ПРОЕКТИРОВАНИЕ СИСТЕМ С МИКРОКОМПЬЮТЕРАМИ

M. Freedman, L. Evans

Designing systems with microcomputers

Prentice-Hall, Inc., Englewood Cliffs
1983

М. Фридмен
Л. Ивенс

ПРОЕКТИРОВАНИЕ СИСТЕМ С МИКРО-КОМПЬЮТЕРАМИ

Перевод с английского
канд. техн. наук **В. В. Копьева, А. И. Роговского**

под редакцией д-ра техн. наук **Я. А. Хетагурова**

Москва «Мир» 1986

ББК 32.973
Ф 88
УДК 681.3

Фридмен М., Ивенс Л.

Ф 88 Проектирование систем с микрокомпьютерами: Пер. с англ. — М.: Мир, 1986. — 405 с., ил.

В книге американских специалистов рассмотрена методика проектирования микрокомпьютеров как единого процесса создания программных и аппаратных средств. При составлении программ на языках высокого уровня рекомендован модульный принцип, упрощающий процесс их отладки. Даны описания широко распространенных за рубежом вычислительных систем, которые могут найти применение в автоматизированных комплексах управления производственными процессами.

Для разработчиков вычислительных систем и программистов, а также студентов соответствующих специальностей вузов.

Ф $\frac{2405000000-028}{041(01)-86}$ 179—86, ч. 1

ББК 32.973
6Ф7.3

Редакция литературы по информатике и электронике

Предисловие редактора перевода

Интенсивное развитие микроэлектроники и повышение степени интеграции открыли новое направление в ВТ — создание микропроцессоров и микрокомпьютеров. Появились вычислительные системы с малым уровнем потребления энергии и универсальными возможностями, которые позволяют решать задачи управления объектами различной физической природы. На основе их применения снижаются затраты на автоматизацию основных технических и вспомогательных процессов. В результате будет решена задача комплексной автоматизации производства во всех отраслях народного хозяйства. Это позволит увеличить производительность труда, уменьшить себестоимость выпускаемой продукции и значительно сократить ручные операции в промышленности и сельском хозяйстве. Однако для широкого развития работ в данном направлении необходимо готовить значительное число инженеров-системотехников, умеющих создавать и применять микропроцессоры и микрокомпьютеры. Кроме того, следует выпускать и много инженеров-математиков, разрабатывающих соответствующее программное обеспечение. Для их обучения необходимо иметь учебники и учебные пособия, посвященные аппаратным и программным средствам современной вычислительной техники.

Несмотря на то что в нашей стране выпущено значительное количество литературы, посвященной разработке и применению микропроцессоров и микрокомпьютеров, предлагаемая читателю книга представляет определенный интерес комплексным подходом и методической целостностью. В ней значительное внимание уделяется разработке документации, необходимой для проектирования систем с микропроцессорами и микрокомпьютерами. Пользуясь этой книгой, читатель, не обладающий глубокими знаниями в области вычислительной техники, сможет достаточно быстро разобраться и в рабочих программах и в способах их составления. Авторы предлагают четыре метода автоматизации проектирования, существенным образом повышающие производительность труда программистов и сокращающие затраты времени на составление и отладку рабочих программ. Включение данных материалов безусловно повышает

ценность книги и позволяет рекомендовать ее специалистам, которые обладают определенным опытом в разработке программного обеспечения.

При чтении книги следует пользоваться той последовательностью изложения, которую рекомендуют авторы. Это упрощает изучение материалов, представленных в ней. Большое число приведенных примеров помогает закрепить знания. Книга написана простым и ясным языком, содержит много иллюстраций и фрагментов программ.

При переводе книги встретились определенные трудности, связанные с необходимостью использования более привычных для советских специалистов терминов. Но для ряда терминов сохранено авторское толкование. В частности, был оставлен без изменения термин «конвертирование программы», под которым следует понимать преобразование описания системы на одном алгоритмическом языке в программу на другом языке с сохранением всех алгоритмических соотношений.

Книга будет полезна инженерам, работающим в области создания аппаратных и программных средств микропроцессоров и микрокомпьютеров, а также аспирантам и студентам, изучающим вычислительную технику.

Перевод книги выполнен А. И. Роговским (гл. 1—3, 5—11, приложения А, Б, В, Г, библиография, предметный указатель) и канд. техн. наук В. В. Копьевым (гл. 4).

Я. А. Хетагуров

Вступительное слово

В делах людей бывает миг прилива:
Он мчит их к счастью, если не упущен,
А иначе все плавание их жизни
Проходит среди мелей и невзгод.

В. Шекспир, «*Юлий Цезарь*», акт IV, сцена III
(перевод *И. Б. Мандельштама*).

В подобном положении, когда от выбора пути зависит очень многое, находимся и мы, проектировщики систем, как начинающие, так и ветераны. В том, что микропроцессор «спущен на воду и готов к отплытию», нет сомнения, но готовы ли мы управлять им, вот в чем вопрос. Поэтому основным содержанием данной книги является руководство по системному проектированию микропроцессорных и микрокомпьютерных систем.

Очевидно, что темпы роста производства систем на основе ЭВМ поистине изумительны: за промежуток времени с конца 50-х до начала 70-х годов число ЭВМ удваивалось и утраивалось столь часто, что этот рост был почти экспоненциальным. Впоследствии, когда сделала большой скачок в своем развитии микроэлектроника, произошло смещение в сторону малых (но выпускающихся в значительно бóльших количествах) вычислительных систем. В настоящее время производство и сбыт микропроцессоров расширяется с такой скоростью, что прежние темпы роста кажутся незначительными. Однако такой быстрый рост не обходится без издержек.

Главной проблемой, вставшей сейчас перед проектировщиками вычислительных систем, является проблема обеспечения быстро расширяющегося сообщества конечных пользователей удобным интерфейсом. По существу проблема заключается в том, чтобы создать такие вычислительные системы, которые позволили бы конечному пользователю выполнять с помощью ЭВМ необходимые действия без глубокого изучения в полном объеме специальной литературы по вычислительной технике. Хорошей базой для процесса создания является развитие самого процесса, а именно разработка более мощных и экономичных средств и методов проектирования систем.

Основным толчком для этого послужило быстрое развитие микроэлектронной технологии, которое позволило снизить стоимость аппаратных средств и в тоже время значительно увеличить возможности программируемого оборудования за счет бóльшего объема памяти, более полных наборов команд и т. д. Трудность заключается в том, что, хотя стоимость аппаратуры снизилась значительно, стоимость изготовления программного

обеспечения снижалась гораздо медленнее Результатом явилось возрастание диспропорции в распределении стоимости типичных микропроцессорных систем: стоимость программного обеспечения в настоящее время превосходит стоимость аппаратуры (по оценке на 1982 г. она составляла 90 % от общей стоимости систем).

Если в качестве препятствия более полному использованию возможностей микроэлектроники рассматривается только экономическая сторона дела, это еще не является самым критическим фактором. Более критичным является состояние дела с людскими ресурсами. В то время как количество и сложность аппаратуры продолжали возрастать значительными темпами (и этому росту не видно никаких физических ограничений), соответствующий рост программного обеспечения систем ограничен как интеллектуальным, так и социальным уровнем развития общества. Трудность состоит в том, что расширение возможностей аппаратуры приводит к соответствующему расширению круга задач программного обеспечения. Поскольку производительность труда при разработке программного обеспечения продолжает оставаться сравнительно низкой, удовлетворение возросшего спроса на программы возможно только за счет дополнительного привлечения людских ресурсов. К сожалению, возможности нашей системы образования по подготовке квалифицированных кадров крайне ограниченны. Даже удвоение числа учебных центров по подготовке программистов и увеличение числа обучающихся не решит проблему, поскольку время обучения длится годами и на подготовку существующих кадров проектировщиков затрачены десятилетия.

Таким образом, очевидно, что решение заключается в другом, а именно в увеличении производительности труда в секторе разработки программного обеспечения и систем. Такова основная тема данной книги.

Авторы, Фридмен и Ивенс, являются специалистами в области проектирования, которым приходилось решать проблемы производительности на стыке науки и индустрии. Предлагаемая ими методология основана на личном богатом опыте, усовершенствованном практикой и преподаванием, и включает высокоуровневые и структурные концепции для решения проблемы повышения производительности при проектировании микропроцессорных систем. Короче говоря, эта методология определяет основные направления развития микроэлектроники.

Они выбрали правильный курс: эта книга, их путеводитель, начинается с важной предпосылки, что она предназначена прежде всего для нужд пользователя/заказчика, что именно ему она должна помочь в его путешествии и достижении намеченной цели. В соответствии с этим определен язык проектирования,

являющийся привычным и естественным для пользователя/заказчика, поскольку выбор средства общения с самого начала помогает устранить неопределенность и периодические издержки на трансляцию и интерпретацию, хотя эти задачи относятся скорее к области лингвистики или юрисдикции, чем к области творческой деятельности проектировщика. Далее предложена методология для определения и анализа целей заказчика и путей их достижения. Неоднократно подчеркивается необходимость всеобъемлющего планирования и методичной реализации этих планов в рамках аппаратной структуры, функциональным характеристикам которой придается особое значение. Постоянно утверждается необходимость разработки модульной структуры и объединения модулей. Особый упор делается на объединение аппаратных средств и программного обеспечения. Отладка, тестирование и сопровождение всех компонентов системы являются постоянным предметом обсуждения. Кроме того, периодическое обращение к одним и тем же примерам в тексте книги помогает объединению всех аспектов темы.

Итак, момент настал, книга перед вами, приступайте к чтению! Возможности для тех, кто хочет отправиться в страну микропроцессоров, самые благоприятные. Счастливого пути!

К. С. Смит

Торонто, Канада

Предисловие

Введение в книгу приведено в начале гл. 1. В *предисловии для преподавателя* обсуждается основанный на материалах книги двухсеместровый курс, который читается в Мичиганском университете, описываются технические средства, используемые при чтении курса, приводятся рекомендации по организации и преподаванию полного университетского и краткого курсов. Здесь же хочется рассказать о том, как мы пришли к намерению написать эту книгу.

Несколько лет назад М. Дэвиду Фридмену было предложено реорганизовать и развить курсы по мини- и микрокомпьютерам, читавшиеся на факультете Электротехники университета. Учитывая опыт преподавания и работы в промышленности, было решено создать новый курс, в котором микрокомпьютер рассматривался бы с системной точки зрения. С этой целью в программу курса были включены такие концепции, как *структурное программирование* и *модульное нисходящее проектирование*, что позволило рассматривать предмет с точки зрения *системотехники*. Курс был построен таким образом, что объединял концепции, ранее входившие в разные циклы лекций, и в то же время был *самостоятельным*, с тем чтобы подготовка студента, прослушавшего его, соответствовала возросшим потребностям промышленности. Нам хотелось, чтобы курс был и интересным, и практичным.

Поскольку подходящего учебника в то время не существовало, подготовкой материала нам пришлось заняться самостоятельно. Стремясь к тому, чтобы книга была независимой, мы обнаружили, что она может служить не только учебником для университета, но также и справочным пособием для специалистов или основой краткого курса для технического персонала и программистов в промышленности. В качестве последнего она успешно использовалась авторами. Книга может быть также использована владельцами персональных ЭВМ, которые сами программируют свои задачи, чтобы помочь организовывать и разрабатывать программное обеспечение с системных позиций.

Каждый из вопросов рассматривается в книге не настолько глубоко, насколько это возможно. Для этого потребовалось бы

написать книгу энциклопедических размеров. Материал построен так, чтобы у студента выработалось представление о каждой концепции, достаточное для проектирования и реализации нетривиальной микрокомпьютерной системы в рамках курсового проекта. В качестве дополнительного материала могут служить руководства, поставляемые вместе с микрокомпьютером. Для того чтобы читатель мог углубить знания по интересующим его вопросам, некоторые из этих руководств приведены в списке литературы, в котором источники перечислены в соответствии с тематикой.

Рукопись книги создавалась с помощью микрокомпьютерной системы обработки текстов, а окончательный вариант книги был получен с помощью фотопечатающего устройства. Мы благодарим руководство Bendix Corporation за предоставленную возможность использования системы обработки текстов и особенно сотрудниц Пенни Дей, Кейрин Хинди, Вирджинию Хенадел и Джуди Браун, которые вводили рукопись в систему. Мы хотим также поблагодарить сотрудников Prentice-Hall, в особенности Бернарда М. Гудвина, нашего редактора, за его постоянную поддержку, анонимных рецензентов, чья критика всегда воспринималась благожелательно, профессора Мичиганского университета Мюррея Х. Миллера, который предложил нам разработать курс, и профессора К. С. Смита из Торонтского университета за его критические замечания и вступительное слово к нашей книге. И наконец, мы хотим выразить признательность нашим ближним за их постоянную поддержку и одобрение: Роберте, Сандре и Крейгу со стороны Фридмена; Кэти и Лайзе со стороны Ивенса.

Нашим многочисленным друзьям, коллегам и студентам, которые неоднократно задавали вопрос: «Когда же будет закончена книга?», мы можем наконец ответить: «Она перед Вами!»

Саусфилд, Мичиган — *М Дэвид Фридмен*

Бока-Рейтон, Флорида — *Ленсинг Б. Ивенс*

Предисловие для преподавателя

Предисловие для преподавателя включает:

- Описание основанного на материалах книги двухсеместрового курса, который читается в Мичиганском университете.
- Описание технических средств, необходимых студентам для проектирования и реализации микрокомпьютерной системы в рамках курсового проекта.
- Рекомендации по использованию других конфигураций технических средств, необходимых для преподавания данного курса.
- Предложения по организации других полных университетских и кратких курсов, в которых могут быть использованы материалы книги.

В Мичиганском университете курс преподается студентам старшего курса и аспирантам. Первый семестр состоит из цикла трехчасовых лекций по каждой из тем в том порядке, в котором они расположены в книге. В течение семестра выполняется ряд домашних заданий в рамках курсового проекта по микрокомпьютерным системам. В конце семестра каждый студент должен подготовить вариант проекта, составленный на языке проектирования.

Задания обсуждаются и оцениваются на семинарах, поскольку все студенты проектируют по существу одну и ту же систему. После обсуждения задания по разработке модульной структуры проекта у каждого студента имеется предварительный список модулей и процедур, составляющих систему. Затем в течение нескольких недель выполняются задания по проектированию процедур. В результате выполнения каждого задания проектируются и обсуждаются процедуры для одного модуля, после чего выдается задание на проектирование следующего модуля. Проектирование процедур в течение первого семестра выполняется студентами вручную, никаких автоматизированных систем редактирования текстов при этом не используется.

Для тех студентов, которые не будут изучать лабораторный курс на втором семестре, предусмотрено завершение курса на первом семестре. Они изучают разделы, связанные с конвертированием написанных на языке проектирования процедур в

один из высокоуровневых языков программирования и объединением системы, но при этом не выполняют соответствующих упражнений.

Лабораторная часть курса выполняется в течение второго семестра, во время которого студенты вводят текст на языке проектирования в вычислительную систему, осуществляют его конверсию в текст на языке PL/M, выполняют компиляцию, объединение и отладку программного обеспечения. После завершения лабораторного курса листинги программ вместе с функциональной спецификацией, схемой модульной структуры, планом объединения и пр. составляют полный набор документации по системе, используемой в качестве курсового проекта. Студенты сохраняют эту документацию в качестве примера разработки проекта для будущих ссылок.

Средства автоматизации проектирования, предлагаемые студентам университета, намного проще тех средств, что описаны в гл. 7. Подготовка текста на языке проектирования и его конверсия в текст на языке PL/M выполняются вручную с использованием перфокарт. Затем перфокарты вводятся в вычислительную систему типа MTS, при этом для коррекции ошибок используется простой редактор строк.

MTS-система представляет собой большую универсальную ЭВМ с разделением времени, которой пользуются студенты Мичиганского университета и его филиалов. Кросс-компилятор с языка PL/M модифицирован таким образом, что эмулирует одну из версий PL/M, поставляемую фирмой Intel. Кросс-компилятор используется для компиляции модулей на языке PL/M и генерации их объектных модулей в кодах процессора Intel 8085. Для ввода этих модулей в вычислительную систему Digital Group SYSTEM III используется обычный *загрузчик*. Вычислительная система Digital Group SYSTEM III содержит микрокомпьютер Zilog Z-80, который может выполнять набор команд микрокомпьютера Intel 8085, так как этот набор является подмножеством набора команд Z-80. Программы затем объединяются и отлаживаются на микрокомпьютере Z-80, для чего может быть использована любая совместимая с Intel 8085 система отладки. Поскольку имеется только одна микрокомпьютерная система типа Digital Group, все студенты делятся на небольшие группы, и для каждой из групп в течение недели выделяется время. Студенты, каждый из которых вначале описывает на языке проектирования, кодирует и компилирует свою часть проекта по отдельности, затем работают вместе, как одна бригада, помогая друг другу загружать, объединять и отлаживать программное обеспечение. Имеется руководство по загрузке программ, полученных с помощью MTS-системы, и по использованию микрокомпьютера Digital Group.

Если доступны более тонкие средства, такие, как редактор текста на экране дисплея, возможно более эффективное взаимодействие студента с ЭВМ. Гл. 7 содержит краткое описание различных редакторов и других средств, являясь предпосылкой к дальнейшему обсуждению использования этих средств в лабораторном курсе.

Книга может быть использована в качестве учебника для других университетских или кратких курсов Гл. 1—4 вместе с дополнительным материалом из гл. 5 и приложением Г могут быть использованы в качестве общего курса по *системному проектированию программного обеспечения.* По материалам книги были подготовлены однодневные семинары, трехдневные краткие курсы, двухнедельные практические курсы и односеместровые университетские курсы. Семинары и краткие курсы могут быть также использованы для повышения квалификации персонала внутри организации. Последнее было использовано авторами, которые объединили форму краткого курса с принципами *групповой разработки*, чтобы создать *группы проектировщиков*, которые вместе учатся и затем вместе работают над реальным проектом.

Глава 1

Введение

Несмотря на то что первые электронные цифровые вычислительные машины появились всего четверть века назад, ЭВМ приобретают все большее и большее значение в повседневной жизни. В настоящее время благодаря широкому распространению дешевых микрокомпьютеров можно ожидать, что в недалеком будущем их влияние еще более усилится. Основной целью настоящей книги является описание методов системного проектирования электронных систем, включающих микрокомпьютеры в качестве своих компонентов. Поэтому уместно будет начать с исследования вопроса: *что понимается под системой?*

1.1. Понятие системы

Система состоит из набора компонентов, выполняющих определенные функции по отношению к внешнему окружению системы. Поэтому, чтобы иметь возможность воспринимать информацию извне и передавать ее во внешнее окружение, система должна быть связана с внешним окружением, т. е. должна иметь входы и выходы. Общее представление системы и ее внешнего окружения показано на рис. 1.1. ВХОД 1, ..., ВХОД n являются входами в систему из внешнего окружения, а ВЫХОД 1, ..., ВЫХОД m — выходами из системы во внешнее окружение.

Из повседневной практики известны примеры систем как естественного происхождения, так и искусственных. Примером естественной системы является существование человека в общественной среде. В данном случае *система* «человек» получает входную информацию через органы чувств, т. е. органы зрения, слуха, осязания, обоняния и вкуса. Способность системы разговаривать, писать, двигаться обеспечивает выход информации. Функции человека меняются в очень широком диапазоне и зависят от конкретной личности. Преподаватель, например, взаимодействует со студентами, передавая им знания и проверяя затем, как эти знания усваиваются.

Примером искусственной системы служит станок для обработки заготовок на предприятии по переработке сырья. Ввод информации в станок осуществляется с помощью штурвалов или рычагов, приводимых в действие оператором. Выходом си-

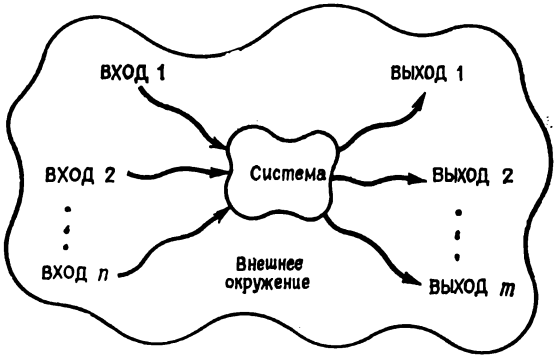


Рис. 1.1. Общее представление системы.

стемы является движение режущего инструмента. Функция станка заключается в изменении размеров и формы сырьевой заготовки при помощи режущего инструмента, движение которого задается входной информацией, введенной оператором. Полученная заготовка может быть затем использована для дальнейшей обработки в процессе производства.

1.2. Электронные системы

Многие искусственные системы в качестве составных частей используют электронные компоненты. Примером системы такого типа служит электронный усилитель. Здесь входом является электрический сигнал в форме волны, а выходом — увеличенная копия входного сигнала. Типичными составными частями электронного усилителя являются резисторы, конденсаторы, транзисторы, диоды и интегральные схемы.

Существуют также более сложные электронные системы, которые используются в задачах комплексных измерений. Многие годы эти системы конструировались из стандартных электронных частей, смонтированных в жесткую конфигурацию, подобно описанному выше усилителю. Однако в последние годы резко уменьшились размеры и стоимость таких вычислительных устройств, как *микрокомпьютеры*, что дало возможность встраивать их в электронные системы в качестве *компонентов*. При этом пользователи системы могут не подозревать о существовании микрокомпьютеров так же, как они не подозревают о существовании конкретных конденсаторов, резисторов или транзисторов. Рассмотрим *микрокомпьютерные компоненты* более подробно.

1.3. Микрокомпьютерные компоненты

Конструирование электронных схем из конденсаторов и резисторов состоит в определении параметров этих компонентов,

измеряемых фарадами и омами, а также в уточнении ограничений по напряжению и мощности. Их функциональные и эксплуатационные характеристики заранее известны. В случае использования транзистора или интегральной схемы функциональные характеристики компонентов не могут быть определены значениями одного или двух параметров. Характеристики таких устройств описываются с помощью функции преобразования, определяющей связь между входом и выходом устройства. Это позволяет осуществить выбор компонентов в соответствии с требованиями, предъявляемыми к электронной схеме.

Микрокомпьютер в отличие от других электронных компонентов не обладает фиксированным набором функциональных характеристик. Его характеристики определяются во время проектирования системы с помощью процесса, называемого *программированием*. Практически неограниченный диапазон программируемых функциональных возможностей микрокомпьютера придает этому компоненту особое значение.

В данной книге исследуются как непрограммируемые характеристики микрокомпьютера, так и методы проектирования и построения программ, определяющих его функциональные характеристики. Электронные компоненты системы (включая микрокомпьютер), которые называются *аппаратными* (или *техническими*) *средствами*, являются сравнительно жесткими и трудно поддающимися изменениям после того, как параметры этих компонентов выбраны и система построена [1]). В противоположность этому программные компоненты, создаваемые во время проектирования и называемые *программным обеспечением*, относительно легко могут быть изменены даже после того, как проектирование завершено и система сконструирована [2]).

Состоит ли система только из аппаратных компонентов или содержит микрокомпьютер с соответствующим программным обеспечением — ее проектирование должно оцениваться с точки зрения эффективности затрат. Проектирование аппаратуры и программного обеспечения должно проводиться на системной основе с целью минимизации как стоимости проектирования, так и времени, затрачиваемого на разработку. Для лучшего понимания этих положений необходимо исследовать понятие *цикла проектирования системы.*

---

[1]) Термин «hardware», которым в английской и американской литературе по вычислительной технике обозначается понятие «аппаратные средства», дословно означает «твердый продукт», «металлические изделия». — *Прим. перев.*

[2]) Термин «software» («программное обеспечение») дословно означает «мягкий, гибкий продукт». — *Прим. перев.*

1.4. Цикл проектирования системы

Первый шаг цикла проектирования системы включает определение набора *требований пользователей* и построение *функциональной спецификации*, вытекающей из требований пользователей. Требования пользователей определяют, что пользователь хочет от системы и что она должна делать. Хорошие системные спецификации определяют функции, выполняемые системой для пользователя после завершения проектирования, уточняя таким образом, насколько система соответствует требованиям пользователя. Они включают описания форматов как на входе, так и на выходе, а также внешние условия, управляющие действиями системы. Функциональная спецификация и требования пользователей являются критериями оценки функциональных характеристик системы после завершения проектирования.

Следующим шагом является проектирование системы на основе функциональной спецификации. Для системы, содержащей только аппаратные компоненты, это означает выбор конфигурации системы, определение значений параметров составляющих частей и способа взаимосвязи этих частей. Аппаратура конструируется, тестируется и объединяется в единое целое, после чего оцениваются ее эксплуатационные характеристики. На каждом шаге цикла проектирования системы могут потребоваться перепроектирование и модификация системы с целью ее соответствия функциональной спецификации. Следует отметить, что, чем раньше в течение цикла проектирования обнаружена проблема, тем меньше затраты на коррекцию. Цикл проектирования системы аппаратных средств показан на рис. 1.2.

Для системы, содержащей микрокомпьютер, требуется проектирование как аппаратных, так и программных средств. Необходимо определить аппаратную и программную конфигурации, определить, из каких *частей* должна состоять система и как эти части должны быть взаимосвязаны. Проектирование аппаратной части может быть выполнено с использованием стандартной методологии проектирования аппаратуры. Проектирование программного обеспечения лучше всего может быть выполнено с использованием *языка проектирования,* подобного естественному языку. Программное обеспечение строится путем преобразования конструкций языка проектирования в *язык программирования микрокомпьютера.* Оно тестируется и одновременно с аппаратурой объединяется в единое целое, после чего оцениваются эксплуатационные характеристики системы. Цикл проектирования системы, содержащей программное обеспечение и аппаратные средства, показан на рис. 1.3. Две части системы часто разрабатываются параллельно, что на рисунке выглядит в виде отдельных *ветвей.*

Рис. 1.2. Цикл проектирования аппаратной системы.

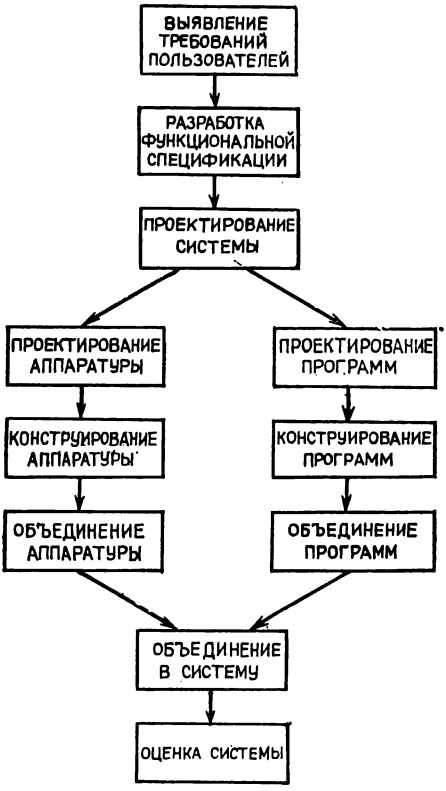


Рис. 1.3. Цикл проектирования системы, содержащей микрокомпьютер.

Благодаря возросшим возможностям микрокомпьютерных компонентов может быть достигнуто более гибкое проектирование с меньшим числом аппаратных компонентов по сравнению с системами, содержащими только аппаратные компоненты. Однако поскольку такая система может очень легко превратиться в чрезмерно сложную и громоздкую, а также с точки зрения эффективности затрат, программное обеспечение должно проектироваться и разрабатываться на системной основе. Одним из основных средств снижения сложности программного обеспечения до приемлемого уровня является использование методологии системного проектирования. Кроме использования языка проектирования системная методология, описываемая в книге, включает использование методов *нисходящего* и *модульного*

проектирования. Поскольку использование языка проектирования занимает центральное место в процессе проектирования системы, необходимо рассмотреть эту концепцию более подробно.

1.5. Язык проектирования

Концепция языка проектирования может быть рассмотрена с помощью простого примера, функциональная спецификация которого иллюстрируется функциями преобразования и блок-схемой, показанными на рис. 1.4. ВХОД 1 и ВХОД 2 являются аналоговыми входами, а ВЫХОД 1 и ВЫХОД 2 — аналоговыми выходами. Система состоит из микрокомпьютера с двумя входами и двумя выходами. Поскольку микрокомпьютер является цифровым устройством, для преобразования входных и выходных сигналов необходимо иметь соответственно аналого-цифровые и цифроаналоговые преобразователи.

Из наличия у системы входов и выходов можно сделать заключение о том, что микрокомпьютер должен иметь возможность *проверять значение* каждого входа, а также *устанавливать* каждый из выходов в определенное *значение.* На уровне языка проектирования для операций *проверки* и *установки* используются простые *конструкции,* смысл которых становится ясным из следующих примеров:

```
ПРОВЕРИТЬ ВХОД 1 И ХРАНИТЬ ЕГО ЗНАЧЕНИЕ
УСТАНОВИТЬ ЗНАЧЕНИЕ ВЫХОД 1 РАВНО 6
```

Необходимо также иметь возможность проверять условия, которым удовлетворяют хранимые значения каждого из входов для установки выходных значений. С этой целью используется *условная* конструкция, которая в общем виде может быть представлена следующим образом:

```
ЕСЛИ условие проверки есть „истина"
  ТО выполнить что-либо
  ИНАЧЕ выполнить что-либо другое
```

Таким образом, для нашего примера описание на языке проектирования имеет значале следующий вид:

```
ПРОВЕРИТЬ ВХОД 1 И ХРАНИТЬ ЕГО ЗНАЧЕНИЕ
ПРОВЕРИТЬ ВХОД 2 И ХРАНИТЬ ЕГО ЗНАЧЕНИЕ
ЕСЛИ ЗНАЧЕНИЕ ВХОД 1 БОЛЬШЕ 4 И МЕНЬШЕ 8
  ТО УСТАНОВИТЬ ЗНАЧЕНИЕ ВЫХОД 1 РАВНО 6
  ИНАЧЕ УСТАНОВИТЬ ЗНАЧЕНИЕ ВЫХОД 1 РАВНО 0
ЕСЛИ ЗНАЧЕНИЕ ВХОД 2 БОЛЬШЕ 2 И МЕНЬШЕ 6
  ТО УСТАНОВИТЬ ЗНАЧЕНИЕ ВЫХОД 2 РАВНО 4
  ИНАЧЕ УСТАНОВИТЬ ЗНАЧЕНИЕ ВЫХОД 2 РАВНО 0
```

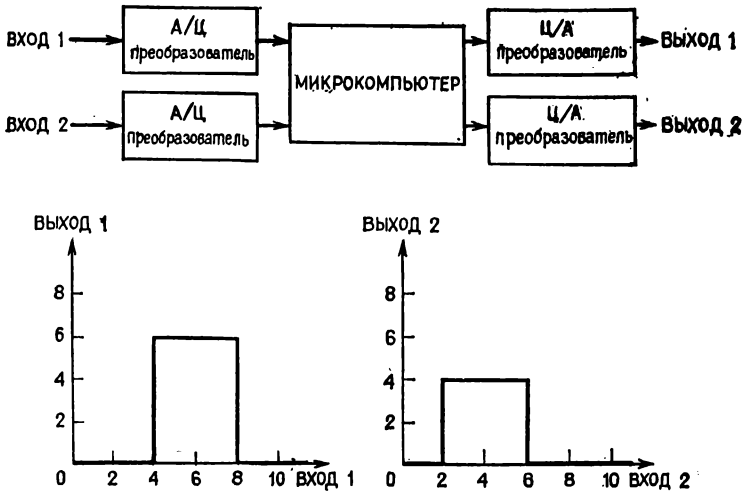


Рис. 1.4. Пример микрокомпьютерной системы.

Когда система функционирует, микрокомпьютер выполняет запрограммированные операции шаг за шагом. Таким образом, если программа в микрокомпьютере соответствует рассмотренному примеру, с функциональной точки зрения ее поведение является правильным. Однако после того, как входы проверены, нет уверенности, что затем при повторной проверке один из них не окажется измененным. Поэтому необходима такая операция, которая позволяла бы выполнять другие операции языка проектирования бесконечное число раз. Для этой цели используется конструкция:

```
ВЫПОЛНЯТЬ НЕПРЕРЫВНО
    .
    .
    .
КОНЕЦ
```

В этой конструкции набор операций, расположенный между ВЫПОЛНЯТЬ НЕПРЕРЫВНО и КОНЕЦ, должен повторяться без конца. С использованием такой конструкции вышеприведенный пример будет выглядеть следующим образом:

```
ВЫПОЛНЯТЬ НЕПРЕРЫВНО
  ПРОВЕРИТЬ ВХОД 1 И ХРАНИТЬ ЕГО ЗНАЧЕНИЕ
  ПРОВЕРИТЬ ВХОД 2 И ХРАНИТЬ ЕГО ЗНАЧЕНИЕ
  ЕСЛИ ЗНАЧЕНИЕ ВХОД 1 БОЛЬШЕ 4 И МЕНЬШЕ 8
    ТО УСТАНОВИТЬ ЗНАЧЕНИЕ ВЫХОД 1 РАВНО 6
    ИНАЧЕ УСТАНОВИТЬ ЗНАЧЕНИЕ ВЫХОД 1 РАВНО 0
  ЕСЛИ ЗНАЧЕНИЕ ВХОД 2 БОЛЬШЕ 2 И МЕНЬШЕ 6
```

```
        ТО УСТАНОВИТЬ ЗНАЧЕНИЕ ВЫХОД 2 РАВНО 4
        ИНАЧЕ УСТАНОВИТЬ ЗНАЧЕНИЕ ВЫХОД 2 РАВНО 0
    КОНЕЦ
```

В этом случае, однажды начавшись, операции проверки значений входов будут повторяться столько времени, сколько система остается в действии.

Из вышеприведенного примера и обсуждения можно сделать следующие выводы:

• Микрокомпьютер является последовательным устройством и в каждый момент времени выполняет только одну операцию.

• Во многих случаях, чтобы изменить функциональное поведение микрокомпьютерной системы, необходимо изменить лишь некоторые операции в описании программы на языке проектирования.

• Микрокомпьютер манипулирует только цифровыми данными. Если в системе имеются аналоговые сигналы, для преобразования входных сигналов в цифровую форму необходим аналого-цифровой преобразователь, а для преобразования выходных сигналов в аналоговую форму — цифроаналоговый преобразователь.

Как уже указывалось, для того чтобы микрокомпьютер мог выполнять операции языка проектирования, необходимо конвертировать описание программы на языке проектирования в программу микрокомпьютера. Разработаны языки программирования для микрокомпьютеров, которые имеют схожие конструкции с конструкциями языка проектирования. Например, некоторые языки программирования содержат конструкцию ЕСЛИ ... ТО ... ИНАЧЕ ..., которая очень похожа на соответствующую конструкцию языка проектирования. Вопросы конвертирования конструкций языка проектирования в программы будут обсуждаться в гл. 5.

1.6. Документация

Одним из наиболее важных факторов добросовестного проектирования систем является хорошая документация. Под хорошей понимается четко организованная, легко читаемая и усваиваемая документация, сжатая, но полная, допускающая внесение изменений. Постепенно продвигаясь в рамках цикла проектирования от требований пользователей и функциональной спецификации к объединению и оценке действующей системы, мы будем показывать, какая информация должна быть включена в документацию на каждом уровне проектирования и построения системы. Чтобы легче было проследить за ходом обсуждения, каждому сегменту документации назначается номер уровня.

1. Требования пользователей и функциональные спецификации
2. Проектная документация системы
3. Программная документация
4. План объединения
5. Техническая документация
6. План отладки аппаратных средств

Рис. 1.5. Организация документации.

На рис. 1.5 показана организация документации для полного цикла проектирования.

Первый уровень документации содержит описания требований пользователей и функциональные спецификации. Второй и третий уровни содержат документацию системы на языке проектирования и программную документацию. Эти два уровня упоминаются вместе, поскольку, как далее будет видно, они тесно соотносятся друг с другом.

Чтобы проследить за каждым шагом процесса объединения, необходимо составить *план объединения.* Это будет четвертым уровнем документации. И наконец, пятый и шестой уровни включают документацию по аппаратным средствам и *план отладки аппаратных средств.* Эти шесть уровней составляют полный набор системной документации, которая может сопровождаться и храниться в соответствии с принятыми нормами.

1.7. Содержание книги

Гл. 2 посвящена способам выявления набора требований пользователей и построению на основе этих требований функциональной спецификации. Исследуются также возможности выбора различных вариантов, встречающихся на фазах планирования и определения спецификаций для каждой системы. В книге используется несколько примеров автоматизированных систем. Эти примеры призваны обеспечить целостность изложения, необходимую для более тщательного понимания всех аспектов проектирования. Кроме того, эти примеры применялись в качестве обучающих и были признаны поучительными и содержательными по смыслу. В гл. 2 исследуются также вопросы учета *человеческих факторов* при проектировании. Поскольку большинство микрокомпьютерных систем каким-либо способом взаимодействуют с людьми, важно учитывать это взаимодействие во время проектирования системы.

В гл. 3 исследуются вопросы проектирования систем на основе разработанной в гл. 2 функциональной спецификации для системы охранной сигнализации. Проектирование систем заключается в их разбиении на отдельные *модули,* которые могут быть реализованы как в виде аппаратных средств, так и в виде программ. Вводится концепция нисходящего проектирования.

В этой же главе рассматриваются вопросы выбора реализации функций с помощью либо аппаратных, либо программных средств.

Гл. 4 начинается с обсуждения вопросов модульной декомпозиции, начиная с проектирования систем и далее, включая описание программного обеспечения на языке проектирования. Проектирование программного обеспечения состоит прежде всего в разработке варианта разбиения программного обеспечения на модули на основе языка проектирования. В этой же главе описываются методы верификации правильности описания программного обеспечения на языке проектирования до того, как оно будет конвертировано в программы на языке программирования микрокомпьютера.

В гл. 5 будет показано, как описание проекта на языке проектирования конвертируется в программы на языке программирования высокого уровня, которые могут быть выполнены микрокомпьютером. Благодаря тому что язык проектирования и некоторые языки программирования высокого уровня подобны друг другу, программирование микрокомпьютера является относительно простой задачей.

В гл. 6 описываются архитектура микрокомпьютера, язык ассемблера и машинный язык, с тем чтобы читатель получил представление о связи между аппаратными средствами и программным обеспечением микрокомпьютера. В гл. 6 также вводятся основные концепции программирования на языке ассемблера с целью ознакомления с ними читателя, у которого может когда-либо возникнуть необходимость их использования.

В гл. 7 обсуждаются вопросы объединения программных модулей в систему программного обеспечения, которая в свою очередь может быть объединена с аппаратными средствами.

В гл. 8 описывается проектирование аппаратных средств системы, содержащей микрокомпьютер, а в гл. 9 показано, как аппаратные модули конструируются и объединяются в действующую систему, готовую к объединению с программным обеспечением. Здесь же описано, как проверить правильность работы аппаратных средств, прежде чем программное обеспечение будет объединено с аппаратными средствами.

В гл. 10 будет показано, как программное обеспечение объединяется с аппаратными средствами, а также как проверить правильность функционирования завершенной системы. Как уже подчеркивалось, проектирование программного обеспечения является сложным процессом, и желательно, чтобы ошибки проектирования и программирования были выявлены до завершения проектирования системы. Поэтому в гл. 4, 7, 9 и 10 описываются средства и методы, которые могут облегчить разра-

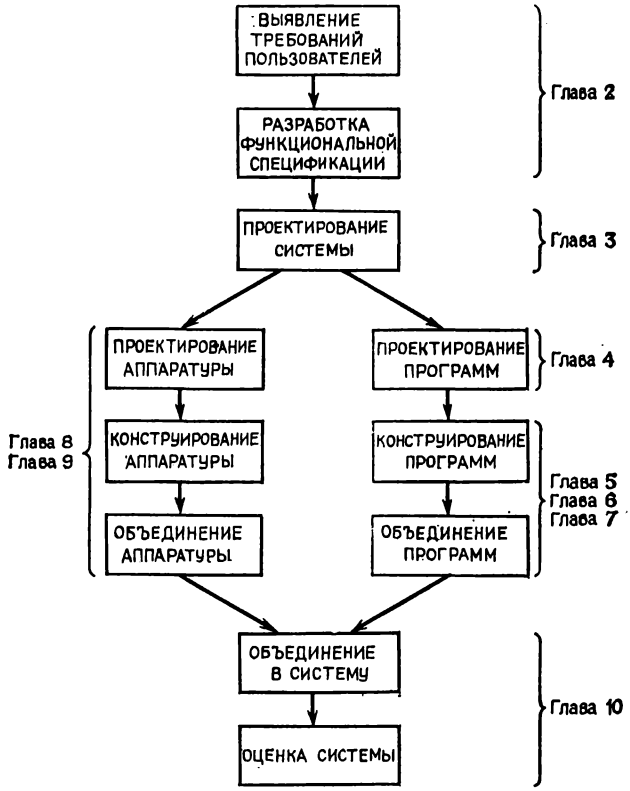


Рис. 1.6. Организация книги и ее связь с циклом проектирования системы.

ботку, отладку и объединение частей системы, обеспечить выявление и устранение ошибок и гарантировать соответствие завершенной системы требованиям пользователей.

Соответствие гл. 2—10 циклу проектирования системы показано на рис. 1.6. Этот рисунок является *маршрутной схемой* для читателя, который должен обращаться к нему по мере своего продвижения по страницам книги.

В последней 11 главе описывается несколько других систем с целью иллюстрации широких возможностей применения микрокомпьютеров.

Целью данной книги является описание эффективных методов проектирования систем, содержащих микрокомпьютеры, наиболее полно и с наименьшими затратами удовлетворяющих требованиям пользователей и соответствующей функциональной спецификации.

1.8. Упражнения

Для обозначения неразделимого набора компонентов, имеющих хорошо определенные функциональные связи, инженеры используют понятие черного ящика.

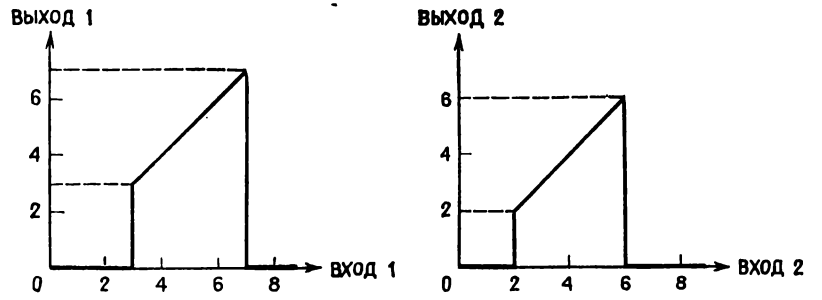


Рис. 1.7.

1.1. Какая связь существует между черным ящиком и системой?

1.2. Приведите известные вам примеры черных ящиков.

1.3. Можно ли считать микрокомпьютер черным ящиком? Объясните почему.

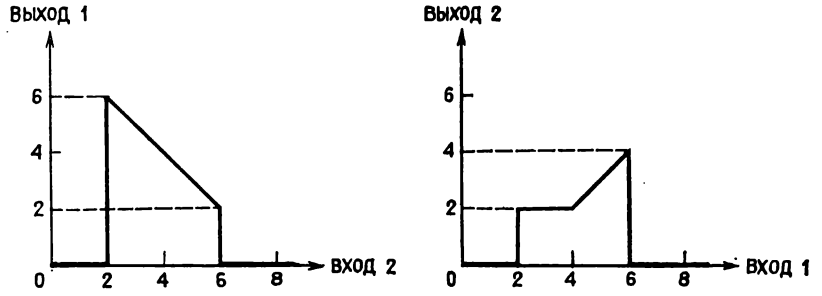


Рис. 1.8.

1.4. Может ли черный ящик содержать микрокомпьютер? Объясните почему.

1.5. Выше было упомянуто несколько способов уменьшения сложности проектирования систем, основанных на использовании микрокомпьютера. Известны ли вам другие способы? Опишите их.

1.6. Каким образом уменьшается сложность проектирования с помощью способов, предложенных вами в упр. 1.5?

1.7. Используя язык проектирования, опишите микрокомпьютерную систему, удовлетворяющую функциональной спецификации, показанной на рис. 1.7.

1.8. Используя язык проектирования, опишите микрокомпьютерную систему, удовлетворяющую функциональной спецификации, показанной на рис. 1.8.

1.9. Почему разработка и сопровождение хорошей документации являются важным вопросом при проектировании систем, основанных на использовании микрокомпьютера?

Глава 2

Требования пользователей и функциональная спецификация

Теперь мы готовы приступить к более детальному описанию цикла проектирования системы. В этой главе мы обсудим, как выявить требования пользователей, и покажем, как на основе этих требований определить функциональную спецификацию системы. Требования пользователей определяют, что пользователь хочет от системы, а функциональная спецификация фиксирует, что система должна делать и как она взаимодействует с окружением. Как только функциональная спецификация определена, она используется вместе с требованиями пользователей в качестве основы для проектирования, реализации и развития системы. По этой причине важно, чтобы как требования пользователей, так и функциональная спецификация были не только полными и точными, но также четкими и легко усваиваемыми.

2.1. Требования пользователей

Требования пользователей определяют, что хочет или в чем нуждается пользователь или потребитель. Требования пользователя могут быть получены во время встреч с пользователем или покупателем с целью выявления его нужд и определения того, что пользователь хочет от системы. Другой подход используется при планировании ассортимента изделий, когда требования пользователей могут быть определены путем изучения рынка сбыта на основе спроса покупателей. Действительно, исследование конкуренции на рынке сбыта помогает определить, что будет и что не будет пользоваться спросом на рынке.

Представим, что компания вследствие планируемого расширения ассортимента выпускаемых изделий намерена разработать автоматизированную систему охранной сигнализации. Попытаемся определить набор требований пользователей к этой системе. На первом шаге необходимо получить информацию, касающуюся того, что система должна делать. Чтобы информация по этому вопросу была наиболее полной, следует тесно взаимодействовать с отделом сбыта, а также с представителями некоторых потребителей. Вопросы, которые задают в первый момент, должны быть связаны только с тем, что должна де-

лать система охранной сигнализации. В частности, необходимо получить ответ на следующие вопросы:

- Какие типы нарушений необходимо обнаруживать?
- Какие действия требуются при обнаружении нарушителя?
- Какие другие особые действия необходимо предусмотреть?

Из ответов на эти вопросы можно сделать вывод относительно того, что будут из себя представлять требования пользователей. Для системы охранной сигнализации набор соответствующих требований пользователей приведен ниже. На основе этих требований можно определить функциональную спецификацию. Система охранной сигнализации должна выполнять следующие действия:

- Обнаруживать, когда открывается дверь или окно.
- Обнаруживать, если кто-то движется внутри охраняемой зоны.
- Иметь возможность предупредить нарушителя и вызвать помощь.
- Обеспечивать возможность восстановления в случае забывчивости оператора.
- Быть несложной в управлении.
- Минимизировать число ложных тревог.

2.2. Функциональная спецификация

Функциональная спецификация должна определять, какие функции должны выполняться для удовлетворения требований пользователей и обеспечения интерфейса между системой и окружением. Таким образом, функциональная спецификация включает два основных компонента:

1. Список функций, выполняемых системой.
2. Описание интерфейса между системой и пользователем.

Так как система проектируется на основе информации, содержащейся как в требованиях пользователей, так и в функциональной спецификации, важно, чтобы функции, которые должны отображать требуемое поведение системы, были описаны достаточно подробно. По отношению к требованиям пользователей системы охранной сигнализации функциональная спецификация должна давать ответы на следующие вопросы:

- Какие средства необходимо предусмотреть для обнаружения несанкционированного открытия двери или окна?
- Какие средства необходимо предусмотреть для обнаружения движения?
- Какие средства необходимо предусмотреть для предупреждения нарушителя и вызова помощи?
- Какие средства необходимо предусмотреть для восстановления системы в случае забывчивости оператора?

А. ВХОДЫ
1. Контактные детекторы.
2. Детектор движения.
3. Переключатель.

Б. ВЫХОДЫ
1. Визуальный сигнал.
2. Звуковой сигнал.

В. ФУНКЦИИ
1. Система включается и восстанавливается с помощью переключателя.
2. Визуальный сигнал включается либо
   (а) при размыкании контактного детектора, либо
   (б) в случае продолжительного возбуждения детектора движения в течение не менее пяти секунд
3. Звуковой сигнал включается через шестьдесят секунд после включения визуального сигнала, если за этот промежуток времени система не восстановлена с помощью переключателя.

Рис. 2.1. Функциональная спецификация системы охранной сигнализации.

• Какие средства необходимо предусмотреть для управления системой?

• Какие средства необходимо предусмотреть для предотвращения ложных тревог?

Ответив на эти вопросы, можно приступить к составлению функциональной спецификации для системы охранной сигнализации. В рассматриваемой системе:

• Для обнаружения несанкционированного открытия двери или окна должны использоваться контактные детекторы.

• Для обнаружения движения должен использоваться ультразвуковой детектор движения. С целью предупреждения ложной тревоги движение должно контролироваться в течение не менее пяти секунд, после чего считается, что обнаружен нарушитель.

• Оператор должен быть предупрежден о том, что он обязан восстановить систему. Предупреждение осуществляется с помощью визуального сигнала, который должен включаться за шестьдесят секунд до того, как будет включен сигнал звуковой тревоги. Если система не восстановлена в течение шестидесяти секунд, для предупреждения нарушителя и вызова помощи включается сигнал звуковой тревоги.

• Для управления системой и ее восстановления должен использоваться кнопочный переключатель.

Эти ответы содержат информацию, необходимую для определения функциональной спецификации. Представим эту спецификацию в форме, удобной для последующих ссылок и использования на других этапах цикла проектирования. Если распределить информацию по категориям ВХОДЫ, ВЫХОДЫ и ФУНКЦИИ, можно представить функциональную спецификацию системы охранной сигнализации в виде документа, показанного на рис. 2.1. Во многих случаях функциональная специ-

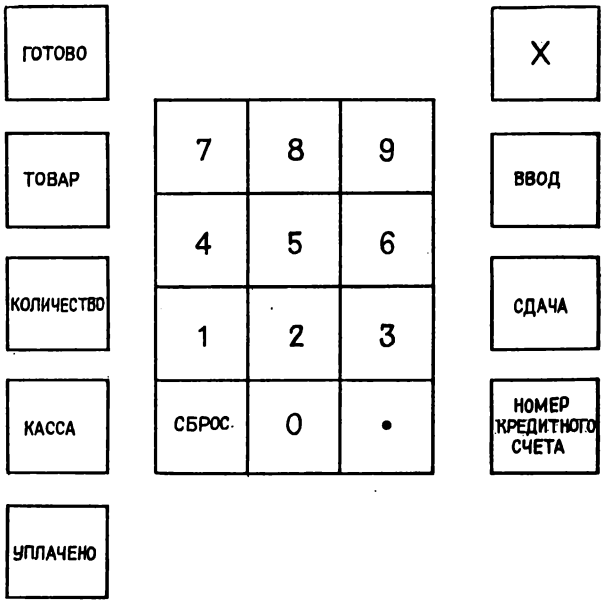


Рис. 2.2. Схема расположения клавиш простого терминала розничной торговли.

фикация включает схемы частей системы. Например, для системы, содержащей кнопочную панель, в функциональную спецификацию может быть включена схема расположения кнопок. Пример такой схемы для простой системы расчета за покупки (автоматизированного кассового аппарата) показан на рис. 2.2.

Теперь, после того как мы завершили составление функциональной спецификации для системы охранной сигнализации, рассмотрим несколько замечаний относительно функциональных спецификаций вообще. Конкретная система, для которой была определена функциональная спецификация, является весьма простой, и поэтому ее список оказался невелик. В более сложных системах функциональная спецификация даже с использованием высокоуровневого представления гораздо длиннее. В таких случаях могут оказаться более приемлемыми декомпозиция системы на несколько *подсистем* меньшего размера и определение функциональной спецификации для каждой из них. В некоторых случаях функциональная спецификация может содержать вместо описания функции описание реализации. Такое описание допустимо, когда необходимо показать, как действует система. В примере с системой охранной сигнализации в функциональной спецификации необходимо было бы

указать, что контактные детекторы соединены последовательно и что в исходном положении они замкнуты. Для системы охранной сигнализации это является важной характеристикой входов, и поэтому будет разумным добавить ее к функциональной спецификации, хотя мы решили не делать этого. Информация, касающаяся реализации, обычно входит в состав *проектной спецификации.* Рассмотрим кратко основные различия между функциональной и проектной спецификациями.

2.3. Функциональная и проектная спецификации

Функциональная спецификация обычно описывает, *что* система делает. В ней не указываются конкретные детали, описывающие, *как* должна быть реализована система. В течение цикла проектирования системы, при продвижении от более общих деталей к более конкретным, часто возникает стремление включить как можно больше частных деталей на более ранних стадиях проектирования. Этого следует избегать.

Чтобы проиллюстрировать это на примере, рассмотрим систему управления запасами. Функциональная спецификация для системы управления запасами должна включать информацию о том, сколько различных типов товаров должна хранить система, нужно ли следить за максимальным и минимальным количеством каждого из типов товаров, необходимо ли использование пароля на входе для разрешения на получение информации. Последняя функция в случае ее реализации позволит запретить доступ к информации и внесение изменений в нее посторонним лицам. Проектная спецификация в этом случае содержит конкретную информацию, касающуюся реализации системы управления запасами, включая организацию данных, организацию доступа к данным и их хранения.

Мы рассмотрели этапы разработки набора требований пользователей и функциональной спецификации. Желательно провести несколько итераций, включающих обсуждение требований пользователей и функциональной спецификации с возможными потребителями или с представителями отдела сбыта, и соответствующую коррекцию требований и спецификации. *Необходимо помнить, что законченная система должна выполнять только то, что ожидает от нее потребитель, и не должна делать ничего такого, чего потребитель не хочет от нее.*

Поскольку взаимодействие между пользователем и системой является очень важным вопросом, рассмотрим далее аспекты проектирования микрокомпьютерных систем, связанные с человеческими факторами.

2.4. Человеческие факторы

В системе охранной сигнализации взаимодействие между системой и пользователем осуществляется преимущественно с помощью кнопочного переключателя. В общем случае, однако, могут существовать другие способы взаимодействия между оператором и системой. Сюда включаются:

- *Тактильные* (*контактные*) *входы.* Пользователь может нажимать кнопки, поворачивать ручки, манипулировать рычагами и т. д.
- *Визуальные выходы.* Визуальный сигнал системы охранной сигнализации является примером визуального выхода. Другие системы используют для вывода информации цифровые индикаторы, телеэкраны, печатающие устройства и т. д.
- *Косвенные входы.* В системе охранной сигнализации косвенный ввод информации осуществляется нарушителем через контактные детекторы и детектор движения. В других системах косвенный ввод может осуществляться оператором. Например, если оператор системы расчета за покупки, в которой имеется устройство автоматического взвешивания, вводит идентификатор товара, то при этом в систему непосредственно вводится цена товара. Устройства чтения штриховых кодов и магнитных карт, которые считывают информацию о товарах непосредственно в систему расчета за покупки, являются другими примерами широко используемых косвенных входов.
- *Звуковые выходы.* Сигнал звуковой тревоги в системе охранной сигнализации является примером звукового выхода. В системе расчета за покупки часто используется зуммер для оповещения оператора о завершении некоторой последовательности операций ввода или об ошибке.

Учет человеческих факторов при проектировании должен приводить к простоте системы и легкости ее использования. Эти цели достигаются посредством проектирования надлежащего системного интерфейса, обеспечивающего экономный обмен информацией между оператором и машиной. Это позволяет уменьшить затраты на обучение оператора и не требует больших усилий при взаимодействии с системой. Система расчета за покупки является примером такой системы, в которой скорость и точность являются существенными факторами. Ошибка при наборе цены товара может привести к тому, что торговое предприятие либо потерпит убыток, либо непреднамеренно обсчитает покупателя. Во многих магазинах самообслуживания покупатели могут отобрать продукты гораздо быстрее, чем кассиры способны их обслужить. Для решения проблемы длинных очередей возможны две альтернативы: либо увеличить количество кассиров, либо повысить производительность их труда. Хо-

рошо спроектированная система расчета за покупки может помочь в этом. Рассмотрим конкретные примеры.

В системе расчета за покупки, проектирующейся для магазина полуфабрикатов, можно предусмотреть клавишу для каждого товара и предоставить возможность управляющему *программировать* каждую клавишу в соответствии с текущими ценами на товары. Это позволяет кассиру *выбить* бифштекс путем нажатия *клавиши бифштекса* вместо нажатия серии цифровых клавиш, следующих за клавишей ВВОД, как это предусмотрено в стандартной системе, клавиатура которой показана на рис. 2.2. В системе расчета за покупки магазина полуфабрикатов стоимость бифштекса всегда одна и та же, поэтому ошибки практически исключены. Передача информации от кассира к системе намного экономичнее, поскольку несколько тактов передачи заменяются одним тактом. Улучшается точность, так как кассир работает с *функциональными клавишами* вместо цифровых клавиш.

В стандартных системах расчета с покупателем, предназначенных для общего розничного использования, может быть спроектирована система, *побуждающая* кассира выполнить некоторую последовательность операций, необходимых для правильного расчета. Как только системе потребуется какая-либо информация, загорается соответствующая сигнальная лампочка, указывающая, какого рода информация требуется. Например, если требуется номер кредитного счета, загорается лампочка *номер кредитного счета.* Только после того, как номер набран кассиром и проверено его соответствие номеру кредитного счета, хранящемуся в системе, лампочка гаснет и кассир может продолжать выполнять следующие операции.

Микрокомпьютеры применяются также во многих системах, ранее включавших только аппаратные компоненты. Примером такой системы является осциллограф с встроенным микрокомпьютером. Наличие микрокомпьютера в осциллографе позволяет отображать на экране ЭЛТ цифровые результаты измерений вместе с аналоговым изображением. Кроме того, это позволяет точнее измерить некоторые параметры сигналов на экране ЭЛТ, которые трудно или невозможно измерить с помощью аналогового осциллографа. Человеческие факторы в вопросах проектирования микрокомпьютерных систем проявляются также в том, что некоторые характеристики существующих систем должны быть учтены при проектировании новой системы. Например, инженер, который использует осциллограф, привык вращать ручки для установки масштаба времени по горизонтали и вертикального усиления. Поэтому, даже если эти функции находятся теперь под управлением микрокомпьютера, проектировщик должен оставить ручки, чтобы пользователю не

пришлось переучиваться для того, чтобы выполнять хорошо знакомые ему функции, особенно если эти ручки хорошо спроектированы с точки зрения человеческих факторов. Предоставим пользователю возможность научиться использовать преимущества, которые дают ему новые функции, связанные с получением цифровой информации, не переучивая его использовать старые методы получения аналоговой информацией.

Этим завершается наш обзор человеческих факторов при проектировании микрокомпьютерных систем.

2.5. Первый уровень документации

Как уже указывалось в гл. 1, требования пользователей и функциональная спецификация составляют первый уровень документации системы. Этот уровень содержит информацию, необходимую для оценки эффективности функционирования системы, и поэтому является важной частью системной документации.

В некоторых системах требуется разработка *руководства пользователя* и *руководства оператора*, в которых описывается работа системы для потребителя. По традиции руководство пользователя готовится под конец цикла проектирования и часто воспринимается как *неизбежное зло*. Однако часто бывает возможно, и это лучший путь, готовить функциональную спецификацию в форме руководства пользователя. Даже если потребуется пересмотр проекта, нет необходимости разделять документы на функциональную спецификацию и руководство пользователя. Действительно, можно извлечь большую пользу из того, что до начала проектирования пользователь или заказчик будут ознакомлены с руководством пользователя. Поэтому для подобных систем руководство пользователя дублирует функциональную спецификацию и является частью первого уровня документации.

2.6. Упражнения

При работе с каждым упражнением необходимо рассмотреть следующие вопросы:

1. Как оператор вводит информацию в систему?
2. Как обеспечивается в системе обратная связь с оператором?
3. Какие внешние функции предусмотрены в системе?

Постарайтесь, чтобы функциональная спецификация, разработанная вами, была ориентирована на то, чтобы система была удобной, простой в управлении, эффективной по затратам и имела бы такие характеристики, которые привлекали бы раз-

личных потребителей. Необходимо иметь в виду, что микрокомпьютерные системы могут выполняться по индивидуальным заказам. Это означает, что из подобных наборов аппаратных средств могут быть построены различные модели, выполняющие различные функции для разных покупателей.

2.1. Представьте, что вы собираетесь построить телевизионный приемник, содержащий микрокомпьютер. Перечислите набор возможных функций, которые могут быть выполнены микрокомпьютером.

2.2. Разработайте набор требований пользователя для телевизионного приемника с встроенным микрокомпьютером.

2.3. Разработайте функциональную спецификацию для телевизионного приемника, вытекающую из требований пользователя, определенных в упр. 2.2.

2.4. Какие человеческие факторы должны приниматься во внимание при проектировании автоматизированного устройства управления уличным светофором? (Не забывайте о том, каково основное назначение устройства управления уличным светофором!)

2.5. Разработайте набор требований пользователя для автоматизированного устройства управления уличным светофором.

2.6. Разработайте функциональную спецификацию для устройства управления уличным светофором, вытекающую из требований пользователя, определенных в упр. 2.5.

2.7. Какие вопросы необходимо задать покупателю электронного спортивного табло, чтобы составить соответствующий набор требований пользователя?

2.8. Разработайте набор требований пользователя для электронного спортивного табло в соответствии с ответами на вопросы из упр. 2.7.

2.9. Разработайте функциональную спецификацию для электронного спортивного табло, вытекающую из требований пользователя, определенных в упр. 2.8.

2.10. Какие вопросы необходимо задать для получения соответствующего набора требований пользователя для автоматизированной системы расчета за покупки (автоматизированного кассового аппарата)?

2.11. Используя ответы на вопросы из упр. 2.10, разработайте набор требований пользователя для автоматизированной системы расчета за покупки.

2.12. Разработайте функциональную спецификацию для автоматизированной системы расчета за покупки, вытекающую из требований пользователя, определенных в упр. 2.11.

Глава 3

Проектирование системы

Выше было показано, как определить набор требований пользователей и построить функциональную спецификацию системы. Мы уже знаем, что система должна делать и как она взаимодействует с внешним окружением. В этой главе будет показано, как на основе функциональной спецификации построить набор *модулей,* составляющих первый уровень проектирования систем. Как только система расчленена на модули, надо отделить аппаратные модули от программных. Аппаратные модули проектируются и реализуются с использованием стандартных или изготовленных по заказу интегральных схем.

Программные модули разбиваются на множество *процедур,* каждая из которых соответствует отдельной функции. Проектирование программного обеспечения включает описание каждой процедуры на языке проектирования, введенном в гл. 1. Метод разработки программного обеспечения, начиная с функциональной спецификации, включающий разбиение системы на модули и разбиение модулей на процедуры, известен как метод *нисходящего проектирования.*

В этой главе будет рассмотрен процесс проектирования системы. Процесс проектирования программного обеспечения будет рассмотрен в гл. 4.

3.1. Предварительное проектирование системы и выбор соотношения между аппаратными и программными средствами

Прежде чем начинать детальное проектирование программных и аппаратных средств, необходимо определить, какие функции лучше выполняются с помощью программного обеспечения микрокомпьютера, а какие — с помощью аппаратных средств. Все функции предварительно должны быть распределены между программными и аппаратными средствами. Желательно, чтобы с этого момента в структуру проекта не вносились никакие изменения, хотя на практике это редко удается. Во время детального проектирования аппаратных и программных средств часто становится очевидным, что некоторые

аппаратные функции могут быть лучше выполнены с помощью программных средств и наоборот. Таким образом, во время последующих стадий процесса проектирования может иметь место модификация предварительного проектного решения.

Общая схема модульной структуры аппаратных средств микрокомпьютерной системы показана на рис. 3.1. Система разбита на модули, которые соответствуют функциям системы ВХОД,

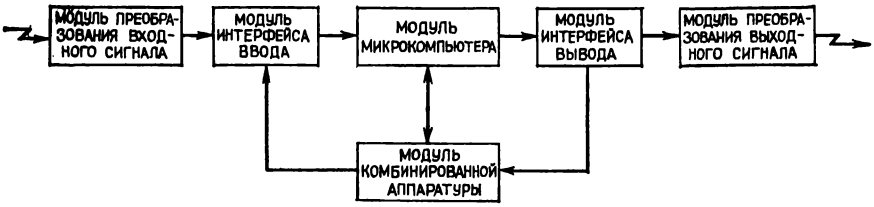


Рис. 3.1. Общая модульная структура аппаратных средств микрокомпьютерной системы.

ВЫХОД, ПРЕОБРАЗОВАНИЕ СИГНАЛА, МИКРОКОМПЬЮТЕР и КОМБИНИРОВАННАЯ АППАРАТУРА. Модули ПРЕОБРАЗОВАНИЯ ВХОДНОГО СИГНАЛА и ПРЕОБРАЗОВАНИЯ ВЫХОДНОГО СИГНАЛА содержат компоненты, необходимые для обмена входными и выходными сигналами с внешней средой. Примерами таких компонентов являются аналого-цифровые и цифроаналоговые преобразователи, позволяющие обрабатывать аналоговые сигналы с помощью микрокомпьютера. Модули ИНТЕРФЕЙСА ВВОДА, МИКРОКОМПЬЮТЕРА и ИНТЕРФЕЙСА ВЫВОДА содержат микрокомпьютер и его компоненты, а также интерфейсные компоненты, необходимые для связи микрокомпьютера с другими модулями системы. Модуль КОМБИНИРОВАННОЙ АППАРАТУРЫ содержит компоненты, необходимые для реализации остальных функций системы. Это как раз те функции, которые могут быть реализованы с помощью как аппаратных, так и программных средств.

В качестве примера выбора одного из возможных решений рассмотрим снова систему охранной сигнализации. Чтобы отличить действительное движение от ложных сигналов, а также для определения момента включения звукового сигнала, в системе необходимо предусмотреть функцию временной задержки. Эта функция может быть реализована одним из следующих способов: либо чисто программным способом — с использованием программы, которая просто считает до пяти или до шестидесяти секунд, либо аппаратным, используя в качестве счетчика времени конфигурацию логических элементов на интеграль-

ных схемах в модуле КОМБИНИРОВАННОЙ АППАРАТУРЫ, либо в комбинации аппаратных и программных средств, используя *интеллектуальный* программируемый таймер на интегральной схеме, содержащейся в модуле КОМБИНИРОВАННОЙ АППАРАТУРЫ. От выбора решения зависит, как проектировать программные и аппаратные средства для функции задержки времени. После того как мы обсудим построение модульной структуры программного обеспечения, мы вернемся к этому вопросу, чтобы проверить правильность выбора. Нисходящее проектирование применимо как для аппаратных, так и для программных средств. Оно позволяет наглядно определить, где взаимодействуют аппаратные и программные средства. Поэтому далее мы обсудим принципы нисходящего проектирования, которые приведут нас к построению *функционально-модульной структуры* программных средств.

3.2. Нисходящее проектирование

Проектирование системы может быть расчленено на несколько функциональных уровней. Обычно высший функциональный уровень проектирования является наиболее общим, а низший — наиболее детализированным. Высший уровень проектирования для аппаратных средств состоит из блочных диаграмм, обозначающих довольно приближенное разбиение. Это может быть уровень подсистем в случае больших систем или уровень модулей для не очень больших систем. Последующие уровни проектирования аппаратуры содержат все более детальное разбиение. Разбиение продолжается до тех пор, пока не будет достигнут уровень таблиц соединений или монтажных схем. Например, следующий уровень модулей ПРЕОБРАЗОВАНИЯ СИГНАЛА может включать аналого-цифровые и цифроаналоговые преобразователи, а также другие устройства преобразования сигналов, необходимые для реализации этого уровня.

Высший уровень проектной документации программного обеспечения состоит из модульной диаграммы системы. Модули высшего уровня содержат наиболее общие, а низшего — наиболее детализированные функции программного обеспечения. Каждый модуль содержит набор процедур, реализующих специфические функции данного модуля. Так, в случае системы охранной сигнализации тот факт, что визуальный сигнал должен появиться вследствие размыкания одного из контактных детекторов, является описанием общей функции и поэтому принадлежит одному из верхних уровней. Детали, касающиеся чувствительности контактного детектора и физического способа включения визуального сигнала, являются деталями более низ-

кого уровня. На нижних уровнях детализации программные модули более тесно связаны с аппаратными модулями системы. В этих случаях часто возникает стремление начать проектирование именно с таких уровней, поскольку они кажутся наиболее легкими и понятными. Этого следует избегать по следующим причинам:

• На начальной стадии может быть неясно, как программные функции нижнего уровня взаимодействуют с функциями верхнего уровня. Если начинать проектирование с функций нижнего уровня, то может случиться так, что потом придется вносить значительные изменения. Начиная проектирование с высшего уровня, мы обеспечиваем возможность не только более быстрой, но и более точной разработки функций нижнего уровня, что позволит уменьшить количество последующих изменений.

• Представим, что после того, как проектирование частично завершено, окажется, что стоимость или затраты времени превышают допустимые. Если при этом модули верхнего уровня уже работают, можно временно игнорировать или исключить те из программных функций, которые еще не завершены. При этом система хотя и не будет полностью соответствовать требованиям пользователей и функциональной спецификации, но уже будет действовать, демонстрируя возможность завершения. Если же используется методика проектирования *снизу вверх*, трудно будет завершить проектирование верхних уровней системы, не продемонстрировав возможностей ее функционирования вообще.

Допустим, что верхний уровень модульной структуры аппаратных средств представлен блочной диаграммой, показанной на рис. 3.1, и перейдем к следующему этапу проектирования, а именно к построению *функционально-модульной* структуры программных средств.

3.3. Функционально-модульная структура

В гл. 1 было отмечено, что микрокомпьютер является последовательным устройством, которое выполняет операции одну за другой. Поэтому на высшем уровне модульной структуры программных средств должна находиться управляющая функция, обеспечивающая последовательное исполнение системой других функций. Будем называть эту управляющую функцию ИСПОЛНИТЕЛЬНОЙ процедурой (или процедурой ИСПОЛНЕНИЯ), подразумевая при этом, что ИСПОЛНИТЕЛЬНЫЙ модуль содержит программные средства, необходимые для реализации ИСПОЛНИТЕЛЬНОЙ процедуры.

А. ВХОДЫ
1. Контактные детекторы.
2. Детектор движения.
3. Переключатель.

Б. ВЫХОДЫ
1. Визуальный сигнал.
2. Звуковой сигнал.

В. ФУНКЦИИ
1. Система включается и восстанавливается с помощью переключателя.
2. Визуальный сигнал включается либо
   (а) при размыкании контактного детектора, либо
   (б) в случае продолжительного возбуждения детектора движения в течение не менее пяти секунд.
3. Звуковой сигнал включается через шестьдесят секунд после включения визуального сигнала, если за этот промежуток времени система не восстановлена с помощью переключателя.

Рис. 3.2. Функциональная спецификация системы охранной сигнализации.

Прежде чем приступать к разбиению системы охранной сигнализации на функциональные модули, рассмотрим функциональную спецификацию, определенную в гл. 2 и воспроизведенную на рис. 3.2. Из этого рассмотрения следует, что система может быть разделена на три части: ВХОД, ВЫХОД и ФУНКЦИИ. В данном случае каждая из частей ВХОД и ВЫХОД может быть реализована в одном модуле, поскольку они являются относительно простыми. В более сложных системах может оказаться необходимым дальнейшее деление этих модулей на несколько входных и выходных.

Для части функциональной спецификации, которую мы назвали ФУНКЦИИ, можно выделить четыре различных модуля:

1. Входное состояние контактного детектора и детектора движения считывается и хранится с помощью процедуры ВХОДНОГО модуля. Однако прежде, чем должно быть предпринято какое-либо действие, необходимо проверить состояние этих входов. Все процедуры, выполняющие проверку и определяющие, какие действия должны быть предприняты по результатам проверки, необходимо сгруппировать вместе в модуль ПРОВЕРКИ.

2. Во время работы системы необходимо проверять состояние переключателя и ожидать изменения его состояния, чтобы предпринять конкретное действие. Процедуры, реализующие функции ожидания, должны быть сгруппированы в модуль ОЖИДАНИЯ.

3. В течение всего времени работы системы необходимо запускать и останавливать таймер, а также считывать его состояние. Процедуры, реализующие функции таймера, входят в модуль ТАЙМЕРА.

4. Если результат проверки одной из процедур модуля ПРОВЕРКИ указывает на то, что обнаружен нарушитель, дол-

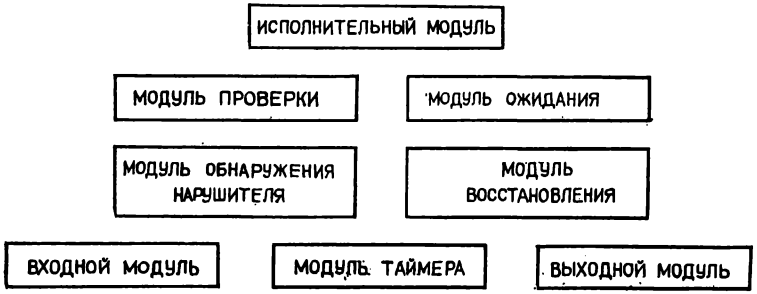


Рис. 3.3. Функционально-модульная структура системы охранной сигнализации.

жна быть вызвана процедура для включения визуального сигнала, запуска таймера на шестьдесят секунд и включения звукового сигнала после окончания работы таймера. Эта процедура должна содержаться в модуле ОБНАРУЖЕНИЯ НАРУШИТЕЛЯ.

Дополнительно к этим четырем модулям необходим пятый функциональный модуль, который не может быть выведен из функциональной спецификации. Когда система охранной сигнализации включается впервые или после выключения питания или когда она восстанавливается с помощью переключателя, она должна быть инициализирована, а все сигналы и таймер восстановлены в первоначальное состояние. Для удобства процедуры, реализующие указанные функции, необходимо сгруппировать в модуль ВОССТАНОВЛЕНИЯ.

Полученная таким образом модульная структура системы охранной сигнализации показана на рис. 3.3. Каждый из модулей размещается на одном из четырех уровней нисходящей иерархии. Нам известно, что ИСПОЛНИТЕЛЬНЫЙ модуль должен находиться на самом верхнем уровне, а ВХОДНОЙ, ВЫХОДНОЙ и ТАЙМЕРА — на самом нижнем. Резонно предположить, что модуль ОБНАРУЖЕНИЯ НАРУШИТЕЛЯ должен находиться на более низком уровне по отношению к модулю ПРОВЕРКИ, однако с определенностью установить, что все связи между модулями соответствуют тому, что изображено на рис. 3.3, не представляется возможным. Определить более точно уровень каждого модуля можно будет только после того, как будут полностью определены процедуры, входящие в модули, и их взаимосвязи.

Читатель мог уже заметить, что выбор наименований модулей осуществляется таким образом, чтобы они соответствовали по смыслу функциям, реализуемым с помощью процедур, содержащихся в модуле. Этот способ выбора наименований модулей настоятельно рекомендуется. Во многих случаях для кон-

кретного модуля можно выбрать несколько подходящих наименований. В таких случаях выбирайте то, что вам нравится. Нет необходимости углубляться в поиски чего-либо лучшего, поскольку не существует *единственно правильного* способа выбора наименования для модуля. Например, ВЫХОДНОЙ модуль рассматриваемой системы с таким же успехом мог бы быть назван СИГНАЛЬНЫМ модулем, так как выходами системы охранной сигнализации являются только сигналы.

Необходимо отметить, что способ определения принадлежности функций и процедур конкретным модулям также зависит от личного предпочтения проектировщика. Если модуль содержит слишком много процедур, он может быть разделен на несколько модулей. С целью упрощения несколько модулей, каждый из которых содержит небольшое число процедур, могут быть объединены в один (возможно, в КОМБИНИРОВАННЫЙ) модуль. В рассматриваемой системе, например, можно легко объединить модули ПРОВЕРКИ, ОЖИДАНИЯ и ОБНАРУЖЕНИЯ НАРУШИТЕЛЯ. Мы не будем, однако, этого делать, потому что наша система вообще не является слишком сложной. Как и во всяких других аспектах проектирования систем, опыт поможет быстро выбрать наиболее приемлемый вариант модульной декомпозиции. Опыт поможет вам также выбрать такие наименования модулей, которые будут иметь для вас смысловое значение и будут понятны тем, кто работает вместе с вами. Модульная структура позволяет легко добавлять функции к системе, а также изменять или удалять часть системы в любой момент цикла ее проектирования. Функциональная декомпозиция не является единственным способом разделения программного обеспечения системы на модули. Другим способом, используемым в сочетании с функциональной декомпозицией, является способ *модульной структуризации данных.*

Чтобы не нарушать последовательность изложения, рассмотрим в следующем разделе способы разбиения модулей на процедуры и затем, в первой части гл. 4,— методику проектирования процедур системы охранной сигнализации. И только затем, после завершения описания методики реализации процедур системы охранной сигнализации с помощью языка проектирования, мы рассмотрим способы модульной структуризации данных.

3.4. Процедуры

Программный модуль состоит из набора функций, принадлежащих этому модулю. В каждом модуле мы определяем набор процедур, реализующих принадлежащие модулю функции.

Рассмотрим, как определяются процедуры каждого из модулей системы охранной сигнализации.

Как было установлено ранее, ИСПОЛНИТЕЛЬНЫЙ модуль состоит из единственной процедуры, которая называется ИСПОЛНИТЕЛЬНОЙ процедурой.

Во ВХОДНОМ модуле считываются входные сигналы. Следовательно, необходимо определить процедуры для каждого из трех входных сигналов. С целью облегчения идентификации будем назначать каждой процедуре наименование, которое соотносится по смыслу с выполняемой функцией. Так, процедуры ВХОДНОГО модуля будут именоваться процедурами СЧИТЫВАНИЯ ПЕРЕКЛЮЧАТЕЛЯ, СЧИТЫВАНИЯ КОНТАКТНЫХ ДЕТЕКТОРОВ и СЧИТЫВАНИЯ ДЕТЕКТОРА ДВИЖЕНИЯ.

Процедуры ВЫХОДНОГО модуля должны включать и восстанавливать звуковой и визуальный сигналы. Функции этого модуля могут быть реализованы с помощью одной, двух или четырех процедур. Можно определить отдельные процедуры для реализации каждой из четырех выходных функций, т. е. включения и восстановления каждого из сигналов. Можно использовать единственную процедуру, выполняющую все четыре функции. Наконец, можно определить две процедуры: одну для включения сигналов, а другую — для их восстановления. Для нашего примера мы выберем последний способ и назовем процедуры ВЫХОДНОГО модуля процедурами ВКЛЮЧЕНИЯ СИГНАЛА и ВОССТАНОВЛЕНИЯ СИГНАЛОВ. Как определить, какой из сигналов должен быть включен процедурой ВКЛЮЧЕНИЯ СИГНАЛА, мы обсудим, когда будем рассматривать методику проектирования этих процедур.

Модуль ПРОВЕРКИ содержит функции проверки состояния контактных детекторов и детектора движения. Поэтому назовем эти процедуры процедурами ПРОВЕРКИ КОНТАКТНЫХ ДЕТЕКТОРОВ, ПРОВЕРКИ ДЕТЕКТОРА ДВИЖЕНИЯ и ПРОВЕРКИ ПРОДОЛЖЕНИЯ ДВИЖЕНИЯ. Процедура ПРОВЕРКИ ДЕТЕКТОРА ДВИЖЕНИЯ проверяет, сработал ли детектор движения, а процедура ПРОВЕРКИ ПРОДОЛЖЕНИЯ ДВИЖЕНИЯ определяет время, в течение которого продолжалось движение (не менее пяти секунд), чтобы устранить возможность ложной тревоги.

В модуле ОЖИДАНИЯ проверяется переключатель, и, если необходимо, система ожидает изменения включенного состояния на выключенное (состояние восстановления) и наоборот. Эти процедуры назовем процедурами ОЖИДАНИЯ ВКЛЮЧЕНИЯ ПЕРЕКЛЮЧАТЕЛЯ и ОЖИДАНИЯ ВЫКЛЮЧЕНИЯ ПЕРЕКЛЮЧАТЕЛЯ.

Модуль ТАЙМЕРА содержит процедуры, обеспечивающие запуск и остановку таймера, а также считывание состояния

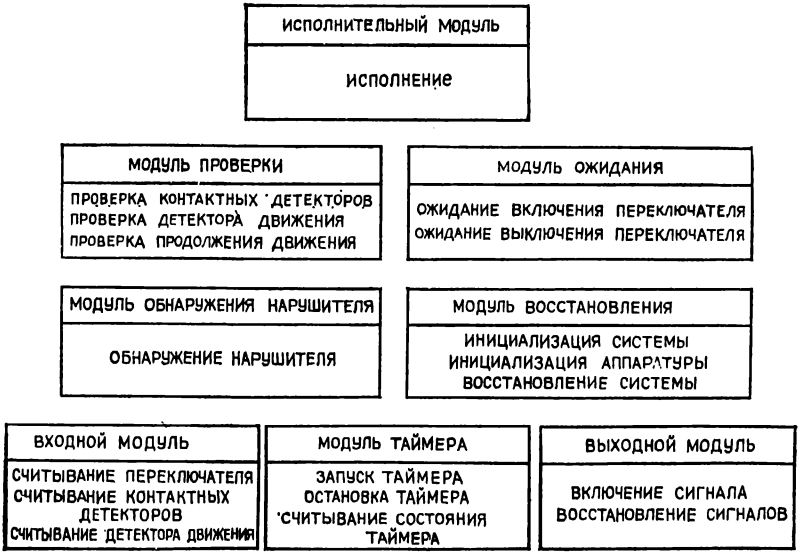


Рис. 3.4. Модульная структура системы охранной сигнализации, включающая процедуры для каждого модуля.

таймера с целью определения момента включения сигнала. Назовем эти процедуры процедурами ЗАПУСКА ТАЙМЕРА, ОСТАНОВКИ ТАЙМЕРА и СЧИТЫВАНИЯ СОСТОЯНИЯ ТАЙМЕРА.

Процедура модуля ОБНАРУЖЕНИЯ НАРУШИТЕЛЯ называется процедурой ОБНАРУЖЕНИЯ НАРУШИТЕЛЯ.

И наконец, модуль ВОССТАНОВЛЕНИЯ содержит три процедуры: процедуру инициализации системы при начальном включении или включении после отключения питания, процедуру инициализации аппаратных средств и процедуру восстановления сигналов и таймера. Назовем эти процедуры процедурами ИНИЦИАЛИЗАЦИИ СИСТЕМЫ, ИНИЦИАЛИЗАЦИИ АППАРАТУРЫ и ВОССТАНОВЛЕНИЯ СИСТЕМЫ. Функции, выполняемые процедурами ИНИЦИАЛИЗАЦИИ АППАРАТУРЫ и ВОССТАНОВЛЕНИЯ СИСТЕМЫ, подобны. Далее мы увидим, как эти процедуры соотносятся одна с другой и с процедурой ИНИЦИАЛИЗАЦИИ СИСТЕМЫ. Полная модульная структура системы охранной сигнализации, включающая процедуры для каждого модуля, показана на рис. 3.4.

Чтобы избежать чрезмерной сложности, в некоторых системах вводятся дополнительные уровни, называемые *подсистемами.*

3.5. Подсистемы

Система охранной сигнализации является не слишком большой и поэтому не очень сложной. В более сложных системах число модулей может быть значительно больше. Как указывалось в гл. 2, более сложные системы разбиваются на подсистемы, при этом функциональную спецификацию разрабатывают для каждой подсистемы. Поэтому разбиение на модули и проектирование каждой подсистемы должно выполняться так, как это выполнялось бы для небольшой системы.

На рис. 3.5. показано, как соотносятся между собой подсистемы и модули большой системы. Подсистемы высшего уровня включают ПОДСИСТЕМНЫЕ ИСПОЛНИТЕЛЬНЫЕ модули, которые содержат процедуры, вызываемые СИСТЕМНОЙ ИСПОЛНИТЕЛЬНОЙ процедурой СИСТЕМНОГО ИСПОЛНИТЕЛЬНОГО модуля. Подсистемы низшего уровня могут не иметь ПОДСИСТЕМНЫХ ИСПОЛНИТЕЛЬНЫХ модулей, если они обеспечивают служебные функции для других подсистем.

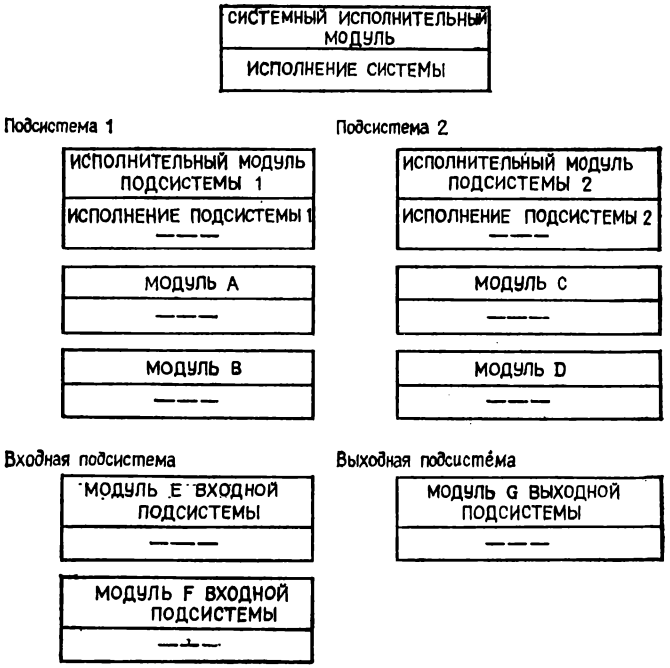


Рис. 3.5. Подсистемы.

Далее мы снова рассмотрим связь между аппаратными и программными модулями, чтобы выявить влияние соотношения между аппаратными и программными средствами на выбор модульной структуры системы.

3.6. Соотношение между аппаратными и программными средствами

Допустим, что таймер системы охранной сигнализации реализован только программными средствами, за исключением механизма генерации тактовых импульсов, способ подключения которого к микрокомпьютеру будет рассмотрен позднее. В этом

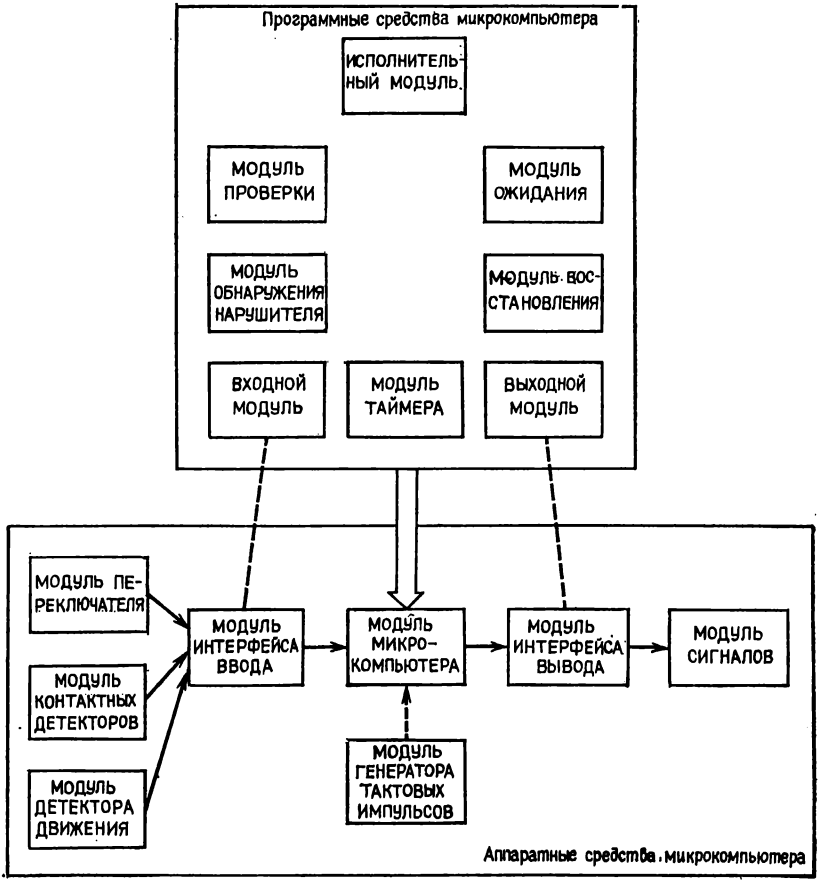


Рис. 3.6. Связь между программными и аппаратными модулями системы охранной сигнализации.

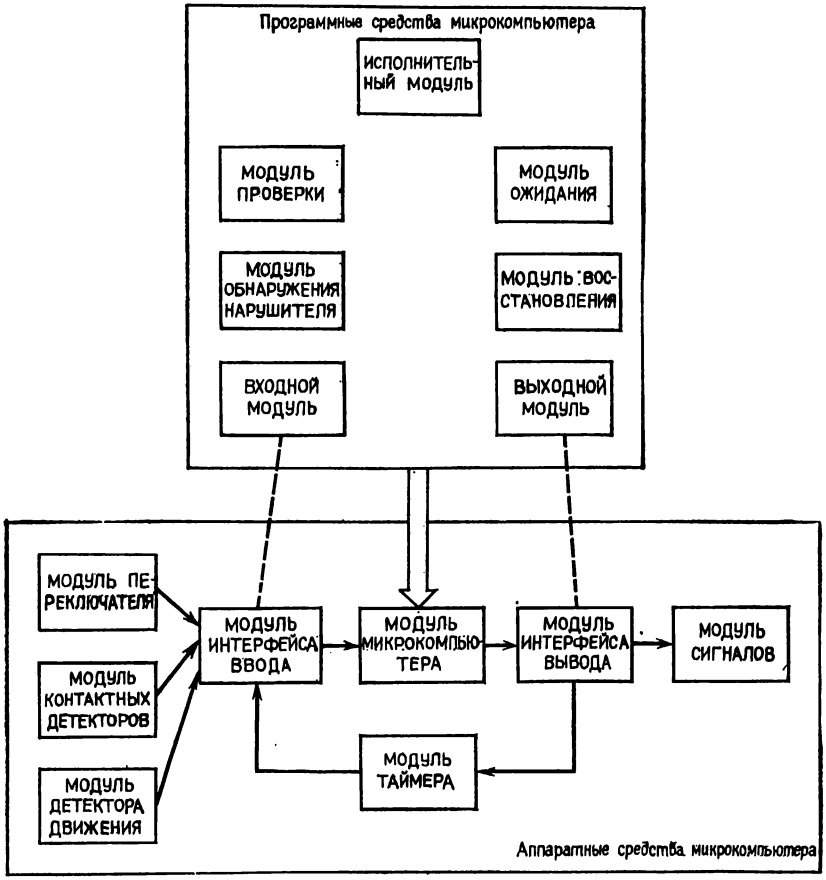


Рис. 3.7. Модульная структура системы охранной сигнализации, содержащая аппаратный модуль таймера.

случае модульная структура программного обеспечения соответствует той, что показана на рис. 3.3. Ее связь с модульной структурой аппаратных средств показана на рис. 3.6. Модуль ПРЕОБРАЗОВАНИЯ ВХОДНОГО СИГНАЛА (рис. 3.1) заменен модулями ПЕРЕКЛЮЧАТЕЛЯ, КОНТАКТНЫХ ДЕТЕКТОРОВ и ДЕТЕКТОРОВ ДВИЖЕНИЯ, а модуль ПРЕОБРАЗОВАНИЯ ВЫХОДНОГО СИГНАЛА — модулем СИГНАЛОВ. Связь между программными модулями и модулем МИКРОКОМПЬЮТЕРА показана двойной стрелкой, так как программные модули реализуются как процедуры микрокомпьютера. Поскольку входная информация передается через модуль ИНТЕРФЕЙСА ВВОДА под управлением программного ВХОД-

НОГО модуля, связь между этими модулями показана пунктирной линией. Подобным же образом показана связь между программным ВЫХОДНЫМ модулем и модулем ИНТЕРФЕЙСА ВЫВОДА.

Если функции таймера будут переданы аппаратным средствам, то модуль ТАЙМЕРА должен быть перемещен из структуры программных средств в аппаратную структуру, как показано на рис. 3.7. Три процедуры таймера будут переданы при этом программным ВХОДНОМУ и ВЫХОДНОМУ модулям, так как управление передачей информации между аппаратным модулем ТАЙМЕРА и микрокомпьютером будет теперь входить в их функции.

3.7. Проектная спецификация

Как указывалось в гл. 2, проектная спецификация содержит специфическую информацию, связанную с реализацией системы. На данном этапе проектная спецификация должна содержать схему или список подсистем, модулей, принадлежащих каждой подсистеме, и процедур, принадлежащих каждому модулю. Как отмечалось ранее, проектная спецификация данного этапа может быть подвержена изменениям в течение цикла проектирования системы.

Несмотря на то что на данном этапе цикла проектирования системы трудно составить иерархический список процедур, рано или поздно он все равно должен быть включен в проектную спецификацию. Этот список, или *дерево вызова процедур*, указывает порядок вызова процедур системы. На рис. 3.8 показано дерево вызова процедур системы охранной сигнализации. После того как будет закончена разработка процедур системы охранной сигнализации, мы снова вернемся к дереву вызова процедур, чтобы показать, как оно выводится.

Как мы уже видели, данные, обрабатываемые системой, должны быть организованы в структуры данных. Описание всех структур данных также необходимо включить в проектную спецификацию. Каждое описание структуры данных должно содержать информацию об организации данных, о месте их хранения в системе и о методах доступа к ним.

В итоге проектная спецификация должна содержать: список подсистем; список модулей; дерево вызова процедур; описание структур данных; другую информацию, не содержащуюся в описании системы на языке проектирования, но необходимую для понимания системы на уровне проектирования.

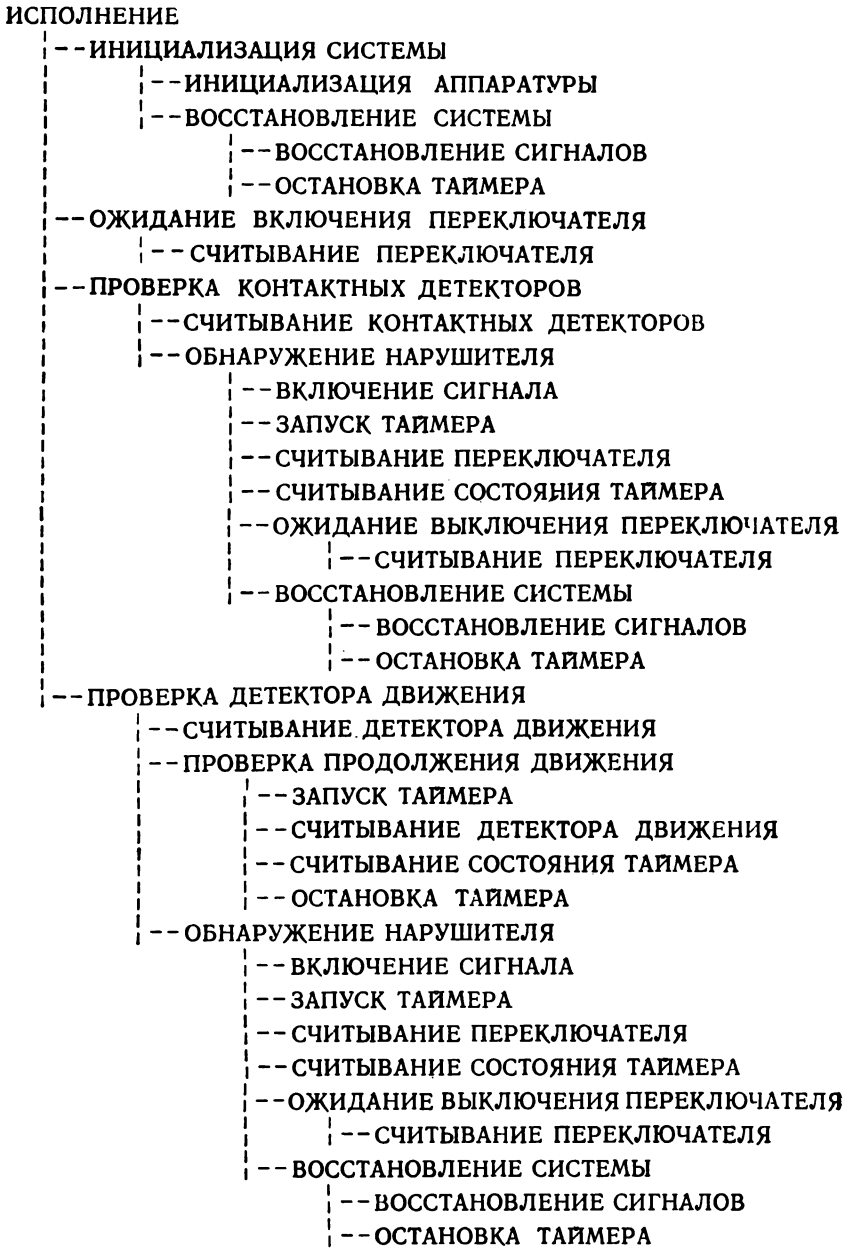


Рис. 3.8. Дерево вызова процедур системы охранной сигнализации.

3.8. Проверка проекта

Чтобы построить систему, соответствующую требованиям пользователей, ее проект должен периодически подвергаться проверке в течение всего цикла проектирования системы. Как указывалось в гл. 2, во время проектирования требования пользователей и функциональная спецификация должны часто анализироваться разработчиками вместе с возможными пользователями системы. Кроме того, после завершения построения модульной структуры системы, после преобразования функциональной спецификации в описание на языке проектирования и после реализации проекта необходимо проводить дополнительные просмотры проекта. Если пользователи или покупатели не могут принимать участие в этих проверках или их участие нежелательно, следует привлекать *специалистов* из других подразделений или организаций.

Часто во время проверки обнаруживается, что реализация части функциональной спецификации является неэффективной с точки зрения затрат. В таком случае может быть предложено альтернативное решение, которое должно быть обязательно рассмотрено возможными пользователями или покупателями. Изменение функциональной спецификации без согласования с пользователями может оказать отрицательное влияние на восприятие системы после завершения ее проектирования.

В данной главе была построена модульная структура системы охранной сигнализации, определены процедуры для каждого программного модуля, а также рассмотрены связи между программными и аппаратными средствами. В следующей главе будет рассмотрена методика проектирования процедур с использованием языка проектирования.

3.9. Упражнения

3.1. Разработайте модульную структуру телевизионного приемника, не имеющего встроенного микрокомпьютера.

3.2. Какие программные модули необходимы для реализации функциональной спецификации для телевизионного приемника с встроенным микрокомпьютером из упр. 2.3? Какие процедуры необходимы для каждого программного модуля?

3.3. Исследуйте влияние соотношения между аппаратными и программными средствами на выбор оптимальной модульной структуры телевизионного приемника с встроенным микрокомпьютером.

3.4. Какие аппаратные и программные модули необходимы для реализации функциональных спецификаций для устройства управления уличным светофором с встроенным

микрокомпьютером из упр. 2.6? Какие процедуры необходимы для каждого программного модуля?

3.5. Исследуйте влияние соотношения между аппаратными и программными средствами на выбор оптимальной модульной структуры устройства управления уличным светофором с встроенным микрокомпьютером.

3.6. Какие аппаратные и программные модули необходимы для реализации функциональных спецификаций для спортивного табло с встроенным микрокомпьютером из упр. 2.9? Какие процедуры необходимы для каждого программного модуля?

3.7. Исследуйте влияние соотношения между аппаратными и программными средствами на выбор оптимальной модульной структуры спортивного табло с встроенным микрокомпьютером.

3.8. Какие аппаратные и программные модули необходимы для реализации функциональных спецификаций для терминала розничной торговли с встроенным микрокомпьютером из упр. 2.12? Какие процедуры необходимы для каждого программного модуля?

3.9. Исследуйте влияние соотношения между аппаратными и программными средствами на выбор оптимальной модульной структуры терминала розничной торговли с встроенным микрокомпьютером.

Проектирование программного обеспечения

ЧАСТЬ I. СИСТЕМА ОХРАННОЙ СИГНАЛИЗАЦИИ

Данная глава состоит из двух частей. В первой части мы показываем, как проектируется система охранной сигнализации. При этом вводится ряд основных концепций языка проектирования. Во второй части рассматриваются общие принципы проектирования программного обеспечения. Продолжается обсуждение концепций языка проектирования на примерах, иллюстрирующих способы их использования.

4.1. Исполнительная процедура

Начнем проектирование процедур системы охранной сигнализации с процедуры самого верхнего уровня, а именно с ИСПОЛНИТЕЛЬНОЙ процедуры. Проектирование самих процедур легко и просто осуществляется с помощью нисходящего метода. Это означает, что все операции каждой из процедур должны выполняться последовательно одна за другой. Для этого используются конструкции языка проектирования ВЫПОЛНИТЬ... КОНЕЦ, ЕСЛИ... ТО... ИНАЧЕ и другие.

