б.Н. Наумова. М.: Радио и связь, 1986.

18. **Науман Г., Майлинг В., Щербина А.** Стандартные интерфейсы для измерительной техники: Пер. с нем. / Под ред. А.С. Бондаревского. М.: Мир, 1982.

19. **Никитин А.М.** Интерфейсы микропроцессорных систем для автоматизации научных исследований: Учеб. пособие. М.: МИФИ, 1986.

20. **Никитюк Н.М.** Программно-управляемые блоки в стандарте КАМАК. М.: Энергия, 1977.

21. **Певчев Ю.Ф., Финогенов К.Г.** Автоматизация физического эксперимента: Учеб. пособие для вузов. М.: Энергоатомиздат, 1986.

22. **Операционная** система ОСРВ СМ ЭВМ: Справочное издание / Г.А. Егоров, В.Л. Кароль, И.С. Мостов и др. М.: Финансы и статистика, 1987.

23. **Операционная** система СМ ЭВМ РАФОС: Справочник / Л. И. Валикова, Г. В. Вигдорчик, А. Ю. Воробьев, А. А. Лукин; Под ред. В. П. Семика. М.: Финансы и статистика, 1984.

24. **Сингер М.** Мини-ЭВМ PDP-11: Программирование на языке ассемблера и организация машины: Пер. с англ. / Под ред. Ю. М. Баяковского. М.: Мир, 1984.

25. **Ступин Ю. В.** Методы автоматизации физических экспериментов и установок на основе ЭВМ. М.: Энергоатомизадат, 1983.

26. **СМ ЭВМ:** Комплексирование и применение / Г. А. Егоров, К. В. Песелев, В. В. Родионов и др.; Под ред. Н. Л. Прохорова. – 2-е изд., перераб. и доп. М.: Финансы и статистика, 1986.

27. **Уокерли Дж.** Архитектура и программирование микроЭВМ. В 2 кн.: Пер. с англ. / Под ред. А. Г. Филиппова. М.: Мир, 1984.

28. **Финогенов К. Г.** Мини-ЭВМ в физическом эксперименте: Основы программирования задач реального времени: Учеб. пособие. М.: МИФИ, 1983.

29. **Финогенов К. Г., Зубец А. А., Шарак М. П.** Мини-ЭВМ в физическом эксперименте: Операционные системы реального времени: Учеб. пособие. М.: МИФИ, 1983.

30. **Финогенов К. Г.** Средства связи измерительной аппаратуры с ЭВМ: Учеб. пособие. М.: МИФИ, 1987.

31. **Финогенов К. Г.** Мини-ЭВМ в физическом эксперименте: Системное программирование: Учеб. пособие. М.: МИФИ, 1988.

32. **Фрэнк Т. С.** PDP-11: Архитектура и программирование: Пер. с англ. М.: Радио и связь, 1986.

33. **Хазанов Б. И.** Интерфейсы измерительных систем. М.: Энергия, 1979.

34. **Хвощ С. Т., Варлинский Н. Н., Попов Е. А.** Микропроцессоры и микроЭВМ в системах автоматического управления: Справочник. Л: Машиностроение, 1987.

35. **Экхауз Р., Моррис Л.** Мини-ЭВМ: организация и программирование: Пер. с англ. / Под ред. Г. П. Васильева. М.: Финансы и статистика. 1983.

36. Микропроцессоры и микропроцессорные комплекты интегральных микросхем: Справочник. В 2 т. М.: Радио и связь, 1988.

37. **Задков В. Н., Пономарев Ю. В.** Компьютер в эксперименте: Архитектура и программные средства систем автоматизации. М.: Наука, 1988.

38. **Захаров И. В.** Техническое обслуживание и эксплуатация микроЭВМ "Электроника 60М". М.: Машиностроение, 1989.

УКАЗАТЕЛЬ ОПИСАННЫХ В КНИГЕ СИСТЕМНЫХ МАКРОКОМАНД РАФОС

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

ОГЛАВЛЕНИЕ

Производственное издание

Финогенов Кирилл Григорьевич

ПРОГРАММИРОВАНИЕ ИЗМЕРИТЕЛЬНЫХ СИСТЕМ РЕАЛЬНОГО ВРЕМЕНИ

Редактор издательства *Н.А. Медведева*
Художественный редактор *Т.А. Дворецкова*
Технические редакторы *М.А. Канониди, Т.Н. Тюрина*
Корректор *С.В. Малышева*
ИБ № 2938

Набор выполнен в издательстве. Подписано в печать с оригинала-макета 22.03.90. Т-08224. Формат 60 x 88 1/16. Бумага офсетная № 2. Печать офсетная. Усл. печ.л. 15,68. Усл.кр.-отт. 15,68. Уч.-изд.л. 18,56. Тираж 34 000 экз. Заказ 6987. Цена 1 р. 30 к.
Энергоатомиздат, 113114, Москва, М-114, Шлюзовая наб., 10.

Отпечатано в ордена Октябрьской Революции и ордена Трудового Красного Знамени МПО "Первая Образцовая типография" Государственного комитета СССР по печати. 113054, Москва, М-54, Валовая ул., 28.