ИСПОЛНИТЕЛЬНАЯ процедура обеспечивает выполнение системой требуемых функций в нужной последовательности. Если возможно, ИСПОЛНИТЕЛЬНАЯ процедура не должна выполнять проверки или принимать решения вместо процедур нижнего уровня. Аналогично тому как исполнительный орган корпорации распределяет обязанности, ИСПОЛНИТЕЛЬНАЯ процедура должна *распределять обязанности* между процедурами следующего, более низкого уровня. Следовательно, она должна содержать такие конструкции языка проектирования, которые позволяют обращаться к этим процедурам и обеспечивают соответствующий отклик.

Рассмотрим проектирование исполнительной процедуры на примере ИСПОЛНИТЕЛЬНОЙ процедуры системы охранной сигнализации, изображенной на рис. 4.1. В верхней строке следом за словом ПРОЦЕДУРА указывается имя процедуры. Это необходимо для идентификации процедуры и позволяет определить ее *параметры*. Для ИСПОЛНИТЕЛЬНОЙ процедуры параметры не определены, поэтому в скобках указана только

```
ПРОЦЕДУРА:      ИСПОЛНИТЕЛЬНАЯ (;)
НАЧАЛО ПРОЦЕДУРЫ
  ВЫЗОВ: ИНИЦИАЛИЗАЦИЯ СИСТЕМЫ (;)
  ВЫПОЛНЯТЬ НЕПРЕРЫВНО
    ВЫЗОВ: ОЖИДАНИЕ ВКЛЮЧЕНИЯ ПЕРЕКЛЮЧАТЕЛЯ (;)
    ВЫЗОВ: ПРОВЕРКА КОНТАКТНЫХ ДЕТЕКТОРОВ (;)
    ВЫЗОВ: ПРОВЕРКА ДЕТЕКТОРА ДВИЖЕНИЯ (;)
  КОНЕЦ
КОНЕЦ ПРОЦЕДУРЫ
```

Рис. 4.1. ИСПОЛНИТЕЛЬНАЯ процедура системы охранной сигнализации.

точка с запятой. Подробнее параметры обсуждаются при рассмотрении примеров, в которых параметры определены.

Операции, выполняемые ИСПОЛНИТЕЛЬНОЙ процедурой, помещаются между фразами НАЧАЛО ПРОЦЕДУРЫ И КОНЕЦ ПРОЦЕДУРЫ. Это позволяет вставлять в процедуру дополнительную описательную информацию в виде текста с целью документирования. Чтобы не путать эту дополнительную информацию с операциями проектирования, ее необходимо разместить между именем процедуры и конструкцией НАЧАЛО ПРОЦЕДУРЫ. Для простоты мы опускаем дополнительную информацию при рассмотрении системы охранной сигнализации, ее использование будет рассмотрено в конце данной главы.

Первая операция ИСПОЛНИТЕЛЬНОЙ процедуры обращается к процедуре ИНИЦИАЛИЗАЦИИ СИСТЕМЫ. Операция

ВЫЗОВ: ИНИЦИАЛИЗАЦИЯ СИСТЕМЫ (;)

вызывает процедуру ИНИЦИАЛИЗАЦИИ СИСТЕМЫ. Эта процедура выполняет свои операции, пока не встретится операция ВОЗВРАТ, которая *возвращает* управление процедуре, из которой она была вызвана, в данном случае — ИСПОЛНИТЕЛЬНОЙ процедуре. ИСПОЛНИТЕЛЬНАЯ процедура после этого продолжает выполнение остальных операций. Операции ИСПОЛНИТЕЛЬНОЙ процедуры, содержащиеся внутри конструкции ВЫПОЛНЯТЬ НЕПРЕРЫВНО... КОНЕЦ, повторяются бесконечно. Таким образом в системе охранной сигнализации обеспечиваются ожидание включения переключателя (ОЖИДАНИЕ ВКЛЮЧЕНИЯ ПЕРЕКЛЮЧАТЕЛЯ) и проверка детекторов (ПРОВЕРКА КОНТАКТНЫХ ДЕТЕКТОРОВ, ПРОВЕРКА ДЕТЕКТОРА ДВИЖЕНИЯ). Эти операции повторяются до тех пор, пока система подключена к источнику питания. Заметим, что если система сигнализации включена, т. е. переключатель находится в рабочем положении, при вызове процедуры ОЖИДАНИЯ ВКЛЮЧЕНИЯ ПЕРЕКЛЮЧАТЕЛЯ управление сразу же передается ИСПОЛНИТЕЛЬНОЙ

процедуре. В этом случае начинается проверка контактных детекторов и детектора движения. Именно такие действия необходимы, когда переключатель находится в рабочем положении. Когда же система сигнализации отключена, управление от процедуры ОЖИДАНИЯ ВКЛЮЧЕНИЯ ПЕРЕКЛЮЧАТЕЛЯ не передается до тех пор, пока переключатель остается в выключенном положении. При этом проверка детекторов не начинается до тех пор, пока система не будет включена переключателем.

Как будет показано ниже, если во время выполнения процедуры ПРОВЕРКИ КОНТАКТНЫХ ДЕТЕКТОРОВ или процедуры ПРОВЕРКИ ДЕТЕКТОРА ДВИЖЕНИЯ обнаружен нарушитель, то предпринимаются соответствующие действия. Детали предпринимаемых действий содержатся в этих последних процедурах. ИСПОЛНИТЕЛЬНАЯ процедура определяет лишь последовательность действий на высшем уровне, в ней не содержатся детали, принадлежащие нижним уровням. Эти детали учитываются, когда проектируются нижние уровни. Таким образом, использование нисходящей методики, при которой детали передаются на нижние уровни, а верхние уровни остаются свободными от деталей, облегчает проектирование системы.

Сама ИСПОЛНИТЕЛЬНАЯ процедура *вызывается*, когда система впервые подключается к источнику питания или когда система восстанавливается после разного рода отказов. Этот механизм описан в гл. 8. Однако, будучи однажды вызванной, ИСПОЛНИТЕЛЬНАЯ процедура не требует возврата к вызывающей процедуре, т. е. не требует операции ВОЗВРАТ. Как будет показано ниже, для правильной работы системы любая другая процедура должна содержать хотя бы одну операцию ВОЗВРАТ.

4.2. Процедуры ИНИЦИАЛИЗАЦИИ и ВОССТАНОВЛЕНИЯ

Модуль ВОССТАНОВЛЕНИЯ содержит процедуры, которые выполняют функции инициализации и восстановления. Как

```
ПРОЦЕДУРА: ИНИЦИАЛИЗАЦИЯ СИСТЕМЫ (;)
НАЧАЛО ПРОЦЕДУРЫ
  ВЫЗОВ: ИНИЦИАЛИЗАЦИЯ АППАРАТУРЫ (;)
  ВЫЗОВ: ВОССТАНОВЛЕНИЕ СИСТЕМЫ (;)
  ВОЗВРАТ
КОНЕЦ ПРОЦЕДУРЫ
```

Рис. 4.2. Процедура ИНИЦИАЛИЗАЦИИ СИСТЕМЫ.

правило, они не слишком сложны. Мы определили три процедуры в модуле ВОССТАНОВЛЕНИЯ системы охранной

сигнализации: ИНИЦИАЛИЗАЦИИ СИСТЕМЫ, ИНИЦИАЛИЗАЦИИ АППАРАТУРЫ и ВОССТАНОВЛЕНИЯ СИСТЕМЫ. Процедура ИНИЦИАЛИЗАЦИИ СИСТЕМЫ, вызываемая ИСПОЛНИТЕЛЬНОЙ процедурой, показана на рис. 4.2. Она вызывает процедуры ИНИЦИАЛИЗАЦИИ АППАРАТУРЫ и ВОССТАНОВЛЕНИЯ СИСТЕМЫ, а затем возвращает управление вызывающей процедуре, которой в данном случае является ИСПОЛНИТЕЛЬНАЯ процедура.

Процедура ИНИЦИАЛИЗАЦИЯ АППАРАТУРЫ осуществляет восстановление и пуск нескольких функциональных

```
ПРОЦЕДУРА: ВОССТАНОВЛЕНИЕ СИСТЕМЫ (;)
НАЧАЛО ПРОЦЕДУРЫ
  ВЫЗОВ: ВОССТАНОВЛЕНИЕ СИГНАЛОВ (;)
  ВЫЗОВ: ОСТАНОВКА ТАЙМЕРА (;)
  ВОЗВРАТ
КОНЕЦ ПРОЦЕДУРЫ
```

Рис. 4.3. Процедура ВОССТАНОВЛЕНИЯ СИСТЕМЫ.

аппаратурных процедур. Более подробное рассмотрение процедуры ИНИЦИАЛИЗАЦИИ АППАРАТУРЫ будет проведено в гл. 8.

Процедура ВОССТАНОВЛЕНИЯ СИСТЕМЫ показана на рис. 4.3. Она вызывает процедуры ВОССТАНОВЛЕНИЯ СИГНАЛОВ и ОСТАНОВКИ ТАЙМЕРА, которые выключают сигналы и таймер. Дальнейшее обсуждение процедур ВХОДНОГО и ВЫХОДНОГО модулей, а также модуля ТАЙМЕРА будет продолжено в гл. 6. После выключения сигналов и таймера управление возвращается вызывающей процедуре. Как будет показано ниже, процедура ВОССТАНОВЛЕНИЯ СИСТЕМЫ вызывается из некоторых других процедур, и в каждом случае, когда она вызывается, она выключает сигналы и таймер и возвращает управление процедуре, которая ее вызвала. Использование отдельных процедур дает двойное преимущество: во-первых, система разделяется на небольшие части, в каждой из которых реализована отдельная легкопонимаемая функция; во-вторых, хотя процедура может вызываться более одного раза за время работы системы, операции процедуры не должны описываться всякий раз, когда в них есть нужда. Когда проектирование процедур системы охранной сигнализации завершится, для читателя было бы полезным переписать ИСПОЛНИТЕЛЬНУЮ процедуру, заменяя каждый из четырех вызовов на конкретные операции, содержащиеся в каждой из вызываемых процедур, и ответить на следующие вопросы: достигнута ли при этом простота и добавило ли повторение операций что-либо к описанию на языке проектирования? Чтобы лучше оценить воз-

можности использования процедур в системах с модульной структурой, читателю необходимо тщательно продумать ответы на поставленные выше вопросы.

4.3. Процедура ОЖИДАНИЯ ВКЛЮЧЕНИЯ ПЕРЕКЛЮЧАТЕЛЯ

Во многих системах необходимо ждать наступления того или иного события. Одним из способов реализации функции ожидания является использование конструкции ВЫПОЛНЯТЬ НЕПРЕРЫВНО... КОНЕЦ с внутренним условием для завершения процедуры в момент выполнения этого условия. Процедура ОЖИДАНИЯ ВКЛЮЧЕНИЯ ПЕРЕКЛЮЧАТЕЛЯ показана на рис. 4.4. Она вызывает процедуру СЧИТЫВАНИЯ

```
ПРОЦЕДУРА: ОЖИДАНИЕ ВКЛЮЧЕНИЯ ПЕРЕКЛЮЧАТЕЛЯ (;)
НАЧАЛО ПРОЦЕДУРЫ
  ВЫПОЛНЯТЬ НЕПРЕРЫВНО
    ВЫЗОВ: СЧИТЫВАНИЕ ПЕРЕКЛЮЧАТЕЛЯ (; ПЕРЕКЛЮЧАТЕЛЬ)
    ЕСЛИ ПЕРЕКЛЮЧАТЕЛЬ ВКЛЮЧЕН
      ТО ВОЗВРАТ
  КОНЕЦ
КОНЕЦ ПРОЦЕДУРЫ
```

Рис. 4.4. Процедура ОЖИДАНИЯ ВКЛЮЧЕНИЯ ПЕРЕКЛЮЧАТЕЛЯ.

ПЕРЕКЛЮЧАТЕЛЯ во ВХОДНОМ модуле и проверяет состояние переключателя: если переключатель не включен, то считывание и проверка повторяются до тех пор, пока переключатель не перейдет во включенное состояние. После этого управление передается вызывающей процедуре. Заметим, что часть ИНАЧЕ из условной конструкции ЕСЛИ... ТО... ИНАЧЕ является не обязательной и может быть опущена. Это эквивалентно записи

ИНАЧЕ НИЧЕГО НЕ ДЕЛАТЬ

Как мы отметили, использование условной конструкции с операцией ВОЗВРАТ обеспечивает простой способ прекращения операций циклической функции, которая реализуется конструкцией ВЫПОЛНЯТЬ НЕПРЕРЫВНО... КОНЕЦ. Примеры этой концепции встречаются и в других процедурах системы охранной сигнализации. Другие конструкции, которые могут быть использованы для прекращения операций цикла, будут введены ниже.

Операция

ВЫЗОВ: СЧИТЫВАНИЕ ПЕРЕКЛЮЧАТЕЛЯ (; ПЕРЕКЛЮЧАТЕЛЬ)

знакомит нас с использованием *параметра,* который в данном случае обеспечивает передачу информации от процедуры

СЧИТЫВАНИЯ ПЕРЕКЛЮЧАТЕЛЯ в процедуру ОЖИДАНИЯ ВКЛЮЧЕНИЯ ПЕРЕКЛЮЧАТЕЛЯ. Рассмотрим концепцию параметров более подробно.

4.4. Параметры

Параметры используются для связи между вызываемой и вызывающей процедурами. Например, в системе охранной сигнализации процедура СЧИТЫВАНИЯ ПЕРЕКЛЮЧАТЕЛЯ считывает состояние переключателя, т. е. определяет, находится ли переключатель в рабочем состоянии или состоянии восстановления. Эта информация затем передается в вызывающую

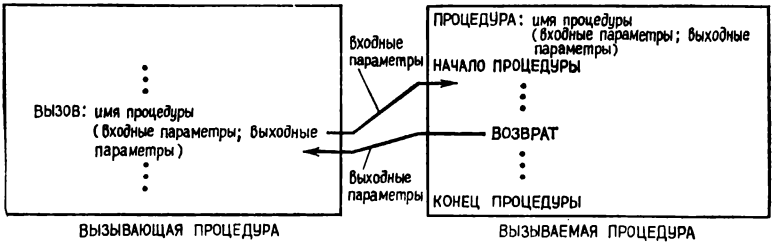


Рис. 4.5. Взаимосвязь входных и выходных параметров с соответствующими процедурами.

процедуру через параметр ПЕРЕКЛЮЧАТЕЛЬ, который расположен в скобках за точкой с запятой. Параметры, которые передают информацию от вызываемой процедуры в вызывающую процедуру (в данном примере от процедуры СЧИТЫВАНИЯ ПЕРЕКЛЮЧАТЕЛЯ в процедуру ОЖИДАНИЯ ВКЛЮЧЕНИЯ ПЕРЕКЛЮЧАТЕЛЯ), известны как *выходные параметры* и располагаются после точки с запятой. Если используется больше одного параметра, отдельные параметры разделяются запятыми. Параметры, передающие информацию в другом направлении, от вызывающей процедуры в вызываемую, называются *входными параметрами* и располагаются перед точкой с запятой. Связи между двумя процедурами и их входными и выходными параметрами показаны на рис. 4.5. Заметим, что *вход* и *выход* определены по отношению к вызываемой процедуре.

Так же как и все имена, имена параметров выбираются таким образом, чтобы они были легкочитаемыми. Каждое имя может включать любую содержательную информацию, если оно используется непротиворечиво. Если мы выберем имя ВЫКЛЮЧАТЕЛЬ вместо ПЕРЕКЛЮЧАТЕЛЬ для выражения состояния переключателя в нашем примере, то соответствую-

щие операции на рис. 4.4 будут выглядеть следующим образом:

```
ВЫЗОВ: СЧИТЫВАНИЕ ПЕРЕКЛЮЧАТЕЛЯ (; ВЫКЛЮЧАТЕЛЬ)
ЕСЛИ ВЫКЛЮЧАТЕЛЬ ВКЛЮЧЕН
  ТО ВОЗВРАТ
```

Информация, передаваемая от процедуры СЧИТЫВАНИЯ ПЕРЕКЛЮЧАТЕЛЯ процедуре ОЖИДАНИЯ ВКЛЮЧЕНИЯ ПЕРЕКЛЮЧАТЕЛЯ с помощью выходного параметра ВЫКЛЮЧАТЕЛЬ, представляет состояние переключателя. Поэтому именно ВЫКЛЮЧАТЕЛЬ должен быть проверен для того, чтобы определить необходимость возврата в вызывающую процедуру. Таким образом, согласованное использование однозначного имени параметра (либо ВЫКЛЮЧАТЕЛЬ, либо ПЕРЕКЛЮЧАТЕЛЬ) обеспечивает правильное функционирование конструкции ЕСЛИ... ТО. Как будет видно далее, для правильного функционирования системы необходимо, чтобы вся информация, обрабатываемая микрокомпьютером, была идентифицирована однозначно. Использование неповторяющихся имен для обозначения каждого элемента информации будет рассмотрено ниже, в разделе структуры данных.

А сейчас продолжим обсуждение проектирования системы охранной сигнализации и рассмотрим процедуры модуля ПРОВЕРКИ.

4.5. Процедура ПРОВЕРКИ КОНТАКТНЫХ ДЕТЕКТОРОВ

Процедура проверки содержит конструкции, определяющие существование конкретных условий. Процедура проверки вызывает также другие процедуры, необходимые для обеспечения

```
ПРОЦЕДУРА: ПРОВЕРКА КОНТАКТНЫХ ДЕТЕКТОРОВ (;)
НАЧАЛО ПРОЦЕДУРЫ
  ВЫЗОВ: СЧИТЫВАНИЕ КОНТАКТНЫХ ДЕТЕКТОРОВ (; КОНТАКТНЫЕ
  ДЕТЕКТОРЫ)
  ЕСЛИ КОНТАКТНЫЕ ДЕТЕКТОРЫ РАЗОМКНУТЫ
    ТО ВЫЗОВ: ОБНАРУЖЕНИЕ НАРУШИТЕЛЯ (;)
  ВОЗВРАТ
КОНЕЦ ПРОЦЕДУРЫ
```

Рис. 4.6. Процедура ПРОВЕРКИ КОНТАКТНЫХ ДЕТЕКТОРОВ.

правильной последовательности планируемых действий. Это аналогично введенному ранее распределению обязанностей в корпорации.

В процедуре ПРОВЕРКИ КОНТАКТНЫХ ДЕТЕКТОРОВ системы охранной сигнализации, которая показана на рис. 4.6,

первой вызывается процедура СЧИТЫВАНИЯ КОНТАКТНЫХ ДЕТЕКТОРОВ. Затем выполняется проверка состояния контактных детекторов. Если они разомкнуты, вызывается процедура ОБНАРУЖЕНИЕ НАРУШИТЕЛЯ, поскольку обнаружено присутствие нарушителя. Как будет видно ниже, сигналами тревоги управляет процедура ОБНАРУЖЕНИЯ НАРУШИТЕЛЯ, и, если система выключена, управление передается процедуре ПРОВЕРКИ КОНТАКТНЫХ ДЕТЕКТОРОВ, которая в свою очередь возвращает управление вызывающей процедуре для ожидания нового включения системы. Если при вызове процедуры ПРОВЕРКИ КОНТАКТНЫХ ДЕТЕКТОРОВ контактные детекторы не разомкнуты, то управление немедленно возвращается вызывающей процедуре.

4.6. Процедура ОБНАРУЖЕНИЯ НАРУШИТЕЛЯ

До сих пор проектирование каждой процедуры в системе охранной сигнализации состояло всего из нескольких операций. Действия, которые необходимо предпринять для обнаружения нарушителя, более объемны. Из описания модуля ОБНАРУЖЕНИЕ НАРУШИТЕЛЯ видно, что при этом включается визуальный сигнал тревоги и запускается 60-секундный таймер. Затем, когда таймер закончит отсчет, включается звуковой сигнал тревоги. Из функциональной спецификации известно, что переключатель должен проверяться неоднократно до тех пор, пока он не будет выключен и система не перейдет в невозбужденное состояние.

На рис. 4.7 показано, что вначале вызываются процедуры ВКЛЮЧЕНИЯ СИГНАЛА и ЗАПУСКА ТАЙМЕРА. Здесь мы имеем два примера использования входных параметров. Так как в системе два сигнала, то необходимо указать, какой из них будет вызываться процедурой ВКЛЮЧЕНИЕ СИГНАЛА. Операции

УСТАНОВИТЬ СИГНАЛ В ВИЗУАЛЬНЫЙ
ВЫЗОВ: ВКЛЮЧЕНИЕ СИГНАЛА (СИГНАЛ;)

вызывают ВКЛЮЧЕНИЕ визуального сигнала, поскольку входной параметр СИГНАЛ *установлен* в состояние ВИЗУАЛЬНЫЙ перед тем, как он используется в последовательности вызовов. Аналогично звуковой сигнал включается путем установки параметра СИГНАЛ в состояние ЗВУКОВОЙ перед вызовом процедуры ВКЛЮЧЕНИЯ СИГНАЛА. Конструкция УСТАНОВИТЬ, как будет показано ниже, используется также для установки выходных параметров перед возвратом из вызываемой процедуры.

Входным параметром для процедуры ЗАПУСК ТАЙМЕРА является промежуток времени в секундах между моментами запуска таймера и его срабатывания. В этом случае величина

```
ПРОЦЕДУРА: ОБНАРУЖЕНИЕ НАРУШИТЕЛЯ (;)
НАЧАЛО ПРОЦЕДУРЫ
  УСТАНОВИТЬ СИГНАЛ В ВИЗУАЛЬНЫЙ
  ВЫЗОВ: ВКЛЮЧЕНИЕ СИГНАЛА (СИГНАЛ;)
  ВЫЗОВ: ЗАПУСК ТАЙМЕРА (60 СЕКУНД;)
  ВЫПОЛНЯТЬ НЕПРЕРЫВНО
    ВЫЗОВ: СЧИТЫВАНИЕ ПЕРЕКЛЮЧАТЕЛЯ (; ПЕРЕКЛЮЧАТЕЛЬ)
    ЕСЛИ ПЕРЕКЛЮЧАТЕЛЬ ВКЛЮЧЕН
      ТО      ВЫПОЛНИТЬ
                        ВЫЗОВ: СЧИТЫВАНИЕ СОСТОЯНИЯ
                        ТАЙМЕРА (; СОСТОЯНИЕ ТАЙМЕРА)
                        ЕСЛИ ТАЙМЕР В СОСТОЯНИИ ПОКОЯ
                           ТО ВЫПОЛНИТЬ
                                        УСТАНОВИТЬ СИГНАЛ
                                        В ЗВУКОВОЙ
                                        ВЫЗОВ: ВКЛЮЧЕНИЕ
                                        СИГНАЛА (СИГНАЛ;)
                                        ВЫЗОВ: ОЖИДАНИЕ
                                        ВКЛЮЧЕНИЯ
                                        ПЕРЕКЛЮЧАТЕЛЯ (;)
                                        ВЫЗОВ:
                                        ВОССТАНОВЛЕНИЕ
                                        СИСТЕМЫ (;)
                                        ВОЗВРАТ
                                  КОНЕЦ
              КОНЕЦ
      ИНАЧЕ ВЫПОЛНИТЬ
                        ВЫЗОВ: ВОССТАНОВЛЕНИЕ СИСТЕМЫ (;)
                        ВОЗВРАТ
              КОНЕЦ
  КОНЕЦ
КОНЕЦ ПРОЦЕДУРЫ
```

Рис. 4.7. Процедура ОБНАРУЖЕНИЯ НАРУШИТЕЛЯ.

промежутка времени (60 СЕКУНД) располагается в процедуре на месте имени параметра и используется непосредственно как входной параметр. Оба этих метода определения *значения* входного параметра эквивалентны и могут быть использованы один вместо другого.

Включив визуальный сигнал и запустив таймер, необходимо ждать выключения переключателя или срабатывания таймера. Процесс ожидания реализуется в процедуре с помощью кон-

струкции выполнения ВЫПОЛНЯТЬ НЕПРЕРЫВНО... КОНЕЦ, внутри которой проверяется состояние переключателя и таймера. Сначала проверяется переключатель; если он находится во включенном состоянии, то с помощью процедуры СЧИТЫВАНИЯ СОСТОЯНИЯ ТАЙМЕРА проверяется состояние таймера. Заметим, что конструкции ВЫПОЛНИТЬ... КОНЕЦ используется в качестве *скобок* для выделения наборов операций, принадлежащих частям ТО и ИНАЧЕ конструкции ЕСЛИ... ТО... ИНАЧЕ. Операции внутри скобок ВЫПОЛНИТЬ... КОНЕЦ не повторяются, как в случае конструкции ВЫПОЛНЯТЬ НЕПРЕРЫВНО... КОНЕЦ. Используемое при этом смещение строк показывает, что набор операций внутри каждой пары скобок ВЫПОЛНИТЬ... КОНЕЦ принадлежит одному уровню, однако уже само использование конструкции ВЫПОЛНИТЬ... КОНЕЦ обеспечивает однозначность последовательности выполнения набора операций независимо от смещения. Эта проблема будет подробнее рассмотрена в разделе документации. Если в рассматриваемом примере при выполнении операций внутри части ТО конструкции ЕСЛИ... ТО.. ИНАЧЕ таймер еще не сработал, то для выполнения не остается никаких других операций. В этом случае выполняются операции внутри конструкции ВЫПОЛНЯТЬ НЕПРЕРЫВНО... КОНЕЦ, вызывая повторную проверку переключателя и таймера. Если в любое время перед срабатыванием таймера будет обнаружено, что переключатель переведен в положение восстановления, то выполняются две операции в части ИНАЧЕ конструкции ЕСЛИ... ТО... ИНАЧЕ. Система при этом восстанавливается и управление передается вызывающей процедуре.

Если во время операции переключатель остается включенным более 60 с, таймер срабатывает и выполняются следующие операции:

```
УСТАНОВИТЬ СИГНАЛ В ЗВУКОВОЙ
ВЫЗОВ: ВКЛЮЧЕНИЕ СИГНАЛА (СИГНАЛ;)
ВЫЗОВ: ОЖИДАНИЕ ВЫКЛЮЧЕНИЯ ПЕРЕКЛЮЧАТЕЛЯ (;)
ВЫЗОВ: ВОССТАНОВЛЕНИЕ СИСТЕМЫ (;)
ВОЗВРАТ
```

В результате звуковой сигнал остается включенным до тех пор, пока в процедуре ОЖИДАНИЯ ВКЛЮЧЕНИЯ ПЕРЕКЛЮЧАТЕЛЯ не будет выключен переключатель. Сигналы и таймер выключаются процедурой ВОССТАНОВЛЕНИЯ СИСТЕМЫ, после чего управление передается вызывающей процедуре.

4.7. Процедура ОЖИДАНИЯ ВЫКЛЮЧЕНИЯ ПЕРЕКЛЮЧАТЕЛЯ

Процедура ОЖИДАНИЯ ВЫКЛЮЧЕНИЯ ПЕРЕКЛЮЧАТЕЛЯ, показанная на рис. 4.8, аналогична процедуре ОЖИДА-

```
ПРОЦЕДУРА: ОЖИДАНИЕ ВЫКЛЮЧЕНИЯ ПЕРЕКЛЮЧАТЕЛЯ (;)
НАЧАЛО ПРОЦЕДУРЫ
  ВЫПОЛНЯТЬ НЕПРЕРЫВНО
    ВЫЗОВ: СЧИТЫВАНИЕ ПЕРЕКЛЮЧАТЕЛЯ (; ПЕРЕКЛЮЧАТЕЛЬ)
    ЕСЛИ ПЕРЕКЛЮЧАТЕЛЬ ВЫКЛЮЧЕН
      ТО ВОЗВРАТ
  КОНЕЦ
КОНЕЦ ПРОЦЕДУРЫ
```

Рис. 4.8. Процедура ОЖИДАНИЯ ВЫКЛЮЧЕНИЯ ПЕРЕКЛЮЧАТЕЛЯ.

НИЕ ВКЛЮЧЕНИЯ ПЕРЕКЛЮЧАТЕЛЯ, за исключением того, что проверка проводится для положения восстановления, а не рабочего положения переключателя.

4.8. Процедура ПРОВЕРКИ ДЕТЕКТОРА ДВИЖЕНИЯ

Процедура ПРОВЕРКИ ДЕТЕКТОРА ДВИЖЕНИЯ показана на рис. 4.9. Она вызывает процедуру СЧИТЫВАНИЯ ДЕ-

```
ПРОЦЕДУРА: ПРОВЕРКА ДЕТЕКТОРА ДВИЖЕНИЯ (;)
НАЧАЛО ПРОЦЕДУРЫ
  ВЫЗОВ: СЧИТЫВАНИЕ ДЕТЕКТОРА ДВИЖЕНИЯ (; ДЕТЕКТОР
  ДВИЖЕНИЯ)
  ЕСЛИ ДЕТЕКТОР ДВИЖЕНИЯ СРАБОТАЛ
     ТО ВЫПОЛНИТЬ
          ВЫЗОВ: ПРОВЕРКА ПРОДОЛЖЕНИЯ ДВИЖЕНИЯ
          (; ПРОДОЛЖЕНИЕ)
          ЕСЛИ УСТАНОВЛЕНО ПРОДОЛЖЕНИЕ
                ТО ВЫЗОВ: ОБНАРУЖЕНИЕ НАРУШИТЕЛЯ (;)
        КОНЕЦ
  ВОЗВРАТ
КОНЕЦ ПРОЦЕДУРЫ
```

Рис. 4.9. Процедура ПРОВЕРКИ ДЕТЕКТОРА ДВИЖЕНИЯ.

ТЕКТОРА ДВИЖЕНИЯ и проверяет, сработал ли детектор движения. Если он не сработал, процедура ПРОВЕРКИ ДЕТЕКТОРА ДВИЖЕНИЯ возвращает управление вызывающей процедуре. Если же он сработал, то вызывается процедура ПРОВЕРКИ ПРОДОЛЖЕНИЯ ДВИЖЕНИЯ, чтобы определить, будет ли детектор движения оставаться возбужденным в течение хотя бы 5 с. Если продолжения движения не выявлено, то процедура ПРОВЕРКИ ДЕТЕКТОРА ДВИЖЕНИЯ

возвращает управление вызывающей процедуре. Если же обнаружено продолжение движения, вызывается процедура ОБНАРУЖЕНИЯ НАРУШИТЕЛЯ для включения сигналов тревоги, как было описано ранее. Параметр ПРОДОЛЖЕНИЕ, который является выходным параметром процедуры ПРОВЕРКИ ПРОДОЛЖЕНИЯ ДВИЖЕНИЯ, показывает, остается ли детектор движения возбужденным в течение 5 с.

4.9. Процедура ПРОВЕРКИ ПРОДОЛЖЕНИЯ ДВИЖЕНИЯ

При возбуждении детектора движения для предотвращения ложной тревоги необходимо убедиться в том, что детектор остается возбужденным в течение промежутка времени не менее 5 с. Эту проверку выполняет процедура ПРОВЕРКИ ПРОДОЛЖЕНИЯ ДВИЖЕНИЯ (рис. 4.10).

```
ПРОЦЕДУРА: ПРОВЕРКА ПРОДОЛЖЕНИЯ ДВИЖЕНИЯ
(; ПРОДОЛЖЕНИЕ)
НАЧАЛО ПРОЦЕДУРЫ
  ВЫЗОВ: ЗАПУСК ТАЙМЕРА (5 СЕКУНД;)
  ВЫПОЛНЯТЬ НЕПРЕРЫВНО
    ВЫЗОВ: СЧИТЫВАНИЕ ДЕТЕКТОРА ДВИЖЕНИЯ (; ДЕТЕКТОР
    ДВИЖЕНИЯ)
    ЕСЛИ ДЕТЕКТОР ДВИЖЕНИЯ СРАБОТАЛ
      ТО      ВЫПОЛНИТЬ
                ВЫЗОВ: СЧИТЫВАНИЕ СОСТОЯНИЯ
                ТАЙМЕРА (; СОСТОЯНИЕ ТАЙМЕРА)
                ЕСЛИ ТАЙМЕР В СОСТОЯНИИ ПОКОЯ
                  ТО ВЫПОЛНИТЬ
                       ВЫЗОВ: ОСТАНОВКА ТАЙМЕРА (;)
                       УСТАНОВИТЬ ПРОДОЛЖЕНИЕ
                       ВОЗВРАТ
                     КОНЕЦ
              КОНЕЦ
      ИНАЧЕ ВЫПОЛНИТЬ
                ВЫЗОВ: ОСТАНОВКА ТАЙМЕРА (;)
                СБРОСИТЬ ПРОДОЛЖЕНИЕ
                ВОЗВРАТ
              КОНЕЦ
  КОНЕЦ
КОНЕЦ ПРОЦЕДУРЫ
```

Рис. 4.10. Процедура ПРОВЕРКИ ПРОДОЛЖЕНИЯ ДВИЖЕНИЯ.

Для начала отсчета пятисекундного интервала вызывается процедура ЗАПУСК ТАЙМЕРА. Необходимо дождаться одной из двух возможных ситуаций, когда либо детектор движения

вернется в исходное состояние, либо сработает таймер. Процесс ожидания реализуется в процедуре с помощью конструкции ВЫПОЛНЯТЬ НЕПРЕРЫВНО... КОНЕЦ, внутри которой проверяется состояние детектора движения и таймера. Если детектор движения остался возбужденным, проверяется таймер процедурой СЧИТЫВАНИЯ СОСТОЯНИЯ ТАЙМЕРА. Если таймер еще не закончил отсчет, другие операции не выполняются, а состояние детектора движения и таймера проверяется снова. Если в любой момент до срабатывания таймера детектор движения вернется в исходное состояние, тогда в части ИНАЧЕ конструкции ЕСЛИ... ТО... ИНАЧЕ выполняются операции:

ВЫЗОВ: ОСТАНОВКА ТАЙМЕРА (;)
СБРОСИТЬ ПРОДОЛЖЕНИЕ
ВОЗВРАТ

При этом таймер останавливается, параметр ПРОДОЛЖЕНИЕ сбрасывается, а управление возвращается вызывающей процедуре.

Если детектор движения остается возбужденным в течение 5 с, то срабатывает таймер и выполняются операции

ВЫЗОВ: ОСТАНОВКА ТАЙМЕРА (;)
УСТАНОВИТЬ ПРОДОЛЖЕНИЕ
ВОЗВРАТ

В результате таймер останавливается, устанавливается параметр ПРОДОЛЖЕНИЕ, а управление возвращается вызывающей процедуре.

Этим завершается рассмотрение проектирования процедур для системы охранной сигнализации. Рассмотрим теперь аспекты языка проектирования, не нашедшие отражения при проектировании системы охранной сигнализации.

ЧАСТЬ II. ОБЩИЕ ПРИНЦИПЫ

На этом заканчивается введение в описание языка проектирования на примере простой системы охранной сигнализации. В данной части главы мы закончим обсуждение языка проектирования и рассмотрение предмета проектирования программного обеспечения в целом.

4.10. Циклы

Конструкция ВЫПОЛНЯТЬ НЕПРЕРЫВНО... КОНЕЦ известна как *цикл*, поскольку операции между фразами ВЫПОЛНЯТЬ НЕПРЕРЫВНО и КОНЕЦ выполняются неоднократно, как будто они образуют бесконечный цикл. Как было показано ранее, использование бесконечного (или неопределенного)

цикла особенно полезно при реализации исполнительной функции в системах реального времени, таких, как система охранной сигнализации. Он используется также для реализации цикла, ограниченного переходом к операции ВОЗВРАТ, вызванного внутренней проверкой условий перехода. Другой конструкцией цикла является *условная* конструкция ВЫПОЛНЯТЬ ПОКА... КОНЕЦ. Для ее иллюстрации на рис. 4.11 изображена

```
ПРОЦЕДУРА: ОЖИДАНИЕ ВКЛЮЧЕНИЯ ПЕРЕКЛЮЧАТЕЛЯ (;)
НАЧАЛО ПРОЦЕДУРЫ
  ВЫЗОВ: СЧИТЫВАНИЕ ПЕРЕКЛЮЧАТЕЛЯ (; ПЕРЕКЛЮЧАТЕЛЬ)
  ВЫПОЛНЯТЬ ПОКА ПЕРЕКЛЮЧАТЕЛЬ ВЫКЛЮЧЕН
    ВЫЗОВ: СЧИТЫВАНИЕ ПЕРЕКЛЮЧАТЕЛЯ (; ПЕРЕКЛЮЧАТЕЛЬ)
  КОНЕЦ
  ВОЗВРАТ
КОНЕЦ ПРОЦЕДУРЫ
```

Рис. 4.11. Пример конструкции ВЫПОЛНЯТЬ ПОКА ... КОНЕЦ.

процедура ОЖИДАНИЯ ВКЛЮЧЕНИЯ ПЕРЕКЛЮЧАТЕЛЯ, в которой конструкция ВЫПОЛНЯТЬ НЕПРЕРЫВНО... КОНЕЦ заменена на конструкцию ВЫПОЛНЯТЬ ПОКА... КОНЕЦ. Операции внутри конструкции ВЫПОЛНЯТЬ ПОКА... КОНЕЦ повторяются до тех пор, пока условия ПЕРЕКЛЮЧАТЕЛЬ ВЫКЛЮЧЕН справедливы. Как только переключатель включается, условие становится ложным, цикл прерывается и выполняются операции, следующие за фразой КОНЕЦ. В этом случае управление возвращается в вызывающую процедуру, так как за конструкцией ВЫПОЛНЯТЬ ПОКА... КОНЕЦ следует операция ВОЗВРАТ. Заметим, что для установки параметра ПЕРЕКЛЮЧАТЕЛЬ в исходное состояние должно быть произведено обращение к процедуре СЧИТЫВАНИЯ ПЕРЕКЛЮЧАТЕЛЯ до начала выполнения условного цикла. Если эту операцию опустить, это приведет к распространенной ошибке проектирования, вызывающей непредсказуемое поведение системы в последующем.

Этот пример также иллюстрирует то, что существует много способов реализации программного обеспечения при проектировании систем. Оба метода проектирования процедуры ОЖИДАНИЯ ВКЛЮЧЕНИЯ ПЕРЕКЛЮЧАТЕЛЯ обеспечивают одинаковое функциональное поведение системы. Какую конструкцию использовать, оставляется на усмотрение проектировщика. И снова опыт и то, что кажется более логичным и простым,— лучшие путеводители в определении выбора того или иного пути в каждой ситуации.

Еще одной часто используемой конструкцией условного цикла является конструкция ВЫПОЛНИТЬ ДЛЯ КАЖДОГО...

КОНЕЦ. Эта конструкция позволяет указать, что операции, содержащиеся в цикле, должны выполняться некоторое количество раз без определения точного числа на уровне проектирования. В качестве примера рассмотрим процедуру СЧИТЫВАНИЯ ПЕРЕКЛЮЧАТЕЛЯ во ВХОДНОМ модуле системы охранной сигнализации. Число переключателей, связанных с системой, может быть неизвестно до момента сборки системы.

```
ПРОЦЕДУРА: СЧИТЫВАНИЕ КОНТАКТНЫХ ДЕТЕКТОРОВ
(; КОНТАКТНЫЕ ДЕТЕКТОРЫ)
НАЧАЛО ПРОЦЕДУРЫ
  ВЫПОЛНИТЬ ДЛЯ КАЖДОГО КОНТАКТНОГО ДЕТЕКТОРА
    ПРОВЕРИТЬ КОНТАКТНЫЙ ДЕТЕКТОР И ХРАНИТЬ ЕГО
    СОСТОЯНИЕ
    ЕСЛИ КОНТАКТНЫЙ ДЕТЕКТОР РАЗОМКНУТ
      ТО ВЫПОЛНИТЬ
           УСТАНОВИТЬ КОНТАКТНЫЕ ДЕТЕКТОРЫ
           В СОСТОЯНИЕ РАЗОМКНУТЫ
           ВОЗВРАТ
         КОНЕЦ
  КОНЕЦ
  УСТАНОВИТЬ КОНТАКТНЫЕ ДЕТЕКТОРЫ В СОСТОЯНИЕ
  НЕРАЗОМКНУТЫ
  ВОЗВРАТ
КОНЕЦ ПРОЦЕДУРЫ
```

Рис. 4.12. Пример конструкции ВЫПОЛНИТЬ ДЛЯ КАЖДОГО ... КОНЕЦ.

Тем не менее мы должны указать, что каждый контактный детектор должен быть проверен и в случае размыкания контактов *любого* контактного детектора должен быть включен сигнал тревоги. Процедура СЧИТЫВАНИЯ КОНТАКТНЫХ ДЕТЕКТОРОВ с использованием конструкции ВЫПОЛНИТЬ ДЛЯ КАЖДОГО... показана на рис. 4.12. При выполнении процедуры проверяется состояние каждого переключателя. Если хотя бы один из них разомкнут, то выходной параметр КОНТАКТНЫЕ ДЕТЕКТОРЫ устанавливается в состояние РАЗОМКНУТЫ и процедура оканчивается операцией ВОЗВРАТ. Если ни один из контактных детекторов не разомкнут, то выполняются операции, следующие после цикла, вызывая установку параметра КОНТАКТНЫЕ ПЕРЕКЛЮЧАТЕЛИ в положение НЕ РАЗОМКНУТЫ и окончание выполнения процедуры. Действительное число контактных детекторов, которые нужно проверить, или механизм для определения их числа могут быть предусмотрены при конвертировании языка проектирования в программный язык; с другой стороны, их число может быть определено и после завершения сборки.

Конструкции присваивания

```
УСТАНОВИТЬ ... НА (В) ...
УСТАНОВИТЬ ...
СБРОСИТЬ
```

```
УСТАНОВИТЬ ВРЕМЯ НА 5 ЧАСОВ
УСТАНОВИТЬ ПРОДОЛЖЕНИЕ
СБРОСИТЬ ПРОДОЛЖЕНИЕ
```

Условные конструкции

```
ЕСЛИ условие проверки есть «истина»
  ТО выполнить что-либо
ЕСЛИ условие проверки есть «истина»
  ТО выполнить что-либо
  ИНАЧЕ выполнить что-либо другое
```

```
ЕСЛИ ПЕРЕКЛЮЧАТЕЛЬ ВКЛЮЧЕН
  ТО ВОЗВРАТ
ЕСЛИ ВЫБРАНЫ ВСЕ ЗАПИСИ КРОМЕ ПОСЛЕДНЕЙ
  ТО ВОЗВРАТ
  ИНАЧЕ ВЫБРАТЬ СЛЕДУЮЩУЮ ЗАПИСЬ В ФАЙЛЕ
```

Конструкции цикла

```
ВЫПОЛНИТЬ
     •
     •
     •
КОНЕЦ
ВЫПОЛНИТЬ ДЛЯ КАЖДОГО ... набора
  предметов
     •
     •
     •
```

```
ЕСЛИ ТАЙМЕР В СОСТОЯНИИ ПОКОЯ
  ТО ВЫПОЛНИТЬ
                ВЫЗОВ: ОСТАНОВКА ТАЙМЕРА (;)
                УСТАНОВИТЬ ПРОДОЛЖЕНИЕ
                ВОЗВРАТ
     КОНЕЦ
ВЫПОЛНИТЬ ДЛЯ КАЖДОГО КОНТАКТНОГО
ДЕТЕКТОРА
  ПРОВЕРИТЬ КОНТАКТНЫЙ ДЕТЕКТОР И ХРАНИТЬ
  ЕГО СОСТОЯНИЕ
  ЕСЛИ КОНТАКТНЫЙ ДЕТЕКТОР РАЗОМКНУТ
    ТО ВЫПОЛНИТЬ
                УСТАНОВИТЬ КОНТАКТНЫЕ
                ДЕТЕКТОРЫ В СОСТОЯНИЕ
                «РАЗОМКНУТЫ»
```

	ВОЗВРАТ КОНЕЦ
КОНЕЦ	КОНЕЦ
ВЫПОЛНЯТЬ НЕПРЕРЫВНО • • •	ВЫПОЛНЯТЬ НЕПРЕРЫВНО ВЫЗОВ: СЧИТЫВАНИЕ ПЕРЕКЛЮЧАТЕЛЯ (; ПЕРЕКЛЮЧАТЕЛЬ) ЕСЛИ ПЕРЕКЛЮЧАТЕЛЬ ВКЛЮЧЕН ТО ВОЗВРАТ
КОНЕЦ	КОНЕЦ
ВЫПОЛНЯТЬ ПОКА условие проверки есть «истина» • • •	ВЫПОЛНЯТЬ ПОКА ПЕРЕКЛЮЧАТЕЛЬ ВЫКЛЮЧЕН ВЫЗОВ: СЧИТЫВАНИЕ ПЕРЕКЛЮЧАТЕЛЯ (; ПЕРЕКЛЮЧАТЕЛЬ)
КОНЕЦ	КОНЕЦ

Управляющие конструкции

ВЫЗОВ: имя процедуры (входные параметры; выходные параметры)	
	ВЫЗОВ: ЗАПУСК СИГНАЛА (СИГНАЛ;)
ВОЗВРАТ	ВОЗВРАТ

Документальные конструкции

ПРОЦЕДУРА: имя процедуры (входные параметры; выходные параметры)	
	ПРОЦЕДУРА: ЗАПУСК СИГНАЛА (СИГНАЛ ;)
НАЧАЛО ПРОЦЕДУРЫ • • •	НАЧАЛО ПРОЦЕДУРЫ • • •
КОНЕЦ ПРОЦЕДУРЫ	КОНЕЦ ПРОЦЕДУРЫ

4.11. Сложные условия

В условной конструкции или конструкции цикла могут быть использованы сложные условия. Примерами сложных условий являются:

ЕСЛИ ПЕРЕКЛЮЧАТЕЛЬ ВКЛЮЧЕН И ТАЙМЕР В СОСТОЯНИИ ПОКОЯ

ВЫПОЛНЯТЬ ПОКА ПЕРЕКЛЮЧАТЕЛЬ ВКЛЮЧЕН И ТАЙМЕР В СОСТОЯНИИ ПОКОЯ.

В сложном условии может быть также использован союз ИЛИ, равно как и любое сочетание И и ИЛИ. Условия могут быть выражены в утвердительной форме, как в вышеприведенных примерах, или в отрицательной, как в следующем примере:

ЕСЛИ ПЕРЕКЛЮЧАТЕЛЬ НЕ ВКЛЮЧЕН ИЛИ ТАЙМЕР В СОСТОЯНИИ ВОЗБУЖДЕНИЯ

Этим завершается введение в язык проектирования. Ниже дан обзор конструкций языка проектирования.

4.12. Обзор конструкций языка проектирования

Конструкции языка проектирования, введенные выше, могут быть разбиты по следующим категориям: конструкции присваивания, условные конструкции, конструкции цикла, управляющие конструкции и документальные конструкции. Все конструкции приведены в соответствие с их распределением по категориям. Каждая конструкция проиллюстрирована примерами (см. стр. 68, 69).

В *заголовке* между первой строкой процедуры и строкой НАЧАЛО ПРОЦЕДУРЫ необходима, как мы вскоре увидим, дополнительная документальная информация.

Теперь перейдем к завершению обсуждения вопросов построения модульных структур систем, которые мы начали в гл. 3.

4.13. Продолжение обсуждения модульных структур

Завершив выполнение фазы проектирования процедур для системы охранной сигнализации, можно повторно рассмотреть модульную структуру, изображенную на рис. 3.3, для того чтобы уточнить нисходящие связи между программными модулями. Поскольку процедура ОБНАРУЖЕНИЯ НАРУШИТЕЛЯ вызывает процедуры в модулях ОЖИДАНИЯ и ВОССТАНОВЛЕНИЯ, необходимо расположить эти модули так, как показано на рис. 4.13. Никакие другие связи между модулями в той структуре, что мы предположили вначале, не меняются. Введем следующие правила для установления уровня модуля:

• Процедура может быть вызвана процедурами, принадлежащими к модулям более выского уровня.

• Процедура может быть вызвана процедурой из того же самого модуля.

Читатель может сам проверить, что рис. 4.13 правильно представляет модульную структуру системы охранной сигнализации. Одним из способов проверки является построение дерева вызова процедур, которое показывает связи между процедурами, как мы определили при первом упоминании об этой концепции в гл. 3. Дерево вызова процедур для системы охранной

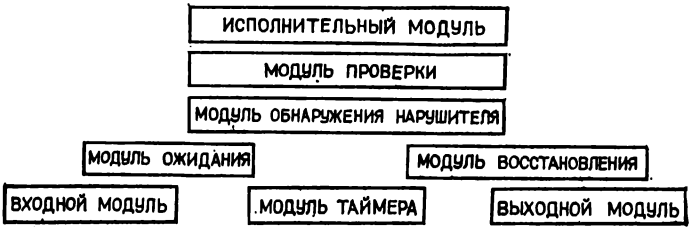


Рис. 4.13. Скорректированная модульная структура программного обеспечения системы охранной сигнализации.

сигнализации показано на рис. 4.14. На рисунке дерево представлено в форме, которая может быть выдана автоматически. Подробнее это будет рассмотрено в гл. 7. Отметим, что, если процедура вызывается более одного раза, в дереве вызова она дублируется.

Система охранной сигнализации представляет собой пример относительно простой модульной структуры. Более того, процесс декомпозиции каждого модуля на процедуры был также достаточно простым. К сожалению, большинство систем не удается спроектировать так просто. В большинстве случаев вначале проводятся приближенное разбиение на модули и декомпозиция каждого из программных модулей на процедуры. После того как будет завершено проектирование некоторых процедур, проводятся уточнение модульной структуры и последующая декомпозиция модулей. После завершения большей части проектирования могут понадобиться дополнительные итерации.

Дерево вызова процедур может быть также использовано для иллюстрации механизма передачи параметров между процедурами. Часть дерева вызова процедур системы охранной сигнализации вместе со схемой передачи параметров изображена на рис. 4.15. Помеченные стрелки обозначают входные и выходные параметры и их имена. Например, КОНТАКТНЫЕ ДЕТЕКТОРЫ — выходной параметр процедуры СЧИТЫВАНИЯ

```
ИСПОЛНЕНИЕ
  |--ИНИЦИАЛИЗАЦИЯ СИСТЕМЫ
  |     |--ИНИЦИАЛИЗАЦИЯ АППАРАТУРЫ
  |     |--ВОССТАНОВЛЕНИЕ СИСТЕМЫ
  |           |--ВОССТАНОВЛЕНИЕ СИГНАЛОВ
  |           |--ОСТАНОВКА ТАЙМЕРА
  |--ОЖИДАНИЕ ВКЛЮЧЕНИЯ ПЕРЕКЛЮЧАТЕЛЯ
  |   |--СЧИТЫВАНИЕ ПЕРЕКЛЮЧАТЕЛЯ
  |--ПРОВЕРКА КОНТАКТНЫХ ДЕТЕКТОРОВ
  |     |--СЧИТЫВАНИЕ КОНТАКТНЫХ ДЕТЕКТОРОВ
  |     |--ОБНАРУЖЕНИЕ НАРУШИТЕЛЯ
  |           |--ВКЛЮЧЕНИЕ СИГНАЛА
  |           |--ЗАПУСК ТАЙМЕРА
  |           |--СЧИТЫВАНИЕ ПЕРЕКЛЮЧАТЕЛЯ
  |           |--СЧИТЫВАНИЕ СОСТОЯНИЯ ТАЙМЕРА
  |           |--ОЖИДАНИЕ ВЫКЛЮЧЕНИЯ ПЕРЕКЛЮЧАТЕЛЯ
  |           |           |--СЧИТЫВАНИЕ ПЕРЕКЛЮЧАТЕЛЯ
  |           |--ВОССТАНОВЛЕНИЕ СИСТЕМЫ
  |                       |--ВОССТАНОВЛЕНИЕ СИГНАЛОВ
  |                       |--ОСТАНОВКА ТАЙМЕРА
  |--ПРОВЕРКА ДЕТЕКТОРА ДВИЖЕНИЯ
  |     |--СЧИТЫВАНИЕ ДЕТЕКТОРА ДВИЖЕНИЯ
  |     |--ПРОВЕРКА ПРОДОЛЖЕНИЯ ДВИЖЕНИЯ
  |     |     |--ЗАПУСК ТАЙМЕРА
  |     |     |--СЧИТЫВАНИЕ ДЕТЕКТОРА ДВИЖЕНИЯ
  |     |     |--СЧИТЫВАНИЕ СОСТОЯНИЯ ТАЙМЕРА
  |     |     |--ОСТАНОВКА ТАЙМЕРА
  |     |--ОБНАРУЖЕНИЕ НАРУШИТЕЛЯ
  |           |--ВКЛЮЧЕНИЕ СИГНАЛА
  |           |--ЗАПУСК ТАЙМЕРА
  |           |--СЧИТЫВАНИЕ ПЕРЕКЛЮЧАТЕЛЯ
  |           |--СЧИТЫВАНИЕ СОСТОЯНИЯ ТАЙМЕРА
  |           |--ОЖИДАНИЕ ВЫКЛЮЧЕНИЯ ПЕРЕКЛЮЧАТЕЛЯ
  |           |           |--СЧИТЫВАНИЕ ПЕРЕКЛЮЧАТЕЛЯ
  |           |--ВОССТАНОВЛЕНИЕ СИГНАЛОВ
  |                       |--ВОССТАНОВЛЕНИЕ СИСТЕМЫ
  |                       |--ОСТАНОВКА ТАЙМЕРА
```

Рис. 4.14. Дерево вызова процедур системы охранной сигнализации.

КОНТАКТНЫХ ДЕТЕКТОРОВ. Так как процедура ПРОВЕРКИ КОНТАКТНЫХ ДЕТЕКТОРОВ вызывает процедуру СЧИТЫВАНИЯ КОНТАКТНЫХ ДЕТЕКТОРОВ, стрелка КОНТАКТНЫЕ ДЕТЕКТОРЫ между двумя процедурами полностью определяет характер связи между ними.

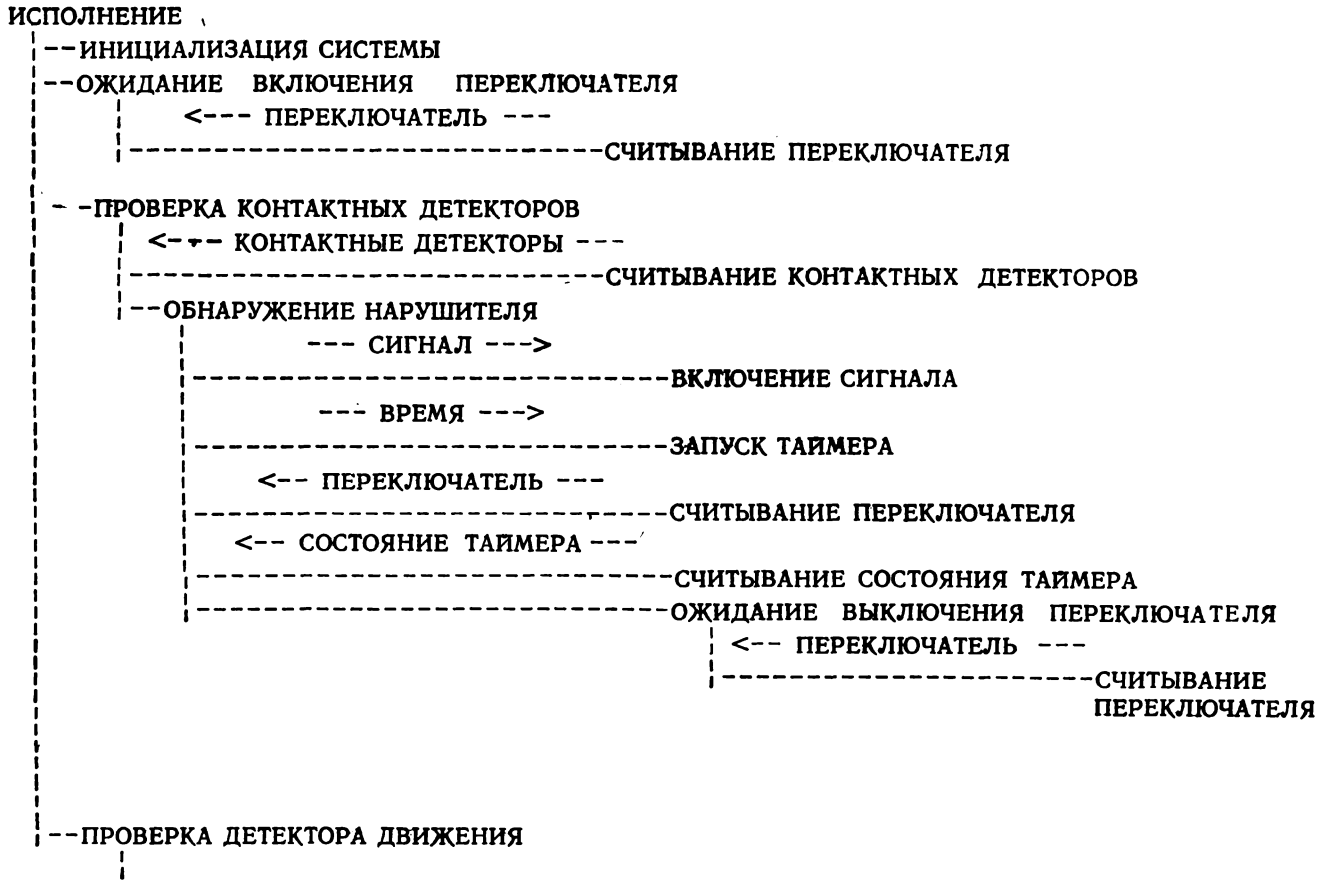
Поскольку точных правил разбиения системы на модули не существует, полезно разработать концепцию *модульной структуры данных*, которая может быть использована вместе с функциональной модульной структурой для описания проекта в первом приближении.

4.14. Модульная структура данных

В большинстве существующих систем информация запоминается в *структурах данных* и извлекается из них во время операций. Структуры данных — это организованные наборы записей данных или информации, которые хранятся в микрокомпьютерной системе. Вопросы организации информации в структуры данных будут рассмотрены в последующих разделах. Сейчас же рассмотрим связи между модульной структурой и структурами данных.

Для обеспечения целостности данных в структуре данных спроектируем отдельный модуль, содержащий описание каждой структуры данных. Затем мы условимся о том, что процедуры в других модулях не имеют непосредственно доступа к данным, записанным в структуре данных рассматриваемого модуля. Для каждого модуля, содержащего структуру данных, предусматриваются процедуры, с помощью которых производится выборка данных из модуля. Представим, например, что в проектируемой системе содержится структура данных, называемая ЗАПИСЬ. Для нас сейчас не важно, какую информацию содержит ЗАПИСЬ, поскольку необходимо выяснить, как происходит управление информацией, а не что она означает. Определим модуль ПОДДЕРЖКИ ЗАПИСИ для структуры данных ЗАПИСЬ. Далее определим процедуру ЧТЕНИЯ ЗАПИСИ в этом модуле, которая считывает информацию из структуры данных ЗАПИСЬ. Поэтому, если процедура другого модуля требует информацию из ЗАПИСИ, она вызывает процедуру ЧТЕНИЯ ЗАПИСИ, которая и осуществляет выборку. Эта концепция иллюстрируется на рис. 4.16, где показано, как процедура ЧТЕНИЯ ЗАПИСИ *считывает* запись из структуры данных ЗАПИСЬ для использования процедурой в модуле ОБРАБОТКИ.

В модуле ПОДДЕРЖКИ ЗАПИСИ могут быть также определены другие процедуры для записи информации в структуру


ИСПОЛНЕНИЕ
--ИНИЦИАЛИЗАЦИЯ СИСТЕМЫ
--ОЖИДАНИЕ ВКЛЮЧЕНИЯ ПЕРЕКЛЮЧАТЕЛЯ
<--- ПЕРЕКЛЮЧАТЕЛЬ ---
СЧИТЫВАНИЕ ПЕРЕКЛЮЧАТЕЛЯ
--ПРОВЕРКА КОНТАКТНЫХ ДЕТЕКТОРОВ
<--- КОНТАКТНЫЕ ДЕТЕКТОРЫ ---
СЧИТЫВАНИЕ КОНТАКТНЫХ ДЕТЕКТОРОВ
--ОБНАРУЖЕНИЕ НАРУШИТЕЛЯ
--- СИГНАЛ --->
ВКЛЮЧЕНИЕ СИГНАЛА
--- ВРЕМЯ --->
ЗАПУСК ТАЙМЕРА
<-- ПЕРЕКЛЮЧАТЕЛЬ ---
СЧИТЫВАНИЕ ПЕРЕКЛЮЧАТЕЛЯ
<-- СОСТОЯНИЕ ТАЙМЕРА ---
СЧИТЫВАНИЕ СОСТОЯНИЯ ТАЙМЕРА
ОЖИДАНИЕ ВЫКЛЮЧЕНИЯ ПЕРЕКЛЮЧАТЕЛЯ
<-- ПЕРЕКЛЮЧАТЕЛЬ ---
СЧИТЫВАНИЕ
ПЕРЕКЛЮЧАТЕЛЯ
--ПРОВЕРКА ДЕТЕКТОРА ДВИЖЕНИЯ

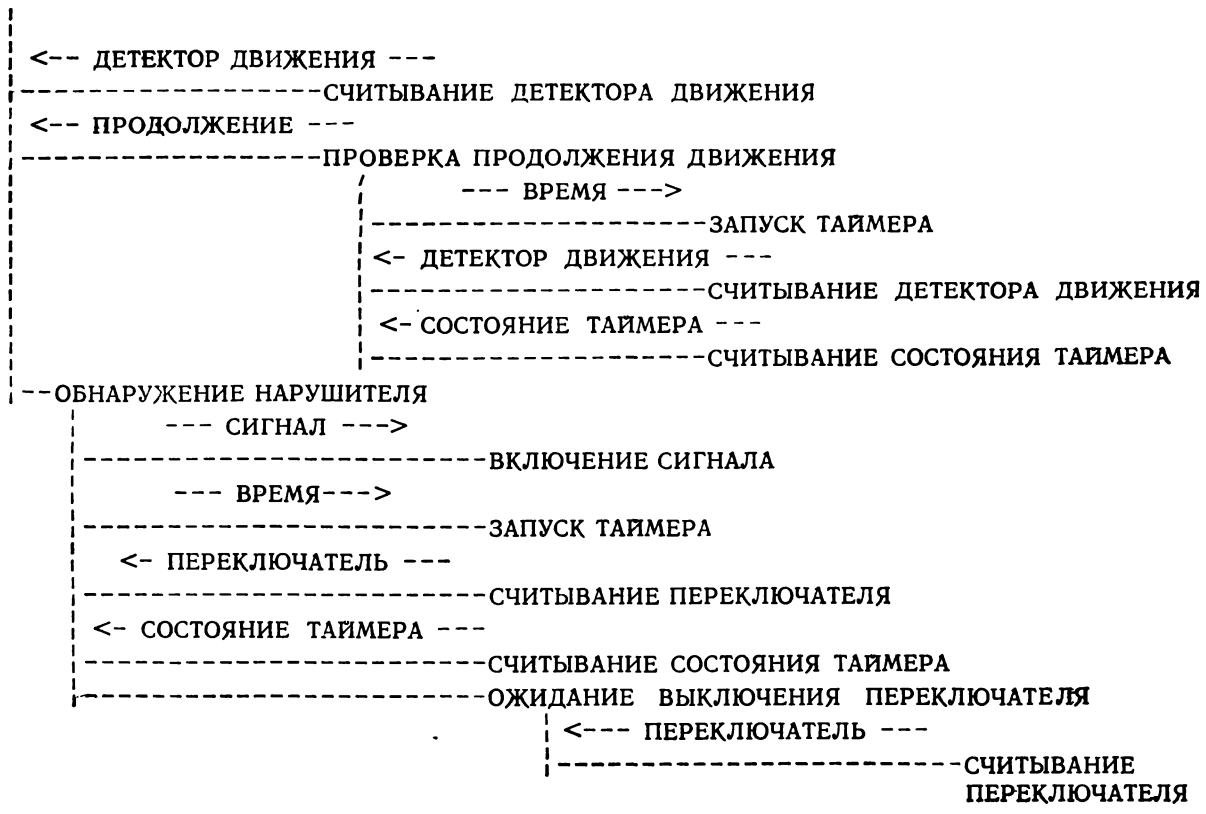


Рис. 4.15. Частичное дерево вызова процедур с параметрами.

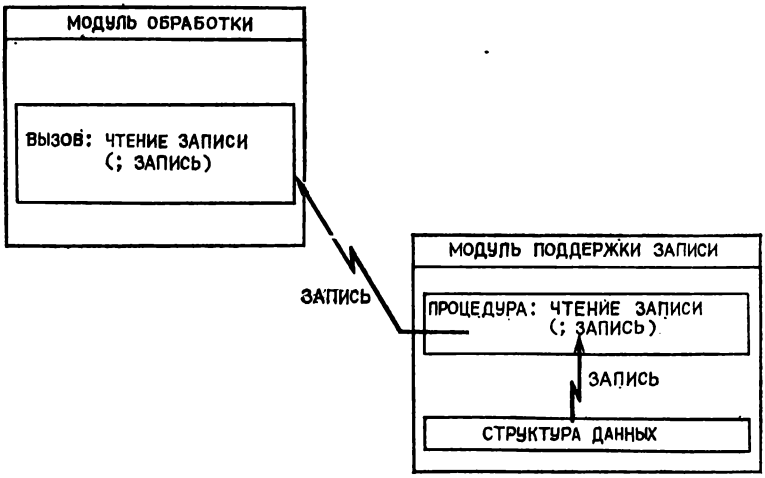


Рис. 4.16. Считывание записи данных из структуры данных ЗАПИСЬ В МОДУЛЕ ПОДДЕРЖКИ ЗАПИСИ.

данных ЗАПИСЬ, для поиска или другой обработки информации в структуре данных. Более подробно последняя концепция будет рассмотрена ниже, в разделе структур данных.

4.15. Отладка описания на языке проектирования

Мы уже рассмотрели, как построить модульную структуру системы и как спроектировать процедуры, принадлежащие каждому модулю. Теперь нужно проверить корректность проектирования для того, чтобы выявить и устранить как можно большее число ошибок до завершения создания системы. При *отладке* системы на уровне языка проектирования могут быть использованы две технологии: *просмотр проектирования* и *выверка потока данных*. Эти технологии могут также быть использованы при рецензировании проекта.

4.16. Просмотр проекта

На этом этапе цикла проектирования нет простого пути для автоматической обработки операций языка проектирования с целью проверки правильности проекта. Поэтому необходимо вручную просмотреть каждую операцию каждой процедуры. Прежде чем начать обсуждение, как выполняется просмотр проекта, необходимо понять, что мы будем проверять во время просмотра.

Если процедура не содержит условных конструкций или конструкций цикла, то проверка правильности работы процедур довольно проста. Операции языка проектирования просматриваются одна за другой, при этом они могут либо содержать желаемое действие, либо нет. Необходимо также проверить согласование входных и выходных параметров, передаваемых между каждой вызываемой и вызывающей процедурами.

Если процедура содержит один или более условных циклов, необходимо проверить, достигнут ли желаемый результат при завершении операций цикла. Иногда достаточно удостовериться, что операции в цикле выполняются определенное число раз. В другом случае может оказаться необходимым проверить, завершен ли определенный вычислительный процесс или имело ли место ожидаемое событие.

Если процедура содержит условные конструкции, необходимо убедиться в том, что выполнение каждого условия по каждой из ветвей приводит к выполнению соответствующих операций. Необходимо также удостовериться в том, что операции вызывают ожидаемое действие в каждой из ветвей. Результат проверки зависит от значений входных параметров процедуры или значений выходных параметров, переданных из вызываемой процедуры. Эти входные и выходные параметры часто зависят от внешних факторов, значения которых во время просмотра неизвестны. Поэтому необходимо предположить, что они имеют значения, соответствующие тем, которые можно ожидать во время реального функционирования системы. Обычно бывает нереально проверить каждый возможный путь через систему, поскольку их число слишком велико. Если система проектировалась на модульной основе, то, выбирая соответствующие значения входных и выходных параметров, можно проверить каждую *часть* каждого пути. Хотя просмотр не может обеспечить полной проверки, он дает уверенность в том, что выявлены наиболее общие ошибки.

Так как многие проекты требуют взаимодействия всех членов коллектива проектировщиков, просмотр должен проводиться ими совместно. Проектировщик, разрабатывающий процедуры каждого отдельного модуля, должен провести просмотр и показать, что должно произойти на каждом шаге каждой процедуры. Другие проектировщики должны критически проанализировать проект и, действуя как *«черные оппоненты»*, попытаться выявить ошибки. Если обнаружена ошибка, то либо проектировщик исправляет ее, либо коллектив разработчиков предлагает пути для ее устранения. Просмотр модуля считается успешным, если в проектном описании модуля не найдено ошибок. Рассмотрим пример просмотра модуля ПРОВЕРКИ

системы охранной сигнализации. Вернемся к рис. 4.6, 4.9 и 4.10, где изображены отдельные процедуры этого модуля.

В процедуре ПРОВЕРКИ КОНТАКТНЫХ ДЕТЕКТОРОВ считывается состояние контактных детекторов и проводится единственная проверка. Потенциальной ошибкой в этой процедуре может быть только неправильное проведение проверки. Так как КОНТАКТНЫЕ ДЕТЕКТОРЫ размыкаются только при наличии нарушителя, то проверка, показанная на рис. 4.6, спроектирована правильно.

Процедура ПРОВЕРКИ ДЕТЕКТОРА ДВИЖЕНИЯ содержит две проверки, которые являются вложенными, и поэтому получаются три возможных пути в процедуре. Сначала считывается детектор движения и проверяется его состояние. Если детектор движения не возбужден, выполняется операция ВОЗВРАТ, на этом процедура завершается. Это один путь, и ясно, что он функционирует корректно. Если детектор возбужден, то система проверяется на продолжение движения, если продолжительное движение не выявлено, то снова выполняется операция ВОЗВРАТ, завершающая процедуру. Это второй путь, и он также функционирует правильно. Если выявлено продолжительное движение, вызывается процедура ОБНАРУЖЕНИЯ НАРУШИТЕЛЯ, которая включает сигналы тревоги. В конце работы процедуры ОБНАРУЖЕНИЯ НАРУШИТЕЛЯ система восстанавливается, управление возвращается к процедуре ПРОВЕРКИ ДЕТЕКТОРА ДВИЖЕНИЯ и выполняется операция ВОЗВРАТ для завершения процедуры. Это третий путь в процедуре. Таким образом, даже в этом более сложном примере простого обсуждения возможностей достаточно для заключения о том, что здесь нет ошибок проектирования.

Процедура ПРОВЕРКИ ПРОДОЛЖЕНИЯ ДВИЖЕНИЯ содержит цикл и две вложенные проверки. Таким образом, необходимо проверить три возможных пути по процедуре, а также убедиться в правильности завершения цикла. Необходимо также убедиться в том, что процедура правильно осуществляет передачу информации обратно к вызывающей процедуре через параметр ПРОДОЛЖЕНИЕ. В этой процедуре пускается пятисекундный таймер и начинается выполнение операций цикла. В течение цикла считывается информация детектора движения и проводится проверка его состояния. Если его состояние изменилось и он больше не возбуждается, таймер останавливается, а параметр ПРОДОЛЖЕНИЕ сбрасывается для обозначения того, что движение продолжалось менее пяти секунд. Управление при этом возвращается в вызывающую процедуру, завершая тем самым цикл. Первый путь показан на рис. 4.17а. Если движение продолжается, цикл повторяется до тех пор, пока не сработает таймер. Второй путь показан на рис. 4.17б.

```
ПРОЦЕДУРА: ПРОВЕРКА ПРОДОЛЖЕНИЯ ДВИЖЕНИЯ
(; ПРОДОЛЖЕНИЕ)
НАЧАЛО ПРОЦЕДУРЫ
  ВЫЗОВ: ЗАПУСК ТАЙМЕРА (5 СЕКУНД;)
  ВЫПОЛНЯТЬ НЕПРЕРЫВНО
    ВЫЗОВ: СЧИТЫВАНИЕ ДЕТЕКТОРА ДВИЖЕНИЯ (; ДЕТЕКТОР
    ДВИЖЕНИЯ)
    ЕСЛИ ДЕТЕКТОР ДВИЖЕНИЯ СРАБОТАЛ
      ТО      ВЫПОЛНИТЬ
                  ---
              КОНЕЦ
       ИНАЧЕ ВЫПОЛНИТЬ
                ВЫЗОВ: ОСТАНОВКА ТАЙМЕРА (;)
                СБРОСИТЬ ПРОДОЛЖЕНИЕ
                ВОЗВРАТ
              КОНЕЦ
  КОНЕЦ
КОНЕЦ ПРОЦЕДУРЫ
```

Рис. 4.17а. Первый путь просмотра процедуры ПРОВЕРКИ ПРОДОЛЖЕНИЯ ДВИЖЕНИЯ.

```
ПРОЦЕДУРА: ПРОВЕРКА ПРОДОЛЖЕНИЯ ДВИЖЕНИЯ
(; ПРОДОЛЖЕНИЕ)
НАЧАЛО ПРОЦЕДУРЫ
  ВЫЗОВ: ЗАПУСК ТАЙМЕРА (5 СЕКУНД;)
  ВЫПОЛНЯТЬ НЕПРЕРЫВНО
    ВЫЗОВ: СЧИТЫВАНИЕ ДЕТЕКТОРА ДВИЖЕНИЯ (; ДЕТЕКТОР
    ДВИЖЕНИЯ)
    ЕСЛИ ДЕТЕКТОР ДВИЖЕНИЯ СРАБОТАЛ
      ТО      ВЫПОЛНИТЬ
                ВЫЗОВ: СЧИТЫВАНИЕ СОСТОЯНИЯ ТАЙМЕРА
                (; СОСТОЯНИЕ ТАЙМЕРА)
                ЕСЛИ ТАЙМЕР В СОСТОЯНИИ ПОКОЯ
                  ТО ВЫПОЛНИТЬ
                       ---
                     КОНЕЦ
              КОНЕЦ
       ИНАЧЕ ВЫПОЛНИТЬ
                ---
              КОНЕЦ
  КОНЕЦ
КОНЕЦ ПРОЦЕДУРЫ
```

Рис. 4.17б. Второй путь просмотра процедуры ПРОВЕРКИ ПРОДОЛЖЕНИЯ ДВИЖЕНИЯ.

```
ПРОЦЕДУРА: ПРОВЕРКА ПРОДОЛЖЕНИЯ ДВИЖЕНИЯ
(; ПРОДОЛЖЕНИЕ)
  НАЧАЛО ПРОЦЕДУРЫ
  ВЫЗОВ: ПУСК ТАЙМЕРА (5 СЕКУНД;)
  ВЫПОЛНЯТЬ НЕПРЕРЫВНО
    ВЫЗОВ: СЧИТЫВАНИЕ ДЕТЕКТОРА ДВИЖЕНИЯ (; ДЕТЕКТОР
    ДВИЖЕНИЯ)
    ЕСЛИ ДЕТЕКТОР ДВИЖЕНИЯ СРАБОТАЛ
      ТО      ВЫПОЛНИТЬ
                ВЫЗОВ: СЧИТЫВАНИЕ СОСТОЯНИЯ ТАЙМЕРА
                (; СОСТОЯНИЕ ТАЙМЕРА)
                ЕСЛИ ТАЙМЕР В СОСТОЯНИИ ПОКОЯ
                  ТО ВЫПОЛНИТЬ
                       ВЫЗОВ: ОСТАНОВКА ТАЙМЕРА (;)
                       УСТАНОВИТЬ ПРОДОЛЖЕНИЕ
                       ВОЗВРАТ
                     КОНЕЦ
              КОНЕЦ
      ИНАЧЕ ВЫПОЛНИТЬ
                 ---
              КОНЕЦ
  КОНЕЦ
КОНЕЦ ПРОЦЕДУРЫ
```

Рис. 4.17в. Третий путь просмотра процедуры ПРОВЕРКИ ПРОДОЛЖЕНИЯ ДВИЖЕНИЯ.

Когда таймер срабатывает, устанавливается параметр ПРОДОЛЖЕНИЕ для обозначения того, что движение продолжалось в течение 5 с. После этого управление передается вызывающей процедуре для завершения цикла. Этот третий путь показан на рис. 4.17в. На каждом рисунке те операции, которые не являются частью рассматриваемого пути, опущены, а интересующие конструкции выделяются жирным шрифтом.

Приведенный пример является характерным примером обсуждения, проводимого при проверке проекта методом просмотра. Проектировщик может также представить наглядный материал в виде схем (рис. 4.17а — 4.17в), которые иллюстрируют три пути по процедуре ПРОВЕРКИ ПРОДОЛЖЕНИЯ ДВИЖЕНИЯ.

Авторами был разработан *эмулятор просмотра* для автоматизации этого процесса. Эмулятор просматривает операции языка проектирования и выводит их на дисплей одну за другой, что позволяет всем участникам просмотра легко наблюдать за последовательностью операций. Проектировщику остается только указать ветвь, по которой должна идти проверка, а также указать, выполняются или нет условия, определяемые в конструкции условного цикла. Необходимо отметить, что

даже автоматический просмотр не может обеспечить исчерпывающей проверки каждого возможного пути в системе. Это *инструмент* в руках проектировщика, который помогает убедиться в том, что система спроектирована в соответствии с требуемыми спецификациями.

Технология просмотра достаточно проста, потому что система построена на модульной основе и каждый модуль разбит на относительно небольшие процедуры. Таким образом, сложность системы уменьшена до уровня, на котором все элементы могут обсуждаться в ходе просмотра. Другим методом, который может быть использован для проверки правильности проектирования, является метод *выверки потока данных.*

4.17. Выверка потока данных

Успешное проведение просмотра системы удостоверяет, что контрольные пути через систему выполнены корректно. Выверка потока данных проверяет правильность обработки информа-

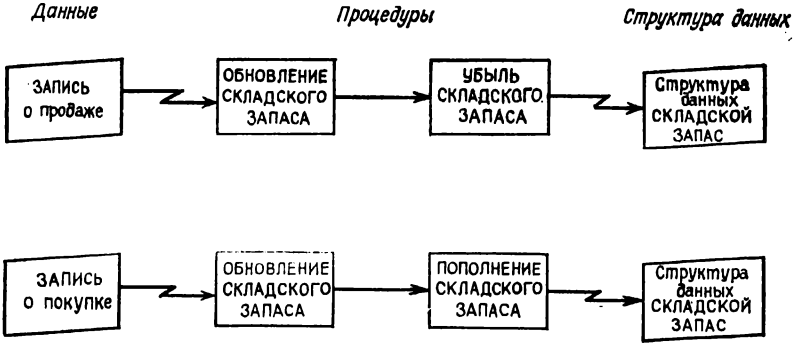


Рис. 4.18. Диаграмма потока данных для струкутры данных СКЛАДСКОЙ ЗАПАС.

ции в системе. Она включает в себя как информацию, которой обмениваются процедуры через параметры, так и информацию, хранимую в структурах или полученную из структур данных.

Диаграммы потоков данных могут наглядно показать прохождение потока информации через систему. Они иллюстрируют передачу информации через процедуры на ее пути из структур (в структуры) данных. На рис. 4.18 изображена диаграмма потока данных для системы управления запасами. Из диаграммы видно, что структура данных СКЛАДСКОЙ ЗАПАС обновляется при каждой продаже или закупке товара. Подготовив диаграммы потока данных перед просмотром проектирования, во время просмотра можно определить, что данные прошли через систему правильно. С другой стороны, потоком

данных можно управлять, и диаграммы потока данных, показывающие движение данных, могут быть сделаны во время просмотра. Затем диаграммы анализируются для того, чтобы определить, правильно ли данные прошли через систему. В любом случае мы используем диаграммы потока данных для проверки правильности прохождения информации через систему.

Эмулятор просмотра, описанный в предыдущем разделе, может генерировать диаграммы потоков данных для каждого просмотра. При этом для каждого отдельного пути генерируется одна диаграмма потока данных. С помощью соответствующего выбора путей во время просмотра может быть получен требуемый набор диаграмм потоков данных.

4.18. Структуры данных

Во время функционирования микрокомпьютер управляет информацией в виде записей данных. Для удобства и уменьшения трудностей при обработке соотносящиеся между собой записи данных объединяются в наборы, которые запоминаются в *структурах данных*. Таким образом, одна структура данных может содержать большое количество взаимосвязанной информации. Организация структуры данных зависит от того, как представлены данные и как они обрабатываются системой.

Простейшая структура данных, состоящая из нескольких элементов данных, называется *списком*. Данные запоминаются в последовательности, соответствующей списку, обычно таким же образом, каким они создаются или генерируются. Структура данных ВХОДНАЯ ЗАПИСЬ системы, которая использует клавишный ввод информации, является примером списка. При каждом нажатии клавиши оператором запоминается *символьный код*, обозначаемый данной клавишей. Когда оператор заканчивает ввод данных, список содержит символьные коды, соответствующие последовательности нажатия клавиш оператором. Примером клавиатуры, предназначенной для использования в системе расчета за покупки, является клавиатура, изображенная на рис. 2.2. В системе этого типа входная последовательность может состоять из значений величины счета или цены изделия. Рассмотрим, как проектируются в таких системах процедуры обработки структуры данных ВХОДНАЯ ЗАПИСЬ в модуле ПОДДЕРЖКИ ВХОДНОЙ ЗАПИСИ.

Процедура ЗАПОЛНЕНИЯ ВХОДНОЙ ЗАПИСИ (рис. 4.19) вызывается, когда оператору необходимо ввести последовательность *цифровых* данных. Вначале в структуре данных ВХОДНАЯ ЗАПИСЬ стирается предыдущая информация. Для считывания символьных кодов, генерируемых во время нажимания клавиш на клавиатуре, используется цикл. Цифровые сим-

```
ПРОЦЕДУРА: ЗАПОЛНЕНИЕ ВХОДНОЙ ЗАПИСИ (; ВХОДНАЯ ЗАПИСЬ)
НАЧАЛО ПРОЦЕДУРЫ
  ВЫЗОВ: СТИРАНИЕ ВХОДНОЙ ЗАПИСИ (;)
  ВЫПОЛНЯТЬ НЕПРЕРЫВНО
    ВЫЗОВ: СЧИТЫВАНИЕ КЛАВИАТУРЫ (; СИМВОЛ)
    ЕСЛИ СИМВОЛ СОДЕРЖИТ КОД "СТИРАНИЕ ВХОДА"
      ТО ВЫЗОВ: СТИРАНИЕ ВХОДНОЙ ЗАПИСИ (;)
    ЕСЛИ СИМВОЛ СОДЕРЖИТ ЦИФРОВОЙ КОД
      ТО ВЫПОЛНИТЬ:
           ВЫЗОВ: ПРОВЕРКА ЗАПОЛНЕНИЯ ВХОДНОЙ ЗАПИСИ
           (; СТАТУС)
           ЕСЛИ СТАТУС ПОКАЗЫВАЕТ ЧТО ВХОДНАЯ ЗАПИСЬ
           НЕ ЗАПОЛНЕНА
             ТО ВЫЗОВ: ДОБАВЛЕНИЕ К ВХОДНОЙ ЗАПИСИ
             (СИМВОЛ;)
             ИНАЧЕ ВЫЗОВ: ЗУММЕР (;)
         КОНЕЦ
    ЕСЛИ ЗНАК СОДЕРЖИТ КОД ОКОНЧАНИЯ
      ТО ВОЗВРАТ
    ЕСЛИ ЗНАК СОДЕРЖИТ ЛЮБОЙ ДРУГОЙ КОД
      ТО ВЫЗОВ: ЗУММЕР (;)
  КОНЕЦ
КОНЕЦ ПРОЦЕДУРЫ
```

Рис. 4.19. Процедура ЗАПОЛНЕНИЯ ВХОДНОЙ ЗАПИСИ.

вольные коды запоминаются во ВХОДНОЙ ЗАПИСИ. Если в какой-либо момент времени нажимается клавиша СТИРАНИЕ ВХОДА, структура данных ВХОДНАЯ ЗАПИСЬ стирается и входная последовательность начинается вновь. При нажатии клавиши окончания операции процедуры заканчиваются и информация, считанная в структуру данных, передается вызывающей процедуре через выходной параметр ВХОДНАЯ ЗАПИСЬ.

Процедура ЗАПОЛНЕНИЯ ВХОДНОЙ ЗАПИСИ вызывает несколько процедур. Процедура СЧИТЫВАНИЯ КЛАВИАТУРЫ находится во ВХОДНОМ модуле. Когда она вызывается, то ждет нажатия клавиш и передает символьный код, соответствующий нажатой клавише через выходной параметр СИМВОЛ. Процедура ЗУММЕР входит в модуль ВЫХОД. Она включает звуковой сигнал, чтобы информировать оператора о возникновении ошибок. В нашем случае звуковой сигнал раздается, когда количество символов в цифровой последовательности, введенной оператором, превышает максимально разрешенное число символов. Кроме того, звуковой сигнал генерируется, когда нажимается любая клавиша, кроме ЦИФРОВОЙ клавиши и клавиши ОКОНЧАНИЯ или СТИРАНИЯ ВХОДА. Оставшиеся процедуры, которые вызываются процедурой

ЗАПОЛНЕНИЯ ВХОДНОЙ ЗАПИСИ, — процедуры СТИРАНИЯ ВХОДНОЙ ЗАПИСИ, ДОБАВЛЕНИЯ К ВХОДНОЙ ЗАПИСИ и ПРОВЕРКИ ЗАПОЛНЕНИЯ ВХОДНОЙ ЗАПИСИ — принадлежат модулю ПОДДЕРЖКИ ВХОДНОЙ ЗАПИСИ.

Процедура СТИРАНИЯ ВХОДНОЙ ЗАПИСИ показана на рис. 4. 20. Она устанавливает каждый элемент структуры данных ВХОДНАЯ ЗАПИСЬ в символьный код *нуля,* тем самым стирая информацию, записанную в структуру данных. Она также устанавливает параметр ИНДИКАТОР ЗАПОЛНЕНИЯ в положение *«пусто»*. Этот параметр используется процедурой ПРОВЕРКИ ЗАПОЛНЕНИЯ ВХОДНОЙ ЗАПИСИ для опре-

```
ПРОЦЕДУРА: СТИРАНИЕ ВХОДНОЙ ЗАПИСИ (;)
НАЧАЛО ПРОЦЕДУРЫ
  ВЫПОЛНИТЬ ДЛЯ КАЖДОГО ЭЛЕМЕНТА ВО ВХОДНОЙ ЗАПИСИ
    УСТАНОВИТЬ ЗНАЧЕНИЕ ЭЛЕМЕНТА В КОД "НУЛЯ"
  КОНЕЦ
  СБРОСИТЬ ИНДИКАТОР ЗАПОЛНЕНИЯ
  ВОЗВРАТ
КОНЕЦ ПРОЦЕДУРЫ
```

• Рис. 4.20. Процедура СТИРАНИЯ ВХОДНОЙ ЗАПИСИ.

деления заполнения структуры данных ВХОДНАЯ ЗАПИСЬ. Число элементов, предусмотренных в структуре данных ВХОДНАЯ ЗАПИСЬ для запоминания символьных кодов, определяется числом знаков в наиболее длинной последовательности, которая может быть введена оператором. Нет необходимости в определении этого максимального числа во время ранних стадий проектирования. Операция языка проектирования ВЫПОЛНИТЬ ДЛЯ КАЖДОГО ЭЛЕМЕНТА ВО ВХОДНОЙ ЗАПИСИ позволяет спроектировать процедуру СТИРАНИЯ ВХОДНОЙ ЗАПИСИ без информации о размерах структуры данных.

Процедура ДОБАВЛЕНИЯ К ВХОДНОЙ ЗАПИСИ показана на рис. 4.21. Она перемещает влево каждый символьный код в структуре данных ВХОДНАЯ ЗАПИСЬ, вставляет новый символьный код (из входного параметра СИМВОЛ) на освободившееся место в правой части структуры данных и увеличивает значение ИНДИКАТОРА ЗАПОЛНЕНИЯ на единицу. Рис. 4.22 иллюстрирует процесс обработки символов в структуре данных процедурой ДОБАВЛЕНИЯ СИМВОЛА К ВХОДНОЙ ЗАПИСИ.

Процедура ПРОВЕРКИ ЗАПОЛНЕНИЯ ВХОДНОЙ ЗАПИСИ показана на рис. 4.23. Она проверяет значение ИНДИКАТОРА ЗАПОЛНЕНИЯ и устанавливает выходной параметр СТАТУС в состояние ЗАПОЛНЕН или НЕЗАПОЛНЕН,

```
ПРОЦЕДУРА: ДОБАВЛЕНИЕ К ВХОДНОЙ ЗАПИСИ (СИМВОЛ;)
НАЧАЛО ПРОЦЕДУРЫ
  СДВИНУТЬ КОДЫ ВХОДНОЙ ЗАПИСИ ВЛЕВО
  УСТАНОВИТЬ КРАЙНИЙ ПРАВЫЙ ЭЛЕМЕНТ ВХОДНОЙ ЗАПИСИ
  В ЗНАЧЕНИЕ СИМВОЛА
  УВЕЛИЧИТЬ ЗНАЧЕНИЕ ИНДИКАТОРА ЗАПОЛНЕНИЯ НА ЕДИНИЦУ
  ВОЗВРАТ
КОНЕЦ ПРОЦЕДУРЫ
```

Рис. 4.21. Процедура ДОБАВЛЕНИЯ К ВХОДНОЙ ЗАПИСИ.

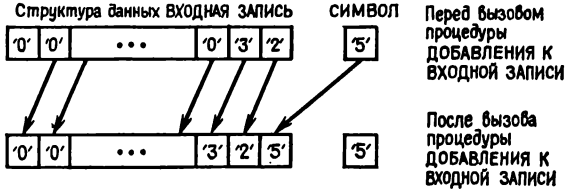


Рис. 4.22. Операции процедуры ДОБАВЛЕНИЯ К ВХОДНОЙ ЗАПИСИ.

```
ПРОЦЕДУРА: ПРОВЕРКА ЗАПОЛНЕНИЯ ВХОДНОЙ ЗАПИСИ
(; СТАТУС)
НАЧАЛО ПРОЦЕДУРЫ
  ЕСЛИ ИНДИКАТОР ЗАПОЛНЕНИЯ ПОКАЗЫВАЕТ ЧТО ВХОДНАЯ
  ЗАПИСЬ ЗАПОЛНЕНА
    ТО ПЕРЕВЕСТИ СТАТУС В СОСТОЯНИЕ "ЗАПОЛНЕНО"
    ИНАЧЕ ПЕРЕВЕСТИ СТАТУС В СОСТОЯНИЕ "НЕЗАПОЛНЕНО"
  ВОЗВРАТ
КОНЕЦ ПРОЦЕДУРЫ
```

Рис. 4.23. Процедура ПРОВЕРКИ ЗАПОЛНЕНИЯ ВХОДНОЙ ЗАПИСИ.

Как уже отмечалось, структура данных может состоять из большого количества связанной информации. Хотя информация может храниться в одном списке или группе списков, естественную организацию данных установить не всегда просто. Однако если информация организована в *иерархическую структуру данных,* то связь между данными становится очевидной. Проиллюстрируем эту концепцию на простом примере.

Предположим, что структура данных, которую назовем ФАЙЛ, состоит из набора записей. Пусть каждая запись состоит из порядкового номера, имени и домашнего адреса. Связь между частями этой структуры данных имеет следующий вид:

```
ФАЙЛ
  ЗАПИСЬ
    ПОРЯДКОВЫЙ НОМЕР
    ИМЯ
    ДОМАШНИЙ АДРЕС
```

Если какая-либо часть записи состоит из нескольких объектов, деление может быть продолжено. Например, если мы разобьём каждое имя на имя, отчество (инициал) и фамилию и каждый адрес на номер дома и название улицы, то иерархическая связь для структуры данных ФАЙЛ будет выглядеть следующим образом·

```
ФАЙЛ
  ЗАПИСЬ
    ПОРЯДКОВЫЙ НОМЕР
    ПОЛНОЕ ИМЯ
      ИМЯ
      ОТЧЕСТВО (ИНИЦИАЛ)
      ФАМИЛИЯ
    ДОМАШНИЙ АДРЕС
      НОМЕР ДОМА
      НАЗВАНИЕ УЛИЦЫ
```

Таким образом, организация данных в иерархической структуре данных легкопредставима с помощью набора идентифицирующих имен. Эти имена показывают уровни информации, содержащиеся в структуре данных.

Для завершения описания структуры данных необходимо добавить *определение структуры данных*, содержащее проектную информацию для каждой части структуры данных. Пример определения структуры данных для структуры данных ФАЙЛ приведен ниже:

МОДУЛЬНАЯ СТРУКТУРА ДАННЫХ: ФАЙЛ

ИМЯ	РАЗМЕР	ТИП	СОДЕРЖАНИЕ	ПРИМЕЧАНИЕ
ФАЙЛ				1
ЗАПИСЬ				
ПОРЯДКОВЫЙ НОМЕР	10	БАЙТ	ЦИФРОВЫЕ КОДЫ	
ПОЛНОЕ ИМЯ				
ИМЯ	10	БАЙТ	АЛФАВИТНЫЕ КОДЫ	
ОТЧЕСТВО (ИНИЦИАЛ)	1	БАЙТ	ТО ЖЕ	
ФАМИЛИЯ	14	БАЙТ	"	
ДОМАШНИЙ АДРЕС				
НОМЕР ДОМА	5	БАЙТ	ЧИСЛОВЫЕ КОДЫ	
НАЗВАНИЕ УЛИЦЫ	20	БАЙТ	АЛФАВИТНЫЕ КОДЫ	

ПРИМЕЧАНИЕ 1: ФАЙЛ СОСТОИТ ИЗ 256 ЗАПИСЕЙ

Хотя, как было показано ранее, часть информации в определении структуры данных может быть опущена, мы показали ее для полноты картины. Заголовок МОДУЛЬНАЯ СТРУКТУРА ДАННЫХ определяет ФАЙЛ как *главную* структуру данных для модуля. Аналогичные заголовки используются для определения структур данных и их использования в качестве входных и выходных параметров или запоминания *локальных* данных внутри процедуры. Доступ к локальным данным может быть осуществлен только внутри процедуры, которая содержит локальную структуру данных. РАЗМЕР определяет, сколько элементарных данных, в данном случае — сколько символьных кодов, может быть заложено в каждой ячейке структуры данных. Заметим, что размер показан только для нижних уровней иерархии структуры данных, так как данные хранятся именно на этих уровнях. ТИП определяет реализацию структуры данных на языке программирования. Для наглядности мы использовали БАЙТ, который является обозначением ТИПА для языка программирования PL/M. СОДЕРЖАНИЕ обозначает, какая информация запоминается в каждой ячейке структуры данных. ПРИМЕЧАНИЕ содержит дополнительную проектную информацию о каждой части структуры данных, которую надо включить в определение структуры данных. Примечания расположены после проектной информации для структур данных и обозначаются номером. При рассмотрении второго уровня документации в конце этой главы будет показано, как определение структур данных взаимосвязано с документацией.

С целью облегчения поиска информации в иерархической структуре данных желательно, чтобы данные были записаны или в алфавитном, или в цифровом порядке по отношению к некоторой *ключевой* информации в структуре данных. В предыдущем примере порядковый номер является примером *цифрового ключа,* фамилия — примером *алфавитного ключа.* В такой системе после каждой коррекции структура данных должна быть отсортирована с целью поддержания правильного порядка среди записей данных. *Сортировка* и *поиск* структур данных будут подробнее рассмотрены в следующих двух разделах.

4.19. Сортировка структур данных

Для сортировки структур данных разработано много методов. Предлагаемый вариант использует простой алгоритм для переупорядочения набора записей в последовательности, соответствующей возрастающей числовой последовательности. Предположим, что записи содержатся в структуре данных, называемой ФАЙЛ, введенной в предыдущем разделе, и что записи будут сортироваться по числовому признаку, используя в качестве

ключа порядковый номер. Термин КЛЮЧ используется в этом примере в общем смысле: пусть КЛЮЧ означает порядковый номер. Рис. 4.24 иллюстрирует процедуру СОРТИРОВКА, которая работает следующим образом. Выбирается первая запись в ФАЙЛЕ. Затем ключ в выбранной записи сравнивается с ключом в каждой из других записей ФАЙЛА. Если находится запись, в которой значения ключа меньше, чем в выбранной записи, эти

```
ПРОЦЕДУРА: СОРТИРОВКА (;)
НАЧАЛО ПРОЦЕДУРЫ
  ВЫБРАТЬ ПЕРВУЮ ЗАПИСЬ В ФАЙЛЕ
  ВЫПОЛНЯТЬ НЕПРЕРЫВНО
    ВЫПОЛНИТЬ ДЛЯ КАЖДОЙ ЗАПИСИ В ФАЙЛЕ СЛЕДУЮЩЕЙ
    ЗА ВЫБРАННОЙ ЗАПИСЬЮ
      ЕСЛИ КЛЮЧ ЗАПИСИ МЕНЬШЕ КЛЮЧА ВЫБРАННОЙ ЗАПИСИ
      ТО ПОМЕНЯТЬ МЕСТАМИ ЗАПИСЬ И ВЫБРАННУЮ ЗАПИСЬ
      И ПРОДОЛЖИТЬ ИСПОЛЬЗУЯ ЗАМЕНЕННУЮ ЗАПИСЬ
    КОНЕЦ
    ЕСЛИ ВЫБРАНЫ ВСЕ ЗАПИСИ КРОМЕ ПОСЛЕДНЕЙ
      ТО ВОЗВРАТ
      ИНАЧЕ ВЫБРАТЬ СЛЕДУЮЩУЮ ЗАПИСЬ
  КОНЕЦ
КОНЕЦ ПРОЦЕДУРЫ
```

Рис. 4.24. Сортировка структуры данных.

две записи меняются местами. Сравнение продолжается с использованием значения ключа замененной записи. После сравнения всех записей по одному разу первая запись в ФАЙЛЕ наверняка находится в правильном положении. Этот же процесс повторяется для следующей и каждой последующей записи. Когда же останется только одна запись, записи в структуре данных будут расположены в правильной числовой последовательности в соответствии с выбранным ключом (порядковым номером).

4.20. Поиск структур данных

Если записи в структуре данных находятся в случайном порядке, единственным путем для нахождения нужной информации является последовательный просмотр записей до нахождения искомой. Хотя эта технология может быть использована и для упорядоченных структур данных, для них существуют более эффективные методы. На рис. 4.25 показана процедура ПОИСК, которая хорошо работает во многих случаях.

Так как записи в структуре данных ФАЙЛ упорядочены в возрастающей числовой последовательности в соответствии с

порядковым номером, рассмотрим сначала запись в середине структуры данных. Если ключ поиска соответствует ключу средней записи, значит задача выполнена. Если ключ поиска меньше чем ключ средней записи, то мы можем исключить из дальнейшего рассмотрения все записи, следующие за данной в этой структуре данных. Если ключ поиска больше, чем ключ

```
ПРОЦЕДУРА: ПОИСК (КЛЮЧ ПОИСКА; ЗАПИСЬ)
НАЧАЛО ПРОЦЕДУРЫ
  ВЫБРАТЬ ЗАПИСЬ В СЕРЕДИНЕ ФАЙЛА
  ВЫПОЛНЯТЬ НЕПРЕРЫВНО
    ЕСЛИ КЛЮЧ ПОИСКА РАВЕН КЛЮЧУ ВЫБРАННОЙ ЗАПИСИ
      ТО      ВЫПОЛНИТЬ
                УСТАНОВИТЬ ЗНАЧЕНИЕ ЗАПИСИ РАВНЫМ
                ЗНАЧЕНИЮ ВЫБРАННОЙ ЗАПИСИ
                ВОЗВРАТ
                КОНЕЦ
    ЕСЛИ КЛЮЧ ПОИСКА МЕНЬШЕ ЧЕМ КЛЮЧ ВЫБРАННОЙ ЗАПИСИ
      ТО      ПРОПУСТИТЬ ЗАПИСИ СЛЕДУЮЩИЕ ЗА ВЫБРАННОЙ
              ЗАПИСЬЮ
      ИНАЧЕ   ПРОПУСТИТЬ ЗАПИСИ ПРЕДШЕСТВУЮЩИЕ
              ВЫБРАННОЙ ЗАПИСИ
    ЕСЛИ В ФАЙЛЕ ОСТАЛИСЬ ЗАПИСИ
      ТО      ВЫБРАТЬ ЗАПИСЬ В СЕРЕДИНЕ ОСТАВШИХСЯ
              ЗАПИСЕЙ
      ИНАЧЕ   ВЫПОЛНИТЬ
                УСТАНОВИТЬ ЗАПИСЬ В СОСТОЯНИЕ
                "СООТВЕТСТВИЕ НЕ НАЙДЕНО"
                ВОЗВРАТ
              КОНЕЦ
  КОНЕЦ
КОНЕЦ ПРОЦЕДУРЫ
```

Рис. 4.25. Поиск в упорядоченной структуре данных.

средней записи, можно исключить из дальнейшего рассмотрения все записи, предшествовавшие данной. Таким образом, даже при отсутствии соответствия в данной точке нет необходимости исследовать половину записей в структуре данных. Проверив среднюю запись из оставшихся, мы можем исключить из рассмотрения половину оставшихся записей. Продолжая этот процесс, мы в конце концов обнаружим либо наличие соответствия, либо его отсутствие, затратив при этом не более $\log_2 n$ шагов вместо $n/2$ шагов, требуемых при прямом поиске. Если, например, структура данных содержит 2000 записей, то для нахождения нужной записи по данной технологии необходимо не более 11 шагов, при прямом же поиске понадобится

не менее 1000 шагов. Конечно, при оценке времени поиска нужно учитывать время сортировки, необходимое для упорядочения структуры данных. Если между двумя сортировками осуществляется много поисковых операций, описанная методика заслуживает внимания. Иначе лучше подвергнуть прямому поиску неупорядоченную структуру данных. Если используется алфавитный ключ, то фразы «*больше чем*» и «*меньше чем*» в процедурах СОРТИРОВКА и ПОИСК могут быть заменены фразами «*алфавитно следует*» или «*алфавитно предшествует*» соответственно.

При рассмотрении системы охранной сигнализации не учитывались детали работы модуля ТАЙМЕР. Одним из возможных способов его реализации является использование механизма *прерывания*. Рассмотрим концепцию прерывания и то, как она влияет на проектирование системы.

4.21. Прерывание

Рассмотрим следующую ситуацию: мы решаем некоторую задачу и знаем, что периодически должны прерываться для того, чтобы выполнить другую работу. Существуют два пути, позволяющие обеспечить своевременное выполнение последней работы. Либо мы должны постоянно наблюдать за появлением некоторых внешних сигналов, которые указывают на то, что необходимо переключиться на другую работу, либо мы просим, чтобы нас *прервали*, когда появятся эти сигналы. В первом случае мы постоянно напрягаем наше внимание, наблюдая за появлением сигналов, рискуя при этом пропустить их появление, будучи, например, в глубокой задумчивости. В последнем случае мы можем спокойно сосредоточиться на решении задачи, так как уверены в том, что своевременно будем прерваны для выполнения другой работы. Таким образом, способ внешнего прерывания будет более эффективным потому, что позволяет рациональнее использовать наше время. Аналогичная ситуация возникает в микрокомпьютере. Проиллюстрируем эту концепцию на примере механизма таймера в системе охранной сигнализации.

На рис. 3.6 модуль ГЕНЕРАТОРА ТАКТОВЫХ ИМПУЛЬСОВ показан в части аппаратного обеспечения системы. Этот модуль генерирует периодический сигнал, используя сетевую линию или кварцевый осциллятор в качестве источника импульсов. В программном обеспечении микрокомпьютера можно предусмотреть механизм уменьшения на единицу числа, представляемого параметром ЗНАЧЕНИЕ ТАЙМЕРА, каждый раз, когда поступает тактовый импульс. Таким образом, если тактовый импульс генерируется каждую одну шестидесятую секунды, то при установке таймера для отсчета 5 с ЗНАЧЕНИЕ ТАЙ-

МЕРА должно быть установлено на 300, а после прохождения пяти секунд должно быть равно нулю. Микрокомпьютер может определить, когда нужно уменьшить ЗНАЧЕНИЕ ТАЙМЕРА, либо ожидая с помощью циклической проверки появления тактового импульса, либо используя специальный прерывающий механизм, который работает аналогично описанному выше примеру с участием человека.

Рассматривая процедуры ОБНАРУЖЕНИЯ НАРУШИТЕЛЯ или ПРОВЕРКИ ПРОДОЛЖЕНИЯ ДВИЖЕНИЯ, мы видим, что в них не предусмотрен механизм *слежения* за появлением тактового импульса. Для такой простой системы, какой является система охранной сигнализации, эта возможность может быть предоставлена путем добавления операции:

ВЫЗОВ: ПРОВЕРКА ТАКТОВОГО ИМПУЛЬСА (;)

в каждую процедуру так, чтобы она предшествовала операции

ВЫЗОВ: СЧИТЫВАНИЕ СОСТОЯНИЯ ТАЙМЕРА (; СОСТОЯНИЕ ТАЙМЕРА):

Однако в более сложных системах этот подход потребует добавления многих таких операций для того, чтобы обеспечить уменьшение ЗНАЧЕНИЯ ТАЙМЕРА при каждом появлении тактового импульса. Кроме того, если необходим быстрый отклик на появление тактового импульса (в отличие от нашего примера), он не может быть получен до тех пор, пока операция ПРОВЕРКА ТАКТОВОГО ИМПУЛЬСА не будет повторена многократно. Возрастание сложности системы вместе с потенциальным ухудшением времени отклика, характеризующие такое проектное решение, приводят к необходимости использования механизма прерывания.

Подробнее вопросы *прерывания* микрокомпьютера с помощью тактового импульса будут рассмотрены в гл. 6. Сейчас же достаточно понять, что, как только появляется тактовый импульс, выполняемая операция прерывается и имеет место *ВЫЗОВ* процедуры ИСПОЛНЕНИЯ ПРЕРЫВАНИЯ. Затем, после уменьшения ЗНАЧЕНИЯ ТАЙМЕРА на единицу, операция ВОЗВРАТ возобновляет прерванную операцию и продолжается естественная последовательность операций, как будто она и не была прервана. Процедуры, необходимые для выполнения прерывания в системе охранной сигнализации, показаны на рис. 4.26. Добавлен исполнительный модуль второго уровня, или модуль ИСПОЛНЕНИЯ ПРЕРЫВАНИЯ, а также модуль УПРАВЛЕНИЯ ПРЕРЫВАНИЕМ. Еще одно изменение необходимо в модуле ТАЙМЕРА, где добавляется процедура ОБРАБОТКИ ПРЕРЫВАНИЯ ТАЙМЕРА.

Теперь могут быть разработаны процедуры модуля ТАЙМЕР. Они показаны на рис. 4.27—4.30. Процедура ИСПОЛНЕНИЯ

ПРЕРЫВАНИЯ показана на рис. 4.31. Операции ЗАПОМНИТЬ СОСТОЯНИЕ СИСТЕМЫ и ВОССТАНОВИТЬ СОСТОЯНИЕ СИСТЕМЫ в процедуре ИСПОЛНЕНИЯ ПРЕРЫВАНИЯ необходимы для того, чтобы операции процедуры ИСПОЛНЕНИЯ ПРЕРЫВАНИЯ и вызываемые ею процедуры не влияли на выполнение прерванных операций. Детально это утверждение будет рассмотрено в гл. 6.

Рис. 4.26. Частичная модульная структура системы охранной сигнализации.

В системе, управляемой прерываниями, можно выделить несколько важных аспектов, вытекающих из модульной структуры системы охранной сигнализации.

• Для организации прерываний в системе строится отдельное дерево вызова процедур. Для системы охранной сигнализации оно показано на рис. 4.32.

• Если какая-либо процедура (например ОСТАНОВКА ТАЙМЕРА) вызывается из процедур в обоих деревьях, она может появиться в обоих деревьях вызова процедур.

• Информация передается из одной части системы в другую через такие параметры, как ЗНАЧЕНИЕ ТАЙМЕРА и СОСТОЯНИЕ ТАЙМЕРА, которые обрабатываются в обоих частях системы.

Во время работы системы тактовый импульс не вызывает прерываний до вызова процедуры ПУСК ТАЙМЕРА, которая устанавливает ЗНАЧЕНИЕ ТАЙМЕРА в необходимую величину и вызывает процедуру РАЗРЕШЕНИЯ ПРЕРЫВАНИЯ. Эта процедура запускает механизм прерывания, при этом тактовый импульс прерывает микрокомпьютер каждую 1/60 с. Во время прерывания запоминается состояние системы, снова разрешается прерывание, поскольку оно было автоматически запрещено в момент возникновения прерывания, ЗНАЧЕНИЕ ТАЙМЕРА уменьшается на единицу, состояние системы восстанавливается и система возвращается к прерванной операции.

```
ПРОЦЕДУРА: ЗАПУСК ТАЙМЕРА (СЕКУНДЫ;)
НАЧАЛО ПРОЦЕДУРЫ
  УСТАНОВИТЬ ЗНАЧЕНИЕ ТАЙМЕРА В 60 СЕКУНД
  УСТАНОВИТЬ ТАЙМЕР В ВОЗБУЖДЕННОЕ СОСТОЯНИЕ
  ВЫЗОВ: РАЗРЕШЕНИЕ ПРЕРЫВАНИЯ (;)
  ВОЗВРАТ
КОНЕЦ ПРОЦЕДУРЫ
```

Рис. 4.27. Процедура ЗАПУСКА ТАЙМЕРА.

```
ПРОЦЕДУРА: ОСТАНОВКА ТАЙМЕРА (;)
НАЧАЛО ПРОЦЕДУРЫ
  ВЫЗОВ: ЗАПРЕЩЕНИЕ ПРЕРЫВАНИЯ (;)
  ВОЗВРАТ
КОНЕЦ ПРОЦЕДУРЫ
```

Рис. 4.28. Процедура ОСТАНОВКИ ТАЙМЕРА.

```
ПРОЦЕДУРА: СЧИТЫВАНИЕ СОСТОЯНИЯ ТАЙМЕРА (; СОСТОЯНИЕ
ТАЙМЕРА)
НАЧАЛО ПРОЦЕДУРЫ
  ПОЛУЧИТЬ ЗНАЧЕНИЕ СОСТОЯНИЯ ТАЙМЕРА
  ВОЗВРАТ
КОНЕЦ ПРОЦЕДУРЫ
```

Рис. 4.29. Процедура СЧИТЫВАНИЯ СОСТОЯНИЯ ТАЙМЕРА.

```
ПРОЦЕДУРА: ОБРАБОТКА ПРЕРЫВАНИЯ ТАЙМЕРА (;)
НАЧАЛО ПРОЦЕДУРЫ
  УМЕНЬШЕНИЕ ЗНАЧЕНИЯ ТАЙМЕРА НА ЕДИНИЦУ
  ЕСЛИ ЗНАЧЕНИЕ ТАЙМЕРА РАВНО НУЛЮ
     ТО ВЫПОЛНИТЬ
              ВЫЗОВ: ОСТАНОВКА ТАЙМЕРА (;)
              УСТАНОВИТЬ ТАЙМЕР В СОСТОЯНИЕ ПОКОЯ
        КОНЕЦ
  ВОЗВРАТ
КОНЕЦ ПРОЦЕДУРЫ
```

Рис. 4.30. Процедура ОБРАБОТКИ ПРЕРЫВАНИЯ ТАЙМЕРА.

```
ПРОЦЕДУРА: ИСПОЛНЕНИЕ ПРЕРЫВАНИЯ (;)
НАЧАЛО ПРОЦЕДУРЫ
  ЗАПОМНИТЬ СОСТОЯНИЕ СИСТЕМЫ
  ВЫЗОВ: РАЗРЕШЕНИЕ ПРЕРЫВАНИЯ (;)
  ВЫЗОВ: ОБРАБОТКА ПРЕРЫВАНИЯ ТАЙМЕРА (;)
  ВОССТАНОВИТЬ СОСТОЯНИЕ СИСТЕМЫ
  ВОЗВРАТ
КОНЕЦ ПРОЦЕДУРЫ
```

Рис. 4.31. Процедура ИСПОЛНЕНИЯ ПРЕРЫВАНИЯ.

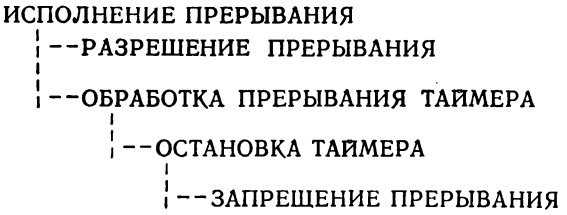


Рис. 4.32. Дерево вызова процедур прерывания в системе охранной сигнализации.

Если значение таймера становится равным нулю, то в дополнение к вышеперечисленным операциям прерывание от тактового импульса запрещается и устанавливается СОСТОЯНИЕ ТАЙМЕРА, указывающее, что требуемый интервал времени истек. Прерывание запрещается до тех пор, пока опять не будет разрешено процедурой ПУСК ТАЙМЕРА.

Необходимо сделать еще одно примечание в отношении концепции прерывания. Так как процедура ОСТАНОВКИ ТАЙМЕРа вызывается в обоих деревьях вызова, возможно возникновением следующей ситуации:

• Процедура ОСТАНОВКИ ТАЙМЕРА вызывается либо из процедуры ВОССТАНОВЛЕНИЯ СИСТЕМЫ, либо ПРОВЕРКИ ПРОДОЛЖЕНИЯ ДВИЖЕНИЯ.

• Во время выполнения процедуры ОСТАНОВКИ ТАЙМЕРА до начала операции ЗАПРЕЩЕНИЯ ПРЕРЫВАНИЯ появляется прерывание от тактового импульса и в ответ на него вызывается программа обработки прерывания (ИСПОЛНЕНИЯ ПРЕРЫВАНИЯ).

• Во время выполнения процедуры ОБРАБОТКИ ПРЕРЫВАНИЯ ТАЙМЕРА, которая вызывается процедурой ИСПОЛНЕНИЯ ПРЕРЫВАНИЯ, снова вызывается процедура ОСТАНОВКИ ТАЙМЕРА.

Это означает, что процедура может быть вызвана в то время, когда ее выполнение прервано. Эта ситуация разрешима, если при конструировании программного обеспечения приняты некоторые предосторожности. Эта концепция подробнее будет рассмотрена в гл. 6. Сейчас мы хотим только отметить, что такая ситуация в микрокомпьютерной системе не представляет каких-либо особых трудностей, тем не менее мы должны быть готовы к ее появлению.

Этим завершается описание прерывания с точки зрения проектирования программного обеспечения. Еще раз прерывание будет рассмотрено с точки зрения высокоуровневого языка программирования в гл. 5 и с точки зрения архитектуры и языка ассемблера в гл. 6.

4.22. Второй уровень документации

Первый уровень документации состоит из требований пользователя и функциональных спецификаций. Функциональные спецификации могут состоять из руководств пользователя. Второй уровень документации включает проектные спецификации программного обеспечения, описанные в гл. 3, а также описание процедур, модулей и подсистем на языке проектирования.

```
ПРОЦЕДУРА: ПРОВЕРКА ПРОДОЛЖЕНИЯ ДВИЖЕНИЯ
(; ПРОДОЛЖЕНИЕ)
НАЧАЛО ПРОЦЕДУРЫ
ВЫЗОВ: ЗАПУСК ТАЙМЕРА (5 СЕКУНД;)
ВЫПОЛНЯТЬ НЕПРЕРЫВНО
ВЫЗОВ: СЧИТЫВАНИЕ ДЕТЕКТОРА ДВИЖЕНИЯ (; ДЕТЕКТОР
ДВИЖЕНИЯ)
ЕСЛИ ДЕТЕКТОР ДВИЖЕНИЯ СРАБОТАЛ
ТО ВЫПОЛНИТЬ
ВЫЗОВ: СЧИТЫВАНИЕ СОСТОЯНИЯ ТАЙМЕРА (; СОСТОЯНИЕ
ТАЙМЕРА)
ЕСЛИ ТАЙМЕР НАХОДИТСЯ В СОСТОЯНИИ ПОКОЯ
ТО ВЫПОЛНИТЬ
ВЫЗОВ: ОСТАНОВКА ТАЙМЕРА (;)
УСТАНОВИТЬ ПРОДОЛЖЕНИЕ
ВОЗВРАТ
КОНЕЦ
КОНЕЦ
ИНАЧЕ
ВЫЗОВ: ОСТАНОВКА ТАЙМЕРА (;)
СБРОСИТЬ ПРОДОЛЖЕНИЯ
ВОЗВРАТ
КОНЕЦ
КОНЕЦ
КОНЕЦ ПРОЦЕДУРЫ
```

Рис. 4.33. Процедура ПРОВЕРКИ ПРОДОЛЖЕНИЯ ДВИЖЕНИЯ, написанная без смещения строк.

Прежде чем завершить описание документации на уровне проектирования, рассмотрим концепцию *смещения* строк, которую мы уже использовали ранее.

В каждой рассматриваемой процедуре мы смещали операции одну относительно другой, чтобы показать их взаимосвязь. Необходимо подчеркнуть, что язык проектирования является полностью однозначным без смещения строк. Смещение обеспечивает читаемость, особенно когда несколько конструкций являются вложенными, и его использование желательно. На рис. 4.33 показана процедура ПРОВЕРКИ ПРОДОЛЖЕНИЯ

ДВИЖЕНИЯ системы охранной сигнализации, переписанная без смещения в ущерб наглядности и ясности.

Ниже приводятся *правила* выполнения смещения, иллюстрируемые примерами.

1. Все *скобки*, такие, как НАЧАЛО ПРОЦЕДУРЫ... КОНЕЦ ПРОЦЕДУРЫ и ВЫПОЛНИТЬ ... КОНЕЦ, выравниваются.

```
ВЫПОЛНЯТЬ ПОКА ПЕРЕКЛЮЧАТЕЛЬ НЕ ВКЛЮЧЕН
  ВЫЗОВ: СЧИТЫВАНИЕ ПЕРЕКЛЮЧАТЕЛЯ (; ПЕРЕКЛЮЧАТЕЛЬ)
КОНЕЦ
```

2. Все основные операции, содержащиеся внутри пары *скобок*, смещаются на одинаковое расстояние.

```
НАЧАЛО ПРОЦЕДУРЫ
  ВЫЗОВ: ИНИЦИАЛИЗАЦИЯ АППАРАТУРЫ (;)
  ВЫЗОВ: ВОССТАНОВЛЕНИЕ СИСТЕМЫ (;)
  ВОЗВРАТ
КОНЕЦ ПРОЦЕДУРЫ
```

3. Слова ТО и ИНАЧЕ условной конструкции располагаются на разных строках и смещаются по отношению к слову ЕСЛИ на одинаковое расстояние. Если внутри части ТО или ИНАЧЕ условной конструкции появляется отдельная операция, то она располагается на той же строке, что и слово ТО или ИНАЧЕ.

```
ЕСЛИ ВСЕ ЗАПИСИ КРОМЕ ПОСЛЕДНЕЙ ВЫБРАНЫ
  ТО ВОЗВРАТ
  ИНАЧЕ ВЫБИРАЕТСЯ СЛЕДУЮЩАЯ ЗАПИСЬ В ФАЙЛЕ
```

4. Если внутри части ТО или ИНАЧЕ условной конструкции содержится более одной операции, то эти операции следует заключить в *скобки* ВЫПОЛНИТЬ ... КОНЕЦ, причем слово ВЫПОЛНИТЬ располагается на той же строке, что и слово ТО или ИНАЧЕ.

```
ЕСЛИ ТАЙМЕР В СОСТОЯНИИ ПОКОЯ
  ТО ВЫПОЛНИТЬ
       ВЫЗОВ: ОСТАНОВКА ТАЙМЕРА (;)
       УСТАНОВИТЬ ПРОДОЛЖЕНИЕ
       ВОЗВРАТ
     КОНЕЦ
```

Когда процедуры языка проектирования записаны по этим правилам, тогда вместе с проектными спецификациями они почти полностью заменяют документацию. Однако на уровне процедур и модулей необходима следующая дополнительная информация:

- Идентификационные номера процедур и модулей.
- Имя проектировщика каждой процедуры и модуля.
- Дата проектирования процедуры или модуля.
- Имена всех, кто вносил изменения в проект.
- Даты внесения изменений в проект.
- Краткие сведения о том, что делают процедура или модуль, если имени процедуры или модуля недостаточно для этих целей.

```
PROCEDURE: RESET SYSTEM (;)                                    P003
************************************************************
  DESIGNED BY: M. D. FREEDMAN                                  9-10-80
  REVISED BY: L. B. EVANS                                      9-20-80
  FUNCTION: RESETS ALARMS AND STOPS TIMER
  MODULE: RESET
  PROCEDURES CALLED:  RESET ALARMS
                      STOP TIMER
  CALLED BY:  INITIALIZE SYSTEM
              INTRUDER DETECTED
************************************************************
BEGIN PROCEDURE
  CALL: RESET ALARMS (;)
  CALL: STOP TIMER (;)
  RETURN
END PROCEDURE
```

Рис. 4.34. Документированная версия процедуры ВОССТАНОВЛЕНИЯ СИСТЕМЫ.

- Имя модуля, которому принадлежит процедура.
- Имена всех процедур, которые вызывает данная процедура.
- Имена всех процедур, которые вызывают данную процедуру.
- Описание каждой структуры данных и параметров, которые обрабатываются данной процедурой.
- Пояснения о назначении каждого параметра в структуре данных, если это не ясно из контекста.

Эта информация может быть размещена между строкой идентификации процедуры и фразой НАЧАЛО ПРОЦЕДУРЫ. На рис. 4.34 показана полностью документированная версия процедуры ВОССТАНОВЛЕНИЯ СИСТЕМЫ, показанная ранее на рис. 4.3. Строки из звездочек используются для выделения области документации для удобства чтения и могут быть опущены.

Многие из позиций, включенных в список выше, требуются также и на модульном уровне. Поэтому *описание модуля* может рассматриваться как часть документации. Одной из важнейших

```
MODULE: MAINTAIN INPUT RECORD                                          M004
------------------------------------------------------------------------------
  DESIGNED BY: M. D. FREEDMAN                                          9-30-78
  PROCEDURES:  CLEAR INPUT RECORD
               ADD TO INPUT RECORD
               TEST IF INPUT RECORD FULL
               GET INPUT RECORD

  EXTERNAL PROCEDURES CALLED:  BEEP
                               READ KEYBOARD

  MODULAR DATA STRUCTURE:  INPUT RECORD

  NAME                  SIZE    TYPE    CONTENTS            NOTES
  ----                  ----    ----    --------            -----
  INPUT RECORD           10     BYTE    NUMERIC CODES
    FULL INDICATOR        1     BYTE    COUNTER
------------------------------------------------------------------------------
PROCEDURE:  CLEAR INPUT RECORD (;)
******************************************************************************
BEGIN PROCEDURE
  DO FOR EACH ELEMENT IN INPUT RECORD
    SET ELEMENT TO 'ZERO' CODE
  END
  SET FULL INDICATOR TO EMPTY
  RETURN
END PROCEDURE
------------------------------------------------------------------------------
PROCEDURE:  ADD TO INPUT RECORD (CHARACTER;)
******************************************************************************
  INPUT PARAMETER: CHARACTER

  NAME                  SIZE    TYPE    CONTENTS            NOTES
  ----                  ----    ----    --------            -----
  CHARACTER              1      BYTE    NUMERIC CODE
******************************************************************************
BEGIN PROCEDURE
  SHIFT INPUT RECORD CODES TO LEFT
  SET RIGHT-HAND INPUT RECORD ELEMENT TO CHARACTER
  INCREMENT FULL INDICATOR
  RETURN
END PROCEDURE
------------------------------------------------------------------------------
```

Рис. 4.35. Модульное описание модуля ОБРАБОТКИ ВХОДНОЙ ЗАПИСИ.

```
PROCEDURE:  TEST IF INPUT RECORD FULL (;STATUS)
*************************************************************************
  OUTPUT PARAMETER:  STATUS

  NAME                SIZE    TYPE    CONTENTS            NOTES
  ----                ----    ----    --------            -----
  STATUS               1      BYTE    FLAG
*************************************************************************
BEGIN PROCEDURE
  IF FULL INDICATOR SHOWS THAT INPUT RECORD IS FULL
    THEN  SET STATUS TO FULL
    ELSE  SET STATUS TO NOT FULL
  RETURN
END PROCEDURE
-------------------------------------------------------------------------
PROCEDURE:  GET INPUT RECORD (;INPUT RECORD)
*************************************************************************
  OUTPUT PARAMETER:  INPUT RECORD

  NAME                SIZE    TYPE    CONTENTS            NOTES
  ----                ----    ----    --------            -----
  INPUT RECORD         10     BYTE    NUMERIC CODES        1

  NOTE 1:  SEE MODULAR DATA STRUCTURE

  LOCAL PARAMETERS:  CHARACTER
                     STATUS

  NAME                SIZE    TYPE    CONTENTS            NOTES
  ----                ----    ----    --------            -----
  CHARACTER            1      BYTE    ALPHANUMERIC CODE
  STATUS               1      BYTE    FLAG
*************************************************************************
BEGIN PROCEDURE
  CALL:  CLEAR INPUT RECORD (;)
  DO FOREVER
    CALL:  READ KEYBOARD (;CHARACTER)
    IF CHARACTER CONTAINS 'CLEAR INPUT' CODE
      THEN  CALL:  CLEAR INPUT RECORD (;)
    IF CHARACTER CONTAINS A NUMERIC CODE
      THEN  DO
              CALL:  TEST IF INPUT RECORD FULL (;STATUS)
              IF STATUS INDICATES INPUT RECORD IS NOT FULL
                THEN  CALL:  ADD TO INPUT RECORD (CHARACTER;)
                ELSE  CALL:  BEEP (;)
            END
    IF CHARACTER CONTAINS A TERMINATOR CODE
      THEN  RETURN
    IF CHARACTER CONTAINS ANY OTHER CODE
      THEN  CALL:  BEEP (;)
  END
END PROCEDURE
-------------------------------------------------------------------------
END MODULE
```

Рис. 4.35. (Продолжение.)

позиций, включенных в описание модуля,— это определение структуры данных для каждой структуры данных, содержащейся в модуле. Так как модульные структуры данных могут обрабатываться любой процедурой, принадлежащей модулю, определения структур данных будут дублироваться в каждой процедуре, если они не содержатся в описании модуля. Пример описания модуля ОБРАБОТКИ ВХОДНОЙ ЗАПИСИ приведен на рис. 4.35. Заметим, что все процедуры, принадлежащие модулю, включены в описание модуля вслед за заголовком модуля.

ИСПОЛНИТЕЛЬНЫЙ	МОДУЛЬ
ИСПОЛНИТЕЛЬНАЯ	ПРОЦЕДУРА
ВХОДНОЙ	МОДУЛЬ
СЧИТЫВАНИЕ ПЕРЕКЛЮЧАТЕЛЯ	ПРОЦЕДУРА
ПЕРЕКЛЮЧАТЕЛЬ	ВЫХОДНОЙ ПАРАМЕТР
СЧИТЫВАНИЕ ДЕТЕКТОРА ДВИЖЕНИЯ	ПРОЦЕДУРА
ДЕТЕКТОР ДВИЖЕНИЯ	ВЫХОДНОЙ ПАРАМЕТР
СЧИТЫВАНИЕ КОНТАКТНЫХ ДЕТЕКТОРОВ	ПРОЦЕДУРА
КОНТАКТНЫЕ ДЕТЕКТОРЫ	ВЫХОДНОЙ ПАРАМЕТР
ОБНАРУЖЕНИЕ НАРУШИТЕЛЯ	МОДУЛЬ
ОБНАРУЖЕНИЕ НАРУШИТЕЛЯ	ПРОЦЕДУРА
ПРОВЕРКА	МОДУЛЬ
ПРОВЕРКА ПРОДОЛЖЕНИЯ ДВИЖЕНИЯ	ПРОЦЕДУРА
ПРОДОЛЖЕНИЕ	ВЫХОДНОЙ ПАРАМЕТР
ПРОВЕРКА ДЕТЕКТОРА ДВИЖЕНИЯ	ПРОЦЕДУРА
ПРОВЕРКА КОНТАКТНЫХ ДЕТЕКТОРОВ	ПРОЦЕДУРА

Рис. 4.36. Список имен системы охранной сигнализации.

Для простоты некоторая документация из заголовков *процедур* на рисунке опущена.

Читатель, наверное, уже заметил, что даже при проектировании простых систем вводится и используется большое количество имен. Каждая подсистема, модуль, процедура, структура данных и параметр имеют имя. Для того чтобы избежать путаницы с именами, полезно использовать *список имен*, в который вносится любое имя сразу после того, как оно определено. В этот список можно вносить и обозначение того, что представляет собой каждое имя, и тогда его можно использовать как *словарь* системы. Пример списка имен для системы охранной сигнализации показан на рис. 4.36. Если имя используется в двух разных смыслах, как, например, имя ИСПОЛНИТЕЛЬНАЯ, то оно включается в список дважды. Список имен

вместе с проектными спецификациями и описанием процедур на языке проектирования должен включаться во второй уровень документации.

Таким образом, полный набор документации второго уровня включает в себя:

- Иерархический список имен подсистем, модулей и процедур. Для этой цели может служить список имен, который включает также имена всех структур и параметров.
- Дерево вызова процедур.
- Определения структур данных для каждой структуры данных, используемых в системе.
- Описание каждой процедуры на языке проектирования.
- Дополнительную информацию, необходимую для понимания системы на уровне проектирования; эта информация может быть размещена либо в заголовке модуля или процедуры, либо, если необходимо, в отдельных документах.

4.23. Упражнения

4.1. Нужны ли структуры данных для телевизора с встроенным микрокомпьютером? Если да, спроектируйте эти структуры данных и, если необходимо, модифицируйте модульную структуру программного обеспечения из упражнения 3.2, включив в нее модули данных. Определите, какие новые процедуры необходимы для модифицированной модульной структуры.

4.2. Спроектируйте каждую процедуру в программных модулях телевизора с встроенным микрокомпьютером.

4.3. Нужны ли структуры данных для устройства управления уличным светофором с встроенным микрокомпьютером? Если да, спроектируйте их и, если необходимо, модифицируйте модульную структуру программного обеспечения из упражнения 3.4, включив в нее модули данных. Какие новые процедуры нужны для этих целей?

4.4. Спроектируйте каждую процедуру в программных модулях устройства управления уличным светофором с встроенным микрокомпьютером.

4.5. Необходимы ли структуры данных для электронного спортивного табло с микрокомпьютером? Если так, спроектируйте эти структуры и, если необходимо, модифицируйте модульную структуру программного обеспечения из упражнения 3.6, включив в нее модули данных. Какие новые процедуры необходимы для модификации модульной структуры?

4.6. Спроектируйте процедуры в программных модулях электронного спортивного табло с микрокомпьютером.

4.7. Спроектируйте структуру данных, необходимую для терминала розничной торговли с встроенным микрокомпьютером

из упражнения 3.8. Модифицируйте модульную структуру программного обеспечения, включив в нее модули данных. Какие новые процедуры необходимы для этих модулей?

4.8. Спроектируйте процедуру программных модулей терминала розничной торговли с встроенным микрокомпьютером.

4.9. Если Вы работаете над проектированием одной из этих систем группой, распределите каждый модуль между участниками группы и пусть каждый подготовит просмотр процедур своего модуля. Если система содержит структуру данных, подготовьте диаграммы потока данных для каждого просмотра.

4.10. Необходимо ли в вашей системе использование подсистемы прерывания? Если да, то почему? Если вы это не сделали, разработайте модульную структуру и спроектируйте подсистему прерывания.

4.11. Составьте дерево вызова процедур для той части системы, которую вы спроектировали.

Конвертирование описания программного обеспечения на языке проектирования в язык высокого уровня

К настоящему времени обсуждение вопросов разработки программного обеспечения на языке проектирования в первом приближении закончено. Язык проектирования понятен и может использоваться для документирования программ. Однако микрокомпьютеры не способны понять язык проектирования. Как мы видели, за исключением нескольких хорошо определенных конструкций, язык проектирования является естественным языком и не предназначен для микрокомпьютера. Необходимо транслировать описание системы на языке проектирования в *язык микрокомпьютера.* Полученный набор процедур на языке микрокомпьютера составляет программное обеспечение, которое придает микрокомпьютеру функциональные свойства, требуемые системой.

В этой главе и в гл. 6 и 7 мы исследуем различные типы языков, понятных микрокомпьютерам. Мы также опишем имеющиеся средства, облегчающие конвертирование программ на языке проектирования в программы на языке микрокомпьютера, а также позволяющие отыскивать и устранять ошибки в конечных программах.

5.1. Языки микрокомпьютера

Связь между различными языками, которые мы предполагаем обсудить, показана на рис. 5.1. На верхнем уровне находится язык проектирования, который был подробно рассмотрен в гл. 4. Описание программы на языке проектирования может быть транслировано либо в высокоуровневый *язык компилятора,* либо в *язык ассемблера* низкого уровня. В конечном счете мы должны получить версию программы на *машинном языке,* поскольку, как мы далее увидим, именно он является тем языком, который понимает микрокомпьютер. Как язык компилятора, так и язык ассемблера могут быть автоматически транслированы в машинный язык с помощью *средств,* называемых *компиляторами* и *ассемблерами.* Компилятор может транслировать либо в язык ассемблера, который требует дальнейшей

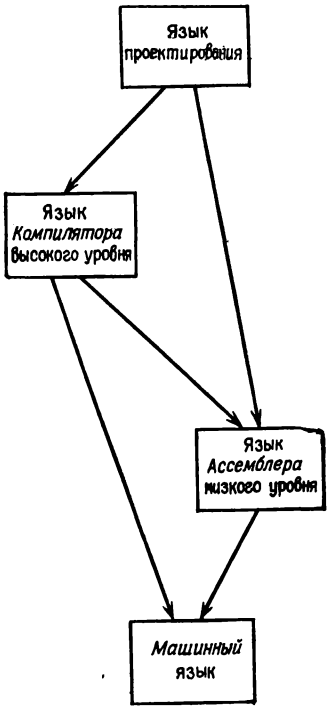


Рис. 5.1. Связь между различными типами языков.

трансляции, как показано на рис. 5.1, либо непосредственно в машинный язык.

Во многих случаях наиболее эффективным способом конвертирования является трансляция языка проектирования в высокоуровневый язык компилятора. Поскольку PL/M является примером языка компилятора, который специально предназначен для использования с микрокомпьютерами, мы опишем его в этой главе. Чтобы читатель имел возможность работать с доступным ему языком, в приложении Г предлагается введение в некоторые другие языки высокого уровня.

Данное описание не является руководством по программированию микрокомпьютера, так как оно является неполным. Однако предлагаемый нами материал является достаточным для того, чтобы читатель смог поработать с упражнениями и понять основные концепции. Для более детального ознакомления с вопросами программирования мы отсылаем читателя к списку литературы.

5.2. Язык PL/M

PL/M является подмножеством языка высокого уровня PL/I. PL/M разработан фирмой Intel Corporation как средство для программирования микрокомпьютеров фирмы Intel. Поэтому он содержит много свойств, позволяющих эффективно разрабатывать программное обеспечение для микрокомпьютера. В то же время он обладает свойствами, позволяющими разрабатывать программное обеспечение на системной модульной и нисходящей основе. В следующих разделах мы опишем свойства языка PL/M настолько детально, чтобы читатель смог конвертировать конструкции языка проектирования в операции PL/M.

5.3. Комментарии PL/M

Комментарии PL/M обозначаются скобками /* и */, так что комментарий в PL/M имеет следующий вид:

```
/*
                ДАННЫЙ ТЕКСТ ЯВЛЯЕТСЯ КОММЕНТАРИЕМ
                                                    */
```

и может занимать любое число строк. Комментарии игнорируются компилятором PL/M и распечатываются в выходном листинге без изменений.

Таким образом мы можем сохранить описание операций на языке проектирования, конвертируя их в комментарии PL/M. Операции PL/M, соответствующие операциям языка проектирования, могут располагаться на следующей строке после комментария. Реализация программы на языке PL/M вместе с комментариями на языке проектирования создает хорошую основу для документирования программ, которые становятся легкочитаемыми и удобными в работе.

Сначала мы опишем способы конвертирования каждой из конструкций языка проектирования, введенных в гл. 4, в соответствующие операции PL/M. Далее мы рассмотрим некоторые конструкции PL/M, которые не имеют аналога в языке проектирования, но которые необходимы для выполнения программ микрокомпьютером.

5.4. Конструкции присваивания

Примеры конструкций присваивания на языке проектирования следующие:

```
ПЕРЕВЕСТИ СТАТУС В СОСТОЯНИЕ «ЗАПОЛНЕНО»
ПЕРЕВЕСТИ СТАТУС В СОСТОЯНИЕ «НЕ ЗАПОЛНЕНО»
УСТАНОВИТЬ ПРОДОЛЖЕНИЕ
СБРОСИТЬ ПРОДОЛЖЕНИЕ
```

Однако в PL/M вместо терминов типа «ЗАПОЛНЕНО» должны использоваться специальные числовые или символьные значения. Пусть, например, нулевое значение СТАТУСА представляет состояние «НЕ ЗАПОЛНЕНО», ненулевое — «ЗАПОЛНЕНО». Тогда операция PL/M

```
STATUS = 0
```

будет означать, что STATUS устанавливается в нулевое значение, и поэтому она эквивалентна конструкции ПЕРЕВЕСТИ СТАТУС В СОСТОЯНИЕ «НЕ ЗАПОЛНЕНО». Аналогично

```
STATUS = 1,
```

означает, что STATUS устанавливается в единичное значение, и поэтому операция эквивалентна конструкции ПЕРЕВЕСТИ СТАТУС В СОСТОЯНИЕ «ЗАПОЛНЕНО».

Для конструкций УСТАНОВИТЬ и СБРОСИТЬ представим, что параметр PL/M *сброшен,* если его значение есть нуль, и *установлен,* если его значение есть единица. Тогда операция PL/M

```
CONTINUOUS = 0;
```

будет эквивалентна конструкции СБРОСИТЬ ПРОДОЛЖЕНИЕ, а операция

```
CONTINUOUS = 1;
```

эквивалентна конструкции УСТАНОВИТЬ ПРОДОЛЖЕНИЕ.

Отметим, что *точка с запятой* означает окончание операции PL/M. Точка с запятой используется в PL/M в качестве ограничителя, который позволяет однозначно отделить одну операцию от другой в случае, если операция занимает несколько строк. Поэтому для правильной интерпретации операций PL/M необходимо аккуратное использование точки с запятой. Такие объекты PL/M, как STATUS и CONTINUOUS, которым в разное время могут быть присвоены различные значения, называются *переменными.* Объекты типа 0 и 1, которые не изменяют своих значений, называются *константами.*

5.5. Конструкции цикла

В языке проектирования конструкциями цикла являются

```
ВЫПОЛНИТЬ ... КОНЕЦ
ВЫПОЛНЯТЬ ПОКА ... КОНЕЦ
ВЫПОЛНИТЬ ДЛЯ КАЖДОГО ... КОНЕЦ
ВЫПОЛНЯТЬ НЕПРЕРЫВНО ... КОНЕЦ
```

В языке PL/M аналогом ВЫПОЛНИТЬ ... КОНЕЦ является конструкция

```
DO;
  •
  •
  •
END;
```

Для конструкции ВЫПОЛНЯТЬ, ПОКА ... КОНЕЦ в PL/M имеется почти идентичная конструкция

```
DO WHILE выражение;
  •
  •
  •
END;
```

Выражение в PL/M используется вместо описательной фразы типа ПЕРЕКЛЮЧАТЕЛЬ ВЫКЛЮЧЕН. В данном случае выражение является *логическим выражением, значением* которого в любой момент времени является либо «истинно», либо «ложно». Примером логического выражения является фраза KEYSWITCH = 0 в операции

```
DO WHILE KEYSWITCH=0;
```

Мы предположили, что переменная KEYSWITCH равна нулевому значению, когда переключатель находится в сброшенном состоянии, и ненулевому, когда он находится во включенном состоянии. Логическое выражение KEYSWITCH = 0 является истинным, как только переключатель выключается, и поэтому операция PL/M эквивалентна конструкции языка проектирования

ВЫПОЛНЯТЬ ПОКА ПЕРЕКЛЮЧАТЕЛЬ ВЫКЛЮЧЕН

Мы ввели здесь понятие выражения PL/M с целью описания конструкций цикла. Более подробно выражения PL/M будут рассмотрены в последующих разделах данной главы.

Конструкция ВЫПОЛНИТЬ ДЛЯ КАЖДОГО ... КОНЕЦ может быть конвертирована в следующую конструкцию PL/M:

```
DO индекс=выражение-1 TO выражение-2;
  •
  •
  •
END;
```

Здесь описательная фраза заменяется *индексом* и двумя выражениями. Конструкция языка проектирования ВЫПОЛНИТЬ ДЛЯ КАЖДОГО ДЕТЕКТОРА в случае, когда имеется пять детекторов, может быть реализована с помощью операции PL/M следующим образом:

```
DO INDEX=1 TO 5;
```

В этом случае INDEX является переменной PL/M, которая сначала принимает значение *выражения-1*, являющегося в нашем примере константой 1. Затем выполняется цикл, значение переменной увеличивается на 1 и цикл выполняется снова до тех пор, пока значение переменной INDEX не станет равным значению *выражения-2*, которое является константой 5. Выполнение цикла заканчивается после того, как он будет выполнен ровно пять раз.

Один из вариантов этой конструкции PL/M позволяет при каждом выполнении цикла увеличивать индекс на величину, отличную от 1. Это осуществляется следующим образом:

```
DO индекс=выражение-1 TO выражение-2 BY выражение-3;
  •
  •
  •
END;
```

Выражение-3 содержит значение величины, на которую индекс получает приращение при каждом выполнении цикла. Например, в конструкции

```
DO INDEX = 1 TO 5 BY 2;
```

INDEX вначале устанавливается равным 1. Затем он принимает значение, равное 3, и наконец, 5. Поэтому цикл выполняется три раза, а не пять, как в предыдущем примере.

Конструкция ВЫПОЛНЯТЬ НЕПРЕРЫВНО ... КОНЕЦ не имеет аналога в PL/M. Однако она может быть реализована с использованием конструкции DO WHILE вместе с выражением, которое всегда истинно. Поскольку выражение 1 = 1 всегда истинно, конструкция PL/M

```
DO WHILE 1 = 1;
  •
  •
  •
END;
```

является эквивалентом конструкции языка проектирования ВЫПОЛНЯТЬ НЕПРЕРЫВНО ... КОНЕЦ.

5.6. Управляющие конструкции

В языке проектирования управляющими являются следующие конструкции

```
ВЫЗОВ:
ВОЗВРАТ
```

В PL/M им соответствуют конструкции:

```
CALL имя-процедуры;
RETURN;
```

Примерами этих конструкций в PL/M вместе с эквивалентными конструкциями языка проектирования являются следующие конструкции:

```
ВЫЗОВ: ОСТАНОВКА ТАЙМЕРА (;)  CALL STOP$TIMER;
ВОЗВРАТ                       RETURN;
```

Компилятор PL/M не допускает пробелов в именах. Поэтому имя процедуры STOP TIMER должно быть либо обозначено без пробела, либо для удобства чтения пробел должен быть заменен другим знаком, например знаком доллара. Поскольку знак доллара игнорируется компилятором, он не влияет на интерпретацию имени.

5.7. Условные конструкции

В языке проектирования условными конструкциями являются:

```
ЕСЛИ условие проверки есть «истина»
  ТО выполнить что-либо
```

и

```
ЕСЛИ условие проверки есть «истина»
  ТО выполнить что-либо
  ИНАЧЕ выполнить что-либо другое
```

Им соответствуют аналогичные конструкции PL/M:

```
IF ВЫРАЖЕНИЕ
  THEN утверждение;
```

и

```
IF выражение
  THEN утверждение-1;
  ELSE утверждение-2;
```

Точка с запятой не должна следовать за выражением, поскольку компилятор PL/M воспринимает фразу IF ... THEN ... как единую конструкцию. В качестве примера можно привести следующую условную конструкцию PL/M:

```
IF KEYSWITCH = 0
  THEN RETURN;
```

которая эквивалентна конструкции языка проектирования:

```
ЕСЛИ ПЕРЕКЛЮЧАТЕЛЬ ВЫКЛЮЧЕН
  ТО ВОЗВРАТ
```

Выражение может определить сложное условие, как в следующем примере:

```
IF KEYSWITCH <> 0 AND TIMER$STATE = 0
```

Эта операция PL/M эквивалентна следующей операции языка проектирования:

ЕСЛИ ПЕРЕКЛЮЧАТЕЛЬ ВКЛЮЧЕН И ТАЙМЕР В СОСТОЯНИИ ПОКОЯ

Символ < > означает *не равно* и при этом мы предположили, что переменная TIMER$STATE становится равной нулю, как только таймер закончит отсчет времени.

```
/* ЕСЛИ ИНДИКАТОР ЗАПОЛНЕНИЯ ПОКАЗЫВАЕТ, ЧТО
   ВХОДНАЯ ЗАПИСЬ ЗАПОЛНЕНА                          */
                          IF FULL$INDICATOR=10
/*    ТО ПЕРЕВЕСТИ СТАТУС В «ЗАПОЛНЕНО»              */
                            THEN STATUS=1;
/*    ИНАЧЕ ПЕРЕВЕСТИ СТАТУС В «НЕ ЗАПОЛНЕНО»        */
                            ELSE STATUS=0;
```

Рис. 5.2. Пример конструкции проверки PL/M.

Пример использования второй условной конструкции типа IF .. THEN ... ELSE ... приведен на рис. 5.2. Показаны как операции языка проектирования, так и соответствующие операции PL/M. Мы предполагаем, что структура данных INPUT RECORD содержит десять символов и поэтому считается заполненной, когда PL/M-переменная FULL$INDICATOR равна десяти.

```
/* ЕСЛИ СИМВОЛ СОДЕРЖИТ ЦИФРОВОЙ КОД                 */
          IF CHARACTER >= '0' AND CHARACTER <= '9'
/*   ТО ВЫПОЛНИТЬ                                    */
               THEN
/*     ЕСЛИ СТАТУС УКАЗЫВАЕТ ЧТО ЗАПИСЬ НЕ
       ЗАПОЛНЕНА                                     */
                  IF CTATUS = 0;
/*       ТО ВЫЗОВ: ДОБАВЛЕНИЕ К ВХОДНОЙ ЗАПИСИ
         (СИМВОЛ;)                                   */
                    THEN CALL ADD$TO$INPUT$RECORD
                    (CHARACTER);
/*           ИНАЧЕ ВЫЗОВ: СИГНАЛ (;)                 */
                    ELSE CALL BEEP;
/*       КОНЕЦ                                       */
```

Рис. 5.3. Пример вложенных конструкций проверки PL/M.

Условные конструкции PL/M могут быть вложенными. Пример вложенной конструкции показан на рис. 5.3. В некоторых случаях скобки DO ... END могут быть опущены.

В этом примере мы видим, что ELSE может следовать за двумя THEN без каких-либо скобок. Когда это случается, ELSE *всегда* сопоставляется с непосредственно предшествующим THEN. Так как скобки DO ... END могут внести ясность при

интерпретации операций PL/M, мы рекомендуем всегда использовать их, чтобы избежать неверного толкования. На рис. 5.4

```
/* ЕСЛИ СИМВОЛ СОДЕРЖИТ ЦИФРОВОЙ КОД                    */
          IF CHARACTER >= '0' AND CHARACTER <= '9'
/*   ТО ВЫПОЛНИТЬ                                       */
               THEN DO;
/*     ЕСЛИ СТАТУС УКАЗЫВАЕТ ЧТО ЗАПИСЬ НЕ ЗАПОЛНЕНА   */
                    IF STATUS = 0
/*     ТО ВЫЗОВ: ДОБАВЛЕНИЕ К ВХОДНОЙ ЗАПИСИ
       (СИМВОЛ;)                                        */
                         THEN CALL ADD$TO$INPUT$RECORD
                         (CHARACTER);
/*     ИНАЧЕ ВЫЗОВ: СИГНАЛ (;)                         */
                         ELSE CALL BEEP;
/*       КОНЕЦ                                          */
                         END;
```

Рис. 5.4. Другая возможная реализация примера на рис. 5.3.

показан предыдущий пример с использованием скобок DO ... ... END. Если в конструкции языка проектирования определено, что ELSE соответствует первому THEN, скобки DO ... END должны использоваться обязательно, как показано на рис. 5.5.

```
/* ЕСЛИ СИМВОЛ СОДЕРЖИТ ЦИФРОВОЙ КОД                    */
          IF CHARACTER >= '0' AND CHARACTER <= '9'
/*   ТО ВЫПОЛНИТЬ                                       */
               THEN DO:
/*     ЕСЛИ СТАТУС УКАЗЫВАЕТ ЧТО ЗАПИСЬ НЕ ЗАПОЛНЕНА   */
                    IF STATUS = 0
/*       ТО ВЫЗОВ: ДОБАВЛЕНИЕ К ВХОДНОЙ ЗАПИСИ
         (СИМВОЛ;)                                      */
                         THEN CALL ADD$TO$INPUT$RECORD
                         (CHARACTER);
/*       КОНЕЦ                                          */
                    END;
/*   ИНАЧЕ ВЫЗОВ: СИГНАЛ (;)                            */
               ELSE CALL BEEP;
```

Рис. 5.5. Пример, показывающий использование ELSE в паре с первым THEN.

Простое смещение ELSE на одну линию с требуемым THEN без соответствующих скобок может не привести к желаемому результату. Как и в языке проектирования, смещение предложений PL/M относительно начала строки служит только для удобства чтения, а интерпретация этих предложений не зависит от смещения.

Для того чтобы проверить символьный код, представляющий

цифровую величину, необходимо сопоставить ему один из символьных кодов последовательности от нуля до девятки включительно. В PL/M символьные коды от нуля до девятки обозначаются как '0', '1', '2', . . ., '9', при этом апострофы позволяют делать различие между символьными кодами, обозначающими числовые величины, и цифрами. Более подробно мы обсудим использование символьных кодов в гл. 6 и приложении Б. Поскольку символ $<=$ обозначает *меньше или равно*, а символ $>=$ обозначает *больше или равно*, проверка цифровых кодов PL/M производится с помощью операции:

```
IF CHARACTER >= '0' AND CHARACTER <= '9'
```

Отдельный входной параметр может быть передан вызываемой процедуре путем заключения имени параметра в круглые скобки, как показано в следующей операции:

```
CALL ADD$TO$INPUT$RECORD (CHARACTER);
```

Более подробно мы рассмотрим параметры PL/M ниже.

На рисунках, которые приводились для иллюстрации конверсии языка проектирования в PL/M, операции PL/M были смещены вправо по отношению к соответствующим операциям языка проектирования. Это смещение используется для удобства чтения и применяется во всех примерах книги.

Выражения и конструкции присваивания PL/M были введены в предыдущих разделах с помощью примеров. Эти концепции развиваются далее в следующих разделах.

5.8. Выражения

Выражения PL/M состоят из набора переменных PL/M и констант, объединеных *операторами*. Например, фраза языка проектирования

```
60 * SECONDS
```

которая использовалась в процедуре ЗАПУСК ТАЙМЕРА системы охранной сигнализации, является выражением PL/M, обозначающим произведение константы 60 и переменной SECONDS. Звездочка (*) является оператором умножения, знак плюс (+), знак минус (—) и слеш (/) — операторами сложения, вычитания и деления соответственно. Другим примером, иллюстрирующим использование *арифметических операторов*, является выражение:

```
3600 * HOURS + 60 * MINUTES + SECONDS
```

Это выражение используется для вычисления времени в секундах из его эквивалентного представления в часах, минутах и секундах. В PL/M умножение и деление *предшествуют* сложению и вычитанию. Следовательно, предыдущее выражение вос-

принимается так же, как если бы оно было представлено со скобками следующим образом:

(3600 * HOURS) + (60 * MINUTES) + SECONDS

Интерпретация этого выражения любым другим способом требует явного использования скобок. Выражения, подобные рассмотренным, называются *арифметическими выражениями.*

Примеры *логических выражений* были рассмотрены в разделах, описывающих циклические условные конструкции. Операторы, используемые в логических выражениях, могут являться *логическими операторами, операторами отношения* или *арифметическими операторами.* Логическими операторами в PL/M являются операторы AND, OR и NOT. Операторы отношения PL/M показаны в следующей таблице:

Оператор	Интерпретация
=	равно
<	меньше, чем
>	больше, чем
<=	меньше или равно
>=	больше или равно
<>	не равно

Когда логическое выражение используется в условном цикле или в условной конструкции, оно в любой момент времени должно иметь значение либо «истина», либо «ложь». Чтобы определить, продолжать ли выполнение операций цикла и должны ли выполняться в условной конструкции операции, следующие за THEN или за ELSE, проверяется значение логического выражения. Если значением является «истинно», продолжают выполняться операции цикла или выполняются операции после THEN. Если значением является «ложно», цикл завершается или выполняются операции части ELSE, если она существует.

Между операторами существует отношение предшествования, как показано в таблице:

Порядок следования	Оператор
1	*, / (умножение, деление)
2	+, — (сложение, вычитание)
3	оператор отношения
4	NOT
5	AND
6	OR

Как и прежде, для изменения указанного порядка предшествования, а также для большей ясности и однозначности при чтении могут использоваться скобки. Если дело касается ясности, круглые скобки играют ту же роль, что и групповые скобки DO ... END. Скобки DO ... END группируют операции, а круглые скобки — части выражений. В том и другом случае группирование с помощью скобок позволяет надеяться, что микрокомпьютер выполнит то, что мы хотим от него.

Этим завершается наше обсуждение выражений PL/M. Теперь мы вернемся к конструкциям присваивания PL/M. Формат конструкции присваивания выглядит следующим образом:

переменная = выражение;

где выражение является арифметическим выражением. После выполнения операции, определенной конструкцией присваивания, значение переменной равно вычисленному значению выражения. В качестве примера можно рассмотреть следующую конструкцию:

```
TIME = 3600 * HOURS + 60 * MINUTES + SECONDS
```

которая вычисляет время в секундах из эквивалентного представления времени в часах, минутах и секундах.

5.9. Описание данных

Как мы уже видели, имеются две различные категории данных в PL/M: константы и переменные. Для каждой переменной компилятору PL/M должен быть известен диапазон значений этой переменной. Все компиляторы PL/M предусматривают по крайней мере два таких диапазона. Если переменная должна принимать не более 256 различных значений, например от 0 до 255, тогда она называется *байтовой* переменной. Если ее использование предполагает не более 65 536 значений, то она называется *адресной* переменной. Адресная переменная состоит из двух байтов, рассматриваемых как один объект. Байтовый или адресный *тип данных* должен быть *описан* для компилятора до того, как этот тип данных будет использован в процедуре. *Описание данных* PL/M имеет следующую форму:

```
DECLARE SECONDS BYTE;
DECLARE TIME ADDRESS;
```

Эти два описания могут быть объединены в одно описание данных:

```
DECLARE SECONDS BYTE, TIME ADDRESS;
```

Описание переменных одного и того же типа может быть также объединено следующим образом:

```
DECLARE (SECONDS, MINUTES, HOURS) BYTE;
```

В PL/M предусмотрено группирование данных в структуры данных. Структура в виде списка данных называется в PL/M *массивом.* В PL/M может быть также определен тип структуры иерархических данных, называемый *структурой.* В следующих разделах мы рассмотрим, как описываются массивы и структуры PL/M.

5.10. Массивы PL/M

В гл. 4 была введена структура данных в виде списка под названием ВХОДНАЯ ЗАПИСЬ. Она была представлена как модульная структура данных для запоминания цифровых символьных кодов в модуле ОБРАБОТКИ ВХОДНОЙ ЗАПИСИ. При описании для этой цели массива PL/M необходимо определить максимальное число символьных кодов, которые должны храниться в массиве. Пусть это число равно десяти, как показано в следующем примере описания структуры данных для ВХОДНОЙ ЗАПИСИ:

МОДУЛЬНАЯ СТРУКТУРА ДАННЫХ: ВХОДНАЯ ЗАПИСЬ

ИМЯ	РАЗМЕР	ТИП	СОДЕРЖАНИЕ	ПРИМЕЧАНИЯ
ВХОДНАЯ ЗАПИСЬ	10	БАЙТ	ЦИФРОВЫЕ КОДЫ	

Используя эту информацию, структуру данных ВХОДНАЯ ЗАПИСЬ можно описать как массив PL/M следующим образом:

```
DECLARE INPUT$RECORD (10) BYTE;
```

Манипулирование элементами массива может производиться с использованием *индекса,* который может быть либо константой, либо переменной, либо выражением. В качестве примера на рис. 5.6 приводится процедура СТИРАНИЯ ВХОДНОЙ ЗАПИСИ (см. рис. 4.20), где показаны операции конвертирования процедур языка проектирования в PL/M. В этом примере переменная INDEX используется в качестве индекса цикла PL/M, а также в качестве индекса массива, а переменная FULL$INDICATOR устанавливается при этом в нуль, чтобы показать, что массив пуст. Так как в языке проектирования не было аналога PL/M-переменной INDEX, *описание структуры данных* для переменной INDEX добавляется во время конверсии в виде заголовка. Читатель должен снова обратить внимание на апострофы, которые позволяют отличать символьный код, например, нуля, '0', от цифрового значения нуля. И нако-

```
/* МОДУЛЬНАЯ СТРУКТУРА ДАННЫХ: ВХОДНАЯ ЗАПИСЬ                    */
/*                                                                */
/* ИМЯ          РАЗМЕР ТИП   СОДЕРЖАНИЕ     ПРИМЕЧАНИЯ          */
/* ---          ------ ---   ----------     ----------          */
/* ВХОДНАЯ ЗАПИСЬ   10   БАЙТ  ЦИФРОВЫЕ КОДЫ                     */
                          DECLARE INPUT$RECORD (10) BYTE;
/*   ИНДИКАТОР ЗАПОЛНЕНИЯ 1  БАЙТ СЧЕТЧИК                        */
                          DECLARE FULL$INDICATOR BYTE;
                                    •
                                    •
                                    •
/* ПРОЦЕДУРА: СТИРАНИЕ ВХОДНОЙ ЗАПИСИ (;)                        */
/* ************************************************************** */
/* ЛОКАЛЬНЫЕ ПАРАМЕТРЫ, НЕ ИСПОЛЬЗУЕМЫЕ В ЯЗЫКЕ ПРОЕКТИРОВАНИЯ   */
/* ИМЯ          РАЗМЕР  ТИП    СОДЕРЖАНИЕ    ПРИМЕЧАНИЯ          */
/* ---          ------  ---    ----------    ----------          */
/* ИНДЕКС         1     БАЙТ   ИНДЕКС                           */
                          DECLARE INDEX BYTE;
/* ************************************************************** */
/* НАЧАЛО ПРОЦЕДУРЫ                                               */
/*   ВЫПОЛНИТЬ ДЛЯ КАЖДОГО ЭЛЕМЕНТА ВХОДНОЙ ЗАПИСИ               */
                                    DO INDEX = 0 TO 9;
/*     УСТАНОВИТЬ ЗНАЧЕНИЕ ЭЛЕМЕНТА В КОД НУЛЯ                    */
                                     INRUT$RECORD (INDEX) = '0';
/*   КОНЕЦ                                                        */
                                    END;
/*   СБРОСИТЬ ИНДИКАТОР ЗАПОЛНЕНИЯ                                */
                                    FULL$INDICATOR = 0;
/*   ВОЗВРАТ                                                      */
                                    RETURN;
/* КОНЕЦ ПРОЦЕДУРЫ                                                */
```

Рис. 5.6. Пример, иллюстрирующий использование PL/M-массива.

нец, следует отметить, что описания массива INPUT$RECORD и переменной FULL$INDICATOR не содержатся внутри самих процедур. Поскольку они являются модульными структурами данных, их описания в PL/M располагаются на уровне модуля, чтобы они были доступны другим процедурам модуля. В то же время описание переменной INDEX располагается внутри

процедуры, поскольку эта переменная является локальным параметром и не содержит информации для остальных процедур модуля.

5.11. Структуры PL/M

В гл. 4 была описана процедура СОРТИРОВКИ, которая обрабатывает записи в структуре данных, названной ФАЙЛОМ. Сейчас мы хотим показать, как описывается PL/M-структура FILE и как обрабатывается информация в структурах PL/M.

Как и прежде, допустим, что каждая запись в структуре FILE содержит имя, домашний адрес и порядковый номер, используемый в качестве ключа. Кроме того, в отличие от примера в гл. 4 допустим, что каждый порядковый номер является положительным числом, не превосходящим 65 536, и представляется адресной переменной, а имя и адрес содержат каждый не более 20 алфавитных или цифровых символов, представляемых символьными кодами. Таким образом, каждая запись состоит из трех элементарных данных, которые могут быть объявлены в одном описании данных PL/M следующим образом:

```
DECLARE SERIAL$NUMBER ADDRESS, NAME (20) BYTE,
STREET$ADDRESS (20) BYTE;
```

Предположив, что в структуре FILE имеется 256 таких записей, мы определим размер структуры FILE, используя *описание структуры* PL/M вместо предыдущего описания следующим образом:

```
DECLARE FILE (256) STRUCTURE (SERIAL$NUMBER ADDRESS,
                              NAME (20) BYTE,
                              STREET$ADDRESS (20) BYTE);
```

Это описание структуры означает, что FILE является структурой PL/M, содержащей 256 записей, каждая из которых состоит из двухбайтового порядкового номера, 20-байтового имени и 20-байтового адреса жительства. Отдельные записи в структуре FILE могут обрабатываться с помощью индекса, значение которого определяет номер записи в диапазоне от 0 до 255. Например, порядковый номер в *десятой* записи структуры FILE определяется следующим образом:

```
FILE (9).SERIAL$NUMBER
```

при этом *точка* связывает имя элементарного данного в записи с именем структуры. Элементы двух массивов в каждой записи обрабатываются с использованием индексов, значения которых изменяются от 0 до 19. Например, первый символьный код, хранящийся в имени *пятнадцатой* записи структуры FILE,

```
/*    МОДУЛЬНАЯ СТРУКТУРА ДАННЫХ: ФАЙЛ                              */
/*                                                                   */
/* ИМЯ              РАЗМЕР ТИП   СОДЕРЖАНИЕ ПРИМЕЧАНИЯ              */
/* ---              ------ ---   ---------- ----------              */
/* ФАЙЛ                                          1                  */
/*   ЗАПИСЬ                                                          */
/*     ПОРЯДКОВЫЙ     1    АДРЕС ЧИСЛО
       НОМЕР                                                       */
/*     ИМЯ           20    БАЙТ  АЛФАВИТНЫЕ КОДЫ                    */
/*     ДОМАШНИЙ      20    БАЙТ  АЛФАВИТНО-ЦИФРОВЫЕ
       АДРЕС                     КОДЫ                              */
/*                                                                   */
/* ПРИМЕЧАНИЕ 1. ФАЙЛ СОСТОИТ ИЗ 256 ЗАПИСЕЙ.                       */
                    DECLARE FILE (256) STRUCTURE
                      (SERIAL$NUMBER ADDRESS,
                      NAME (20) BYTE,
                      STREET$ADDRESS (20) BYTE);
/* ------------------------------------------------------------- */
/* ПРОЦЕДУРА: СОРТИРОВКА (;)                                     */
/* ************************************************************* */
/*   ЛОКАЛЬНЫЕ ПАРАМЕТРЫ, НЕ ИСПОЛЬЗУЕМЫЕ В ЯЗЫКЕ
     ПРОЕКТИРОВАНИЯ                                               */
/*                                                                */
/*   ИМЯ        РАЗМЕР  ТИП    СОДЕРЖАНИЕ  ПРИМЕЧАНИЯ             */
/*   ---        ------  ---    ----------  ----------             */
/*   ИНДЕКС1      1     БАЙТ   ИНДЕКС                            */
                  DECLARE INDEX1 BYTE;
/*   ИНДЕКС2      1     БАЙТ   ИНДЕКС                            */
                  DECLARE INDEX2 BYTE;
/*   ИНДЕКС3      1     БАЙТ   ИНДЕКС                            */
                  DECLARE INDEX3 BYTE;
/*   ЯЧЕЙКА       1     АДРЕС ЗАПОЛНЕННЫЙ АДРЕС
     ПОРЯДКОВОГО
     НОМЕРА                                                      */
                  DECLARE SAVE$SERIAL$NUMBER ADDRESS;
/*   ЯЧЕЙКА ИМЕНИ 1     БАЙТ   ЗАПОЛНЕННЫЙ БАЙТ                  */
                  DECLARE SAVE$NAME BYTE;
/*   ЯЧЕЙКА       1     БАЙТ   ЗАПОЛНЕННЫЙ БАЙТ
     ДОМАШНЕГО
     АДРЕСА                                                      */
                  DECLARE SAVE$STREET BYTE
/* ************************************************************* */
/* НАЧАЛО ПРОЦЕДУРЫ                                              */
/*   ВЫБРАТЬ ПЕРВУЮ ЗАПИСЬ В ФАЙЛЕ                               */
                    INDEX1 = 0;
```

Рис. 5.7. Пример, иллюстрирующий использование PL/M-структуры.

```
/*  ВЫПОЛНЯТЬ НЕПРЕРЫВНО                                    */
                    DO WHILE 1 = 1;
/*    ВЫПОЛНИТЬ ДЛЯ КАЖДОЙ ЗАПИСИ В ФАЙЛЕ
      СЛЕДУЮЩЕЙ ЗА ВЫБРАННОЙ ЗАПИСЬЮ                        */
/*                                                          */
                      DO INDEX2 = INDEX1 + 1 TO 255;
*/      ЕСЛИ КЛЮЧ ЗАПИСИ МЕНЬШЕ КЛЮЧА ВЫБРАННОЙ
        ЗАПИСИ                                              */
                    IF FILE (INDEX2).SERIAL$NUMBER <
                     FILE (INDEX1).SERIAL$NUMBER
*/        ТО ПОМЕНЯТЬ МЕСТАМИ ЗАПИСЬ И ВЫБРАННУЮ            */
/*        ЗАПИСЬ И ПРОДОЛЖИТЬ, ИСПОЛЬЗУЯ ЗАМЕНЕННУЮ
          ЗАПИСЬ                                            */
                        THEN DO:
                            SAVE$SERIAL$NUMBER =
                               FILE (INDEX1)
                               .SERIAL$NUMBER;
                            FILE (INDEX1)
                               .SERIAL$NUMBER =
                               FILE (INDEX2)
                               .SERIAL$NUMBER;
                            FILE (INDEX2)
                               .SERIAL$NUMBER =
                               SAVE$SERIAL$NUMBER;
                            DO INDEX3 = 0 TO 19;
                              SAVE$NAME = FILE
                              (INDEX1).NAME(INDEX3);
                              FILE (INDEX1)
                              .NAME(INDEX3) =
                              FILE (INDEX2).NAME
                              (INDEX3);
                              FILE (INDEX2).NAME
                              (INDEX3) = SAVE$NAME;
                              SAVE$STREET = FILE
                              (INDEX1)
                              .STREET$ADDRESS
                              (INDEX3);
                              FILE (INDEX1)
                              .STREET$ADDRESS
                              (INDEX3) = FILE(INDEX2)
                              .STREET$ADDRESS
                              (INDEX3);
                              FILE (INDEX2)
                              .STREET$ADDRESS
                              (INDEX3) = SAVE$STREET;
                            END;
```

Рис. 5.7. (Продолжение.)

```
                                    END;
                                END;
/*      ЕСЛИ ВЫБРАНЫ ВСЕ ЗАПИСИ КРОМЕ ПОСЛЕДНЕЙ                    */
                                IF INDEX1 = 254
/*        ТО ВОЗВРАТ                                               */
                                    THEN RETURN;
/*        ИНАЧЕ ВЫБРАТЬ СЛЕДУЮЩУЮ ЗАПИСЬ В ФАЙЛЕ                   */
                                    ELSE INDEX1 = INDEX1 + 1;
/*    КОНЕЦ                                                        */
                                END;
/* КОНЕЦ ПРОЦЕДУРЫ                                                 */
```

Рис. 5.7. (Продолжение.)

определяется следующим образом:

```
FILE (14).NAME (0)
```

а *тринадцатый* символьный код, хранящийся в домашнем адресе *двадцать первой* записи структуры FILE, определяется как

```
FILE (20).STREET$ADDRESS (12)
```

На рис. 5.7 мы воспроизводим процедуру сортировки из гл. 4 с целью иллюстрации использования структур PL/M.

До сих пор в наших примерах с использованием PL/M мы опускали описания, необходимые для определения имен процедур и идентификации их входных и выходных параметров. Следующими мы рассмотрим именно эти вопросы.

5.12. Описание процедур

Для таких процедур, как процедура СТИРАНИЯ ВХОДНОЙ ЗАПИСИ, которая не имеет входных или выходных параметров, *описание процедуры* имеет следующий вид:

```
CLEAR$INPUT$RECORD: PROCEDURE;
```

Конец процедуры обозначается фразой END следующим образом:

```
END CLEAR$INPUT$RECORD;
```

Имя процедуры во фразе END указывать не обязательно, однако если оно указывается, то должно в точности соответствовать имени в описании процедуры.

Представленное выше описание процедур годится только в том случае, если процедура вызывается внутри одного модуля. Если же процедура вызывается процедурами, содержащимися в других модулях, компилятору PL/M должно быть сообщено об этом. Для этой цели используется *общий* (public) *атрибут*.

Примером назначения процедуре общего атрибута является описание процедуры BEEP в следующей форме:

```
BEEP: PROCEDURE PUBLIC;
```

В дополнение к этому процедура BEEP должна быть описана как *внешняя процедура* в каждом модуле, содержащем процедуры, вызывающие BEEP. *Описание внешней процедуры* имеет следующий вид:

```
BEEP: PROCEDURE EXTERNAL;
END BEEP
```

Мы еще будем и далее иллюстрировать использование общих атрибутов и описаний внешней процедуры. Сначала же рассмотрим, как используется описание процедур PL/M для идентификации входных параметров процедуры и возврата выходных параметров из процедуры PL/M.

5.13. Параметры

Входные параметры, которые могут быть как константами, так и переменными, идентифицируются в описании процедуры. При этом имя входного параметра, заключенное в скобки, помещается в описание процедуры. Например, описание процедуры ДОБАВЛЕНИЯ К ВХОДНОЙ ЗАПИСИ имеет следующий вид:

```
ADD$TO$INPUT$RECORD: PROCEDURE (CHARACTER);
```

Если используется более одного входного параметра, имена параметров внутри скобок разделяются запятыми. В одном из предыдущих разделов мы видели, как вызывающая процедура идентифицировала и передавала значения входных параметров в вызываемую процедуру. Читатель может вспомнить, что процедура ДОБАВЛЕНИЯ К ВХОДНОЙ ЗАПИСИ вызывалась следующим образом:

```
CALL ADD$TO$INPUT$RECORD (CHARACTER);
```

Этот пример иллюстрирует, как идентифицируется переменный входной параметр CHARACTER и как его значение передается в вызываемую процедуру. Примером использования константы в качестве входного параметра являются описание процедуры и операция вызова для процедуры ЗАПУСК ТАЙМЕРА в следующем виде:

```
START$TIMER: PROCEDURE (SECONDS) PUBLIC;
CALL START$TIMER (5);
```

Если для процедуры, которой передается переменный входной параметр, требуется описание внешней процедуры, оно

имеет следующий вид:

```
ACTUATE$ALARM: PROCEDURE (ALARM) EXTERNAL;
  DECLARE ALARM BYTE;
  END ACTUATE$ALARM;
```

Отметим, что тип данных должен быть обязательно описан в описании внешней процедуры, несмотря на то что он должен также быть описан в самой процедуре. Если процедуре передается константа, то ей должно быть присвоено в описании внешней процедуры имя для того, чтобы иметь возможность описать тип данных для этой константы. Это иллюстрируется следующим примером для процедуры ЗАПУСКА ТАЙМЕРА:

```
START$TIMER: PROCEDURE (SECONDS) EXTERNAL
  DECLARE SECONDS BYTE;
END START$TIMER;
```

Если процедуре передается более одного входного параметра, число их в каждой операции вызова, в соответствующих описаниях процедур и в каждом описании внешней процедуры должно быть одним и тем же. Более того, каждому входному параметру должен быть назначен один и тот же тип данных в операции вызова и в каждом описании вызываемых процедур. Это означает, что если первому входному параметру в операции вызова назначен тип данных BYTE, такой же тип данных должен быть назначен первому входному параметру во всех соответствующих описаниях процедур.

Единственный выходной параметр может быть возвращен процедурой PL/M простым способом. Во-первых, в описании процедуры должен быть определен тип данных выходного параметра:

```
TEST$IF$INPUT$RECORD$FULL: PROCEDURE BYTE;
```

Таким образом, процедура ПРОВЕРКИ ЗАПОЛНЕНИЯ ВХОДНОЙ ЗАПИСИ возвращает байтовую переменную в качестве выходного параметра. Затем должно быть идентифицировано *значение* выходного параметра с помощью конструкции ВОЗВРАТ, модифицированной в PL/M следующим образом:

```
RETURN выражение;
```

Так, для процедуры ПРОВЕРКИ ЗАПОЛНЕНИЯ ВХОДНОЙ ЗАПИСИ возврат переменной величины STATUS осуществляется с помощью конструкции ВОЗВРАТА следующим образом:

```
RETURN STATUS;
```

Для вызова процедуры, возвращающей выходной параметр, операция вызова в PL/M не используется. Вместо этого ис-

пользуется конструкция, подобная конструкции присваивания в PL/M:

```
STATUS = TEST$IF$INPUT$RECORD$FULL
```

Полностью процедура ПРОВЕРКИ ЗАПОЛНЕНИЯ ВХОДНОЙ ЗАПИСИ приведена на рис. 5.8 вместе с дополнительными конструкциями, иллюстрирующими использование рассмотренных концепций. Поскольку ИНДИКАТОР ЗАПОЛНЕНИЯ является частью модульной структуры данных, описание этой переменной размещается в описании модуля. Описание данных и операция вызова PL/M, которые показаны ниже процедуры ПРОВЕРКИ ЗАПОЛНЕНИЯ ВХОДНОЙ ЗАПИСИ, являются частью процедуры ЗАПОЛНЕНИЯ ВХОДНОЙ ЗАПИСИ. Они показаны для иллюстрации того, как предыдущая процедура вызывается последующей процедурой. Следует отметить, что процедура, вызываемая другой процедурой, находящейся в том же модуле, *должна* предшествовать вызывающей процедуре, как показано в примере. Отметим также, что переменная STATUS описана дважды: по одному разу в каждой процедуре. Таким образом, существует две версии переменной STATUS, и каждая процедура манипулирует со своей собственной версией. Информация передается от одной версии переменной STATUS к другой только тогда, когда вызывается процедура ПРОВЕРКИ ЗАПОЛНЕНИЯ ВХОДНОЙ ЗАПИСИ с переменной STATUS в качестве выходного параметра. В действительности же информация просто *копируется* из одной версии переменной STATUS в другую, обеспечивая таким образом целостность предыдущей версии независимо от того, как обрабатывается последующая.

И последнее относительно выходных параметров: если процедура, возвращающая выходной параметр, вызывается из другого модуля, ей должен быть назначен общий атрибут. В качестве примера рассмотрим процедуру СЧИТЫВАНИЯ ПЕРЕКЛЮЧАТЕЛЯ:

```
READ$KEYBOARD: PROCEDURE BYTE PUBLIC;
```

Каждый модуль, содержащий процедуру, вызывающую процедуру СЧИТЫВАНИЯ ПЕРЕКЛЮЧАТЕЛЯ, должен содержать описание внешней процедуры в следующем виде:

```
READ$KEYBOARD: PROCEDURE BYTE EXTERNAL;
END READ$KEYBOARD;
```

Описанные методы практически не работают с массивами и структурами PL/M. Перечислять каждый элемент массива или структуры PL/M в качестве входного параметра не только утомительно, но и в высшей степени неэффективно. Кроме того, так как процедура PL/M может возвратить только отдельный

```
/* МОДУЛЬНАЯ СТРУКТУРА ДАННЫХ: ВХОДНАЯ ЗАПИСЬ          */
/*                                                     */
/* ИМЯ          РАЗМЕР ТИП  СОДЕРЖАНИЕ  ПРИМЕЧАНИЯ    */
/* ---          ------ ---  ----------  ----------    */
/* ВХОДНАЯ        10   БАЙТ ЦИФРОВЫЕ
   ЗАПИСЬ                   КОДЫ                      */
/*   ИНДИКАТОР    1    БАЙТ СЧЕТЧИК
     ЗАПОЛНЕНИЯ                                      */
            DECLARE FULL$INDICATOR BYTE;
                              .
                              .
                              .
/* ------------------------------------------------- */
/* ПРОЦЕДУРА: ПРОВЕРКА ЗАПОЛНЕНИЯ ВХОДНОЙ ЗАПИСИ
   (; СТАТУС)                                        */
          TEST$IF$INPUT$RECORD$FULL: PROCEDURE BYTE;
/* ***************************************************** */
/*   ВЫХОДНОЙ ПАРАМЕТР: СТАТУС                        */
/*                                                     */
/*   ИМЯ      РАЗМЕР   ТИП    СОДЕРЖАНИЕ   ПРИМЕЧАНИЯ */
/*   ---      ------   ---    ----------   ---------- */
/*   СТАТУС     1      БАЙТ   ФЛАЖОК                  */
              DECLARE STATUS BYTE;
/* ***************************************************** */
/* НАЧАЛО ПРОЦЕДУРЫ                                    */
/*   ЕСЛИ ИНДИКАТОР ЗАПОЛНЕНИЯ ПОКАЗЫВАЕТ, ЧТО
     ВХОДНАЯ ЗАПИСЬ ЗАПОЛНЕНА                          */
              IF FULL$INDICATOR = 10
/*     ТО ПЕРЕВЕСТИ СТАТУС В СОСТОЯНИЕ "ЗАПОЛНЕНО"    */
                 THEN STATUS = 1;
/*     ИНАЧЕ ПЕРЕВЕСТИ СТАТУС В СОСТОЯНИЕ
       "НЕ ЗАПОЛНЕНО"                                  */
                 ELSE STATUS = 0;
/*   ВОЗВРАТ                                           */
                 RETURN STATUS;
/* КОНЕЦ ПРОЦЕДУРЫ                                     */
           END TEST$IF$INPUT$RECORD$FULL;
/* ------------------------------------------------- */
/*   ЛОКАЛЬНЫЙ ПАРАМЕТР: СТАТУС                       */
/* ------------------------------------------------- */
/*   ИМЯ     РАЗМЕР   ТИП    СОДЕРЖАНИЕ   ПРИМЕЧАНИЯ */
/*   ---     ------   ---    ----------   ---------- */
/*   СТАТУС     1      БАЙТ   ФЛАЖОК                  */
              DECLARE STATUS BYTE;
                              .
                              .
                              .
```

```
/*    ВЫЗОВ: ПРОВЕРКА ЗАПОЛНЕНИЯ ВХОДНОЙ ЗАПИСИ
      (; СТАТУС)                                    */
            STATUS=TEST$IF$INPUT$RECORD$FULL;
```

Рис. 5.8. Пример, иллюстрирующий передачу выходных параметров в PL/M

выходной параметр, этот метод не может быть использован для возврата массива или структуры. В PL/M имеется несколько других способов передачи информации в виде массива или структуры от одной процедуры к другой. Метод, который более всего подходит для наших целей, использует концепцию *базированных переменных.*

5.14. Базированные переменные

Поскольку трудно передать процедуре в качестве входного параметра информацию, содержащуюся в массиве или структуре PL/M, и невозможно возвратить ее в качестве выходного параметра, вместо этого мы будем передавать *указатель*, который определяет, *где* размещена информация. Это позволит процедуре иметь доступ к информации в массиве или структуре PL/M так, как будто процедура получила копию этой информации. Для иллюстрации этой концепции используем процедуру ПОИСКА И ПЕЧАТИ ЗАПИСИ, описание которой на языке проектирования показано на рис. 5.9.

```
ПРОЦЕДУРА: ПОИСК И ПЕЧАТЬ ЗАПИСИ (КЛЮЧ ПОИСКА;)
НАЧАЛО ПРОЦЕДУРЫ
  ВЫЗОВ: ПОИСК (КЛЮЧ ПОИСКА; ЗАПИСЬ)
  ВЫЗОВ: ПЕЧАТЬ (ЗАПИСЬ;)
  ВОЗВРАТ
КОНЕЦ ПРОЦЕДУРЫ
```

Рис. 5.9. Процедура, иллюстрирующая использование базированных переменных.

В процедуре ПОИСКА И ПЕЧАТИ ЗАПИСИ (SEARCH AND PRINT RECORD) вызывается процедура ПОИСКА (SEARCH) структуры данных ФАЙЛ (FILE) по совпадению с КЛЮЧОМ ПОИСКА (SEARCH KEY). Если совпадение обнаружено, через выходной параметр ЗАПИСЬ (RECORD) возвращается полная запись (порядковый номер, имя, домашний адрес), соответствующая поисковому образу. В противном случае перед возвращением ЗАПИСИ в вызывающую процедуру в качестве выходного параметра в нее помещается сообщение 'СОВПАДЕНИЯ НЕ НАЙДЕНО' (например, составляющая записи ИМЯ устанавливается в 'СОВПАДЕНИЯ НЕ НАЙДЕНО'). Затем ЗАПИСЬ передается процедуре ПЕЧАТЬ (PRINT) в качестве входного параметра и, если требуется, печатается

вышеуказанное сообщение. Мы уже видели, как может быть описана структура данных ФАЙЛ в структуре PL/M. Структура данных ЗАПИСЬ, которая используется в качестве выходного параметра в процедуре ПОИСКА, описывается в структуре PL/M следующим образом:

```
DECLARE RECORD STRUCTURE (SERIAL$NUMBER ADDRESS,
                          NAME (20) BYTE,
                          STREET$ADDRESS (20) BYTE);
```

Затем, если найдено совпадение, соответствующая запись перемещается из ФАЙЛА в ЗАПИСЬ с помощью конструкции присваивания PL/M в следующем виде:

```
RECORD.SERIAL$NUMBER = FILE (INDEX1).SERIAL$NUMBER;
RECORD.NAME (INDEX2) = FILE (INDEX1).NAME (INDEX2);
```

В этом примере INDEX1 используется для выбора записи в структуре FILE, для которой найдено совпадение, INDEX2 используется для перемещения каждого символа из выбранной части NAME структуры FILE в часть NAME структуры RECORD. Указатель на RECORD затем возвращается в качестве выходного параметра. Этот указатель (назовем его RECORD$POINTER) должен быть описан в процедуре как адресная переменная PL/M. Кроме того, во время выполнения процедуры ПОИСК указатель RECORD$POINTER должен быть установлен в соответствующее значение, которое указывает на структуру RECORD. Это выполняется с помощью конструкции присваивания следующим образом:

```
RECORD$POINTER =. RECORD;
```

Рис. 5.10 иллюстрирует эти концепции для процедуры ПОИСКА. На рис. 5.11 показано конвертирование процедуры ПОИСКА И ПЕЧАТИ ЗАПИСИ в PL/M. В этой процедуре указатель, который возвратился из процедуры ПОИСКА в качестве выходного параметра, передается процедуре ПЕЧАТИ в качестве входного параметра. Следует отметить, что PL/M-описание структуры ЗАПИСЬ в процедуре ПОИСКА И ПЕЧАТИ ЗАПИСИ опущено, так как информация структуры в процедуре явным образом не используется.

На рис. 5.12 показано, как информация, содержащаяся в структуре ЗАПИСЬ, становится доступной для использования процедуре ПЕЧАТИ. В этой процедуре структура ЗАПИСЬ описывается как *базированная* или размещенная там, куда указывает RECORD$POINTER. Использование ключевого слова BASED в PL/M-описании приводит к тому, что все обращения к параметру по имени ЗАПИСЬ (RECORD) означает обращение к структуре ЗАПИСЬ, которая была описана в процедуре

```
/* ПРОЦЕДУРА: ПОИСК (КЛЮЧ ПОИСКА; ЗАПИСЬ)                    */
      SEARCH: PROCEDURE (SEARCH$KEY) ADDRESS PUBLIC;
/* ************************************************************ */
                                  .
                                  .
                                  .
/*   ВЫХОДНОЙ ПАРАМЕТР: ЗАПИСЬ                                 */
/*                                                              */
/*   ИМЯ        РАЗМЕР  ТИП     СОДЕРЖАНИЕ  ПРИМЕЧАНИЯ          */
/*   ---        ------  ---     ----------  ----------          */
/*   ЗАПИСЬ                                                     */
/*     ПОРЯД-      1     АДРЕС  ЧИСЛО
      КОВЫЙ
      НОМЕР                                                    */
/*     ИМЯ        20     БАЙТ   АЛФАВИТНЫЕ КОДЫ                */
/*     ДОМАШ-     20     БАЙТ   АЛФАВИТНО-ЦИФРОВЫЕ КОДЫ
      НИЙ
      АДРЕС                                                    */
                  DECLARE RECORD STRUCTURE (SERIAL$NUMBER
                    ADDRESS, NAME (20) BYTE,
                    STREET$ADDRESS 20 BYTE),
/*   ЛОКАЛЬНЫЙ ПАРАМЕТР: УКАЗАТЕЛЬ ЗАПИСИ                      */
/*                                                              */
/*   ИМЯ        РАЗМЕР  ТИП     СОДЕРЖАНИЕ  ПРИМЕЧАНИЯ          */
/*   ---        ------  ---     ----------  ----------          */
/*   УКАЗАТЕЛЬ    1     АДРЕС  УКАЗАТЕЛЬ
      ЗАПИСИ                                                   */
                  DECLARE RECORD$POINTER ADDRESS;
/* ************************************************************ */
                                  .
                                  .
                                  .
/*   УСТАНОВИТЬ ЗНАЧЕНИЕ ЗАПИСИ РАВНЫМ ЗНАЧЕНИЮ
      ВЫБРАННОЙ ЗАПИСИ                                         */
                                  .
                                  .
                                  .
/*   ВОЗВРАТ                                                  */
                                    RECORD$POINTER =.RECORD;
                                    RETURN RECORD$POINTER;
                                  .
                                  .
                                  .
```

Рис. 5.10. Передача PL/M-структуры в качестве выходного параметра с использованием указателя.

поиска. Таким образом, информация, хранящаяся в структуре ЗАПИСЬ процедуры ПОИСКА, может быть сравнительно просто использована операциями процедуры ПЕЧАТИ.

Концепция базированных переменных, которую мы только что рассмотрели, позволяет непосредственно адресоваться к

```
/* ПРОЦЕДУРА: ПОИСК И ПЕЧАТЬ ЗАПИСИ (КЛЮЧ ПОИСКА;)    */
                SEARCH$AND$PRINT$RECORD: PROCEDURE
                (SEARCH$KEY) PUBLIC;
/* ************************************************************ */
/*   ВХОДНОЙ ПАРАМЕТР: КЛЮЧ ПОИСКА                             */
/*                                                              */
/*   ИМЯ          РАЗМЕР ТИП    СОДЕРЖАНИЕ ПРИМЕЧАНИЯ          */
/*   ---          ------ ---    ---------- ----------          */
/*   ПОИСКОВЫЙ      1    АДРЕС КЛЮЧ
     КЛЮЧ                                                     */
                DECLARE SEARCH$KEY ADDRESS;
/*                                                              */
/*   ЛОКАЛЬНЫЙ ПАРАМЕТР: УКАЗАТЕЛЬ ЗАПИСИ                      */
/*                                                              */
/*   ИМЯ          РАЗМЕР ТИП    СОДЕРЖАНИЕ ПРИМЕЧАНИЯ          */
/*   ---          ------ ---    ---------- ----------          */
/*   УКАЗАТЕЛЬ      1    АДРЕС УКАЗАТЕЛЬ
     ЗАПИСИ                                                   */
                DECLARE RECORD$POINTER ADDRESS;
/* ************************************************************ */
/* НАЧАЛО ПРОЦЕДУРЫ                                           */
/*   ВЫЗОВ: ПОИСК (КЛЮЧ ПОИСКА; ЗАПИСЬ)                       */
                RECORD$POINTER = SEARCH (SEARCH$KEY);
/*   ВЫЗОВ: ПЕЧАТЬ (ЗАПИСЬ;)                                  */
                CALL PRINT (RECORD$POINTER);
/*   ВОЗВРАТ                                                   */
                RETURN;
/* КОНЕЦ ПРОЦЕДУРЫ                                            */
              END SEARCH$AND$PRINT$RECORD;
```

Рис. 5.11. Передача указателя от одной процедуры к другой.

информации модуля PL/M из процедур другого модуля PL/M. Это не согласуется с нашими правилами проектирования структур данных, обособленных одним модулем. Однако передача копии полной структуры из одной процедуры PL/M в другую не может быть выполнена автоматически, и настаивать на этом можно только за счет увеличения сложности программного обеспечения. Кроме того, некоторая степень обособленности достигается именно за счет использования указателей, которые

```
/* ПРОЦЕДУРА: ПЕЧАТЬ (ЗАПИСЬ;)                                    */
                        PRINT: PROCEDURE (RECORD$POINTER)
                          PUBLIC;
/* ************************************************************** */
/*   ЛОКАЛЬНЫЙ ПАРАМЕТР: УКАЗАТЕЛЬ ЗАПИСИ                        */
/*                                                                */
/*   ИМЯ          РАЗМЕР ТИП    СОДЕРЖАНИЕ ПРИМЕЧАНИЯ             */
/*   ---          ------ ---    ---------- ----------             */
/*   УКАЗАТЕЛЬ      1    АДРЕС СТРУКТУРА
     ЗАПИСИ                    УКАЗАТЕЛЯ                         */
                          DECLARE RECORD$POINTER ADDRESS;
/*                                                                */
/*   ВХОДНОЙ ПАРАМЕТР: ЗАПИСЬ                                    */
/*                                                                */
/*   ИМЯ          РАЗМЕР ТИП    СОДЕРЖАНИЕ ПРИМЕЧАНИЯ             */
/*   ---          ------ ---    ---------- ----------             */
/*   ЗАПИСЬ                                        1              */
/*     ПОРЯДКО-     1    АДРЕС ЧИСЛО
       ВЫЙ НОМЕР                                                  */
/*     ИМЯ         20    БАЙТ  АЛФАВИТНЫЕ КОДЫ                    */
/*     ДОМАШНИЙ    20    БАЙТ  АЛФАВИТНО-ЦИФРОВЫЕ
       АДРЕС                   КОДЫ                             */
/*                                                                */
/* ПРИМЕЧАНИЕ 1: ЗАПИСЬ БАЗИРОВАНА В СООТВЕТСТВИИ
   С УКАЗАТЕЛЕМ ЗАПИСИ.                                           */
/*                                                                */
                          DECLARE (RECORD BASED
                            RECORD$POINTER) STRUCTURE
                            (SERIAL$NUMBER ADDRESS,
                            NAME (20) BYTE, STREET$ADDRESS (20)
                            BYTE);
/*   ЛОКАЛЬНЫЙ ПАРАМЕТР: ИНДЕКС                                  */
/*                                                                */
/*   ИМЯ        РАЗМЕР   ТИП    СОДЕРЖАНИЕ  ПРИМЕЧАНИЯ           */
/*   ---        ------   ---    ----------  ----------           */
/*   ИНДЕКС       1      БАЙТ   ИНДЕКС                          */
                          DECLARE INDEX BYTE;
/* ************************************************************** */
                                    .
                                    .
                                    .
/*   ВЫПОЛНИТЬ ДЛЯ КАЖДОГО СИМВОЛА В ПОРЦИИ "ИМЯ"
     ЗАПИСИ                                                     */
                                  DO INDEX = 0 TO 19;
/*     ВЫЗОВ: ВЫВОД СИМВОЛА (СИМВОЛ;)                          */
                                    CALL OUTPUT$CHARACTER
                                    (RECORD.NAME (INDEX));
/*   КОНЕЦ                                                      */
```

Рис. 5.12. Пример, иллюстрирующий использование указателя для доступа к PL/M-структуре.

```
END;
  .
  .
  .
```

Рис. 5.12. (Продолжение.)

разрешают доступ к соответствующим данным только тем процедурам, которым передается указатель. В нашем примере при копировании данных из структуры ФАЙЛ в структуру ЗАПИСЬ и затем при передаче PL/M-указателя RECORD$POINTER процедуре ПЕЧАТИ данные в структуре ФАЙЛ являются достаточно обособленными. По этой причине мы допускаем *разделение* данных, которое является следствием концепции базированных переменных. Отметим, что PL/M-описание указателя RECORD$POINTER *должно предшествовать* его использованию в PL/M-описании структуры данных RECORD.

В нашем примере структура данных являлась PL/M-структурой, состоящей из единственной записи. Чтобы завершить обсуждение вопроса о базированных переменных, рассмотрим, как могут быть переданы от одной процедуры к другой *массив* PL/M или структура PL/M, состоящая из *многих записей.* В любом случае указатель должен быть установлен в соответствующее значение и передан от одной процедуры другой в качестве входного или выходного параметра. Например, для массива INPUT$RECORD указатель устанавливается операцией:

```
INPUT$RECORD$POINTER = .INPUT$RECORD;
```

а для структуры PL/M — операцией:

```
FILE$POINTER = .FILE;
```

Затем, чтобы воспользоваться информацией, содержащейся в структурах данных, мы используем в PL/M-описании указатель для определения размещения структур данных. Это выполняется следующим образом:

```
DECLARE (INPUT$RECORD BASED INPUT$RECORD$POINTER)
(10) BYTE;
DECLARE (FILE BASED FILE$POINTER) (256) STRUCTURE
                                    (SERIAL$NUMBER ADDRESS,
                                    NAME (20) BYTE,
                                    STREET$ADDRESS (20) BYTE);
```

Следует помнить, что всегда, когда в PL/M используются указатели, они сначала должны быть описаны как *адресные* переменные.

5.15. Инициализация

Когда мы описывали процедуру СТИРАНИЯ ВХОДНОЙ ЗАПИСИ, мы показали, как во время работы системы инициализируется массив *переменных* PL/M. В некоторых случаях информация, хранящаяся в структуре данных, представляется *константами*, которые никогда не меняются в процессе работы системы. Примером таких данных служит последовательность *правил* (control schedule), используемых в автоматизированной системе вспрыскивания топлива, предназначенной для управления расходованием топлива в двигателях внутреннего сгорания. Структуру данных, содержащих последовательность правил, необходимо менять только в том случае, если меняется тип двигателя. Обычно в течение времени жизни автомобиля этого не случается.

Для инициализации структуры данных PL/M, содержащей константы, в описание данных вставляется атрибут DATA, в котором перечисляются значения констант, предназначенных для использования. В качестве примера рассмотрим инициализацию массива CONTROL$SCHEDULE:

```
DECLARE CONTROL$SCHEDULE (10) BYTE
        DATA (10, 15, 25, 40, 60, 76, 88, 98, 106, 110);
```

Для инициализации констант типа BYTE, как в рассмотренном примере, должны использоваться значения от 0 до 255. Для инициализации констант типа ADDRESS могут использоваться значения от 0 до 65 535. В качестве значения константы при инициализации структуры данных может быт выбрана строка символьных кодов. Для инициализации каждой константы типа BYTE необходим один символьный код, а для инициализации константы типа ADDRESS — два символьных кода. В качестве примера инициализации с помощью строки символьных кодов рассмотрим инициализацию массива MESSAGE:

```
DECLARE MESSAGE (17) BYTE DATA ('PUSH START BUTTON');
```

В этом примере MESSAGE (0) содержит символьный код 'P', MESSAGE (1) — символьный код 'U' и т. д.

Этим заканчивается описание способов обработки данных и структур данных в PL/M. Краткое описание языка программирования PL/M содержится в приложении В.

5.16. Модули PL/M

При конвертировании в PL/M процедур и описаний структур данных, входящих в состав модуля, необходимо также конвертировать в PL/M описание самого модуля. Рассматривая описание модуля ОБРАБОТКИ ВХОДНОЙ ЗАПИСИ

```
/*  MODULE: MAINTAIN INPUT RECORD                                    M004   */
                                                  MAINTAIN$INPUT$RECORD: DO;
/*  ------------------------------------------------------------------------ */
/*    DESIGNED BY:  M. D. FREEDMAN                                 9-30-78   */
/*                                                                           */
/*    PROCEDURES:      CLEAR INPUT RECORD                                    */
/*                     ADD TO INPUT RECORD                                   */
/*                     TEST IF INPUT RECORD FULL                             */
/*                     GET INPUT RECORD                                      */
/*                                                                           */
/*    EXTERNAL PROCEDURES CALLED:  BEEP                                      */
                                                  BEEP: PROCEDURE EXTERNAL;
                                                  END BEEP;
/*                                 READ KEYBOARD                             */
                                                  READ$KEYBOARD: PROCEDURE BYTE EXTERNAL;
                                                  END READ$KEYBOARD;
/*    MODULAR DATA STRUCTURE:  INPUT RECORD                                  */
/*                                                                           */
/*    NAME              SIZE    TYPE    CONTENTS          NOTES            */
/*    ----              ----    ----    --------          -----            */
/*    INPUT RECORD        10    BYTE    NUMERIC CODES                      */
                                                  DECLARE INPUT$RECORD (10) BYTE;
/*      FULL INDICATOR     1    BYTE    COUNTER                            */
                                                  DECLARE FULL$INDICATOR BYTE;
/*  ------------------------------------------------------------------------ */
/*  PROCEDURE:  CLEAR INPUT RECORD ()                                        */
                                                CLEAR$INPUT$RECORD: PROCEDURE;
/*  ************************************************************************ */
/*    LOCAL PARAMETER NOT USED IN DESIGN LANGUAGE                            */
/*                                                                           */
/*    NAME              SIZE    TYPE    CONTENTS          NOTES            */
/*    ----              ----    ----    --------          -----            */
/*    INDEX                1    BYTE    INDEX                              */
                                                  DECLARE INDEX BYTE;
/*  ************************************************************************ */
/*  BEGIN PROCEDURE                                                          */
/*    DO FOR EACH ELEMENT IN INPUT RECORD                                    */
                                                  DO INDEX = 0 TO 9;
/*      SET ELEMENT TO 'ZERO' CODE                                           */
                                                    INPUT$RECORD (INDEX) = '0';
/*    END                                                                    */
                                                  END;
/*    SET FULL INDICATOR TO EMPTY                                            */
                                                  FULL$INDICATOR = 0;
/*    RETURN                                                                 */
                                                  RETURN;
/*  END PROCEDURE                                                            */
                                                END CLEAR$INPUT$RECORD;
/*  ------------------------------------------------------------------------ */
```

Рис. 5.13. Модуль ОБРАБОТКИ ВХОДНОЙ ЗАПИСИ.

```
/*  PROCEDURE:  ADD TO INPUT RECORD (CHARACTER;)                          */
                                        ADD$TO$INPUT$RECORD: PROCEDURE (CHARACTER);
/*  ********************************************************************* */
/*    INPUT PARAMETER: CHARACTER                                          */
/*                                                                        */
/*    NAME              SIZE    TYPE    CONTENTS          NOTES         */
/*    ----              ----    ----    --------          -----         */
/*    CHARACTER           1     BYTE    NUMERIC CODE                    */
                                        DECLARE CHARACTER BYTE;
/*    LOCAL PARAMETER NOT USED IN DESIGN LANGUAGE                         */
/*                                                                        */
/*    NAME              SIZE    TYPE    CONTENTS          NOTES         */
/*    ----              ----    ----    --------          -----         */
/*    INDEX               1     BYTE    INDEX                           */
                                        DECLARE INDEX BYTE;
/*  ********************************************************************* */
/*  BEGIN PROCEDURE                                                       */
/*    SHIFT INPUT RECORD CODES TO LEFT                                    */
                                        DO INDEX = 0 TO 8;
                                          INPUT$RECORD (INDEX) = INPUT$RECORD (INDEX + 1);
                                        END;
/*    SET RIGHT-HAND INPUT RECORD ELEMENT TO CHARACTER                    */
                                        INPUT$RECORD (9) = CHARACTER;
/*    INCREMENT FULL INDICATOR                                            */
                                        FULL INDICATOR = FULL$INDICATOR + 1;
/*    RETURN                                                              */
                                        RETURN;
/*  END PROCEDURE                                                         */
                                      END ADD$TO$INPUT$RECORD;
/*  --------------------------------------------------------------------- */
/*  PROCEDURE: TEST IF INPUT RECORD FULL (;STATUS)                        */
                                      TEST$IF$INPUT$RECORD$FULL: PROCEDURE BYTE;
/*  ********************************************************************* */
/*    OUTPUT PARAMETER: STATUS                                            */
/*                                                                        */
/*    NAME              SIZE    TYPE    CONTENTS          NOTES         */
/*    ----              ----    ----    --------          -----         */
/*    STATUS              1     BYTE    FLAG                            */
                                        DECLARE STATUS BYTE;
/*  ********************************************************************* */
/*  BEGIN PROCEDURE                                                       */
/*    IF FULL INDICATOR SHOWS THAT INPUT RECORD IS FULL                   */
                                        IF FULL$INDICATOR = 10
/*      THEN  SET STATUS TO FULL                                          */
                                          THEN  STATUS = 1;
/*      ELSE  SET STATUS TO NOT FULL                                      */
                                          ELSE  STATUS = 0;
/*    RETURN                                                              */
                                        RETURN STATUS;
/*  END PROCEDURE                                                         */
                                      END TEST$IF$INPUT$RECORD$FULL;
/*  --------------------------------------------------------------------- */
```

Рис. 5.13. (Продолжение.)

```
/*  PROCEDURE: GET INPUT RECORD (;INPUT RECORD)                          */
                                        GET$INPUT$RECORD: PROCEDURE ADDRESS PUBLIC;
/*  ******************************************************************** */
/*    OUTPUT PARAMETER: INPUT RECORD                                     */
/*                                                                       */
/*    NAME              SIZE    TYPE    CONTENTS          NOTES        */
/*    ----              ----    ----    --------          -----        */
/*    INPUT RECORD       10     BYTE    NUMERIC CODES       1          */
/*                                                                       */
/*    NOTE 1: MODULAR DATA STRUCTURE                                     */
/*                                                                       */
/*    LOCAL PARAMETERS: CHARACTER                                        */
/*                      STATUS                                           */
/*                                                                       */
/*    NAME              SIZE    TYPE    CONTENTS          NOTES        */
/*    ----              ----    ----    --------          -----        */
/*    CHARACTER          1      BYTE    ALPHANUMERIC CODE                */
                                        DECLARE CHARACTER BYTE;
/*    STATUS             1      BYTE    FLAG                             */
                                        DECLARE STATUS BYTE;
/*  ******************************************************************** */
/*  BEGIN PROCEDURE                                                      */
/*    CALL: CLEAR INPUT RECORD (:)                                       */
                                        CALL CLEAR$INPUT$RECORD;
/*    DO FOREVER                                                         */
                                        DO WHILE 1 = 1;
/*      CALL: READ KEYBOARD (;CHARACTER)                                 */
                                          CHARACTER = READ$KEYBOARD;
/*      IF CHARACTER CONTAINS 'CLEAR INPUT' CODE                         */
                                          IF CHARACTER = 'C'
/*        THEN  CALL: CLEAR INPUT RECORD (;)                             */
                                            THEN  CALL CLEAR$INPUT$RECORD;
/*      IF CHARACTER CONTAINS A NUMERIC CODE                             */
                                          IF CHARACTER >= '0' AND CHARACTER <= '9'
/*        THEN  DO                                                       */
                                            THEN  DO;
/*              CALL: TEST IF INPUT RECORD FULL (;STATUS)                */
                                                    STATUS = TEST$IF$INPUT$RECORD$FULL;
/*              IF STATUS INDICATES INPUT RECORD IS NOT FULL             */
                                                    IF STATUS = 0
/*                THEN  CALL: ADD TO INPUT RECORD (CHARACTER;)           */
                                                      THEN  CALL ADD$TO$INPUT$RECORD (CHARACTER);
/*                ELSE  CALL: BEEP (;)                                   */
                                                      ELSE  CALL BEEP;
/*            END                                                        */
                                                  END;
/*      IF CHARACTER CONTAINS A TERMINATOR CODE                          */
                                          IF CHARACTER = 'T'
/*        THEN  RETURN                                                   */
                                            THEN  DO;
                                                    INPUT$RECORD$POINTER = .INPUT$RECORD;
                                                    RETURN INPUT$RECORD$POINTER;
                                                  END;
/*      IF CHARACTER CONTAINS ANY OTHER CODE                             */
                                          IF (CHARACTER < '0' OR CHARACTER > '9') AND
                                                        CHARACTER <> 'C' AND CHARACTER <> 'T
/*        THEN  CALL: BEEP (;)                                           */
                                            THEN  CALL BEEP;
/*    END                                                                */
                                        END;
/*  END PROCEDURE                                                        */
                                      END GET$INPUT$RECORD;
/*  -------------------------------------------------------------------- */
/*  END MODULE                                                           */
                                    END MAINTAIN$INPUT$RECORD;
```

Рис. 5.13. (Продолжение.)

(MAINTAIN INPUT RECORD — см. рис. 4.35), мы видим, что необходимо конвертировать только первую и последнюю строки. Первая строка

```
MODULE: MAINTAIN INPUT RECORD
```

конвертируется в PL/M-описание модуля

```
MAINTAIN$INPUT$RECORD: DO;
```

а фраза, обозначающая конец модуля,

```
END MODULE
```

конвертируется в

```
END MAINTAIN$INPUT$RECORD;
```

В последней фразе имя модуля можно опустить, но если оно используется, то должно в точности соответствовать имени в описании модуля.

Описания внешних процедур следуют за описанием модуля, а за ними в свою очередь следуют модульные структуры данных. Сами процедуры следуют за заголовком модуля в том порядке, в котором они следуют в описании модуля. На рис. 5.13 показан конвертированный модуль MAINTAIN INPUT RECORD Читатель должен обратить внимание на следующее:

- Некоторая информация заголовка, например список вызывающих процедур, в примере опущена, чтобы не перегружать рисунок лишними деталями.
- В заголовке структуры данных предусмотрена фраза LOCAL PARAMETER NOT USED IN DESIGN LANGUAGE, чтобы вставить в PL/M-описание такие параметры, которые используются как индексы или счетчики, но не встречаются в языке проектирования.
- На практике каждая процедура печатается на отдельной странице, чтобы облегчить чтение, хотя весь модуль воспринимается компилятором PL/M, как целое. Мы не делали этого в примере, чтобы проиллюстрировать связь частей модуля друг с другом.

5.17. Другие свойства PL/M

Концепции ввода, вывода и прерываний обсуждались в предыдущих главах. В PL/M существуют операции, позволяющие реализовать эти и другие функции, однако эти операции могут быть быстрее восприняты, если мы отложим их обсуждение до тех пор, пока не рассмотрим архитектуру микрокомпьютера и язык ассемблера. Поэтому мы закончим описание PL/M в конце гл. 6. Прежде чем перейти к обсуждению следующих вопросов, рассмотрим требования к документации по языку программирования.

5.18. Третий уровень документации

Третий уровень документации состоит из описания процедур и модулей системы в виде программ на языке программирования. Поскольку описание документации в виде программ содержит описание на языке проектирования в виде комментариев, второй уровень документации содержится в третьем. После того как описание программ на языке проектирования будет конвертировано в язык программирования, не следует поддерживать отдельно документацию в виде описания на языке проектирования. Попытки поддерживать две идентичные копии описаний на языке проектирования, одна из которых содержит программную версию системы, может привести к разногласиям. Кроме того, можно в любой момент восстановить второй уровень документации из третьего путем выборочной печати только предложений на языке проектирования. Мы будем обсуждать это в гл. 7.

5.19. Упражнения

5.1. Конвертируйте процедуры системы управления телевизионным приемником с встроенным микрокомпьютером в язык высокого уровня типа PL/M.

5.2. Конвертируйте процедуры устройства управления уличным светофором с встроенным микрокомпьютером в язык высокого уровня типа PL/M.

5.3. Конвертируйте процедуры системы управления электронным спортивным табло в язык высокого уровня типа PL/M.

5.4. Конвертируйте процедуры терминала розничной торговли в язык высокого уровня типа PL/M.

Глава 6

Архитектура микрокомпьютера

ЧАСТЬ I. ЯЗЫК АССЕМБЛЕРА

В этой главе мы расширим наше представление о программных средствах до уровня языка ассемблера. Однако для правильного понимания языка ассемблера необходимо сначала ознакомиться с *архитектурой* микрокомпьютера. Для этой цели в качестве примера будет использоваться микрокомпьютер Intel 8085.

Как и в случае языка программирования высокого уровня, мы не будем пытаться полностью охватить все аспекты архитектуры микрокомпьютера или языка ассемблера. Несмотря на то что в некоторых случаях может оказаться необходимым конвертирование программ с языка проектирования в язык ассемблера, мы предполагаем, что в большинстве случаев программные средства должны конвертироваться в язык высокого уровня, например PL/M, и затем автоматически транслироваться в машинный язык с помощью компилятора. Но даже используя такой язык, как PL/M, часто бывает необходимо понимать, что делает микрокомпьютер, выполняя операции PL/M. Наше введение в архитектуру микрокомпьютера и язык ассемблера как раз и предназначено для того, чтобы читатель имел возможность ознакомиться с этими вопросами. Для более глубокого ознакомления с языком ассемблера и архитектурой микрокомпьютера читателю предлагается обратиться к литературе, указанной в конце книги.

6.1. Архитектура микрокомпьютера и язык ассемблера

Подобно тому как архитектура здания определяет, каким видит его наблюдатель, так архитектура микрокомпьютера определяет, каким видит микрокомпьютер тот, кто понимает язык ассемблера и кто проектирует его аппаратный интерфейс. Нас не будут интересовать кирпичи, строительный раствор, т. е. нас будет интересовать не то, как построить микрокомпьютер, а только то, как он работает в нашей системе. Поэтому мы должны знать не только, как он физически организован, но и какие операции он может выполнять. Рассмотрим вначале физическую сущность микрокомпьютера Intel 8085, а затем по

мере необходимости будем добавлять детали о том, как он функционирует.

Структурная схема физической части архитектуры микрокомпьютера Intel 8085 показана на рис. 6.1. Модуль ИНТЕРФЕЙСА ВВОДА состоит из *мультиплексора,* назначением которого является выбор из множества входов набора входных линий при выполнении команд ввода. Число или *адрес,* являющиеся частью любой команды ввода, определяют конкретный

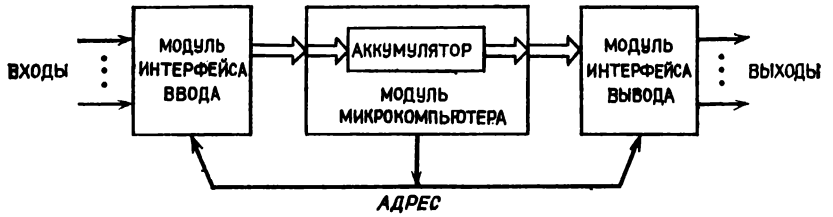


Рис. 6.1. Блок-схема архитектуры МИКРОКОМПЬЮТЕРА Intel 8085.

порт ввода или выбранный набор входных линий. Таким образом, команда ввода имеет следующую форму:

Прочитать порт ввода *n* и переслать считанные данные в АККУМУЛЯТОР.

Здесь *n* — *адрес* конкретного порта ввода. После того как выполнение команды ввода закончено, информация, считанная по выбранным линиям, содержится в специальной ячейке модуля МИКРОКОМПЬЮТЕРА, которая называется АККУМУЛЯТОРОМ. Модуль ИНТЕРФЕЙСА ВЫВОДА состоит из мультиплексора, который пересылает информацию, обработанную модулем МИКРОКОМПЬЮТЕРА, в выбранный *порт вывода* или набору выходных линий. Для определения выбранного порта вывода используется число или *адрес.* Команда вывода имеет вид:

Переслать данные из АККУМУЛЯТОРА в порт вывода *m.*

Здесь *m* — *адрес* конкретного порта вывода. Данные, пересылаемые в выбранный порт вывода, извлекаются из АККУМУЛЯТОРА модуля МИКРОКОМПЬЮТЕРА.

Модуль МИКРОКОМПЬЮТЕРА более детально показан на рис. 6.2. АРИФМЕТИЧЕСКО-ЛОГИЧЕСКОЕ УСТРОЙСТВО выполняет в микрокомпьютере арифметические операции. Оно способно выполнять сотни тысяч операций в секунду, и поэтому, чтобы поддерживать такую высокую скорость, команды и данные должны вводиться в него автоматически.

ПАМЯТЬ является местом хранения как данных, так и команд. Над ней можно производить операции со значительной

скоростью, соответствующей скорости работы АРИФМЕТИЧЕСКО-ЛОГИЧЕСКОГО УСТРОЙСТВА. Единицей информации в микрокомпьютере Intel 8085 является 8-битовый *байт*. В каждой *ячейке* ПАМЯТИ хранится один байт, который содержит либо команду, либо единицу данных, либо часть того или другого. Так как ПАМЯТЬ содержит много ячеек, для их различия используются *адреса*. *Адрес памяти* является числом, определяющим конкретную ячейку памяти таким же образом, каким адрес ввода или вывода определяет конкретный порт ввода или

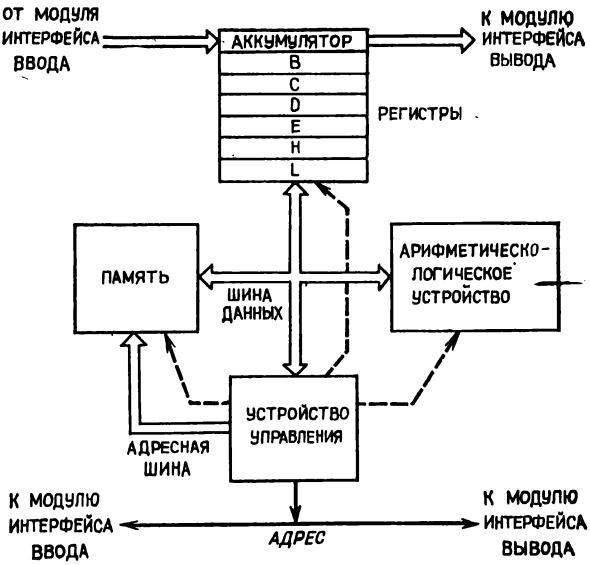


Рис. 6.2. Модуль МИКРОКОМПЬЮТЕРА Intel 8085.

вывода. АККУМУЛЯТОР функционально является ячейкой памяти, в которой может храниться байт данных, однако, так как он расположен вне ПАМЯТИ, при его использовании нет необходимости определять адрес. Следует заметить, что в микрокомпьютере Intel 8085, как и в большинстве других микрокомпьютеров, имеется несколько регистров и поэтому предусмотрены средства для обращения к каждому регистру в любой момент времени.

ШИНА ДАННЫХ — общий канал данных, через который информация передается между любыми блоками модуля МИКРОКОМПЬЮТЕРА. Следует отметить, что на рисунке каналы от модуля ИНТЕРФЕЙСА ВВОДА к АККУМУЛЯТОРУ и от него к ИНТЕРФЕЙСУ ВЫВОДА показаны отдельно от ШИНЫ ДАННЫХ. В действительности, ШИНА ДАННЫХ микро-

компьютера Intel 8085 объединяет все каналы. Однако для упрощения будем считать и далее в этой главе, что они разделены.

УСТРОЙСТВО УПРАВЛЕНИЯ предназначено для извлечения команд из ПАМЯТИ, для определения действий, необходимых для выполнения каждой команды, и для возбуждения линий управления (показанных на рис. 6.2 штрихами), чтобы обеспечить правильное выполнение каждой команды. В качестве примера иллюстрации работы УСТРОЙСТВА УПРАВЛЕНИЯ мы используем процедуру языка проектирования, рассмотренную в гл. 1 и воспроизведенную на рис. 6.3. Операция

ПРОВЕРИТЬ ВХОД 1 И ХРАНИТЬ ЕГО ЗНАЧЕНИЕ

является первой операцией в цикле процедуры. Для реализации этой операции нужны две команды языка ассемблера:

Прочитать порт ввода 1 и переслать прочитанные данные в АККУМУЛЯТОР.
Хранить данные, содержащиеся в АККУМУЛЯТОРЕ, в ячейке памяти, адрес которой задается величиной VALUE1.

Мы предположили, что состояние ВХОД 1 считывается через порт ввода 1 и что адрес памяти символически представлен величиной VALUE1. Возможно, что число, соответствующее адресу памяти, будет присвоено VALUE1. Таким образом, мы видим, что команды языка ассемблера определяют передачу информации в рамках архитектуры микрокомпьютера. Для удобства вместо команд ассемблера обычно используются их мнемонические аббревиатуры. Например, для микрокомпьютера Intel 8085 команды языка ассемблера, приведенные выше, принимают следующий вид:

```
IN   1
STA VALUE1
```

Сводка мнемонических команд ассемблера вместе с их интерпретацией приводится далее в этой главе на рис. 6.36. Читатель по мере необходимости может обращаться к этой сводке команд.

Сами команды также должны храниться в ПАМЯТИ. Допустим, что команды — для нашего примера — хранятся последовательно, начиная с ячейки памяти 2000. В этом случае команда ввода, требующая 16 бит, или 2 байта памяти, займет ячейки памяти 2000 и 2001; команда хранения, требующая 24 бита, или 3 байта памяти, займет ячейки 2002, 2003 и 2004. Чтобы показать, как команды распределены в ПАМЯТИ, адреса ячеек памяти, содержащих команды, размещаются в строке рядом с командами, как показано ниже:

```
2000 IN   1
2002 STA VALUE1
2005 ...
```

Хотя при написании программ на языке ассемблера нет необходимости принимать во внимание в явном виде адреса ячеек памяти, присутствие этих адресов рядом с командами позволит лучше понять работу УСТРОЙСТВА УПРАВЛЕНИЯ.

```
ВЫПОЛНЯТЬ НЕПРЕРЫВНО
  ПРОВЕРИТЬ ВХОД 1 И ХРАНИТЬ ЕГО ЗНАЧЕНИЕ
  ПРОВЕРИТЬ ВХОД 2 И ХРАНИТЬ ЕГО ЗНАЧЕНИЕ
  ЕСЛИ ЗНАЧЕНИЕ ВХОД 1 БОЛЬШЕ 4 И МЕНЬШЕ 8
    ТО УСТАНОВИТЬ ЗНАЧЕНИЕ ВЫХОД 1 РАВНО 6
    ИНАЧЕ УСТАНОВИТЬ ВЫХОД 1 РАВНО 0
  ЕСЛИ ЗНАЧЕНИЕ ВХОД 2 БОЛЬШЕ 2 И МЕНЬШЕ 6
    ТО УСТАНОВИТЬ ЗНАЧЕНИЕ ВЫХОД 2 РАВНО 4
    ИНАЧЕ УСТАНОВИТЬ ЗНАЧЕНИЕ ВЫХОД 2 РАВНО 0
КОНЕЦ
```

Рис. 6.3. Пример процедуры из гл. 1.

Во время выполнения первых двух команд в микрокомьютере имеют место следующие действия:

1. Команда ввода IN 1 считывается УСТРОЙСТВОМ УПРАВЛЕНИЯ из ячеек памяти 2000 и 2001. Этот шаг называется *циклом выборки*..

2. УСТРОЙСТВО УПРАВЛЕНИЯ посылает адрес порта ввода 1 в модуль ИНТЕРФЕЙСА ВВОДА и дает команду модулю ИНТЕРФЕЙСА ВВОДА выбрать порт ввода 1, прочитать логические состояния его входных линий и переслать байт данных, содержащий эти состояния, в АККУМУЛЯТОР.

3. УСТРОЙСТВО УПРАВЛЕНИЯ дает команду АККУМУЛЯТОРУ хранить информацию, полученную из модуля ИНТЕРФЕЙСА ВВОДА. Последние два шага составляют *исполнительный цикл.*

Этим завершается выполнение команды ввода. Отметим, что выполнение команды состоит из цикла выборки и исполнительного цикла.

4. Затем команда хранения STA VALUE1 считывается УСТРОЙСТВОМ УПРАВЛЕНИЯ из ячеек памяти 2002, 2003 и 2004.

5. УСТРОЙСТВО УПРАВЛЕНИЯ дает команду АККУМУЛЯТОРУ переслать копию данных в ПАМЯТЬ. При этой операции информация, хранящаяся в АККУМУЛЯТОРЕ, не разрушается.

6. УСТРОЙСТВО УПРАВЛЕНИЯ одновременно пересылает в ПАМЯТЬ адрес памяти, представленный VALUE1.

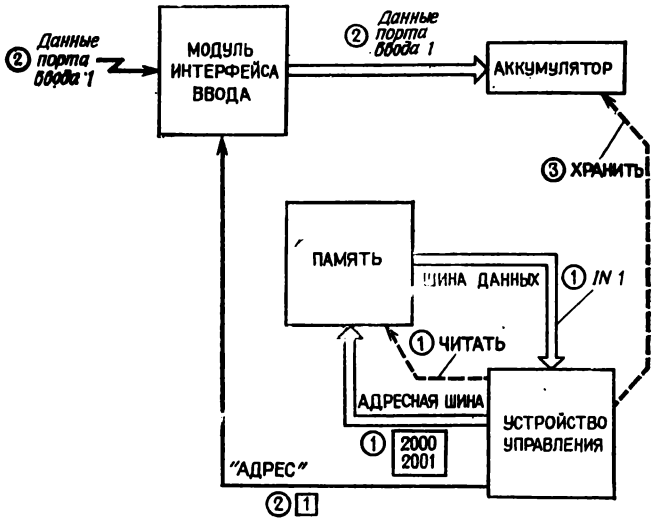


Рис. 6.4а. Выполнение команды ВХОД 1

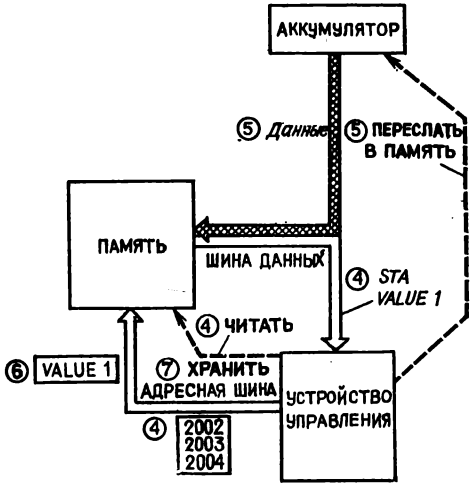


Рис. 6.4б. Выполнение команды ХРАНИТЬ ЗНАЧЕНИЕ 1.

7. УСТРОЙСТВО УПРАВЛЕНИЯ дает команду ПАМЯТИ хранить информацию, полученную из АККУМУЛЯТОРА, в ячейке, определенной адресом.

Выполнение этих двух команд, показывающее, как УСТРОЙСТВО УПРАВЛЕНИЯ манипулирует информацией, наглядно иллюстрируется на рис. 6.4а и рис. 6.4б Числа, обведенные кружком, соответствуют номерам шагов, описанных выше.

Адреса заключены в прямоугольные рамки, команды УСТРОЙСТВА УПРАВЛЕНИЯ, адресованные другим компонентам микрокомпьютера, выделены **жирным шрифтом**, а данные и команды, пересылаемые между компонентами микрокомпьютера, обозначены курсивом. Необходимо отметить, что на рис. 6.2 и 6.4 АДРЕСНАЯ ШИНА и ШИНА ДАННЫХ показаны как раздельные шины. Как будет видно позднее, физически они не разделены. Однако для упрощения будем считать здесь и далее в этой главе, что они являются разными шинами.

Подобно описанной выше, операция

ПРОВЕРИТЬ ВХОД 2 И ХРАНИТЬ ЕГО ЗНАЧЕНИЕ

эквивалентна двум командам языка ассемблера:

```
2005 IN  2
2007 STA VALUE2
2010 ...
```

Третья операция рассматриваемой процедуры является операцией тестирования:

ЕСЛИ ЗНАЧЕНИЕ ВХОД 1 БОЛЕЕ 4 И МЕНЕЕ 8,
ТО УСТАНОВИТЬ ЗНАЧЕНИЕ ВЫХОД 1 РАВНО 6
ИНАЧЕ УСТАНОВИТЬ ЗНАЧЕНИЕ ВЫХОД 1 РАВНО 0

Вспомним, что значение ВХОДА 1 хранится в ячейке памяти, символически представленной VALUE1. Таким образом, сначала необходимо считать значение ВХОДА 1 из ячейки памяти VALUE1 и сравнить его с числом 4. Это функционально эквивалентно четырем командам языка ассемблера:

```
2010 LDA VALUE1
2013 CPI 4
2015 JZ  ELSE1
2018 JC  ELSE1
2021 ...
```

Команда LDA VALUE1 подобна команде STA VALUE1. Однако данные при этом пересылаются в обратном направлении, т. е. *загружаются* в АККУМУЛЯТОР из ПАМЯТИ. Команда CPI 4 эквивалентна следующей команде:

Сравнить значение числа, хранящегося в АККУМУЛЯТОРЕ, с числом 4 и пометить результат установкой *флажков*.

Флажками называются биты, которые устанавливаются или сбрасываются для того, чтобы проследить за результатами операций. Например, если число в АККУМУЛЯТОРЕ равно 6, то во время операции сравнения флажки устанавливаются, чтобы показать, что 6 больше 4. Две следующие команды, JZ ELSE1— *Переход к ELSE1, если флажок нуля установлен*, и

JC ELSE1 — *Переход к ELSE1, если флажок переноса установлен* — проверяют эти флажки следующим образом:

Если флажок нуля установлен, указывая, что значение величины в АККУМУЛЯТОРЕ равно значению операнда сравнения (которое равно 4 в нашем примере), то следующей выполняется команда, которая идентифицируется меткой ELSE 1. Иначе выполняется команда JC ELSE 1, которая проверяет флажок переноса. Если флажок переноса установлен, указывая, что значение величины в АККУМУЛЯТОРЕ меньше значения операнда, следующей выполняется команда с меткой ELSE1. Иначе выполняется команда размещения в ячейке 2021.

Таким образом, в результате выполнения этих двух команд команда из ячейки 2021 выполняется только в том случае, если значение величины в АККУМУЛЯТОРЕ больше 4.

Для другой части условий операции из описания на языке проектирования используются следующие команды:

```
2021 CPI 8
2023 JZ  ELSE1
2026 JNC ELSE1
2029 ...
```

Выполнение этих команд аналогично описанному выше, за исключением команды JNC ELSE1 — *Переход к ELSE1, если флажок переноса не установлен.* В результате последовательность команд, начиная с ячейки 2029, выполняется только в том случае, если значение величины в АККУМУЛЯТОРЕ меньше 8 и больше 4. Последующие команды, следовательно, соответствуют той части условной операции УСТАНОВИТЬ ЗНАЧЕНИЕ ВЫХОД 1 РАВНО 6, которая начинается со слова ТО. Эти команды на языке ассемблера имеют следующий вид:

```
2029 MVI A, 6
2031 OUT 1
2033 JMP CONT1
2036 ...
```

Команда MVI A, 6 — *Переслать число* 6 *в АККУМУЛЯТОР* — помещает число 6 в АККУМУЛЯТОР. Команда OUT 1 эквивалентна следующей команде:

Переслать данные из АККУМУЛЯТОРА в порт вывода 1.

и, следовательно, число 6 пересылается в порт вывода 1. Команда JMP CONT1 — *Переход к CONT1* — необходима, чтобы избежать выполнения части условной операции, начинающейся со слова ИНАЧЕ, следующей за частью, начинающейся словом ТО. Она означает переход к выполнению команды с меткой CONT1 независимо от того, установлены флажки или нет.

```
                                ВЫПОЛНЯТЬ НЕПРЕРЫВНО
                                  ПРОВЕРИТЬ ВХОД 1 И ХРАНИТЬ
                                  ЕГО ЗНАЧЕНИЕ
  2000  START:  IN    1
  2002          STA   VALUE1
;                               ПРОВЕРИТЬ ВХОД 2 И ХРАНИТЬ
;                               ЕГО ЗНАЧЕНИЕ
  2005          IN    2
  2007          STA   VALUE2
;                               ЕСЛИ ЗНАЧЕНИЕ ВХОД 1 БОЛЬШЕ 4
;                               И МЕНЬШЕ 8
  2010          LDA   VALUE1
  2013          CPI   4
  2015          JZ    ELSE1
  2018          JC    ELSE1
  2021          CPI   8
  2023          JZ    ELSE1
  2026          JNC   ELSE1
;                                 ТО УСТАНОВИТЬ ЗНАЧЕНИЕ
;                                 ВЫХОД 1 РАВНО 6
  2029          MVI   A, 6
  2031          OUT   1
  2033          JMP   CONT1
;                                 ИНАЧЕ УСТАНОВИТЬ ЗНАЧЕНИЕ
;                                 ВЫХОД 2 РАВНО 0
  2036  ELSE1:  MVI   A, 0
  2038          OUT   1
;                               ЕСЛИ ЗНАЧЕНИЕ ВХОД 2 БОЛЬШЕ 2
;                               И МЕНЬШЕ 6
  2040 CONT1::  LDA   VALUE2
  2043          CPI   2
  2045          JZ    ELSE2
  2048          JC    ELSE2
  2051          CPI   6
  2053          JZ    ELSE2
  2056          JNC   ELSE2
;                                 ТО УСТАНОВИТЬ ЗНАЧЕНИЕ
;                                 ВЫХОД 2 РАВНО 4
  2059          MVI   A, 4
  2061          OUT   2
  2063          JMP   CONT2
;                                 ИНАЧЕ УСТАНОВИТЬ ЗНАЧЕНИЕ
;                                 ВЫХОД 2 РАВНО 0
  2066  ELSE2;  MVI   A, 0
  2068          OUT   2
;                               КОНЕЦ
  2070  CONT2:  JMP   START
  2073          ...
```

Рис. 6.5. Пример процедуры на языке ассемблера.

Отметим, что CONT 1 является сокращением от CONTINUE 1, поскольку ассемблер микрокомпьютера Intel 8085 ограничивает слова шестью символами, хотя многие ассемблеры ограничивают длину слова не столь строго.

Для части операции проверки, начинающейся словом ИНАЧЕ, требуются следующие команды:

```
2036 ELSE1: MVI A, 0
2038        OUT 1
2040 CONT1: ...
```

которые означают, что число 0 должно быть послано в порт вывода 1.

Оставшаяся часть процедуры состоит из аналогичных операций (рис. 6.5). Комментарии к процедурам языка ассемблера обозначаются точкой с запятой и могут быть продолжены до конца строки. Операции языка проектирования для нашего примера показаны на рис. 6.5 как комментарии.

При описании архитектуры микрокомпьютера Intel 8085 и в примерах использовался только один регистр — АККУМУЛЯТОР, однако, как упоминалось ранее, в микрокомпьютере Intel 8085 существуют другие регистры. Организация этих регистров показана на рис. 6.2. Некоторые из них являются регистрами общего назначения, как АККУМУЛЯТОР, другие имеют специальные функции. В следующем разделе будут описаны различные способы определения операндов или вычисления адресов памяти для микрокомпьютера Intel 8085. При этом можно будет увидеть, каково назначение каждого из регистров и как они используются.

6.2. Адресация

Команда должна либо содержать операнд, либо в ней должно быть указано, откуда можно получить данные, используемые в качестве операнда. С другой стороны, в случае использования команд хранения или пересылки в них должно быть определено, куда помещаются данные. В этом разделе будут рассмотрены и способы непосредственного определения операндов, и получение их путем вычислений адресов ячеек памяти, содержащих операнды и другую информацию.

Непосредственная адресация. В некоторых командах, например MVI A, 6, указанное в качестве операнда число используется непосредственно во время выполнения команды и не интерпретируется как адрес. Такой способ адресации называется *непосредственной адресацией*, так как число в команде находится на месте, обычно предназначенном для адреса, но

используется непосредственно вместо операнда. В зависимости от команды возможно использование как 8-битовых, так и 16-битовых непосредственных адресов. В примере MVI A, 6 число 6 хранится как 8-битовое число, поскольку для загрузки операнда предназначен 8-разрядный АККУМУЛЯТОР. В языке ассемблера микрокомпьютера Intel 8085 имеется только одна команда, использующая 16-битовый непосредственный адрес, при этом для загрузки операнда требуется пара регистров, содержащая 16 разрядов. В команде LXI H, 6 — *Загрузить число 6 в пару регистров H и L* — число 6 хранится как 16-битовое число, поскольку для загрузки операнда предназначена 16-разрядная пара регистров, включающая регистры H и L.

Прямая адресация. В некоторых командах 16-битовое число определяет действительный адрес памяти. Таким способом может быть определено до 65 536 различных ячеек ПАМЯТИ. Этот способ адресации называется *прямой адресацией.* Примерами команд, использующих прямую адресацию, служат команды LDA VALUE1 и JZ ELSE1. В первом примере символическим именем VALUE1 обозначен 16-битовый адрес ячейки памяти, содержащей используемый операнд. Во втором примере символическое имя ELSE 1 обозначает 16-битовый адрес ячейки памяти, содержащей команду, которая должна выполняться следующей, если флажок нуля установлен. В обоих случаях во время грансляции программы ассемблером в машинный язык символическое имя заменяется 16-битовым адресом. В нашем примере ELSE1 будет заменено числом 2036, представленным в виде 16-битового двоичного числа.

Несмотря на то что при прямой адресации для определения адреса всегда требуется 16-битовое число, размер операнда зависит от конкретной команды. Команда LDA VALUE1 означает, что в АККУМУЛЯТОР должен быть загружен один байт данных, хранящийся в ячейке памяти VALUE1. В языке ассемблера микрокомпьютера Intel 8085 существуют только две команды прямой адресации, определяющие 16-битовые операнды: LHLD — *Загрузить пару регистров HL из ПАМЯТИ* — и SHLD — *Хранить пару регистров HL в ПАМЯТИ.* По команде LHLD байт ячейки памяти, определенной прямым адресом, пересылается в регистр L, а байт ячейки памяти, адрес которой на единицу больше прямого адреса, пересылается в регистр H. Команда SHLD аналогична, за исключением того, что два байта пересылаются из пары регистров HL в ПАМЯТЬ.

Косвенная адресация. Следующий тип адресации, используемый в языке ассемблера микрокомпьютера Intel 1085, называется *косвенной адресацией.* В этом случае действительный адрес ячейки памяти, который должен быть использован в

команде, содержится в другой ячейке. В некоторых микрокомпьютерах второй ячейкой является ячейка памяти, и адрес этой *второй ячейки* определяется в команде. В микрокомпьютере Intel 8085 ячейкой памяти, содержащей действительный адрес, является одна из пар регистров BC, DE или HL, и поэтому нет необходимости явно указывать косвенный адрес. Примером команды, использующей косвенную адресацию, является команда MOV A, M, которая эквивалентна следующей команде:

Переслать данные, хранящиеся в ячейке памяти, адрес которой хранится в паре регистров HL, в АККУМУЛЯТОР.

Буква M в команде после запятой указывает, что пересылаемые данные должны быть извлечены из ячейки памяти. В языке ассемблера микрокомпьютера Intel 8085 второй элемент (M в нашем примере) всегда обозначает, откуда пересылаются данные, а первый (A в нашем примере) — куда. Отметим, что две команды

```
LXI  H, VALVE1
MOV  A, M
```

эквивалентны одной команде

```
LDA VALUE1
```

так как адрес, условно обозначенный как VALUE1, загружается в пару регистров HL командой LXI H, VALUE1 до того, как будет выполняться команда MOV A, M.

Команды LDAX B и LDAX D являются примерами косвенной адресации, когда действительный адрес находится в одной из пар регистров BC или DE. В любом случае данные пересылаются из ПАМЯТИ в АККУМУЛЯТОР, поскольку команды LDAX B и LDAX D эквивалентны следующим командам:

Загрузить АККУМУЛЯТОР данными из ячейки памяти, адрес которой содержится в парном регистре BC и DE.

Регистровая адресация. В языке ассемблера микрокомпьютера Intel 8085 имеется группа команд типа *регистр-в-регистр*, для которых не требуется адресов памяти. Примером команды типа регистр-в-регистр является команда пересылки MOV A, H. При выполнении команды данные из регистра H копируются в АККУМУЛЯТОР без изменения содержимого регистра H.

Индексация. Прежде чем закончить с вопросами адресации, рассмотрим, как используется в языке ассемблера микрокомпьютера Intel 8085 индекс цикла. Для этой цели воспользуемся процедурой СТИРАНИЯ ВХОДНОЙ ЗАПИСИ, введенной в гл. 4. Можно использовать один из двух способов: индекс

хранится в ПАМЯТИ и загружается каждый раз, когда это необходимо, или же индекс сразу загружается в регистр и вычисляется непосредственно в регистре. Второй способ иллюстрируется на рис. 6.6 для цикла на языке проектирования:

```
ВЫПОЛНИТЬ ДЛЯ КАЖДОГО ЭЛЕМЕНТА ВО ВХОДНОЙ ЗАПИСИ
  УСТАНОВИТЬ ЭЛЕМЕНТ В КОД "НУЛЯ"
КОНЕЦ
```

Отметим, что первые три команды на языке ассемблера инициализируют регистры: регистр C устанавливается в качестве счетчика, регистр B загружается символьным кодом нуля, а пара регистров HL загружается адресом первого байта

```
;                       ВЫПОЛНИТЬ ДЛЯ КАЖДОГО ЭЛЕМЕНТА
;                       ВО ВХОДНОЙ ЗАПИСИ
        MVI   C, 10
        MVI   B, '0'
        LXI   H, INREC
;                         УСТАНОВИТЬ ЭЛЕМЕНТ В КОД "НУЛЯ"
LOOP:   MOV   M, B
        INX   H
        DCR   C
;                       КОНЕЦ
        JNZ   LOOP
```

Рис. 6.6. Реализация на языке ассемблера циклической конструкции языка проектирования.

ВХОДНОЙ ЗАПИСИ. Во время выполнения цикла символьный код нуля пересылается из регистра B в ячейку памяти, адрес которой находится в паре регистров HL, значение адреса в паре регистров HL увеличивается на 1 командой INX H, а значение счетчика уменьшается на 1 командой DCR C. Команда сравнения не нужна, так как команда DCR устанавливает флажок нуля, если значение уменьшаемого числа станет равным нулю. Таким образом, пока счетчик не равен нулю, цикл повторяется. Читатель может сравнить реализацию цикла на языке ассемблера с реализацией того же цикла на языке PL/M, показанной на рис. 5.6.

В предыдущих примерах было показано, как микрокомпьютер Intel 8085 адресуется к ПАМЯТИ, пересылает информацию, выполняет проверки и исполняет команды ввода и вывода. Этим самым было показано, как работают РЕГИСТРЫ, ПАМЯТЬ, ШИНА ДАННЫХ и УСТРОЙСТВО УПРАВЛЕНИЯ. Ничего не было сказано, как действует АРИФМЕТИЧЕСКО-ЛОГИЧЕСКОЕ УСТРОЙСТВО, которое, как указыва-

лось ранее, выполняет в микрокомпьютере арифметические операции. Вернемся к рассмотрению АРИФМЕТИЧЕСКО-ЛОГИЧЕСКОГО УСТРОЙСТВА.

6.3. Арифметическо-логическое устройство

АРИФМЕТИЧЕСКО-ЛОГИЧЕСКОЕ УСТРОЙСТВО выполняет операции над данными, переданными ему из ПАМЯТИ или РЕГИСТРОВ через ШИНУ ДАННЫХ Оно пересылает результаты обратно в ПАМЯТЬ или в РЕГИСТРЫ также через ШИНУ ДАННЫХ. АРИФМЕТИЧЕСКО-ЛОГИЧЕСКОЕ

ADD	M	Сложить операнд, хранящийся в ячейке памяти, адрес которой хранится в паре регистров HL, с операндом в АККУМУЛЯТОРЕ.
SUB	D	Вычесть операнд, хранящийся в регистре D, из операнда в АККУМУЛЯТОРЕ.
ADI	20	Сложить число 20 с операндом в АККУМУЛЯТОРЕ.
ADC	M	Сложить операнд, хранящийся в ячейке памяти, адрес которой хранится в паре регистров HL, с операндом в АККУМУЛЯТОРЕ и прибавить значение флажка переноса к младшему биту суммы.
SBI	35	Вычесть число 35 из операнда в АККУМУЛЯТОРЕ, а также вычесть значение флажка переноса из младшего бита разности.

Рис. 6.7. Примеры команд сложения и вычитания микрокомпьютера Intel 8085.

УСТРОЙСТВО выполняет не только арифметические операции над данными, но может выполнять также и логические операции.

Арифметические команды микрокомпьютера Intel 8085 включают команды сложения и вычитания, а также различные вариации этих команд, используемых, как мы далее увидим, для специальных целей. В языке ассемблера микрокомпьютера Intel 8085 нет команд умножения и деления, хотя такие команды и существуют в других микрокомпьютерах. Далее будет кратко описано, как в микрокомпьютере Intel 8085 можно использовать команду сложения для реализации алгоритма умножения.

За некоторым исключением, все команды сложения и вычитания включают использование АККУМУЛЯТОРА. Двумя основными командами сложения и вычитания являются следующие команды:

Сложить операнд с содержимым АККУМУЛЯТОРА
Вычесть операнд из содержимого АККУМУЛЯТОРА.

Прибавляемый или вычитаемый операнд может находиться в регистре, в ПАМЯТИ или может быть определен в самой команде как непосредственный адрес. Если он находится в ПАМЯТИ, применяется косвенная адресация с использованием пары регистров HL. Прямая адресация для команд сложения и

вычитания в языке ассемблера микрокомпьютера Intel 8085 не предусмотрена. Примеры команд сложения и вычитания показаны на рис. 6.7. Команды, прибавляющие или вычитающие значения флажка переноса, могут быть использованы для сложения и вычитания операндов, длина которых более одного байта. В случае вычитания флажок переноса представляет скорее *заем*, а не перенос. Поэтому SBI означает *Вычитание с займом* (*Subtract with borrow*). На рис. 6.8 показано, как в языке ассемблера микрокомпьютера Intel 8085 используется команда ADC для выполнения многобайтного сложения.

```
                              ОЧИСТИТЬ ФЛАЖОК ПЕРЕНОСА
        SUB   A
;                             ВЫПОЛНИТЬ ДЛЯ КАЖДОГО БАЙТА
        LXI   D, OP1 + 3
        LXI   H, OP2 + 3
        MVI   C, 4
LOOP:   LDAX  D
;                               СЛОЖИТЬ СЛЕДУЮЩИЕ БАЙТЫ
;                               ОПЕРАНДА С ФЛАЖКОМ ПЕРЕНОСА
        ADC   M
        STAX  D
        DCX   D
        DCX   H
        DCR   C
;                             КОНЕЦ
        JNZ   LOOP
```

Рис. 6.8. Пример многобайтного сложения на языке ассемблера.

В этом примере два 32-битовых числа хранятся в ячейках памяти OP1, OP1 + 1, OP1 + 2, OP1 + 3 и OP2, OP2 + 1, OP2 + 2, OP2 + 3. Младшие байты каждого числа хранятся в ячейках памяти OP1 + 3 и OP2 + 3, а старшие — в ячейках памяти OP1 и OP2. Регистр C используется в качестве счетчика, а пары регистров DE и HL используются для косвенной адресации. Команда SUB A используется для очистки флажка переноса, так как он не возбуждается при вычитании числа из самого себя. Внутри цикла байты складываются, начиная с младших, один с другим и с флажком переноса, чтобы обеспечить перенос между байтами. Результат находится в ячейках памяти, начиная с OP1.

Чтобы расширить пример до многобайтных чисел любой длины, необходимо изменить только три команды инициализации. Цикл при этом не меняется. Таким же образом, заменив

команду ADC на SBB[1]) или *Вычитание с займом*, можно выполнять многобайтное вычитание. Читатель должен отметить, что команды уменьшения на 1 изменяют флажок переноса, и поэтому он не может быть изменен между двумя последовательными выполнениями команд ADC или SBB в цикле.

Команда DAD является единственной командой сложения, которая не использует АККУМУЛЯТОР. Она эквивалентна следующей команде:

Прибавить содержимое конкретной пары регистров к паре регистров HL.

Поскольку здесь складываются операнды длиной два байта, DAD означает *Сложение удвоенной длины* (*Double length add*).

Для однобайтных операндов команда ADD A удваивает значение 8-битового числа в АККУМУЛЯТОРЕ. Аналогичное действие выполняет команда DAD H с 16-битовыми числами в паре регистров HL. В обоих случаях это эквивалентно сдвиганию каждого бита операнда влево на один разряд, а старшего бита — во флажок переноса. Это свойство будет использовано в процедуре умножения, показанной на рис 6.9.

В этом примере предполагается, что множимое и множитель являются 8-битовыми положительными числами. Предполагается, что множитель находится в регистре H, а множимое — в регистре C. Регистр E используется в качестве счетчика, а пара регистров HL используется для накапливания частичной суммы произведения по мере ее формирования. Следует отметить, что множитель постепенно сдвигается за пределы регистра H, освобождая место для частичной суммы произведения по мере ее формирования в паре регистров HL. Для этой цели используется команда DAD H. Команда DAD B используется для сложения множимого, представленного 16-битовым числом, с частичной суммой произведения при условии, что очередной бит множителя, сдвинутый во флажок переноса, не равен нулю.

Другими арифметическими командами микрокомпьютера Intel 8085, использующими АРИФМЕТИЧЕСКО-ЛОГИЧЕСКОЕ УСТРОЙСТВО, являются команды увеличения и уменьшения на 1. Мы уже рассмотрели много примеров их использования и поэтому больше не будем касаться этого вопроса. Команды сравнения идентичны командам вычитания, при этом

[1]) Команда языка ассемблера микрокомпьютера Intel 8085 SBB отличается от команды SBI тем, что операндом является ячейка памяти, адрес которой хранится в паре регистров HL, в то время как операндом команды SBI является непосредственное число, не превышающее 256. Из этого следует, что команда SBB использует косвенную адресацию, в то время как SBI — непосредственную. — *Прим. перев.*

также происходит вычитание, однако во внимание принимается не результат вычитания, а только состояние флажка. Мы несколько раз использовали команду непосредственного сравнения CPI. Другая команда сравнения, CMP, сравнивает байт одного из РЕГИСТРОВ или байт ПАМЯТИ с байтом АККУМУЛЯТОРА. Для всех команд сравнения при равенстве байтов устанавливается флажок нуля, а если сравниваемый байт

```
;                          ПРЕОБРАЗОВАТЬ МНОЖИМОЕ
;                          В 16-БИТОВОЕ ЧИСЛО
          MVI   B, 0
                           ИНИЦИАЛИЗИРОВАТЬ ЧАСТИЧНУЮ СУММУ
;                          ПРОИЗВЕДЕНИЯ
          MVI   L, 0
;                          ВЫПОЛНИТЬ ДЛЯ КАЖДОГО БИТА
;                          МНОЖИТЕЛЯ
          MVI   E, 8
;                            СДВИНУТЬ ЧАСТИЧНУЮ СУММУ
;                            ПРОИЗВЕДЕНИЯ И МНОЖИТЕЛЬ ВЛЕВО
                             НА ОДИН БИТ
  LOOP:   DAD   H
;                            ЕСЛИ СТАРШИЙ БИТ МНОЖИТЕЛЯ
;                            РАВЕН ЕДИНИЦЕ
          JNC   SKIP
;                              ТО СЛОЖИТЬ МНОЖИМОЕ
;                              С ЧАСТИЧНОЙ СУММОЙ
                               ПРОИЗВЕДЕНИЯ
          DAD   B
;                          КОНЕЦ
  SKIP:   DCR   E
          JNZ   LOOP
```

Рис. 6.9. Пример умножения на языке ассемблера.

больше байта АККУМУЛЯТОРА, устанавливается флажок переноса. Отметим, что оба байта при выполнении команд сравнения рассматриваются как положительные числа.

Этим заканчивается описание арифметических команд, выполняемых АРИФМЕТИЧЕСКО-ЛОГИЧЕСКИМ УСТРОЙСТВОМ. Перейдем к рассмотрению логических команд.

6.4. Логические команды

Логические команды микрокомпьютера Intel 8085 можно разбить на три группы: команды с двумя операндами, однооперандные и команды флажка переноса.

Команды с двумя операндами. Команды *с двумя операндами* обрабатывают данные, содержащиеся в АККУМУЛЯТОРЕ, как массив битов в соответствии с информацией, содержащейся во втором операнде. В приведенной ниже таблице истинности показано, как устанавливается значение каждого бита в результате выполнения команд ANA — *И с АККУМУЛЯТОРОМ (AND with ACCUMULATOR REGISTER)* и ORA — *ИЛИ с АККУМУЛЯТОРОМ (OR with ACCUMULATOR REGISTER)*.

Бит в АККУМУЛЯТОРЕ	Бит во втором операнде	Бит в И результате	Бит в ИЛИ результате
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Имеются версии этих команд с непосредственной адресацией, а именно ANI и ORI. Рассмотрим примеры, иллюстрирующие их использование.

Если на клавиатуре нажать кнопку, то при этом вырабатывается символьный код. Ранее было показано, как работать непосредственно с символьными кодами. Однако удобнее преобразовать символьные коды, представляющие цифровые клавиши, в числа, которые они представляют, чтобы можно было производить с ними арифметические операции. Для такого преобразования может быть использована команда И, поскольку каждый цифровой символьный код представляет 8-битовый код, младшие четыре бита которого эквивалентны численному значению символа. Это можно видеть из таблицы символьных кодов для цифр от 0 до 9:

'0'	0011 0000	'5'	0011 0101
'1'	0011 0001	'6'	0011 0110
'2'	0011 0010	'7'	0011 0111
'3'	0011 0011	'8'	0011 1000
'4'	0011 0100	'9'	0011 1001

Представление, используемое в таблице, является частью Стандартного американского кода для обмена информацией ASCII (American Standard Code for Information Interchange).

Для преобразования этих символьных кодов в числа старшие четыре бита должны быть заменены на нули, а остальные — остаться неизмененными. Если символьный код ASCII находится в АККУМУЛЯТОРЕ, то командой ANI 15 он преобразуется в эквивалентную числовую величину, поскольку десятичное число 15 равно двоичному 0000 1111.

Обратная операция преобразования 4-битового числа в символьный код ASCII может быть выполнена с помощью команды ИЛИ. Так как десятичное число 48 равно двоичному 0011 0000, 4-битовые числа от 0 до 9, находящиеся в АККУМУЛЯТОРЕ, могут быть преобразованы в соответствующий символьный код ASCII с помощью команды ORI 48. Последняя операция является полезной для преобразования числа в символьный код с целью вывода на устройство, воспринимающее символьные коды ASCII.

Команды с одним операндом. Логические команды *с одним операндом* являются командами циклического сдвига, которые перемещают данные, находящиеся в АККУМУЛЯТОРЕ, внутри регистра. В частности, каждый бит перемещается влево или вправо на один разряд. Бит, сдвигаемый за пределы крайнего левого или правого разряда, перемещается на освободившееся место с другого конца регистра, а также во флажок переноса. В других вариантах команд циклического сдвига флажок переноса присоединяется к АККУМУЛЯТОРУ во время выполнения команды и рассматривается так, как будто он является частью регистра. Командами циклического сдвига являются следующие команды:

RLC Сдвинуть циклически АККУМУЛЯТОР влево.
RRC Сдвинуть циклически АККУМУЛЯТОР вправо.
RAL Сдвинуть циклически АККУМУЛЯТОР влево с присоединенным флажком переноса.
RAR Сдвинуть циклически АККУМУЛЯТОР вправо с присоединенным флажком переноса.

Команды И и ИЛИ используются вместе с командами циклического сдвига для преобразования символьных кодов в 4-битовые числа и *упаковки* двух таких чисел в байт. Эти команды используются также для *распаковки* чисел и преобразования их в символьные коды. Используемые для этих целей процедуры языка ассемблера показаны на рис. 6.10 и 6.11. Рис. 6.12 иллюстрирует пошаговое выполнение процедуры упаковки. Предполагается, что вначале два символьных кода хранятся в регистрах C и D, а упакованные числа хранятся в регистре C. В примере с распаковкой предполагается, что упакованные числа находятся в регистре C, а распакованные коды хранятся в регистрах C и D.

Команды флажка переноса. Команды флажка переноса используются для того, чтобы *Установить флажок переноса*— STC (*Set the carry flag*) и *Дополнить флажок* (*изменить состояние флажка*) *переноса* — CMC (*Complement the carry flag*). В одном из предыдущих примеров на рис. 6.8 использовалась команда SUB A для сброса флажка переноса. Однако

```
;                ПРЕОБРАЗОВАТЬ ПЕРВЫЙ СИМВОЛЬНЫЙ КОД
;                В ЧЕТЫРЕХБИТОВОЕ ЧИСЛО
  MOV   A, C
  ANI   15
;                ПЕРЕСЛАТЬ ЧЕТЫРЕХБИТОВОЕ ЧИСЛО В ДРУГУЮ
;                ПОЛОВИНУ АККУМУЛЯТОРА
  RLC
  RLC
  RLC
  RLC
;                ВРЕМЕННО ХРАНИТЬ ЧИСЛО
  MOV   C, A
;                ПРЕОБРАЗОВАТЬ ВТОРОЙ СИМВОЛЬНЫЙ КОД
;                В ЧЕТЫРЕХБИТОВОЕ ЧИСЛО
  MOV   A, D
  ANI   15
;                УПАКОВАТЬ ОБА ЧИСЛА ВМЕСТЕ
  ORA   C
;                ХРАНИТЬ УПАКОВАННЫЕ ЧИСЛА
  MOV   C, A
```

Рис. 6.10. Преобразование символьных кодов в четырехбитовые числа и упаковка чисел в байты.

```
;                УДАЛИТЬ ЛЕВОЕ ИЗ УПАКОВАННЫХ ЧИСЕЛ
  MOV   A, C
  ANI   15
;                ПРЕОБРАЗОВАТЬ ПРАВОЕ ЧИСЛО В СИМВОЛЬНЫЙ
;                КОД
  ORI   48
;                ХРАНИТЬ СИМВОЛЬНЫЙ КОД
  MOV   D, A
;                УДАЛИТЬ ПРАВОЕ ИЗ УПАКОВАННЫХ ЧИСЕЛ
  MOV   A, C
  ANI   240
;                ПЕРЕМЕСТИТЬ ЛЕВОЕ ЧИСЛО В ДРУГУЮ ПОЛОВИНУ
;                АККУМУЛЯТОРА
  RLC
  RLC
  RLC
  RLC
;                ПРЕОБРАЗОВАТЬ ЧИСЛО В СИМВОЛЬНЫЙ КОД
  ORI   48
;                ХРАНИТЬ СИМВОЛЬНЫЙ КОД
  MOV   C, A
```

Рис. 6.11. Распаковка четырехбитовых чисел и преобразование их в символьные коды.

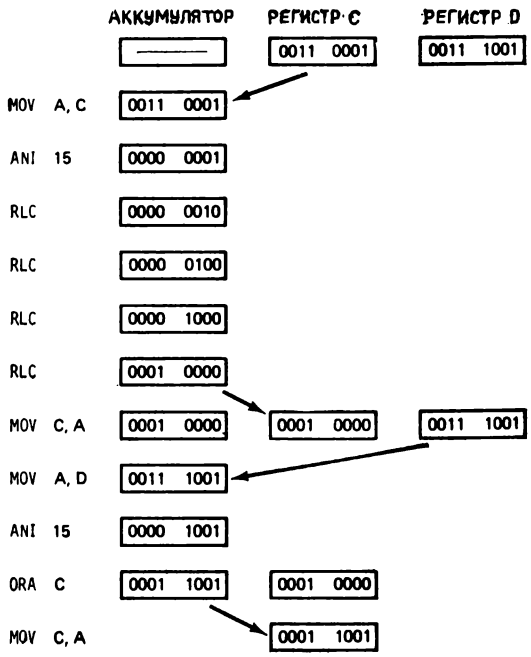


Рис. 6.12. Пошаговая иллюстрация упаковки.

вычитание очищает также АККУМУЛЯТОР. Чтобы очистить флажок переноса, не затрагивая АККУМУЛЯТОР, можно использовать команду STC и следом за ней команду CMC. Таким образом, с помощью этих двух команд можно управлять флажком переноса.

Мы уже использовали несколько команд перехода языка ассемблера микрокомпьютера Intel 8085. Теперь рассмотрим все управляющие команды.

6.5. Команды управления

Команды управления языка ассемблера микрокомпьютера Intel 8085 включают команды перехода, а также команды вызова и возврата. Команды каждого из этих трех типов можно подразделить на безусловные, т. е. выполняемые по мере того, как они встречаются, и условные, которые выполняются при определенных условиях. Условия проверки определяются (рис. 6.13) флажками переноса, нуля, знака и четности.

Безусловными командами управления являются команды *перехода* JMP (*jump*), *вызова* CALL и *возврата* RET (*re-*

turn). Условные команды управления обозначаются в языке ассемблера микрокомпьютера Intel 8085 начальными буквами безусловных команд J (jump), C (call) или R (return), после которых ставятся одна или две буквы, определяющие условие следующим образом:

C	Команда выполняется, если флажок переноса установлен.
NC	Команда выполняется, если флажок переноса *не* установлен.
Z	Команда выполняется, если флажок нуля установлен.
NZ	Команда выполняется, если флажок нуля *не* установлен.
M	Команда выполняется, если бит знака установлен, т. е. если результат отрицательный.
P	Команда выполняется, если бит знака *не* установлен, т. е. если результат положительный.
PE	Команда выполняется, если флажок четности установлен.
PO	Команда выполняется, если флажок четности *не* установлен.

Таким образом, JZ, JC и JNC означают *Переход, если флажок нуля установлен* (*Jump if the zero flag is set*), *Переход, если флажок переноса установлен* (*Jump if the carry flag is set*), и *Переход, если флажок переноса не установлен* (*Jump if the carry flag is not set*), соответственно. Аналогично CZ, CC и CNC означают *Вызов процедуры, если флажок нуля установлен* (*Call the specified procedure if the zero flag is set*), *Вызов процедуры, если флажок переноса установлен* (*Call the specified procedure if the carry flag is set*), и *Вызов процедуры, если флажок переноса не установлен* (*Call the specified procedure if the carry flag is not set*).

Флажок переноса	Флажок переноса устанавливается, чтобы показать, что имел место перенос или заем во время выполнения команд сложения, вычитания или сложения чисел удвоенной длины. Команда циклического сдвига может переслать ЕДИНИЧНЫЙ бит во флажок переноса, тем самым устанавливая флажок. Флажок переноса всегда сбрасывается при выполнении команд И/ИЛИ.
Флажок нуля	Флажок нуля устанавливается, чтобы показать, что получен нулевой результат при выполнении команд сложения, вычитания, сравнения, И, ИЛИ, увеличения или уменьшения байта на единицу.
Флажок знака	Флажок знака устанавливается, чтобы показать, что получен отрицательный результат при выполнении команд сложения, вычитания, сравнения, И, ИЛИ, увеличения или уменьшения байта на единицу.
Флажок четности	Флажок четности устанавливается, чтобы показать, что результат выполнения команд сложения, вычитания, сравнения, И, ИЛИ, увеличения или уменьшения байта на единицу имеет четное число ЕДИНИЧНЫХ битов.

Рис. 6.13. Условия, влияющие на состояние флажков микрокомпьютера Intel 8085.

Кроме того, RZ и RC означают *Возврат в вызывающую процедуру, если флажок нуля установлен* (*Return to calling procedure if the zero flag is set*), и *Возврат в вызывающую процедуру, если флажок переноса установлен* (*Return to calling procedure if the carry flag is set*).

Примером команды условного возврата является следующая конструкция на языке проектирования:

```
ЕСЛИ ПЕРЕКЛЮЧАТЕЛЬ ОТКЛЮЧЕН
  ТО ВОЗВРАТ
```

Предположим, как это уже делалось в гл. 5, что переменная ПЕРЕКЛЮЧАТЕЛЬ (KEYSW) равна нулю, когда ПЕРЕКЛЮЧАТЕЛЬ находится в отключенном состоянии. Тогда предыдущая конструкция языка проектирования эквивалентна следующим командам на языке ассемблера:

```
;                ЕСЛИ ПЕРЕКЛЮЧАТЕЛЬ ОТКЛЮЧЕН
  LDA  KEYSW
  CPI  0
;                ТО ВОЗВРАТ
  RZ
```

Значение KEYSW помещается в АККУМУЛЯТОР и сравнивается с нулем, при этом флажок нуля устанавливается, если значение KEYSW равно нулю. Команда *возврата* выполняется только в случае, если флажок нуля установлен, что означает, что переключатель отключен.

Этим завершается в основном наше введение в архитектуру микрокомпьютера Intel 8085 и язык ассемблера. Однако для понимания того, как микрокомпьютер в действительности выполняет команды управления и как используются прерывания, мы должны знать, как действует стек. Рассмотрим это в следующем разделе.

6.6. Стек

Стек представляет специально выделенный участок памяти, используемый прежде всего для хранения информации, связанной с вызовом процедур. Мы сначал опишем, как работает стек, затем — как он реализован в микрокомпьютере Intel 8085, и наконец, как он используется во время функционирования системы.

Функционально стек является памятью *магазинного типа* (LIFO — *last in-first out*). Он не требует явной адресации, а для операций со стеком используются две основные команды PUSH и POP (последняя иногда называется PULL). Данные заносятся в стек по команде PUSH, а извлекаются из него по команде POP. Операции со стеком показаны на рис. 6.14. Когда

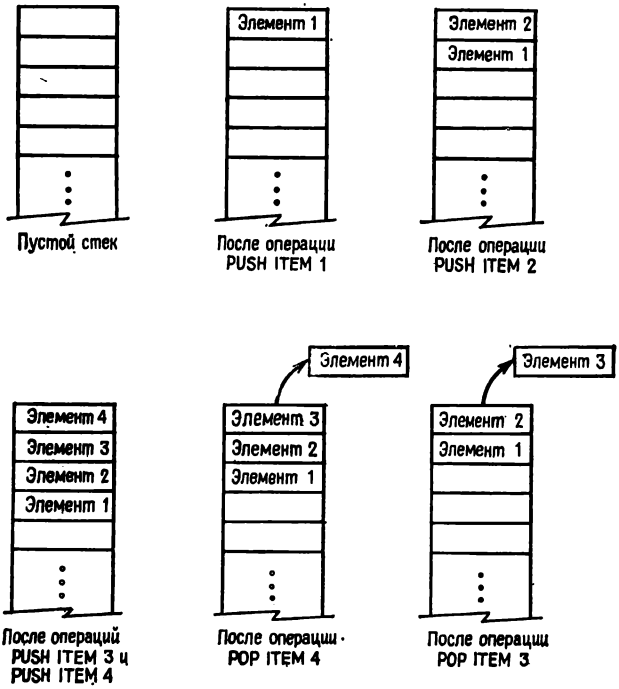


Рис. 6.14. Функциональные операции со стеком

очередной элемент «заталкивается» (push) в стек, все элементы, находящиеся ниже его, перемещаются *вниз* на один уровень. Если же элемент «выталкивается» (pop) из вершины стека, все нижние элементы перемещаются на один уровень *вверх*. Таким образом, информация хранится в стеке в том порядке, в котором она туда поступает, а извлекается в обратном порядке.

Во время работы микрокомпьютерной системы операции ВЫЗОВА и ВОЗВРАТА применяются для начала и остановки исполнения вызываемой процедуры. Так как процедура может быть вызвана из разных мест различных процедур, необходимо сохранить информацию о том, откуда была вызвана процедура. Это дает возможность начать выполнение операции, следующей за операцией ВЫЗОВА, сразу же после того, как в вызываемой процедуре встретится операция ВОЗВРАТА. На рис. 6.15 показан пример трехкратного вызова процедуры другой процедурой. Стрелками показана последовательность передачи управления от вызывающей процедуры к вызываемой и наоборот. Стек предусматривает соответствующий способ хранения информации, необходимой для правильной передачи управле-

ния при выполнении операции ВОЗВРАТА. Это осуществляется следующим образом: операция ВЫЗОВА эквивалентна двум операциям:

ЗАНЕСТИ в стек указатель операции, следующей за операцией CALL.
Начать исполнение вызванной процедуры.

а операция ВОЗВРАТА эквивалентна также двум операциям:

ИЗВЛЕЧЬ указатель из стека.
Исполнить операцию, обозначенную указателем.

На рис. 6.16 показаны манипуляции со стеком и указателями.

Мы уже видели, что во время работы системы вызовы процедур могут быть *вложенными*. Это означает, что вызванная

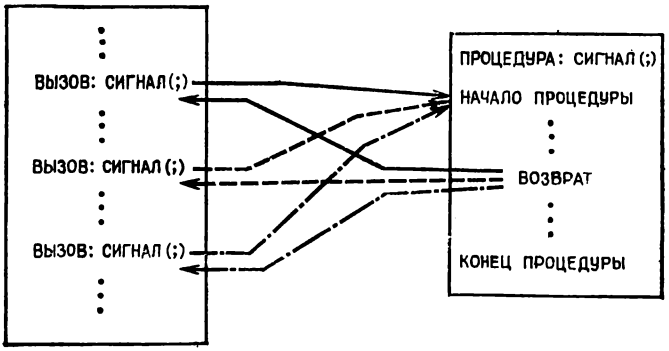


Рис. 6.15. Пример, иллюстрирующий многократное выполнение процедуры.

процедура может вызывать другие процедуры. На рис 6.17 показано, как прослеживается информация о вызовах вложенных процедур во время работы системы охранной сигнализации. Следует отметить, что каждый раз указатель в вершине стека обеспечивает надлежащее продолжение исполнения вызывающей процедуры после выполнения операции ВОЗВРАТА в вызываемой процедуре.

Представленное здесь описание иллюстрирует принципиальную схему работы стека. На практике стек работает несколько иначе по двум причинам:

- Каждый раз, когда исполняются операции PUSH или POP, все данные, хранимые в стеке, должны перемещаться вниз или вверх. Это требование неоправданно увеличивает затраты времени и усложняет аппаратуру.
- Размеры стека не могут быть неограниченными, как до сих пор предполагалось.

В микрокомпьютере Intel 8085, как и в большинстве других микрокомпьютеров с встроенной реализацией стека, операции со стеком моделируются простым способом. Для стека выделен отдельный раздел ПАМЯТИ и предусмотрен *указатель стека.* Указатель стека в микрокомпьютере Intel 8085 хранится в 16-разрядном *регистре указателя стека*, функцией которого явля-

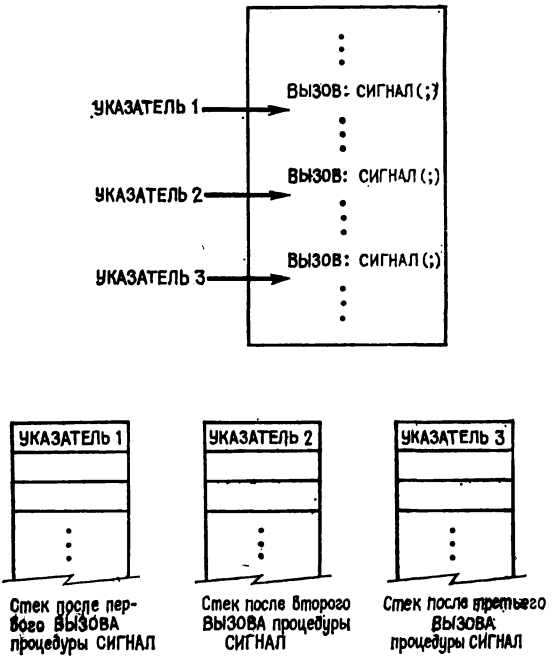


Рис. 6.16. Манипулирование со стеком и указателями при многократном выполнении процедуры.

ется слежение за текущим расположением *вершины* стека. Каждый раз при выполнении команд PUSH и POP указатель стека устанавливается соответствующим образом. Хотя ПАМЯТЬ микрокомпьютера Intel 8085 организована побайтно, стек всегда манипулирует порциями информации длиной 16 бит. Эта информация содержит либо адрес, используемый в качестве указателя командой RET, либо данные, полученные из пары регистров. Таким образом, каждая операция со стеком состоит из двух шагов (рис. 6.18). В этом примере команда PUSH B пересылает в стек два байта из пары регистров BC, а команда PUSH D пересылает в стек два байта из пары регистров DE. Затем команда POP B пересылает последние два

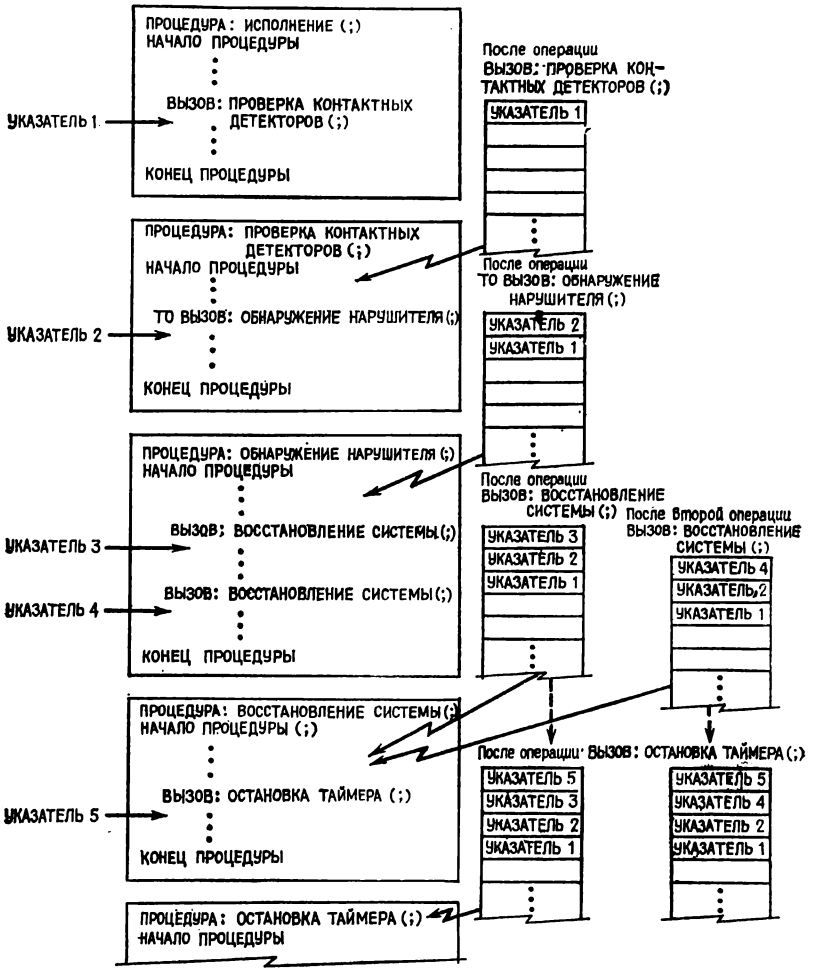


Рис. 6.17. Вызовы вложенных процедур.

байта из стека в пару регистров BC, а команда POP D пересылает предыдущие два байта в пару регистров DE. Конечным результатом этого примера является перестановка данных в парах регистров.

Микропроцессор Intel 8085 включает следующие команды, использующие стек или манипулирующие с указателем стека: CALL, RET, PUSH, POP, SPHL или *Загрузить УКАЗАТЕЛЬ СТЕКА из пары регистров HL* (*Load the STACK POINTER from the HL register pair*) и LXI SP или *Загрузить УКАЗА-*

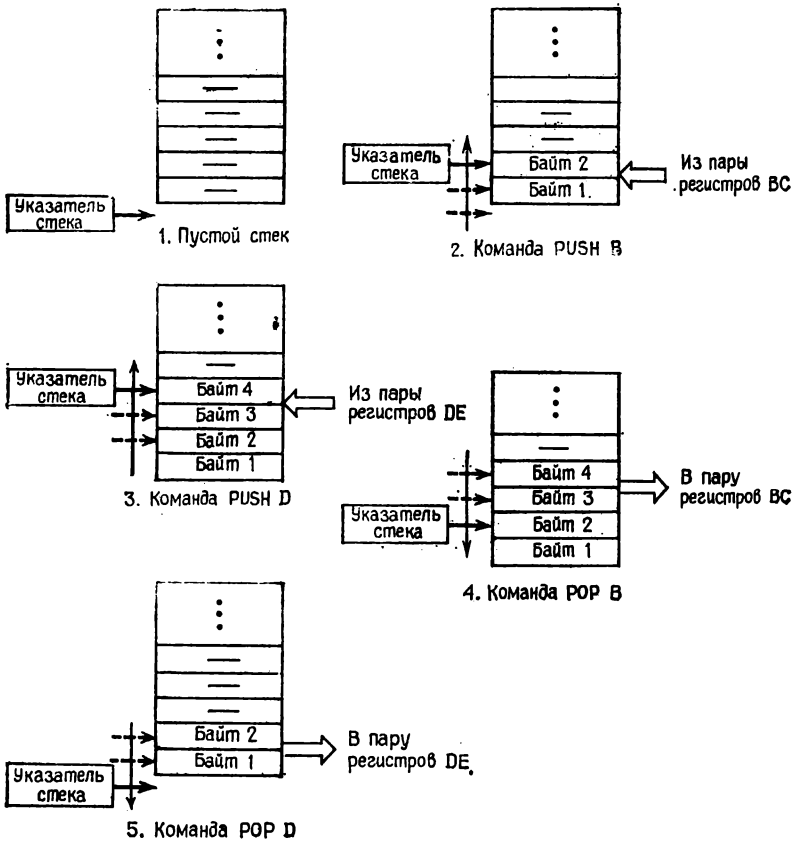


Рис. 6.18. Имитация операций со стеком.

ТЕЛЬ СТЕКА непосредственно (Load the STACK POINTER immediate). Последние две команды используются для инициализации указателя стека перед использованием стека. Инициализация указателя стека обеспечивает правильное использование выделенного для стека участка ПАМЯТИ. Таким образом, указатель стека необходимо инициализировать *прежде*, чем в ИСПОЛНИТЕЛЬНОЙ процедуре встретится первая команда CALL. По этой причине операция инициализации указателя стека должна находиться в ИСПОЛНИТЕЛЬНОЙ процедуре, а *не* в процедуре ИНИЦИАЛИЗАЦИИ СИСТЕМЫ, как предполагалось в примерах гл. 4. Читатель, наверное, уже отметил, что стековые команды функционируют попарно. Так, каждая команда CALL сочетается с командой RET, а каждая команда PUSH сочетается с командой POP. Такое *парное сочетание*

обычно проявляется естественным образом. Однако ошибки при проектировании или программировании могут нарушить парность, в результате чего может возникнуть одна из двух аварийных ситуаций:

• Команда RET может использовать неправильный указатель или данные, не являющиеся действительным адресом указателя.

• Указатель стека может быть установлен так, что он будет указывать на ячейку вне выделенного для стека участка памяти. Последнее может быть усугублено тем, что в результате непредусмотрительности стек окажется заполненным и будет сделана попытка выполнить команду PUSH или CALL. Микропроцессор Intel 8085 не предусматривает способов автоматического обнаружения таких ситуаций с целью их устранения. Читатель должен знать, что такие ситуации могут встретиться.

Стек используется также для передачи входных параметров процедурам, для возврата выходных параметров из процедур и для облегчения управления системой во время прерываний. Мы исследуем эти концепции в следующем разделе.

6.7. Передача параметров

Мы уже видели, что функционирование подсистем, содержащих микрокомпьютеры, сопровождается передачей входных и выходных параметров между процедурами. При использовании PL/M программист не должен задумываться над тем, как это происходит, так как это возложено на компилятор. Если же используется ассемблер, то ответственность за организацию и управление параметрами возлагается на программиста. С целью упрощения использования параметров фирма Intel установила серию стандартов, обеспечивающих единообразие при передаче параметров. Программист должен знать и использовать эти стандарты, чтобы процедуры, написанные с помощью языка ассемблера и PL/M, использовались одинаковым способом и могли вызывать одна другую. Описанные ниже стандарты применимы только к программам, разработанным с использованием системы разработки программ для микрокомпьютеров фирмы Intel (Intel Microcomputer Development System) и операционной системы ISIS. В других системах используются другие соглашения.

Выходные параметры передаются через регистры. Данные, являющиеся выходными параметрами, пересылаются в АККУМУЛЯТОР, если это байтовая переменная или константа, и в регистр HL, если это адресная переменная или константа. Вызывающая процедура затем извлекает данные из соответствую-

щего регистра или пары регистров сразу же после выполнения команды RET. На рис. 6.19 показано, как передаются выходные параметры из процедуры ПРОВЕРКИ ЗАПОЛНЕНИЯ

```
        .
        .
        .
LDA  STATUS
RET
        .
        .
        .
```

Рис. 6.19а. Команды языка ассемблера, реализующие PL/M-операцию RETURN STATUS; в процедуре ПРОВЕРКИ ЗАПОЛНЕНИЯ ВХОДНОЙ ЗАПИСИ (TINREC).

```
        .
        .
        .
CALL  TINREC
STA   STATUS
        .
        .
        .
```

Рис. 6.19б. Команды языка ассемблера, реализующие PL/M-операцию STATUS = TESTIFIF$INPUT$RECORD$FULL; в вызывающей процедуре.

ВХОДНОЙ ЗАПИСИ (TINREC — TEST IF INPUT RECORD FULL) в вызывающую процедуру на языке ассемблера. Эти команды реализуют PL/M-конструкцию

```
RETURN  STATUS;
```

в процедуре ПРОВЕРКИ ЗАПОЛНЕНИЯ ВХОДНОЙ ЗАПИСИ (TINREC) и конструкцию

```
STATUS = TEST$IF$INPUT$RECORD$FULL;
```

в вызывающей процедуре, в которой STATUS является байтовой переменной.

Для передачи входных параметров используются и регистры, и стек. Если в процедуру передается один параметр, то независимо от того, является ли он байтовой или адресной переменной или константой, передача происходит через пару регистров BC. Если передаются два параметра, то первый передается через пару регистров BC, а второй — через пару регистров DE. Если необходимо передать более двух параметров в вызываемую процедуру, все они, кроме последних двух, заносятся в стек, а последние два передаются через пары регистров BC и DE, как было описано выше. На рис. 6.20 приведен пример передачи параметров для PL/M-конструкции

```
CALL ADD$TO$INPUT$RECORD (CHARACTER1, CHARACTER2, CHARACTER3);
```

который для наглядности модифицирован из соответствующей процедуры системы охранной сигнализации

В этом примере копия CHAR1 передается в регистр L по команде LHLD. Так как команда LHLD пересылает 16-битовый операнд, то при этом также используется регистр H, однако данные, помещаемые в него, излишни и поэтому игнорируются. После этого пара регистров HL помещается в стек. Сле-

```
        .
        .
        .
LHLD    CHAR1
PUSH    H
LHLD    CHAR2
MOV     C, L
LHLD    CHAR3
MOV     E, L
CALL    ADDREC
        .
        .
        .
```

Рис. 6.20а. Последовательность входных параметров для вызова процедуры ДОБАВЛЕНИЯ К ВХОДНОЙ ЗАПИСИ (ADDREC).

```
        .
        .
        .
LXI     H, CHAR3
MOV     M, E
LXI     H, CHAR2
MOV     M, C
POP     D
POP     B
LXI     H, CHAR1
MOV     M, C
PUSH    D
        .
        .
        .
```

Рис. 6.20б. Хранение входных параметров в процедуре ДОБАВЛЕНИЯ К ВХОДНОЙ ЗАПИСИ (ADDREC).

дующей в регистр L пересылается копия CHAR2 и затем помещается в регистр C командой MOV C, L. Таким же образом копия CHAR3 пересылается в регистр E. В конце вызывается процедура ДОБАВЛЕНИЯ К ВХОДНОЙ ЗАПИСИ (ADDREC — ADDCOINPUT$RECORD), при этом в стек помещается указатель и начинается выполнение вызванной процедуры.

Адрес ячейки памяти, используемой для хранения копии CHAR3 в процедуре ДОБАВЛЕНИЯ К ВХОДНОЙ ЗАПИСИ,

пересылается в пару регистров HL по команде LXI H, CHAR3. Затем копия CHAR3 из регистра E пересылается в ячейку памяти по команде MOV M, E. Подобным же образом копия CHAR2 из регистра C пересылается в ячейку памяти, предназначенную для ее хранения. Указатель адреса из вершины стека временно перемещается в пару регистров DE. Копия CHAR1 извлекается из стека с помощью команды POP B, которая пересылает ее в регистр C. Пересылаемый этой же командой байт в регистр B игнорируется. Байт из регистра C затем пересылается в ячейку памяти, предназначенную для хранения CHAR1. После этого указатель адреса, временно хранящийся в паре регистров DE, засылается обратно в стек и становится доступным для команды RET, завершающей выполнение процедуры.

Таким образом, стек обеспечивает простой механизм для передачи процедурам произвольного числа параметров. Если число входных параметров чрезмерно велико или если необходимо передать из вызываемой процедуры более одного параметра, часто используется способ передачи указателей, описанный в гл. 5. Команды ассемблера, которые используются для этой цели и которые, следовательно, эквивалентны PL/M-операции

```
RECORD$POINTER =. RECORD;
```

следующие:

```
LXI  H, RECORD
SHLD RECPTR
```

в результате чего RECPTR может передаваться от процедуры к процедуре, как было описано выше. Чтобы понять, как используется RECPTR, рассмотрим сначала, как осуществляется непосредственный доступ к структуре данных RECORD в МОДУЛЕ ОБРАБОТКИ ЗАПИСИ. В этом случае команды

```
LXI  H, RECORD
MOV  A, M
```

пересылают первый байт из RECORD в АККУМУЛЯТОР. Доступ к следующим байтам RECORD осуществляется путем добавления соответствующего *смещения* к значению пары регистров HL перед выполнением команды пересылки. В процедуре, которой передаются параметры, команды

```
LHLD RECPTR
MOV  A, M
```

также вызывают пересылку первого байта из RECORD в АККУМУЛЯТОР. Это возможно благодаря тому, что содержимое пары регистров HL то же самое, что и после выполнения команды LXI H, RECORD в процедуре МОДУЛЯ ОБРАБОТКИ

ЗАПИСИ или после выполнения команды LHLD RECPTR в процедуре, которой RECPTR передается в качестве параметра. То, что здесь описано, является средством языка ассемблера, эквивалентным способу для передачи структур или массивов между процедурами в языке PL/M с использованием базированных переменных.

6.8. Прерывания

Концепция прерываний в общих чертах обсуждалась в гл. 4. В этом разделе мы рассмотрим, как выполняются прерывания, как при этом используется стек, а также различные команды ассемблера, действие которых прямо связано с функцией прерываний. Начнем с описания того, как УСТРОЙСТВО УПРАВЛЕНИЯ реализует возможности прерываний.

Вспомним, что прежде, чем выполнить команду, УСТРОЙСТВО УПРАВЛЕНИЯ должно прочитать ее из ПАМЯТИ. Однако перед этим УСТРОЙСТВО УПРАВЛЕНИЯ должно проверить *сигнал запроса на прерывание*, выдаваемый модулем ПРЕРЫВАНИЙ, как показано на рис. 6.21. Если сигнала нет, УСТРОЙСТВО УПРАВЛЕНИЯ считывает из памяти следующую команду и выполняет все, что было описано ранее. Если же есть сигнал запроса на прерывание, то сначала УСТРОЙСТВО УПРАВЛЕНИЯ должно выполнить команду, по своим функциям похожую на команду CALL. Как мы далее увидим, эта команда засылается в УСТРОЙСТВО УПРАВЛЕНИЯ модулем ПРЕРЫВАНИЙ через ШИНУ ДАННЫХ тогда, когда получен сигнал ПОДТВЕРЖДЕНИЯ ЗАПРОСА НА ПРЕРЫВАНИЕ. *Вызываемая* таким способом процедура располагается в области памяти, начиная с ячейки, адрес которой также определяется модулем ПРЕРЫВАНИЙ. Процедура, управляющая выполнением процедур в зависимости от вида прерывания, называется процедурой ИСПОЛНЕНИЯ ПРЕРЫВАНИЙ. Так как команда модуля ПРЕРЫВАНИЙ, запускающая процедуру ИСПОЛНЕНИЯ ПРЕРЫВАНИЙ, по существу является командой ВЫЗОВА (CALL), адрес прерванной команды автоматически заносится в стек. Это позволяет продолжить выполнение прерванной команды после завершения выполнения процедуры ИСПОЛНЕНИЯ ПРЕРЫВАНИЙ.

Процедура ИСПОЛНЕНИЯ ПРЕРЫВАНИЙ системы охранной сигнализации воспроизведена на рис. 6.22. В гл. 4 мы не имели возможности объяснить, почему необходимы операции ЗАПОМНИТЬ СОСТОЯНИЕ СИСТЕМЫ и ВОССТАНОВИТЬ СОСТОЯНИЕ СИСТЕМЫ. Теперь, когда описана архитектура микрокомпьютера и известно, как она соотносится с прерываниями, мы можем обсудить, зачем нужны эти операции.

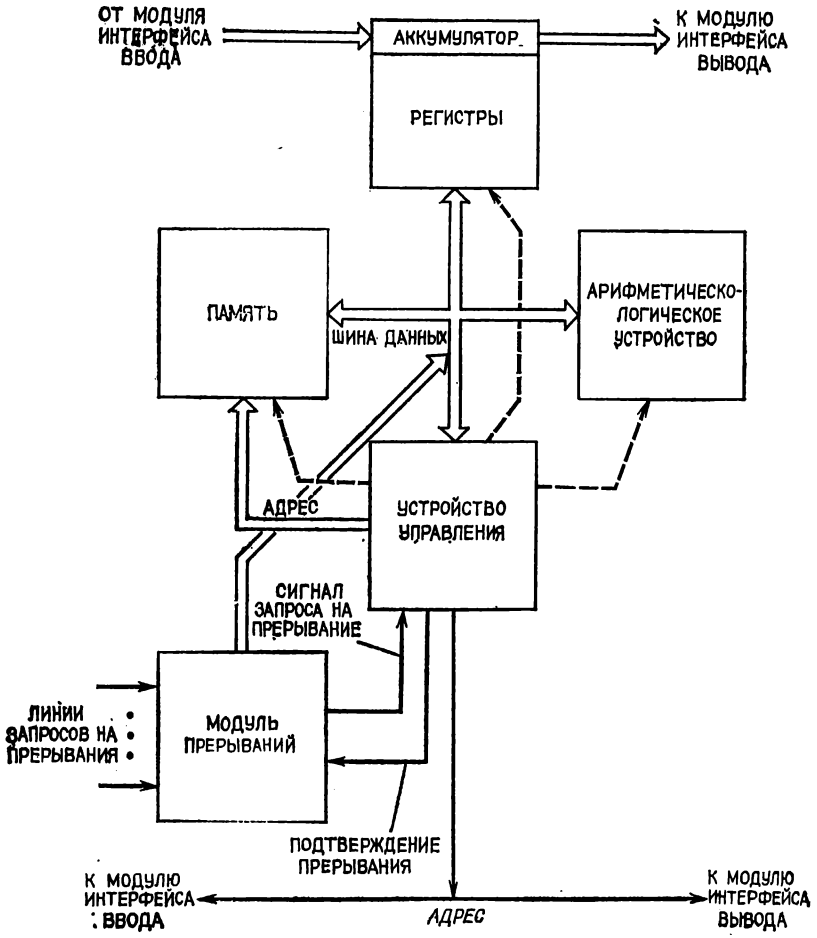


Рис. 6.21. Модуль ПРЕРЫВАНИЙ и его связь с модулем МИКРОКОМПЬЮТЕРА Intel 8085.

Запрос на прерывание может возникнуть в любой момент времени. Так, например, не существует способа, позволяющего гарантировать, что внутренние регистры или флажки микрокомпьютера не содержат испорченной информации, которая может привести к неправильному срабатыванию системы. Поэтому необходимо предусмотреть способы сохранения содержимого регистров и флажков сразу же после начала выполнения процедуры ИСПОЛНЕНИЯ ПРЕРЫВАНИЯ, а затем восстановления их содержимого до того, как будет продолжено выполнение прерванной процедуры. Стек (рис. 6.23) является

подходящим средством для этих целей. Отметим, что команды POP выполняются не в том же порядке, что команды PUSH. Обратный порядок необходим, чтобы обеспечить правильное

```
ПРОЦЕДУРА: ИСПОЛНЕНИЕ ПРЕРЫВАНИЙ (;)
НАЧАЛО ПРОЦЕДУРЫ
  ЗАПОМНИТЬ СОСТОЯНИЕ СИСТЕМЫ
  ВЫЗОВ: РАЗРЕШЕНИЕ ПРЕРЫВАНИЯ (;)
  ВЫЗОВ: ОБРАБОТКА ПРЕРЫВАНИЯ ТАЙМЕРА (;)
  ВОССТАНОВИТЬ СОСТОЯНИЕ СИСТЕМЫ
  ВОЗВРАТ
КОНЕЦ ПРОЦЕДУРЫ
```

Рис. 6.22. Процедура ИСПОЛНЕНИЯ ПРЕРЫВАНИЙ системы охранной сигнализации.

восстановление каждого сохраненного элемента. В этом примере символьное имя PSW обозначает СЛОВО СОСТОЯНИЯ ПРОГРАММ (PROGRAM STATUS WORD), которое состоит из АККУМУЛЯТОРА и флажков.

```
;                ПРОЦЕДУРА: ИСПОЛНЕНИЕ ПРЕРЫВАНИЙ (;)
;                НАЧАЛО ПРОЦЕДУРЫ
;                  ЗАПОМНИТЬ СОСТОЯНИЕ СИСТЕМЫ
  PUSH  H
  PUSH  D
  PUSH  B
  PUSH  PSW
;                  ВЫЗОВ: РАЗРЕШЕНИЕ ПРЕРЫВАНИЯ (;)
  CALL  ENABLE
;                  ВЫЗОВ: ОБРАБОТКА ПРЕРЫВАНИЯ ТАЙМЕРА (;)
  CALL  PROCES
;                  ВОССТАНОВИТЬ СОСТОЯНИЕ СИСТЕМЫ
  POP   PSW
  POP   B
  POP   D
  POP   H
;                  ВОЗВРАТ
  RET
;                КОНЕЦ ПРОЦЕДУРЫ
```

Рис. 6.23. Реализация процедуры ИСПОЛНЕНИЯ ПРЕРЫВАНИЙ на языке ассемблера.

В системе охранной сигнализации был предусмотрен модуль УПРАВЛЕНИЯ ПРЕРЫВАНИЯМИ, содержащий процедуры РАЗРЕШЕНИЯ ПРЕРЫВАНИЯ и ЗАПРЕЩЕНИЯ ПРЕРЫВАНИЯ. В нашем простом примере с использованием микрокомпьютера Intel 8085 процедура РАЗРЕШЕНИЯ ПРЕРЫВАНИЯ должна содержать только две команды: EI, или *Разрешить прерывание* (*Enable Interrupt*), и RET. Аналогично этому

процедура ЗАПРЕЩЕНИЯ ПРЕРЫВАНИЯ должна содержать команды DI, или *Запретить прерыванце* (*Disable Interrupt*), и RET. Как указывалось в гл. 4, во время выполнения прерывания все другие прерывания запрещаются. Поэтому необходимо восстановить механизм прерываний в одной из процедур прерывания, чтобы система могла откликаться на последующие запросы на прерывания. Как показано на рис. 6.23, система прерываний восстанавливается процедурой РАЗРЕШЕНИЯ ПРЕРЫВАНИЙ, которая вызывается процедурой ИСПОЛНЕНИЯ ПРЕРЫВАНИЙ.

Система охранной сигнализации содержит единственный источник прерываний, а именно сигнал от генератора тактовых импульсов. Однако многие другие системы, содержащие микрокомпьютер, откликаются на запросы от многих источников прерываний. Следующим будет рассмотрен вопрос, как осуществляется управление такими прерываниями в микрокомпьютере Intel 8085.

6.9. Многоуровневые прерывания

В системах, которые должны откликаться на запросы от многих источников прерываний, необходимо предусмотреть средство для распознавания запросов на прерывания. На рис. 6.24 показано, как это должно быть выполнено функционально с использованием языка проектирования. Наименования запросов на прерывания выбраны только для наглядности и не несут никакой смысловой нагрузки. Главное здесь то, что

```
ПРОЦЕДУРА: ИСПОЛНЕНИЕ ПРЕРЫВАНИЙ (;)
НАЧАЛО ПРОЦЕДУРЫ
  ЗАПОМНИТЬ СОСТОЯНИЕ СИСТЕМЫ
  ВЫЗОВ: РАЗРЕШЕНИЕ ПРЕРЫВАНИЯ (;)
  ВЫЗОВ: ПРОВЕРКА ЗАПРОСА НА ПРЕРЫВАНИЕ (· ЗАПРОС)
  ЕСЛИ ЗАПРОС ОТ ТАЙМЕРА
    ТО ВЫЗОВ: ОБРАБОТКА ПРЕРЫВАНИЙ ТАЙМЕРА (;)
  ЕСЛИ ЗАПРОС ОТ КОНТАКТНОГО ДЕТЕКТОРА
    ТО ВЫЗОВ: ОБРАБОТКА ПРЕРЫВАНИЙ ОТ КОНТАКТНОГО
    ДЕТЕКТОРА (;)
  ЕСЛИ ЗАПРОС ОТ АВАРИЙНОЙ КНОПКИ
    ТО ВЫЗОВ: ОБРАБОТКА ПРЕРЫВАНИЙ АВАРИЙНОЙ КНОПКИ (;)
  ВОССТАНОВИТЬ СОСТОЯНИЕ СИСТЕМЫ
  ВОЗВРАТ
КОНЕЦ ПРОЦЕДУРЫ
```

Рис. 6.24. Функциональная процедура ИСПОЛНЕНИЯ ПРЕРЫВАНИЙ для многоуровневой системы прерываний

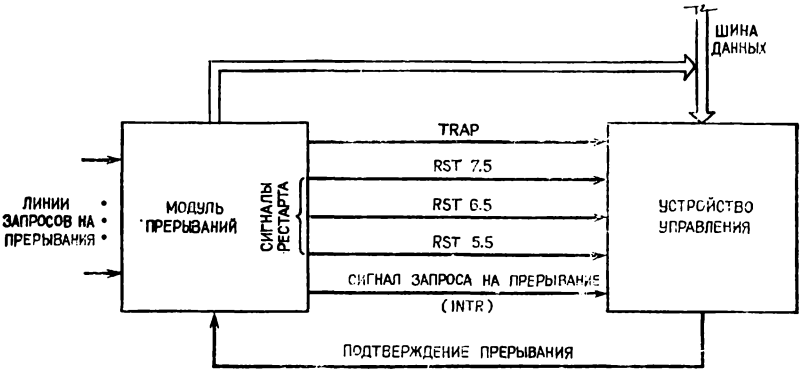


Рис. 6.25. Модуль ПРЕРЫВАНИЙ многоуровневой системы прерываний.

для определения вида запроса и выбора процедуры для обработки каждого прерывания необходимо выполнить серию проверочных операций. Эти операции проверки могут быть реализованы программным способом, путем дословного транслирования каждой конструкции языка проектирования; однако в микрокомпьютере Intel 8085 имеются аппаратные средства прерываний, обеспечивающие более удобное средство для этих целей.

Как показано на рис. 6.25, линии запроса прерываний связаны либо с сигналом запроса на прерывание INTR (Interrupt Request Signal), либо с одним из сигналов рестарта RST, либо с сигналом TRAP. Если какой-либо из этих сигналов включен линией запроса прерываний, УСТРОЙСТВО УПРАВЛЕНИЯ подтверждает запрос на прерывание и запускает систему прерываний. Способы запуска для каждого из этих сигналов отличаются один от другого.

Когда возбуждается линия запроса прерываний, связанная с сигналом запроса на прерывание (INTR), УСТРОЙСТВО УПРАВЛЕНИЯ, завершив выполнение предыдущей команды, не начинает выполнение следующей. Вместо этого оно посылает сигнал ПОДТВЕРЖДЕНИЯ ЗАПРОСА, который означает требование к модулю ПРЕРЫВАНИЙ на запуск команды CALL. Эта команда пересылается в УСТРОЙСТВО УПРАВЛЕНИЯ через ШИНУ ДАННЫХ. Выполнение команды CALL, пересланной в УСТРОЙСТВО УПРАВЛЕНИЯ, вызывает занесение адреса прерванной команды в стек и выполнение команды, находящейся в ячейке памяти, адрес которой определен в команде CALL. В общем случае в ячейках памяти, определяемых каждой предусмотренной модулем ПРЕРЫВАНИЙ командой CALL, должен быть размещен *вектор прерываний*,

реализованный с помощью команды безусловного *перехода*. Каждый такой вектор прерываний вызывает выполнение соответствующей процедуры прерываний. Следует отметить, что показанные на рис. 6.24 операции ЗАПОМНИТЬ СОСТОЯНИЕ СИСТЕМЫ, ВЫЗОВ: РАЗРЕШЕНИЕ ПРЕРЫВАНИЯ (;), ВОССТАНОВИТЬ СОСТОЯНИЕ СИСТЕМЫ и ВОЗВРАТ должны повторяться в каждой процедуре ОБРАБОТКИ. Это необходимо потому, что операции проверки и вызова являются теперь *аппаратными* операциями и могут выполняться в различной последовательности, как показано на примере модифицированной *процедуры* ИСПОЛНЕНИЯ ПРЕРЫВАНИЙ на рис. 6.26. На рис. 6.27 показана процедура ОБРАБОТКИ ПРЕРЫВАНИЙ ТАЙМЕРА, иллюстрирующая формат такой процедуры. Этот рисунок следует сравнить с рис. 4.30. На рис. 6.28 показаны команды ассемблера, необходимые для реализации части этой процедуры вместе с соответствующими командами *перехода* в ячейках памяти 0—63 для некоторых векторов прерываний. Техника прерываний, автоматически вызывающая выполнение процедур прерываний с использованием заранее определенных адресов памяти и векторов прерываний, известна под названием *векторных прерываний*.

На рис. 6.25 показаны также четыре других сигнала запроса на прерывание: TRAP, RST 7.5, RST 6.5 и RST 5.5. Выбор необычных наименований последних трех сигналов связан с историческими причинами, которые не имеют отношения к делу. Эти сигналы прерываний не требуют, чтобы в модуле ПРЕРЫВАНИЙ была предусмотрена команда CALL. УСТРОЙСТВО УПРАВЛЕНИЯ автоматически выполняет команду РЕСТАРТ для *вызова* команды из ячейки памяти 36, если *включен* сигнал TRAP, или из ячеек 60, 52 или 44, если *включены* сигналы RST 7.5, RST 6.5 или RST 5.5 соответственно. Таким образом, эти четыре сигнала обеспечивают дополнительную возможность векторных прерываний. Так как операции типа РЕСТАРТ выполняются под управлением только УСТРОЙСТВА УПРАВЛЕНИЯ, эти прерывания часто называют *внутренними* прерываниями в противоположность другим векторным прерываниям, называемым *внешними прерываниями*. Следует отметить, что все эти виды прерываний обеспечивают возможность отклика на любой из серии многоуровневых запросов на прерывание без обращений к программам проверки, необходимым для определения вида запроса на прерывание.

В случае одновременного возникновения двух запросов на прерывание система должна решить, на какой из них она должна откликаться. С этой целью каждому сигналу запроса на прерывание в соответствии со стандартами фирмы Intel назначен *приоритет*. Сигнал с высшим приоритетом признается пер-

вым в случае одновременного появления нескольких сигналов.

```
ПРОЦЕДУРА: ИСПОЛНЕНИЕ ПРЕРЫВАНИЙ (;)
**************************************************************************
  ДАННАЯ ПРОЦЕДУРА РЕАЛИЗОВАНА АППАРАТНЫМ СПОСОБОМ
**************************************************************************
НАЧАЛО ПРОЦЕДУРЫ
  ВЫЗОВ: ПРОВЕРКА ЗАПРОСА НА ПРЕРЫВАНИЕ (; ЗАПРОС)
  ЕСЛИ ЗАПРОС ОТ ТАЙМЕРА
    ТО ВЫЗОВ: ОБРАБОТКА ПРЕРЫВАНИЙ ТАЙМЕРА (;)
  ЕСЛИ ЗАПРОС ОТ КОНТАКТНОГО ДЕТЕКТОРА
    ТО ВЫЗОВ: ОБРАБОТКА ПРЕРЫВАНИЙ КОНТАКТНОГО
    ДЕТЕКТОРА (;)
  ЕСЛИ ЗАПРОС ОТ АВАРИЙНОЙ КНОПКИ
    ТО ВЫЗОВ: ОБРАБОТКА ПРЕРЫВАНИЙ АВАРИЙНОЙ КНОПКИ (;)
КОНЕЦ ПРОЦЕДУРЫ
```

Рис 6.26. *Процедура* ИСПОЛНЕНИЯ ПРЕРЫВАНИЙ для аппаратно-реализованной системы прерываний.

Относительные приоритеты для сигналов микропроцессора Intel 8085 показаны в таблице:

TRAP	наивысший
RST 7.5	•
RST 6.5	•
RST 5.5	•
INTR	низший

Таким образом, сигнал прерывания TRAP должен использоваться для распознавания внешних событий, которые являются

```
ПРОЦЕДУРА: ОБРАБОТКА ПРЕРЫВАНИЙ ТАЙМЕРА (;)
        ******************************************************
        НАЧАЛО ПРОЦЕДУРЫ
          ЗАПОМНИТЬ СОСТОЯНИЕ СИСТЕМЫ
          ВЫЗОВ: РАЗРЕШЕНИЕ ПРЕРЫВАНИЯ (;)
          УМЕНЬШИТЬ ЗНАЧЕНИЕ ТАЙМЕРА НА ЕДИНИЦУ
          ЕСЛИ ЗНАЧЕНИЕ ТАЙМЕРА РАВНО НУЛЮ
            ТО ВЫПОЛНИТЬ
                ВЫЗОВ: ОСТАНОВКА ТАЙМЕРА (;)
                УСТАНОВИТЬ ТАЙМЕР В СОСТОЯНИЕ ПОКОЯ
              КОНЕЦ
          ВОССТАНОВИТЬ СОСТОЯНИЕ СИСТЕМЫ
          ВОЗВРАТ
        КОНЕЦ ПРОЦЕДУРЫ
```

Рис. 6.27. Процедура ОБРАБОТКИ ПРЕРЫВАНИЙ ТАЙМЕРА.

```
;                       ВЕКТОР ПРЕРЫВАНИЙ ДЛЯ ВЫЗОВА
;                       ПО АДРЕСУ 0
   0    JMP    TIMER
;                       ВЕКТОР ПРЕРЫВАНИЙ ДЛЯ ВЫЗОВА
;                       ПО АДРЕСУ 8
   8    JMP    SWITCH
;                       ВЕКТОР ПРЕРЫВАНИЙ ДЛЯ ВЫЗОВА
;                       ПО АДРЕСУ 16
  16    JMP    PANIC
          .                 .
          .                 .
          .                 .
;                       ВЕКТОР ПРЕРЫВАНИЙ ДЛЯ ВЫЗОВА
;                       ПО АДРЕСУ 56
  56    JMP    . . .        .
                            .
                            .
-----------------------------------------------------
                            .
                            .
                            .
;                       ПРОЦЕДУРА: ОБРАБОТКА ПРЕРЫВАНИЙ
;                       ТАЙМЕРА (;)
;                       НАЧАЛО ПРОЦЕДУРЫ
;                         ЗАПОМНИТЬ СОСТОЯНИЕ СИСТЕМЫ
TIMER: PUSH   H
       PUSH   D
       PUSH   B
       PUSH   PSW
;                         ВЫЗОВ: РАЗРЕШЕНИЕ ПРЕРЫВАНИЯ (;)
       CALL   ENABLE
          .                 .
          .                 .
          .                 .
;                         ВОССТАНОВИТЬ СОСТОЯНИЕ СИСТЕМЫ
       POP    PSW
       POP    B
       POP    D
       POP    H
;                         ВОЗВРАТ
       RET
;                       КОНЕЦ ПРОЦЕДУРЫ
```

Рис. 6.28. Реализация векторных прерываний на языке ассемблера.

достаточно срочными, чтобы требовать очень быстрого отклика при их возникновении.

Примером события с высоким приоритетом является предупреждение о падении напряжения в сети электропитания, которое может привести к полному выключению всей системы.

В промежуток времени между моментами, когда обнаружится начало падения напряжения и когда напряжение упадет до уровня, при котором дальнейшее выполнение операций станет невозможным, микропроцессор может выполнить много команд. Поэтому система за этот промежуток времени может сама автоматически прекратить работу. Если напряжение восстановится, то прерывание может быть использовано для возобновления выполнения прерванной операции.

Низший приоритет прерываний назначен сигналу INTR. Однако этот сигнал может быть связан с несколькими линиями запроса на прерывание. Для этих линий приоритет устанавливается модулем ПРЕРЫВАНИЙ, который определяет, какая из различных команд CALL должна быть послана в УСТРОЙСТВО УПРАВЛЕНИЯ в случае одновременного возбуждения двух линий. Приоритеты некоторых специфичных прерываний могут быть определены системным проектировщиком и даже — в зависимости от конкретной аппаратной реализации модуля ПРЕРЫВАНИЙ — могут изменяться во время работы системы.

Описанные способы приоритетных прерываний определяют, какой из нескольких одновременно полученных системой запросов на прерывание должен выполняться первым. Однако эти способы не препятствуют тому, чтобы выполняемая процедура прерываний была прервана другой процедурой. Как указывалось ранее, во время выполнения прерывания механизм прерывания автоматически отключается. Из этого следует, что если механизм прерываний не восстанавливается до конца выполнения процедуры прерывания, то эта процедура не может быть прервана. Нет необходимости и даже нежелательно проектировать систему с такими свойствами, так как задержка в выполнении прерывания может привести к неэффективной работе системы. Следующим мы рассмотрим вопрос, касающийся *совмещенных прерываний.*

6.10. Совмещенные прерывания

Если обработка запроса на прерывание может быть прервана другим запросом на прерывание, то может оказаться, что в некоторый момент времени будет выполняться одновременно несколько процедур обработки прерываний. Поскольку обработка этих запросов на прерывание совмещена во времени, они называются *совмещенными прерываниями.*

Использование стека для управления процедурами обработки прерываний позволяет управлять совмещенными прерываниями без дополнительных усилий. Однако в каждой системе управляемых прерываний некоторые запросы на прерывания

могут иметь бо́льшую срочность (или важность), чем другие. Чтобы предотвратить прерывание процесса обработки запроса на прерывание другим запросом меньшей важности, необходимо установить систему приоритетов. В микрокомпьютере Intel 8085, как указывалось в предыдущей главе, определение порядка обработки нескольких одновременных запросов на прерывание является аппаратной функцией. Определение возможности прерывания процесса обработки запроса на прерыва-


```
ЕСЛИ РАЗРЯД РАЗРЕШЕНИЯ МАСКИРОВАНИЯ ЕСТЬ "ЕДИНИЦА"
   ТО ВЫПОЛНИТЬ ДЛЯ КАЖДОГО РАЗРЯДА МАСКИРОВАНИЯ
         УСТАНОВИТЬ РАЗРЯД МАСКИРОВАНИЯ В УСТРОЙСТВЕ УПРАВЛЕНИЯ РАВНЫМ ЗНАЧЕНИЮ
         РАЗРЯДА МАСКИРОВАНИЯ В АККУМУЛЯТОРЕ
   КОНЕЦ
```

Рис. 6.29. Интерпретация байта в АККУМУЛЯТОРЕ при выполнении команды SIM.

ние другим запросом реализуется способом комбинирования аппаратных и программных средств.

Аппаратные средства реализуют следующие виды приоритетов при обработке прерываний:

• Запрос на прерывание типа TRAP всегда действует и не может быть запрещен. Поэтому он имеет наивысший приоритет как по отношению к обработке прерываний, так и по отношению к отклику на запрос.

• Все остальные запросы на прерывание могут быть запрещены либо автоматически, вследствие выполнения предыдущего запроса, либо вследствие выполнения команды DI — *Запретить прерывание.*

Из последней группы прерываний некоторые запросы, а именно RST 5.5, RST 6.5 и RST 7.5, могут быть с помощью программных средств отключены выборочно, или *маскированы.* Из этого следует, что запрос на прерывание типа INTR имеет более высший приоритет обработки, чем запросы типа РЕСТАРТ, поскольку он не может быть маскирован. Для маскировки запросов на прерывание типа РЕСТАРТ используются команда *Установить маску прерываний* или SIM (*Set interrupt mask*) и соответствующая маска в АККУМУЛЯТОРЕ. В результате выполнения команды SIM любой из трех запросов на прерывание типа РЕСТАРТ может быть либо маскирован (если разряд маскирования равен ЕДИНИЦЕ), либо размаскирован (если разряд маскирования равен НУЛЮ). Маска длиной в один байт, которая должна быть помещена в АККУМУЛЯТОР до начала выполнения команды SIM, определяет

действия, выполняемые в соответствии со схемой на рис. 6.29. Как показано на рисунке, если разряд *разрешения маскирования* равен НУЛЮ, остальные разряды маски игнорируются. Как далее будет видно, это позволяет использовать команду SIM для других целей, определяемых остальными разрядами. Если значение разряда *разрешения маскирования* равно ЕДИНИЦЕ, разряды маски в УСТРОЙСТВЕ УПРАВЛЕНИЯ устанавливаются или сбрасываются в соответствии со значениями

	ФЛАЖОК ОЖИДАНИЯ RST 7.5	ФЛАЖОК ОЖИДАНИЯ RST 6.5	ФЛАЖОК ОЖИДАНИЯ RST 5.5	ФЛАЖОК РАЗРЕШЕНИЯ	РАЗРЯД МАСКИРОВАНИЯ RST 7.5	РАЗРЯД МАСКИРОВАНИЯ RST 6.5	РАЗРЯД МАСКИРОВАНИЯ RST 5.5

Рис. 6.30. Интерпретация байта состояния прерываний в АККУМУЛЯТОРЕ при выполнении команды RIM.

разрядов маски АККУМУЛЯТОРА. Например, последовательность команд

```
MVI A, 13
SIM
```

устанавливает разряды маски RST 7.5 и RST 5.5, блокируя тем самым указанные линии запросов на прерывания, и сбрасывает разряд маски RST 6 5, разрешая тем самым прерывание по линии запроса на прерывание RST 6.5, поскольку значение числа 13 равно 0000 1101 в двоичном представлении. Таким образом, приоритеты обработки прерываний для запросов на прерывания типа РЕСТАРТ могут быть установлены во время разработки программного обеспечения.

Иногда в программах бывает необходимо проверить состояние разрядов маски прерываний. В микрокомпьютере Intel 8085 эта возможность обеспечивается с помощью команды *Читать маску прерываний* или RIM (*Read interrupt mask*). Во время выполнения команды RIM АККУМУЛЯТОР загружается байтом информации, значение которого интерпретируется, как показано на рис. 6.30. Три бита маски размещаются в трех младших разрядах АККУМУЛЯТОРА. Если значение разряда *флажка разрешения* равно ЕДИНИЦЕ, это означает, что прерывания разрешены. Если значение разряда *флажка разрешения* равно НУЛЮ, это означает, что прерывания заблокированы либо автоматически, либо командой DI или *Запретить прерывания*. Каждый *флажок ожидания* указывает, что получен соответствующий запрос на прерывание типа РЕСТАРТ, но прерывание по этому запросу еще не выполнено. Это может произойти либо тогда, когда прерывания заблокированы, либо когда установлен в 1 соответствующий биг маски. Необходимо

отметить, что отложенный запрос на прерывание не игнорируется, выполнение прерывания по этому запросу только отложено до завершения обработки запросов на прерывания с более высоким приоритетом.

Этим завершается общее описание архитектуры микрокомпьютеров. До сих пор описание архитектуры микрокомпьютера Intel 8085 рассматривалось с точки зрения языка ассемблера. Рассмотрим далее некоторые более сложные вопросы архитектуры.

ЧАСТЬ II. ИЗБРАННЫЕ ВОПРОСЫ

В ч. I данной главы была описана базовая структура микрокомпьютера Intel 8085. В этой части будут рассмотрены некоторые относящиеся к архитектуре вопросы, например такие, как повышение эффективности ввода-вывода с помощью прерываний. Мы также завершим обсуждение языка PL/M примерами программирования ввода-вывода и прерываний с помощью PL/M.

6.11. Реентерабельность

В гл. 4 обсуждался случай, когда во время обработки прерывания вызывалась и начинала выполняться та же самая процедура, выполнение которой было прервано. Для того чтобы функционирование системы в таких случаях не нарушалось, процедура должна удовлетворять следующим условиям:

- Во время прерывания процедуры для хранения всех указателей, регистров и флажков должен быть использован стек [1].
- Стек должен быть также использован для хранения всех переменных, находящихся в обработке в момент прерывания процедуры.

Процедура, удовлетворяющая этим условиям, называется *реентерабельной* процедурой. Заметим, что все процедуры, вызываемые реентерабельной процедурой, должны быть также реентерабельными. Это необходимо для того, чтобы любая из них могла быть вызвана в тот момент, когда ее выполнение прервано. Однако нет необходимости в том, чтобы все процедуры модуля были реентерабельными. Таким свойством должны обладать только те процедуры, которые появляются как в де-

[1]) Процесс сохранения значений указателей, регистров и флажков после возникновения прерывания и их последующего восстановления называется *контекстным переключением. — Прим. перев.*

реве вызова, в вершине которого находится процедура ИСПОЛНЕНИЯ, так и в дереве вызова, в вершине которого находится процедура ИСПОЛНЕНИЯ ПРЕРЫВАНИЯ.

6.12. Ввод-вывод и прерывания

Операции, выполняемые введенными ранее командами ввода и вывода, обычно называются *программируемым вводом-выводом.* В этом разделе будет рассмотрен способ повышения эффективности ввода-вывода с помощью прерываний.

В примере гл. 1 ввод и вывод выполняются асинхронно. Это означает, что как для ввода, так и для вывода ожидание отсутствует. Значения разрядов порта ввода используются в момент выполнения команды ВВОДА. Аналогичным образом выход модифицируется во время выполнения команды ВЫВОДА. Однако во многих системах приходится иметь дело с устройствами синхронного ввода или вывода или с устройствами, в которых между чтением двух следующих одна за другой порциями входной информации или записью двух порций выходной существует фиксированный промежуток времени ожидания. В обоих случаях необходимо быть уверенным, что последовательное выполнение команд ВВОДА и ВЫВОДА разделено во времени и гарантирует правильное функционирование системы. Поясним это на примере.

Представим, что в микрокомпьютер поступает сообщение для печати на пишущей машинке, состоящее из строки символьных кодов. Допустим, что пишущая машинка может печатать со скоростью 30 символ/с, что является почти предельным для электромеханических устройств печати. Таким образом, если передача каждого символа на пишущую машинку с помощью микрокомпьютера производится чаще, чем через 33 мс, существует большая вероятность того, что некоторые из символов, посланные на пишущую машинку, не будут напечатаны. Наиболее простой способ, гарантирующий, что каждый символ будет напечатан, заключается в том, что прежде, чем посылать очередной символ на пишущую машинку, необходимо подождать, пока не будет напечатан предыдущий. Это означает, что микрокомпьютер должен непрерывно проверять сигнал ЗАНЯТО от пишущей машинки до тех пор, пока этот сигнал не изменится, указывая на то, что пишущая машинка больше не занята и что следующий символ может быть послан. Этот процесс проверки и ожидания может быть повторен, пока не будет напечатано все сообщение. Описание процедур на языке проектирования, показанное на рис. 6.31 и 6.32, иллюстрирует эти концепции.

```
ПРОЦЕДУРА: ВЫВОД СООБЩЕНИЯ НА ПИШУЩУЮ МАШИНКУ
(СООБЩЕНИЕ;)
НАЧАЛО ПРОЦЕДУРЫ
  ВЫПОЛНИТЬ ДЛЯ КАЖДОГО СИМВОЛА В СООБЩЕНИИ
    ВЫЗОВ: ОЖИДАНИЕ ОСВОБОЖДЕНИЯ ПИШУЩЕЙ МАШИНКИ (;)
    ВЫЗОВ: ВЫВОД НА ПИШУЩУЮ МАШИНКУ (СИМВОЛ;)
  КОНЕЦ
  ВОЗВРАТ
КОНЕЦ ПРОЦЕДУРЫ
```

Рис. 6.31. Процедура ВЫВОДА СООБЩЕНИЯ НА ПИШУЩУЮ МАШИНКУ.

```
ПРОЦЕДУРА: ОЖИДАНИЕ ОСВОБОЖДЕНИЯ ПИШУЩЕЙ МАШИНКИ (;)
НАЧАЛО ПРОЦЕДУРЫ
  ВЫПОЛНЯТЬ НЕПРЕРЫВНО
    ВЫЗОВ: ЧТЕНИЕ СИГНАЛА "ЗАНЯТО" (;ЗАНЯТО)
    ЕСЛИ СИГНАЛ "ЗАНЯТО" НЕ УСТАНОВЛЕН
      ТО ВОЗВРАТ
  КОНЕЦ
КОНЕЦ ПРОЦЕДУРЫ
```

Рис. 6.32. Процедура ОЖИДАНИЯ ОСВОБОЖДЕНИЯ ПИШУЩЕЙ МАШИНКИ.

```
ПРОЦЕДУРА: ПОДГОТОВКА К ВЫВОДУ СООБЩЕНИЯ НА ПИШУЩУЮ
МАШИНКУ (СООБЩЕНИЕ;)
НАЧАЛО ПРОЦЕДУРЫ
  ВЫЗОВ: ПРОВЕРКА ОКОНЧАНИЯ ПЕЧАТИ ПРЕДЫДУЩЕГО
  СООБЩЕНИЯ (;СТАТУС)
  ЕСЛИ СТАТУС УКАЗЫВАЕТ ЧТО ПЕЧАТАЕТСЯ ПРЕДЫДУЩЕЕ
  СООБЩЕНИЕ
    ТО    ВЫПОЛНИТЬ
            ВЫЗОВ: ПОМЕСТИТЬ СООБЩЕНИЕ В ОЧЕРЕДЬ
            (СООБЩЕНИЕ;)
            ВОЗВРАТ
          КОНЕЦ
    ИНАЧЕ ВЫПОЛНИТЬ
            ВЫЗОВ: УСТАНОВИТЬ ЧТО ПЕЧАТАЕТСЯ СООБЩЕНИЕ (;)
            ВЫЗОВ: ЗАПОМНИТЬ ТЕКУЩЕЕ СООБЩЕНИЕ
            (СООБЩЕНИЕ;)
            ВЫЗОВ: ВЫБРАТЬ СИМВОЛ ИЗ ТЕКУЩЕГО
            СООБЩЕНИЯ (; СИМВОЛ)
            ВЫЗОВ: ВЫВОД НА ПИШУЩУЮ МАШИНКУ (СИМВОЛ;)
            ВЫЗОВ: РАЗМАСКИРОВАТЬ И РАЗРЕШИТЬ
            ПРЕРЫВАНИЕ ПИШУЩЕЙ МАШИНКИ (;)
            ВОЗВРАТ
          КОНЕЦ
КОНЕЦ ПРОЦЕДУРЫ
```

Рис. 6.33. Процедура ПОДГОТОВКИ К ВЫВОДУ СООБЩЕНИЯ НА ПИШУЩУЮ МАШИНКУ.

```
ПРОЦЕДУРА: ОБРАБОТКА ПРЕРЫВАНИЙ ПИШУЩЕЙ МАШИНКИ (;)
НАЧАЛО ПРОЦЕДУРЫ
  ЗАПОМНИТЬ СОСТОЯНИЕ СИСТЕМЫ
  ВЫЗОВ: РАЗРЕШЕНИЕ ПРЕРЫВАНИЯ (;)
  ВЫЗОВ: ПРОВЕРКА ОКОНЧАНИЯ ТЕКУЩЕГО СООБЩЕНИЯ
  (; СТАТУС)
  ЕСЛИ СТАТУС УКАЗЫВАЕТ ЧТО ТЕКУЩЕЕ СООБЩЕНИЕ
  НЕ ОКОНЧЕНО
    ТО ВЫПОЛНИТЬ
       ВЫЗОВ: ВЫБРАТЬ СИМВОЛ ИЗ ТЕКУЩЕГО СООБЩЕНИЯ
       (; СИМВОЛ)
       ВЫЗОВ: ВЫВОД НА ПИШУЩУЮ МАШИНКУ (СИМВОЛ;)
       ВОССТАНОВИТЬ СОСТОЯНИЕ СИСТЕМЫ
       ВОЗВРАТ
     КОНЕЦ
    ИНАЧЕ ВЫПОЛНИТЬ
       ВЫЗОВ: ПРОВЕРИТЬ НАЛИЧИЕ СООЩЕНИЯ В ОЧЕРЕДИ
       (; СТАТУС)
       ЕСЛИ СТАТУС УКАЗЫВАЕТ НАЛИЧИЕ СООБЩЕНИЯ
       В ОЧЕРЕДИ
         ТО ВЫПОЛНИТЬ
            ВЫЗОВ: ВЫБРАТЬ СООБЩЕНИЕ ИЗ ОЧЕРЕДИ
            (; СООБЩЕНИЕ)
            ВЫЗОВ: ЗАПОМНИТЬ ТЕКУЩЕЕ СООБЩЕНИЕ
            (СООБЩЕНИЕ;)
            ВЫЗОВ: ВЫБРАТЬ СИМВОЛ ИЗ ТЕКУЩЕГО СООБЩЕНИЯ
            (; СИМВОЛ)
            ВЫЗОВ: ВЫВОД НА ПИШУЩУЮ МАШИНКУ (СИМВОЛ;)
            ВОССТАНОВИТЬ СОСТОЯНИЕ СИСТЕМЫ
            ВОЗВРАТ
          КОНЕЦ
         ИНАЧЕ ВЫПОЛНИТЬ
            ВЫЗОВ: МАСКИРОВАТЬ ПРЕРЫВАНИЕ ПИШУЩЕЙ
            МАШИНКИ (;)
            ВЫЗОВ: УСТАНОВИТЬ ОКОНЧАНИЕ ПЕЧАТИ
            СООБЩЕНИЯ (;)
            ВОССТАНОВИТЬ СОСТОЯНИЕ СИСТЕМЫ
            ВОЗВРАТ
          КОНЕЦ
     КОНЕЦ
КОНЕЦ ПРОЦЕДУРЫ
```

Рис. 6.34. Процедура ОБРАБОТКИ ПРЕРЫВАНИЙ ПИШУЩЕЙ МАШИНКИ.

В этом примере микрокомпьютер затрачивает значительную часть времени на непроизводительную проверку сигнала ЗАНЯТО от пишущей машинки. Как указывалось ранее, непроизводительные затраты определяют разницу между тем, что система может и что она выполняет в действительности. Если использовать сигнал ЗАНЯТО для генерации запроса на прерывание каждый раз, когда состояние «занято» меняется на состояние «не занято», то процедура прерывания может служить для вывода символов на пишущую машинку. Микрокомпьютер ничем больше не связан непосредственно с пишущей машинкой, кроме отклика на прерывание. Таким образом, повышается не только эффективность использования микрокомпьютера, но также появляется возможность разделить операции ввода и вывода между микрокомпьютером и несколькими медленными устройствами ввода и вывода. На рис. 6.33 показана процедура на языке проектирования, которая подготавливает систему к выводу сообщения на пишущую машинку через прерывание. Следует отметить, что если во время выполнения процедуры в обработке находится другое сообщение, то подготавливаемое сообщение ставится в очередь. Если же нет, то сообщение запоминается, первый символ сообщения выводится на печать, а система прерываний размаскируется и разрешается прерывание. На рис. 6.34 показана процедура ОБРАБОТКИ ПРЕРЫВАНИЯ ОТ ПИШУЩЕЙ МАШИНКИ, которая выполняется по запросу на прерывание от пишущей машинки. Эта процедура выполняет следующие действия: если в текущем сообщении остается хотя бы один символ, то он выводится на пишущую машинку. Если печать текущего сообщения закончена и если в очереди находится следующее сообщение, процедура начинает печатать следующее сообщение. В остальных случаях прерывание от пишущей машинки маскируется. Аналогичные процедуры прерывания могут быть использованы для других устройств ввода и вывода.

Для обеспечения быстрого ввода и вывода данных используется *канал ПДП*, который является более эффективным, чем программируемый ввод-вывод или ввод-вывод, управляемый с помощью прерываний.

6.13. Канал прямого доступа к памяти (ПДП)

Как было показано выше, программируемый ввод-вывод более эффективен в случае, если он управляется прерываниями. Однако при этом необходимо выполнять процедуры прерываний для каждого байта, передаваемого от микрокомпьютера к каждому устройству ввода-вывода и наоборот. На выполнение шагов прерывания процессора, сохранение состояния системы,

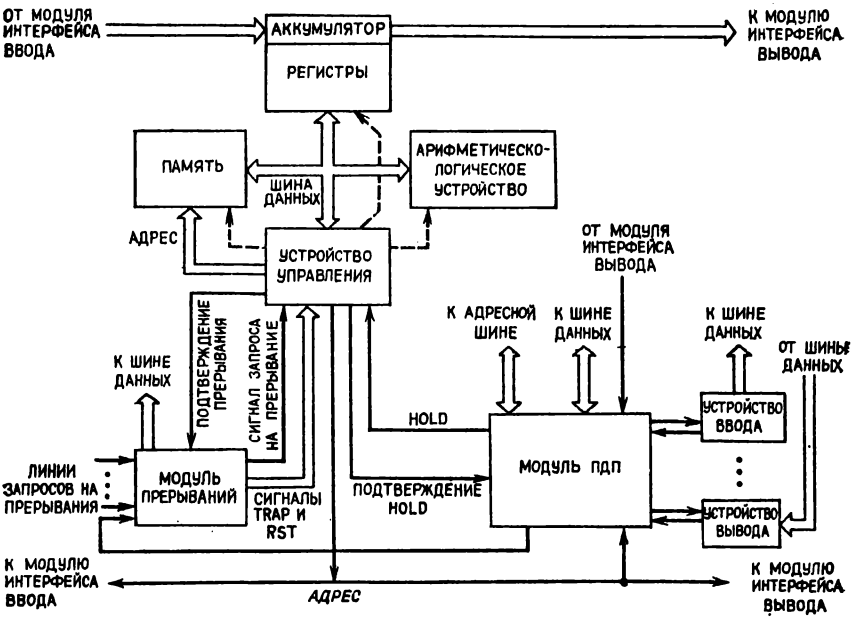


Рис. 6.35. Модуль ПДП и его связь с модулем МИКРОКОМПЬЮТЕРА Intel 8085.

получение следующего байта, выполнение команды ВЫВОД (или выполнение команды ВВОД и хранение прочитанного байта) и восстановление системы требуется значительное время. Таким образом, программируемый ввод-вывод, даже если он использует прерывания, является подходящим для сравнительно медленных устройств ввода-вывода. *Канал прямого доступа к памяти*, или ПДП, не требует выполнения команд ввода-вывода или процедуры прерывания для каждого байта. Поэтому он используется для быстрых устройств ввода-вывода.

Канал ПДП реализован аппаратными средствами и требует, чтобы команды микропроцессора выполнялись только в начале и конце передачи целого блока данных. Аппаратные средства, включающие модуль ПДП, показаны на рис. 6.35. Модуль ПДП хранит в ПАМЯТИ информацию о том, откуда или куда передаются данные, какой канал ПДП (входной или выходной) задействован, какое количество байт осталось передать или получить. Чтобы подготовить к работе канал ПДП, выполняются команды программируемого ввода-вывода, в результате которых адрес первой ячейки блока памяти и длина блока засылаются в модуль ПДП. В микрокомпьютере Intel 8085 указывается *длина блока, уменьшенная на единицу*. Команда

запуска канала ПДП, посылаемая в модуль ПДП, завершает операции, необходимые для запуска канала. После этого микропроцессор может выполнять другие задачи, в то время как завершение передачи данных возлагается на модуль ПДП.

Ниже рассматривается последовательность шагов, которые необходимо выполнить модулю ПДП для завершения вывода данных.

1. Сигнал ЗАНЯТО от устройства вывода контролируется модулем ПДП до тех пор, пока он не изменится, указывая тем самым, что устройство не занято.

2. После этого модуль ПДП *включает* сигнал HOLD, запрашивая у УСТРОЙСТВА УПРАВЛЕНИЯ разрешение на использование ШИНЫ ДАННЫХ.

3. Как только УСТРОЙСТВО УПРАВЛЕНИЯ завершит выполнение операций с ШИНОЙ ДАННЫХ и сможет передать ее, оно *включает* сигнал ПОДТВЕРЖДЕНИЯ HOLD. Этим самым дается разрешение модулю ПДП на использование ШИНЫ ДАННЫХ.

4. После того как модуль ПДП получит это разрешение, он посылает адрес следующей ячейки блока в ПАМЯТЬ через АДРЕСНУЮ ШИНУ. Из соответствующей ячейки памяти считывается байт и посылается в устройство вывода по ШИНЕ ДАННЫХ. Одновременно модуль ПДП командует устройству вывода принять байт данных с шины.

5. Проверяется значение длины блока, хранящееся в модуле ПДП.

6. Если это значение не равно нулю, то оно уменьшается на единицу, а значение адреса в модуле ПДП увеличивается на единицу, указывая на следующую ячейку памяти. Затем операция продолжается, начиная с шага 1.

7. Если значение длины блока равно нулю, передача блока завершена и операция ПДП приостанавливается до следующего запуска.

Необходимо отметить, что передача данных по каналу ПДП лишь незначительно влияет на работу микрокомпьютера. Микрокомпьютер функционирует с максимальным быстродействием, за исключением того момента, когда на шаге 4 считывается ПАМЯТЬ. В этом отличие от программируемого ввода-вывода, который требует выполнения процедуры прерывания для каждого байта, передаваемого в устройство вывода.

Подобным же образом модуль ПДП работает при вводе данных. Как только сигнал ЗАНЯТО указывает, что следующий байт готов к передаче от устройства ввода в ПАМЯТЬ, выполняется последовательность операций, подобная описанным выше. Основное отличие заключается в том, что байт в устройстве ввода должен быть доступен до того, как УСТРОЙ-

СТВУ УПРАВЛЕНИЯ будет послан запрос на использование ШИНЫ ДАННЫХ. Кроме того, когда модуль ПДП получит разрешение на использование ШИНЫ ДАННЫХ, байт из устройства ввода посылается в ПАМЯТЬ и хранится в ячейке памяти, адрес которой определяется модулем ПДП. И в этом случае влияние передачи данных по каналу ПДП на работу микропроцессора минимальное. Прежде чем закончить обсуждение этого вопроса, рассмотрим, какую роль играют прерывания при обмене по каналу ПДП.

Представим следующую ситуацию: микрокомпьютер завершил подготовку порции информации для вывода, в то время как вывод предыдущей порции по каналу ПДП не закончился. Поскольку устройство вывода занято, вывод новой порции информации должен быть отложен до завершения передачи предыдущей порции. Микрокомпьютер помещает запрос на вывод блока в очередь и продолжает выполнение других задач, ожидая, пока завершится вывод предыдущего блока. С целью повышения эффективности желательно, чтобы вывод нового блока начинался сразу же после завершения вывода предыдущего. Таким образом, здесь также уместны соображения относительно использования прерываний. Микропроцессор должен прервать выполнение других операций, когда передача предыдущего блока завершена, и выполнить процедуру обработки прерывания, определяющую наличие блока для передачи в очереди. Если это так, то процедура обработки прерывания должна выполнить команды, необходимые для начала передачи блока, находящегося в очереди. Подобный механизм может быть использован для обеспечения эффективного запуска при вводе данных.

6.14. Машинный язык

Понятия архитектуры микропроцессора и языка ассемблера были введены для того, чтобы читатель смог разобраться в деталях прерываний и ввода-вывода. Для конвертирования описания программного обеспечения, составленного на языке проектирования, язык высокого уровня, например PL/M, предпочтительнее языка ассемблера. Использование языка ассемблера оправданно только в тех случаях, когда, например, система, включающая критические циклы синхронизации, не может быть надлежащим образом реализована с использованием PL/M. И даже в этом случае он должен быть использован только для реализации критического цикла. Так как машинный язык относительно труден для понимания и оттого, что между ним и языком ассемблера существует взаимооднозначное соответствие, машинный язык *никогда* не должен

использоваться при разработке программного обеспечения для микрокомпьютера.

Однако, как было показано на рис. 5.1, машинный язык является тем, чем мы в конце концов должны завершить разработку программы, поскольку микрокомпьютер выполняет команды только на машинном языке. К тому же, как мы далее увидим, умение распознавать и понимать команды машинного языка может оказаться полезным на этапе интеграции цикла проектирования системы. По этой причине мы включили в данный раздел краткое описание команд машинного языка.

Команды языка ассемблера пишутся с использованием алфавитных и цифровых символов, так что мы легко можем прочитать их. Однако было бы неразумным пытаться заставить микрокомпьютер *понимать* команды языка ассемблера непосредственно. Поэтому каждая команда языка ассемблера транслируется в код машинного языка, который может быть воспринят и выполнен микрокомпьютером.

В качестве примера рассмотрим команду языка ассемблера IN 1. Эта команда представляется на машинном языке последовательностью битов 1101 1011 0000 0001, где 1101 1011 представляет часть IN, а 0000 0001 — число 1. Поскольку работать с последовательностью битов довольно трудно, для представления команд машинного языка часто используется шестнадцатеричное обозначение (см. приложение А). Например, IN 1 эквивалентно DB 01 в шестнадцатеричном представлении. На рис. 6.36 перечислены команды машинного языка микрокомпьютера Intel 8085, соответствующие большинству команд языка ассемблера, рассмотренных в данной главе. Рисунок включает интерпретацию каждой команды, которая может быть использована для ссылок. Следует заметить, что этот список не претендует на полноту.

На рисунке используются следующие обозначения:

- Буква M в команде языка ассемблера указывает, что используемый адрес содержится в паре регистров HL (косвенная адресация).
- Символьное имя PROC используется для обозначения имени вызываемой процедуры. Предполагается, что первая команда процедуры PROC находится в ячейке памяти 04 05 (в шестнадцатеричном представлении).

ЗАМЕЧАНИЕ: В каждой команде микрокомпьютера Intel 8085, использующей прямую адресацию, байты адреса располагаются в обратном порядке.

- Символьное имя ELSE используется для обозначения прямой адресации к ячейке памяти, содержащей команду, которая должна выполняться следующей в случае выполнения условия,

Язык ассемблера		Машинный язык	Интерпретация команд
ADC	M	8E	Сложить 8-битовое число из ячейки памяти, адрес которой находится в паре регистров HL, с содержимым АККУМУЛЯТОРА и флажком переноса.
ADD	C	81	Сложить 8-битовое число, находящееся в регистре C, с содержимым АККУМУЛЯТОРА.
ADD	M	86	Сложить 8-битовое число из ячейки памяти, адрес которой находится в паре регистров HL, с содержимым АККУМУЛЯТОРА.
ADI	20	C6 14	Сложить число 20 с содержимым АККУМУЛЯТОРА.
ANI	15	E6 0F	Логическая операция И над двоичным представлением числа 15 и содержимым АККУМУЛЯТОРА.
CALL	PROC	CD 05 04	Вызвать процедуру, первая команда которой находится в ячейке PROC.
CC	PROC	DC 05 04	Вызвать процедуру, первая команда которой находится в ячейке памяти PROC, если флажок переноса установлен.
CMC		3F	Дополнить флажок (изменить состояние флажка) переноса
CMP	C	B9	Сравнить 8-битовое число в регистре C с числом в АККУМУЛЯТОРЕ и установить флажки.
CNC	PROC	D4 05 04	Вызвать процедуру, первая команда которой находится в ячейке памяти PROC, если флажок переноса не установлен.
CPI	4	FE 04	Сравнить число 4 с числом в АККУМУЛЯТОРЕ и установить флажки.
CZ	PROC	CC 05 04	Вызвать процедуру, первая команда которой находится в ячейке памяти PROC, если флажок нуля установлен.
DAD	D	19	Сложить 16-битовое число, находящееся в паре регистров DE, с числом в паре регистров HL.
DCR	C	0D	Уменьшить на 1 значение 8-битового числа в регистре C.
DCX	D	1B	Уменьшить на 1 значение 16-битового числа в паре регистров DE.
DI		F3	Запретить прерывание.
EI		FB	Разрешить прерывание.

Рис. 6.36. Подмножество команд машинного языка Intel 8085.

Язык ассемблера		Машинный язык	Интерпретация команд
IN	1	DB 01	Считать данные из порта ввода 1 и поместить их в АККУМУЛЯТОР.
INX	H	23	Увеличить на 1 значение 16-битового числа в паре регистров HL.
JC	ELSE	DA 13 27	Переход к команде в ячейке памяти ELSE, если флажок переноса установлен.
JMP	ELSE	C3 13 27	Переход к команде в ячейке памяти ELSE.
JNC	ELSE	D2 13 27	Переход к команде в ячейке памяти ELSE, если флажок переноса не установлен.
JNZ	ELSE	C2 13 27	Переход к команде в ячейке памяти ELSE, если флажок нуля не установлен.
JZ	ELSE	CA 13 27	Переход к команде в ячейке памяти ELSE, если флажок нуля установлен.
LDA	VALUE	3A 2E 28	Загрузить (переслать) в АККУМУЛЯТОР данные из ячейки памяти VALUE.
LDAX	B	0A	Загрузить (переслать) в АККУМУЛЯТОР данные из ячейки памяти, адрес которой находится в паре регистров BC.
LHLD	VALUE	2A 2E 28	Загрузить (переслать) в пару регистров HL данные из ячейки памяти VALUE.
LXI	H, VALUE	21 2E 28	Загрузить (переслать) в пару регистров HL адрес ячейки памяти VALUE.
MOV	A, H	7C	Переслать данные из регистра H в АККУМУЛЯТОР.
MOV	A, M	7E	Переслать данные из ячейки памяти, адрес которой находится в паре регистров HL, в АККУМУЛЯТОР.
MOV	M, A	77	Переслать данные из АККУМУЛЯТОРА в ячейку памяти, адрес которой находится в паре регистров HL.
MVI	A, 6	3E 06	Переслать число 6 в АККУМУЛЯТОР.
ORA	C	B1	Логическая операция ИЛИ над содержимым регистра C и АККУМУЛЯТОРА.
ORI	48	F6 30	Логическая операция ИЛИ над двоичным значением числа 48 и содержимым АККУМУЛЯТОРА.
OUT	1	D3 01	Переслать данные из АККУМУЛЯТОРА в порт вывода 1.
POP	PSW	F1	Извлечь из стека и хранить данные в Слове состояния программы (в АККУМУЛЯТОРЕ и флажках).

Рис. 6.36. (Продолжение).

Язык ассемблера		Машинный язык	Интерпретация команд
PUSH	H	E5	Занести в стек значение пары регистров HL.
RAL		17	Циклический сдвиг АККУМУЛЯТОРА влево с использованием флажка переноса.
RAR		1F	Циклический сдвиг АККУМУЛЯТОРА вправо с использованием флажка переноса.
RC		D8	Возврат, если флажок переноса установлен.
RET		C9	Возврат.
RIM		20	Читать маску прерываний и переслать данные в АККУМУЛЯТОР.
RLC		07	Циклический сдвиг АККУМУЛЯТОРА влево.
RRC		0F	Циклический сдвиг АККУМУЛЯТОРА вправо.
RZ		C8	Возврат, если флажок нуля установлен.
SBI	35	DE 23	Вычесть число 35 из содержимого АККУМУЛЯТОРА, а также вычесть флажок.
SHLD	VALUE	22 2E 28	Хранить в ячейке (переслать в ячейку) памяти VALUE данные из пары регистров HL.
SIM		30	Установить маску прерываний в соответствии с данными в АККУМУЛЯТОРЕ.
SPHL		F9	Загрузить (переслать) в регистр указателя стека адрес из пары регистров HL.
STA	VALUE	32 2E 28	Хранить в ячейке (переслать в ячейку) памяти VALUE данные из АККУМУЛЯТОРА.
STAX	D	12	Хранить в ячейке (переслать в ячейку) памяти, адрес которой находится в паре регистров DE, данные из АККУМУЛЯТОРА.
STC		37	Установить флажок переноса.
SUB	C	91	Вычесть 8-битовое число, находящееся в регистре C, из АККУМУЛЯТОРА.

Рис. 6.36. (Продолжение).

определенного командой условного *перехода*. В нашем примере предполагается, что этот адрес равен 27 13 в шестнадцатеричном представлении.

• Символьное имя VALUE используется для обозначения прямой адресации к ячейке памяти, содержащей операнд. В команде LXI адрес, представленный символьным именем VALUE, сам является операндом (непосредственная адресация). В нашем примере предполагается, что этот адрес равен 28 2E в шестнадцатеричном представлении. Отметим, что байты непосредственного адреса в команде LXI также хранятся в обратном порядке.

Этим завершается обсуждение архитектуры микропроцессора, но прежде, чем закончить главу, нам хотелось бы показать, как могут быть выполнены некоторые операции, рассмотренные в данной главе, на языке PL/M.

6.15. Завершение описания языка PL/M

В гл. 5 мы обещали завершить рассмотрение операций ввода-вывода, прерываний и некоторых других функций на языке PL/M в этой главе. Теперь эти операции могут быть сопоставлены с эквивалентными командами на языке ассемблера.

Операция на языке проектирования

ПРОВЕРИТЬ ВХОД 1 И ХРАНИТЬ ЕГО ЗНАЧЕНИЕ

может быть представлена командами языка ассемблера

```
IN    1
STA   VALUE1
```

Эквивалентной операцией на языке PL/M является

```
VALUE1=INPUT(1);
```

Параметр в скобках является номером порта ввода. После того как данное считано в порт ввода, его значение присвоено переменной VALUE1. Переменная слева от знака равенства в операции ввода PL/M может являться элементом массива PL/M или структуры.

Операция на языке проектирования

УСТАНОВИТЬ ЗНАЧЕНИЕ ВЫХОД 1 РАВНО 0

эквивалентна командам языка ассемблера

```
MVI   A, 0
OUT   1
```

На языке PL/M эквивалентной является операция

```
OUTPUT(1)=0;
```

Параметр внутри скобок является номером порта вывода. Выражение справа от знака равенства определяет значение данного, которое должно быть послано в порт вывода. Для определения этого значения может быть использовано любое действительное выражение PL/M.

Манипуляция стеком на PL/M производится таким же образом, что и на языке ассемблера. Действительно, поскольку операции PL/M CALL и RETURN транслируются в соответствующие команды языка ассемблера, использование стека в точности соответствует тому, как это описывается на языке ассемблера. Инициализация указателя стека может быть выполнена либо автоматически компилятором PL/M, либо про-

граммистом. В любом случае это должно быть сделано в ИСПОЛНИТЕЛЬНОЙ процедуре до того, как будет вызвана любая другая процедура. Для стека назначается область памяти, расположение которой определяется позднее, когда законченная программа будет размещаться в ПАМЯТИ. Вопросы *размещения программ* будут рассмотрены в гл. 7.

Как уже указывалось, передача параметров через стек осуществляется автоматически компилятором PL/M, который использует при этом описание данных. Для обеспечения правильной передачи всех параметров компилятор вставляет команды языка ассемблера как в вызывающую, так и в вызываемую процедуры.

Для обработки прерываний необходимо идентифицировать процедуры *обработки* каждого запроса на прерывание. Идентификация процедур производится при описании процедур следующим образом:

```
PROCESS$TIMER$INTERRUPT: PROCEDURE INTERRUPT 0;
```

Это описание процедуры содержит информацию, позволяющую компилятору автоматически вставить в ячейку памяти 0 команду *перехода* к процедуре PROCESS$TIMER$INTERRUPT (рис. 6.28). Компилятор вставляет также команды, соответствующие операциям языка проектирования ЗАПОМНИТЬ СОСТОЯНИЕ СИСТЕМЫ и ВОССТАНОВИТЬ СОСТОЯНИЕ СИСТЕМЫ, что также показано на рис. 6.28. Векторам прерываний в ячейках 8, 16, ..., 56 в описаниях процедур ставятся в соответствие числа 1, 2, ..., 7.

Для запросов на прерывания типа TRAP, RST 5.5, RST 6.5 и RST 7.5 необходимо предусмотреть команды, устанавливающие векторы прерываний в ячейках памяти 36, 44, 52 и 60 соответственно. Векторы прерываний могут быть также установлены и в других ячейках памяти, однако это выходит за рамки наших обсуждений.

Команды *Разрешить прерывание* и *запретить прерывание* реализуются в PL/M операциями ENABLE и DISABLE, при этом ответственность за их присутствие в процедурах возлагается на программиста.

Операции, эквивалентные командам для обработки прерываний SIM и RIM, реализуются в PL/M двумя заранее объявленными процедурами. Для установки масок используется процедура S$MASK, а для чтения информации маски — процедура R$MASK. Если какая-либо из них используется, она должна быть описана как внешняя процедура. Описание S$MASK в качестве внешней процедуры имеет следующий

вид:

```
S$MASK: PROCEDURE (MASK) EXTERNAL;
  DECLARE MASK BYTE;
END S$MASK;
```

а процедуры R$MASK:

```
R$MASK: PROCEDURE BYTE EXTERNAL;
END R$MASK;
```

Для того чтобы установить маски прерывания, переменная MASK вначале устанавливается в соответствии со значением, определяемым информацией, показанной на рис. 6.29. Затем вызывается процедура S$MASK, при этом MASK используется в качестве параметра. Так, следующие операции устанавливают маски для запросов на прерывания RST 7.5 и RST 5.5 и сбрасывают маску для запроса на прерывание RST 6.5:

```
MASK = 13;
CALL S$MASK (MASK);
```

Для считывания информации маски операция PL/M имеет вид

```
MASK = R$MASK;
```

После выполнения этой операции переменная MASK содержит информацию маски прерываний в соответствии с рис. 6.30.

Наконец, для того, чтобы процедура PL/M была реентерабельной, используется описание процедуры в виде

```
STOP$TIMER: PROCEDURE REENTRANT;
```

Компилятор PL/M определяет пространство в стеке для каждой переменной, массива и структуры, описанных в процедуре. Благодаря этому процедура может быть вызвана во время обработки прерывания даже в том случае, если прервано было ее собственное выполнение. Запомните, что каждая процедура, вызываемая реентерабельной процедурой, должна быть также объявлена реентерабельной.

Мы закончили обсуждение архитектуры микропроцессора и материала, оставшегося неоконченным в гл. 5. Далее мы рассмотрим средства разработки, отладки и интеграции программного обеспечения.

6.16. Упражнения

6.1. Выберите спроектированную вами простую процедуру и конвертируйте ее в команды языка ассемблера для микрокомпьютера Intel 8085 или подобного ему. Используйте при этом руководство, поставляемое изготовителем, если вам недостаточно материала, представленного в книге, или если вы вы-

брали другой микрокомпьютер. Сравните трудности, возникающие при конвертировании описания процедуры на языке проектирования в язык ассемблера, с конвертированием в один из языков высокого уровня, например PL/M.

6.2. В отдельной процедуре не следует мешать конструкции языка высокого уровня с командами языка ассемблера. Однако в отдельной системе возможна комбинация нескольких процедур, которые должны конвертироваться в язык высокого уровня, с другими процедурами, конвертируемыми в язык ассемблера. Конечно, в таких системах, содержащих оба типа процедур, с целью их правильного функционирования должны использоваться непротиворечивые соглашения для взаимосвязи между модулями и процедурами. Какие преимущества дает конвертирование процедур в язык ассемблера по сравнению с языком высокого уровня? Какие недостатки возникают при этом? При каких условиях преимущества могут возобладать над недостатками?

Глава 7

Разработка, отладка и объединение программного обеспечения

ЧАСТЬ I. ОСНОВНЫЕ СРЕДСТВА

Цикл проектирования системы состоит из разработки требований пользователей, преобразования этих требований в функциональную спецификацию системы, создания описания системы на языке проектирования, конвертирования проекта в аппаратно-программную реализацию, объединения частей проекта, его отладки и проверки правильности работы. На каждом шаге цикла проектирования должна создаваться и поддерживаться документация. Бóльшая часть документации программного обеспечения состоит из *текстов*. Поэтому мы должны иметь возможность производительно создавать и модифицировать тексты. Мы должны иметь возможность транслировать программные модули, содержащие процедуры, в модули на машинном языке и связывать их в работающую систему. Мы должны также иметь возможность тщательно отлаживать работу системы, чтобы быть уверенными, что она работает правильно. Существуют специальные вспомогательные *средства,* применяющиеся на каждом шаге цикла проектирования. В данной главе мы рассмотрим, каким образом некоторые из этих средств могут быть использованы для системной разработки, отладки и объединения программного обеспечения систем, основанных на использовании микрокомпьютера. Средства для отладки аппаратуры и интеграции программного обеспечения в аппаратуру описываются в гл. 9 и 10.

7.1. Средства разработки систем

Средства разработки систем, которые мы описываем в следующих разделах и в гл. 10, состоят из программных средств, которые реализованы либо на микрокомпьютерной системе разработки, встраиваемой в разрабатываемую систему, либо на большой ЭВМ общего назначения. Более того, эти средства разработаны при тех же ограничениях, которые накладываются на разработку всего программного обеспечения. Поэтому каждое средство удовлетворяет набору требований пользователей и функциональной спецификации, которая была разработана группой программистов, проектировавших это средство. В ре-

зультате средства, которые проектировались для одной и той же цели, часто значительно отличаются тем, как они достигают свою цель. Из-за этих различий мы можем описать только то, что́ эти средства должны делать вообще. Когда мы приводим конкретные примеры, необходимо понимать, что они служат только для иллюстрации. Результаты, полученные при помощи двух подобных средств, могут отличаться даже в том случае, если они выполняют подобные задачи. Хотя описания многих из этих средств являются независимыми от типа ЭВМ,

Организация информации	Проектирование программного обеспечения	Конструирование программного обеспечения	Объединение программного обеспечения
Операционная система	Автоматический редактор	Автоматический редактор	План интеграции
Информационные файлы	Эмулятор сквозного просмотра	Транслятор	Файлы команд
	Генератор дерева вызова	Редактор связей	Трассировщик
		Файлы команд	Монитор
			Внутрисхемный эмулятор
			Микрокомпьютерный анализатор

Рис. 7.1. Средства разработки систем.

на которой эти средства должны быть реализованы, мы предположили вначале, что эти средства реализованы на микрокомпьютерной системе разработки или встроены в разрабатываемую систему. После того как опишем основные средства, мы покажем, как используются для разработки микрокомпьютерного программного обеспечения средства, реализованные на большой ЭВМ.

На рис. 7.1 перечислены некоторые средства в соответствии с их основными функциями. По мере того как мы будем описывать эти средства, читатель должен обращаться к рисунку, чтобы представлять себе место каждого из них в общей картине. Некоторые из этих средств, например редакторы, работают независимо от разрабатываемой аппаратуры. Другие, например *внутрисхемный эмулятор,* используются совместно с разрабатываемой аппаратурой. Наконец, такие средства, как *трассировщик,* встраиваются в прикладное программное обеспечение и могут быть использованы во время всех фаз программного и аппаратного объединения.

Большинство из перечисленных средств обрабатывают текстовую информацию. Текстовая информация должна храниться таким образом, чтобы быть легкодоступной для обработки.

Поэтому мы рассмотрим сначала методы хранения текстовой информации и доступа к ней, используемые во время разработки микрокомпьютерных систем.

7.2. Организация информации

При использовании языка проектирования для написания программ и конвертирования их в программы на языке программирования обычно бывает достаточно ручки и бумаги. Однако уже при использовании автоматизированных средств для трансляции операций языка программирования в команды машинного языка первые должны иметь машиночитаемый формат. Для хранения информации в такой форме могут служить перфоленты, перфокарты или устройства магнитной памяти. Многие системы используют для этой цели устройства памяти на *магнитных дисках*. Мы будет предполагать, что вся информация, которая создается вновь или обрабатывается в течение цикла проектирования системы, хранится на магнитных дисках. Средства для идентификации хранимой информации и для доступа к ней предусматриваются *операционной системой*.

С целью облегчения доступа к хранимой на магнитных дисках информации с помощью операционной системы информация разбивается на файлы. Каждому файлу присваивается имя, состоящее из двух частей: из символьного *имени файла* и *расширения имени*. Для чтения информации с диска или записи на диск он должен быть поставлен на устройство, называемое дисководом. Так как система может иметь более одного дисковода, наряду с именем файла и расширением имени должен быть определен *идентификатор дисковода*. Примером имени файла, используемого для хранения модуля EXECUTIVE, может являться символьное имя EXEC. В микрокомпьютерной системе разработки фирмы Intel имена файлов ограничиваются до шести символов. Расширение имени, используемое с каждым именем файла, должно выбираться таким образом, чтобы облегчить распознавание различных файлов, связанных с каждым модулем или процедурой. Например, файлам, содержащим тексты на языке проектирования и/или программирования, может быть присвоено расширение PLM. Файлам, содержащим команды машинного языка, сгенерированным компилятором PL/M, может быть присвоено расширение OBJ (объектный файл). Если магнитный диск поставлен на дисковод номер один, спецификации файла, содержащего программные тексты, и файла, содержащего машинные команды, могут быть представлены в следующем виде:

```
:F1:EXEC.PLM
:F1:EXEC.OBJ
```

Описанные нами соглашения установлены для операционной системы ISIS, используемой микрокомпьютерной системой разработки фирмы Intel. Другими системами могут использоваться другие соглашения и другие символы, отличные от точки и двоеточия.

В следующих разделах мы предполагаем, что каждое из описываемых средств использует возможности операционной системы для чтения информации из определенного файла на диске и для записи информации в файл на другом диске. Последний файл может быть связан с предыдущим, т. е. он может иметь то же имя с другим расширением или иметь полностью отличающееся имя. Мы начнем с *редактора*, который является средством для создания и модификации информации, хранимой в файлах на диске.

7.3. Редактор

Следом за разработкой функциональной спецификации в рамках цикла проектирования системы идет этап создания программного обеспечения, который состоит из написания текста для каждой процедуры и модуля и хранения текста в соответственно поименованных файлах. Для этой цели используется *редактор*, или *редактор текста.* Редактор используется также для внесения изменений в текст и коррекции ошибок при необходимости. Широко известны два типа редакторов: *редактор строк* и *маркерный редактор.*

Редактор строк. Редактор строк обрабатывает информацию двумя способами. В простейшем случае каждой строке текста присваивается номер независимо от того, является ли это комментарием (как, например, операция языка проектирования) или операцией языка программирования. Номера строк могут быть присвоены оператором или назначены автоматически редактором по требованию оператора. Например, оператор может потребовать, чтобы первой строке был присвоен номер 100, а номер каждой последующей строки увеличивался по сравнению с предыдущей на 10. В этом случае номера строк следовали бы следующим образом: 100, 110, 120, ... и т. д. до тех пор, пока не будет напечатан весь текст или пока оператор не прекратит присваивание номеров. Ошибки корректируются путем перепечатывания строк, содержащих ошибки. Например, если строка 120 содержит текст

```
120 BEGIN PROCDURE
```

то после того, как будет напечатано

```
120 BEGIN PROCEDURE
```

исправленный текст автоматически заменяет текст, содержащий ошибку. В качестве альтернативы может быть использована команда *Заменить последовательность символов* CD *последовательностью* CED *в строке 120*, которая приведет к тому же результату. Если строка пропущена, ей может быть присвоен любой номер в интервале между двумя номерами, между которыми должна быть вставлена пропущенная строка. Если строка будет напечатана с таким номером, она автоматически будет вставлена между двумя строками.

Для проверки правильности текста по команде *Печать* или *Список* может быть выдан полный текст на устройстве отображения, которым обычно бывает пишущая машинка или дисплей телевизионного типа, называемый ЭЛТ-терминалом. Текст может быть также выдан на устройстве печати. Если необходимо вставить строку между двумя другими соседними строками, редактору может быть дана команда *Изменить последовательность текста*, в результате которой могут быть присвоены новые номера пустым строкам. Затем могут быть вставлены новые строки, как было описано выше.

Другим способом, с помощью которого редактор может устанавливать последовательность строк, является использование *указателей*. Указатели определяют конкретную строку и конкретный символ. Затем выполняется команда редактора, относящаяся к символу и строке, определенными указателями. Типичными командами модификации текста являются команды *Удалить следующие четыре строки, Удалить следующие три символа* и *Вставить следующий текст*. Существуют команды для перемещения указателя без изменения текста. Например, команды *Переместить указатели назад* (или *вперед*) *на пять строк* и *Переместить указатели в начало* (или *в конец*) *текста* являются командами такого типа.

Редакторы строк имеют также команды *Поиска* и *Замены*, например *Найти и выдать следующее вхождение строки символов* CALL, или *Заменить следующее* (или *каждое*) *вхождение строки символов* CALL: *строкой* CALL.

Поскольку печатание длинных команд является утомительным занятием, обычно команды укорачиваются, часто до одного символа. Например, вместо команды *Удалить следующие три символа* часто применяются сокращения *УЗ* или *ЗУ*. К сожалению, стандартизация в этом деле практически отсутствует, и часто разные редакторы используют различные обозначения для одних и тех же команд, и наоборот. Это отсутствие стандартизации может привести к разрушению информации, когда персонал, работающий с общеинженерными средствами, встречает различные редакторы.

Маркерный редактор. Маркерный редактор выдает текст на экране ЭЛТ-терминала в том виде, в котором он набран оператором. Изменения вносятся с помощью перемещения *маркера* (полоски, светового пятна или стрелки) в то место, где необходимо внести изменение, и набора текста непосредственно на экране дисплея. В большинстве маркерных редакторов предусмотрены средства для вставки символов в существующую строку или удаления символов из строки без полной перепечатки текста. Подобным же образом сравнительно легко может быть вставлена или удалена целая строка. Поскольку между тем, что появляется на экране, и текстом в создаваемом или редактируемом файле существует взаимно-однозначное соответствие, можно легко выполнить любую коррекцию, а также уменьшить количество перепечатываемых символов по сравнению с редактором строк.

Редактор текста является общеупотребительным средством для генерирования и обновления текстов безотносительно к тому, являются ли эти тексты общей документацией, текстом на языке проектирования или на языке программирования. Однако редактор текста не уменьшает общее количество печатания, необходимого для создания новых текстов. Все новые тексты должны быть когда-либо введены инженером, программистом или техником. Было разработано средство, являющееся расширением маркерного редактора, с целью оказания помощи при автоматической генерации текстов. Мы будем называть это средство *автоматизированным редактором* и опишем его в следующем разделе.

7.4. Автоматизированный редактор

В дополнение к функциям, обеспечиваемым маркерным редактором, автоматизированный редактор предусматривает следующие функции: автоматическую генерацию конструкций языка проектирования, автоматическую генерацию строк текста, автоматическое смещение строк, полуавтоматическое конвертирование текстов на языке проектирования в язык программирования и автоматическое извлечение из листинга текста на языке проектирования.

Невозможно полностью автоматизировать генерацию новых текстов. Однако многие слова, фразы и конструкции неоднократно повторяются в процедурах и модулях языка проектирования. Автоматический редактор позволяет вставлять в текст целые слова, фразы или конструкции путем использования функциональной клавиши. Текст, вставляемый с помощью функциональной клавиши, может состоять из конструкций языка проектирования, из последовательностей слов, определенных

пользователем, или из операций используемого языка программирования. Строки текста языка проектирования, которые могут быть сгенерированы автоматически, включают полные заголовки описаний на языке проектирования, содержащие имя программиста, дату, имена процедур и модулей и т. д. Последовательности символов, определенные пользователем, включают имена подсистем, модулей процедур, структур данных и параметров с тем, чтобы не повторять эти имена при печатании каждый раз, когда они встречаются. Стандартные строки текста языка программирования включают такие слова, как BYTE и ADDRESS, для языка PL/M. Таким образом, назначая строки текста функциональным клавишам и используя ключевые слова языка проектирования с помощью автоматизированного редактора, можно значительно сократить количество печатаемых символов. Таблица имен, поддерживаемая автоматизированным редактором, содержит строки, определенные пользователем, и может быть распечатана и использована как список имен для второго уровня документации.

Автоматизированный редактор обеспечивает автоматическое смещение левого края строки каждый раз, когда с помощью функциональной клавиши вставляется конструкция DO или END. Это свойство избавляет проектировщика от необходимости вставлять пробелы вручную или использовать табуляцию.

Если в результате внесенных изменений возникает необходимость изменить смещения, то это происходит автоматически. При необходимости перемещается каждая строка текста, обеспечивая надлежащее смещение конструкций языка проектирования или языка программирования. При этом могут быть легко обнаружены ошибки, появляющиеся вследствие отсутствия или избытка конструкций типа DO или END. Даже при отсутствии ошибок такого типа возможно, что полученный текст будет значительно отличаться от того, что хотел получить проектировщик. Если это так, можно вставить в подходящие места *скобки* DO ... END, чтобы исправить текст и проверить его после изменения смещения.

Когда модуль или процедура программы на языке проектирования закончены, они должны быть конвертированы в программу на языке программирования. Как указывалось ранее, это означает преобразование каждой операции языка проектирования в комментарий и вставление соответствующих операций языка программирования в текст. Преобразование стандартных конструкций языка проектирования в комментарии, раздвигание этих комментариев для того, чтобы было можно вставить текст на языке программирования, и вставка конструкций языка программирования могут быть выполнены автоматически. Подобным же образом имена в программе на языке проекти-

рования могут быть автоматически конвертированы в символы языка программирования. Это преобразование может быть инициировано с помощью функциональной клавиши. Для языка PL/M к каждой строке комментариев добавляется пара ограничителей /* и */. Затем после каждой стандартной конструкции языка проектирования вставляется новая строка и, если возможно, в нее помещается эквивалентная операция PL/M. Например, конструкция ВЫПОЛНЯТЬ НЕПРЕРЫВНО транслируется в DO WHILE 1 = 1; следующим образом:

```
/* ВЫПОЛНЯТЬ НЕПРЕРЫВНО */
   DO WHILE 1 = 1;
```

Смещение строк на языке программирования устанавливается в соответствии с текстом языка проектирования. Имена преобразуются в последовательность символов PL/M с соответствующим вставлением знака доллара и точек. После того как выполнено, насколько это возможно, автоматическое конвертирование, оставшиеся операции PL/M могут быть конвертированы вручную. Во многих случаях последний шаг может быть облегчен использованием стандартных последовательностей символов языка программирования или строк, определенных пользователем и вставляемых с помощью функциональных клавиш.

После этого текстовый файл содержит программные модули или процедуры как на языке проектирования, так и на языке программирования. Полный текст может быть распечатан на пишущей машинке или на устройстве печати. Если он распечатывается на устройстве печати, строка на языке программирования сдвигается вправо, как показано ниже, чтобы обеспечить формат документации, которого мы придерживаемся в примерах данной книги.

```
/* ВЫПОЛНЯТЬ НЕПРЕРЫВНО                    */
                              DO WHILE 1 = 1;
```

В случае использования языка ассемблера комментарии на *языке проектирования* сдвигаются вправо, как показано в примерах гл. 6. Вместо общего листинга пользователь может запросить листинг, который содержит только часть текста, содержащую комментарии на языке проектирования. В этом случае строки текста на языке программирования удаляются из текста, как и все символы, обозначающие комментарии. Таким образом, мы можем извлечь оригинальный текст на языке проектирования даже в том случае, если он уже конвертирован в текст на языке программирования. Последнее свойство автоматического редактора позволяет нам поддерживать в единственном текстовом файле документацию для каждого модуля и процедуры как на уровне языка проектирования, так и на уровне языка программирования.

7.5. Эмулятор сквозного просмотра

Как было подчеркнуто, чем скорее обнаружены ошибки в проекте в течение цикла проектирования системы, тем легче они могут быть исправлены. Просмотр описания программы на языке проектирования позволяет найти ошибки на уровне языка проектирования. Однако чтобы получить при этом максимальный выигрыш, от участников требуется большое напряжение сил. *Эмулятор сквозного просмотра* может несколько уменьшить усилия, требуемые во время просмотра, путем автоматического пошагового просмотра программы на языке проектирования. Определение последовательности процедур в результате *вызова* и *возврата* не требует от участников перелистывания страниц, как это происходит при ручном сквозном просмотре. Для каждой конструкции проверки может быть по выбору сохранено состояние просмотра, а затем просмотр может быть возобновлен из сохраненного состояния, чтобы обе альтернативы проверки могли быть оценены при одинаковых условиях. Если требуется, после каждого просмотра может быть автоматически выдана схема потоков данных. Схема потоков данных представляет дополнительную информацию о проекте, которая помогает проверить правильность проекта.

7.6. Генератор дерева вызова

Дерево вызова процедур было введено в гл. 3. Мы предположили тогда, что дерево вызова процедур может быть сгенерировано автоматически. Для этой цели было разработано специальное средство, называемое генератором дерева вызова.

```
ИСПОЛНЕНИЕ
 |--ИНИЦИАЛИЗАЦИЯ СИСТЕМЫ
 |--ОЖИДАНИЕ ВКЛЮЧЕНИЯ ПЕРЕКЛЮЧАТЕЛЯ
 |--ПРОВЕРКА КОНТАКТНЫХ ДЕТЕКТОРОВ
 |--ПРОВЕРКА ДЕТЕКТОРА ДВИЖЕНИЯ
```

Рис. 7.2. Первые два уровня дерева вызова системы охранной сигнализации.

Генератор исследует ИСПОЛНИТЕЛЬНУЮ процедуру программы на языке проектирования и создает первые два уровня дерева вызова (рис. 7.2). Затем исследуется каждая из процедур, вызываемая ИСПОЛНИТЕЛЬНОЙ процедурой, и создается третий уровень дерева. Генератор продолжает исследовать каждый последующий уровень вызываемых процедур, пока не пройдет дерево до самого нижнего уровня, после чего распечатывает полное дерево вызова процедур, как показано

```
ИСПОЛНЕНИЕ
  |--ИНИЦИАЛИЗАЦИЯ СИСТЕМЫ
  |     |--ИНИЦИАЛИЗАИЯ АППАРАТУРЫ
  |     |--ВОССТАНОВЛЕНИЕ СИСТЕМЫ
  |           |--ВОССТАНОВЛЕНИЕ СИГНАЛОВ
  |           |--ОСТАНОВКА ТАЙМЕРА
  |--ОЖИДАНИЕ ВКЛЮЧЕНИЯ ПЕРЕКЛЮЧАТЕЛЯ
  |     |--СЧИТЫВАНИЕ ПЕРЕКЛЮЧАТЕЛЯ
  |--ПРОВЕРКА КОНТАКТНЫХ ДЕТЕКТОРОВ
  |     |--СЧИТЫВАНИЕ КОНТАКТНЫХ ДЕТЕКТОРОВ
  |     |--ОБНАРУЖЕНИЕ НАРУШИТЕЛЯ
  |           |--ВКЛЮЧЕНИЕ СИГНАЛА
  |           |--ЗАПУСК ТАЙМЕРА
  |           |--СЧИТЫВАНИЕ ПЕРЕКЛЮЧАТЕЛЯ
  |           |--СЧИТЫВАНИЕ СОСТОЯНИЯ ТАЙМЕРА
  |           |--ОЖИДАНИЕ ВЫКЛЮЧЕНИЯ ПЕРЕКЛЮЧАТЕЛЯ
  |           |     |--СЧИТЫВАНИЕ ПЕРЕКЛЮЧАТЕЛЯ
  |           |--ВОССТАНОВЛЕНИЕ СИСТЕМЫ
  |                 |--ВОССТАНОВЛЕНИЕ СИГНАЛОВ
  |                 |--ОСТАНОВКА ТАЙМЕРА
  |--ПРОВЕРКА ДЕТЕКТОРА ДВИЖЕНИЯ
  |     |--СЧИТЫВАНИЕ ДЕТЕКТОРА ДВИЖЕНИЯ
  |     |--ПРОВЕРКА ПРОДОЛЖЕНИЯ ДВИЖЕНИЯ
  |     |     |--ЗАПУСК ТАЙМЕРА
  |     |     |--СЧИТЫВАНИЕ ДЕТЕКТОРА  ДВИЖЕНИЯ
  |     |     |--СЧИТЫВАНИЕ СОСТОЯНИЯ ТАЙМЕРА
  |     |     |--ОСТАНОВКА ТАЙМЕРА
  |     |     |
  |     |--ОБНАРУЖЕНИЕ НАРУШИТЕЛЯ
  |           |--ВКЛЮЧЕНИЕ СИГНАЛА
  |           |--ЗАПУСК ТАЙМЕРА
  |           |--СЧИТЫВАНИЕ ПЕРЕКЛЮЧАТЕЛЯ
  |           |--СЧИТЫВАНИЕ СОСТОЯНИЯ ТАЙМЕРА
  |           |--ОЖИДАНИЕ ВЫКЛЮЧЕНИЯ ПЕРЕКЛЮЧАТЕЛЯ
  |           |     |--СЧИТЫВАНИЕ ПЕРЕКЛЮЧАТЕЛЯ
  |           |--ВОССТАНОВЛЕНИЕ СИСТЕМЫ
  |                 |--ВОССТАНОВЛЕНИЕ СИГНАЛОВ
  |                 |--ОСТАНОВКА ТАЙМЕРА
```

Рис. 7.3. Дерево вызова процедур системы охранной сигнализации.

в примере на рис. 7.3. Если процедурам в целях отладки присвоены номера, они должны быть вставлены в дерево вызова процедур, хотя на рисунке они и не показаны. Дерево вызова процедур может быть создано из части текста на языке проектирования даже после того, как описание проекта конвертировано в язык программирования. Таким образом, дерево вызова процедур может быть сгенерировано после внесения изменений в описание проекта.

Генератор дерева вызова может также использоваться для выборочной вставки списка вызываемых процедур в заголовок каждой процедуры на языке проектирования.

7.7. Компилятор

Компилятор транслирует модули и процедуры, написанные на языке высокого уровня, например PL/M, в команды машинного языка. Функциональные связи между компилятором и его входами и выходами показаны на рис. 7.4. Вход в компилятор состоит из текстового файла, содержащего операции языка программирования высокого уровня, конвертированные из версии программного обеспечения на языке проектирования. Компилятор считывает входной текст из файла построчно. Если строка содержит комментарий, то она игнорируется. Если строка содержит текст, не являющийся комментарием, компилятор проверяет текст на *синтаксические* ошибки. Если синтаксических ошибок не обнаружено, текст транслируется в одну или несколько команд машинного языка. Эти команды хранятся в файле для последующего использования редактором связей. Компилятор передает также каждую строку входного текста печатающему устройству для создания входного листинга.

Пример входного листинга, созданного PL/M компилятором для ИСПОЛНИТЕЛЬНОГО модуля системы охранной сигнализации, показан на рис. 7.5. В начале листинга помещается *блок заголовка*, содержащий соответствующую информацию: имя модуля, имя файла команд на машинном языке и системную команду, запускающую выполнение компилятора.

За блоком заголовка следует текст на языке PL/M с двумя колонками номеров, смещенных влево. Первая из этих колонок содержит порядковый номер каждой строки текста, не являющейся комментарием. Вторая колонка содержит номера, обозначающие *уровень вложенности* каждой PL/M-операции, установленный в описаниях процедур и конструкциями DO ... END;. Так, строка 11 принадлежит уровню 2, указывающему, что данная строка находится внутри процедуры, а строка 13 принадлежит уровню 3, так как она находится и внутри процедуры, и внутри конструкции DO WHILE 1 = 1; ...

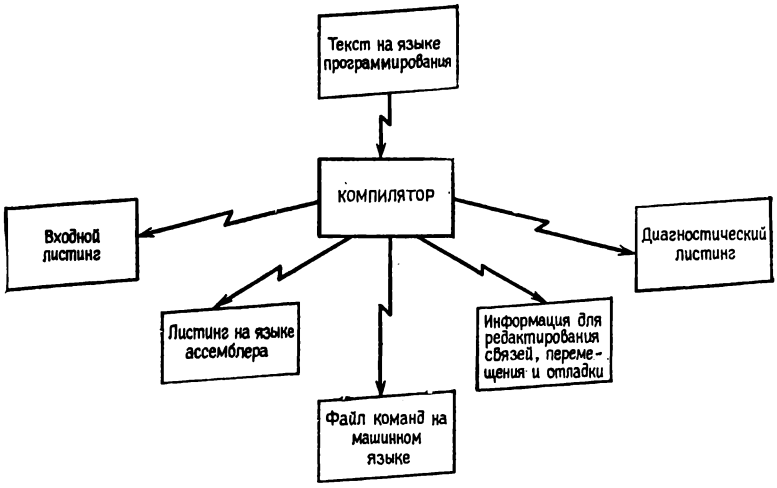


Рис. 7.4. Компилятор.

END;. Следует отметить, что фраза END; рассматривается так, как будто находится внутри процедуры, которую эта фраза ограничивает. Поэтому фраза END; в строке 16 расположена на уровне 3.

В конце входного листинга находится *заключительный блок,* содержащий дополнительную информацию о модуле. В рассматриваемом примере для хранения команд требуется 23 байта памяти, для хранения PL/M-переменных не требуется ни одного байта, поскольку переменные не были описаны, и во время выполнения этого модуля будут необходимы максимум два дополнительных уровня стека. Кроме того, в заключительном блоке указывается, что входной текст содержал 43 строки, включая комментарии на языке проектирования, и что во время трансляции синтаксических ошибок не обнаружено.

Если запрашивается *листинг ассемблера,* который также содержит соответствующие команды машинного языка, созданные компилятором, то он выдается вместе с входным листингом. Листинг ассемблера для ИСПОЛНИТЕЛЬНОГО модуля системы охранной сигнализации показан на рис. 7.6. Запрос осуществляется с помощью *ключевого слова* CODE в управляющей системной команде, содержащейся в блоке заголовка листинга. Листинг ассемблера содержит всю информацию, представленную во входном листинге, а также команды ассемблера и машинного языка, которые чередуются со строками текста PL/M. Например, операция PL/M из строки 11

```
CALL INITIALIZE$SYSTEM;
```

транслируется в

```
0000 CD0000 CALL INITIALIZESYSTEM
```

Первые две колонки содержат адрес команды относительно начала модуля и команду машинного языка в шестнадцатеричном представлении. Третья колонка содержит соответствующую команду языка ассемблера CALL INITIALIZESYSTEM, которая в данном случае почти идентична операции PL/M.

Комментарий языка ассемблера может быть помещен в любом месте строки после точки с запятой. Например, оба следующих предложения

```
; STATEMENT ≠ 10
; PROC EXECUTIVE
```

являются комментариями ассемблера. Комментарии на языке проектирования могут оказаться отделенными от соответствующих операций PL/M, если компилятор вставит между ними команды ассемблера. Например, команда ассемблера

```
000D CD0000 CALL TESTSWITCHES
```

была вставлена между комментарием PL/M

```
/* CALL: TEST MOTION DETECTOR (;) */
```

и соответствующей операцией PL/M

```
CALL TEST$MOTION$DETECTOR;
```

Это является особенностью компилятора PL/M, который иногда печатает комментарий прежде, чем напечатает команду ассемблера, соответствующую предыдущей операции PL/M. Примером, в котором операция PL/M и соответствующие команды ассемблера следуют непосредственно одна за другой, являются операция

```
DO WHILE 1 = 1;
```

и соответствующая ей последовательность команд ассемблера.

Метки ассемблера вставляются в листинг, если это требуется командами перехода. Например, @1: и @2: являются метками, требуемыми командами перехода JMP @1 и JNZ @2 соответственно.

Прежде чем расстаться с этим примером, сделаем еще два замечания. PL/M-операция DO WHILE 1 = 1; (строка 12) является единственной операцией в нашем примере, которой соответствует более одной команды ассемблера. Как указывалось ранее, обычно это не так. Второе замечание состоит в том, что компилятор вставляет команду *возврата* (или RET) в конце процедуры. Это делается компилятором для *каждой* процедуры независимо от того, нужно это или нет и имеется ли

```
PL/M-80 COMPILER

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE EXECUTIVE
OBJECT MODULE PLACED IN :F1:EXEC.OBJ
COMPILER INVOKED BY: PLM80 :F1:EXEC.PLM

                /* MODULE: EXECUTIVE                                    MODULE #1  */
                                                EXECUTIVE: DO;
                /*************************************************************** */
                /* DESIGNED BY: MDF                15-OCT-80                       */
                /*************************************************************** */
                /*    EXTERNAL PROCEDURE DECLARATIONS                              */
                /*                                                                 */
  2   1                                             INITIALIZE$SYSTEM: PROCEDURE EXTERNAL;
  3   2                                             END INITIALIZE$SYSTEM;
                /*                                                                 */
  4   1                                             WAIT$FOR$KEYSWITCH$ARMED: PROCEDURE EXTERNAL;
  5   2                                             END WAIT$FOR$KEYSWITCH$ARMED;
                /*                                                                 */
  6   1                                             TEST$SWITCHES: PROCEDURE EXTERNAL;
  7   2                                             END TEST$SWITCHES;
                /*                                                                 */
  8   1                                             TEST$MOTION$DETECTOR: PROCEDURE EXTERNAL;
  9   2                                             END TEST$MOTION$DETECTOR;
                /*--------------------------------------------------------------- */
                /* PROCEDURE: EXECUTIVE (;)                                        */
 10   1                                        EXECUTIVE: PROCEDURE PUBLIC;
                /*************************************************************** */
                /*    DESIGNED BY: MDF             15-OCT-80                       */
                /*    MODULE: EXECUTIVE                                            */
                /*************************************************************** */
                /* BEGIN PROCEDURE.                                                */
                /*    CALL: INITIALIZE SYSTEM (;)                                  */
 11   2                                           CALL INITIALIZE$SYSTEM;
                /*    DO FOREVER                                                   */
 12   2                                           DO WHILE 1 = 1;
                /*      CALL: WAIT FOR KEYSWITCH ARMED (;)                         */
 13   3                                             CALL WAIT$FOR$KEYSWITCH$ARMED;
                /*      CALL: TEST SWITCHES (;)                                    */
 14   3                                             CALL TEST$SWITCHES;
                /*      CALL: TEST MOTION DETECTOR (;)                             */
 15   3                                             CALL TEST$MOTION$DETECTOR;
                /*    END;                                                         */
 16   3                                           END;
                /* END PROCEDURE                                                   */
 17   2                                        END EXECUTIVE;
                /*************************************************************** */
                /* END MODULE                                                      */
 18   1                                        END EXECUTIVE;

MODULE INFORMATION:

    CODE AREA SIZE      = 0017H     23D
    VARIABLE AREA SIZE  = 0000H      0D
    MAXIMUM STACK SIZE  = 0002H      2D
    43 LINES READ
    0 PROGRAM ERROR(S)
END OF PL/M-80 COMPILATION
```

Рис. 7.5. Входной листинг PL/M-компилятора исполнительного модуля системы охранной сигнализации.

в конце процедуры PL/M-операция RETURN. Это также является особенностью данного компилятора. Этим самым предотвращается *сквозное выполнение* процедур подряд до самого конца, в результате чего встречающиеся команды или данные могут быть неправильно обработаны. В случае если бы такая ситуация произошла, было бы очень трудно обнаружить ошибку во время отладки.

Если компилятор обнаруживает во время компиляции синтаксическую ошибку, он выдает во входном листинге диагностические сообщения. Для того чтобы проиллюстрировать это, мы ввели синтаксическую ошибку в ИСПОЛНИТЕЛЬНЫЙ модуль системы охранной сигнализации и воспроизвели на рис. 7.7 входной листинг компилятора для неправильной программы. В имя процедуры WAIT FOR KEYSWITCH ARMED внесена ошибка, так что имя в вызове не соответствует имени, используемому в описании внешней процедуры. Поскольку ошибочное имя (или *идентификатор*) не описано, компилятор показывает, что обнаружена ошибка (ERROR # 105). Кроме того, когда имя в конструкции *вызова* содержится в описании внешней процедуры, PL/M-компилятором автоматически предполагается, что это имя является адресной переменной. Поскольку ошибочное имя не описано таким образом, другое сообщение об ошибке, #118, указывает, что имя не описано как адресное. Это иллюстрирует еще одну особенность PL/M-компилятора, а именно его полноту: единственная синтаксическая ошибка часто сопровождается несколькими сообщениями. От-

```
PL/M-80 COMPILER

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE EXECUTIVE
OBJECT MODULE PLACED IN :F1:EXEC.OBJ
COMPILER INVOKED BY: PLM80 :F1:EXEC.PLM CODE

                  /* MODULE: EXECUTIVE                                    MODULE #1 */
    1                                        EXECUTIVE: DO;
                  /*************************************************************** */
                  /* DESIGNED BY: MDF                 15-OCT-80                    */
                  /*************************************************************** */
                  /*    EXTERNAL PROCEDURE DECLARATIONS                           */
                  /*                                                              */
    2   1                                      INITIALIZE$SYSTEM: PROCEDURE EXTERNAL;
    3   2                                      END INITIALIZE$SYSTEM;
                  /*                                                              */
    4   1                                      WAIT$FOR$KEYSWITCH$ARMED: PROCEDURE EXTERNAL;
    5   2                                      END WAIT$FOR$KEYSWITCH$ARMED;
                  /*                                                              */
    6   1                                      TEST$SWITCHES: PROCEDURE EXTERNAL;
    7   2                                      END TEST$SWITCHES;
                  /*                                                              */
    8   1                                      TEST$MOTION$DETECTOR: PROCEDURE EXTERNAL,
    9   2                                      END TEST$MOTION$DETECTOR;
                  /*-------------------------------------------------------------- */
```

```
                /*  PROCEDURE: EXECUTIVE (;)                                    */
  10    1                                       EXECUTIVE: PROCEDURE PUBLIC;
                                               STATEMENT # 10
                        ; PROC  EXECUTIVE
                /*************************************************************** */
                /*    DESIGNED BY: MDF                 15-OCT-80                 */
                /*    MODULE: EXECUTIVE                                          */
                /*************************************************************** */
                /*  BEGIN PROCEDURE                                             */
                /*    CALL: INITIALIZE SYSTEM (;)                               */
  11    2                                           CALL INITIALIZE$SYSTEM;
                                               ; STATEMENT # 11
                /*  DO FOREVER                                                  */
                0000  CD0000          CALL    INITIALIZESYSTEM
  12    2                                           DO WHILE 1 = 1;
                                               ; STATEMENT # 12
                        @1:
                0003  3E01            MVI     A,1H
                0005  FE01            CPI     1H
                0007  C21600          JNZ     @2
                /*      CALL: WAIT FOR KEYSWITCH ARMED (;)                      */
  13    3                                           CALL WAIT$FOR$KEYSWITCH$ARMED;
                                               ; STATEMENT # 13
                /*      CALL: TEST SWITCHES (;)                                 */
                000A  CD0000          CALL    WAITFORKEYSWITCHARMED
  14    3                                           CALL TEST$SWITCHES;
                                               ; STATEMENT # 14
                /*      CALL: TEST MOTION DETECTOR (;)                          */
                000D  CD0000          CALL    TESTSWITCHES
  15    3                                           CALL TEST$MOTION$DETECTOR;
                                               ; STATEMENT # 15
                /*    END;                                                      */
                0010  CD0000          CALL    TESTMOTIONDETECTOR
  16    3                                           END;
                                               ; STATEMENT # 16
                0013  C30300          JMP     @1
                        @2
                /*  END PROCEDURE                                               */
  17    2                                       END EXECUTIVE;
                                               ; STATEMENT # 17
                0016  C9              RET
                /*************************************************************** */
                /*  END MODULE                                                  */
  18    1                                       END EXECUTIVE;

MODULE INFORMATION:

    CODE AREA SIZE      = 0017H     23D
    VARIABLE AREA SIZE  = 0000H      0D
    MAXIMUM STACK SIZE  = 0002H      2D
    43 LINES READ,
    0 PROGRAM ERROR(S)
END OF PL/M-80 COMPILATION
```

Рис. 7.6. Листинг на языке ассемблера PL/M-компилятора исполнительного модуля системы охранной сигнализации.

метим также, что каждое сообщение об ошибке определяет номер утверждения, содержащего синтаксическую ошибку. Иногда сообщение об ошибке не следует непосредственно за

```
PL/M-80 COMPILER

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE EXECUTIVE
OBJECT MODULE PLACED IN :F1:EXEC.OBJ
COMPILER INVOKED BY: PLM80 :F1:EXEC.PLM

                /*  MODULE: EXECUTIVE                                   MODULE #1  */
  1                                                 EXECUTIVE: DO;
                /*****************************************************************  */
                /*  DESIGNED BY: MDF                 15-OCT-80                       */
                /*****************************************************************  */
                /*    EXTERNAL PROCEDURE DECLARATIONS                               */
                /*                                                                  */
  2   1                                               INITIALIZE$SYSTEM: PROCEDURE EXTERNAL;
  3   2                                               END INITIALIZE$SYSTEM;
                /*                                                                  */
  4   1                                               WAIT$FOR$KEYSWITCH$ARMED: PROCEDURE EXTERNAL;
  5   2                                               END WAIT$FOR$KEYSWITCH$ARMED;
                /*                                                                  */
  6   1                                               TEST$SWITCHES: PROCEDURE EXTERNAL;
  7   2                                               END TEST$SWITCHES;
                /*                                                                  */
  8   1                                               TEST$MOTION$DETECTOR: PROCEDURE EXTERNAL;
  9   2                                               END TEST$MOTION$DETECTOR;
```

```
                /*--------------------------------------------------------------------- */
                /*  PROCEDURE: EXECUTIVE (;)                                            */
  10    .                                       EXECUTIVE: PROCEDURE PUBLIC;
                /*********************************************************************** */
                /*    DESIGNED BY: MDF                 15-OCT-80                        */
                /*    MODULE: EXECUTIVE                                                 */
                /*********************************************************************** */
                /*  BEGIN PROCEDURE                                                     */
                /*    CALL: INITIALIZE SYSTEM (;)                                       */
  11    2                                         CALL INITIALIZE$SYSTEM;
                /*    DO FOREVER                                                        */
  12    2                                         DO WHILE 1 = 1;
                /*      CALL: WATCH FOR KEYSWITCH ARMED (;)                             */
  13    3                                           CALL WATCH$FOR$KEYSWITCH$ARMED;
*** ERROR #105, STATEMENT #13, NEAR 'WATCHFORKEYSWITCHARMED'    UNDECLARED IDENTIFIER
                /*      CALL: TEST SWITCHES (;)                                         */
*** ERROR #118, STATEMENT #13, NEAR 'WATCHFORKEYSWITCHARMED',   INVALID INDIRECT CALL, IDENTIFIER NOT AN ADDRESS SCALAR
  14    3                                           CALL TEST$SWITCHES;
                /*      CALL: TEST MOTION DETECTOR (;)                                  */
  15                                                CALL TEST$MOTION$DETECTOR;
                /*    END;                                                              */
  16    2                                         END;
                /*  END PROCEDURE                                                       */
  17    1                                       END EXECUTIVE;
                /*********************************************************************** */
                /*  END MODULE                                                          */
  18    1                                       END EXECUTIVE;

MODULE INFORMATION:

    CODE AREA SIZE      = 0014H    20D
    VARIABLE AREA SIZE  = 0001H     1D
    MAXIMUM STACK SIZE  = 0002H     2D
    43 LINES READ
    2 PROGRAM ERROR(S)
END OF PL/M-80 COMPILATION
```

Рис. 7.7. Входной листинг PL/M-компилятора для модуля, содержащего ошибку.

строкой, содержащей ошибку. Более того, номер утверждения в сообщении об ошибке не всегда указывает место нахождения синтаксической ошибки. Обычно если синтаксическая ошибка находится не в указанной строке, то она находится в строке, непосредственно следующей за сообщением об ошибке. Имеются два исключения из этого правила: (1) Некоторые синтаксические ошибки не обнаруживаются до тех пор, пока не будет проверена последняя строка процедуры или модуля. Сообщение об этой синтаксической ошибке помещается в конце входного листинга перед заключительным блоком. (2) Синтаксическая ошибка в строке может быть не обнаружена до тех пор, пока не будет обнаружено противоречие где-нибудь в другом месте процедуры или модуля. Компилятор не может определить, какая из двух строк содержит синтаксическую ошибку. В сообщении об ошибке поэтому указывается, что синтаксическая ошибка находится в последней строке, там, где она была обнаружена.

Один из выходов компилятора, показанных на рис. 7.4, не появляется ни в одном листинге. Он содержит информацию для редактирования, перемещения и отладки. Эта информация хранится в файле машинных команд и используется, как мы увидим далее, редактором связей и перемещающим загрузчиком, а также для устранения семантических ошибок при отладке.

7.8. Ассемблер

Ассемблер транслирует команды языка ассемблера в команды машинного языка. Он действует наподобие компилятора, за исключением того, что входной листинг является листингом языка ассемблера. Иначе говоря, ассемблер считывает построчно входной текст из файла на диске, игнорирует комментарии ассемблера, проверяет каждую команду на наличие ошибок и транслирует каждую команду в команду на машинном языке. Он также готовит информацию для редактора связей и загрузчика и для отладки.

Пример входного листинга ассемблера для микропроцессора Intel 8085 показан на рис. 7.8. Каждой операции языка проектирования предшествует точка с запятой, указывая, что это комментарий. В этом примере псевдокоманда DS выполняет ту же функцию, что и описание байтовых данных в PL/M. Псевдокоманды DSEG (сегмент данных) и CSEG (сегмент команд) используются для разделения областей памяти, резервируемых для данных, от областей, резервируемых для команд. Кроме того, буквы D и C в третьей колонке указывают, ссылается

адресная часть команды на адрес в сегменте данных или в сегменте команд. Эта информация необходима, как мы увидим позднее, при организации связей.

7.9. Файл машинных команд

Одним из выходов, создаваемых как компилятором, так и ассемблером, является файл, содержащий команды машинного языка для транслируемого модуля. Когда они выполняются, это означает выполнение операций языка программирования, которые содержались в исходном тексте на языке программирования. Однако прежде, чем команды машинного языка могут быть выполнены, они должны быть размещены в памяти микрокомпьютера. Поскольку каждая процедура или модуль транслируются по отдельности, то во время трансляции неизвестно, где будут расположены в ПАМЯТИ команды других модулей. Возникает вопрос, как собрать оттранслированные по отдельности программные модули в работающую систему? Чтобы ответить на него, рассмотрим концепцию *перемещаемости,* которая приведет нас к понятиям редактирования связей и перемещающей загрузки.

Перемещаемым модулем называется такой модуль, который может быть размещен в любом месте ПАМЯТИ *после того,* как он оттранслирован. Во время трансляции предполагается, что первой ячейкой памяти, которая может быть занята командами машинного языка каждого модуля, является нулевая ячейка. Все остальные ячейки памяти, необходимые для модуля, назначаются так, как будто это предположение является правильным. Для того чтобы можно было восстановить команды обращения к памяти, во время трансляции создается информация для редактора связей, перемещающего загрузчика и отладки. Эта информация вместе с командами на машинном языке хранится в файле команд на машинном языке, или *объектном файле.* Позже, когда модули объединяются, или *связываются* вместе, и когда каждому модулю назначается специальная область в ПАМЯТИ (когда модуль *перемещается в* ПАМЯТИ), адрес в каждой команде обращения к памяти настраивается с помощью вышеуказанной информации для редактора связей и перемещающего загрузчика. Независимо от порядка, в котором совершаются редактирование и перемещение, результат должен быть одним и тем же: каждая команда обращения к памяти должна содержать действительный адрес процедуры, структуры данных или команды.

Для иллюстрации понятия перемещения используем листинг ассемблера ИСПОЛНИТЕЛЬНОГО модуля системы охранной сигнализации из примера на рис. 7.6. Команда JMP @1

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0
 LOC  OBJ        LINE        SOURCE STATEMENT

                    1 ;                         PROCEDURE: MICROCOMPUTER SYSTEM EXAMPLE (;)
                    2 ;                         ************************************************************
                    3 ;                         DESIGNED BY: MDF            LAST UPDATE: 08-07-81 LBE
                    4 ;
                    5 ;                         LOCAL PARAMETERS NOT USED IN DESIGN LANGUAGE:
                    6 ;                                      NAME          TYPE          SIZE
                    7 ;                                      ----          ----          ----
                    8 ;                                      VALUE1        BYTE          --
                    9           DSEG
0000                10 VALUE1:  DS   1
                    11 ;                                     VALUE2        BYTE          --
0000                12 VALUE2:  DS   1
                    13 ;                        ************************************************************
                    14 ;                        BEGIN PROCEDURE
                    15          CSEG
                    16 ;                          DO FOREVER
                    17 ;                            EXAMINE INPUT 1 AND SAVE ITS VALUE
0000 DB01           18 START:   IN   1
0002 320000   D     19          STA  VALUE1
                    20 ;                            EXAMINE INPUT 2 AND SAVE ITS VALUE
0005 DB02           21          IN   2
0007 320000   D     22          STA  VALUE2
                    23 ;                            IF THE VALUE OF INPUT 1 IS GREATER THAN 4 AND LESS THAN 8
```

```
000A 3A0000   D   24          LDA  VALUE1
000D FE04         25          CPI  4
000F CA2400   C   26          JZ   ELSE1
0012 DA2400   C   27          JC   ELSE1
0015 FE06         28          CPI  6
0017 CA2400   C   29          JZ   ELSE1
001A D22400   C   30          JNC  ELSE1
                  31                        THEN  SET OUTPUT 1 TO THE VALUE 6
001D 3E06         32          MVI  A,6
001F D301         33          OUT  1
0021 C32800   C   34          JMP  CONT1
                  35                        ELSE  SET OUTPUT 1 TO THE VALUE 0
0024 3E00         36 ELSE1:   MVI  A,0
0026 D301         37          OUT  1
                  38                    IF THE VALUE OF INPUT 2 IS GREATER THAN 2 AND LESS THAN 6
0028 3A0100   D   39 CONT1:   LDA  VALUE2
002B FE02         40          CPI  2
002D CA4200   C   41          JZ   ELSE2
0030 DA4200   C   42          JC   ELSE2
0033 FE06         43          CPI  6
0035 CA4200   C   44          JZ   ELSE2
0038 D24200   C   45          JNC  ELSE2
                  46                        THEN  SET OUTPUT 2 TO THE VALUE 4
003B 3E04         47          MVI  A,4
003D D302         48          OUT  2
003F C34600   C   49          JMP  CONT2
                  50                        ELSE  SET OUTPUT 2 TO THE VALUE 0
0042 3E00         51 ELSE2:   MVI  A,0
0044 D302         52          OUT  2
                  53                    END
0046 C30000   C   54 CONT2:   JMP  START
                  55                END PROCEDURE
0000          C   56          END  START
```

Рис. 7.8. Листинг на языке ассемблера микрокомпьютера Intel 8085.

является командой безусловного перехода в ячейку памяти 0003. Но эта ячейка памяти была назначена относительно нулевой ячейки памяти, которая рассматривается компилятором как содержащая первую команду процедуры. Когда перемещающим загрузчиком процедуре назначается конкретный адрес загрузки, к адресной части команды прибавляется значение *константы перемещения,* обеспечивая тем самым правильное выполнение команды перехода. Например, если ИСПОЛНИТЕЛЬНАЯ процедура перемещается в памяти таким образом, что ее первая команда помещается в ячейке памяти 1000H, то адресная часть команды безусловного перехода настраивается перемещающим загрузчиком относительно этой ячейки памяти. Таким образом, выполняемой командой будет команда JMP 1003H, или C3 03 10, как это требуется для правильной работы системы. Заметим, что в листинге ассемблера отсутствует информация для редактирования связей перемещения и отладки, что адресные байты в машинных командах хранятся в обратном порядке (03 10 вместо 10 03) и что добавление символа H к числу означает шестнадцатеричную интерпретацию этого числа.

В примере на рис. 7.6 все команды *вызова* имеют один и тот же нулевой адрес. Когда каждому модулю отводится конкретный участок памяти и процедуры связываются вместе, адресная часть каждой команды CALL заменяется действительным адресом соответствующей процедуры. Для того чтобы эти адреса заменялись правильно, информация для связей, перемещения и отладки содержит символические имена каждой вызываемой процедуры. Предусматривается также информация, указывающая, какие команды CALL соответствуют каждому из символических имен. Необходимо заметить, что адреса в командах CALL в листинге ассемблера не обязательно должны быть нулевыми, как показано в примере. Обсуждение вопроса, когда это случается, а также другие конкретные детали использования информации для связей, перемещения и отладки выходят за рамки нашего рассмотрения.

7.10. Редактор связей, перемещающий загрузчик и загрузчик

Как мы уже видели, связывание представляет собой процесс *соединения* процедур через вызовы этих процедур, а перемещение — процесс назначения командам программы конкретных ячеек памяти. В общем случае редактирование связей может быть выполнено как до, так и после перемещения. Однако это в значительной степени зависит от самих программ — редакторов связей и перемещающих загрузчиков, поставляемых изготовителями и поставщиками. Мы предполагаем, что редактор связей и перемещающий загрузчик могут использоваться в любой последовательности. Программы, предназначенные для

редактирования связей и перемещения объектных файлов, полученных с помощью компиляторов и ассемблеров микрокомпьютера Intel 8085, работают именно таким образом.

Входы редактора связей содержат следующую информацию:

• Имена файлов на диске, содержащих машинные команды для каждой процедуры или модуля, которые должны быть связаны.

• Имя файла на диске, в котором должна храниться отредактированная программа.

• Набор необязательных ключевых слов, которые обеспечивают выдачу дополнительной отладочной информации для программиста.

Кроме отредактированной программы редактор связей предусматривает выдачу диагностических сообщений в случае обнаружения ошибок во время редактирования связей.

Если программа, в которой редактируются связи, еще не размещена, редактор связей не может назначить абсолютные адреса памяти для вызовов или для команд обращения к памяти. Он назначает новые перемещаемые адреса и настраивает все адреса в процедурах вызова и командах обращения к памяти относительно нового *начального адреса.* При этом создается новый перемещаемый модуль на машинном языке из отдельных процедур и модулей, которые были связаны вместе. Связанные таким образом модули могут снова связываться вместе до тех пор, пока, наконец, все вызовы не будут привязаны к соответствующим процедурам. После этого перемещающий загрузчик назначает конкретные ячейки памяти каждой команде в связанном модуле и настраивает адреса каждой команды обращения к памяти, завершая при этом генерацию готового к выполнению модуля.

Если перемещающий загрузчик используется до редактора связей или для размещения модуля, содержащего небольшое число связанных модулей, процедурам в этих модулях назначаются абсолютные адреса ячеек ПАМЯТИ. Вызовы в этих модулях соединяются затем с соответствующими процедурами во время последующего шага редактирования связей. При использовании таким способом шаги редактирования связей и перемещения должны чередоваться столько, сколько необходимо для полного редактирования всех связей программы и правильного размещения в ПАМЯТИ.

После того как создан выполняемый программный модуль, он может быть загружен в память для выполнения. Если память состоит из постоянных запоминающих устройств (с возможностью только чтения), процесс загрузки состоит в том, чтобы записать в эти устройства на длительный срок программу на машинном языке. Если используется память типа опера-

тивного запоминающего устройства (с возможностью чтения и записи), такая, например, как во время отладки и объединения, для загрузки программы на машинном языке в ПАМЯТЬ, используется *загрузчик*. В гл. 8 мы рассмотрим постоянные и оперативные запоминающие устройства более подробно.

7.11. Автоматическое конструирование программного обеспечения

Выполнение каждого из описанных программных средств запускается командой операционной системы, которая указывает, что данное средство должно делать. Например, редактору связей должны быть указаны имена файлов на диске, содержащих модули, которые необходимо связать. Для микрокомпьютерной системы разработки фирмы Intel команда операционной системы для запуска редактора связей при редактировании связей системы охранной сигнализации выглядит следующим образом:

```
LINK :F1:EXEC.OBJ, :F1:TEST.OBJ, :F1:WAIT.OBJ, :F1:INTRUD.OBJ, &
     :F1:RESET.OBJ, :F1:TIMER.OBJ, :F1:INPUT.OBJ, :F1:OUTPUT.OBJ &
     PLM80.LIB, SYSTEM.LIB TO :F1:BURG.LNK
```

Символ амперсанда, &, позволяет продолжить команду в другой строке. В примере опущены необязательные ключевые слова. Мы видим, что печатание и ввод команд операционной системы для редактирования связей систем с большим числом модулей и необязательных ключевых слов является занятием довольно утомительным и может стать источником ошибок. Поэтому многие микрокомпьютерные системы разработки позволяют хранить команды операционной системы в файле команд, чтобы не перепечатывать их каждый раз, когда это необходимо. Для выполнения команд, хранящихся в *файле команд* микрокомпьютерной системы разработки Intel, необходимо набрать команду операционной системы

```
SUBMIT :F1:BURG
```

Это означает, что должны быть выполнены команды, хранящиеся в файле BURG.CSD на магнитном диске, находящемся в дисководе номер один. CSD — пассивное расширение имени файла, определенного в команде SUBMIT.

Пример файла команд, содержащий последовательность системных команд, необходимых для запуска PL/M-компилятора, редактора связей и перемещающего загрузчика:

```
PLM80 :F1:%0.PLM
LINK :F1:EXEC.OBJ, :F1:TEST.OBJ, :F1:WAIT.OBJ, :F1:INTRUD.OBJ, &
     :F1:RESET.OBJ, :F1:TIMER.OBJ, :F1:INPUT.OBJ, :F1:OUTPUT.OBJ, &
     PLM80.LIB, SYSTEM.LIB TO :F1:BURG.LNK
LOCATE :F1:BURG.LNK MAP
```

В этом примере имя файла, содержащего предназначенный для компиляции модуль PL/M, явно не определено, вместо него используются символы %0. Если в одном из модулей или в процедуре меняется текст, например, с целью коррекции ошибки, обнаруженной во время интеграции, то необходимо компилировать снова только один модуль или одну процедуру, где обнаружена ошибка. Однако для того, чтобы включить новую версию скорректированного модуля или процедуры в систему, необходимо снова редактировать связи всей системы, а затем снова размещать ее в памяти. Представим, что эти команды хранятся в файле BURG.CSD на магнитном диске, находящемся на дисководе номер один. Команда SUBMIT, заменяющая символы %0 именем файла INPUT и инициирующая выполнение команд в файле, имеет следующий вид:

```
SUBMIT :F1:BURG (INPUT)
```

Эта команда вызывает аналогичные действия, что и команда

```
PLM80 :F1:INPUT.PLM
```

которая может быть определена явным образом перед командами LINK и LOCATE в вышеприведенном примере.

7.12. Использование большой ЭВМ

В предыдущем разделе мы предполагали, что программные средства, используемые для проектирования программного обеспечения, реализованы на микрокомпьютерной системе разработки. Одной из альтернатив является использование на начальных фазах проектирования средств, реализованных на большой ЭВМ. Информация, хранимая в файлах на магнитном диске большой ЭВМ, может быть перенесена на магнитный диск микрокомпьютерной системы разработки, чтобы завершить выполнение остальных фаз цикла проектирования системы. Процесс переноса информации магнитного диска большой ЭВМ на магнитный диск микрокомпьютерной системы разработки называется *выгрузкой*. Рассмотрим кратко преимущества и недостатки использования большой ЭВМ в цикле проектирования системы.

Емкость магнитной дисковой памяти в микрокомпьютерной системе разработки ограничена. Поэтому может оказаться необходимым разделить программу между несколькими магнитными дисками. Затем, чтобы связать программу, созданные транслятором объектные файлы должны быть скопированы на общий магнитный диск, чтобы все они одновременно были доступны редактору связей. Большая ЭВМ имеет бóльшую емкость дисковой магнитной памяти, поэтому все программное

обеспечение может быть доступным одновременно. Одним из эффективных способов работы является сопровождение программных версий на языке проектирования и на языке программирования только на большой ЭВМ. После того как модуль завершен, он выгружается на временный или *рабочий* диск, транслируется в машинный язык и хранится на диске в микрокомпьютерной системе разработки только в виде объектного файла. В другом случае, когда на большой ЭВМ имеется соответствующий транслятор языка программирования, трансляция может быть выполнена на большой ЭВМ до выгрузки. В любом случае две системы (большая ЭВМ и микрокомпьютер) должны выполнять для группы программистов взаимно исключающие функции.

Если группа программистов состоит из двух или более человек, то в случае отсутствия большой ЭВМ может потребоваться несколько микрокомпьютерных систем разработки, чтобы уменьшить время ожидания на разработку системы. Микрокомпьютерные системы разработки являются значительно более дорогостоящими, чем рабочие места или терминалы для большой ЭВМ. Так, если используется большая ЭВМ, можно за умеренную цену обеспечить отдельный терминал для каждого программиста из группы разработки программного обеспечения. Микрокомпьютерные системы разработки необходимы только во время последних фаз цикла проектирования системы после выгрузки программного обеспечения. Поэтому использование больших ЭВМ может уменьшить основные капиталовложения в оборудование, необходимое для разработки проекта.

При использовании любой системы необходимо предусмотреть способы восстановления программного обеспечения после сбоев системы или питания. В следующей главе мы рассмотрим, как это должно делаться для систем, основанных на использовании микрокомпьютера, таких, например, как система охранной сигнализации. При использовании ЭВМ для разработки программного обеспечения необходимо позаботиться о том, чтобы информация, хранящаяся на магнитных дисках, могла быть восстановлена в случае ее разрушения или искажения во время сбоев системы и питания. Наиболее употребительным способом предотвращения потерь информации, хранящейся на магнитных дисках, является периодическое копирование информации на *дублирующие* магнитные диски или магнитные ленты. Эта предосторожность позволит использовать дубликат информации, если основная копия будет разрушена. Если магнитные диски *дублируются* ежедневно, то в случае сбоя будет потеряно не более одной дневной порции работы. Если для разработки программного обеспечения используется большая ЭВМ, периодическое дублирование возлагается на ру-

ководителя группы обслуживания ЭВМ. Если же используется микрокомпьютерная система разработки, ответственность за периодическое дублирование возлагается на руководителя группы проектирования микрокомпьютерной системы. Таким образом, другим преимуществом использования большой ЭВМ для хранения проектной документации является то, что не нужно заботиться о дублировании — это выполняется автоматически и планомерно.

Кроме того, некоторые из специальных программных средств, например генератор дерева вызова, работают в микрокомпьютерной системе разработки значительно медленнее, чем на большой ЭВМ. Поэтому дерево вызова в случае использования большой ЭВМ может обновляться намного чаще.

Недостатком большой ЭВМ является то, что в случае отказа системы технические средства большой ЭВМ становятся недоступными для проектировщиков и работа может замедлиться или остановиться вообще. Если же используется несколько микрокомпьютерных систем разработки, то очень редко все они сразу выходят из строя, работа при этом может быть продолжена, хотя, возможно, более медленными темпами.

Таким образом, большая ЭВМ имеет некоторые преимущества перед микрокомпьютерными системами разработки. Она имеет также и недостатки. Какой подход использовать — это зависит от количества разработчиков, доступных средств, сроков разработки, а также других факторов, которые могут меняться от одного проекта к другому. Решать, какой подход является лучшим в каждом конкретном случае, относится скорее к компетенции руководителя.

Мы упоминали ранее, что как компилятором, так и ассемблером на выходе создается отладочная информация. Далее мы рассмотрим, как с помощью отладочной информации, а также специально предназначенных для этого средств производятся отладка и интеграция программного обеспечения в работающую систему.

ЧАСТЬ II. ОБЪЕДИНЕНИЕ МОДУЛЕЙ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ В СИСТЕМУ

До сих пор мы описывали системный подход к проектированию программного обеспечения и конвертированию программ на языке проектирования в язык программирования. В ч. I настоящей главы мы описали доступные нам средства, которые могут быть использованы при разработке программного обеспечения систем, основанных на использовании микрокомпьютера. Каждое из этих средств может помочь обнаружить различные типы ошибок. Сквозной эмулятор обнаруживает ошибки в потоке управления внутри процедур, а также в потоке данных.

Генератор дерева вызова обнаруживает ошибки в управлении процедурами, а также указывает, какие процедуры должны быть реентерабельными. Трансляторы (компилятор и ассемблер) обнаруживают синтаксические ошибки в языке программирования. Однако и после использования всех этих средств неизбежно остаются как проектные, так и программные ошибки или дефекты. Для отыскания и исправления оставшихся ошибок могут быть использованы программные средства отладки. Во второй части главы мы рассмотрим эти средства, а также системную методологию отладки и объединения модулей программного обеспечения. Применение такой системной методологии объединения модулей гарантирует эффективное использование средств отладки, при этом можно с большой достоверностью считать, что оставшиеся ошибки будут обнаружены и устранены.

7.13. Объединение модулей программного обеспечения

После того как ошибки, обнаруженные с помощью сквозного эмулятора, генератора дерева вызова и транслятора языка программирования, исправлены, имеется большой соблазн отредактировать связи программы, разместить ее, загрузить в память и запустить для выполнения, чтобы посмотреть, работает ли она. Такой подход не рекомендуется по двум причинам. Во-первых, работа программы будет скорее всего непредсказуемой, в результате чего будет почти невозможно определить, какие остались ошибки. Во-вторых, даже если программа работает надлежащим образом, нельзя доказать, что все оставшиеся ошибки устранены, и в то же время обнаружить их будет очень трудно.

Системный подход при объединении модулей программного обеспечения состоит в пошаговой нисходящей проверке и объеденении модулей программного обеспечения следующим образом:

- На первом шаге объединения проверяется работа процедур только самого верхнего уровня: ИСПОЛНИТЕЛЬНОЙ и ИНИЦИАЛИЗАЦИИ СИСТЕМЫ. Проверяются правильность их работы и последовательность вызова процедур нижнего уровня. Во время этого шага выполнение других процедур, вызываемых ИСПОЛНИТЕЛЬНОЙ процедурой, не допускается. Далее мы опишем, как осуществляется выборочное выполнение процедур.
- Отладив ИСПОЛНИТЕЛЬНУЮ процедуру и процедуру ИНИЦИАЛИЗАЦИИ СИСТЕМЫ, можно приступать ко второму шагу и проверить работу одной из процедур, вызываемых ИСПОЛНИТЕЛЬНОЙ процедурой.

• При продолжении второго шага объединения возможны два направления в зависимости от характеристик объединяемых модулей системы. В некоторых системах работа каждой из оставшихся процедур, вызываемых ИСПОЛНИТЕЛЬНОЙ процедурой, проверяется по очереди, и таким образом процедуры объединяются в систему. Затем проверяется по одной работа процедур третьего уровня, и эти процедуры объединяются таким же образом. Этот подход может быть повторен для каждого из оставшихся уровней, пока все оставшееся программное обеспечение полностью не будет отлажено и объединено. Следует отметить, что правильное выполнение условных и циклических конструкций в процедуре зависит от информации, полученной либо от входных параметров процедуры, либо от выходных параметров, возвращаемых вызванными процедурами. Поэтому необходимо иметь возможность изменять значения этих параметров, чтобы тщательнее проверить выполнение процедуры. Далее будет показано, как во время отладки можно изменять такие параметры.

• В системах, состоящих из нескольких различных подсистем, для второго и последующих этапов объединения может оказаться более предпочтительным следующий подход. После того как отлажена одна из ПОДСИСТЕМНЫХ ИСПОЛНИТЕЛЬНЫХ процедур, вызываемых ИСПОЛНИТЕЛЬНОЙ процедурой, отлаживаются и объединяются по одной все процедуры, вызываемые ПОДСИСТЕМНОЙ ИСПОЛНИТЕЛЬНОЙ процедурой, после чего проверяется работа процедур следующего уровня подсистемы и т. д. Затем выполняется каждая из оставшихся ПОДСИСТЕМНЫХ ИСПОЛНИТЕЛЬНЫХ процедур, вызываемых ИСПОЛНИТЕЛЬНОЙ процедурой, и объединяются соответствующие подсистемы. Благодаря взаимодействию между подсистемами может оказаться необходимым при объединении части этих подсистем нарушить естественную последовательность.

• После того как объединение подсистем и модулей, не управляемых прерываниями, закончено, можно приступить к объединению прерываемых программ. Этот шаг объединения выполняется либо путем запуска на выполнение процедуры ИСПОЛНЕНИЯ ПРЕРЫВАНИЙ, если такая существует, либо путем использования программной процедуры для вызова процедуры ОБРАБОТКИ ПРЕРЫВАНИЙ. Объединение программ прерываний производится таким же способом, как было описано выше. Если нет процедуры ИСПОЛНЕНИЯ ПРЕРЫВАНИЙ, выполнение каждой из процедур ОБРАБОТКИ ПРЕРЫВАНИЯ может быть вызвано отдельно, вследствие чего объединение может быть выполнено очень просто.

Описанные шаги объединения часто могут быть выполнены на микрокомпьютерной системе разработки. Для этого микрокомпьютерная система разработки должна иметь возможность для непосредственного выполнения или для *эмуляции* (имитации) выполнения программ. Если эмуляция возможна только на большой ЭВМ, объединение программного обеспечения может быть выполнено на ней. Шаги объединения, во время которых должны объединяться модули ВВОДА, ВЫВОДА и ПРЕРЫВАНИЙ, могут потребовать конструирования и проверки специальной аппаратуры до завершения этих шагов объединения. Поэтому их описание откладывается до тех пор, пока мы не рассмотрим вопросы совместного объединения программного обеспечения и аппаратных средств в гл. 10. Мы также откладываем до гл. 10 описание альтернативного подхода, в котором объединение модулей программного обеспечения комбинируется с объединением частей системы.

Описанная методология объединения является сравнительно простой. Тем не менее необходимо планировать каждый шаг заранее, чтобы обеспечить тщательное проведение объединения частей системы. Важно также планировать, как должны меняться параметры, чтобы объединение прошло гладко. С этой целью необходимо до начала объединения частей системы подготовить *план объединения*. Кроме того, необходимо заранее разработать методы, позволяющие выборочно выполнять процедуры и легко менять параметры. Эта методика составляет основу *средства*, которое мы называем *трассировщиком*. План объединения и реализация трассировщика во время объединения обсуждаются в следующих разделах.

7.14. План объединения

Как отмечалось в предыдущем разделе, объединение микрокомпьютерной системы лучше всего выполнять, группируя шаги, выполняемые во время объединения в этапы. Целью каждого этапа объединения должна являться конкретная работающая часть системы. Например, один из этапов объединения может состоять из шагов, необходимых для полного объединения подсистемы. Другой этап объединения может включать объединения модуля или группы модулей.

План объединения является документом, описывающим этапы объединения. Он состоит из подробного описания каждого этапа процесса объединения и определяет, какая часть системы должна быть работающей в конце каждого этапа. Описание этапа объединения должно указывать, какие процедуры (модули или подсистемы) должны быть проверены и какие функции должна выполнять система после завершения этапа объедине-

ния. Кроме того, должны быть определены проверочные значения для тех параметров, которые должны меняться во время этапа объединения. Если также могут быть оценены или определены каким-либо способом значения вычисляемых параметров, эти значения должны быть включены в план объединения. Это позволит более объективно оценить системы во время объединения.

Далее в этой главе мы обсудим, как проследить за ходом разработки системы в рамках цикла ее проектирования. План объединения позволяет оценить ход разработки во время отладки и объединения. Прежде чем перейти к аспектам управления циклом проектирования системы, рассмотрим средства, которые могут быть использованы для отладки и объединения, начиная с трассировщика.

7.15. Трассировщик

Трассировщик реализуется как программный модуль, который связан с другими модулями разрабатываемой системы. Он содержит структуру данных и некоторые процедуры, управляющие выполнением процедур системы во время объединения. Структура данных трассировщика состоит из параметра ОТДЕЛЬНЫЙ ШАГ (SINGLE STEP) и трех списков: СПИСОК УПРАВЛЕНИЯ ПРОЦЕДУРАМИ (PROCEDURE CONTROL LIST), СПИСОК ПАРАМЕТРОВ (PARAMETER LIST) и СПИСОК ВЫЗЫВАЕМЫХ ПРОЦЕДУР (PROCEDURE CALLED LIST). При вызове каждой из процедур, если установлен параметр ОТДЕЛЬНЫЙ ШАГ, выдается идентификационный номер процедуры и система ждет от оператора команды для перехода. СПИСОК УПРАВЛЕНИЯ ПРОЦЕДУРАМИ содержит управляющую информацию для каждой процедуры объединяемой системы. Эта информация указывает, должна ли выполняться процедура, или она должна быть пропущена, должно ли выполнение системы быть прервано при вызове процедуры. В списке также указывается, должен ли регистрироваться идентификационный номер процедуры в СПИСКЕ ВЫЗЫВАЕМЫХ ПРОЦЕДУР. Когда процедура пропускается, значения ее выходных параметром должны быть определены процедурой ТРАССИРОВЩИКА (COMMON STUB). СПИСОК ПАРАМЕТРОВ содержит проверочные значения выходных параметров, которые используются для проверки условий, если процедура пропускается.

Процедуры связываются с модулем ТРАССИРОВЩИКА через процедуру ТРАССИРОВЩИКА. Процедура ТРАССИРОВЩИКА вызывается каждой процедурой системы, за исключением ИСПОЛНИТЕЛЬНОЙ процедуры и процедуры ИНИЦИАЛИЗАЦИИ СИСТЕМЫ. Однако по желанию проектировщика

процедура ИНИЦИАЛИЗАЦИИ СИСТЕМЫ может вызвать процедуру ТРАССИРОВЩИКА *после того,* как она вызовет процедуру ИНИЦИАЛИЗАЦИИ ТРАССИРОВЩИКА (INITIALIZE COMMON STUB), которая будет описана ниже. Операции языка проектирования, которые должны быть включены в процедуры для этой цели, имеют следующий вид:

```
ВЫЗОВ: ТРАССИРОВЩИК (ИД НОМЕР; ПАРАМЕТР 1, ...
        ..., ПАРАМЕТР N, ПРОПУСК)
ЕСЛИ УСТАНОВЛЕН ПРОПУСК
  ТО ВОЗВРАТ
```

Входной параметр ИД НОМЕР является номером, однозначно определяющим каждую процедуру для ее идентификации в модуле ТРАССИРОВЩИКА. ПАРАМЕТР 1, ..., ПАРАМЕТР N являются выходными параметрами вызывающих процедур, значения которых устанавливаются процедурой ТРАССИРОВЩИКА из СПИСКА ПАРАМЕТРОВ. Параметр ПРОПУСК (SKIP)

```
ПРОЦЕДУРА: ПРОВЕРКА ПРОДОЛЖЕНИЯ ДВИЖЕНИЯ;
(; ПРОДОЛЖЕНИЕ)
НАЧАЛО ПРОЦЕДУРЫ
  ВЫЗОВ: ТРАССИРОВЩИК (4; ПРОДОЛЖЕНИЕ, ПРОПУСК)
  ЕСЛИ УСТАНОВЛЕН ПРОПУСК
    ТО ВОЗВРАТ
  ВЫЗОВ: ЗАПУСК ТАЙМЕРА (5 СЕКУНД;)
      •
      •
      •
КОНЕЦ ПРОЦЕДУРЫ
```

Рис. 7.9. Вызов процедуры трассировщика.

используется для пропуска процедуры. На рис. 7.9 показан пример использования этих операций в процедуре.

Модуль ТРАССИРОВЩИКА также содержит процедуру ИНИЦИАЛИЗАЦИИ ТРАССИРОВЩИКА, которая должна вызываться процедурой ИНИЦИАЛИЗАЦИИ СИСТЕМЫ. Как указывалось выше, процедура ИНИЦИАЛИЗАЦИИ СИСТЕМЫ может вызывать процедуру ТРАССИРОВЩИКА только после того, как она вызовет процедуру ИНИЦИАЛИЗАЦИИ ТРАССИРОВЩИКА. Процедура ИНИЦИАЛИЗАЦИИ ТРАССИРОВЩИКА используется для установки или сброса параметра ОТДЕЛЬНЫЙ ШАГ и для инициализации списков в структуре трассировщика. Кроме того, необходимы процедуры для обновления информации в параметре ОТДЕЛЬНЫЙ ШАГ и в СПИСКЕ УПРАВЛЕНИЯ ПРОЦЕДУРАМИ и СПИСКЕ ПАРАМЕТРОВ, а также процедура для взимодействия оператора с системой, позволяющая проверять информацию в СПИСКЕ ВЫЗЫ-

ВАЕМЫХ ПРОЦЕДУР. Эти процедуры называются процедурами ОБНОВЛЕНИЯ ОТДЕЛЬНОГО ШАГА (UPDATE SINGLE STEP), ОБНОВЛЕНИЯ СПИСКА УПРАВЛЕНИЯ ПРОЦЕДУРАМИ (UPDATE PROCEDURE CONTROL LIST), ОБНОВЛЕНИЯ СПИСКА ПАРАМЕТРОВ (UPDATE PARAMETER LIST) и ВЗАИМОДЕЙСТВИЯ С ОПЕРАТОРОМ (INTERACT WITH OPERATOR). Описание полного модуля ТРАССИРОВЩИКА на языке проектирования показано на рис. 7.10. Рассмотрим, как используются возможности модуля ТРАССИРОВЩИКА при проведении объединения системы.

Представим, что данные для параметра ОТДЕЛЬНЫЙ ШАГ и для СПИСКА УПРАВЛЕНИЯ ПРОЦЕДУРАМИ и СПИСКА ПАРАМЕТРОВ подготовлены и хранятся либо на магнитной ленте, либо в файле на диске и могут быть прочитаны процедурой ПОЛУЧЕНИЯ ДАННЫХ ОБНОВЛЕНИЯ (GET UPDATE DATA). Детали этого процесса могут меняться для каждой системы, поскольку они зависят от того, как хранятся данные. Набор данных, используемый для инициализации системы в начале объединения, должен вызывать пропуск каждой процедуры и занесение идентификационного номера каждой вызываемой процедуры в СПИСОК ПРОЦЕДУР. Для просмотра СПИСКА ВЫЗЫВАЕМЫХ ПРОЦЕДУР необходимо поместить в него код ОСТАНОВКИ ВЫПОЛНЕНИЯ для одной из процедур, вызываемых ИСПОЛНИТЕЛЬНОЙ процедурой. Параметр ОДИН ШАГ во время первой инициализации модуля ТРАССИРОВЩИКА должен быть сброшен.

Чтобы начать объединение, запускается ИСПОЛНИТЕЛЬНАЯ процедура. Система, включая модуль ТРАССИРОВЩИКА, инициализируется ИСПОЛНИТЕЛЬНОЙ процедурой при вызове процедуры ИНИЦИАЛИЗАЦИИ СИСТЕМЫ. Затем ИСПОЛНИТЕЛЬНАЯ процедура вызывает процедуры следующего уровня в обычной последовательности. Однако выполнения процедур не происходит, записываются только их номера в модуль ТРАССИРОВЩИКА. Когда вызывается процедура, для которой установлен код ОСТАНОВКИ ВЫПОЛНЕНИЯ процедурой ВЗАИМОДЕЙСТВИЯ С ОПЕРАТОРОМ (рис. 7.10), печатается сообщение

ПЕЧАТАТЬ СПИСОК ИД? (PRINT ID LIST?)

Если ответ оператора утвердительный, печатается или выдается на экране дисплея список идентификационных номеров вызываемых процедур. Затем печатается сообщение для выбора варианта работы

ВКЛЮЧИТЬ МОНИТОР? (ENTER MONITOR?)

в соответствии с которым оператор может запустить выполнение монитора. Описание монитора и принципов его работы

будет приведено далее. Наконец, оператору может быть выдано сообщение

ПРОДОЛЖИТЬ ВЫПОЛНЕНИЕ? (CONTINUE EXECUTION?)

в ответ на которое оператор может либо продолжить работу до следующей остановки, либо полностью остановить работу. Прогоняя систему несколько раз и делая остановки в выбранных точках, можно проверить с помощью листинга вызываемых процедур, является ли последовательность вызова процедур правильной. Если во время выполнения система не доходит до процедуры, у которой установлен код ОСТАНОВКИ ВЫПОЛНЕНИЯ, в таком случае может быть либо установлен параметр ОТДЕЛЬНЫЙ ШАГ, либо дополнительные коды ОСТАНОВКИ ВЫПОЛНЕНИЯ путем обновления данных СПИСКА УПРАВЛЕНИЯ ПРОЦЕДУРАМИ. Причину нарушения последовательности вызова процедур можно определить, либо прогоняя систему по отдельным шагам и прослеживая выполнение от

```
MODULE: COMMON STUB
*****************************************************************************
  DESIGNED BY: MDF                    16-OCT-80
*****************************************************************************
  PROCEDURES: INITIALIZE COMMON STUB
              COMMON STUB
              UPDATE PROCEDURE CONTROL LIST
              UPDATE PARAMETER LIST
              UPDATE SINGLE STEP
              INTERACT WITH OPERATOR
*****************************************************************************
  COMMON STUB
  DATA STRUCTURE: SINGLE STEP
                  PROCEDURE CONTROL LIST
                  PARAMETER LIST
                  PROCEDURES CALLED LIST
-----------------------------------------------------------------------------
PROCEDURE: INITIALIZE COMMON STUB (;)
*****************************************************************************
  DESIGNED BY: MDF                    16-OCT-80
  MODULE: COMMON STUB
*****************************************************************************
BEGIN PROCEDURE
  CALL: UPDATE SINGLE STEP (;)
  CALL: UPDATE PROCEDURE CONTROL LIST (;)
  CALL: UPDATE PARAMETER LIST (;)
  CLEAR PROCEDURES CALLED LIST
  RETURN
END PROCEDURE
-----------------------------------------------------------------------------
```

Рис. 7.10. Модуль трассировщика.

```
PROCEDURE: COMMON STUB (ID NUMBER;SKIP)
*************************************************************************
  DESIGNED BY: MDF                  16-OCT-80
  MODULE: COMMON STUB
*************************************************************************
BEGIN PROCEDURE
  IF CODE IN PROCEDURE CONTROL LIST FOR ID NUMBER INDICATES RECORD
                                                    PROCEDURE CALLED
    THEN  SAVE ID NUMBER IN NEXT LOCATION OF PROCEDURES CALLED LIST
  IF CODE IN PROCEDURE CONTROL LIST FOR ID NUMBER INDICATES STOP
    THEN  CALL: INTERACT WITH OPERATOR (;)
    ELSE  DO
            IF SINGLE STEP SET
              THEN  DO
                      CALL: DISPLAY ID (ID NUMBER;)
                      CALL: WAIT FOR CONTINUE COMMAND (;)
                    END
          END
  IF CODE IN PROCEDURE CONTROL LIST FOR ID NUMBER INDICATES SKIP
    THEN  SET SKIP
    ELSE  RESET SKIP
  RETURN
END PROCEDURE
-------------------------------------------------------------------------
PROCEDURE: UPDATE PROCEDURE CONTROL LIST (;)
*************************************************************************
  DESIGNED BY: MDF                  16-OCT-80
  MODULE: COMMON STUB .
  CALLED BY: INITIALIZE COMMON STUB
*************************************************************************
BEGIN PROCEDURE
  CALL: GET UPDATE DATA (;DATA)
  PUT DATA INTO PROCEDURE CONTROL LIST
  RETURN
END PROCEDURE
-------------------------------------------------------------------------
PROCEDURE: UPDATE PARAMETER LIST (;)
*************************************************************************
  DESIGNED BY: MDF                  16-OCT-80
  MODULE: COMMON STUB
  CALLED BY: INITIALIZE COMMON STUB
*************************************************************************
BEGIN PROCEDURE
  CALL: GET UPDATE DATA (;DATA)
  PUT DATA INTO PARAMETER LIST
  RETURN
END PROCEDURE
-------------------------------------------------------------------------
```

Рис. 7.10. (Продолжение.)

```
PROCEDURE: UPDATE SINGLE STEP (;)
*************************************************************************
  DESIGNED BY: MDF                    16-OCT-80
  MODULE: COMMON STUB
  CALLED BY: INITIALIZE COMMON STUB
*************************************************************************
BEGIN PROCEDURE
  CALL: GET UPDATE DATA (;DATA)
  IF DATA INDICATES THAT SINGLE STEP IS DESIRED
    THEN  SET SINGLE STEP
    ELSE  RESET SINGLE STEP
  RETURN
END PROCEDURE
-------------------------------------------------------------------------
PROCEDURE: INTERACT WITH OPERATOR (;)
*************************************************************************
  DESIGNED BY: MDF                    16-OCT-80
  MODULE: COMMON STUB
  CALLED BY: COMMON STUB
*************************************************************************
BEGIN PROCEDURE
  CALL  OUTPUT MESSAGE ('PRINT ID LIST? ';)
  CALL: GET ANSWER (;ANSWER)
  IF ANSWER IS 'YES'
    THEN  DO FOR EACH ENTRY IN PROCEDURES CALLED LIST
            PRINT ENTRY
          END
  CALL  OUTPUT MESSAGE ('ENTER MONITOR? ',)
  CALL  GET ANSWER (;ANSWER)
  IF ANSWER IS 'YES'
    THEN  CALL  MONITOR (,)
  CALL  OUTPUT MESSAGE ('CONTINUE EXECUTION? ',)
  CALL  GET ANSWER (;ANSWER)
  IF ANSWER IS 'YES'
    THEN  RETURN
    ELSE  CALL  STOP EXECUTION (,)
END PROCEDURE
-------------------------------------------------------------------------
END MODULE
```

Рис. 7.10. (Продолжение.)

процедуры к процедуре, либо путем исследования листингов вызываемых процедур, если используется второй способ. Причина нарушения может быть устранена путем коррекции операций языка проектирования и языка программирования. После того как завершен первый этап объединения программного обеспечения, можно приступать к остальным этапам, надлежащим образом измененяя управляющие коды в СПИСКЕ УПРАВЛЕНИЯ ПРОЦЕДУРАМИ. Если пропускается процедура, возвра-

щающая выходной параметр, значение этого параметра должно быть предусмотрено при обновлении СПИСКА ПАРАМЕТРОВ. На практике, чтобы проверить работу системы при различных условиях, система должна прогоняться несколько раз с различными значениями параметров.

Таким образом, трассировщик является средством, способствующим проведению объединения программного обеспечения на системной основе. При использовании его вместе с планом объединения наборы данных для СПИСКА УПРАВЛЕНИЯ ПРОЦЕДУРАМИ И СПИСКА ПАРАМЕТРОВ могут быть подготовлены заблаговременно, что позволит провести объединение быстро и эффективно. Результаты вычислений и листинги вызываемых процедур могут храниться вместе с планом объединения, представляя отчет, который позволяет легко проследить за ходом разработки в процессе объединения. Этот отчет является весьма ценным материалом как для группы разработчиков проекта, так и для руководства. Каждый член группы проектировщиков может проследить за состоянием разработки проекта и заранее знать, когда может потребоваться его помощь во время объединения.

После того как объединение системы завершено, модуль ТРАССИРОВЩИКА может быть отключен. Это выполняется путем установки параметра ОТДЕЛЬНЫЙ ШАГ и всех управляющих кодов таким образом, что все процедуры выполняются обычным способом. Однако, если позднее будет обнаружена ранее не замеченная ошибка или появится необходимость внести в системы непредусмотренное изменение, все возможности, предусмотренные трассировщиком, остаются доступными. Эти возможности могут быть реализованы путем установки параметра ОТДЕЛЬНЫЙ ШАГ или ввода новых данных, содержащих соответствующие управляющие коды, в СПИСОК УПРАВЛЕНИЯ ПРОЦЕДУРАМИ.

7.16. Реализация трассировщика

Мы описали модуль ТРАССИРОВЩИКА с точки зрения его функциональных возможностей и реализации на языке проектирования. Его реализация является сравнительно простой по большинству аспектов. Программа на языке проектирования конвертируется в программный язык, транслируется обычным образом и связывается с системой во время редактирования связей. Однако два аспекта реализации трассировщика нуждаются в дополнительном рассмотрении. Первый заключается в том, что нам необходим механизм управления вызовами процедур ТРАССИРОВЩИКА с переменным числом выходных параметров. Вторым аспектом является то, что нам может понадобиться более эффективный механизм для пропуска процедур,

чем тот, который был описан в предыдущем разделе и который требуется для использования параметра ПРОПУСК и условной конструкции в вызывающей процедуре.

Одним из простых способов обеспечения переменного числа выходных параметров является использование различных процедур для каждого случая. Так, мы могли бы реализовать набор процедур, которые вызываются следующим образом:

ВЫЗОВ: ТРАССИРОВЩИК (ИД НОМЕР; ПРОПУСК)
ВЫЗОВ: ТРАССИРОВЩИК 1 (ИД НОМЕР; ПАРАМЕТР 1, ПРОПУСК)
ВЫЗОВ: ТРАССИРОВЩИК 2 (ИД НОМЕР; ПАРАМЕТР 1, ПАРАМЕТР 2 ПРОПУСК)

и т. д. Каждая из процедур могла бы быть представлена в формате, показанном на рис. 7.11. Хотя может показаться, что при

```
PROCEDURE: COMMON STUB N (ID NUMBER;PARAMETER 1, ..., PARAMETER N,SKIP)
*************************************************************************
  DESIGNED BY: MDF                  16-OCT-80
  MODULE: COMMON STUB
*************************************************************************
BEGIN PROCEDURE
  IF CODE IN PROCEDURE CONTROL LIST FOR ID NUMBER INDICATES RECORD
                                                    PROCEDURE CALLED
    THEN  SAVE ID NUMBER IN NEXT LOCATION OF PROCEDURES CALLED LIST
  IF CODE IN PROCEDURE CONTROL LIST FOR ID NUMBER INDICATES STOP
    THEN  CALL: INTERACT WITH OPERATOR (;)
    ELSE  DO
            IF SINGLE STEP SET
              THEN  DO
                      CALL: DISPLAY ID (ID NUMBER;)
                      CALL: WAIT FOR CONTINUE COMMAND (;)
                    END
          END
  IF CODE IN PROCEDURE CONTROL LIST FOR ID NUMBER INDICATES SKIP
    THEN  DO
            GET N PARAMETERS FOR ID NUMBER FROM PARAMETER LIST
            SET SKIP
          END
    ELSE  RESET SKIP
  RETURN
END PROCEDURE
```

Рис. 7.11. Процедура трассировщика, модифицированная для выходных параметров.

этом необходимо большое число процедур, однако это не так. Например, как мы уже видели, если используется PL/M для микрокомпьютера Intel 8085, имеются только три возможности: PL/M-процедура может либо не возвращать ни одного па-

раметра, либо возвратить единственный параметр типа BYTE или единственный параметр типа ADDRESS. Таким образом, для микрокомпьютера Intel 8085 необходимы только три процедуры ТРАССИРОВЩИКА.

Из-за ограничения, накладываемого на количество выходных параметров PL/M, становится неудобно возвращать параметр ПРОПУСК каждый раз при вызове процедуры ТРАССИРОВЩИКА. Поэтому нам необходимо иметь более эффективный механизм пропуска процедур, если используется PL/M. Такой механизм обеспечивается с помощью стека. В гл. 6 мы видели, как используется в микрокомпьютере Intel 8085 стек для хранения указателей, необходимых для операций ВОЗВРАТА (RETURN). При выполнении процедуры элемент в вершине стека содержит указатель на процедуру, вызвавшую процедуру ТРАССИРОВЩИКА. Назовем эту процедуру *условной* процедурой. Второй элемент в стеке является указателем процедуры, которая вызвала условную процедуру. Поэтому выполнение условной процедуры пропускается, если перед выполнением операции ВОЗВРАТА в процедуре ТРАССИРОВЩИКА выполняется команда, эквивалентная команде POP. После этого продолжается выполнение непосредственно той операции, которая вызвала условную процедуру. Если условная процедура должна возвращать выходной параметр типа BYTE, значение этого параметра из СПИСКА ПАРАМЕТРОВ должно быть помещено в АККУМУЛЯТОР прежде, чем закончится выполнение процедуры. В другом случае, если условная процедура возвращает выходной параметр типа ADDRESS, значение параметра из СПИСКА ПАРАМЕТРОВ должно быть помещено в пару регистров HL. Рис. 7.12 иллюстрирует, как это может быть выполнено в программе на языке PL/M для выходных параметров как типа BYTE, так и типа ADDRESS с использованием операции RETURN.

В PL/M нет операции POP. Однако предусмотрено средство, позволяющее непосредственно манипулировать указателем стека. Так, PL/M-операция

```
STACKPTR = STACKPTR + 2;
```

устанавливает указатель стека таким образом, что второй элемент стека становится верхним и таким образом достигается результат команды POP. Для этой цели мы используем операцию языка проектирования УСТАНОВИТЬ СТЕК НА ПРОПУСК ВЫЗЫВАЮЩЕЙ ПРОЦЕДУРЫ. Вспомним, что память микрокомпьютера Intel 8085, которая используется для стека, организована так, что первый элемент, помещенный в стек, хранится в двух старших адресных байтах области памяти, отведенной для стека. Поэтому указатель стека должен быть уменьшен

на два при каждой операции типа PUSH (или эквивалентной ей) или увеличен на два при каждой операции типа POP (или эквивалентной). Как указывалось ранее, манипуляции с указателем стека в PL/M производятся автоматически. За исключением инициализации указателя стека и реализации функции пропуска в процедуре ТРАССИРОВЩИКА, мы не рекомендуем использовать непосредственную манипуляцию указателем стека в прикладных программах. Это может привести к неправильному срабатыванию, которое будет трудно обнаружить.

```
/*PROCEDURE: COMMON STUB RETURNING ADDRESS (ID NUMBER;PARAMETER)          */
                                        COMMON$STUB$RETURNING$ADDRESS:
                                        PROCEDURE (ID$NUMBER) ADDRESS PUBLIC;
/*************************************************************************/
/*  DESIGNED BY: MDF                16-OCT-80                            */
/*  MODULE: COMMON STUB                                                  */
/*                                                                       */
/*  INPUT PARAMETER:                                                     */
/*                  NAME        TYPE        SIZE                         */
/*                  ----        ----        ----                         */
/*                ID NUMBER     BYTE         --                          */
                                                  DECLARE ID$NUMBER BYTE;
/*                                             *     ---                 */
/*  OUTPUT PARAMETER:                                                    */
/*                  NAME        TYPE        SIZE                         */
/*                  ----        ----        ----                         */
/*                PARAMETER     ADDRESS      --                          */
                                                  DECLARE PARAMETER ADDRESS;
/*************************************************************************/
/*BEGIN PROCEDURE                                                        */
/*  IF CODE IN PROCEDURE CONTROL LIST FOR ID NUMBER INDICATES RECORD     */
/*                                                  PROCEDURE CALLED     */
/*    THEN  SAVE ID NUMBER IN NEXT LOCATION OF PROCEDURES CALLED LIST    */
/*  IF CODE IN PROCEDURE CONTROL LIST,FOR ID NUMBER INDICATES STOP       */
/*    THEN  CALL: INTERACT WITH OPERATOR (;)                             */
/*    ELSE  DO                                                           */
/*            IF SINGLE STEP SET                                         */
/*              THEN  DO                                                 */
/*                      CALL: DISPLAY ID (ID NUMBER;)                    */
/*                      CALL: WAIT FOR CONTINUE COMMAND (;)              */
/*                    END                                                */
/*          END                                                          */
/*  IF CODE IN PROCEDURE CONTROL LIST FOR ID NUMBER INDICATES SKIP       */
/*    THEN  DO                                                           */
                                                  THEN  DO;
/*            GET PARAMETER FOR ID NUMBER FROM PARAMETER LIST            */
                                                    PARAMETER = PARAMETER$LIST (ID$NUMBER);
/*            ADJUST STACK TO SKIP CALLING PROCEDURE                     */
                                                    STACKPTR = STACKPTR + 2;
/*            RETURN                                                     */
                                                    RETURN PARAMETER;
/*          END                                                          */
                                                  END;
/*. RETURN                                                               */
/*END PROCEDURE                                                          */
                                        END COMMON$STUB$RETURNING$ADDRESS;
```

Рис. 7.12а. Частично конвертированная в PL/M процедура трассировщика для выходного параметра типа байт.

```
/*PROCEDURE  COMMON STUB RETURNING BYTE (ID NUMBER,PARAMETER) *          */
                                        COMMON$STUB$RETURNING$BYTE
                                        PROCEDURE (ID$NUMBER) BYTE PUBLIC
/*************************************************************************/
/*  DESIGNED BY  MDF                16-OCT-80                            */
/*  MODULE  COMMON STUB                                                  */
/*                                                                       */
/*  INPUT PARAMETER:                                                     */
/*                  NAME        TYPE          SIZE                       */
/*                  ----        ----          ----                       */
/*                ID NUMBER     BYTE          --                         */
                                                  DECLARE ID$NUMBER BYTE;
/*                                                                       */
/*  OUTPUT PARAMETER:                                                    */
/*                  NAME        TYPE          SIZE                       */
/*                  ----        ----          ----                       */
/*                PARAMETER     BYTE          --                         */
                                                  DECLARE PARAMETER BYTE;
/*************************************************************************/
/*BEGIN PROCEDURE                                                        */
/*  IF CODE IN PROCEDURE CONTROL LIST FOR ID NUMBER INDICATES RECORD     */
/*                                                  PROCEDURE CALLED     */
/*    THEN  SAVE ID NUMBER IN NEXT LOCATION OF PROCEDURES CALLED LIST    */
/*  IF CODE IN PROCEDURE CONTROL LIST FOR ID NUMBER INDICATES STOP       */
/*    THEN  CALL: INTERACT WITH OPERATOR (;)                             */
/*    ELSE  DO                                                           */
/*            IF SINGLE STEP SET                                         */
/*              THEN  DO                                                 */
/*                      CALL: DISPLAY ID (ID NUMBER;)                    */
/*                      CALL: WAIT FOR CONTINUE COMMAND (;)              */
/*                    END                                                */
/*          END                                                          */
/*  IF CODE IN PROCEDURE CONTROL LIST FOR ID NUMBER INDICATES SKIP       */
/*    THEN  DO                                                           */
                                                  THEN  DO;
/*            GET PARAMETER FOR ID NUMBER FROM PARAMETER LIST            */
                                                    PARAMETER = PARAMETER$LIST (ID$NUMBER);
/*            ADJUST STACK TO SKIP CALLING PROCEDURE                     */
                                                    STACKPTR = STACKPTR + 2;
/*            RETURN                                                     */
                                                    RETURN PARAMETER;
/*          END                                                          */
                                                  END;
/*  RETURN                                                               */
/*END PROCEDURE                                                          */
                                              END COMMON$STUB$RETURNING$BYTE;
```

Рис. 7.12б. Частично конвертированная в PL/M процедура трассировщика для выходного параметра адресного типа.

На этом описание использования трассировщика во время объединения завершается. Далее мы рассмотрим другие средства, которые могут быть использованы во время объединения, а также средства автоматического конструирования программного обеспечения и автоматизации управляющих функций, необходимых в течение цикла проектирования системы.

7.17. Другие средства объединения

Во время объединения часто бывает желательно проверить вычисленное значение параметра. Так как эту возможность может обеспечить монитор, мы предусмотрели способ вызова монитора в процедуре ТРАССИРОВЩИКА. Монитор представляет программный модуль, который включен в микрокомпьютерную систему разработки и который может быть включен в разрабатываемую систему. В последнем случае он может включаться в виде удаляемого модуля, который доступен только во время объединения и для последующих аварийных ситуаций. В дополнение к возможности исследовать информацию в памяти монитор предусматривает способ связи оператора с микрокомпьютером на более низком уровне по сравнению со способом связи, предусматриваемым операционной системой микрокомпьютерной системы разработки. Оператору доступны следующие команды монитора:

- Выдать содержимое одной или нескольких ячеек памяти или регистров или прочитать входной порт. Выдача может производиться в шестнадцатеричной, десятичной или любой другой системе счисления по выбору оператора.
- Заменить текущее значение содержимого ячейки памяти другим определенным значением или переслать другое значение в выходной порт.
- Начать выполнение программы, находящейся в памяти, с команды, хранящейся в определенной ячейке памяти.

Таким образом, монитор обеспечивает связь с микрокомпьютерной системой на уровне машинного языка. Читатель, однако, может заметить, что начинать выполнение с любой команды, за исключением первой команды ИСПОЛНИТЕЛЬНОЙ процедуры или команды трассировщика, где выполнение может быть приостановлено, довольно рискованно, так как значения параметров могут быть неопределенными, а стек может содержать неправильные или посторонние указатели.

Как было описано, монитор обеспечивает связь между оператором и системой на низком уровне. Чтобы использовать монитор для проверки значения параметра, например ПЕРЕКЛЮЧАТЕЛЬ, адрес ячейки памяти, в которой хранится этот параметр, должен быть известен оператору. Адрес параметра может быть получен из списка символьных имен или таблицы, выдаваемой загрузчиком. Однако в некоторых системах таблица символьных имен может храниться и размещаться в памяти после загрузки программы для выполнения. В этом случае средство, подобное монитору, может обращаться к таблице символьных имен, когда система останавливается во время объединения. При этом оператор может прямо затребовать вы-

дачу значения параметра ПЕРЕКЛЮЧАТЕЛЬ, не зная адреса ячейки памяти. Средством, которое обладает такой подобной монитору возможностью, является *внутрисхемный эмулятор*. Во внутрисхемном эмуляторе предусмотрены также другие функции, которые являются полезными во время объединения системы и которые будут описаны в гл. 10. Описание другого средства объединения, которое называется *микрокомпьютерным анализатором,* также откладывается до гл. 9.

7.18. Руководство проектом

Мы уже видели, насколько важным является тщательное планирование каждого этапа объединения перед началом отладки системы. Не менее важной является оценка сроков завершения каждого шага цикла проектирования системы. Такие оценки составляют план, который должен включать даты окончания каждого из следующих основных разделов:

- Окончание составления документации для требований пользователей и функциональных спецификаций для системы и каждой из подсистем.
- Согласование требований пользователей и функциональной спецификации с заказчиками.
- Завершение предварительного проектирования системы.
- Завершение построения модульной структуры системы.
- Завершение проектирования программного обеспечения для каждой процедуры и модуля.
- Конвертирование проектного описания в язык программирования и исключение синтаксических ошибок.
- Окончание каждого этапа объединения.
- Проверка функционирования системы в соответствии с требованиями пользователей и функциональной спецификации. Этот последний шаг может включать формальную приемку-тестирование, если этого требует заказчик.

В данном разделе мы описали, как подготовить календарный план проекта. Мы также обсудили, какие шаги цикла проектирования системы необходимо отмечать, чтобы обеспечить окончание проекта в соответствии с календарным планом.

Заблаговременная оценка сроков разработки проекта довольно трудна, особенно для тщательно разрабатываемой программной системы. Однако мы утверждаем, что при использовании системных методов и средств возможно предусмотреть реальные сроки при условии, что система имеет модульную структуру и разделена на процедуры. Поскольку сложность процедур ограничена, можно оценить размер и область применения каждой из них (возможно, даже до того, как завершено ее описание на языке проектирования), а также усилия, необ-

ходимые для того, чтобы спроектировать, построить, протестировать каждую процедуру и объединить ее в систему. Используя эти оценки, можно надлежащим образом распределить подсистемы и модули среди проектировщиков и разработать реалистичный календарный план для полного проекта.

Руководствуясь этим планом, несложно проследить за каждым шагом цикла проектирования системы с тем, чтобы завершить ее своевременно. Можно также подготовить схему, содержащую наименования подсистем, модулей и процедур, а также имена ответственных за выполнение каждой процедуры и даты завершения каждого шага. Подобную же схему необходимо предусмотреть для каждого этапа объединения. После того как шаг завершен, это может быть отмечено на схеме, и таким образом можно прослеживать за ходом выполнения проекта. Могут быть также отмечены любые проблемы, встретившиеся в процессе разработки, и оценено их влияние на остальную часть проекта. Если необходимо, при этом могут быть заново оценены стоимость и сроки разработки.

Эта методика руководства проектом, содержащим программное обеспечение, является особенно полезной, если проектирование процедур, их конвертирование в язык программирования и их объединение в систему поручается разным людям. Без системного подхода в руководстве можно упустить из виду какой-либо шаг проектирования, что может неблагоприятно повлиять на сроки разработки проекта в целом.

7.19. Четвертый уровень документации

Четвертый уровень документации состоит прежде всего из плана объединения и схемы руководства календарными сроками проекта. Поскольку объединение будет продолжено, результаты тестирования, полученные во время шагов объединения, могут быть сохранены и составят часть четвертого уровня документации. Дополнительно к этому все изменения сроков разработки и причины этих изменений должны быть также включены в четвертый уровень документации. Таким образом, четвертый уровень документации состоит преимущественно из информации для руководства проектом в отличие от других уровней, которые состоят преимущественно из информации для проектирования и реализации.

На этом мы закончили обсуждение объединения программного обеспечения. Вопросы, касающиеся объединения аппаратных средств и системы в целом, рассматриваются в гл. 9 и 10 соответственно. В следующей главе мы рассмотрим, как проектируются аппаратные средства для системы, содержащей микрокомпьютер.

7.20. Упражнения

7.1. На рис. 7.1 перечислены средства, которые могут быть использованы в течение цикла проектирования системы. Однако не все они могут быть доступны сразу. Так как ручной способ разработки системы и программного обеспечения неэффективен, а все описанные средства сразу недоступны, перечислите минимальный набор средств, который являлся бы приемлемым для вас при разработке микрокомпьютерной системы. Дайте обоснование вашего выбора.

7.2. Начиная с минимального набора средств, выбранного в упр. 7.1, перечислите оставшиеся средства в порядке их важности. Для каждого средства дайте обоснование, почему вы предпочитаете его остальным средствам.

7.3. Предложите план объединения программного обеспечения для телевизионного приемника с встроенным микрокомпьютером (см. упражнения в гл. 2—5).

7.4. Предложите план объединения программного обеспечения для устройства управления уличным светофором с встроенным микрокомпьютером (см. упражнения в гл. 2—5).

7.5. Предложите план объединения программного обеспечения для электронного спортивного табло (см. упражнения в гл. 2—5).

7.6. Предложите план объединения программного обеспечения для терминала розничной торговли с встроенным микрокомпьютером (см. упражнения в гл. 2—5).

7.7. Модифицируйте проектное описание телевизионного приемника с встроенным микрокомпьютером так, чтобы можно было использовать во время объединения модуль трассировщика.

7.8. Модифицируйте проектное описание устройства управления уличным светофором с встроенным микрокомпьютером так, чтобы можно было использовать во время объединения модуль трассировщика.

7.9. Модифицируйте проектное описание электронного спортивного табло так, чтобы можно было использовать во время объединения модуль трассировщика.

7.10. Модифицируйте проектное описание терминала розничной торговли так, чтобы можно было использовать во время объединения модуль трассировщика.

7.11. Если ваша система использует PL/M микрокомпьютера Intel 8085, завершите конвертирование процедур трассировщика из языка проектирования в PL/M.

7.12. Если ваша система не использует PL/M микрокомпьютера Intel 8085, то:

а) модифицируйте, если необходимо, проектное описание процедур для работы с используемым вами языком и микрокомпьютером;

б) конвертируйте версию процедур на языке проектирования в используемый вами язык.

7.13. Трудно оценить время и усилия, затраченные для полного завершения каждого шага плана объединения, даже работая в условиях промышленного производства. Данное упражнение иллюстрирует, какие шаги должны быть предприняты для этого, хотя конечный результат и невозможно будет проверить в ваших условиях. Для одного из планов объединения программного обеспечения, предложенных в упр. 7.3—7.6:

а) определите размер и сферу применения каждой процедуры, необходимой для реализации каждого шага плана объединения. Используйте любые разумные оценки, например приняв трудоемкость небольшой и наиболее простой процедуры за 1, оценивайте трудоемкость процедуры, которая в n раз сложнее или требует в n раз больше затрат времени, как n;

б) сложите все эти оценки для всех процедур и умножьте на вашу оценку времени, необходимого для полной разработки процедуры, трудоемкость которой принята за 1. Эта оценка должна включать время подготовки проектного описания, время просмотра, время для конвертирования проектного описания в язык программирования и время для отладки и объединения процедур в систему.

Последние оценки должны быть распределены пропорционально, так как процедуры редко отлаживаются и объединяются отдельно. Не забудьте также включить оценку затраченного времени на коррекцию и перепроектирование.

Глава 8

Проектирование аппаратных средств

До того как появилась технология интегральных схем, электронные системы конструировались из таких дискретных компонентов, как гранзисторы, конденсаторы и резисторы. Аппаратные средства проектировались на уровне электронных схем, и сложность системы определялась прежде всего стоимостью проектирования и компоновки схем.

В начале 60-х годов появились первые элементы интегральных схем, и проектирование аппаратных средств приобрело функциональный характер. Стали доступными такие элементы интегральных схем, как усилители, вентильные и триггерные схемы. Проектировщики аппаратных средств смогли сосредоточить внимание на взаимосвязи этих элементов в модули В то время стоимость проектирования аппаратных средств определялась прежде всего стоимостью проектирования модулей. Многие системы, не имевшие ранее практического применения, стали экономически эффективными.

В течение 60-х и в начале 70-х годов стало возможным производить интегральные схемы, содержащие большое число компонентов. В результате законченный *модуль*, содержащий множество вентилей, триггеров и буферов, мог быть теперь сконструирован в виде интегральной схемы на *отдельном* кристалле или микросхемы. Стало возможным строить блоки из таких микросхемных *модулей,* как сумматоры, мультиплексоры и регистры сдвига. Стоимость проектирования аппаратных средств стала определяться преимущественно стомостью проектрования систем и подсистем. Снова стало экономически выгодным строить системы, которые ранее не имели практического применения.

Усовершенствование технологии интегральных схем в настоящее время вызвало быстрое снижение стоимости целого класса ЭВМ, называемых *мини-компьютерами.* В некоторых приложениях стало возможным встраивать мини компьютер в электронные системы и началась новая эра проектирования электронных систем. Стоимость электронной системы больше не определяется только стоимостью аппаратуры. Для надлежащего функционирования системы стала необходимой разработка программного обеспечения мини-компьютера. Во многих случаях

проектирование аппаратуры стало заключаться в обеспечении интерфейса между мини-компьютером и его внешними устройствами. Таким образом, стоимость аппаратуры стала уменьшаться в общем объеме затрат на проектирование системы которые теперь стали включать стоимость разработки программ. И снова экономически выгодным стал новый класс сложных приложений для электронных систем.

В течение 70-х годов стало возможным производить на основе небольшого числа интегральных микросхем на кристаллах целые вычислительные системы или микрокомпьютеры. Наличие таких микрокомпьютерных *компонентов* еще более уменьшило долю затрат на аппаратную часть систем по сравнению с их общей стоимостью. Как и для мини-компьютера, большинство аппаратных проектных решений связано с обеспечением интерфейса между микрокомпьютером и его внешней средой. Однако в отличие от мини-компьютера, который обычно покупается в виде полностью собранной подсистемы, микрокомпьютер может быть приобретен либо в виде набора микросхемных модулей или печатных плат, либо в виде набора несобранных микросхем на отдельных кристаллах. В этой главе будет рассмотрено проектирование аппаратных средств микрокомпьютеров с двух точек зрения: как проектируются аппаратные средства микрокомпьютерных систем с использованием интегральных схем на кристаллах и как проектируется интерфейс между микрокомпьютером и его внешней средой. Как и в случае программного обеспечения, мы не пытаемся предложить всеобъемлющее толкование вопросов проектирования аппаратуры. Материал данной главы является лишь введением в проектирование аппаратных средств микрокомпьютерных систем. Для более тщательного ознакомления читатель отсылается к списку литературы. Мы начнем обсуждение с вопроса, рассмотренного в гл. 3, а именно с построения модульной структуры аппаратных средств.

8.1. Модульная структура аппаратных средств

Модульная структура аппаратных средств микрокомпьютерной системы, представленная в общем виде в гл. 3, воспроизведена на рис. 8.1. На рисунке показаны связи между модулем микрокомпьютера и его внешней средой. На нем не показаны те аппаратные модули, которые связаны с модулями преобразования сигналов и являются частью расширения системы. Связь этих модулей с микрокомпьютером лучше показать на примере.

В упражнениях в конце гл. 3 перед читателем была поставлена задача разработать модульную структуру аппаратного

обеспечения телевизионного приемника с встроенным микрокомпьютером. На рис. 8.2 показана часть модульной структуры телевизионного приемника с целью иллюстрации взаимосвязи аппаратных модулей с модулями преобразования сигналов. Принципы построения модульной структуры аппаратных средств подобны принципам, применяемым для модульной структуры программного обеспечения системы. Это означает,

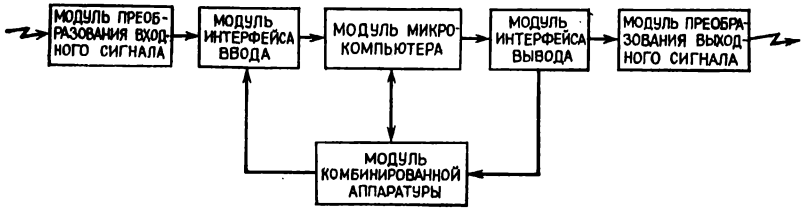


Рис. 8.1. Общее представление модульной структуры аппаратных средств микрокомпьютерной системы.

что все функции, предусмотренные в конкретном модуле, должны быть взаимосвязаны.

В примере с телевизионным приемником контраст и яркость изображения, движение устройства вращения антенны, приведение в действие механизма переключения каналов в модуле СЕЛЕКТОРА КАНАЛОВ управляются микрокомпьютером. Чтобы управлять этими функциями, микрокомпьютер имеет доступ к четырем входам: интенсивность освещения комнаты (от фотоэлемента), команда вращения антенны (от ручки вращения антенны), номер канала (от селектора каналов) и установка переключателя настройки антенны. Микрокомпьютер либо постоянно проверяет эти входы, либо откликается на прерывания, возникающие в момент изменения состояния входов. В обоих случаях в ответ на возникшие изменения состояния входов должна выполняться соответствующая программная процедура. Если во время работы меняется сигнал на выходе фотоэлемента, указывая тем самым, что изменилось освещение комнаты, сигналы управления контрастностью и яркостью, посылаемые в модуль ВИДЕО, должны соответствующим образом компенсировать это изменение. Если переключатель селектора каналов перемещается в новое положение, должна быть выработана и послана в модуль СЕЛЕКТОРА КАНАЛОВ соответствующая команда выбора канала. Как только будет выбран новый телевизионный канал, устройство вращения антенны автоматически настраивается на такое положение, которое лучше всего подходит для приема выбранного канала. Рассмотрим, какая информация необходима для первоначальной настройки устройства вращения антенны.

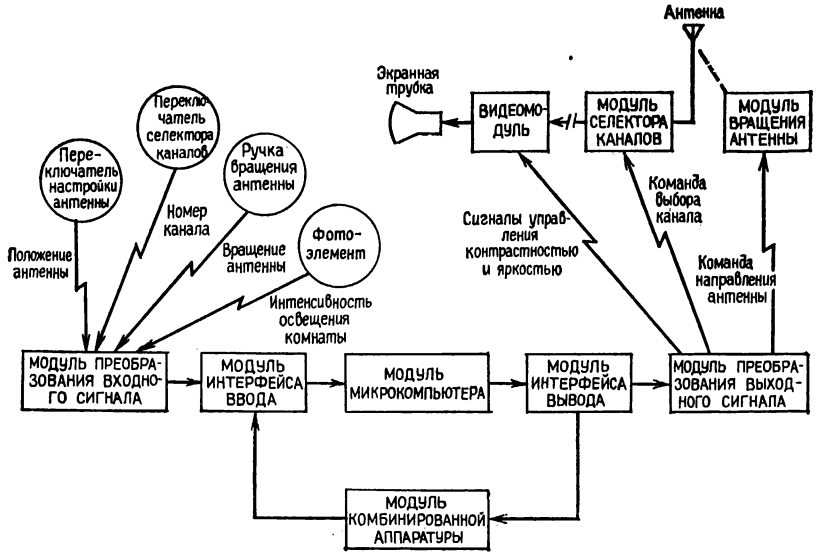


Рис. 8.2. Часть маодульной структуры аппаратных средств телевизионного приемника с встроенным микрокомпьютером.

Когда телевизионный приемник настраивается впервые или когда начинает работать новая телевизионная станция, положение антенны выбирается следующим образом. Включается переключатель настройки антенны и ручкой вращения антенны регулируется ее положение. После того как найдено оптимальное положение антенны для выбранного канала, переключатель настройки антенны выключается. Микрокомпьютер хранит информацию управления антенной для выбранного канала в структуре данных УПРАВЛЕНИЕ АНТЕННОЙ. Этот процесс повторяется для каждого первоначально настраиваемого канала. Информация в структуре данных УПРАВЛЕНИЕ АНТЕННОЙ используется для настройки антенны каждый раз, когда телевизионный приемник переключается на новый канал.

В модульной структуре программного обеспечения *связь* между модулями определяется механизмом *вызова* процедур и передачи входных и выходных параметров, а также с помощью дерева вызова процедур. В модульной структуре аппаратных средств связь между модулями может быть определена с помощью диаграмм, как, например, на рис. 8.2. Однако для сложных систем диаграмма модульной структуры аппаратных средств может оказаться слишком громоздкой. Иногда желательно расчленить аппаратную часть проекта на подсистемы таким же способом, как и программную часть. Таким образом,

как уже указывалось, имеется много аналогий между проектированием программного и аппаратного обеспечения.

Мы ограничим пока описание аппаратных средств описанием модулей, включающих микрокомпьютер и его интерфейсы. В частности, мы рассмотрим некоторые вопросы программирования самого нижнего уровня, которые тесно связаны с выбором компромисса между аппаратными и программными средствами.

8.2. Модуль микрокомпьютера

Когда появились первые микрокомпьютеры, чтобы построить микрокомпьютерную систему, необходимо было приобрести набор микросхем и скомпоновать их. К сожалению, в то время было очень мало средств для тестирования собранного микрокомпьютера, проектирования прикладного программного обеспечения или проверки достоверности функционирования собранной системы. Появление микрокомпьютерных систем разработки помогло преодолеть трудности, связанные с разработкой как программных, так и аппаратных средств. В частности, появление полностью тестируемых модулей, а также микрокомпьютерных и логических анализаторов помогло преодолеть трудности при разработке аппаратных средств. Использование микрокомпьютерных систем разработки было рассмотрено в гл. 7. Использование микрокомпьютерных и логических анализаторов будет описано в гл. 9. В данной главе мы ограничимся рассмотрением интегральных микросхем и модулей, из которых может быть построена аппаратная часть микрокомпьютерной системы.

Большинство изготовителей микрокомпьютерных компонентов выпускают широкий ассортимент продукции — от интегральных схем на кристаллах до полностью смонтированных и протестированных модулей в виде печатных плат или полностью собранных систем. Для разработки макетной оценочной модели системы или для мелкосерийного производства экономически более выгодны модули в виде печатных плат или собранные системы. При разработке продукции, которая должна выпускаться крупными партиями, часто более выгодно приобретать микросхемы, а затем монтировать и проверять модули в процессе производства. Этот подход является особенно эффективным, если бóльшая часть микрокомпьютерных систем, например интерфейс ввода-вывода, делается на заказ и должна в любом случае монтироваться и проверяться при изготовлении.

На рис. 8.3 показана функциональная схема одноплатного микрокомпьютерного модуля Intel iSBC 80/05. При этом возможно использование других печатных плат, содержащих

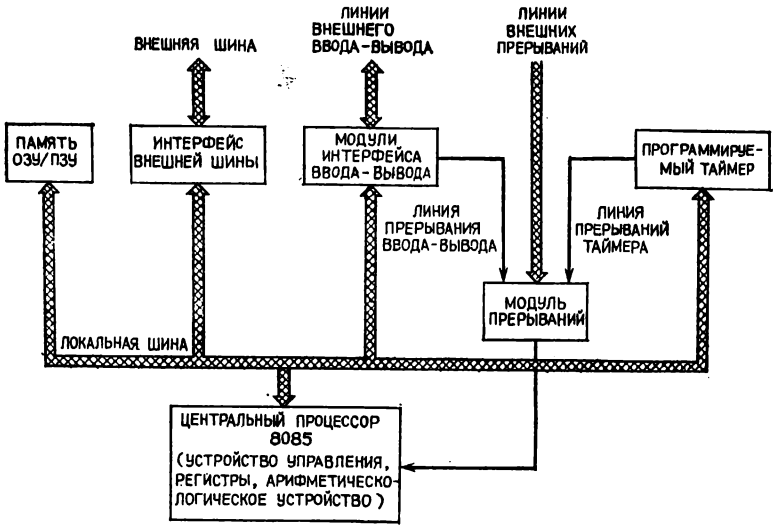


Рис. 8.3. Схема одноплатного микрокомпьютера Intel iSBC 80/05.

интегральные схемы на кристаллах, для увеличения числа доступных ячеек памяти или портов ввода-вывода, для расширения возможностей системы прерываний или для обеспечения возможности блочной передачи данных по каналу ПДП. Таким образом, имея несколько подключенных друг к другу печатных плат, блок питания и примитивное устройство управления в виде кнопки восстановления, можно скомпоновать работающую микрокомпьютерную систему. Спроектировать аппаратную часть системы, содержащей микрокомпьютер, сравнительно несложно, особенно если используются готовые модули. Однако для того, чтобы изучить возможность и взаимосвязь функций системы, нам необходимо рассмотреть, как конструируются аппаратные модули микрокомпьютера непосредственно из интегральных схем на кристаллах.

8.3. Интегральные схемы микрокомпьютера

Интегральные схемы на кристаллах, используемые для построения микрокомпьютера, делятся на три категории: центральный процессор, память и ввод-вывод. Однако имеется много микросхем, объединяющих эти функции. Например, такой однокристальный микрокомпьютер, как Intel 8048, выполняет все три функции, так что для некоторых приложений нет необходимости в каких-либо других микросхемах для аппаратной части микрокомпьютера. Однако для многих других приложе-

ний, чтобы аппаратура могла функционировать как микрокомпьютерная система, необходимо по нескольку микросхем каждого вида. Рассмотрим каждый из трех типов микросхем, а также способы их подключения.

8.4. Центральный процессор

Центральный процессор микрокомпьютера включает устройство управления, арифметическо-логическое устройство, несколько регистров и несложный интерфейс, который обеспечивает подключение к центральному процессору микросхем памяти и ввода-вывода. Функциональные характеристики микрокомпьютера Intel 8085 были описаны в гл. 6, а на рис. 8.3 показаны его функциональные взаимосвязи с другими микросхемами, образующими одноплатный микрокомпьютер. В данном разделе будут рассмотрены специальные сигналы, которые должны передаваться от одной интегральной схемы к другой, управляя работой микрокомпьютера. Эти сигналы, обозначенные сокращенными наименованиями, схематично показаны на рис. 8.4. Они объединены в соответствии с их функциональным назначением в следующие группы: «Информация шины», «Управление шиной», «Управление прерываниями», «Последовательный ввод-вывод», «Инициализация устройств» и «Управление синхронизацией». Рис. 8.5 содержит краткое описание каждого из сигналов микрокомпьютера Intel 8085.

На рис. 8.6 показано, как подключаются микросхемы памяти и периферийных устройств к центральному процессору Intel 8085. Рисунок иллюстрирует только те сигналы, которые необходимы для построения *несложного* микрокомпьютера.

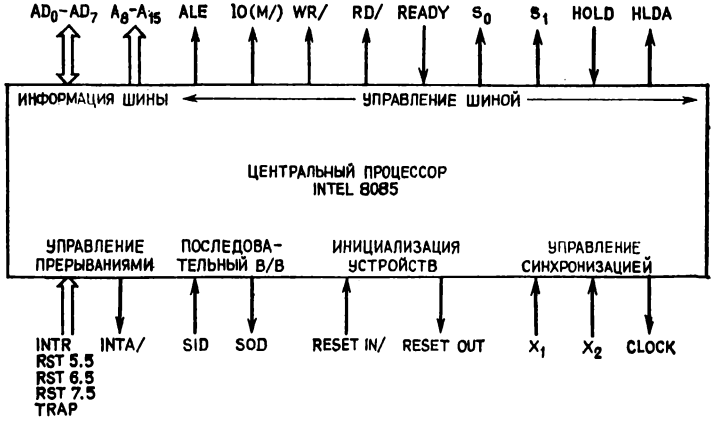


Рис. 8.4. Сигналы интегральной микросхемы Intel 8085.

Обозначение фирмы Intel	Описание сигнала	
AD_0	Адрес-данные 0	Младший разряд адресной шины или шины данных
•	•	•
•	•	•
•	•	•
AD_7	Адрес-данные 7	Разряд 7 адресной шины или старший разряд шины данных
A_8	Адрес 8	
•	•	•
•	•	•
•	•	•
A_{15}	Адрес 15	Старший разряд адресной шины
ALE	Шина адреса-данных содержит текущий адрес, который необходимо фиксировать в памяти или в периферийном устройстве (разрешение фиксации адреса)	
IO (M/)	Адресная шина содержит адрес порта ввода-вывода, если сигнал имеет высокий уровень, или адрес памяти, если сигнал имеет низкий уровень	
WR/	Записать данные в память или переслать данные в порт ввода-вывода	
RD/	Читать данные из памяти или из порта ввода-вывода	
READY	Память или порт ввода-вывода готовы переслать данные центральному процессору; память или порт ввода-вывода готовы принять данные от центрального процессора	
S_0, S_1	Разряды состояния центрального процессора	
HOLD	Внешнее устройство запрашивает шину	
HLDA	Шина передается центральным процессором внешнему устройству	
INTR RST 5.5 RST 6.5 RST 7.5 TRAP	Сигналы управления прерываниями, описанные в гл. 6.	
INTA/	Центральный процессор подтверждает запрос по сигналу прерывания INTR	
SID	Последовательный ввод данных (используется вместе с командой RIM)	
SOD	Последовательный вывод данных (используется вместе с командой SIM)	

Рис. 8.5. Сигналы центрального процессора Intel 8085.

Обозначение фирмы Intel	Описание сигнала
RESET IN/	Сбрасывает адрес следующей команды в нуль (эквивалентно JMP 0), восстанавливает шину и систему управления прерываниями и регистры центрального процессора
RESET OUT/	Включается, когда восстанавливается центральный процессор
X_1, X_2	Подключение управляющих компонентов к внутреннему генератору тактовых импульсов центрального процессора или вход для сигналов внешнего генератора тактовых импульсов
CLOCK	Выход внутреннего генератора тактовых импульсов центрального процессора, который используется для синхронизации других устройств

Рис. 8.5. (Продолжение).

Поскольку шина обеспечивает основной интерфейс между компонентами микрокомпьютера, начнем описание аппаратных средств микрокомпьютера Intel 8085 с нее. Для однозначности описания аппаратуры будем использовать соглашение о том, что сигнал может быть либо *включен*, либо *выключен*, что будет означать соответственно *активное* или *неактивное* состояние сигнала. Для сигнала *высокой активности,* обозначаемого наименованием без слеша (наклонной черты), *включение* соответствует высокому уровню сигнала или состоянию логической ЕДИНИЦЫ, а *выключение* — низкому уровню сигнала или состоянию логического НУЛЯ. Для сигнала *низкой активности,* обозначаемого наименованием со слешем, *включение* соответствует низкому уровню сигнала или состоянию логического НУЛЯ, а *выключение* — высокому уровню сигнала или состоянию логической ЕДИНИЦЫ.

В гл. 6 шины данных и адресные шины микрокомпьютера Intel 8085 были описаны как раздельные. Хотя функционально они и являются раздельными, восемь младших линий адресной шины являются физически совмещенными с линиями шины данных. Физическая совместимость линий шины необходима из-за того, что число соединений между центральным процессором Intel 8085 и его внешними устройствами ограничено сорока контактными выводами. Во время работы в адресной шине ($AD_0 — AD_7$ и $A_8 — A_{15}$) размещается 16-разрядный адрес, при этом включается сигнал ALE (Address latch enable — Разрешение фиксации адреса). Логическая схема ФИКСАТОРА АДРЕСА/СЕЛЕКТОРА БЛОКА посылает восемь младших разрядов адреса ($AD_0 — AD_7$) в устройство фиксации Intel 8212, как показано на рис. 8.7. Устройство фиксации *фиксирует*

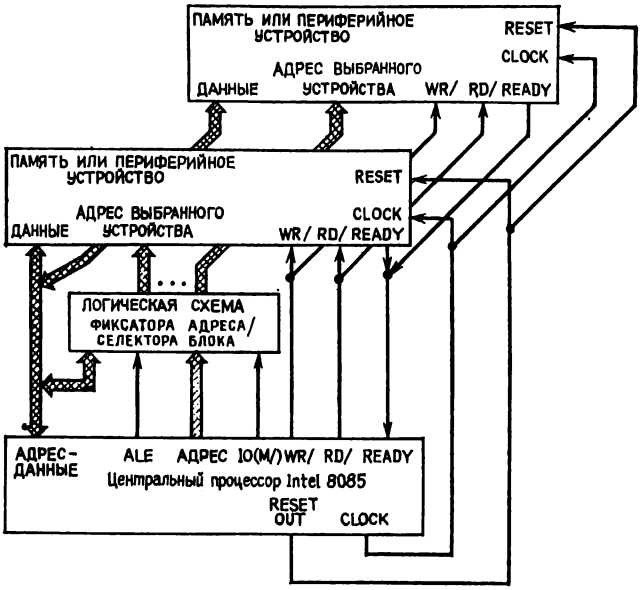


Рис. 8.6. Схема подключения памяти и периферийных устройств к центральному процессору Intel 8085.

или хранит эту информацию до тех пор, пока не будет послана новая информация. Если данные в ячейке памяти, адрес которой был зафиксирован, должны быть прочитаны центральным процессором, включается сигнал чтения RD/. Блок памяти, содержащий адресуемую ячейку памяти, помещает данные из нее в шину данных $AD_0 - AD_7$, которые передаются затем в центральный процессор. На рис. 8.8 показаны временные соотношения между этими операциями при выполнении команды языка ассемблера LDA VALUE. В гл. 6 мы отмечали, что при выполнении этой команды данные, хранящиеся в ячейке памяти, адрес которой представлен символическим именем VALUE, пересылаются в АККУМУЛЯТОР. Следует отметить, что старший байт адреса остается в части адресной шины $A_8 - A_{15}$ в течение всего времени, когда включен сигнал RD/, и поэтому не должен храниться в блоке фиксации. Дешифратор логической схемы ФИКСАТОРА АДРЕСА/СЕЛЕКТОРА БЛОКА обеспечивает для каждого блока памяти соответствующий сигнал выбора блока (устройства). Сигналы выбора блока обеспечивают отклик каждого блока памяти только в случае правильной адресации. Так как в отдельном блоке памяти содержится много адресов, дешифратор устанавливает соответствие между адресами, назначенными конкретному блоку, и сигналом

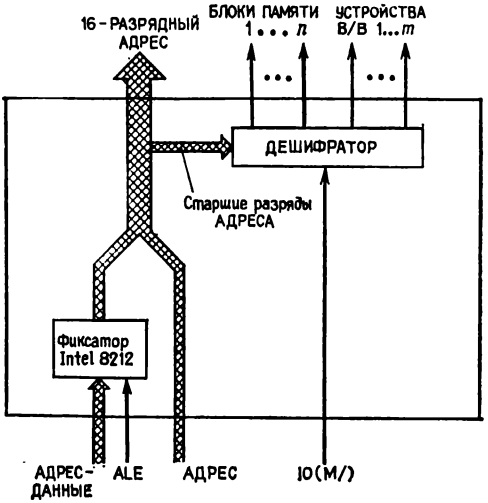


Рис. 8.7. Логическая схема ФИКСАТОРА АДРЕСА/СЕЛЕКТОРА БЛОКА.

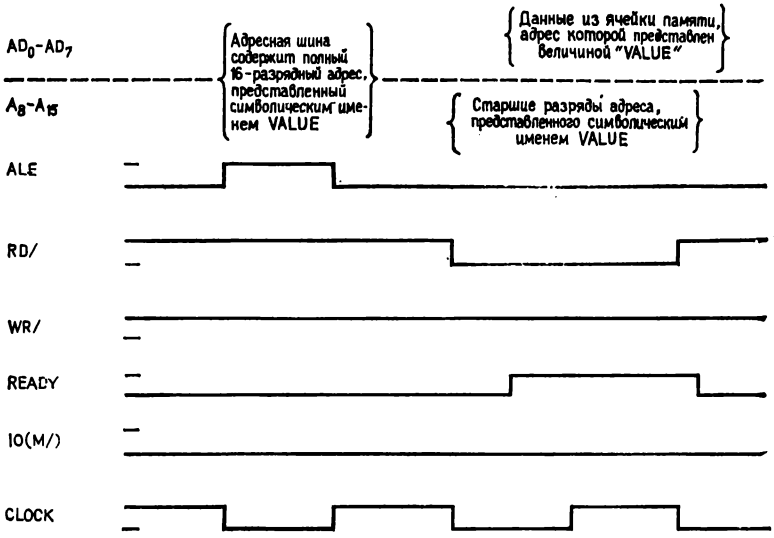


Рис. 8.8. Временная диаграмма операций шины во время выполнения команды LDA VALUE.

выбора этого блока. Необходимо обратить внимание на то, что сигнал IO(M/) должен иметь низкий уровень при выборе блока памяти.

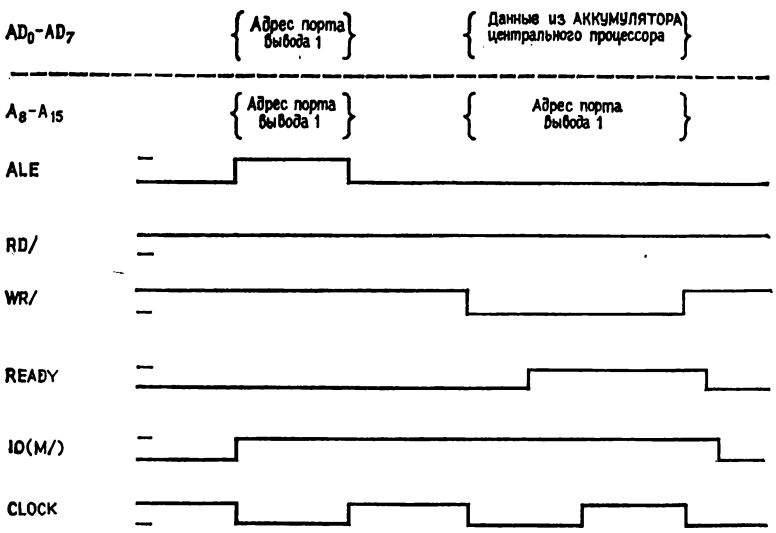


Рис. 8.9. Временная диаграмма операций шины во время выполнения команды OUT 1.

Если необходмо переслать данные из центрального процессора в ячейку памяти, вместо сигнала RD/ включается сигнал WR/. Данные помещаются центральным процессором в шину данных и пересылаются в адресуемую ячейку памяти.

Если данные необходимо послать не в ячейку памяти, а в порт вывода или получить их из порта ввода, то во время этой операции сигнал IO(M/) должен иметь высокий уровень, как показано на рис. 8.9 для команды OUT 1. Следует отметить, что адрес порта ввода или вывода является 8-разрядным, а не 16-разрядным числом. Для согласованности он размещается в обеих частях адресной шины в течение того времени, когда включен сигнал ALE. Однако в то время, когда включен сигнал RD или WR/, адрес остается в части A_8—A_{15} адресной шины, а данные размещаются в шине данных. Дешифратор использует 8-разрядный адрес порта и состояние сигнала IO(M/) для того, чтобы сгенерировать соответствующий сигнал выбора блока для каждого периферийного устройства. Каждому отдельному периферийному устройству может быть назначено несколько адресов портов. Дешифратор отображает все адреса портов, назначенные конкретному устройству, в сигнал(ы) выбора блока.

Сигнал READY используется для перевода центрального процессора в состояние ожидания, пока память или порт ввода-вывода завершат размещение данных в шине данных или из-

влечение данных из нее. Он обеспечивает синхронизацию, если, например, блок памяти не может выполнять операции так же быстро, как центральный процессор. Это позволяет использовать медленную память или периферийные блоки в приложениях, где приемлема невысокая скорость операций или недоступны высокоскоростные компоненты. Сигнал READY позволяет также осуществить синхронизацию при подключении микрокомпьютера Intel 8085 к системной шине, которую мы рассмотрим в одном из последующих разделов.

Другими сигналами шины являются сигналы S_0 и S_1, HOLD и HLDA. S_0 и S_1 являются разрядами состояния центрального процессора. Центральный процессор может находиться в одном из четырех состояний, обозначаемых следующим образом:

	S_1	S_0
ОСТАНОВ	0	0
ЗАПИСЬ	0	1
ЧТЕНИЕ	1	0
ВЫБОРКА	1	1

Состояние ОСТАНОВ указывает, что центральный процессор не использует шину. Состояние ЗАПИСЬ соответствует *записи в периферийное устройство*, как в примере для OUT 1 (рис. 8.9), и *записи в память*. Состояние ЧТЕНИЕ соответствует *чтению из периферийного устройства*, как в примере для LDA VALUE (рис. 8.8), и *чтению из памяти*. Состояние ВЫБОРКА аналогично состоянию ЧТЕНИЕ, (рис. 8.8), за исключением того, что адрес является адресом ячейки памяти, которая содержит байт команды, а не байт данных. После завершения состояния ВЫБОРКА байт команды, полученный из памяти через шину данных, хранится в УСТРОЙСТВЕ УПРАВЛЕНИЯ центрального процессора, как было описано в гл. 6. В зависимости от типа команды могут следовать подряд до трех состояний ВЫБОРКИ. Подобным же образом могут встретиться подряд до двух состояний ЧТЕНИЯ или ЗАПИСИ, если производятся операции над 16-разрядными данными.

Сигнал HOLD используется для передачи управления шиной внешнему устройству с целью его подключения к памяти или к порту ввода-вывода. Внешнее устройство включает сигнал HOLD, когда запрашивает центральный процессор передать управление шиной. После того как центральный процессор завершит текущую операцию и прекратит использование шины, он сообщает об освобождении шины включением сигнала HLDA. Затем центральный процессор продолжает выполнение операций

до тех пор, пока ему вновь не потребуется использование шины. В этом случае процессор должен ждать, пока внешнее устройство не выключит сигнал HOLD. После выключения сигнала HOLD центральный процессор выключает сигнал HLDA и продолжает операции с использованием шины обычным образом.

На этом мы завершаем описание шины микрокомпьютера Intel 8085. Поскольку операции с памятью и интерфейсом ввода-вывода самым теснейшим образом связаны с операциями с шиной, мы рассмотрим далее запоминающие устройства и интерфейсы ввода-вывода, используемые в микрокомпьютерных системах. После того как обсуждение этих вопросов будет завершено, мы рассмотрим остальные сигналы центрального процессора, представленные на рис. 8.5.

8.5. Запоминающие устройства

До сих пор мы предполагали, что каждая ячейка памяти может использоваться как для хранения, так и извлечения хранимой информации. Эти типы операций необходимы для той части памяти, которая используется для манипуляции структурами данных, содержащих изменяемые данные или параметры, или в качестве стека. Запоминающее устройство, используемое таким образом, называется *памятью с возможностью считывания и записи* (*read-write memory*). Однако по историческим причинам запоминающие устройства с возможностью считывания и записи называются *оперативными запоминающими устройствами* или ОЗУ (иногда — *запоминающими устройствами с произвольным доступом* или ЗУПД) [1]). Несмотря на то что термины *«считывание и запись»*, *«оперативный* (*произвольный*) *доступ»* не являются синонимами, тем не менее мы будем придерживаться стандартной терминологии и использовать сокращение ОЗУ для обозначения запоминающих устройств с возможностью считывания и записи. Различаются *статические* и *динамические* ОЗУ. Одно из отличий заключается в том что динамическое ОЗУ может содержать бóльшее количество информации в одном кристалле, чем статическое. Другое отличие заключается в том, что информация, хранимая в динамическом ОЗУ, должна периодически регенерироваться или *обновляться*. Поэтому при проектировании аппаратуры должен быть предусмотрен периодический сигнал обновления.

Другим типом запоминающих устройств является память с возможностью только считывания или постоянное запоминающее

[1]) Авторы используют сокращение RAM от Random Access Memory — запоминающее устройство с произвольным доступом. — *Прим. перев.*

устройство — ПЗУ [1]). Информация, хранящаяся в ПЗУ, либо не может быть изменена вообще, либо возможность ее изменения требует дополнительных затрат. Таким образом, ПЗУ используется для хранения структур данных, содержащих постоянные данные, или команд, составляющих программную часть системы. Команды обычно размещаются в ПЗУ только после того, как программное обеспечение объединено в систему и проверена правильность его функционирования. Постоянные данные и команды обычно не меняются в течение всего времени работы системы. Используя для хранения информации память, в которую невозможна запись, можно не заботиться о том, что информация может быть утеряна в результате падения напряжения или сбоя центрального процессора и случайной записи в ячейки памяти, где хранится постоянная информация.

Для того чтобы использовать в системе оба типа памяти, необходимо разделить всю память на две части: часть, в которой хранятся команды и постоянные структуры данных, и часть, в которой хранятся структуры данных, значения которых могут меняться во время операций. Большинство компиляторов, ассемблеров, редакторов связей и загрузчиков имеют средства для этой цели. При обсуждении PL/M в гл. 5 было описано использование атрибута DATA в описании данных с инициализацией PL/M-переменной, массива или структуры. Использование этого атрибута означает, что переменная, массив или структура назначена постоянной части памяти вместе с командами. Следовательно, для реализации этой части памяти можно использовать ПЗУ. Переменные, массивы и структуры, которые описаны без атрибута DATA, а также часть памяти, резервируемая для стека, должны быть размещены в изменяемой части памяти, для которой может быть использовано ОЗУ.

Имеется несколько типов ПЗУ. Они различаются по способу начального *программирования* или записи и по тому, насколько трудно изменить информацию после того, как ПЗУ запрограммировано. ПЗУ, *программируемые с помощью маски*, при изготовлении используют информацию, определенную заказчиком, независимо от того, представляет эта информация данные или команды. Такое ПЗУ не может быть изменено после его изготовления. В условиях крупносерийного производства программируемые ПЗУ являются весьма экономичными. За исключением начальных затрат, массовое производство ПЗУ является сравнительно недорогостоящим.

Другим типом постоянной памяти является ПЗУ *с плавкими перемычками*, которое может быть запрограммировано пользователем. Будучи однажды запрограммировано, оно уже не мо-

[1]) Авторы используют сокращение ROM от read only memory — память с возможностью только считывания. — *Прим. перев.*

жет быть изменено. В микрокомпьютерных системах разработки имеется устройство, называемое *программатором* ПЗУ, которое позволяет пользователю программировать ПЗУ путем пережигания плавких перемычек. Таким образом, необходимы затраты рабочей силы и времени для программирования каждого такого ПЗУ, что делает их более дорогостоящими по сравнению с ПЗУ, программируемыми с помощью маски. Тем не менее ПЗУ с плавкими перемычками являются очень выгодными в мелкосерийном производстве. Для них не нужны начальные затраты, как в случае ПЗУ, программируемых с помощью маски.

Третьим типом ПЗУ является *программируемое постоянное запоминающее устройство с возможностью стирания*, или СППЗУ. Оно может быть запрограммировано пользователем таким же способом, что и ПЗУ с плавкими перемычками. Многие программаторы ПЗУ могут быть использованы как для ПЗУ с плавкими перемычками, так и для СППЗУ. Однако информация, хранящаяся в СППЗУ, может быть удалена или стерта, а устройство — перепрограммировано. Большинство типов СППЗУ стирается при освещении их ультрафиолетовым светом в течение не менее десяти минут. Преимущество СППЗУ перед ПЗУ с плавкими перемычками заключается в том, что если обнаружена ошибка во время объединения системы, то нет необходимости выбрасывать СППЗУ, как это было бы в случае, если использовалось ПЗУ с плавкими перемычками. СППЗУ может быть стерто и записано заново. Недостатком СППЗУ является то, что оно не так надежно, как ПЗУ с плавкими перемычками. Запрограммированная в СППЗУ информация может теряться со временем, особенно если устройство не защищено от света и радиации. ПЗУ с плавкими перемычками и СППЗУ часто называют *программируемыми* ПЗУ или ППЗУ.

Четвертым типом ПЗУ является *электроизменяемое* ПЗУ, или ЭИПЗУ. ЭИПЗУ подобно ОЗУ, поскольку содержащаяся в нем информация может быть изменена во время работы, хотя и намного медленнее, чем в ОЗУ. Однако информация не теряется при выключении напряжения, в этом отношении ЭИПЗУ подобно ПЗУ.

На этом завершается описание различных типов запоминающих устройств на кристаллах, используемых проектировщиком систем. Далее мы рассмотрим интегральные схемы, используемые в качестве интерфейсов ввода-вывода.

8.6. Периферийные интерфейсные микросхемы

В следующих разделах описывается несколько общих периферийных микросхем, используемых с микрокомпьютером Intel 8085. Похожие периферийные устройства используются в

других микрокомпьютерных системах, они также могут быть подключены к микрокомпьютеру Intel 8085 и работать с ним. Аналогичным образом устройства, спроектированные для работы с микрокомпьютером Intel 8085, могут быть использованы с другими микрокомпьютерами. Будут описаны параллельный периферийный интерфейс общего назначения, последовательный связной интерфейс общего назначения и контроллер ПДП.

8.7. Параллельный периферийный интерфейс

Параллельный периферийный интерфейс общего назначения Intel 8255 (рис. 8.10) является весьма гибким устройством, которое может быть *запрограммировано* для выполнения самых

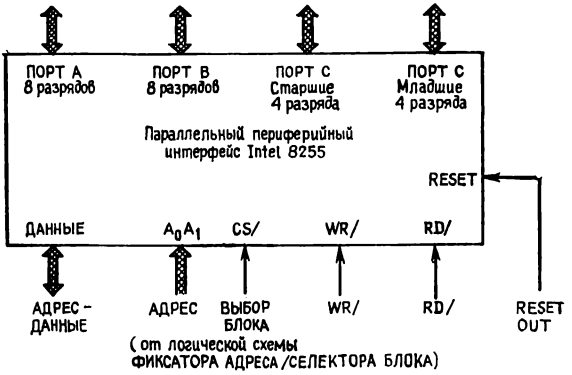


Рис. 8.10. Параллельный периферийный интерфейс общего назначения Intel 8255.

различных функций. У него имеются три 8-разрядных порта, которые могут быть использованы как для ввода, так и для вывода. Порт С разделен на два 4-разрядных порта, которые могут быть использованы либо как порты ввода-вывода, либо для других целей, как будет видно из дальнейшего. Периферийный интерфейс подключается к шине микрокомпьютера Intel 8085, используя сигналы, показанные на рис. 8.10. Две адресные линии A_1 и A_0 дополняют сигнал выбора блока, получаемый из логической схемы селектора блока. Рассмотрим, как эти три сигнала вместе с сигналами RD/ и WR/ управляют операциями устройства Intel 8255.

Когда сигнал выбора блока CS/ выключен, устройство игнорирует сигналы других линий управления. Однако если сигнал CS/ включен и при этом включен сигнал RD/ и выключен WR/, то это интерпретируется как запрос центрального процессора на ввод данных. В шину данных передается байт из

порта A, B или C в зависимости от логических состояний сигналов A_1 и A_0, показанных в таблице:

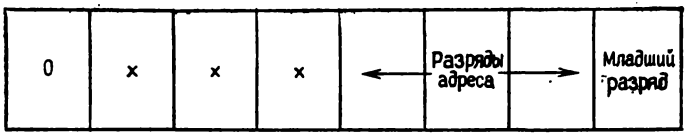
A_1	A_0	
0	0	Порт A
0	1	Порт B
1	0	Порт C

Как видно из таблицы, никакие две операции не могут выполняться одновременно, и, кроме того, они зависят от того, как инициализировано устройство. Если же сигнал CS/ включен, сигнал RD/ выключен, а сигнал WR/ включен, то запрос от центрального процессора интерпретируется как операция вывода. Байт, помещенный центральным процессором в шину данных, либо пересылается в порт A, B и C, либо интерпретируется устройством как *управляющее слово* в зависимости от логических состояний сигналов A_1 и A_0, как показано в таблице:

A_1	A_0	
0	0	Порт A
0	1	Порт B
1	0	Порт C
1	1	Управляющее слово

Так же как и во время ввода, никакие две операции не могут выполняться одновременно.

Управляющее слово используется либо для вывода бита данных через порт C, либо для выбора режима работы устройства. В первом случае можно использовать переменную с двумя состояниями (например, включение-выключение света), не затрагивая другие переменные. Во втором случае устройство Intel 8255 может быть *инициализировано* таким образом, что будет работать в одном из трех режимов: в основном режиме ввода-вывода (Режим 0), в стробированном режиме ввода-вывода (Режим 1) или в режиме двунаправленного интерфейса (Режим 2). Эти три режима влияют только на передачу информации между портами и внешними устройствами. Интерфейс между периферийным интерфейсным устройством и шиной микрокомпьютера Intel 8085 работает для всех трех режимов обычным образом, как было описано выше.



← Не используются →

0	0	0	Установить разряд 0 равным значению младшего разряда
0	0	1	Установить разряд 1 равным значению младшего разряда
0	1	0	Установить разряд 2 равным значению младшего разряда
0	1	1	Установить разряд 3 равным значению младшего разряда
1	0	0	Установить разряд 4 равным значению младшего разряда
1	0	1	Установить разряд 5 равным значению младшего разряда
1	1	0	Установить разряд 6 равным значению младшего разряда
1	1	1	Установить разряд 7 равным значению младшего разряда

Рис. 8.11. Управляющая информация, используемая для вывода отдельного бита.

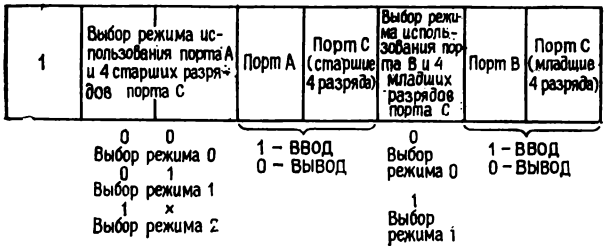


Рис. 8.12. Управляющая информация, используемая для выбора режима.

Как только управляющее слово попадает в устройство, проверяется значение его старшего разряда: если оно равно НУЛЮ, то младшие четыре разряда используются для вывода отдельного бита (рис. 8.11). Три разряда используются в качестве *адреса*, определяющего, какой из восьми разрядов порта должен быть установлен, а четвертый определяет, как должен быть установлен адресуемый разряд. Например, управляющее слово 0XXX 0111 определяет, что значение разряда 3 порта С должно быть установлено равным ЕДИНИЦЕ, в то время как 0XXX 1010 определяет, что значение разряда 5 порта С должно быть установлено равным НУЛЮ. Буква X используется для обозначения разряда, состояние которого игнорируется. Отметим снова, что никакие две операции не могут выполняться одновременно.

Если старший разряд управляющего слова равен ЕДИНИЦЕ, то остальные разряды используются для выбора режима работы устройства (рис. 8.12). Мы проиллюстрируем выбор режима с помощью нескольких примеров и опишем, как работает устройство в каждом из режимов. Управляющее слово 1001 0010 означает, что порты А и В работают в основном режиме ввода, а порт С — в основном режиме вывода. В основном режиме порт может быть использован либо для ввода, либо для вывода, но не для того и другого одновременно. В этом режиме выводимые величины *фиксируются,* т. е. разряды сохраняют свое текущее состояние до тех пор, пока не будут изменены командой вывода. Вводимые величины в этом режиме не фиксируются, передаваемые в центральный процессор данные определяются состоянием входных линий во время выполнения команды ввода. Таким образом, для рассматриваемого примера допустимыми командами центрального процессора являются команды ввода, *адресующиеся* к порту А ($A_1 = 0$, $A_0 = 0$) или к порту В ($A_1 = 0$, $A_0 = 1$), и команда вывода, адресующаяся к порту С ($A_1 = 1$, $A_0 = 0$). Следует отметить, что в основном режиме возможно использование четырех разрядов порта С

для ввода, а других четырех — для вывода. Это может быть выполнено, например, с помощью управляющего слова 1001 0011. Если порт С используется таким образом, проектировщик программного обеспечения или программист должны отдавать себе отчет в том, что команда вывода в порт С в этом способе вывода затрагивает только четыре разряда. Аналогично команда ввода, которая обращается к порту С, передает только четыре значимых бита центральному процессору. Остальные четыре бита в байте, полученном центральным процессором, должны быть проигнорированы в программе.

Управляющее слово 1011 0100 задает другой пример выбора режима. В этом примере порт А работает в стробирующем режиме ввода, а порт В — в стробирующем режиме вывода. В стробирующем режиме порт также может быть использован либо для ввода, либо для вывода. Разряды порта С используются в качестве управляющих сигналов для синхронизации работы портов А и В с их внешними устройствами. Эти сигналы называются *стробирующими* или *согласующими* сигналами. В этом режиме *фиксируются* как вводимые величины, так и выводимые. Вкратце согласование стробированного режима ввода осуществляется следующим образом. Когда устройство ввода, подключенное к порту ввода, готово послать данные в устройство Intel 8255, оно включает одну из линий порта С, играющую роль *стробирующего входного* сигнала для порта ввода. Устройство откликается тем, что пересылает вводимые данные на хранение в буферный регистр (фиксирует данные) и включает другую линию порта С, обозначенную как сигнал *заполнения буфера ввода* для порта ввода. Третья линия порта С может быть использована в качестве сигнала *запроса на прерывание* центрального процессора. Этот сигнал указывает, что буфер ввода содержит готовые к вводу данные. Сигнал запроса на прерывание может быть *маскирован* с помощью специальной команды вывода бита в порт С, которая устанавливает или сбрасывает разряд маски прерываний, когда устройство работает в режиме стробирования.

Согласование стробированного режима вывода заключается в следующем. Когда байт выводимых данных посылается из центрального процессора в устройство Intel 8255, он фиксируется в буферном регистре вывода. Устройство включает линию порта С, обозначенную как сигнал *заполнения буфера вывода* для порта вывода. Когда устройство вывода, подключенное к порту вывода, примет данные из порта вывода, оно включает линию порта С, обозначенную как сигнал *подтверждения* для порта вывода. Третья линия порта С может быть использована в качестве сигнала *запроса на прерывание* центрального процессора, указывающего, что порт вывода готов принять новые

данные. Как и в режиме ввода, сигнал запроса на прерывание может быть маскирован для подавления запросов на прерывания.

Порты A и B не обязательно должны работать одновременно в одном и том же режиме. Например, управляющее слово 1011 0000 переводит порт A в стробированный режим ввода, а порт B — в основной режим вывода.

Третьим режимом является режим двунаправленного интерфейса. Он применяется только к порту A, который действует в этом случае как стробированная, полностью фиксированная двунаправленная шина. В качестве примера рассмотрим управляющее слово 11XX X100, которое переводит порт A в режим двунаправленного интерфейса. Оно также переводит порт B в стробированный режим вывода. Для того чтобы использовать порт A в качестве двунаправленной шины, требуется пять линий порта C в качестве следующих управляющих сигналов: стробирования ввода, заполнения буфера ввода, заполнения буфера вывода, подтверждения и запроса на прерывание. Действия порта A в режиме двунаправленного интерфейса подобны описанным выше для стробированного режима, за исключением того, что пересылки ввода и вывода не могут выполняться в любой последовательности.

В режимах стробирования и двунаправленного интерфейса может выполняться команда ввода для чтения *данных* из порта C. Байт *данных*, считываемый в центральный процессор по этой команде, содержит информацию, указывающую *состояние* каждого из сигналов согласования, сигналов запросов на прерывания и состояния масок прерываний. На этом описание параллельного периферийного интерфейса общего назначения Intel 8255 заканчивается.

8.8. Последовательный связной интерфейс

Описанный выше периферийный интерфейс является параллельным устройством. Основная часть информации передается между интерфейсом и внешними устройствами по восемь бит за цикл через каждый из портов. Однако многие периферийные устройства, особенно те, которые используются для передачи информации на дальние расстояния, работают последовательно, передавая информацию по одному биту за цикл. Микрокомпьютер Intel 8085 может работать в последовательном режиме ввода-вывода одним из следующих способов: либо выполняя команды SIM и RIM, либо используя устройство связного интерфейса, например Intel 8251.

Команды SIM и RIM, описанные в гл. 6, используются преимущественно для манипулирования данными маски прерыва-

ний. Однако, как указывалось в гл. 6, эти команды могут быть также использованы для последовательной связи. Команда RIM может быть использована для считывания одного бита последовательных данных в старший разряд АККУМУЛЯТОРА. Считывание происходит по линии последовательного ввода данных SID (serial input data) (рис. 8.4). Аналогичным образом, если в АККУМУЛЯТОРЕ при выполнении команды SIM установлен разряд разрешения последовательного вывода, один бит последовательных данных пересылается из АККУМУЛЯТОРА в линию последовательного вывода данных SOD (serial output data) микрокомпьютера Intel 8085. Интерпретация разрядов АККУМУЛЯТОРА во время выполнения команды SIM показана на рис. 8.13. Использование этого способа последовательного ввода-вывода возможно при условии, если программные процедуры, оперирующие командами SIM и RIM, обеспечивают строгую синхронизацию последовательного интерфейса. Если же используется устройство Intel 8251, управляющая информация и данные передаются между устройством и центральным процессором параллельно. При этом управление синхронизацией автоматически осуществляется устройством связного интерфейса. Поскольку устройство Intel 8251 часто является более эффективным с точки зрения выбора между аппаратными и программными средствами, то мы описываем использование именно этого устройства для последовательной связи, а не команд RIM и SIM.

На рис. 8.14 схематически показано устройство связного интерфейса общего назначения Intel 8251[1]). У него имеется отдельный набор линий последовательного вывода в секции ПЕРЕДАЧА, отдельный набор линий последовательного ввода в секции ПРИЕМ и набор сигналов *управления модемами*, которые используются для упрощения интерфейса между устройством связи и *модемом*. *Модемом* называется устройство, используемое для подключения цифровых систем к телефонной линии при передаче информации на большие расстояния. Как показано на рис. 8.14, устройство Intel 8251 подключено к шине микрокомпьютера Intel 8085. Для управления данными необходимы единственная адресная линия, которая называется C(D/) (control — data), и сигнал выбора блока, получаемый из логической схемы выбора блока. Эти сигналы, а также сигналы RD/ и WR/ управляют работой устройства Intel 8251.

[1]) В ряде источников (см., например, Хоуп Г., Проектирование цифровых вычислительных устройств на интегральных схемах. — М.: Мир, 1984) устройство Intel 8251 называется USART (Universal Synchronous/Asynchronous Receiver/Transmitter или УСАПП (Универсальный синхронно-асинхронный приемопередатчик). Термин «связной интерфейс» чаще используется для обозначения БИС 6850 фирмы Motorola. — *Прим. перев.*

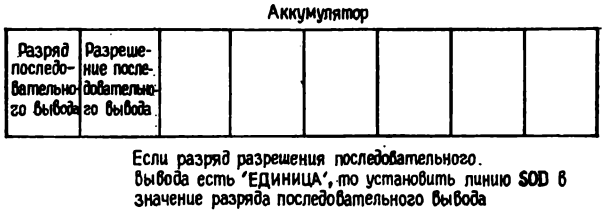


Рис. 8.13. Интерпретация разрядов последовательного вывода в АККУМУЛЯТОРЕ при выполнении команды SIM.

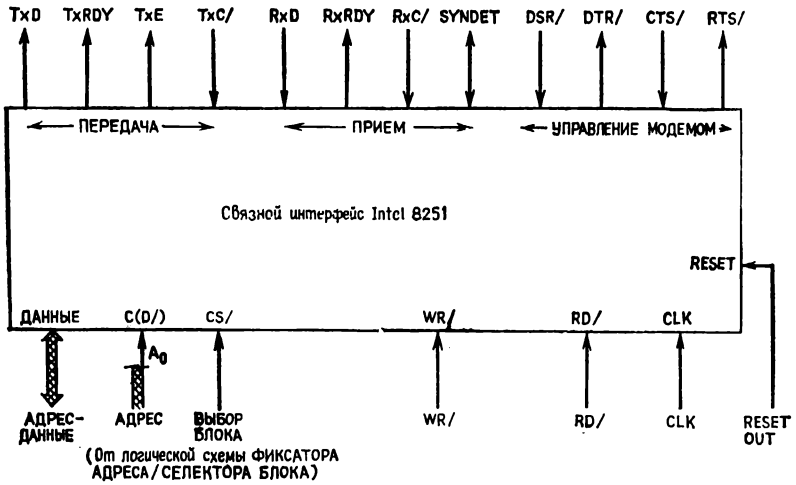


Рис. 8.14. Схема связного интерфейса общего назначения Intel 8251.

Так же как и устройство периферийного интерфейса, устройство связного интерфейса игнорирует информацию управляющих линий шины при выключенном сигнале выбора блока CS/. Когда сигнал CS/ включен и при этом включен RD/ и выключен WR/, устройство интерпретирует запрос центрального процессора как операцию ввода и передает ему через шину данных байт информации. Этот байт является байтом данных, полученным через последовательный ввод RxD, если C(D/) находится в состоянии «данные» (низкий уровень сигнала), или байтом управления/состояния, если C(D/) находится в состоянии «управление» (высокий уровень сигнала). Если же CS/ включен и при этом RD/ выключен, а WR/ включен, запрос интерпретируется как операция вывода центрального процессора. Байт, помещенный центральным процессором в шину данных, считается либо байтом данных и посылается в линию последователь-

ного вывода TxD, если сигнал C(D/) принимает низкий уровень, либо байтом управляющей информации, если сигнал C(D/) принял высокий уровень.

Управляющая информация, посылаемая в устройство Intel 8251, используется либо для выбора режима, либо для передачи устройству командной информации. Эта информация должна соответствовать строгому протоколу, обеспечивающему правильную работу устройства. Управляющий байт, следующий за появлением сигнала RESET, *должен* содержать информацию для выбора режима, которая настраивает устройство на желаемый режим работы. Этот байт управления режимом выбирает либо синхронный, либо асинхронный режим обмена данными, устанавливает коэффициент отношения частоты тактовых импульсов TxC/ или RxC/ к скорости передачи информации, или *бодскорости*, определяет число передаваемых или принимаемых битов в каждом байте данных, определяет тип проверки данных (либо на четность, либо на нечетность, либо никакой) и т. д. При работе в синхронном режиме в устройство должны поступать один или два *символа синхронизации* в зависимости от специальных характеристик синхронного режима. Эти символы используются для синхронизации работы устройства с внешними устройствами, так как это является единственным способом обеспечения правильности работы в синхронном режиме. Перед каждой передачей данных необходимо послать в устройство управляющий байт, содержащий командную информацию. Командный байт используется для разрешения или запрещения передачи или приема последовательных данных, для управления выходами модемов и т. д. После начального командного байта в устройство может быть послан либо байт данных, либо дополнительный командный байт в любой последовательности.

При считывании порта состояния устройства Intel 8251 в центральный процессор засылаются состояния различных линий управления. Кроме того, в центральный процессор поступают сообщения о любом из трех возможных типов встречающихся ошибок. Этими ошибками могут являться ошибки по четности, ошибка переполнения или ошибка формата. Ошибка по четности определяется сравнением принятого бита четности с битами принятых данных. Если обнаружена ошибка, то устанавливается *флажок ошибки по четности.* Если принятый символ не считывается в центральный процессор до того, как будет принят другой символ, устанавливается *флажок ошибки переполнения.* И наконец, если устройство работает в асинхронном режиме и при передаче символа отсутствует бит завершения символа, устанавливается *флажок ошибки формата.* Таким образом, устройство обладает значительными возможностями для

обнаружения ошибок и обеспечения правильности и достоверности передаваемых данных.

Полное обсуждение интерфейса между устройством связного интерфейса Intel 8251 и последовательным периферийным устройством или модемом выходит за рамки данной книги. Тем не менее мы кратко опишем сигналы, используемые для этой цели. Сигнал RxC/ (*receiver clock* — сигнал *синхронизации приема*) является внешним сигналом синхронизации, частота которого должна соответствовать скорости последовательной передачи данных в бодах по входной линии RxD (*receive data* — линия *приема данных*), умноженной на коэффициент отношения частоты, определяемый ранее описанным управляющим байтом. Аналогично этому сигнал TxC (*transmitter clock* — сигнал *синхронизации передачи*) является внешним сигналом, частота которого должна соответствовать скорости передачи последовательных данных, посылаемых в выходную линию TxD (*transmit data* — линия *передачи данных*), умноженной на тот же самый коэффициент отношения частоты. Таким образом, скорость передачи в бодах может отличаться от скорости приема в бодах, хотя для большинства приложений эти скорости обычно равны. При работе в синхронном режиме сигнал SYNDET (*sync detect* — сигнал *обнаружения синхронизации*) используется для указания того факта, что либо во время ввода был получен, либо во время вывода был передан символ синхронизации. Центральный процессор определяет способ использования сигнала SYNDET, устанавливая бит обнаружения внешней синхронизации в управляющем байте в ЕДИНИЦУ при вводе и в НУЛЬ при выводе.

Если устройство работает с модемом, то кроме описанных выше сигналов используются следующие четыре сигнала управления модемом:

DSR/ (*data set ready*) — сигнал *готовности модема*, который устанавливается модемом для сообщения устройству о том, что телефонное оборудование, подключенное к модему, готово к работе.

DTR/ (*data terminal ready*) — сигнал *готовности устройства* к приему данных, который устанавливается устройством для сообщения модему о том, что микрокомпьютер готов к работе.

CTS/ (*clear to send*) — сигнал *готовности к передаче*, который устанавливается модемом при запросе устройства на передачу данных.

RTS/ (*request to send*) — сигнал *запроса на передачу*, который устанавливается устройством при запросе модема на передачу данных.

Сигнал TxRDY (*transmitter ready*) — сигнал *готовности передатчика к обмену* означает, что устройство готово принять из

центрального процессора символ для вывода. Этот сигнал либо непосредственно воспринимается центральным процессором, либо используется для генерации запроса на прерывание. Аналогично предыдущему сигнал RxRDY (*receiver ready*) — сигнал *готовности приемника* к обмену — означает, что символ принят устройством и что устройство готово переслать этот символ центральному процессору. Так же как и в предыдущем случае, сигнал RxRDY может быть воспринят непосредственно или использован для запроса на прерывание. Последний сигнал TxE (*transmitter empty — передатчик «пуст»*) означает, что в устройстве нет символа для обмена. В синхронном режиме этот сигнал означает, что передаются символы синхронизации, являющиеся *заполнителями*. Если одна и та же линия используется как для передачи, так и для приема, т. е. устройство работает в *полудуплексном* режиме, сигнал TxE может быть использован для обозначения конца режима передачи. После этого центральным процессором может быть послан командный байт для перевода устройства в режим приема.

Третьим устройством периферийного интерфейса, который мы опишем, является контроллер ПДП.

8.9. Контроллер ПДП

Контроллер ПДП микрокомпьютера Intel 8085 является гибким устройством, которое обеспечивает поблочный обмен данных по каналу ПДП между памятью и периферийными устройствами. Работа контроллера ПДП с функциональной точки зрения обсуждалась в гл. 6 (см. рис. 6.35). Здесь же будет описана реализация системы, содержащей контроллер ПДП типа Intel 8257.

Одно устройство Intel 8257 содержит четыре независимых канала ПДП. Как и другие периферийные устройства, описанные выше, устройство Intel 8257 может быть *запрограммировано* для работы в различных режимах. Во время работы центральный процессор может адресоваться к устройству Intel 8257 как к обычному периферийному устройству. Такой режим работы называется *подчиненным* режимом и служит для пересылки команд и адресов в устройство и для получения от него информации о состоянии устройства. В *основном* режиме работы устройство непосредственно управляет действиями системы по обмену данных. В этом режиме устройство сначала должно запросить центральный процессор освободить шину. Когда это будет выполнено, устройство дает команду памяти считать данные из шины (записать данные в шину) и одновременно периферийному устройству записать данные в шину (считать данные из шины). Таким образом, информация передается между

памятью и периферийным устройством без вмешательства центрального процессора. Микрокомпьютер Intel 8085, содержащий контроллер ПДП типа Intel 8257, показан на рис. 8.15. Мы используем эту систему для описания различий между системами, содержащими контроллер ПДП и не содержащими его.

Так как контроллер ПДП должен одновременно управлять считыванием и записью как в памяти, так и в периферийном устройстве, он должен иметь для этих целей различные сигналы. Так, сигналам IO(M/), WR/ и RD/ соответствуют сигналы MEMW/ (*memory write — запись в память*) и MEMR/ (*memory read — чтение из памяти*) для памяти и сигналы IOW/ (*I/O write — запись в устройство ввода-вывода*) и IOR/ (*I/O read — чтение из устройства ввода-вывода*) для периферийных устройств, управляемых контроллером ПДП. ГЕНЕРАТОР УПРАВЛЕНИЯ ЛИНИЯМИ преобразует сигналы центрального процессора в вышеописанные сигналы, которые затем направляются в память и периферийным устройствам, управляемым контроллером ПДП. В результате этой модификации работа центрального процессора не изменяется.

Логическая схема ФИКСАТОРА АДРЕСА /СЕЛЕКТОРА БЛОКА, которая была показана на рис. 8.7, при использовании с контроллером ПДП Intel 8257 должна быть модифицирована. Модифицированная логическая схема ФИКСАТОРА АДРЕСА/ВЫБОРА БЛОКА показана на рис. 8.16. При использовании контроллера ПДП требуются две различные конфигурации адресной шины: одна для *подчиненного* режима и одна — для *основного*. Во время работы контроллера ПДП в подчиненном режиме восемь старших разрядов адресной шины принимаются непосредственно по линиям центрального процессора $A_8 — A_{15}$. Восемь младших разрядов пересылаются по линиям центрального процессора $AD_0 — AD_7$ и фиксируются управляющим сигналом ALE. Эти разряды доступны тристабильному буферу, если сигнал контроллера ПДП AEN (*address enable — адрес доступен*) *выключен*, так как сигнал AEN указывает, что контроллер ПДП работает в подчиненном режиме, когда сигнал выключен. Если контроллер ПДП работает в основном режиме, восемь старших разрядов адресной шины принимаются по линиям контроллера ПДП $D_0 — D_7$, которые мультиплексно подключены к линиям центрального процессора $AD_0 — AD_7$. Они фиксируются управляющим сигналом ADSTB и доступны тристабильному буферу, если сигнал AEN *включен*. Сигнал AEN при включении указывает, что контроллер ПДП работает в основном режиме. В последнем случае восемь младших разрядов адреса получаются непосредственно с линией $A_0 — A_7$ контроллера ПДП.

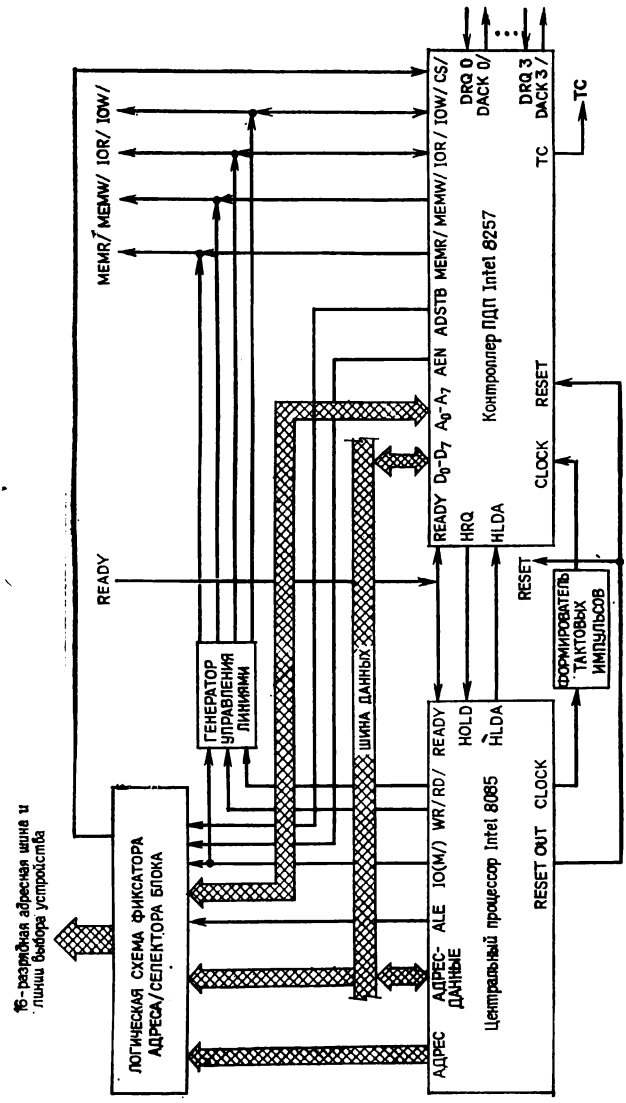


Рис. 8.15. Схема микрокомпьютера Intel 8085, содержащего контроллер ПДП Intel 8257.

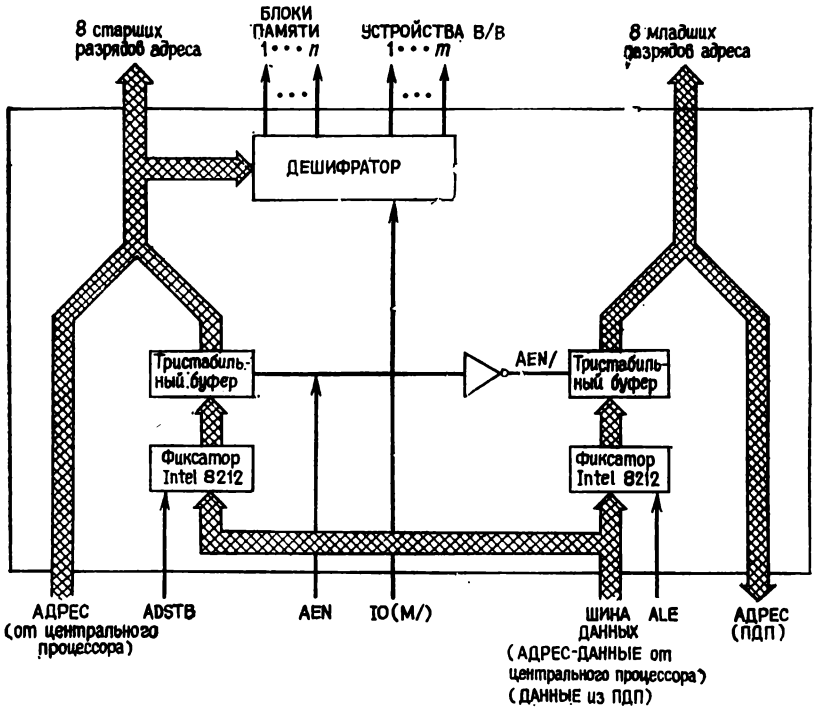


Рис. 8.16. Логическая схема ФИКСАТОРА АДРЕСА/СЕЛЕКТОРА БЛОКА микрокомпьютера Intel 8085, содержащего контроллер ПДП Intel 8257.

Прежде чем будет осуществлен поблочный обмен, канал ПДП и подключенное к нему периферийное устройство должны быть инициализированы. Инициализация выполняется программным способом путем засылки соответствующей информации в контроллер ПДП и в периферийные устройства. Рис. 8.17 показывает, как используется эта информация в зависимости от логического состояния адресных линий A_0 — A_3. Так как адрес ячейки памяти микрокомпьютера Intel 8085 представляется 16-разрядным числом, то для пересылки адреса блока памяти в контроллер ПДП требуется две передачи. Однако для каждого канала предусмотрен только один *адрес выбора блока*. Поэтому *обе* передачи осуществляются в один и тот же *адрес выбора блока*. В первой передаче посылается младший байт адреса блока памяти, а во второй — старший. Для обеспечения правильной синхронизации во время пересылки адреса блока памяти оба байта должны быть посланы один за другим. Кроме того, на это время должна быть заблокирована система прерываний, чтобы не допустить передачи посторонних данных.

A_3	A_2	A_1	A_0	
0	0	0	0	Адрес блока памяти для канала 0
0	0	0	1	Длина блока/режим для канала 0
0	0	1	0	Адрес блока памяти для канала 1
0	0	1	1	Длина блока/режим для канала 1
0	1	0	0	Адрес блока памяти для канала 2
0	1	0	1	Длина блока/режим для канала 2
0	1	1	0	Адрес блока памяти для канала 3
0	1	1	1	Длина блока/режим для канала 3
1	0	0	0	Установка режима (запись) Состояние (чтение)

Рис. 8.17. Инициализация контроллера ПДП.

Управляющее слово, содержащее информацию о длине блока и режиме, содержит 14-разрядное число, обозначающее длину блока, и два разряда для обозначения режима. Это слово должно быть послано в устройство с соблюдением тех же соглашений, которые были приняты относительно адреса блока памяти. Полученные устройством младшие четырнадцать разрядов интерпретируются как число $n - 1$, где n является числом передаваемых байтов. Два разряда для обозначения режима, являющиеся старшими, интерпретируются следующим образом:

0	1	Запись
1	0	Чтение

При установке контроллера ПДП в режим записи (чтения) осуществляется обмен входными (выходными) блоками между памятью и периферийным устройством.

Если центральный процессор выполняет программную передачу входных данных, используя один из восьми адресов портов ввода-вывода, описанных в соответствии с рис. 8.17, данные считываются из устройства следующим образом. Если используется *адрес* порта ввода-вывода, который был использован для инициализации адреса блока памяти для канала, в центральный процессор пересылается текущий адрес памяти для канала. Если используется *адрес*, который был использован для инициализации информации о длине блока/режима для канала, в центральный процессор пересылается текущая информация о длине блока и режиме для канала. Каждая из этих входных передач требует, чтобы были выполнены две следуемые одна за другой команды ввода, использующие один и

Режим автозагрузки	Режим останова ТС	Расширенный режим записи	Режим циклического приоритета	Канал 0	Канал 1	Канал 2	Канал 3

Рис. 8.18. Байт установки режима.

тот же *адрес* порта ввода-вывода. Как и прежде, должны быть приняты соответствующие предосторожности, чтобы не допустить передачи посторонних данных.

После того как регистры ПДП инициализированы вышеописанным способом, работа контроллера ПДП начинается с программируемой передачи входных данных в порт ввода-вывода по *адресу* 0001, что является операцией *установки режима* (рис. 8.17). Информация, содержащаяся в байте, посылаемом устройству во время этой операции, отпирает или блокирует каждый из четырех каналов либо разрешает или запрещает четыре режима работы контроллера ПДП, как показано на рис. 8.18.

Теперь мы опишем, как контроллер ПДП выполняет обмен данных в основном режиме после того, как он был инициализирован для поблочного обмена в подчиненном режиме. Допустим, что канал 0 (каналы ПДП нумеруются от 0 до 3) работает в режиме записи, т. е. периферийное устройство, подключенное к каналу 0, является входным устройством, что адрес первого блока памяти и длина блока посланы в устройство и хранятся в одном из его внутренних регистров и что канал открыт. Когда байт прочитан периферийным устройством и готов к передаче в память, периферийное устройство включает сигнал DRQ 0 (*DMA request* 0 — *запрос ПДП по каналу* 0). После этого контроллер ПДП запрашивает управление шиной, включая сигнал HRQ (*hold request* — *запрос на использование шины*). После того как центральный процессор завершит текущие операции с шиной, он передает управление шиной и включает сигнал HLDA. Как только сигнал HLDA получен, контроллер ПДП включает сигнал DACK 0/ (*DMA acknowledge* 0 — *подтверждение запроса на использование шины*), который может быть использован как сигнал выбора блока для контроллера периферийного устройства. Затем контроллер ПДП посылает адрес памяти в логическую схему ФИКСАТОРА АДРЕСА/СЕЛЕКТОРА БЛОКА, работа которого была описана ранее. Для завершения процесса контроллер ПДП включает сигналы IOR/ и MEMW/.

Периферийное устройство откликается на сигнал IOR/, помещая байт в шину данных. Выбранный блок памяти записы-

вает данные из шины в адресуемые ячейки памяти. Когда передача байта завершена, сигналы IOR/ и MEMW/ выключаются, и если передача блока не завершена, хранимые во внутренних регистрах адрес памяти и длина блока соответственно увеличиваются и уменьшаются на единицу. После этого периферийное устройство выключает сигнал DRQ 0, контроллер ПДП выключает сигналы DACK 0 и HRQ, а центральный процессор выключает сигнал HLDA и снова берет управление шиной. Отметим, что, если сигнал DRQ 0 не выключен, контроллер ПДП не передает управление шиной и сразу же выполняет другие передачи.

При использовании канала 0 для вывода периферийное устройство включает сигнал DRQ 0, когда оно готово принять байт из памяти. Последовательность включения аналогична случаю ввода, за исключением того, что для передачи данных из памяти через шину периферийному устройству включаются сигналы IOW/ и MEMR/.

При использовании контроллера ПДП типа Intel 8257 нет необходимости в адресе для выбора периферийного устройства. Периферийное устройство, которое включает сигнал DRQ, автоматически откликается на сигналы IOR/ или IOW/ контроллера ПДП, когда оно получает сигнал DACK/. Другие периферийные устройства, подключенные к контроллеру ПДП, игнорируют эти сигналы. Поэтому, если в системе совместно с контроллером ПДП используются другие периферийные устройства, не имеющие прямого доступа к памяти, необходимо предусмотреть средства, предотвращающие ошибочный запуск таких устройств, и некорректную передачу данных. Одним из способов является использование сигналов WR/ и RD/ для таких устройств. Другим способом является использование сигнала AEN, который формируется устройством Intel 8257 для блокировки во время работы всех устройств, не имеющих прямого доступа к памяти.

Если передаваемый байт является последним в блоке, то включается сигнал TC (*terminal count — отсчет конечного слова*), показанный на рис. 8.15. Этот сигнал может быть использован для сообщения центральному процессору о необходимости повторной инициализации канала ПДП для передачи следующего блока. Сигнал TC может быть либо считан центральным процессором программным способом, либо может быть использован в качестве сигнала прерывания.

Если выбран режим *автозагрузки* (рис. 8.18), канал 2 может быть использован для повторной передачи блока без повторной инициализации канала ПДП центральным процессором после завершения каждой передачи. Адрес блока, длина блока и информация о режиме для следующей передачи блока по каналу

2 могут быть посланы в канал 3 во время работы канала 2. В этом случае канал 2 после завершения очередной передачи может быть инициализирован автоматически, используя информацию, содержащуюся в канале 3. Если необходимо, сигнал TC, указывающий на завершение передачи блока, может быть использован для сообщения центральному процессору о необходимости перезагрузки канала 3 при подготовке к новой передаче следующего блока. При этом сигнал TC может быть либо считан центральным процессором, либо использован как сигнал прерывания. Необходимо отметить, что в режиме автозагрузки канал 3 не должен использоваться для каких-либо других целей.

В режиме *останова TC* канал автоматически запирается, как только передача блока завершена. Исключение допускается для канала 2, который не запирается в конце передачи блока, если при этом установлен режим автозагрузки. Так как на другие каналы режим автозагрузки не влияет, то они всегда запираются в конце передачи блока в режиме останова TC.

Расширенный режим *записи* может быть использован с некоторыми типами запоминающих и периферийных устройств для повышения эффективности работы системы. В этом режиме предусмотрена специальная синхронизация сигналов IOW/ и MEMW/ с целью лучшего согласования характеристик памяти и периферийных устройств. Обсуждение деталей этого режима работы мы опускаем.

Последним мы рассмотрим режим *циклического приоритета.* Обычно каналы ПДП работают с фиксированным приоритетом. Канал 0 имеет наивысший приоритет, а каналы 3 — низший. Таким образом, если в то время, когда контроллер ПДП принял управление шиной, включаются сигналы DRQ для двух или более периферийных устройств, то сигнал DACK/ включается для канала с высшим приоритетом, после чего происходит передача байта между подключенным к каналу устройством и памятью. Другие устройства ставятся в очередь в соответствии с приоритетом. Следовательно, обычный режим допускает монополизацию системы ПДП одним устройством. В режиме циклического приоритета приоритеты каналов меняются после передачи каждого байта, предотвращая таким образом монополизацию системы единственным устройством. В этом режиме после того, как канал обслужен, каналу со следующим порядковым номером присваивается наивысший приоритет для передачи следующего байта. Затем присваиваются приоритеты остальным каналам в соответствии с возрастанием их номеров. При этом считается, что канал 0 по приоритету следует за каналом 3.

В дополнение к возможности считывания адреса и длины блока, а также текущего режима работы каждого канала при

0	0	0	ФЛАЖОК ОБНОВЛЕНИЯ	СОСТОЯНИЕ ТС Канал 0	СОСТОЯНИЕ ТС Канал 1	СОСТОЯНИЕ ТС Канал 2	СОСТОЯНИЕ ТС Канал 3

Рис. 8.19. Интерпретация байта состояния каналов ПДП.

выполнении программируемой передачи ввода может быть считана информация о состоянии с использованием *адреса* 0001. Интерпретация битов байта состояния, полученного таким способом, показана на рис. 8.19. Четыре бита состояния ТС указывают, какие каналы сгенерировали сигнал ТС после того, как байт состояния был прочитан последний раз. Таким образом, биты состояния ТС указывают, была ли завершена передача блока в течение этого промежутка времени.

В режиме автозагрузки *флажок обновления* устанавливается, когда сигнал ТС включен, и остается в этом положении, пока регистры канала 2 обновляются из регистров канала 3. Следовательно, чтобы предотвратить преждевременное обновление информации в регистрах канала 3, необходимо считать байт состояния и проверить флажок обновления прежде, чем сделать попытку перезагрузить регистры канала 3.

На этом обсуждение переферийных интерфейсных устройств заканчивается. Читатель, которого интересуют дополнительные сведения, как, например, обозначения контактных выводов или вопросы синхронизации, должен обратиться к публикациям фирмы Intel и другим источникам, перечисленным в списке литературы. А сейчас вернемся к центральному процессору Intel 8085 и рассмотрим подробнее сигналы, введенные ранее в этой главе.

8.10. Сигналы микрокомпьютера Intel 8085 (окончание)

Оставшиеся сигналы центрального процессора Intel 8085 (рис. 8.5) воспроизведены на рис. 8.20. Их можно разделить на четыре категории: сигналы управления прерываниями, линии последовательных данных, линии восстановления и линии синхронизации.

Сигналы управления прерываниями INTR, RST 5.5, RST 6.5, RST 7.5 и TRAP были описаны в гл. 6. INTA/ включается центральным процессором для подтверждения того, что запрос на прерывание по линии INTR принят и что обработка прерывания началась. Как мы увидим, сигнал INTA/ может быть использован контроллером прерываний для засылки команд RESTART или CALL в центральный процессор. Назначение команд RESTART и CALL и их использование были описаны

Обозначение фирмы Intel	Описание сигнала
INTR, RST 5.5, RST 6.5, RST 7.5, TRAP	Сигналы управления прерываниями, описанные в гл. 6
INTA/	Центральный процессор подтверждает запрос по сигналу прерывания INTR
SID	Последовательный ввод данных (используется вместе с командой RIM)
SOD	Последовательный вывод данных (используется вместе с командой SIM)
RESET IN/	Сбрасывает адрес следующей команды в нуль (эквивалентно JMP 0), восстанавливает шину и систему управления прерываниями и регистры центрального процессора
RESET OUT/	Включается, когда восстанавливается центральный процессор
X_1, X_2	Подключение управляющих компонентов к внутреннему генератору тактовых импульсов центрального процессора или вход для сигналов внешнего генератора тактовых импульсов
CLOCK	Выход внутреннего генератора тактовых импульсов центрального процессора, который используется для синхронизации других устройств

Рис. 8.20. Сигналы центрального процессора Intel 8085.

в гл. 6. Контроллер прерываний Intel 8259 и его использование для реализации операций модуля ПРЕРЫВАНИЙ описываются в следующем разделе.

Линии последовательных данных SID и SOD и их использование с командами RIM и SIM были кратко описаны в разд. 8.7.

Линии восстановления используются для инициализации аппаратуры. RESET IN/ является внешним сигналом, который может быть получен от пусковой кнопки или от несложной схемы задержки времени, запускаемой при включении питания. Он восстанавливает внутреннее состояние центрального процессора и сигналов шины, сбрасывает систему прерываний и устанавливает в нуль содержимое регистра центрального процессора, в котором находится адрес следующей выполняемой команды. Поэтому при восстановлении системы центральный процессор сразу же начинает выполнять команду, хранящуюся в нулевой ячейке памяти. Сигнал RESET OUT включается, как только процессор восстановлен. Обычно он посылается в другие устройства системы для того, чтобы они были восстановлены одновременно с процессором. Если в нулевой ячейке памяти

хранится команда *перехода* к ИСПОЛНИТЕЛЬНОЙ процедуре, то после каждого восстановления начинается ее выполнение. В системе охранной сигнализации необходимо инициализировать указатель стека *прежде*, чем будет вызвана процедура ИНИЦИАЛИЗАЦИИ СИСТЕМЫ. Эта процедура затем вызывает процедуры ИНИЦИАЛИЗАЦИИ АППАРАТУРЫ и ВОССТАНОВЛЕНИЯ СИСТЕМЫ, и на этом инициализация системы завершается. Аналогичные процедуры должны вызываться в других системах. Как мы уже видели, процедура ВОССТАНОВЛЕНИЯ СИСТЕМЫ сбрасывает и инициализирует программные средства, за исключением указателя стека. Процедура ИНИЦИАЛИЗАЦИИ АППАРАТУРЫ выдает команды для *программирования* интерфейсных устройств. Таким образом, во время восстановления полностью инициализируются все аппаратные и программные функции. Следует отметить, что некоторые аппаратные команды должны выполняться в определенной последовательности в соответствии со специальными протоколами. По этой причине инициализация аппаратной части отделена от инициализации программной части, а процедура ИНИЦИАЛИЗАЦИИ АППАРАТУРЫ выполняется только тогда, когда система восстанавливается извне.

И наконец, линии X_1 и X_2 предназначены для подключения внешних управляющих компонентов к внутреннему генератору тактовых импульсов для синхронизаци тактовой частоты системы. Другим способом синхронизации работы внешних устройств между собой и с центральным процессором является подключение к линии X_1 внешнего генератора тактовых импульсов. При этом по линии CLOCK воспроизводятся копии этих сигналов, которые могут быть посланы другим устройствам.

8.11. Контроллер прерываний

Как мы уже видели, микрокомпьютер Intel 8085 имеет внутреннюю пятиуровневую систему прерываний. Поэтому для простых систем никаких дополнительных аппаратных прерываний не требуется. Однако во многих системах необходимо выполнять большое количество параллельных обработок, и поэтому пяти уровней прерываний может оказаться недостаточно. Контроллер прерываний Intel 8259 обеспечивает возможность аппаратного подключения до восьми линий для обработки сигналов запросов на прерывания. Он также обеспечивает обработку управляющей информации, необходимой для того, чтобы центральный процессор непосредственно откликался на эти внешние прерывания, как было описано в разд. 6.9. Помимо этого контроллер Intel 8259, являясь гибким устройством, позволяет реализовать множество стратегий прерываний. Например,

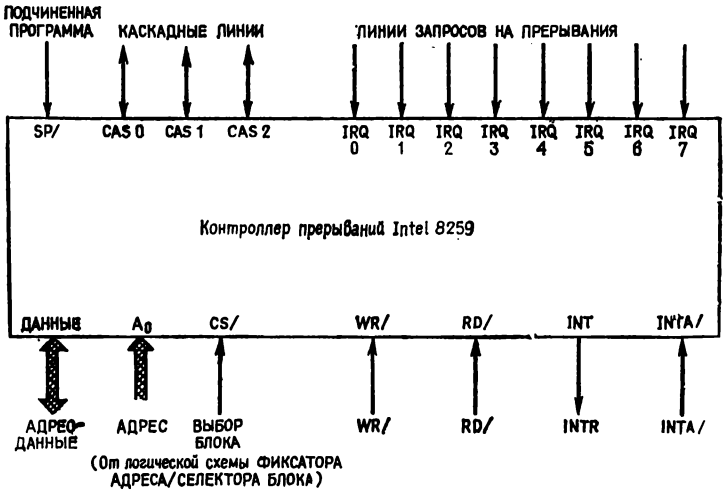


Рис. 8.21. Контроллер прерываний Intel 8259.

подключение дополнительных устройств Intel 8259 позволяет сравнительно просто довести число уровней векторных прерываний до шестидесяти четырех. Усовершенствованная версия контроллера прерываний Intel 8259A может подключаться и к микрокомпьютеру Intel 8085, и к Intel 8086, В этом разделе мы обсудим, как контроллер прерываний Intel 8259 обеспечивает возможность расширенных прерываний для центрального процессора Intel 8085, а также рассмотрим, как он может быть *запрограммирован* во время инициализации системы, обеспечивая требуемую стратегию прерываний.

Рис. 8.21 иллюстрирует, как контроллер прерываний Intel 8259 подключается к шине микрокомпьютера Intel 8085. Сигнал CS/ приходит из логической схемы выбора блока, A_0 обеспечивается адресной шиной, а сигналы WR/ и RD/ вырабатываются либо непосредственно центральным процессором, либо иногда ГЕНЕРАТОРОМ УПРАВЛЕНИЯ ЛИНИЯМИ (как сигналы IOW/ и IOR/) в системе, содержащей контроллер ПДП. Сигналы INT (или INTR) и INTA/ являются сигналами запроса на прерывание и подтверждения прерывания, которые связывают модуль прерываний с центральным процессором. Линии IRQ_0—IRQ_7 являются линиями внешних запросов на прерывания, а линии CAS_0—CAS_2 — *каскадными линиями*, позволяющими подключить дополнительный контроллер прерываний с целью расширения системы прерываний путем увеличения числа уровней прерываний сверх восьми. SP/ является входом *подчиненной программы*, который позволяет устанавли-

вать среди устройств иерархию типа *«основной — подчиненный»* [1]) с целью корректного функционирования системы. Далее в этом разделе мы опишем, как последние сигналы используются для расширения системы прерываний. Но сначала будет описано использование одного устройства Intel 8259 для управления восемью линиями внешних прерываний. В этом случае сигнал SP/ должен принять высокий уровень (или значение,

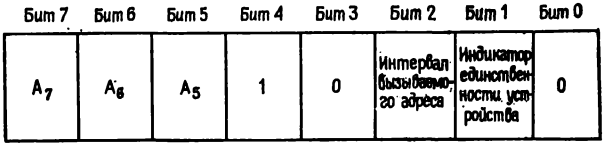


Рис. 8.22. Интерпретация первого байта инициализации ($A_0 = 0$).

равное ЕДИНИЦЕ), указывая тем самым устройству, что оно работает как основное, т. е. непосредственно управляет прерываниями, связанными с сигналом INTR.

Перед началом работы устройство Intel 8259 должно быть инициализировано. Для этого в устройство посылаются два программируемых выходных байта инициализации. Адресный бит A_0 первого байта должен быть равен НУЛЮ, а бит 4 байта инициализации — ЕДИНИЦЕ. Этот протокол позволяет отличить байты инициализации от командных слов, используемых для *программирования* устройства. Интерпретация битов первого байта инициализации показана на рис. 8.22. Бит 0 и бит 3 равны НУЛЮ, бит 1 является *индикатором единственности устройства*, бит 2 является *интервалом вызываемого адреса*, а биты 5—7 содержат биты $A_5 — A_7$ адреса *блока векторного прерывания*. Так как в системе работает один контроллер прерываний, индикатор единственности устройства должен быть установлен в ЕДИНИЦУ. Интервал вызываемого адреса определяет количество байтов, выделенных для адресов вызываемых команд, предусмотренных устройством для каждого из восьми уровней прерываний. Адрес вызываемой команды размещается в четырех отдельных байтах (если бит 2 равен ЕДИНИЦЕ) или в восьми отдельных байтах (если бит 2 равен НУЛЮ). Адресный бит A_0 должен быть равен ЕДИНИЦЕ, когда посылается второй байт инициализации. Второй байт инициализации содержит биты $A_8 — A_{15}$ адреса блока векторного прерывания. При использовании адресной информации, содержащейся в двух байтах инициализации, интервала вызываемого

[1]) В отечественной литературе используется также термин *«ведущий — ведомый»*. — *Прим. перев.*

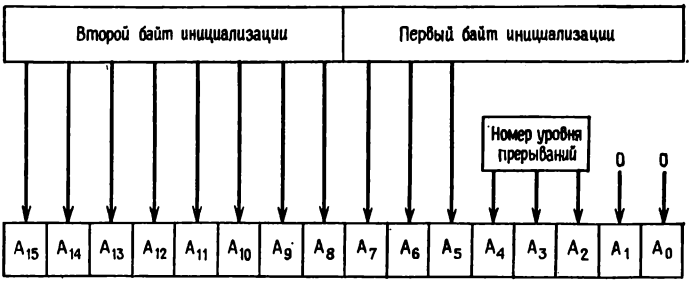


Рис. 8.23а. Генерация адреса вектора прерывания для четырехбайтного варианта.

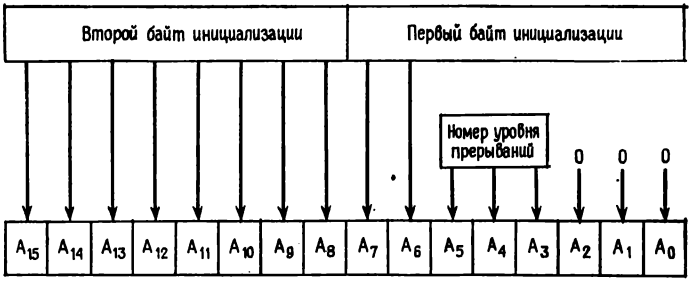


Рис. 8.23б. Генерация адреса вектора прерывания для восьмибайтного варианта.

адреса и номера уровня прерывания для каждой линии запроса на прерывание могут быть сгенерированы различные адреса (рис. 8.23а и рис. 8.23б) для четырех- и восьмибайтного вариантов соответственно.

После инициализации контроллер прерываний Intel 8259 функционирует следующим образом. Когда включается линия запроса на прерывание, устройство включает сигнал INT, посылаемый центральному процессору. После того как центральный процессор откликнется на это включением сигнала INTA/, контроллер прерываний помещает команду *вызова* в шину адрес-данных. Центральный процессор выключает сигнал INTA/ и выполняет команду. Так как для выполнения команды *вызова* требуется двухбайтная прямая адресация, сигнал INTA/ включается центральным процессором еще два раза. Каждый раз при включении сигнала INTA/ контроллер прерываний помещает байт адреса вектора прерываний, сформированный в соответствии с рис. 8.23, в шину адрес-данных, завершая этим последовательность операций.

Если устройству не послано никакого командного слова после его инициализации, ни одно из прерываний не маскируется. В этом случае при одновременном включении двух или более

линий запросов на прерывания для первого прерывания выбирается линия с наименьшим номером. Все остальные прерывания низшего уровня автоматически задерживаются до тех пор, пока устройство не выдаст центральному процессору команду *окончания прерывания.* Командные слова могут быть использованы для *программирования* контроллера прерываний и таким образом могут менять режим его работы в любой момент времени. Мы проиллюстрируем использование командных слов на нескольких примерах. Если в устройство послан байт с адресным битом A_0, установленным в ЕДИНИЦУ, и если устройство не ожидает байта инициализации, посланный байт воспринимается как *первое* командное слово в последовательности из трех командных слов. Это первое командное слово устанавливает или сбрасывает каждый бит маски прерываний, если соответствующий бит в командном слове равен соответственно ЕДИНИЦЕ или НУЛЮ. Мы описали концепцию маскированного прерывания в гл. 6. Если бит маски прерываний установлен, воздействие включенных линий запросов на прерывания задерживается до тех пор, пока маска не будет сброшена.

Дополнительные командные слова, посланные в устройство, позволяют использовать его в одном из следующих режимов: в *автоматическом* режиме, в котором уровню прерывания автоматически присваивается низший приоритет после того, как было обработано прерывание по этому уровню, или в режиме *специальной маски,* который позволяет использовать контроллер прерываний в режиме *последовательного опроса (полинга)* и который подготавливает устройство к посылке в центральный процессор информации о состоянии устройства. Дополнительные командные слова могут быть также использованы для получения от устройства информации о состоянии и для посылки в устройство команд окончания прерываний, как было указано ранее. Мы не будем обсуждать детально, как используются командные слова для указанных целей. Читатель может справиться об этом в руководствах фирмы Intel, упомянутых в списке литературы.

До сих пор мы предполагали, что используется не более восьми линий внешних прерываний и что в системе работает один контроллер прерываний. На рис. 8.24 показано каскадное подключение нескольких устройств Intel 8259 для обеспечения дополнительных уровней прерываний. Сигнал SP/ для устройства, используемого в качестве *основного,* принимает высокий уровень (ЕДИНИЦА), как и в обычном случае, а для остальных устройств Intel 8259 сигнал SP/ принимает низкий уровень (НУЛЬ), чтобы эти устройства работали как подчиненные. Сигнал INT каждого подчиненного устройства подключается к одному из входов IRQ основного устройства. Кроме того, ка-

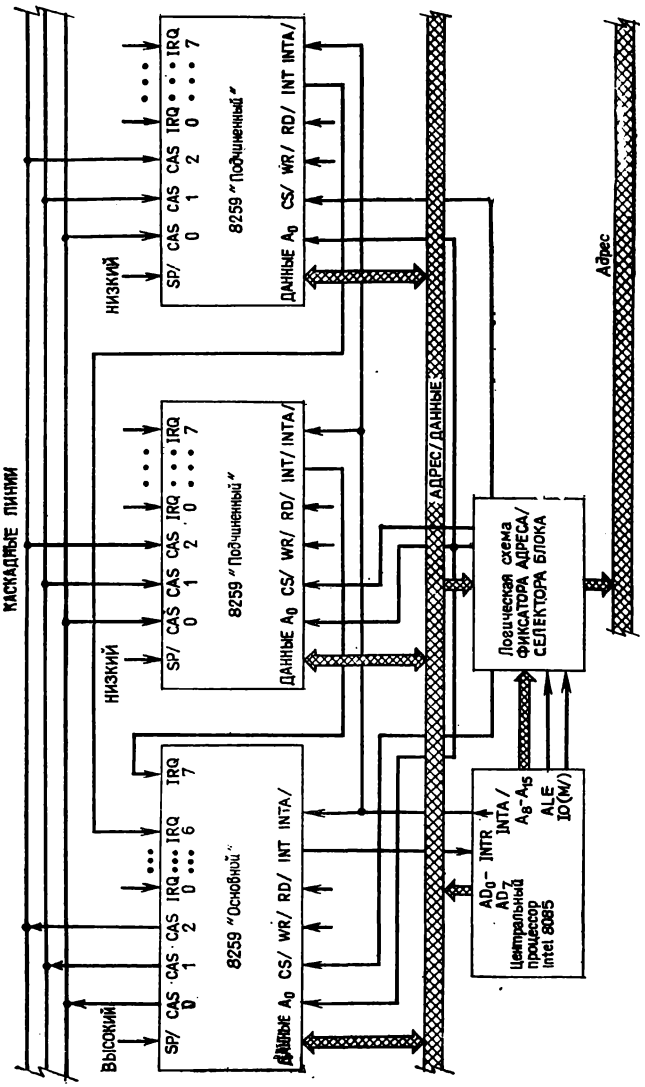


Рис. 8.24. Каскадное подключение контроллеров прерываний Intel 8259.

скадные линии CAS_0, CAS_1 и CAS_2 всех устройств соединяются вместе, как будто они являются *шиной* (рис. 8.24). Каскадные линии служат для передачи информации подчиненным устройствам от основного. Поэтому состояние сигнала SP/ для каждого устройства определяет, используются ли каскадные линии данного устройства для передачи или приема информации.

Конфигурация на рис. 8.24 содержит три контроллера прерываний Intel 8259. Такая конфигурация позволяет использовать до двадцати двух уровней прерываний. Максимально возможная для системы конфигурация может содержать девять устройств Intel 8259. В максимальной конфигурации к каждому из входов IRQ основного устройства подключены сигналы INT каждого из подчиненных устройств, что позволяет использовать до шестидесяти четырех уровней прерываний.

Когда контроллеры прерываний соединены в каскадную схему, для их правильного функционирования необходимо инициализировать каждое из устройств. Инициализация выполняется следующим образом. В каждое из устройств посылается последовательность из трех байтов инициализации (а не двух, как было в случае одного устройства). Индикатор единственности устройства, или 1-й бит первого байта инициализации, должен быть во всех случаях равен НУЛЮ. Остальные биты первого и второго байтов инициализации устанавливают интервал вызываемого адреса и адреса блоков векторных прерываний. Следует заметить, что для каждого устройства должны быть установлены различные блоки векторных прерываний. Индикатор единственности устройства в первом байте инициализации показывает, что устройства соединены в каскадную схему и что поэтому для каждого устройства необходим третий байт инициализации. Когда в устройство посылается третий байт инициализации, адресный бит A_0 должен быть равен ЕДИНИЦЕ. Третий байт инициализации, посылаемый в основное устройство, содержит биты для каждой его линии IRQ. Если бит равен ЕДИНИЦЕ, это указывает, что соответствующая линия IRQ подключена к сигналу INT от подчиненного устройства. Если же бит равен НУЛЮ, то линия IRQ либо не используется, либо подключена непосредственно к внешнему сигналу запроса на прерывание. Для схемы, показанной на рис. 8.24, значение третьего байта инициализации для основного устройства должно быть равным 1100 0000. Это означает, что IRQ_6 и IRQ_7 подключены к подчиненным устройствам. Третий байт инициализации, посылаемый в каждое подчиненное устройство, содержит в трех младших битах идентификатор, указывающий номер линии IRQ основного устройства, к которому подключено подчиненное устройство. Так, значения третьих байтов инициализации для двух подчиненных устройств в нашем примере

должны быть равны 0000 0110 и 0000 0111. Рассмотрим теперь, как функционирует каскадная система прерываний.

Если включается одна из подключенных к основному устройству линий внешних прерываний (IRQ_0—IRQ_5 в нашем примере), то последовательность включений и сигналов идентична тому, что было описано для системы с одним устройством. Если же включена одна из линий запроса на прерывание, подключенная к подчиненному устройству, оно включает свой сигнал INT. Основное устройство затем включает свой сигнал INT обычным способом и после включения центральным процессором сигнала INTA/ посылает центральному процессору команду вызова. Когда сигнал INTA/ включается второй и третий раз, основное устройство командует подчиненному, запрос которого пришел, послать адрес вектора прерывания центральному процессору. Это выполняется путем размещения идентификатора подчиненного устройства в каскадных линиях. Этим самым подчиненному устройству дается команда непосредственно откликнуться на сигнал INTA/. Если одновременно включаются две или более линий запроса на прерывание, приоритеты определяются как основным, так и подчиненными устройствами. Сначала основное устройство определяет приоритеты среди подчиненных устройств и своих собственных линий запросов на прерывания. Затем выбранное подчиненное устройство определяет приоритеты между своими линиями запросов на прерывания. При этом каждому контроллеру прерываний должно быть послано отдельное командное слово. Поэтому устройства могут работать в различных режимах. Для завершения каждого цикла обработки прерываний должны быть посланы два командных слова *окончания прерывания:* одно для основного устройства и одно для подчиненного, чей запрос на прерывание обслуживался.

Этим завершается описание контроллера прерываний Intel 8259. При описании этого устройства, запоминающих устройств и интерфейсных устройств ввода-вывода мы не касались вопросов технологии интегральных схем. Читатель отсылается к списку литературы для обращения к источникам, которые описывают различные технологии (N-МОП, К-МОП, Н-МОП, ТТЛ, $И^2Л$, ЭСЛ и т. д.), используемые при производстве интегральных схем на кристаллах, и которые описывают, как технология влияет на характеристики устройств.

8.12. Концепции системной шины

Описанная в предыдущих разделах шина является *внутренней шиной,* используемой для подключения компонентов микрокомпьютерной системы, центральным процессором которой является микросхема Intel 8085. Используются также другие

типы микрокомпьютерных шин, которые обеспечивают связь между внутренними шинами различных модулей. Одной из таких шин является Intel Multibus. Multibus является примером *универсальной системной шины*, которая может быть использована для подключения модулей *любой* системы с встроенным микрокомпьютером. Для шины Multibus Институтом инженеров по электротехнике и электронике (IEEE) были разработаны стандартные спецификации под названием IEEE-796. Модули, которые могут быть подключены к шине Multibus, могут быть такими простыми, как модули памяти или модули ввода-вывода, а также такими сложными, как модули ПДП или законченные микрокомпьютерные модули. Последние могут содержать различные центральные процессоры или архитектуры. Multibus позволяет формировать *мультипроцессорные* или многомашинные комплексы, подключая 8- и 16-разрядные микрокомпьютеры. Фирмой Intel Corporation используется соглашение, которое состоит в том, что компоненты одноплатного микрокомпьютера, такого, как iSBC 80/05, должны быть взаимосвязаны через внутреннюю шину. К другим платам они подключаются через Multibus. Кроме одноплатных микрокомпьютеров имеются совместимые с шиной Multibus модули памяти, модули ввода-вывода для связи с внешними цифровыми устройствами, модули ввода-вывода с аналого-цифровыми и цифроаналоговыми преобразователями для связи с внешними аналоговыми устройствами, модули контроллеров связи, модули контроллеров ввода-вывода специального назначения, содержащие контроллеры ПДП или дисковые контроллеры, и т. д.

На рис. 8.25 показано использование Multibus для построения мультипроцессорной системы. На рисунке показаны m микрокомпьютеров, каждый из которых может иметь свою собственную память, контроллеры ввода-вывода и прерываний. Показаны также n контроллеров ПДП, включающие такие специальные контроллеры ввода-вывода, как дисковый контроллер, p модулей памяти и s модулей интерфейсов ввода-вывода, включающие как цифровые, так и аналоговые интерфейсы и контроллеры связи. Модуль микрокомпьютера или контроллера ПДП может действовать как основная шина и, следовательно, управлять обменом данных с шиной Multibus либо действовать как подчиненная шина и передавать и принимать данные через шину под управлением одного из других модулей, действующих в качестве основной шины. Модуль памяти или ввода-вывода может действовать только как подчиненная шина. Показанная на рис. 8.25 *логическая схема согласования Multibus* служит для разрешения конфликтных ситуаций, возникающих, когда нескольким основным подсистемам требуется одновременно использовать Multibus.

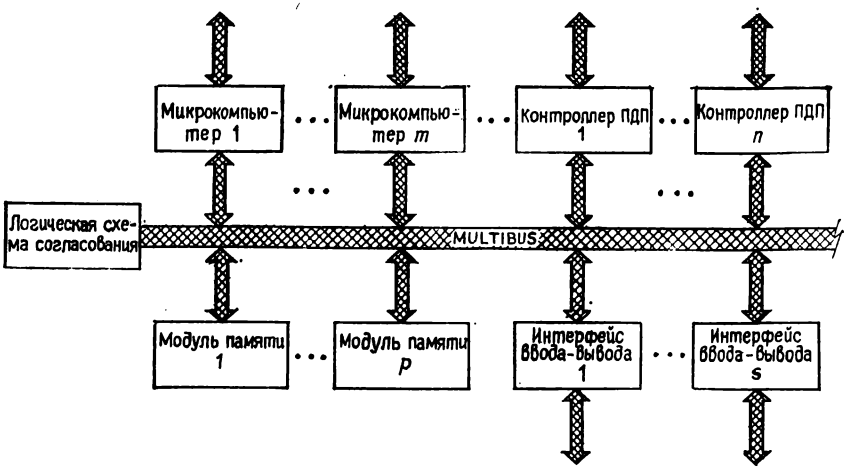


Рис. 8.25. Шина Intel Multibus.

Multibus имеет 16-разрядную шину данных. Информация может передаваться либо в виде 8-битовых байтов, использующих только половину линий шины данных, либо в виде 16-битовых слов, использующих все 16 разрядов шины. Он имеет также 20-разрядную адресную шину. Основной 8-разрядный модуль, например микрокомпьютер Intel 8085, использует шестнадцать адресных линий для связи с модулем памяти и восемь адресных линий для связи с портом в модуле интерфейса ввода-вывода. Основной 16-разрядный модуль, например Intel 8086, использует все двадцать адресных линий для связи с модулем памяти и до шестнадцати адресных линий для связи с портом в модуле интерфейса ввода-вывода. В шине Multibus предусмотрено также восемь линий прерываний. Поэтому модули, подключенные к шине Multibus, могут прерывать основную шину через линии прерываний шины Multibus, что обеспечивает дополнительные уровни прерываний.

Мы не будем детально описывать работу шины Multibus. Вместо этого рассмотрим ее работу с функциональной точки зрения. При установлении связи с подчиненным модулем модуль микрокомпьютера выполняет команду обращения к памяти или команду программного ввода-вывода. Если содержащийся в команде адрес памяти или адрес порта не обращается ни к одному из модулей, находящихся на той же плате, *интерфейс шины Multibus* (рис. 8.26) воспринимает это как обращение к модулю, расположенному на другой плате, и автоматически выдает запрос на использование шины Multibus. Если одновременно поступает несколько запросов на использование шины, логическая схема согласования Multibus разрешает такие ситуа-

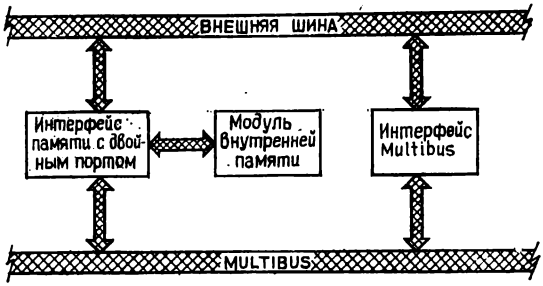


Рис. 8.26. Интерфейс между шиной микрокомпьютера и шиной Multibus.

ции в соответствии с текущим приоритетом. Как только интерфейс шины Multibus получит разрешение на использование шины, он завершает выполнение команды обращения к памяти или команды программного ввода-вывода, устанавливая связь с адресуемым модулем через шину Multibus. Если адресуемая ячейка находится в памяти другого микрокомпьютера, доступ к ней осуществляется через интерфейс памяти с двойным портом, который обеспечивает доступ к памяти либо из внутренней шины, либо из шины Multibus. Таким образом, интерфейс памяти с двойным портом позволяет модулю микрокомпьютера действовать как подчиненному устройству, когда его память доступна другому модулю. В то же время интерфейс шины Multibus позволяет действовать модулю микрокомпьютера как основному устройству, когда необходимо осуществить доступ к памяти, находящейся вне модуля микрокомпьютера, или к устройству ввода-вывода. Когда программные команды ввода-вывода *адресуются* к модулю ПДП, он действует как подчиненное устройство. Это позволяет микрокомпьютеру посылать в модуль ПДП данные инициализации и получать от него информацию о состоянии устройства.

Для связи микрокомпьютерных систем с такими устройствами ввода-вывода, как принтеры, используется другой тип универсальной системной шины. Примером шины такого типа является шина HP-IB фирмы Hewlett-Packard, которая соответствует стандарту IEEE-488 и служит для связи цифровых приборов в систему. Поскольку многие приборы в настоящее время содержат микрокомпьютеры, шина HP-IB на деле является микрокомпьютерной универсальной шиной. Шина S/100, которая согласуется со стандартом IEEE-696, является весьма гибкой шиной подобно шине Multibus. Она пользуется большой популярностью в области персональных микрокомпьютеров.

8.13. Вопросы экономики при выборе соотношения между аппаратными и программными средствами

В разд. 8.8 мы обсуждали проблемы выбора между аппаратными и программными средствами, в частности, использовать ли для последовательного обмена связной интерфейс Intel 8251 или проектировать программные процедуры, использующие команды RIM и SIM. Мы описали использование устройства Intel 8251, поскольку этот подход является экономически более выгодным с точки зрения выбора между аппаратными и программными средствами. Однако мы не рассмотрели всех экономических последствий такого решения. Сделаем это теперь.

Экономические последствия выбора соотношения между аппаратными и программными средствами включают следующие вопросы:

- Стоимость аппаратуры.
- Затраты на разработку аппаратных средств.
- Объем выпускаемой продукции.
- Затраты на разработку программного обеспечения.
- Срок жизни изделия.
- Время разработки и плановые сроки.
- Добавленная стоимость (условно-чистая продукция).
- Влияние выбора на характеристики системы.

При принятии решения необходимо рассмотреть все вышеперечисленные факторы. В качестве примера рассмотрим следующие предварительные оценки системы:

Стоимость одного устройства	$ 5	Включает стоимость подключения одного устройства к системе
Затраты на разработку аппаратных средств	$4000	Разность между стоимостью разработки аппаратуры с устройством и без него. Включает стоимость объединения
Объем производства	1000	устройств в год
Затраты на разработку программного обеспечения	$8000	Разность между стоимостью разработки программного обеспечения с устройством и без него. Включает стоимость объединения
Срок жизни изделия	4	года
Время разработки	1	месяц при разработке аппаратным способом
	2	месяца — программным
Влияние на общие сроки разработки	на 1/2	месяца дольше при разработке программного варианта

Таким образом, для аппаратной реализации дополнительные затраты в расчете на один год составят:

$$\text{Стоимость одного устройства} \times \text{Объем производства} + \\ + \frac{\text{Затраты на разработку аппаратуры}}{\text{Срок жизни изделия}} = \\ = \$5 \times 1000 + \frac{\$4000}{4} = \\ = \$6000.$$

При программной реализации дополнительные затраты в расчете на один год составят:

$$\frac{\text{Затраты на разработку программ}}{\text{Срок жизни изделия}} = \frac{\$8000}{4} = \$2000.$$

Следовательно, расчет затрат показывает, что программный подход в данном случае более выгоден. Читатель может проверить сам, что для небольшого объема производства экономически выгодной будет аппаратная реализация, а для большого — программная реализация может стать более выгодной. Следует учесть, что предложение о независимости стоимости одного устройства от объема производства является не совсем правильным и что это должно быть принято во внимание при более точной оценке затрат.

Кроме затрат мы должны также принимать во внимание другие вышеперечисленные факторы. Хотя в сроках разработки имеется определенное преимущество для аппаратной реализации, однако общее превышение сроков разработки составляет не более полумесяца, так как некоторые задачи объединения могут перекрываться. Мы также предположили, что персонал способен использовать преимущества перекрывающихся сроков. В этом случае влиянием на плановые сроки окончания разработки можно пренебречь. Если бы задержка в плановых сроках составляла несколько месяцев, необходимо было бы принять во внимание расходы, связанные с дополнительной зарплатой, которые могут возникнуть в результате увеличения сроков разработки изделия.

В зависимости от конкуренции может оказаться важной такая характеристика, как добавленная стоимость [1]). При отсутствии конкуренции отпускная стоимость может включать в себя дополнительные расходы, а за счет избыточной добавленной стоимости может быть получена чистая прибыль. В этом случае выгодным может оказаться наиболее быстрый способ разработки

[1]) Добавленная стоимость — стоимость изделия за вычетом стоимости материалов и полуфабрикатов, израсходованных в процессе производства. — *Прим. перев.*

системы, а дополнительными издержками можно пренебречь. В случае острой конкуренции, когда отпускная стоимость должна оставаться низкой, прибыль не может быть получена за счет добавленной стоимости. В этом случае необходимо рассматривать такое решение, которое связано с наименьшими затратами.

В заключение необходимо оценить влияние соотношения между аппаратными и программными средствами на характеристики производительности системы. Например, использование программного подхода для телеобработки может замедлить общую скорость работы системы. Может оказаться необходимым заменить центральный процессор на более производительный или использовать другой способ аппаратной реализации телеобработки в зависимости от того, что является более предпочтительным или экономически выгодным. Кроме того, использование программного подхода вместо аппаратного может увеличить потребность в памяти, при этом затраты на аппаратуру вместо того, чтобы снизиться, могут возрасти.

Таким образом, оценка соотношения между аппаратными и программными средствами становится многомерной задачей и должна быть тщательно рассмотрена прежде, чем проектирование будет завершено. Модульный подход к проектированию микрокомпьютерных систем позволяет заменять аппаратные модули программными — и наоборот — даже на сравнительно поздних этапах цикла проектирования. Однако необходимо предпринять все усилия, чтобы оценить выбор проектного решения на возможно более ранних этапах цикла проектирования системы, чтобы не пришлось позднее менять эти решения.

Прежде чем закончить с вопросами проектирования аппаратных средств, рассмотрим пятый уровень документации или уровень документации аппаратуры.

8.14. Пятый уровень документации

На пятом уровне документации нам необходимы как функциональные описания аппаратных средств, так и схематические и механические чертежи и рисунки, иллюстрирующие расположение и взаимосвязь устройств и модулей. Функциональное описание подобно функциональной спецификации и состоит в основном из текста. Поэтому оно может разрабатываться и поддерживаться подобно системной и программной документации. Схематические диаграммы и механические чертежи должны разрабатываться и поддерживаться с использованем стандартной технологии инженерного проектирования. Возможно использование на данном уровне систем графического отображения для автоматизации разработки и поддержки графической документации.

8.15. Упражнения

8.1. Рассмотрите, что является более предпочтительным при выборе интерфейса ввода-вывода микрокомпьютера телевизионного приемника (упражнения 3.2 и 3.3): параллельный периферийный интерфейс или контроллер ПДП Дайте обоснование своего выбора.

8.2. Рассмотрите требования ввода-вывода для устройства управления уличным светофором с встроенным микрокомпьютером (упражнения 3.4 и 3.5). Что предпочтительнее, параллельный периферийный интерфейс или последовательный связной интерфейс? Почему?

8.3. Какой тип интерфейса ввода-вывода предпочтительнее для электронного спортивного табло (упражнения 3.6 и 3.7)? Дайте обоснование своего выбора.

8.4. Рассмотрите интерфейс ввода-вывода между микрокомпьютером, клавиатурой и дисплеем в терминале розничной торговли (упражнения 3.8 и 3.9). Какой тип интерфейса вы предпочитаете? Почему?

Глава 9

Отладка и объединение аппаратных средств

В гл. 7 мы обсудили, как выполняются отладка и объединение программных модулей, и описали средства, используемые для этих целей. В гл. 8 мы обсудили проектирование различных аппаратных модулей, образующих систему. В настоящий момент мы готовы обсудить, как выполняются отладка и объединение аппаратных модулей, и описать средства, которые могут быть использованы при решении этих задач.

В течение цикла проектирования микрокомпьютерной системы желательно до начала объединения программного обеспечения получить аппаратные средства, которые являются не только действующими, но и по возможности свободными от ошибок. Во многих системах аппаратные модули бывают спроектированы и готовы к отладке прежде, чем будут завершены программные модули. С точки зрения плановых сроков проекта особенно важно устранить из аппаратуры как можно больше недоделок прежде, чем начнется объединение системы. Чем больше проблем обнаружено и скорректировано в течение фазы объединения аппаратных средств, тем больше времени остается для решения других вопросов при объединении системы.

В этой главе мы обсудим основные методы отладки аппаратуры, рассмотрим рекомендации по составлению *плана отладки аппаратных средств*, обсудим вопросы *статического* и *динамического* тестирования аппаратуры, рассмотрим пример использования нижнеуровневых программных модулей ввода-вывода для отладки аппаратных интерфейсных модулей микрокомпьютера и в конце опишем имеющиеся средства разработки и отладки аппаратуры.

9.1. Методы отладки аппаратных средств

Любая микрокомпьютерная система является уникальной в отношении конкретной конфигурации ее аппаратных средств. Поэтому мы будем использовать системный подход в вопросах отладки аппаратуры с тем, чтобы рассмотренные методы отладки можно было применять для самых различных конфигураций аппаратных средств. В этом разделе мы обсудим основные ме-

тоды, используемые для отладки аппаратных модулей перед объединением системы.

На рис. 9.1 показана обобщенная схема аппаратных модулей микрокомпьютерной системы. Мы используем эту схему для обсуждения вопросов отладки каждого из аппаратных модулей, показанных на рисунке. Чтобы определить, какая методика отладки гребуется для конкретного модуля, нам надо рас-

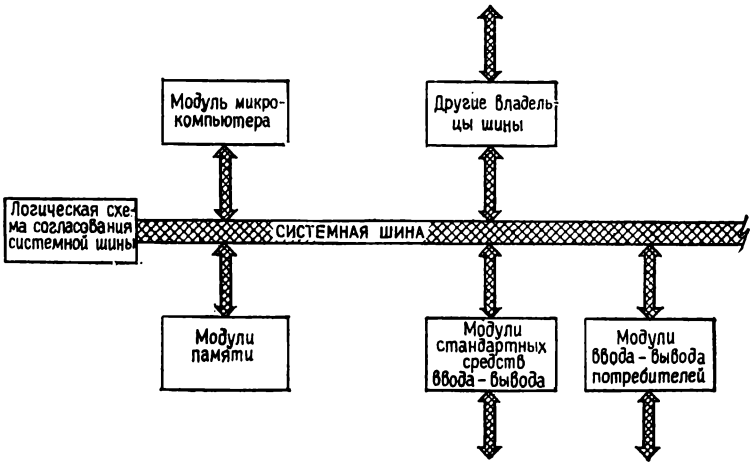


Рис. 9.1. Обобщенная схема аппаратных модулей.

смотреть, как эти модули взаимодействуют с программными средствами.

К аппаратным модулям, содержащим *только аппаратные средства*, т. е. модулям, которые слабо взаимодействуют или никак не взаимодействуют с программным обеспечением, применимы обычные методы отладки аппаратуры. Аппаратные модули, взаимодействующие с программным обеспечением, требуют комбинированной программно-аппаратной методики отладки. Из модулей, показанных на рис. 9.1, модуль МИКРОКОМПЬЮТЕРА, модули ПАМЯТИ и модули СТАНДАРТНОГО ВВОДА-ВЫВОДА являются в значительной степени программно-управляемыми. Модули ВВОДА-ВЫВОДА ПОТРЕБИТЕЛЕЙ и модули ДРУГИХ ВЛАДЕЛЬЦЕВ ШИНЫ меньше взаимодействуют с программными средствами и часто содержат большое количество *только аппаратных средств*.

Вначале мы обсудим, как выполняется отладка модулей, состоящих только из аппаратных средств. Этот класс аппаратных модулей имеет преимущество, состоящее в том, что отладка может быть начата сразу же после их конструирования и не требует координации со сроками разработки программных

средств. Вначале лучше всего проверить модуль с помощью статических автономных средств отладки, чтобы удостовериться в его правильном функционировании. После статической отладки модуля необходимо проверить его работу в динамике, чтобы убедиться, что его временные характеристики являются правильными. В следующих разделах мы обсудим методику статической и динамической отладки более подробно. После того как проверена правильность работы модуля в автономном динамическом режиме, он должен быть объединен с другими аппаратными модулями, чтобы проверить правильность его работы под

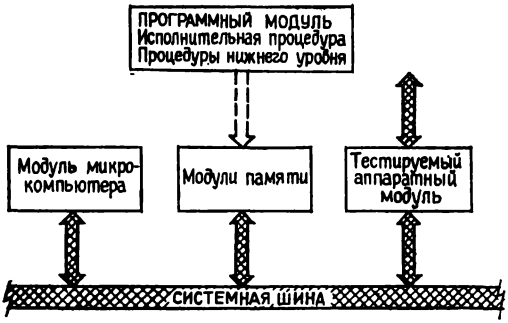


Рис. 9.2. Конфигурация для динамической отладки программно-управляемых аппаратных модулей.

управлением других частей системы. Систематическое последовательное продвижение от статической отладки к автономной динамической и далее к комплексной динамической отладке обеспечивает эффективный способ обнаружения, выделения и исправления в аппаратуре как проектных ошибок, так и конструкционных. Легче исправить очевидные недостатки и проверить функционирование модуля при статических условиях, чем в динамике. В то же время вопросы синхронизации могут быть легче разрешены, когда работа каждого модуля проверена в динамике отдельно от других модулей. Труднее всего обнаружить и выделить недостатки, встречающиеся в работе собранной системы. Поэтому наиболее эффективными являются идентификация и исправление как можно большего числа ошибок до того, как модули начнут работать в составе системы.

Отладка аппаратных модулей, работающих под управлением программных средств, может быть выполнена подобным же образом. Основное отличие состоит в том, что для полной отладки аппаратных модулей требуются программные средства. Как и прежде, мы начнем со статической отладки, необходимой для проверки правильного функционирования модуля. Затем модуль МИКРОКОМПЬЮТЕРА и модули ПАМЯТИ вместе с

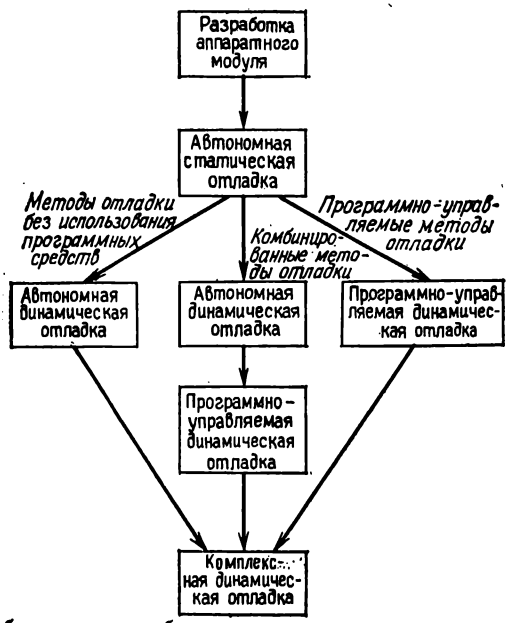


Рис. 9.3. Способы отладки и объединения аппаратных средств.

соответствующими тестирующими программами используются для динамической отладки программно-управляемых аппаратных модулей. На рис. 9.2 показана конфигурация модулей, используемая для такого тестирования. Так как программно-управляемые аппаратные модули тесно связаны с программным обеспечением, для управления аппаратурой во время динамического тестирования должны использоваться те программные процедуры нижнего уровня, которые проектируются для использования в конечной системе. Поэтому модули МИКРОКОМПЬЮТЕРА и ПАМЯТИ должны быть сконструированы и проверены ранее других модулей в рамках цикла проектирования с тем, чтобы они были доступны, когда появится необходимость тестирования программно-управляемых аппаратных модулей. Для обеспечения требуемой последовательности при динамическом тестировании необходимо предусмотреть процедуру ИСПОЛНЕНИЯ ТЕСТИРОВАНИЯ, которая вызывает существующие программные процедуры нижнего уровня в соответствующем порядке. Процедура ИСПОЛНЕНИЯ ТЕСТИРОВАНИЯ должна вначале вызывать процедуру инициализации. Она может также содержать цикл ВЫПОЛНЯТЬ НЕПРЕРЫВНО... КОНЕЦ для вызова процедур нижнего уровня при непрерывном тестировании аппаратуры. После того как тестирующие программные средства отлажены и работают правильно, можно

начинать процесс отладки аппаратуры. Он может выполняться так, как будто выполняется автономное динамическое тестирование только аппаратных средств. После того как обнаружены и исправлены ошибки в каждом аппаратном модуле, необходимо объединить все аппаратные модули, которые взаимодействуют между собой. Однако для трех модулей, которые взаимодействуют только с модулями МИКРОКОМПЬЮТЕРА и ПАМЯТИ, динамическая отладка и объединение заканчиваются одновременно.

Аппаратный модуль может содержать средства, которые должны тестироваться как без использования программного обеспечения, так и в программно-управляемой среде. В этом случае объединение состоит из комбинации методов, применимых для аппаратных средств, и методов программно-управляемой отладки. На рис. 9.3 представлена совокупность описываемых методов отладки и объединения аппаратных средств. В следующих разделах мы обсудим более подробно вопросы статической и динамической отладки и рассмотрим примеры программно-управляемой отладки аппаратных средств.

9.2. Автономная статическая отладка

В этом разделе мы рассмотрим аппаратные функции, которые могут быть отлажены статически. Мы также рассмотрим инструментальные средства, которые лучше всего подходят для этой цели. Как выше указывалось, автономная статическая отладка предназначена для проверки правильности функционирования модуля в соответствии с проектом. Различные функции, тестируемые во время отладки, включают проверку уровней напряжения питания, логической схемы выбора устройства, логических схем формирователя шины и регулятора направления шины, логической схемы генератора тактовых импульсов и любых других логических схем, входы которых можно возбуждать, наблюдая статические изменения на выходе. Инструментальные средства, используемые для проверки каждой из этих функций, перечислены на рис. 9.4. Каждое из этих средств будет подробно рассмотрено несколько позднее.

Очень важным моментом при проведении статической отладки модуля является проверка уровней напряжения питания как в точках подключения источника питания, так и на каждой интегральной схеме модуля. Если напряжение питания выходит за пределы определенных ограничений или подключено неправильно, модуль либо не будет работать совсем, либо будет работать ненадежно.

Следующими отлаживаются формирователи адресных линий, линий данных и управляющих линий, а также связанная с ними

Отлаживаемые функции	Цифровой вольтметр	Осциллограф	Имитатор сигналов	Логический анализатор
Питание шины и микросхем	X	X		
Формирование направления шины			X	X
Формирователи шины	X		X	X
Генераторы синхронизирующих сигналов		X		
Логическая схема выбора блока			X	X
Другие логические схемы	X		X	X

Рис. 9.4. Инструментальные средства для статической отладки.

логическая схема формирователя направления шины. Затраты времени на проверку того, что каждый бит, сформированный интерфейсом шины, передается в правильном направлении при различных условиях статического функционирования, оборачиваются в конечном итоге экономией времени.

Если аппаратный модуль содержит генератор тактовых импульсов, проверка правильности характеристик частоты, времени нарастания и амплитуды импульсов также должна быть выполнена во время статической отладки до начала динамической отладки.

В это же время может быть выполнена отладка других логических схем, включая логическую схему селектора устройства, которая дешифрует информацию адресной и управляющей шины и генерирует сигналы выбора устройства, необходимые во время работы системы. Важно удостовериться, что при размещении адреса устройства в шине оно выбирается правильно, с тем чтобы минимизировать проблемы выбора устройств во время динамической отладки аппаратуры.

9.3. Автономная динамическая отладка

Далее рассмотрим функции аппаратных средств, которые могут быть отлажены в динамике. Динамическая отладка проводится для тех функций каждого аппаратного модуля, которые выполняются при работе со скоростью, сравнимой со скоростью работы завершенной системы. Тесты, выполняемые во время автономной динамической отладки, проверяют правильность синхронизации и характеристик сигналов этих функций. Проблемы, вызванные неправильной синхронизацией или наличием шума, очень трудно обнаружить. Поэтому с меньшими затратами эти проблемы могут быть исправлены на уровне модулей, чем во время объединения системы.

Отлаживаемые функции	Имитатор сигналов	Импульсный генератор	Осциллограф	Логический анализатор
Сигналы шины	×	×	×	×
Логическая схема согласования шины	×	×	×	×
Время доступа к памяти	×	×	×	×
Логическая схема приоритетов шины	×	×	×	×
Логическая схема синхронизации	×	×	×	×

Рис. 9.5. Инструментальные средства для динамической отладки.

Аппаратные функции, которые могут быть проверены во время динамической отладки, включают логику согласования сигналов и приоритетов шины, время доступа к памяти, характеристики сигналов шины и другие функции, связанные с синхронизацией сигналов. Средства, используемые для отладки каждой из этих функций, перечислены на рис. 9.5. В отличие от статической отладки, когда для каждой функции требовалась особая конфигурация приборов, при динамической отладке каждой функции требуются все приборы, а именно: имитатор статических сигналов, импульсный генератор, осциллограф и логический анализатор. Во время динамической отладки имитатор статических сигналов и импульсный генератор используются на входе модуля, а осциллограф и логический анализатор — для наблюдения за выходами модуля.

Динамическая отладка начинается с проверки характеристик всех сигналов адресной шины, шины данных и управляющих сигналов. Времена нарастания и амплитудные уровни каждого из этих сигналов должны лежать в пределах допусков, определенных спецификациями на каждую интегральную схему, используемую в системе. Если к драйверу шины подключена слишком большая нагрузка, это может ухудшить характеристики времен нарастания и уровней потенциалов. Если необходимо, для регулирования шины можно добавить дополнительные усилители. Характеристики сигналов шины могут также ухудшаться за счет шумов при передаче на большие расстояния. Поэтому на практике часто используют усилители шины и приемники на обоих концах каждой шины с целью минимизации влияния шумов и для повышения надежности передаваемых по шине данных.

Если модуль содержит логические схемы согласования шины [1]) для обмена с памятью или устройствами ввода-выво-

[1]) У авторов для обозначения процесса опознания сигналов при асинхронной передаче данных используется термин *handshaking* (рукопожатие), который переводится в отечественной литературе как «квитирование» (см. также гл. 11). — *Прим. перев.*

да, необходимо проверить правильность синхронизации сигналов согласования. Большинство системных шин используют протокол согласования для обеспечения положительного подтверждения каждой передачи данных. Если в каком-либо модуле, использующем системную шину, логика согласования работает неправильно, обмен между модулями может задерживаться или не выполняться вообще.

Если время доступа к модулю памяти является критическим фактором для работы системы, оно может быть измерено и проверено во время автономной динамической отладки. При использовании имитатора статических сигналов для генерации действительных адресов памяти и импульсного генератора для генерации управляющих сигналов время доступа к памяти может быть измерено с помощью осциллографа или логического анализатора.

Модули, взаимодействующие с модулем СОГЛАСОВАНИЯ НАГРУЗКИ СИСТЕМНОЙ ШИНЫ, должны отлаживаться динамически для проверки того, что они правильно посылают информацию в модуль СОГЛАСОВАНИЯ НАГРУЗКИ ШИНЫ и принимают информацию от него. Если эти функции реализованы неправильно, система может заблокироваться при параллельной работе модуля с другими аппаратными модулями системы.

В конце отлаживаются и проверяются остальные логические схемы тестируемого модуля, связанные с синхронизацией и последовательностью выполнения сигналов. Здесь также необходимо просмотреть и проверить согласование во времени и характеристики сигналов, как было описано выше.

9.4. Программно-управляемая динамическая отладка

В этом разделе мы обсудим динамическую отладку аппаратных модулей, взаимодействующих с программным обеспечением. После того как статическое тестирование модулей закончено, программные процедуры нижнего уровня, которые проектируются для использования в конечной системе, применяются для управления аппаратными средствами при динамическом тестировании. Примерами аппаратных модулей, которые должны отлаживаться под управлением программного обеспечения, являются модули аналогового ввода-вывода, модули параллельного и последовательного цифрового ввода-вывода, программируемые таймеры, программируемые контроллеры прямого доступа к памяти (ПДП), программируемые контроллеры прерываний и другие программно-управляемые периферийные устройства.

Инструментальные средства, используемые для программно-управляемой отладки, включают те же средства, которые

использовались во время статической и динамической отладки, и, кроме того, такие средства, как микрокомпьютерный анализатор и внутрисхемный эмулятор, позволяющие одновременно наблюдать как программные, так и аппаратные события. Эти средства позволяют оперировать каждым аппаратным модулем, используя программные тесты, и одновременно наблюдать за выполнением программных средств и их влиянием на аппаратуру в условиях реального времени. Микрокомпьютерный анализатор и внутрисхемный эмулятор могут также подавать программно-синхронизируемые запускающие сигналы в осциллограф. Эта возможность позволяет исследовать динамические характеристики программно-управляемых сигналов.

Рассмотрим в качестве примера программно-управляемую динамическую отладку модуля последовательного связного интерфейса. Для обеспечения интерфейса между системной шиной и портом последовательного ввода-вывода в модуле используется устройство связного интерфейса Intel 8251, рассмотренное в гл. 8. Спецификация порта ввода-вывода согласуется со стандартом RS-232-C Ассоциации электронной промышленности (EIA). Для обеспечения требуемой скорости передачи (в бодах) в качестве генератора синхроимпульсов используется программируемый таймер Intel 8253. Блочная структура модуля показана на рис. 9.6.

Конфигурация для проведения отладки этого модуля показана на рис. 9.7. Для отображения выводимых модулем символов и для ввода символов в модуль к нему подключен терминал со стандартным интерфейсом типа RS-232-C. Для управления программно-синхронизируемыми событиями и для их отображения к модулю МИКРОКОМПЬЮТЕРА подключен микрокомпьютерный анализатор или внутрисхемный эмулятор. Для контроля за работой аппаратных функций модуля может быть использован логический анализатор или осциллограф.

Как указывалось выше, процедура ИСПОЛНЕНИЯ ТЕСТИРОВАНИЯ предназначена для вызова существующих программных процедур нижнего уровня в соответствующем порядке. Процедура ИСПОЛНЕНИЯ ТЕСТИРОВАНИЯ должна вызывать процедуру ИНИЦИАЛИЗАЦИИ АППАРАТУРЫ и содержать цикл ВЫПОЛНЯТЬ НЕПРЕРЫВНО... КОНЕЦ для повторного вызова процедур нижнего уровня. Описание на языке проектирования соответствующих процедур для использования во время отладки показано на рис. 9.8—9.11. Эти же процедуры показаны на рис. 9.12—9.15 после конвертирования языка проектирования в язык ассемблера Intel 8085. Показаны также листинги ассемблера Intel 8085, так как они содержат адреса команд. Прежде чем приступать к конкретным

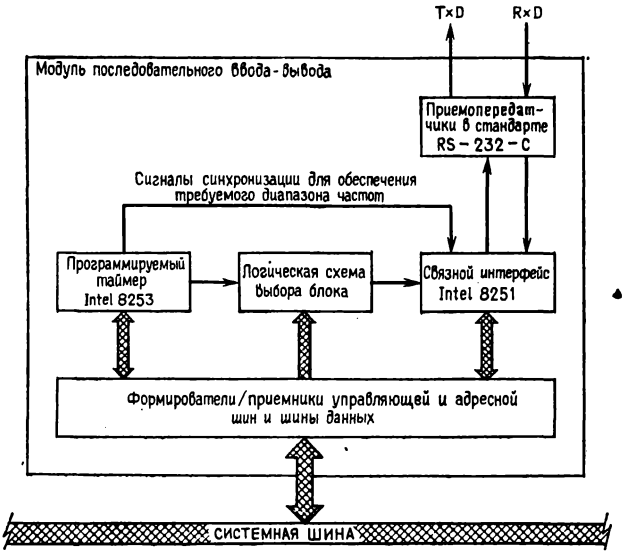


Рис. 9.6 Блок-схема модуля последовательного связного интерфейса.

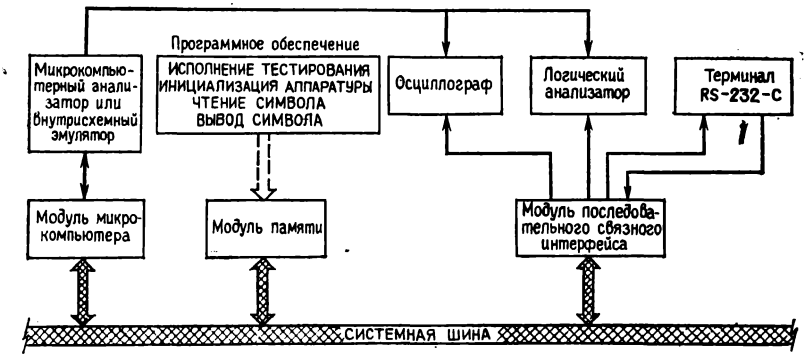


Рис. 9.7. Конфигурация для отладки модуля последовательного связного интерфейса.

шагам отладки, совершим беглый обзор этих проверочных процедур.

Процедура ИСПОЛНЕНИЯ ТЕСТИРОВАНИЯ сначала вызывает процедуру ИНИЦИАЛИЗАЦИИ АППАРАТУРЫ, а затем непрерывно вызывает процедуры ЧТЕНИЯ СИМВОЛА и ВЫВОДА СИМВОЛА. Каждый символ, полученный от процедуры ЧТЕНИЯ СИМВОЛА как выходной параметр, немедленно посылается процедуре ВЫВОД СИМВОЛА в качестве

```
PROCEDURE: TEST EXECUTIVE (;)
*****************************************************************************
DESIGNED BY:    L. B. EVANS                    01-26-81
LOCAL PARAMETER:  CHARACTER
*****************************************************************************
BEGIN PROCEDURE
  CALL: INITIALIZE HARDWARE (;)
  DO FOREVER
    CALL: READ CHARACTER (;CHARACTER)
    CALL: OUTPUT CHARACTER (CHARACTER;)
  END
END PROCEDURE
*****************************************************************************
```

Рис. 9.8. Процедура ИСПОЛНЕНИЯ ТЕСТИРОВАНИЯ.

входного параметра. Эта повторяющаяся последовательность чтения-вывода обеспечивает внешнее динамическое воздействие для отладки модуля последовательного связного интерфейса.

Процедура ИНИЦИАЛИЗАЦИИ АППАРАТУРЫ выполняет функции установления связного интерфейса Intel 8251 и таймера Intel 8253 в рабочий режим, требуемый приложениями. В первой части этой процедуры инициализируется таймер таким образом, чтобы он действовал как генератор прямоугольных импульсов, обеспечивая требуемые сигналы синхронизации RxC/ и TxC/ для связного интерфейса. Выбранный режим, в котором работает связной интерфейс, использует коэффициент умножения, равный шестнадцати. Это означает, что частота сигналов синхронизации должна быть в шестнадцать раз больше требуемой скорости передачи в бодах. Требуемые сигналы синхронизации генерируются таймером путем деления тактовой частоты системной шины на соответствующее целое число. На рис. 9.16 перечислены делители, требуемые для различной частоты сигналов синхронизации (предполагается, что тактовая частота системной шины равна 921,6 кГц). На рис. 9.13 команды в ячейках памяти 0010 H—0013 H (символом H обозначаются шестнадцатеричные величины) вызывают пересылку байта режима в устройство таймера и использование таймера номер 2 (в устройстве имеются три таймера) в качестве генератора прямоугольных импульсов. Байт режима готовит устройство к приему следующих двух байтов, которые используются как 16-разрядный делитель. Вначале пересылается младший байт делителя, а вслед за ним — старший, что реализуется с помощью команд в ячейках памяти 0014 H — 001 B H (рис. 9.13). В нашем примере для использования во время отладки выбрана скорость передачи 9600 бод, т. е. аппаратура отлаживается при той же скорости, с которой она будет использоваться в действительности. В случае когда тре-

```
PROCEDURE  INITIALIZE HARDWARE (;)
*************************************************************************
DESIGNED BY  L. B. EVANS              02-16-81
LOCAL PARAMETER NOT USED IN DESIGN LANGUAGE:  BAUD RATE FACTOR
PORT DEFINITIONS  INTEL 8253 INTERVAL TIMER
                  INTEL 8251 COMMUNICATIONS INTERFACE
*************************************************************************
BEGIN PROCEDURE
  INITIALIZE TIMER #2 TO ACT AS A SQUARE WAVE GENERATOR (MODE 3) AT 9600 BAUD
  INITIALIZE COMMUNICATIONS INTERFACE FOR RS-232-C TERMINAL
    RESET CHIP AND ASSURE THAT IT IS READY TO RECEIVE MODE BYTE
    SEND MODE BYTE TO SET CHIP TO 16X BAUD RATE, 8 BITS, NO PARITY, 1 STOP BIT
    SEND COMMAND BYTE TO ENABLE TRANSMIT AND RECEIVE AND SET DTR/ TO 0 AND RTS/ TO 0
  RETURN
END PROCEDURE
*************************************************************************
```

Рис. 9.9. Процедура ИНИЦИАЛИЗАЦИИ АППАРАТУРЫ.

```
PROCEDURE: READ CHARACTER (;CHARACTER)
*************************************************************************
DESIGNED BY:  L. B. EVANS            01-26-81
OUTPUT PARAMETER:  CHARACTER
*************************************************************************
BEGIN PROCEDURE
  WAIT UNTIL CHARACTER IS AVAILABLE AT INPUT PORT
  SET CHARACTER TO CHARACTER CODE READ FROM INPUT PORT
  RETURN
END PROCEDURE
*************************************************************************
```

Рис. 9.10. Процедура ЧТЕНИЯ СИМВОЛА.

```
PROCEDURE: OUTPUT CHARACTER (CHARACTER;)
*************************************************************************
DESIGNED BY:  L. B. EVANS            01-26-81
INPUT PARAMETER: CHARACTER
*************************************************************************
BEGIN PROCEDURE
  WAIT FOR OUTPUT PORT TO COMPLETE PREVIOUS CHARACTER TRANSMISSION
  SET OUTPUT PORT TO CHARACTER CODE TO BE TRANSMITTED
  RETURN
END PROCEDURE
*************************************************************************
```

Рис. 9.11. Процедура ВЫВОДА СИМВОЛА.

```
:F1:ASM80 EXEC.ASM PRINT (EXEC.LST) DEBUG
ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0
 LOC  OBJ          LINE        SOURCE STATEMENT
                      1 ;                          PROCEDURE: TEST EXECUTIVE (;)
                      2 ;                          ***************************************************
                      3 ;                          DESIGNED BY: L. B. EVANS          01-26-81
                      4 ;                          LOCAL PARAMETER: CHARACTER
                      5           DSEG
 0000                 6   CHAR:  DS    1
                      7 ;                          ***************************************************
                      8 ;                          BEGIN PROCEDURE
                      9   EXEC:
                     10 ;                            CALL: INITIALIZE HARDWARE (;)
                     11           CSEG
 0000 CD0100    D    12           CALL  INIT
                     13 ;                            DO FOREVER
                     14   LOOP:
                     15 ;                              CALL: READ CHARACTER (;CHARACTER)
 0003 CD0200    D    16           CALL  READ
 0006 320000    D    17 .         STA   CHAR
                     18 ;                              CALL: OUTPUT CHARACTER (CHARACTER;)
 0009 210000    D    19           LXI   H,CHAR
 000C 4E             20           MOV   C,M
                     21 ;                            END
 000D C30300    C    22           JMP   LOOP
                     23 ;                          END PROCEDURE
                     24 ;                          ***************************************************
```

Рис. 9.12. Листинг ассемблера для процедуры ИСПОЛНЕНИЯ ТЕСТИРОВАНИЯ.

```
LOC  OBJ  LINE      SOURCE STATEMENT

          25 ;                        PROCEDURE: INITIALIZE HARDWARE (;)
          26 ;                        ************************************************************
          27 ;                        DESIGNED BY: L. B. EVANS       02-16-81
          28 ;                        LOCAL PARAMETER NOT USED IN DESIGN LANGUAGE:  BAUD RATE FACTOR
          29          DSEG
0006      30  BR9600  EQU  0006H
          31 ;                        PORT DEFINITIONS: INTEL 8253 INTERVAL TIMER
0033      32  TIMCTL  EQU  33H
0032      33  TIMER2  EQU  32H
          34 ;                                    INTEL 8251 COMMUNICATIONS INTERFACE
0021      35  SERCTL  EQU  21H
0020      36  DATUM   EQU  20H
          37 ;                        ************************************************************
          38 ;                        BEGIN PROCEDURE
          39  INIT:
          40 ;                          INITIALIZE TIMER #2 TO ACT AS A SQUARE WAVE GENERATOR (MODE 3)
          41          CSEG                                                        AT 9600 BAUD
0010 3EB6 42          MVI  A,0B6H
0012 D333 43          OUT  TIMCTL
0014 3E06 44          MVI  A,LOW (BR9600)
0016 D332 45          OUT  TIMER2
0018 3E00 46          MVI  A,HIGH (BR9600)
001A D332 47          OUT  TIMER2
          48 ;                            INITIALIZE COMMUNICATIONS INTERFACE FOR RS-232-C TERMINAL
          49 ;                              RESET CHIP AND ASSURE THAT IT IS READY TO RECEIVE MODE BYTE
001C 3E00 50          MVI  A,00H
001E D321 51          OUT  SERCTL
0020 D321 52          OUT  SERCTL
0022 D321 53          OUT  SERCTL
0024 3E40 54          MVI  A,40H
0026 D321 55          OUT  SERCTL
          56 ;                            SEND MODE BYTE TO SET CHIP TO 16X BAUD RATE, 8 BITS,
0028 3E4E 57          MVI  A,4EH                                          NO PARITY, 1 STOP BIT
002A D321 58          OUT  SERCTL
          59 ;                            SEND COMMAND BYTE TO ENABLE TRANSMIT AND RECEIVE
002C 3E27 60          MVI  A,27H                                    AND SET DTR/ TO 0 and RTS/ TO 0
002E D321 61          OUT  SERCTL
          62 ;                            RETURN
0030 C9   63          RET
          64 ;                        END PROCEDURE
          65 ;                        ************************************************************
```

Рис. 9.13. Листинг ассемблера для процедуры ИНИЦИАЛИЗАЦИИ АППАРАТУРЫ.

буются меньшие скорости, их можно получить путем модификации процедуры ИНИЦИАЛИЗАЦИИ АППАРАТУРЫ.

Вторая часть процедуры ИНИЦИАЛИЗАЦИИ АППАРАТУРЫ инициализирует устройство связного интерфейса. Для инициализации устройства требуются следующие три шага:

1. Установить устройство в исходное состояние, обеспечив его готовность к приему байта режима.
2. Переслать байт режима в устройство.
3. Послать командный байт в устройство.

Чтобы обеспечить установку устройства в исходное положение перед тем, как послать байт режима, вначале из програм-

```
LOC  OBJ          LINE        SOURCE STATEMENT

                    66 ;                          PROCEDURE: READ CHARACTER (;CHARACTER)
                    67 ;                          ************************************************************
                    68 ;                          DESIGNED BY: L. B. EVANS        01-26-81
                    69 ;                          OUTPUT PARAMETER: CHARACTER
                    70            DSEG
0001                71   CHAR1:   DS   1
                    72 ;                          ************************************************************
                    73 ;                          BEGIN PROCEDURE
                    74   READ:
                    75 ;                            WAIT UNTIL CHARACTER IS AVAILABLE AT INPUT PORT
                    76            CSEG
0031 DB21           77   WAIT1:   IN   SERCTL
0033 FE02           78            CPI  02H
0035 CA3100     C   79            JZ   WAIT1
                    80 ;                            SET CHARACTER TO CHARACTER CODE READ FROM INPUT PORT
0038 DB20           81            IN   DATUM
003A E67F           82            ANI  7FH
003C 320100     D   83            STA  CHAR1
                    84 ;                            RETURN
003F C9             85            RET
                    86 ;                          END PROCEDURE
                    87 ;                          ************************************************************
```

Рис. 9.14. Листинг ассемблера для процедуры ЧТЕНИЯ СИМВОЛА.

```
LOC  OBJ          LINE        SOURCE STATEMENT

                    88 ;                          PROCEDURE: OUTPUT CHARACTER (CHARACTER;)
                    89 ;                          ************************************************************
                    90 ;                          DESIGNED BY: L. B. EVANS        01-26-81
                    91 ;                          INPUT PARAMETER: CHARACTER
                    92            DSEG
0002                93   CHAR2:   DS   1
                    94 ;                          ************************************************************
                    95 ;                          BEGIN PROCEDURE
                    96            CSEG
                    97 ; OUTPUT:
0040 210200     D   98            LXI  H,CHAR2
0043 71             99            MOV  M,C
                   100 ;                            WAIT FOR OUTPUT PORT TO COMPLETE PREVIOUS CHARACTER
0044 DB21          101   WAIT2:   IN   SERCTL                          TRANSMISSION
0046 FE01          102            CPI  01H
0048 CA4400     C  103            JZ   WAIT2
                   104 ;                            SET OUTPUT PORT TO CHARACTER CODE TO BE TRANSMITTED
004B 3A0200     D  105            LDA  CHAR2
004E D320          106            OUT  DATUM
                   107 ;                            RETURN
0050 C9            108            RET
                   109 ;                          END PROCEDURE
                   110 ;                          ************************************************************
                   111 ;                          END MODULE
                   112 ;            END
```

Рис. 9.15. Листинг ассемблера для процедуры ВЫВОДА СИМВОЛА.

Скорость в бодах	Частота синхронизации, кГц	Делитель
9600	153,6	6
4800	76,8	12
2400	38,4	24
1200	19,2	48
300	4,8	192

Рис. 9.16. Делители для сигналов синхронизации, генерируемых таймером.

мы посылается последовательность из четырех байтов. Четырехбайтная последовательность интерпретируется таким образом, что эти байты воспринимаются устройством как командные, а последний байт — как программная команда установки устройства в исходное положение. Команды в ячейках памяти 001C H — 0027 H на рис. 9.13 вызывают пересылку этой последовательности в устройство. Три нулевых байта обеспечивают готовность устройства к приему байта программной команды из ячейки памяти 40 H, обеспечивающей установку устройства в исходное положение.

Как мы видели в гл. 8, устройство связного интерфейса воспринимает байт, принятый вслед за программным или аппаратным сбросом, как байт режима. Поэтому следующий байт, посланный в устройство, интерпретируется как байт режима. Байт режима посылается в устройство командами, находящимися в ячейках памяти 0028 H — 002B H. Он устанавливает устройство в режим асинхронных связей, устанавливает коэффициент передачи равным шестнадцати, определяет число битов в каждом байте данных равным восьми без проверки на четность, определяет один стартовый бит и один бит останова. Следующий байт, посылаемый в устройство, интерпретируется как командный байт и посылается командами в ячейках памяти 002C H—002F H. Он приводит устройство в готовность к приему и передаче последовательных данных и устанавливает в исходное положение сигналы готовности к приему данных (DTR/) и готовности к пересылке (RTS/). Этим завершается инициализация устройства таймера Intel 8253 и устройства связного интерфейса Intel 8251 для тестирования системы.

Процедурой ЧТЕНИЯ СИМВОЛА предусмотрены ожидание получения устройством связного интерфейса очередного символа, считывание символа в центральный процессор из порта ввода, подключенного к устройству связного интерфейса, хранение символа в ячейке памяти CHAR1 и передача символа в качестве выходного параметра процедуре ИСПОЛНЕНИЯ

ТЕСТИРОВАНИЯ. Цикл в ячейках памяти 0031 H—0037 H включает считывание байта состояния из устройства связного интерфейса и проверку значения второго младшего бита в байте состояния, указывающего, получен ли символ. Если бит указывает, что символ принят, цикл завершается и выполняется команда ввода, которая считывает символ в центральный процессор. На этом шаге устройство связного интерфейса подготавливается также к приему следующего символа.

Процедура ВЫВОДА СИМВОЛА принимает символ от процедуры ИСПОЛНЕНИЕ ТЕСТИРОВАНИЯ в качестве входного параметра, хранит символ в ячейке памяти CHAR2, ожидает, пока не завершится передача предыдущего символа, пересланного в порт вывода, подключенный к устройству связного интерфейса, и пересылает новый символ в порт вывода для передачи. Цикл, подобный тому, что был описан в процедуре ЧТЕНИЯ СИМВОЛА, используется для считывания байта состояния из устройства для определения его готовности к приему нового данного. Когда байт состояния указывает, что передача разрешена, символ пересылается в порт вывода, подключенный к устройству связного интерфейса. Это выполняется с помощью команд в ячейках памяти 004B H — 004F H, показанных на рис. 9.15.

Теперь, после того как мы описали отладочную конфигурацию модуля связного интерфейса и программы для тренировки модуля, мы обсудим шаги, используемые во время отладки. В общих чертах мы должны проверить, что модуль правильно инициализируется и что последовательные цифровые данные принимаются и передаются в соответствии со спецификациями для модуля. Во время отладки микрокомпьютерный анализатор или внутрисхемный эмулятор используются для управления программами и для отображения результатов, полученных во время выполнения программ. Адреса ячеек памяти, содержащиеся в командах на рис. 9.12—9.15, используются для обеспечения средств отладки информацией, которая указывает, где начать выполнение (начальные точки) и где остановить выполнение (точки прерываний). Выходы микрокомпьютерного анализатора и внутрисхемного эмулятора могут быть использованы для синхронизации или запуска осциллографа или логического анализатора, подключенного к модулю. Эта конфигурация позволяет отобразить внутренние сигналы в модуле синхронно с выполнением программ.

Программно-управляемая динамическая отладка устройства связного интерфейса включает следующие шаги:

1. Проверить вызов процедуры ИНИЦИАЛИЗАЦИИ АППАРАТУРЫ процедурой ИСПОЛНЕНИЯ ТЕСТИРОВАНИЯ.

2. Проверить правильность пересылки последовательности байтов в порт вывода процедурой ИНИЦИАЛИЗАЦИИ АППАРАТУРЫ.

3. Используя осциллограф, прозерить правильность сигналов синхронизации, генерируемых устройством таймера после его инициализации.

4. Убедиться в том, что процедура ЧТЕНИЯ СИМВОЛА вызывается и что микрокомпьютер многократно считывает байт состояния из устройства интерфейса связей и проверяет значение бита в байте состояния, который указывает, принят ли символ устройством.

5. Нажимая клавиши терминала или другого входного устройства, работающего в стандарте RS-232-C, проверить значение бита, указывающего, что символ принят, и убедиться, что символьный код ASCII, соответствующий нажатой клавише, считывается центральным процессором и помещается в ячейку памяти, обозначенную именем CHAR в процедуре ИСПОЛНЕНИЯ.

6. Убедиться, что процедура ВЫВОДА СИМВОЛА вызывается и что ее входным параметром является тот же символьный код ASCII, что был принят на шаге 5.

7. Убедиться, что байт состояния, считываемый из устройства связного интерфейса, указывает, что устройство готово к приему символа до того, как символ послан в устройство. Проверить также правильность символьного кода ASCII, пересылаемого в порт вывода. На экране дисплея терминала должен появиться символ, соответствующий нажатой клавише на шаге 5.

8. Повторить шаги 4—7 для других символов.

В этом разделе мы использовали устройство связного интерфейса в качестве примера для демонстрации того, как организовать программно-управляемую динамическую отладку аппаратного модуля. Рассмотренные нами отладочная конфигурация и программные процедуры являются характерными для проведения отладки других подобных модулей. Проведение программно-управляемой динамической отладки является особенно важной для систем, содержащих интеллектуальные периферийные устройства. Как было ранее упомянуто, обнаруженные во время модульной отладки ошибки корректируются значительно проще и с меньшими затратами, чем на этапе объединения системы.

9.5. Инструментальные средства, используемые для разработки и отладки аппаратуры

В этом разделе мы рассмотрим приборы, используемые во время автономной статической, автономной динамической и

программно-управляемой динамической фаз отладки аппаратных средств. Из-за большой сложности микрокомпьютерных систем для их отладки требуются более мощные средства, чем при отладке традиционной электронной аппаратуры. Необходимость таких средств возникает, в частности, оттого, что микрокомпьютерные системы ориентированы на использование системной шины. Как мы уже видели, сигнальные линии шины часто работают в режиме разделения времени, т. е. по одним и тем же сигнальным линиям в разные времена посылаются различные типы информации. Поэтому приборы, используемые для отладки, должны обеспечивать возможность выборочного исследования информации шины.

Далее мы рассмотрим наиболее важные свойства некоторых приборов, используемых во время отладки аппаратуры, и опишем принципы их действия. Этими приборами являются цифровой вольтметр, осциллограф, имитатор статических сигналов, импульсный генератор, логический анализатор, программируемый логический анализатор, микрокомпьютерный логический анализатор и внутрисхемный эмулятор.

Цифровой вольтметр. Цифровой вольтметр используется во время статической отладки для проверки уровней напряжения питания и измерения постоянных токовых характеристик сигналов. Вольтметр является лучшим инструментом для определения того, что, принимая логическое значение ЕДИНИЦА или НУЛЬ, данный сигнал находится в пределах допустимых значений. Одним из допустимых значений сигнала шины является уровень напряжения, который соответствует *неопределенному* логическому уровню, если этот сигнал формируется *тристабильными* буферами, находящимися одновременно в неактивном состоянии. Это неопределенное логическое состояние может продолжаться до тех пор, пока один из тристабильных формирователей шины не установит сигнал в состояние либо логического НУЛЯ, либо логической ЕДИНИЦЫ. Однако неопределенный логический уровень может также указывать на наличие ошибки. Это может случаться, если формирователь сигнала чрезмерно загружен или если сигнальная линия является неуправляемой. В последнем случае вольтметр покажет напряжение смещения относительно плавающей входной линии. Большинство цифровых вольтметров позволяет также измерить сопротивление и ток. Так, вольтметр может быть использован для проверки электропроводности и измерения тока части модуля в зависимости от напряжения.

Осциллограф. Оециллограф является важным инструментом для динамической отладки. Он наиболее полезен при измерении переменных токовых характеристик логических сигналов. Осциллограф может быть использован в качестве автономного

инструмента для наблюдения установившихся сигналов внутри модуля при проверке таких характеристик, как время нарастания и убывания сигнала и частота сигнала. Его полезность повышается во время других фаз динамической отладки, если используется сигнал запуска от других устройств для синхронизации осциллографа с программными средствами.

Как указывалось ранее, во время динамической отладки необходимо наблюдать за сигналами шины в течение определенных интервалов времени, соответствующих конкретным отлаживаемым функциям. С целью выявления и отображения процессов, происходящих в шине только в течение требуемого временного интервала, используется логический анализатор или внутрисхемный эмулятор, которые обеспечивают программируемый сигнал запуска для осциллографа. Во время работы логический анализатор (или внутрисхемный эмулятор) ждет, пока не обнаружит требуемое событие, и как только это случается, вырабатывает импульс для запуска осциллографа, обеспечивая таким образом отображение только тех сигналов, которые представляют интерес. Для событий, имеющих малую длительность или редко встречающихся, необходим аналоговый или цифровой *запоминающий* осциллограф, который позволяет рассмотреть сигналы детально. Если наблюдаемые события являются программно-управляемыми, можно создать цикл ВЫПОЛНЯТЬ НЕПРЕРЫВНО... КОНЕЦ, который содержит программы для генерации события. При выполнении этого цикла событие рассматривается с помощью обычного осциллографа.

Имитатор статических сигналов [1]). Имитатор статических сигналов является полезным средством во время как статической, так и динамической отладки. С его помощью можно подготовить входные тесты для модуля, который в конечной системе будет работать под управлением другого модуля. Схема имитатора статических сигналов показана на рис. 9.17. Во время работы имитатор статических сигналов подключается к адресным и управляющим линиям и линиям данных тестируемого модуля. Следует заметить, что, если модуль подключен к шине, может понадобиться обеспечить подключение не менее тридцати двух входов модуля. Имитатор статических сигналов содержит также набор дискретных светоизлучающих диодов (СИД), которые показывают состояние каждого переключаемого разряда. Как показано на рисунке, в нем используются тристабильные формирователи, так что он может быть

[1]) Функции имитатора статических сигналов подобны функциям устройства тестирования посредством статических сигналов, описанного в кн. Дж. Коффрон, Технические средства микропроцессорных систем. — М.: Мир, 1983. Там же подробно описана методика тестирования посредством статических сигналов. — *Прим. перев.*

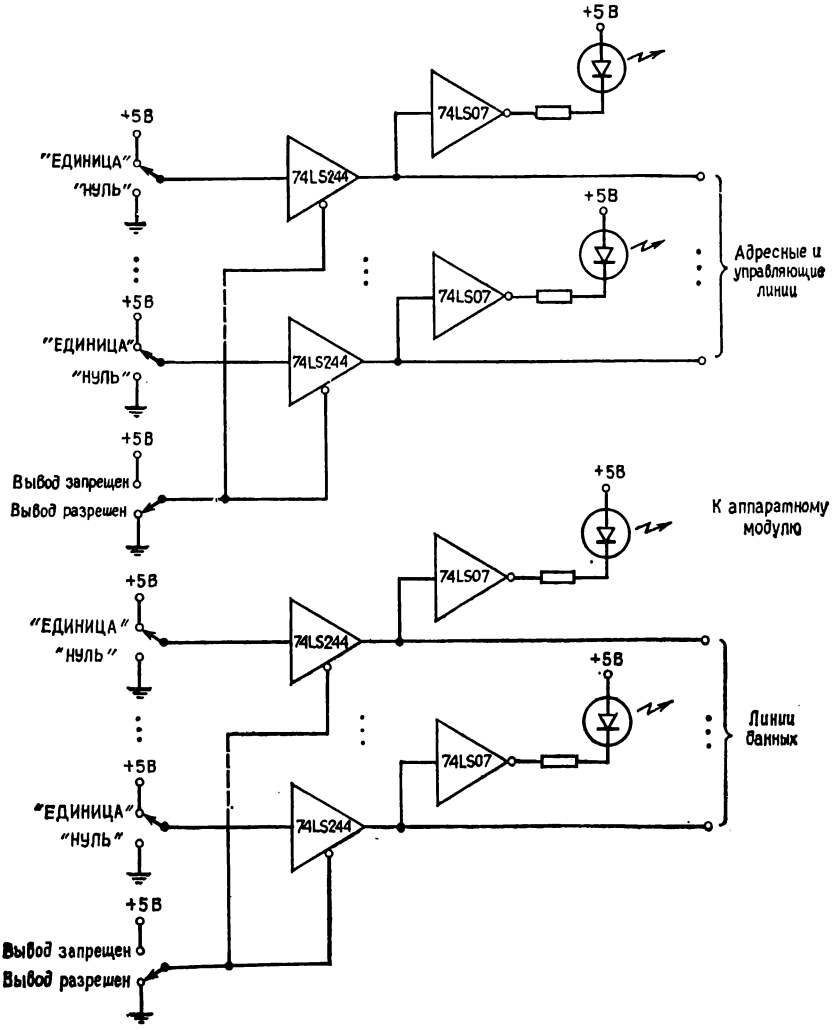


Рис. 9.17. Схема имитатора статических сигналов.

подключен к выходным линиям соответствующим способом. Имитатор статических сигналов используется также для обеспечения статических входов в модуль во время динамической отладки.

Импульсный генератор. Импульсный генератор используется во время автономной динамической отладки для обеспечения синхронизирующих и управляющих сигналов, вырабатываемых

другим модулем в законченной системе. Импульсный генератор позволяет осуществлять динамическое управление модулем в автономной конфигурации для проверки согласования во времени. Если для обеспечения входа модуля используется обычный звуковой генератор, следует подключить его через вентиль того же типа, который будет использоваться в конечной схеме, с целью обеспечения правильного согласования импеданса.

Логический анализатор. Логический анализатор является интеллектуальным устройством накопления цифровых данных и мощным инструментальным средством для отладки аппаратных модулей как во время статической, так и во время динамической отладки. Логические анализаторы существуют в различных видах и обладают множеством возможностей. Логический анализатор может быть в виде съемного дополнительного устройства для осциллографа, в виде автономного устройства, использующего устройство отображения на ЭЛТ или СИД, или в виде дополнения к внутрисхемному эмулятору, являющемуся частью микрокомпьютерной системы разработки. Хотя мы делаем только краткий обзор возможностей логического анализатора и принципов его работы, мы подчеркиваем, что он является чрезвычайно полезным инструментом для отладки аппаратных модулей микрокомпьютерных систем. Во время работы к логическому анализатору может быть подключено до тридцати двух цифровых сигналов аппаратного модуля, состояние которых затем отображается в зависимости от наличия определенного условия запуска. Условие запуска обычно требует, чтобы некоторые из этих сигналов находились в определенных состояниях. Как только условие запуска выполняется, включается устройство отображения, и состояния сигналов отображаются для каждого последующего периода времени. С целью синхронизации сигнал из тестируемого аппаратного модуля вводится в логический анализатор, принуждая его работать в режиме выборки данных. Некоторые логические анализаторы позволяют пользователю определять для выборки только определенные состояния сигналов при выполнении условия запуска. Многие из них могут также измерять и регистрировать абсолютные промежутки времени между двумя состояниями сигнала. Последнее свойство является особенно полезным во время динамической отладки, поскольку оно позволяет проверить синхронизацию сигналов. Логический анализатор является лучшим средством для исследования сигналов шины. Поскольку он имеет возможность запускать выдачу при наступлении определенных событий, он позволяет делать выборочное извлечение из непрерывного потока информации в шине.

Программируемый логический анализатор. Программируемый логический анализатор обеспечивает выполнение основных

функций, рассмотренных выше. Он отличается от обычного логического анализатора тем, что спецификации условий запуска и режимы выдачи могут быть автоматически загружены в логический анализатор не с клавиатуры, а непосредственно из компьютера. Эта возможность является особенно полезной, если тест, имеющий относительно сложную спецификацию, должен повторяться много раз. Кроме того, если одна и та же спецификация теста должна использоваться для большого количества модулей, программируемый логический анализатор обеспечивает возможность выполнения одного и того же теста для каждого модуля.

Микрокомпьютерный логический анализатор. Микрокомпьютерный логический анализатор извлекает данные путем прямого подключения к центральному процессору микрокомпьютера. Обычно его подключение к конкретному центральному процессору, например Intel 8085, обеспечивается с помощью специального *персонального* модуля. Так как характеристики центрального процессора, к которому подключается микрокомпьютерный анализатор, известны, он может обеспечить программируемую выдачу контролируемых сигналов. Он конвертирует эти сигналы в команды машинного языка и выдает их в виде команд на языке ассемблера. Это позволяет нам проследить за выполнением программ центральным процессором. Выдача *ретранслированных* программ является особенно полезной во время программно-управляемой динамической отладки, поскольку это позволяет нам одновременно наблюдать события и в аппаратуре, и в программе.

Микрокомпьютерный логический анализатор может содержать вспомогательный зонд, который может быть подключен к источнику внутренних сигналов в аппаратном модуле. Состояния этих аппаратных сигналов могут быть затем выданы наряду с ретранслированными командами центрального процессора. Соответствующее условие запуска для микрокомпьютерного логического анализатора может быть определено как адрес в адресной шине центрального процессора, как данные в шине данных центрального процессора или как комбинация состояний сигналов вспомогательного зонда. Такая гибкость позволяет пользователю настраивать микрокомпьютерный логический анализатор на выдачу любого из программных или аппаратных событий. На сегодняшний день микрокомпьютерный логический анализатор является, вероятно, наиболее эффективным инструментом для аппаратного объединения широкого круга приложений. Однако, поскольку микрокомпьютеры становятся более мощными, становится все труднее создать инструментальное средство, которое может автоматически выполнять операцию ретрансляции и оставаться сравнительно дешевым.

Так, в тот момент, когда пишутся эти строки, легкодоступными являются микрокомпьютерные логические анализаторы для работы с 8-разрядными микрокомпьютерами, но недоступны анализаторы для работы с 16-разрядными микрокомпьютерами. Поэтому внутрисхемный эмулятор, рассматриваемый далее, представляет хорошее альтернативное средство для использования при отладке систем, основанных на использовании 16-разрядных микрокомпьютеров.

Внутрисхемный эмулятор. Внутрисхемный эмулятор был введен в гл. 7 и будет еще обсуждаться в гл. 10 для иллюстрации его использования во время объединения системы. В этой главе мы обсудим его применение для отладки и объединения аппаратуры. Внутрисхемный эмулятор выполняет многие из функций, которые выполняет микрокомпьютерный логический анализатор во время программно-управляемой динамической отладки. Он обеспечивает пользователя средствами запуска и прекращения выполнения программ, для исследования значений данных до и после выполнения программного сегмента, для исследования значений данных, считываемых из порта ввода или записываемых в порт вывода, и для вывода команд на языке ассемблера, соответствующем выполняемым командам машинного языка. Некоторые внутрисхемные эмуляторы имеют зонд данных, который может быть использован для сбора данных во время выполнения программ. Однако большинство внутрисхемных эмуляторов не обеспечивают таких гибких и общих возможностей, какими обладают логические анализаторы. Если внутрисхемный эмулятор не обеспечивает адекватных возможностей сбора данных, вместе с ним может быть использован внешний логический анализатор. В этом случае внутрисхемный эмулятор управляет выполнением программ и выдает выполняемые команды, а логический анализатор выдает состояния требуемых внутренних аппаратных сигналов. В этом режиме работы логический анализатор синхронизируется прямо или косвенно с внутрисхемным эмулятором. При прямой синхронизации внутрисхемный эмулятор обеспечивает сигнал запуска логического анализатора, позволяя ему начать накопление и выдачу данных. При косвенной синхронизации логический анализатор может быть запущен событием, которое должно быть обнаружено в аппаратуре и которое непосредственно относится к выполнению программы. Например, если для запуска логического анализатора служит устройство выбора линии, выдача синхронизируется с выполнением команды, которая вызывает пересылку информации в выбранное устройство или чтение из выбранного устройства. Как только наступает момент синхронизации, следующие за этим команды выдаются внутрисхемным эмулятором, а соот-

ветствующие состояния сигналов — логическим анализатором. Поскольку обе выдачи относятся к одному и тому же моменту времени, они могут соответствующим образом сравниваться и анализироваться.

9.6. Шестой уровень документации

Шестой уровень документации содержит план объединения и отладки аппаратных средств. Как и на любом другом этапе цикла проектирования системы, выполняемая работа должна быть соответствующим образом описана. Для отладки аппаратных средств необходимо разработать план системной отладки, которого должны придерживаться как члены коллектива проектировщиков во время отладки первоначального образца, так и персонал, в чьи обязанности входит отладка систем, запущенных в производство, а также сопровождение и обслуживание готовых систем после их поставки.

План отладки каждого модуля должен быть написан тем, кто отвечает за отладку данного модуля. Хороший план отладки должен содержать пять обязательных компонентов, которые мы и рассмотрим.

Отлаживаемые функции модуля. В этом разделе плана отладки описываются функции модуля, которые должны тестироваться до объединения системы. По большей части эти тесты выполняются автономно. Однако возможно, что некоторые функции лучше отлаживать с другими аппаратными модулями, и поэтому отладка должна выполняться во время объединения системы. Обычно в эту категорию попадают сильно взаимосвязанные функции модуля.

Методика отладки и диаграмма конфигурации. Этот раздел включает описание общей методики тестирования и высокоуровневую блочную диаграмму конфигурации отладки каждого модуля. Диаграмма должна показывать, как к тестируемому модулю подключается отладочное оборудование и другие ранее отлаженные или одновременно тестируемые аппаратные модули.

Требуемое отладочное оборудование. Эта часть плана отладки состоит из списка отладочного оборудования, требуемого для отладки каждого модуля. Этот список является важным при распределении оборудования, когда одновременно тестируются несколько модулей. Он может также быть использован на ранних стадиях проектирования для определения необходимости дополнительных закупок или аренды отладочного оборудования.

Требуемые программные средства. Этот раздел включает список программных модулей, требуемых во время отладки

аппаратуры. Эти модули состоят из процедур, являющихся частью проектируемого программного обеспечения, а также из процедур, специально разработанных для отладки аппаратуры. Как указывалось ранее, необходимо предусмотреть процедуру ИСПОЛНЕНИЯ ТЕСТИРОВАНИЯ для вызова этих процедур в цикле с целью тренировки отлаживаемых аппаратных функций.

Детальная последовательность тестирования. Этот раздел описывает последовательность каждого тестирования шаг за шагом. Каждый шаг должен включать достаточное количество деталей для того, чтобы можно было проводить тестирование, не нуждаясь в дополнительной информации.

На этом мы заканчиваем обсуждение вопросов отладки и объединения аппаратных средств. В гл. 10 мы рассмотрим, как аппаратные и программные модули, которые отлаживались отдельно, объединяются во время сборки системы.

9.7. Упражнения

9.1. Разработайте план объединения аппаратуры для телевизионного приемника с встроенным микрокомпьютером (см. упражнения гл. 2—5).

9.2. Разработайте план объединения аппаратуры для устройства управления уличным светофором с встроенным микрокомпьютером (см. упражнения гл. 2—5).

9.3. Разработайте план объединения аппаратуры для электронного спортивного табло с встроенным микрокомпьютером (см. упражнения гл. 2—5).

9.4. Разработайте план объединения аппаратуры для терминала розничной торговли (см. упражнения гл. 2—5).

Глава 10

Объединение и оценка системы

Мы теперь знаем, как проектируются, конструируются и объединяются программные и аппаратные модули и подсистемы. К настоящему моменту все модули работоспособны. Аппаратные модули объединены, как и большинство программных модулей. Нам осталось встроить программное обеспечение в аппаратуру и объединить обе части в законченную работающую систему. Оценкой законченной системы с точки зрения удовлетворения требований пользователей и функциональной спецификации завершается задача построения микрокомпьютерной системы.

10.1. Объединение программного обеспечения с аппаратурой

В гл. 7 мы описали план объединения и сконцентрировали свое внимание на фазах объединения программного обеспечения. Фазы объединения аппаратуры были рассмотрены в гл.9 вместе с планом отладки аппаратуры. Теперь рассмотрим оставшуюся часть плана объединения, который содержит фазы объединения системы и включает следующие вопросы:

- Встраивание программного обеспечения в аппаратуру и проверка правильности работы встроенных программ. На ранних фазах объединения программного обеспечения для проверки функционирования программ были использованы возможности внутрисхемного эмулятора как средства разработки микрокомпьютерных систем, поскольку аппаратные средства были еще не отлажены.
- Объединение программных модулей, для которых требуется работоспособность аппаратных модулей ВВОДА, ВЫВОДА и ПРЕРЫВАНИЙ.

Встраивание программного обеспечения в аппаратуру может быть выполнено одним из следующих способов. Первый способ использует возможности внутрисхемного эмулятора встраивать программные модули в аппаратуру по одному, проверяя каждый встроенный модуль. Во втором способе все программные модули встраиваются в аппаратуру одновременно, а их работа проверяется путем использования возможностей трассировщика

и других средств, таких, как микрокомпьютерный анализатор. В любом случае после того, как программы будут встроены в аппаратуру, можно начинать отладку модулей ВВОДА, ВЫВОДА и ПРЕРЫВАНИЙ.

Модуль ВВОДА проверяют, используя процедуры нижнего уровня, которые при взаимодействии с аппаратурой вырабатывают определенные входные величины, а также исследуя значения внутренних системных параметров, зависящих от входных величин. Модули нижнего уровня являются работоспособными, поскольку они уже использовались во время отладки аппаратуры. Для проверки модуля ВЫВОДА необходимо предусмотреть такие входные величины, в зависимости от которых на выходе можно получить заранее предсказанные значения. После этого исследуются выходы системы либо непосредственно, либо путем наблюдения за выполнением некоторых действий, управляемых выходами.

Модуль ПРЕРЫВАНИЙ проверяют, выполняя программу, которая не содержит прерываний, и вызывая при этом аппаратные прерывания. После того как система прерываний проверена, можно использовать естественные прерывания для окончательной проверки работы модуля ПРЕРЫВАНИЙ.

10.2. Внутрисхемный эмулятор

Внутрисхемный эмулятор был описан в гл. 7, так как в нем предусмотрена возможность символической адресации памяти во время объединения программного обеспечения. Однако внутрисхемный эмулятор имеет и другие возможности, которые могут быть использованы во время объединения системы для постепенного встраивания программного обеспечения в разрабатываемую аппаратную систему. При использовании внутрисхемного эмулятора программное обеспечение перемещается из системы разработки микрокомпьютера в разрабатываемую аппаратную систему по одному или по два модуля за один раз, а затем проверяются рабочие характеристики этих модулей. Рассмотрим подробнее, как это происходит.

На рис. 10.1 показано подключение внутрисхемного эмулятора во время объединения системы. Интегральная схема микропроцессора вынимается из своего панельного гнезда в разрабатываемой аппаратной системе и заменяется штепсельным разъемом кабеля внутрисхемного эмулятора. Первоначально в системе нет программируемых ПЗУ и панельные гнезда памяти пусты. Программное обеспечение загружается в память микрокомпьютерной системы разработки и выполняется микропроцессором внутрисхемного эмулятора с использованием аппаратуры, доступной внутрисхемному эмулятору через подклю-

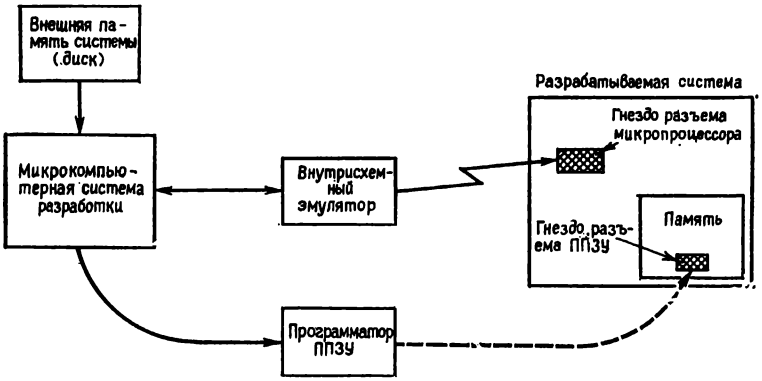


Рис. 10.1. Использование внутрисхемного эмулятора во время объединения системы.

ченный разъем кабеля. Затем после проверки правильности работы один или два программных модуля прожигаются в ППЗУ с помощью программатора ППЗУ, после чего ППЗУ вставляется в зарезервированное для него панельное гнездо памяти. Затем производится реконфигурация программного обеспечения внутрисхемного эмулятора в микрокомпьютерной системе разработки таким образом, что встроенные программные модули выполняются из аппаратных ППЗУ, в то время как остальная часть программных модулей выполняется из памяти микрокомпьютерной системы разработки. Эта операция возможна, потому что аппаратное ППЗУ доступно микропроцессору внутрисхемного эмулятора через разъем кабеля. Таким образом, внутрисхемный эмулятор может иметь выборочный доступ к памяти в любой системе. Поскольку объединение программного обеспечения уже закончено, прожигание ППЗУ и перемещение вследствие этого программного обеспечения в аппаратуру и проверка правильности его работы являются сравнительно простой задачей. Так же несложно постепенно прекратить использование ОЗУ микрокомпьютерной системы разработки и перейти к использованию аппаратной ОЗУ системы.

Одно из преимуществ использования внутрисхемного эмулятора для перемещения программных модулей в аппаратную систему по одному за один раз заключается в том, что это минимизирует число необходимых итераций по сравнению со случаем, когда прожигание ППЗУ производится во время отладки. Однако, если в нашем распоряжении нет внутрисхемного эмулятора, программное обеспечение может быть встроено за один шаг. Все ППЗУ прожигаются одновременно, а для проверки работы встроенного программного обеспечения ис-

пользуются трассировщик, монитор и микрокомпьютерный анализатор.

Как только программное обеспечение, которое было отлажено во время объединения программного обеспечения, начнет функционировать в аппаратуре, можно начинать отладку модулей ВВОДА, ВЫВОДА и ПРЕРЫВАНИЙ. В следующем разделе мы рассмотрим, как распределить программу между отдельными блоками ППЗУ с целью минимизации перепрограммирования ППЗУ при исправлении ошибок, обнаруженных на более поздних этапах объединения системы.

10.3. Встраивание программного обеспечения в аппаратуру

Использование программируемых ПЗУ определяет метод встраивания программного обеспечения в аппаратуру во время объединения системы. Модули прожигаются в ППЗУ по одному или по два. Затем ППЗУ встраиваются в аппаратуру и с помощью внутрисхемного эмулятора проверяются их рабочие характеристики. Далее, когда все ППЗУ прожжены, программное обеспечение полностью размещено в аппаратуре и работает без помощи микрокомпьютерной системы разработки, начинается заключительный этап объединения системы.

Как неоднократно указывалось, ошибки проектирования или программирования могут встретиться в любой момент процесса объединения. Как только ошибка обнаружена, она должна быть исправлена. Для этого требуется исправить или заново перепроектировать не менее одной процедуры, после чего исправленные процедуры транслируют, связи редактируют и вновь размещают программы. При исправлении процедуры существует большая вероятность того, что некоторые процедуры не будут размещены в тех же ячейках памяти, где они находились до исправления. ППЗУ, содержащие перемещаемые процедуры, должны быть перепрограммированы. Более того, если модуль, находящийся в ППЗУ, вызывает процедуру, которая переместилась, такое ППЗУ становится непригодным для работы и должно быть также перепрограммировано. Перепрограммирование или новое прожигание ППЗУ является дорогостоящей операцией. Поэтому мы рассмотрим метод, который сводит к минимуму вероятность перепрограммирования существующих ППЗУ при коррекции программ.

Метод заключается в использовании *вектора перехода* при редактировании связей для каждой процедуры, вызываемой процедурой из другого модуля. Рис. 10.2а и рис. 10.2б иллюстрируют два способа организации связей: стандартный и с помощью вектора перехода. Способ организации связей с помощью вектора перехода требует, чтобы каждый вызов был

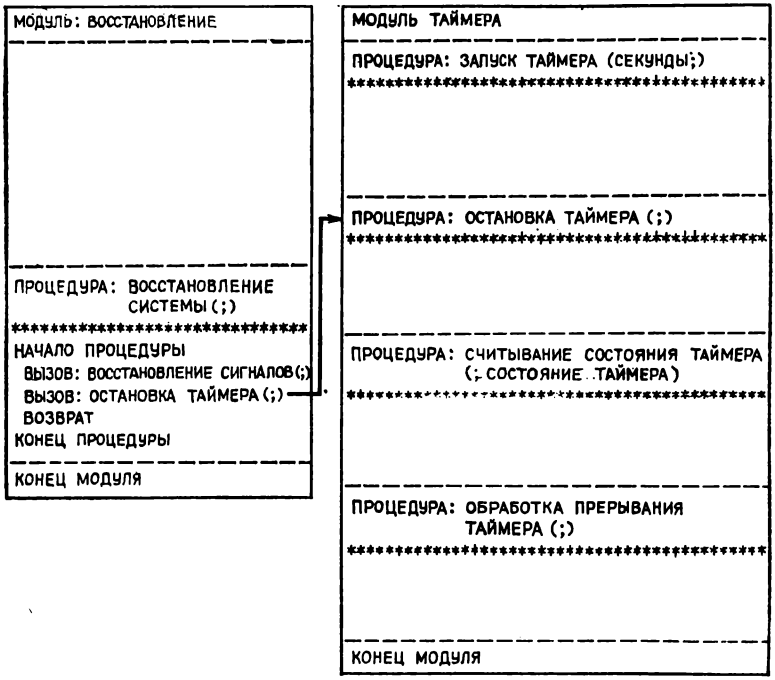


Рис. 10.2а. Стандартный способ организации связей при вызове процедур.

снабжен дополнительной операцией. Вместо непосредственного вызова процедуры вызывается *процедура вектора перехода*, которая реализуется с помощью единственной операции, подобной операции GOTO в PL/M и команде *безусловного перехода* в языке ассемблера. Эта операция вызывает выполнение фактической процедуры так, как если бы она вызывалась непосредственно. Векторы перехода размещаются в отдельном модуле, называемом *модулем вектора перехода*, так что при конвертировании программ из описания на языке проектирования в язык программирования процедуры вектора перехода остаются сгруппированными вместе. Если имя модуля вектора перехода предшествует имени модуля процедуры в команде редактирования связей (link), это означает, что векторы переходов разместятся в ячейках ППЗУ с меньшими значениями адресов. В результате независимо от того, как будут модифицированы сами процедуры, связи между вызывающими процедурами из других модулей и векторами переходов меняться не будут. Как только модули связаны и размещены в ППЗУ,

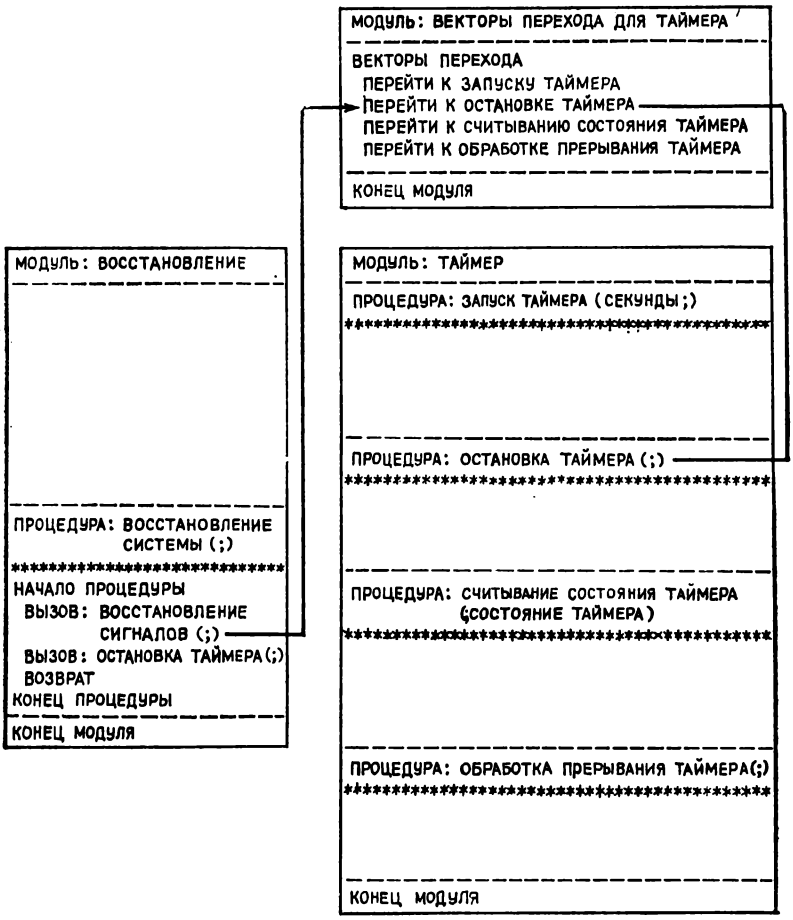


Рис. 10.2б. Способ организации связей при вызове процедур с помощью вектора перехода.

они должны быть связаны вместе с другими модулями системы для привязки всех межмодульных вызовов к соответствующим процедурам векторов переходов. Наконец, после того, как будет выполнен последний шаг, каждый комбинированный вектор перехода и модуль процедур могут быть прожжены в предназначенном для этого ППЗУ.

Хотя организация связей с помощью вектора перехода широко использовалась в пятидесятых и шестидесятых годах, в настоящее время она встречается редко, а редакторы связей для микрокомпьютеров вообще не обеспечивают такой возмож-

ности. Однако многие микрокомпьютерные языки высокого уровня позволяют пользователю реализовать связи с помощью вектора перехода программным путем. Если необходимо, это может быть также реализовано в программах на языке ассемблера. На рис. 10.3 показана реализация связей с помощью вектора перехода для модуля TIMER на языке PL/M микрокомпьютера Intel 8085. Для реализации каждой процедуры вектора перехода используется операция PL/M GOTO, которая эквивалентна команде *безусловного перехода* в языке ассемблера, а также процедура описания. Следует отметить, что процедурам векторов переходов назначены имена, а самим процедурам — набор вторичных имен. Вторичные имена описаны в модуле векторов переходов как *внешние метки*, так как PL/M не позволяет использовать в операции GOTO обращение к имени процедуры. Последовательность шагов при вызове процедуры STOP TIMER показана на рисунке стрелками.

Входные и выходные параметры, передаваемые между процедурами PL/M, не должны описываться в описании процедуры вектора перехода. Они должны описываться там, где используются, а именно в описании действительных процедур. Компилятор PL/M затем будет передавать эти параметры между вызывающей процедурой и фактически вызываемой процедурой в соответствии с необходимостью. Протокол вектора перехода для процедуры с одним входным параметром показан на рис. 10.3 в процедуре START TIMER (TIMER 1), а протокол для процедуры, которая возвращает один байт в качестве выходного параметра,— в процедуре READ TIMER STATE (TIMER 3).

До сих пор мы предполагали, что каждый программный модуль размещается в целом числе ППЗУ и что какая-то часть памяти в разумных пределах остается неиспользованной, обеспечивая возможность последующей коррекции ошибок. Однако размер памяти, требуемой для модуля, редко соответствует размеру ППЗУ. Часто оказывается необходимым комбинировать модули, особенно небольшие, для того, чтобы память использовалась рационально. Организация связей с помощью вектора переходов может быть использована только тогда, когда имеется возможность объединить несколько модулей. Чтобы встроить эти модули в одно ППЗУ, имена всех модулей вектора переходов, предназначенных для ППЗУ, должны предшествовать именам модулей фактических процедур в команде редактирования связей (link). В том случае, когда процедуры вектора переходов для комбинированного модуля будут размещены, они окажутся смежными в ППЗУ, как указывалось выше.

Редактор связей и загрузчик микрокомпьютеров Intel 8086 являются примерами средств, которые работают иначе, чем

```
/* МОДУЛЬ: ВЕКТОРЫ ПЕРЕХОДА ДЛЯ ТАЙМЕРА                    */
                    TIMER$TRTANSFER$VECTORS: DO;
/* ------------------------------------------------------- */
/* ВНЕШНИЕ ОПИСАНИЯ                                        */
                    DECLARE (TIMER1, TIMER2,TIMER3,TIMER4)
                    LABEL EXTERNAL;
/* ------------------------------------------------------- */
/* ВЕКТОРЫ ПЕРЕХОДА                                        */
/* ПЕРЕЙТИ К ЗАПУСКУ ТАЙМЕРА                               */
                    START$TIMER: PROCEDURE PUBLIC;
                      GOTO TIMER1;
                    END START$TIMER;
/* ПЕРЕЙТИ К ОСТАНОВКЕ ТАЙМЕРА                             */
                    STOP$TIMER: PROCEDURE PUBLIC;
                      GOTO TIMER2;
                      END STOP$TIMER;
/* ПЕРЕЙТИ К СЧИТЫВАНИЮ СОСТОЯНИЯ ТАЙМЕРА                  */
                      READ$TIMER$STATE: PROCEDURE PUBLIC;
                      GOTO TIMER3;
                      END READ$TIMER$STATE;
/* ПЕРЕЙТИ К ОБРАБОТКЕ ПРЕРЫВАНИЯ ТАЙМЕРА                  */
                    PROCESS$TIMER$INTERRUPT: PROCEDURE PUBLIC;
                      GOTO TIMER4;
                      END PROCESS$TIMER$INTERRUPT;
/* ------------------------------------------------------- */
/* КОНЕЦ МОДУЛЯ                                            */
                    END TIMER$TRANSFER$VECTORS
```

```
/* МОДУЛЬ: ТАЙМЕР                                          */
                    TIMER: DO;
/* ------------------------------------------------------- */
/* ПРОЦЕДУРА: ЗАПУСК ТАЙМЕРА (СЕКУНДЫ)                     */
                    TIMER1: PROCEDURE (SECONDS) PUBLIC;
                      DECLARE SECONDS BYTE;
/*******************************************************  */
/*                                                         */
/*                                                         */
/*                                                         */
/* ------------------------------------------------------- */
/* ПРОЦЕДУРА: ОСТАНОВКА ТАЙМЕРА (;)                        */
                    TIMER2: PROCEDURE PUBLIC;
/*******************************************************  */
/*                                                         */
/*                                                         */
/*                                                         */
/* ------------------------------------------------------- */
/* ПРОЦЕДУРА:СЧИТЫВАНИЕ СОСТОЯНИЯ ТАЙМЕРА (;СОСТОЯНИЕ ТАЙМЕРА) */
                    TIMER3: PROCEDURE BYTE PUBLIC;
                      DECLARE TIMER$STATE BYTE;
/*******************************************************  */
/*                                                         */
/*                                                         */
/*                                                         */
/* ------------------------------------------------------- */
/* ПРОЦЕДУРА: ОБРАБОТКА ПРЕРЫВАНИЯ ТАЙМЕРА (;)             */
                    TIMER4: POCEDURE PUBLIC;
/*******************************************************  */
/*                                                         */
/*                                                         */
/*                                                         */
/* ------------------------------------------------------- */
/* КОНЕЦ МОДУЛЯ                                            */
                    END TIMER;
```

Рис. 10.3. Реализация связей с помощью вектора перехода для PL/M.

описанные нами соответствующие средства микрокомпьютера Intel 8085. В Intel 8086 все модули связываются вместе одной операцией и затем связанные программы размещаются в ППЗУ за один шаг. Чтобы разместить модули по отдельным ППЗУ, перемещающий загрузчик предусматривает начальную ячейку памяти ППЗУ для каждого модуля или группы модулей. Когда используются модули векторов переходов, имена модулей в команде связи должны появляться в порядке, описанном для редактора связей Intel 8085. Следует заметить, что в этом случае модули векторов переходов должны быть переведены на язык ассемблера, поскольку PL/M микрокомпьютера Intel 8086 не позволяет непосредственно реализовать организацию связей с помощью вектора переходов.

10.4. Другие методы объединения

До сих пор мы предполагали, что большинство программных средств объединено, встроено в аппаратуру и вместе с оставшимися программами объединено с аппаратными средствами. Однако во многих микрокомпьютерных системах имеется много взаимосвязей между программными и аппаратными средствами. В таких системах часто более предпочтительным является комбинированный подход к объединению программных средств и системы.

Если выбран комбинированный подход, некоторые аппаратные средства должны быть работоспособными ранее других в рамках цикла проектирования системы. Оставшаяся часть аппаратных средств добавляется во время объединения по мере необходимости. План объединения должен определять, какие аппаратные и программные модули должны быть доступны в начале каждой фазы объединения и какие должны быть закончены в течение каждой фазы.

Первые две или три фазы объединения включают объединение только программных средств и часто завершаются с помощью тех средств, которые предоставляет микрокомпьютерная система разработки. Когда первоначальные фазы закончены, для подключения микрокомпьютерной системы разработки к аппаратуре используется внутрисхемный эмулятор. Программные модули, которые взаимодействуют с соответствующими аппаратными модулями, отлаживаются и объединяются. Объединенные программные модули прожигаются в ППЗУ и встраиваются в аппаратуру. Затем проверяются их рабочие характеристики. В течение оставшихся фаз объединения эти объединенные модули используются непосредственно, без моделирования их характеристик. На ранних этапах процесса объединения с системой может быть объединен трассировщик.

Он используется во время объединения вместе с другими средствами отладки.

Примером системы, в которой видны преимущества комбинированного подхода во время объединения, является система расчета за покупки. Большинство программ системы расчета за покупки непосредственно откликаются тем или иным способом на входные сигналы, возникающие при нажатии функциональных или числовых клавиш. Поэтому наличие работоспособных интерфейсных модулей клавиатуры на самых ранних стадиях процесса объединения системы является определенным преимуществом. В этом случае отладка будет заключаться в нажатии различных последовательностей клавиш и в регистрации откликов системы. Полезно также, если на ранних стадиях объединения будут отлажены интерфейсные модули устройства отображения, так как большинство ответов системы оператор получает через устройство алфавитно-цифрового отображения. В данном случае отладка системы проводится естественным образом. Более того, при этом гораздо раньше становится возможным оценить поведение системы с точки зрения учета человеческих факторов. Это позволит по мере необходимости сравнительно легко вносить изменения в интерфейсные модули.

И в период объединения, и после завершения системы необходимо оценивать рабочие характеристики с точки зрения их соответствия как требованиям пользователей, так и функциональной спецификации. Рассмотрим далее, как производится оценка рабочих характеристик системы.

10.5. Оценка системы

В течение всего цикла проектирования системы нашей целью являлось построение микрокомпьютерной системы, отвечающей требованиям пользователей и функциональной спецификации. Мы рассмотрели методику проектирования и конструирования программных и аппаратных модулей, а также способы объединения этих модулей в работающую систему. Мы также рассмотрели, как используются средства разработки программного обеспечения, экономящие время и усилия и обеспечивающие получение полной документации по завершении системы. Рассмотрим теперь, как оценить рабочие характеристики законченной системы, отвечающей требованиям пользователей.

Если система построена для отдельного заказчика, для нее может быть проведена серия *приемочных испытаний,* назначенных заказчиком. Во время этих испытаний *заказчик* определяет, соответствует ли работа системы его требованиям. Если система проходит приемочные испытания, заказчик принимает ее в эксплуатацию и осваивает, предполагая, что система ра-

ботает без ошибок. Таким образом, когда мы оцениваем рабочие характеристики системы, мы должны быть в значительной мере убеждены, что она пройдет приемочные испытания, как бы строги они ни были.

Хотя окончательная оценка системы проводится после того, как система построена и работает, готовиться к этому следует в течение всего цикла проектирования системы. Оценка системы и приемочные испытания начинаются в период просмотра требований пользователей, проходящего с участием заказчика. Затем, когда мы проводим просмотр функциональной спецификации и начинаем проектирование высшего уровня, процесс оценки системы продолжается. Во время отладки и объединения мы добиваемся выполнения таких условий, которые указывают на то, что модули работают правильно и по отдельности, и вместе. Таким образом, к моменту завершения системы большинство оценок уже проведено. Остается испытать систему как в искусственных, так и естественных условиях работы.

Одной из проблем, часто возникающих к концу цикла проектирования системы, является то, что система не может функционировать с требуемой скоростью, хотя при этом она работает правильно с функциональной точки зрения. В нашем случае одной из причин такого положения является то, что метод описания программ на языке проектирования делает упор на ясность и точность в ущерб скорости выполнения. Однако в большинстве систем реального времени только малая часть программного обеспечения является критичной по отношению к достижению высокой скорости функционирования системы. Следовательно, необходимо выявить и изолировать критичные модули. Затем, для повышения скорости их работы, эти модули либо проектируются заново, либо реализуются с помощью более эффективных операций языка программирования или, если необходимо, с помощью команд языка ассемблера. В крайнем случае микрокомпьютер заменяется на более быстродействующий. Использование языка проектирования и языка программирования высокого уровня (такого, как PL/M) делает сравнительно легкой замену одного микрокомпьютера другим с незначительной затратой усилий.

10.6. Сопровождение системы

В течение цикла проектирования системы мы стремились спроектировать и построить систему, не содержащую программных или аппаратных *дефектов* в момент поставки заказчику. Но как аппаратные, так и программные компоненты могут отказать во время функционирования. Причины этих двух типов отказов различны. Аппаратура отказывает потому, что физиче-

ские и электрические характеристики меняются со временем, вызывая ухудшение рабочих характеристик функционирования. Программы отказывают, если они плохо спроектированы или недостаточно отлажены. Всем этим вопросам в данной книге уделено достаточное внимание. Присущая некоторым системам сложность не позволяет проводить отладку программного обеспечения при всех мыслимых условиях их работы. Поэтому существует необходимость сопровождения системы заказчиком после ее внедрения.

Если обнаружена программная ошибка, необходимо прежде всего выявить и изолировать источник ошибки. Хорошая документация является неоценимым средством при выявлении и устранении программной ошибки в неправильно функционирующей системе. Методы самодокументирования, описанные нами, позволяют *ответственному за сопровождение программного обеспечения*, не участвовавшему в проектировании системы, *восстановить* систему. Такое средство, как трассировщик, может помочь сузить круг поиска до размеров конкретной процедуры или модуля. Как только выявлена ошибочная процедура, она может быть исправлена путем внесения соответствующих изменений в программу на языке проектирования или языке программирования. Впоследствии система должна быть заново тщательно отлажена, чтобы удостовериться в правильности ее работы и в том, что внесенные изменения не создали новых проблем.

После того как *ошибка* исправлена и проверены рабочие характеристики *восстановленной* системы, должны быть подготовлены новые ПЗУ для каждой готовой системы. Ранее описанные методы, минимизирующие число прожигаемых ППЗУ во время разработки системы, минимизируют также число блоков ПЗУ, забракованных и замененных, если программная *ошибка* обнаружена в поле ПЗУ.

Таким образом, рассмотренная нами методология системного проектирования оказывает глубокое влияние на каждую стадию разработки и сопровождения микрокомпьютерной системы. Она упрощает проектирование, конструирование, документацию и восстановление системы.

Итак, мы прошли по книге весь маршрут, о котором говорили в гл. 1. Далее мы рассмотрим несколько других приложений микрокомпьютеров с целью иллюстрации их широких возможностей.

10.7. Упражнения

10.1. Спроектируйте серию приемочных испытаний для телевизионного приемника с встроенным микрокомпьютером (см. упражнения в гл. 2—5).

10.2. Спроектируйте серию приемочных испытаний для устройства управления уличным светофором со встроенным микрокомпьютером (см. упражнения в гл. 2—5).

10.3. Спроектируйте серию приемочных испытаний для электронного спортивного табло (см. упражнения в гл. 2—5).

10.4. Спроектируйте серию приемочных испытаний для терминала розничной торговли (см. упражнения в гл. 2—5).

Примеры применения

Мы уже видели, что микрокомпьютер является гибким компонентом, когда он используется в системе охранной сигнализации или в терминале розничной торговли. Микрокомпьютерный компонент может быть использован во многих других приложениях, например таких, которые использовались в упражнениях. Ниже перечислены некоторые другие применения микрокомпьютера, для которых он является особенно полезным:

- Игры.
- *Интеллектуальные* терминалы.
- Системы управления автомобильным двигателем.
- Электробытовые товары.

В данной главе мы рассмотрим эти приложения с целью дальнейшей иллюстрации широких потенциальных возможностей микрокомпьютера, а также с целью ознакомления читателя с другими возможными концепциями. Читателю предлагается спроектировать какую-либо часть описываемых приложений. Используйте упражнения в конце главы в качестве примеров упрощенных версий реальных систем.

11.1. Игры

Телевизионные игры возникли как увлечение, и, как это часто случается с увлечениями, интерес к ним после начального ажиотажа быстро упал. Однако игры, основанные на использовании микрокомпьютера, имеют по сравнению с играми на электронных схемах гораздо большую гибкость, и поэтому они снова подогрели интерес покупателей к телевизионным играм и выдвинули их из категории просто развлечений. Большинство игр на основе микрокомпьютера используют экран на ЭЛТ. Кроме того, разработано еще несколько видов игр с использованием микрокомпьютера, таких, как шахматы, которые не используют экран. В этом разделе мы будем иметь дело только с самыми простыми играми на основе микрокомпьютера.

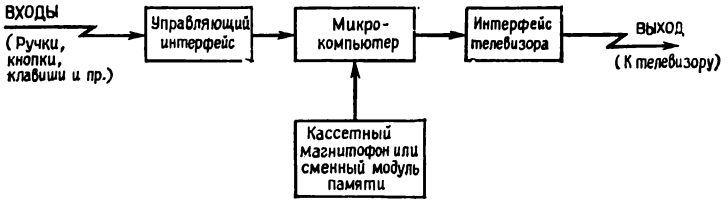


Рис. 11.1. Схема телевизионной игры с применением микрокомпьютера.

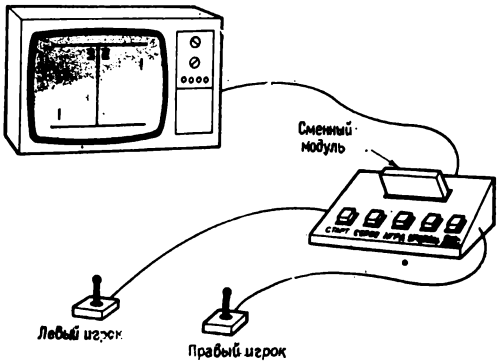


Рис. 11.2. Изображение экрана телевизора и панели управления для телевизионной игры.

На рис. 11.1 показана принципиальная схема телевизионной игры на основе микрокомпьютера. Программа для конкретной игры либо считывается в память микрокомпьютера с кассетной магнитной ленты, либо вводится из сменного модуля памяти. Система выводит на экран телевизора *игровое поле* и ждет команды СТАРТ, означающей, что игроки готовы начать игру (рис. 11.2). Как только клавиша СТАРТ нажата, игра начинается. *Подается мяч,* который движется через экран, а каждый игрок старается передвигать свою *ракетку* таким образом, чтобы *отбить* или отразить мяч, когда он направляется в ту половину *корта*, которая принадлежит игроку. Система ведет счет и высвечивает его цифрами на экране (рис. 11.2). Кнопка сброса позволяет в любой момент остановить и возобновить игру. Дополнительные клавиши (рис. 11.2) позволяют варьировать скорость мяча и другие игровые параметры, чтобы игра не надоела. Хотя другие игры, которые вводятся с магнитной ленты или из сменного модуля памяти, могут значительно отличаться, все они требуют, чтобы на экране телевизора появлялось движущееся изображение. Используя в качестве примера игру с *мячом* и *ракетками*, рассмотрим, как это происходит с помощью микрокомпьютера.

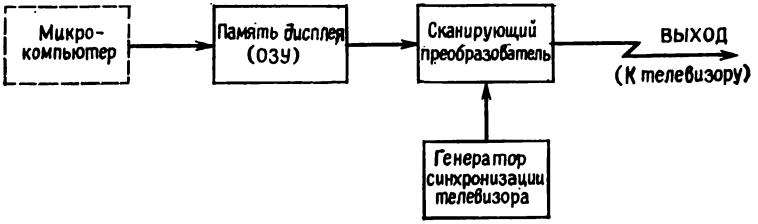


Рис. 11.3. Интерфейс телевизора.

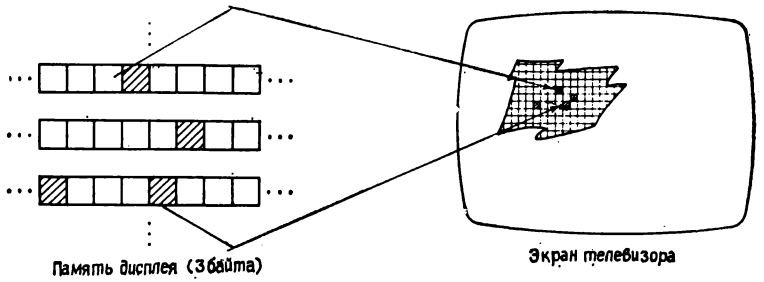


Рис. 11.4. Соответствие между четырьмя битами в памяти дисплея и четырьмя элементами на экране телевизора.

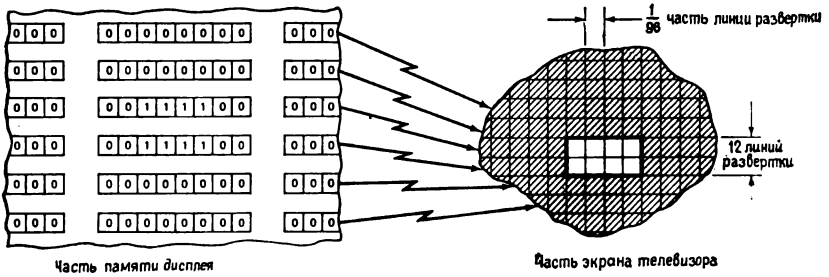


Рис. 11.5. Изображение светящегося прямоугольника на экране телевизора.

На рис. 11.3 показана часть системы, которая включает микрокомпьютер и интерфейс телевизора. Экран телевизора представляется в виде системы небольших квадратов, каждому из которых соответствует один разряд в памяти дисплея, как показано на рис. 11.4 для четырех элементов. Поскольку нет необходимости в слишком большой детализации, число используемых элементов изображения обычно выбирается девяносто шесть на восемьдесят. Таким образом, в нашем примере для изображения всех элементов строки на экране телевизора тре-

буется девяносто шесть битов или двенадцать байтов в памяти дисплея. Кроме того, каждая двенадцатибайтная строка изображения на экране состоит из шести расположенных одна за другой линий развертки. Следовательно, для восьмидесяти строк изображения должно быть сгенерировано 480 линий развертки, что вполне достаточно для покрытия стандартного телевизионного экрана. Сканирующий преобразователь работает в режиме непрерывного поблочного обмена ПДП. Он извлекает каждый байт из памяти дисплея в момент прохождения сканирующей линией части экрана, которой соответствует данный байт. Сигнал на экране телевизора модулируется таким образом, что квадрат на экране высвечивается, если соответствующий бит в памяти дисплея равен ЕДИНИЦЕ, или остается темным, если бит равен НУЛЮ. Рис. 11.5 иллюстрирует эту концепцию для небольшого светящегося прямоугольника.

Когда микрокомпьютер обновляет информацию в памяти дисплея, светящаяся часть экрана меняет свое положение, что выглядит как движение. Рассмотрим эту концепцию, используя светящийся прямоугольник на рис. 11.5. Допустим, что прямоугольник изображает движущийся объект, который должен перемещаться горизонтально слева направо на расстояние, соответствующее 60 квадратам или пяти восьмым ширины экрана. Допустим также, что прямоугольник должен переместиться на это расстояние за одну секунду. Таким образом, в течение каждой одной шестидесятой секунды (16,7 мс) прямоугольник должен переместиться на расстояние одного квадрата вправо. Микрокомпьютер через определенный промежуток времени, равный одной шестидесятой секунды, выполняет прерывания. Во время каждого прерывания биты, изображающие прямоугольник, сдвигаются в памяти дисплея на один разряд *вправо*. Для примера на рис. 11.5 это движение выполняется путем сдвига битов в каждой строке прямоугольника, как показано ниже:

```
... 00111100 00000000... Первоначальное положение
... 00011110 00000000... После первого прерывания
... 00001111 00000000... После второго прерывания
... 00000111 10000000... После третьего прерывания
... 00000011 11000000... После четвертого прерывания
```

Этот пример иллюстрирует четыре последовательных прерывания.

Движение в произвольном направлении достигается при помощи независимой обработки горизонтального и вертикального направлений, путем сдвига битов для горизонтального перемещения, описанного выше, и сдвига битов *от строки к строке* для вертикального перемещения. Синхронизация между микрокомпьютером и дисплеем не нужна, случайные искажения, возни-

кающие вследствие считывания сканирующим пребразователем памяти дисплея в то время, когда она изменяется, незаметны благодаря инерционности зрительного восприятия человека.

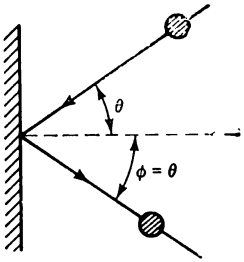


Рис. 11.6. Отражение мяча от стенки в телевизионной игре.

В действующей системе несколько объектов, например *мяч* и две *ракетки*, могут одновременно находиться в движении, а движущийся объект может *сталкиваться* с неподвижным или с другим движущимся объектом. Объекты должны рассматриваться как отдельные сущности, чтобы не допустить слияния двух движущихся объектов, в результате чего они могут стать неразличимыми в памяти. Таким образом, информация в памяти микрокомпьютера о каждом движущемся объекте должна храниться и обрабатываться отдельно. Эта информация может быть объединена перед загрузкой в память дисплея. Для этой цели может быть предложено много стратегий, однако мы не намерены обсуждать далее этот аспект телевизионных игр.

Игровая стратегия должна быть заключена в программном обеспечении, она представляет собой функцию правил игры. Однако если мяч ударяется о неподвижное препятствие, например о *стенку*, он должен отскочить в направлении, определяемом законом физики, как показано на рис. 11.6. В приведенном примере тригонометрические вычисления не нужны. Горизонтальная составляющая перемещения мяча изменяет направление на обратное, в то время как вертикальная остается неизменной. Поэтому после столкновения мяч продолжает движение с той же скоростью, но с горизонтальной составляющей, направленной в противоположную сторону. Аналогичные правила применяются для горизонтальных стенок. Если мяч сталкивается с ракеткой, новое направление движения зависит от того, находится ли ракетка в момент удара в покое или в движении. Если ракетка неподвижна, мяч отражается так же, как от стены. Если же ракетка движется, скорость мяча в вертикальном направлении также меняется. Например, если мяч и ракетка в момент столкновения имеют одинаковое направление движения, например вниз (рис. 11.7), вертикальная составляющая перемещения мяча после столкновения увеличивается. Это вызывает изменение направления движения мяча, как показано на рис. 11.7. На рисунке также показана стратегия отскока мяча в случае перемещения мяча и ракетки в противоположных направлениях, при этом вертикальная составляющая перемещения мяча уменьшается. В особом случае, когда вертикальная составляющая перемещения мяча перед столкновением невелика,

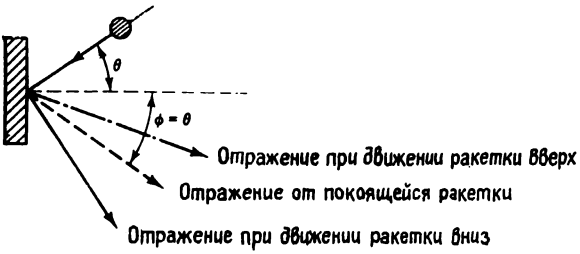


Рис. 11.7. Отражение мяча от движущейся ракетки.

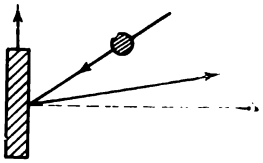


Рис. 11.8. Отражение мяча от ракетки, движущейся с большой скоростью.

а ракетка движется быстро в противоположном направлении, вертикальная составляющая перемещения мяча может изменить свое направление на обратное, как показано на рис. 11.8. Как указывалось выше, возможны также другие стратегии. Следует отметить, что наиболее интересные стратегии не обязательно требуют применения сложных программных алгоритмов. На самом деле, при хорошем проектировании *сложные* стратегии реализуются с помощью сравнительно простых алгоритмов.

Прежде чем покончить с телевизионными играми, рассмотрим кратко несколько непосредственно связанных с ними вопросов. Чтобы повысить интерес к игре, можно сопровождать каждый удар мяча звуковым сигналом. Для этой цели может быть использован простой *зуммер,* включающийся через интерфейс телевизора под управлением микрокомпьютера. Игра станет еще интересней, если будет меняться тональность сигнала и для каждого удара микрокомпьютер будет выбирать различные сигналы. Для выбора сигнала в зависимости от конкретных условий может быть разработана соответствующая стратегия. Например, сигнал низкого тона может включаться тогда, когда при встрече ракетки с мячом вертикальная составляющая перемещения ракетки почти отсутствует. Чем больше вертикальная составляющая перемещения ракетки, тем тональность сигнала во время удара становится выше. Аналогичные стратегии могут быть предложены для случая удара мяча о стену: здесь тональность сигнала может меняться в зависимости от угла и скорости перемещения мяча.

Мы предполагали, что каждой *строке* байтов в памяти дисплея соответствуют шесть линий развертки на экран телевизора. Однако коммерческие телевизоры имеют чересстрочную развертку: нечетные строки образуют один кадр развертки, а четные — другой. Поэтому каждой *строке* байтов в памяти дисплея соответствуют только три последовательные линии развертки.

Интерфейсная схема телевизионного дисплея выдает на экран дисплея шесть следующих подряд линий развертки для каждой строки в памяти дисплея.

Другим способом повышения интереса к телевизионной игре является использование цвета. Например, *игровое поле* может быть одного цвета, фон — другого, а мяч и ракетки — третьего. Для отображения цвета требуются дополнительные разряды в памяти дисплея для каждого элемента изображения. Если предположить, что для каждого элемента изображения предусмотрен один дополнительный разряд, объем памяти дисплея должен быть удвоен, а число цветов изображения может быть увеличено до четырех. Хотя использование только четырех цветов может показаться слишком бедным, часто этого оказывается достаточным: несколько элементов могут изображаться одним и тем же цветом без ущерба для зрительного восприятия. Например, один и тот же цвет может быть использован для мяча и изображения счета игры, поскольку мяч и счет появляются в разных частях экрана. В цветной системе сканирующий преобразователь должен модулировать носитель цвета в соответствии с мультиразрядным кодом каждого элемента изображения. Схемы, используемые для этого, сравнительно просты и незначительно увеличивают стоимость игры.

Гибкость микрокомпьютера позволяет проектировать и разыгрывать практически неограниченное число телевизионных игр. Нас может сдерживать только сложность программных алгоритмов, необходимых для реализации конкретной игровой стратегии. Можно даже изобрести такую *систему разработки*, которая позволит пользователю придумывать свои собственные игры. Для такой системы должен быть предусмотрен набор программных процедур, обеспечивающих для пользователя большое количество различных функций. Пользователь *определяет* правила игры, а система разработки объединяет процедуры в игру, которая может быть выполнена с помощью микрокомпьютера. Для такого *программирования* требуется хитроумная система разработки, позволяющая непосвященному пользователю *запрограммировать* систему. Дальнейшее обсуждение этого аспекта *системного проектирования* не входит в наши цели.

11.2. Интеллектуальные терминалы

Как указывалось в гл. 7, интеллектуальный терминал представляет собой устройство, с помощью которого оператор взаимосвязан с вычислительной системой. Терминал состоит из устройства ввода типа алфавитно-цифровой клавиатуры, подобной клавиатуре пишущей машинки, и устройства вывода в виде печатающего устройства или в виде экрана телевизора в

случае терминала на ЭЛТ. Многие терминалы сравнительно просты. Обмен с вычислительной системой ведется символами. При нажатии клавиши на клавиатуре соответствующий код символа пересылается в вычислительную машину. Как только символьный код будет получен терминалом от вычислительной машины, он тут же печатается или высвечивается на экране терминала.

Однако добавление микрокомпьютера к видеотерминалу может придать последнему черты *интеллектуальности*. При нажатии клавиши соответствующие коды символов помещаются в память микрокомпьютера и высвечиваются на экране. Используя возможности микрокомпьютера внутри терминала, можно легко исправить ошибки. Если высвеченный на экране текст является правильным и полным, оператор может, нажимая соответствующую клавишу, дать команду микрокомпьютеру переслать текст в вычислительную машину в виде *целого блока*. Преимущество интеллектуального терминала состоит также в том, что с его помощью могут быть исправлены типографские ошибки прежде, чем текст будет передан в ЭВМ.

Микрокомпьютер, встроенный в видеотерминал, позволяет без значительных затрат реализовать много других функций. Например, терминал может быть настроен для связи с ЭВМ, которая использует любой из числа доступных терминалу *протоколов связи*. Таким образом, терминал может быть взаимозаменяем без дополнительной настройки с любыми типами ЭВМ. Хотя вопросы цифровой взаимосвязи выходят за рамки данной книги, рассмотрим два примера для иллюстрации этих концепций.

Обычный протокол связи требует, чтобы между ЭВМ и терминалом происходил *обмен сообщениями* с использованием *управляющих кодов* для определения момента начала передачи данных. Например, когда ЭВМ готова к приему данных от терминала, она посылает в терминал управляющий код *запроса* на пересылку блока данных. Терминал откликается на это, пересылая на ЭВМ блок данных, заканчивающийся управляющим кодом *завершения передачи*. Этот протокол показан на рис. 11.9.

На рис. 11.10 показан более сложный протокол связи с *квитированием*. В этом случае запрос может быть сделан ЭВМ до того, как она будет готова к приему данных. Запрос должен быть затем подтвержден терминалом, который пересылает в ЭВМ управляющий код *готовности*. После того как ЭВМ приняла управляющий код *готовности* и если она готова принять данные, она делает запрос на передачу данных, посылая управляющий код *начала передачи* в терминал. Терминал откликается на это, пересылая на ЭВМ блок данных, заканчивающийся

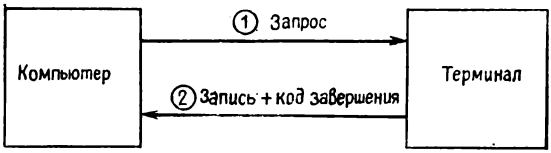


Рис. 11.9. Простой протокол связи.

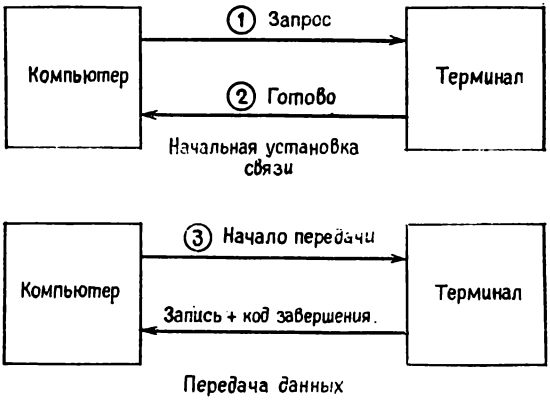


Рис. 11.10. Протокол связи с квитированием.

управляющим кодом *завершения передачи*. Микрокомпьютер обеспечивает гибкость, позволяя осуществить без существенных затрат множество таких протоколов для одного терминала.

Управляющие коды *запроса* и *готовности* предусмотрены в наборе символьных кодов ASCII, описанном в приложении Б. Однако для обозначения всех необходимых команд *интеллектуального* терминала может оказаться недостаточным набора стандартных управляющих кодов. Поэтому широко используются специальные кодовые последовательности или так называемые последовательности *переключения* кода. В этом случае последовательность из управляющего символа *переключения* кода (*escape*) и следующих за ним алфавитно-цифровых символьных кодов воспринимается как команда. Обычно бывает достаточно одного алфавитно-цифрового символьного кода вслед за символом *переключения* кода. Для сложных команд могут быть переданы дополнительные символьные коды. Точное число дополнительных символьных кодов зависит от символьного кода, следующего непосредственно вслед за управляющим символом *переключения* кода, однако оно может зависеть также и от последующих кодов. Несколько последовательностей переключения кода, используемых в терминалах серии Хьюлетт-

Escape	A	Переместить маркер на одну строку вверх
Escape	B	Переместить маркер на одну строку вниз
Escape	C	Переместить маркер на одну колонку вправо
Escape	D	Переместить маркер на одну колонку влево
Escape	H	Переместить маркер в начало экрана в первую колонку первой строки
Escape	J	Очистить экран
Escape	K	Удалить символы от маркера до конца строки
Escape	L	Вставить пустую строку над строкой, содержащей маркер
Escape	M	Удалить строку, содержащую маркер
Escape	P	Удалить символ, обозначенный маркером
Escape	S	Переместить текст циклически на одну строку вверх
Escape	T	Переместить текст циклически на одну строку вниз

Рис. 11.11. Пример последовательностей переключения кода.

*Escape**pA	Поднять перо
*Escape**pB	Опустить перо
*Escape**pa300, 200Z	Поднять перо и переместить его в точку с координатами (300, 200)
*Escape**pb550, 150Z	Опустить перо и прочертить линию от текущей точки до точки с координатами (550, 150).

Примечание 1. Заглавные и строчные символы интерпретируются одинаковым образом, заглавные символы играют роль кода завершения графической последовательности переключения.
Примечание 2. Символ Z используется только в качестве кода завершения и не интерпретируется никаким другим образом.

Рис. 11.12. Пример графических последовательностей переключения кода.

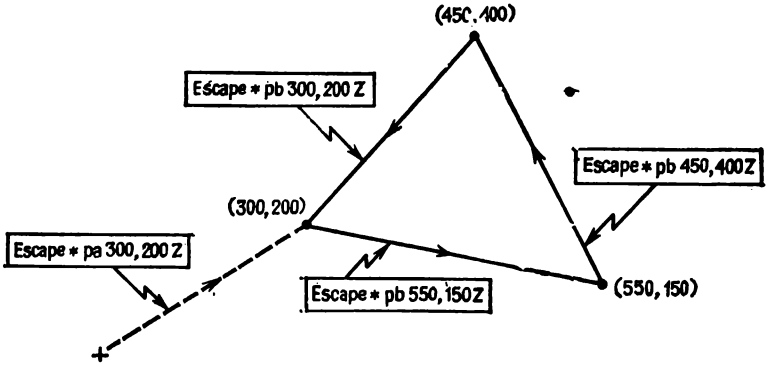


Рис. 11.13. Использование графических последовательностей переключения.

Паккард 264х, показано на рис. 11.11. В графических терминалах последовательности переключения кода используются для получения линий на экране дисплея. Примеры графических последовательностей переключения кода, используемых в графических терминалах серии Хьюлетт-Паккард 264х, показаны на рис. 11.12. Рис. 11.13 иллюстрирует использование графиче-

ских последовательностей переключения кода для получения простой геометрической фигуры на экране графического терминала серии Хьюлетт-Паккард 264х.

11.3. Системы управления автомобильным двигателем

Для микрокомпьютера с его постоянно уменьшающейся стоимостью аппаратуры и быстрым возрастанием возможностей проникнуть под капот автомобиля было только делом времени. Внедрение управления автомобильным двигателем в реальном масштабе времени было ускорено общественным и правительственным давлением, направленным на уменьшение загрязнения воздуха автомобилем и уменьшение расхода топлива. Эти цели могут быть достигнуты только при условии, если горючая смесь, вводимая в двигатель внутреннего сгорания, и время опережения зажигания будут регулироваться более точно, чем это может быть выполнено с помощью механического карбюратора и распределителя. Однако для того, чтобы управлять задачами впуска топлива и распределения зажигания, микрокомпьютер должен воспринять огромное количество информации об окружающей среде в режиме реального времени и учитывать характеристики конкретного управляемого двигателя. Полное описание системы управления двигателем выходит за рамки нашей дискуссии. Вместо этого мы рассмотрим упрощенное описание для иллюстрации основных концепций.

Блок-схема системы управления двигателем, основанной на использовании микрокомпьютера, показана на рис. 11.14. Модули преобразования входного и выходного сигналов для этой системы и набор правил управления двигателем являются уникальными для данного типа систем и заслуживают особого внимания. В качестве датчика магистрального давления, а также для других подобных аналоговых входных сигналов может быть использован стандартный аналого-цифровой преобразователь. Совсем другой тип преобразователя входного сигнала требуется для сигнала распределения времени зажигания, который состоит из серии импульсов, частота которых пропорциональна скорости вращения маховика двигателя. Этот преобразователь входного сигнала должен измерять временной интервал между двумя последовательными импульсами и преобразовывать значение этого интервала в цифровую величину, пропорциональную каждому временному интервалу. Скорость вращения двигателя, соответствующая каждому временному интервалу, вычисляется микрокомпьютером путем умножения величины, обратной цифровому значению временного интервала, полученного от преобразователя входного сигнала, на подходящий градуировочный коэффициент. Третий тип входного

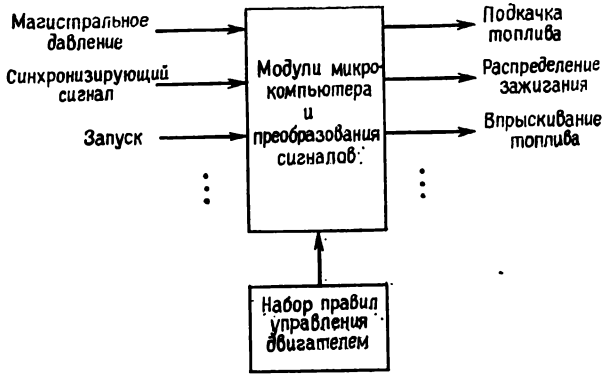


Рис. 11.14. Схема микрокомпьютерной системы управления двигателем внутреннего сгорания.

сигнала предусматривает индикацию состояния типа «включено-выключено» и может быть представлен однобитовой бинарной переменной. Сигнал запуска, например, обозначает, что двигатель заводится стартером.

Модуль преобразования выходного сигнала должен генерировать множество пусковых сигналов. Сигнал топливного насоса представляет собой сигнал с двумя состояниями, который включает топливный насос перед запуском двигателя стартером и выключает его при выключении ключа зажигания. Сигнал распределения зажигания состоит из последовательности импульсов, точное распределение которых контролируется микрокомпьютером с целью обеспечения опережения зажигания, оптимального для данного двигателя и внешних условий. Сигнал впрыскивания топлива представляет собой импульс, длительность и момент возникновения которого определяются микрокомпьютером.

Правила управления двигателем описываются набором таблиц, содержащих многозначные функциональные связи, необходимые для управления пусковыми сигналами с использованием информации от входных сигналов. Чтобы стоимость системы оставалась в разумных пределах, эти правила должны быть относительно компактными. Так, например, они не могут хранить необходимую управляющую информацию для всех возможных комбинаций входных сигналов, поэтому микрокомпьютер должен проводить интерполяцию хранимых величин с целью экономного использования управляющей информации в правилах управления двигателем. Кроме того, микрокомпьютер должен изменять значения некоторых из пусковых сигналов с очень большой скоростью, когда двигатель работает на высоких обо-

ротах. Поэтому архитектура микрокомпьютера для системы управления двигателем внутреннего сгорания должна выбираться такой, чтобы вычисления, необходимые для интерполяции, выполнялись быстро. Сложный микрокомпьютер общего назначения легко выполняет требуемые интерполяции с большой скоростью, однако для данного применения стоимость его слишком высока. В результате оказывается более выгодным спроектировать специальную интегральную схему, содержащую модули микрокомпьютера и преобразования сигналов для системы управления двигателем внутреннего сгорания. Принципы, изложенные в данной книге, могут быть использованы для проектирования систем, содержащих микрокомпьютер специального назначения. Проектирование аппаратуры при этом ограничивается проектированием одной интегральной схемы, поскольку все устройства объединяются вместе. А поскольку язык программирования высокого уровня здесь вряд ли пригоден, язык проектирования должен непосредственно конвертироваться в язык ассемблера. Кроме этих двух отличий, наша методология целиком применима для проектирования систем, содержащих микрокомпьютер специального назначения.

11.4. Электробытовые товары

К группе электробытовых товаров относятся холодильники, электроплиты, духовки, стиральные машины и сушилки. Сравнительно недавно изготовители начали заменять электромеханические таймеры и устройства управления микрокомпьютерными схемами. Применение микрокомпьютеров позволяет в значительной степени повысить гибкость использования бытовых товаров и добавляет новые функциональные возможности при сравнительно небольшом возрастании стоимости.

Примером применения, где в полной мере используются возможности управления с помощью микрокомпьютера, является микроволновая печь. Поскольку приготовление пищи микроволновым способом требует меньше времени по сравнению с обычным, желательно, чтобы с помощью цифрового таймера и устройства управления пользователю были предоставлены средства, повышающие точность и облегчающие управление печью. Кроме того, в этом приложении наиболее полно используются преимущества возрастания гибкости благодаря наличию встроенного микрокомпьютера.

Микроволновая печь управляется путем изменения двух параметров: времени приготовления пищи и уровня мощности нагревателя. Микрокомпьютер позволяет управлять печью путем установки точной комбинации мощности и времени варки при приготовлении пищи. Например, если продукты, которые

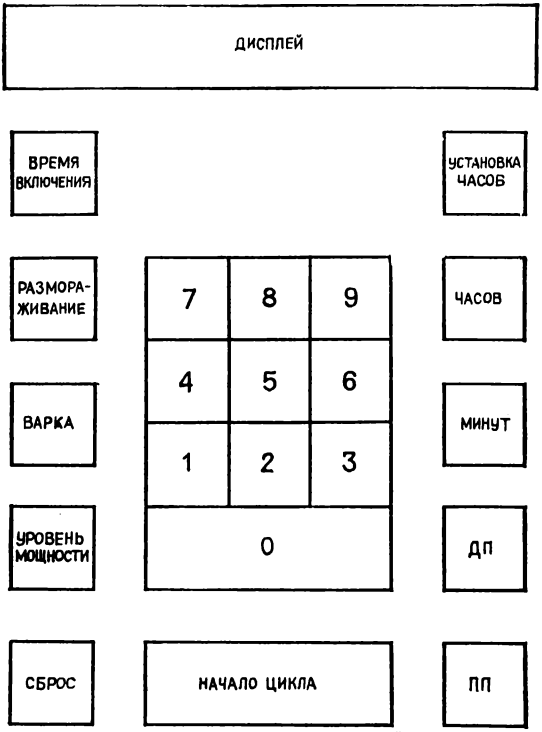


Рис. 11.15. Схема панели управления микроволновой печи.

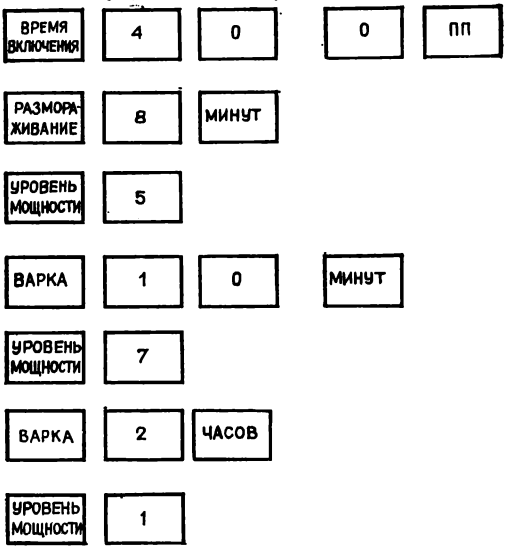


Рис. 11.16. Пример программирования режимов приготовления пищи.

должны быть сварены, находятся в замороженном состоянии, их необходимо сначала разморозить при среднем уровне мощности нагревателя в течение определенного периода времени, а затем варить при высоком уровне мощности в течение другого периода времени. После этого продукты могут сохраняться теплыми при самом малом уровне мощности, пока их не вытащат из печи и не подадут на стол.

Уровни мощности нагревателя и интервалы времени легко устанавливаются при помощи устройства с кнопками и цифрового индикатора. Один из вариантов панели управления микроволновой печью показан на рис. 11.15. Типичная последовательность при установке режима приготовления пищи показана на рис. 11.16.

Каждый прямоугольник в последовательности обозначает нажатие кнопки. Функциональные кнопки, например кнопка РАЗМОРАЖИВАНИЕ, загораются при нажатии, цифровые кнопки при нажатии высвечиваются на индикаторе панели. В примере на рис. 11.16 печь устанавливается на включение в 4.00 после полудня. Она размораживает продукты в течение восьми минут при уровне мощности нагревателя 5, варит их в течение десяти минут при уровне мощности 7 и поддерживает сваренные продукты в горячем состоянии в течение двух часов при уровне мощности 1. Для запуска печи, чтобы она начала заданный цикл в 4.00 после полудня, необходимо нажать клавишу НАЧАЛО ЦИКЛА. Цикл может быть прерван в любой момент нажатием кнопки ВЫКЛЮЧЕНИЕ.

Когда световой индикатор не используется для установки режима варки, он может показывать текущее время суток. Для установки времени суток сначала необходимо нажать кнопку УСТАНОВКА ЧАСОВ, затем с помощью кнопок ЧАСЫ и МИНУТЫ устанавливается правильное время, а с помощью кнопок ДП и ПП устанавливается период времени «до полудня» или «после полудня». Для восстановления системы в нормальное состояние кнопка УСТАНОВКА ЧАСОВ нажимается второй раз.

11.5. Заключение

Примеры и упражнения, представленные в данной главе, имеют цель заставить читателя задуматься о других способах использования микрокомпьютеров. Практически нет ограничений для множества приложений, которые могут быть реализованы с использованием микрокомпьютера как компонента. Однако для того, чтобы это было экономически оправданно и выполнено с наименьшими трудностями, требуется, чтобы работа проводилась на системной основе и чтобы сложность была

минимальной. Концепции проектирования, рассмотренные в данной книге, могут помочь читателю достичь этой цели простыми средствами.

11.6. Упражнения

Системы, описанные в гл. 11, являются более сложными по сравнению с теми, которые использовались в качестве упражнений в предыдущих главах. Поэтому полное проектирование одной из этих систем в классных условиях может оказаться трудным. Однако можно определить упрощенные версии этих систем, которые могут быть использованы в качестве примеров для решения в классе. Ниже для иллюстрации рассматривается упрощенный пример телевизионной игры.

11.1. Описать процедуры на языке проектирования для телевизионной игры со следующими характеристиками:

а) Имеются две ракетки, по одной для каждого игрока, которые могут перемещаться вверх и вниз.

б) У каждой вертикальной стенки поставлены ворота.

в) Игра начинается нажатием кнопки НАЧАЛО.

г) После начала мяч перемещается с постоянной скоростью под углом сорок пять градусов, пока не попадет в ворота. После этого счет изменяется и система ждет, пока не будет снова нажата кнопка НАЧАЛО.

д) При столкновении мяча со стенкой или с ракеткой он отражается и продолжает движение с той же скоростью под углом сорок пять градусов.

е) Игра прекращается после того, как один из игроков наберет пятнадцать очков. Нажатием кнопки СБРОС счет сбрасывается, и можно начинать игру снова. Кнопка СБРОСА может быть нажата в любой момент независимо от счета.

Системы счисления

Система счисления представляет цифровую информацию. Наиболее известной системой счисления является *десятичная* система счисления. Она включает набор из десяти символов, показанных на рис. А.1, и *десятичной точки.* Числа представ-

Наименование символа	Символ
нуль	0
один	1
два	2
три	3
четыре	4
пять	5
шесть	6
семь	7
восемь	8
девять	9

Рис. А.1. Символы десятичной системы счисления.

ляются последовательностью этих символов. Значение каждого символа в последовательности зависит от его позиции относительно десятичной точки. Если десятичная точка опущена, предполагается, что она находится правее самого правого символа последовательности. Каждый символ в последовательности отличается эт соседних символов множителем, кратным десяти.

Таким образом, последовательность 325 обозначет число, значение которого определяется выражением

$$3 \times 10^2 + 2 \times 10^1 + 5 \times 10^0$$

или выражением «триста двадцать пять». Соответственно последовательность 25,2 обозначает число, значение которого определяется выражением

$$2 \times 10^1 + 5 \times 10^0 + 2 \times 10^{-1}$$

или выражением «двадцать пять и две десятых».

Число десять называется *основанием* десятичной системы счисления. В цифровых вычислительных системах используются также *двоичная, восьмеричная* и *шестнадцатеричная* системы счисления. Основаниями для этих систем служат соответственно числа 2, 8 и 16.

Как мы уже видели, информация хранится и обрабатывается в микрокомпьютере в виде *битов* или групп битов. Бит, или *двоичная цифра* (*binary digit* — сокращенно *bit*), является переменной, которая принимает значение, равное нулю или единице — одному из двух символов, существующих в двоичной системе счисления. Последовательность битов, представляющая цифровую информацию, может быть преобразована в десятичное число путем умножения каждого бита на соответствующую степень двух. Например, последовательность 0010 1101 обозначает десятичное число 45, поскольку

$$\begin{aligned} 0010\ 1101 &= 0\times 2^7 + 0\times 2^6 + 1\times 2^5 + 0\times 2^4 + \\ &\quad + 1\times 2^3 + 1\times 2^2 + 0\times 2^1 + 1\times 2^0 = \\ &= 32 + 4 + 8 + 1 = \\ &= 45 \end{aligned}$$

Двоичные числа, содержащие дробные части, могут быть преобразованы в десятичные подобным же образом. Следует отметить, что двоичная точка используется для разделения целой и дробной частей двоичного числа так же, как и десятичная при разделении соответствующих частей десятичного числа.

В микрокомпьютерных системах восьмеричная (основание — восемь) и шестнадцатеричная (основание — шестнадцать) системы счисления используются прежде всего для удобства чтения двоичной информации. Между этими системами счисления и двоичной системой счисления существует простая связь. Каждый символ в восьмеричной системе счисления соответствует трем битам соответствующего двоичного числа, а каждый символ в шестнадцатеричной системе счисления — четырем битам соответствующего двоичного числа. Эти соответствия показаны на рис. А.2 и А.3. Так, например, число 0010 1101 эквивалентно числу 055 (00 101 101) в восьмеричной системе счисления и числу 2D в шестнадцатеричной. Чтобы определить значение восьмеричного или шестнадцатеричного числа, используются правила, подобные правилам преобразования двоичного числа в десятичное. Поэтому восьмеричное число 055 эквивалентно десятичному числу 45, поскольку

$$\begin{aligned} 055 &= 0\times 8^2 + 5\times 8^1 + 5\times 8^0 = \\ &= 40 + 5 = \\ &= 45 \end{aligned}$$

Наименование символа	Символ	Соответствующее двоичное число
нуль	0	000
один	1	001
два	2	010
три	3	011
четыре	4	100
пять	5	101
шесть	6	110
семь	7	111

Рис. А.2. Символы восьмеричной системы счисления.

Наименование символа	Символ	Соответствующее двоичное число
нуль	0	0000
один	1	0001
два	2	0010
три	3	0011
четыре	4	0100
пять	5	0101
шесть	6	0110
семь	7	0111
восемь	8	1000
девять	9	1001
десять	A	1010
одиннадцать	B	1011
двенадцать	C	1100
тринадцать	D	1101
четырнадцать	E	1110
пятнадцать	F	1111

Рис. А.3. Символы шестнадцатеричной системы счисления.

Соответственно шестнадцатеричное число 2D также эквивалентно десятичному числу 45, поскольку

$$\begin{aligned} 2D &= 2 \times 16^1 + 13 \times 16^0 = \\ &= 32 + 13 = \\ &= 45 \end{aligned}$$

Как мы видели в гл. 6, шестнадцатеричное представление используется для упрощения действий с командами на машинном языке, так как работать с последовательностью шестнадцатеричных символов легче, чем с последовательностью битов. Поэтому команда, которая в языке ассемблера Intel 8085 имеет представление ADC, в шестнадцатеричном представлении

будет иметь вид 8E, а не 1000 1110, как она в действительности представлена в микрокомпьютере. Никаких числовых значений не связано с машинным представлением данной команды, которая воспринимается микрокомпьютером как нечисловая величина. Однако в других случаях часть команды может быть представлена числовым значением. Например, в машинном представлении команды DE 23 для микрокомпьютера Intel 8085 часть DE является представлением команды языка ассемблера SBI, а часть 23 является шестнадцатеричным числом, представляющим операнд, десятичное значение которого равно 35. Таким образом, команда машинного языка, представленная шестнадцатеричной последовательностью DE 23, эквивалентна команде языка ассемблера SBI 35. Аналогичным образом, если последовательность битов или шестнадцатеричных символов используется для представления символьных кодов, их числовое значение не принимается во внимание. Символьные коды обсуждаются в приложении Б.

Символьные данные

В гл. 5 было введено понятие символьных кодов чисел. В гл. 6 было показано, как используются символьные коды чисел. В данном приложении описывается более общая концепция символьных данных, используемых для представления и обработки информации, содержащей алфавитные и цифровые символы, а также такие символы, как запятые, точки, вопросительные знаки и прочее. Как и раньше, будет использоваться представление в коде ASCII, или в Стандартном американском коде для обмена информацией (American Standard Code for Information Interchange), которое повсеместно используется в микрокомпьютерных системах.

Рис. Б.1 содержит стандартное представление символов в коде ASCII. Каждый код показан в двоичном и шестнадцатеричном представлении. Крайний левый бит каждого двоичного кода, или *бит четности*, равен нулю. Это соглашение называется *нулевым паритетом*. Всего имеется четыре возможных варианта установки бита четности, а именно:

1. Бит четности всегда равен НУЛЮ (нулевой паритет).
2. Бит четности всегда равен ЕДИНИЦЕ (единичный паритет).
3. Бит четности устанавливается в НУЛЬ, если число единичных битов кода четное (четный паритет).
4. Бит четности устанавливается в НУЛЬ, если число единичных битов кода нечетное (нечетный паритет).

Последние два варианта могут быть использованы для контроля при передаче данных по линиям связи. В локальных микрокомпьютерных системах, например в линии связи между локальным терминалом и микрокомпьютером, контроль по четности употребляется редко. Обычно в этом случае используется нулевой или единичный паритет.

Как мы уже видели, в языке PL/M алфавитно-цифровые символы заключаются в апострофы. В этом случае если компилятор PL/M транслирует предложение, например

```
IF CHARACTER >= '0' AND CHARACTER <= '9'
```

то при этом он замещает символы '0' и '9' кодами ASCII 0011 0000 и 0011 1001 соответственно. При выполнении соот-

Символьное представление	Представление в кодах ASCII	
	Двоичное	Шестнадцатеричное
NUL	0000 0000	00
SOH	0000 0001	01
STX	0000 0010	02
ETX	0000 0011	03
EOT	0000 0100	04
ENQ	0000 0101	05
ACK	0000 0110	06
BEL	0000 0111	07
BS	0000 1000	08
HT	0000 1001	09
LF	0000 1010	0A
VT	0000 1011	0B
FF	0000 1100	0C
CR	0000 1101	0D
SO	0000 1110	0E
SI	0000 1111	0F
DLE	0001 0000	10
DC1	0001 0001	11
DC2	0001 0010	12
DC3	0001 0011	13
DC4	0001 0100	14
NAK	0001 0101	15
SYN	0001 0110	16
ETB	0001 0111	17
CAN	0001 1000	18
EM	0001 1001	19
SUB	0001 1010	1A
ESC	0001 1011	1B
FS	0001 1100	1C
GS	0001 1101	1D
RS	0001 1110	1E
US	0001 1111	1F
пробел	0010 0000	20
!	0010 0001	21
"	0010 0010	22
#	0010 0011	23
$	0010 0100	24
%	0010 0101	25
&	0010 0110	26
'	0010 0111	27
(	0010 1000	28
)	0010 1001	29
*	0010 1010	2A
+	0010 1011	2B
,	0010 1100	2C

Рис. Б.1. Стандартное представление в кодах ASCII.

Символьное представление	Представление в кодах ASCII	
	Двоичное	Шестнадцатеричное
—	0010 1101	2D
.	0010 1110	2E
/	0010 1111	2F
0	0011 0000	30
1	0011 0001	31
2	0011 0010	32
3	0011 0011	33
4	0011 0100	34
5	0011 0101	35
6	0011 0110	36
7	0011 0111	37
8	0011 1000	38
9	0011 1001	39
:	0011 1010	3A
;	0011 1011	3B
<	0011 1100	3C
=	0011 1101	3D
>	0011 1110	3E
?	0011 1111	3F
@	0100 0000	40
A	0100 0001	41
B	0100 0010	42
C	0100 0011	43
D	0100 0100	44
E	0100 0101	45
F	0100 0110	46
G	0100 0111	47
H	0100 1000	48
I	0100 1001	49
J	0100 1010	4A
K	0100 1011	4B
L	0100 1100	4C
M	0100 1101	4D
N	0100 1110	4E
O	0100 1111	4F
P	0101 0000	50
Q	0101 0001	51
R	0101 0010	52
S	0101 0011	53
T	0101 0100	54
U	0101 0101	55
V	0101 0110	56
W	0101 0111	57
X	0101 1000	58
Y	0101 1001	59
Z	0101 1010	5A

Рис. Б.1. (Продолжение).

Символьное представление	Представление в кодах ASCII	
	Двоичное	Шестнадцатеричное
[	0101 1011	5B
\	0101 1100	5C
]	0101 1101	5D
^	0101 1110	5E
—	0101 1111	5F
'	0110 0000	60
a	0110 0001	61
b	0110 0010	62
c	0110 0011	63
d	0110 0100	64
e	0110 0101	65
f	0110 0110	66
g	0110 0111	67
h	0110 1000	68
i	0110 1001	69
j	0110 1010	6A
k	0110 1011	6B
l	0110 1100	6C
m	0110 1101	6D
n	0110 1110	6E
o	0110 1111	6F
p	0111 0000	70
q	0111 0001	71
r	0111 0010	72
s	0111 0011	73
t	0111 0100	74
u	0111 0101	75
v	0111 0110	76
w	0111 0111	77
x	0111 1000	78
y	0111 1001	79
z	0111 1010	7A
{	0111 1011	7B
|	0111 1100	7C
}	0111 1101	7D
~	0111 1110	7E
вычеркивание	0111 1111	7F

Рис. Б.1. (Продолжение).

Управляющие символы

NUL	Пусто
SOH	Начало заголовка
STX	Начало текста
ETX	Конец текста
EOT	Конец передачи
ENQ	Запрос
ACK	Подтверждение
BEL	Звонок
BS	Возврат на шаг
HT	Горизонтальная табуляция
LF	Перевод строки
VT	Вертикальная табуляция
FF	Перевод формата
CR	Возврат каретки
SO	Национальный регистр
SI	Латинский регистр
DLE	Авторегистр 1
DC1	Управление устройством 1
DC2	Управление устройством 2
CD3	Управление устройством 3
DC4	Управление устройством 4
NAK	Отрицание
SYN	Синхронизация
ETB	Конец блока
CAN	Аннулирование
EM	Конец носителя
SUB	Замена
ESC	Авторегистр 2 (символ кода переключения)
FS	Разделитель файлов
GS	Разделитель групп
RS	Разделитель записей
US	Разделитель элементов

Рис. Б.1. (Продолжение).

ветствующих команд микрокомпьютер воспринимает эти коды как цифровые величины. Поскольку цифровые коды ASCII расположены в возрастающей последовательности один за другим, проверка определяет, находится ли символьный код, обозначенный как CHARACTER, между кодами '0' и '9' и является ли он, таким образом, цифровым кодом.

В такой же последовательности расположены алфавитные символьные коды. Это означает, что код ASCII для буквы 'A' имеет наименьшее цифровое значение, а для буквы 'Z' — наибольшее. Таким образом, если предложение на языке PL/M

```
IF CHARACTER >= 'C' AND CHARACTER <= 'Q'
```

транслируется компилятором и затем выполняются соответствующие команды, символьный код, обозначенный как CHA-

RACTER, проверяется, чтобы определить, представляет ли он букву, расположенную между 'C' и 'Q'.

Как показано на рис. Б.1, имеется представление в кодах ASCII для прописных и строчных алфавитных символов, цифровых символов, таких символов, как !, ", #, $ и др., а также для управляющих символов. Первые три типа кодов вместе с кодами пробела, перевода строки и возврата каретки обеспечивают возможность цифрового представления, обработки и передачи обычных текстов. Управляющие коды первоначально использовались для облегчения представления информации в кодах ASCII при передаче на большие расстояния по телеграфным линиям. В микрокомпьютерных системах они используются как для связи, так и для расширения управляющих возможностей микрокомпьютера. Примеры использования управляющих кодов для связи терминала с микрокомпьютером, а также для управления графическим терминалом были приведены в гл. 11.

Сводка конструкций языка PL/M

В приложении содержится сводка примеров, иллюстрирующих преобразование конструкций языка проектирования в PL/M.

Описание модуля.

```
/* МОДУЛЬ: ОБРАБОТКА ВХОДНОЙ ЗАПИСИ                          */
                        MAINTAIN$INPUT$RECORD: DO;
                                  .
                                  .
                                  .
/* КОНЕЦ МОДУЛЯ                                              */
                        END MAINTAIN$INPUT$RECORD
```

Описание данных с инициализацией.

```
/* МОДУЛЬНАЯ СТРУКТУРА ДАННЫХ: УПРАВЛЯЮЩИЕ
   ДАННЫЕ                                                    */
/*                                                           */
/* ИМЯ            РАЗМЕР ТИП  СОДЕРЖАНИЕ ПРИМЕЧАНИЯ         */
/* ---            ------ ---  ---------- ----------         */
*/ УПРАВЛЯЮЩИЕ 10       БАЙТ ЦИФРОВЫЕ
   ДАННЫЕ                    ВЕЛИЧИНЫ                       */
                  DECLARE CONTROL$SCHEDULE (10) BYTE
                  DATA (10, 15, 25, 40, 60, 76, 88, 98, 106, 110);
```

Описание процедур.

```
/* ПРОЦЕДУРА: СТИРАНИЕ ВХОДНОЙ ЗАПИСИ (;)                    */
                        CLEAR$INPUT$RECORD: PROCEDURE;
                                  .
                                  .
                                  .
/* КОНЕЦ ПРОЦЕДУРЫ                                           */
                        END CLEAR$INPUT$RECORD;
/* ПРОЦЕДУРА: СИГНАЛ (;)                                     */
                        BEEP: PROCEDURE PUBLIC;
                                  .
                                  .
                                  .
/* КОНЕЦ ПРОЦЕДУРЫ                                           */
                        END BEEP;
```

Описание данных.

```
/*  ВЫХОДНОЙ ПАРАМЕТР: СТАТУС                                              */
/*                                                                         */
/*  ИМЯ                  РАЗМЕР   ТИП     СОДЕРЖАНИЕ        ПРИМЕЧАНИЯ   */
/*  ---                  ------   ---     ----------        ----------   */
/*  СТАТУС               1        БАЙТ    ФЛАЖОК                         */
                                                DECLARE STATUS BYTE;
/*  МОДУЛЬНАЯ СТРУКТУРА ДАННЫХ: ВХОДНАЯ ЗАПИСЬ                             */
/*                                                                         */
/*  ИМЯ                  РАЗМЕР   ТИП     СОДЕРЖАНИЕ        ПРИМЕЧАНИЯ   */
/*  ---                  ------   ---     ----------        ----------   */
/*  ВХОДНАЯ ЗАПИСЬ       10       БАЙТ    ЦИФРОВЫЕ КОДЫ                  */
                                        DECLARE INPUT$RECORD (10) BYTE;
/*  МОДУЛЬНАЯ СТРУКТУРА ДАННЫХ: ФАЙЛ                                       */
/*                                                                         */
/*  ИМЯ                  РАЗМЕР   ТИП     СОДЕРЖАНИЕ        ПРИМЕЧАНИЯ   */
/*  ---                  ------   ---     ----------        ----------   */
/*  ФАЙЛ                                                        1          */
/*    ЗАПИСЬ                                                             */
/*      ПОРЯДКОВЫЙ НОМЕР 10       АДРЕС   ЦИФРОВЫЕ КОДЫ                  */
/*      ИМЯ              20       БАЙТ    АЛФАВИТНЫЕ КОДЫ                */
/*      ДОМАШНИЙ АДРЕС   20       БАЙТ    АЛФАВИТНЫЕ КОДЫ                */
/*                                                                         */
/*  ПРИМЕЧАНИЕ 1: ФАЙЛ СОСТОИТ ИЗ 256 ЗАПИСЕЙ.                            */
/*                                                                         */
                    DECLARE FILE (256) STRUCTURE
                          (SERIAL$NUMBER ADDRESS,
                          NAME (20) BYTE,
                          STREET$ADDRESS (20) BYTE);
```

Описание процедуры с входным параметром.

```
/* ПРОЦЕДУРА: ДОБАВЛЕНИЕ К ВХОДНОЙ ЗАПИСИ (СИМВОЛ;)  */
        ADD$TO$INPUT$RECORD: PROCEDURE (CHARACTER);
              .
              .
              .
/* КОНЕЦ ПРОЦЕДУРЫ                                    */
        END ADD$TO$INPUT$RECORD;
```

Описание процедуры с выходным параметром.

```
/* ПРОЦЕДУРА: ПРОВЕРКА ЗАПОЛНЕНИЯ ВХОДНОЙ ЗАПИСИ     */
  (;СТАТУС)                                          */
        TEST$IF$INPUT$RECORD$FULL: PROCEDURE BYTE;
              .
              .
              .
        RETURN STATUS
              .
              .
              .
/* КОНЕЦ ПРОЦЕДУРЫ                                    */
        END TEST$IF$INPUT$RECORD$FULL;
```

Описание процедуры со структурой данных в качестве входного параметра.

```
/* ПРОЦЕДУРА: ПЕЧАТЬ (ЗАПИСЬ;)                       */
                    PRINT: PROCEDURE (RECORD$POINTER)
                    PUBLIC;
/* ****************************************************** */
/* ЛОКАЛЬНЫЙ ПАРАМЕТР: УКАЗАТЕЛЬ ЗАПИСИ               */
/*                                                    */
/* ИМЯ          РАЗМЕР ТИП   СОДЕРЖАНИЕ ПРИМЕЧАНИЯ */
/* ---          ------ ---   ---------- ---------- */
/* УКАЗАТЕЛЬ      1    АДРЕС УКАЗАТЕЛЬ
   ЗАПИСИ                   СТРУКТУРЫ              */
                    DECLARE RECORD$POINTER ADDRESS;
/* ВХОДНОЙ ПАРАМЕТР: ЗАПИСЬ                          */
/*                                                    */
/* ИМЯ          РАЗМЕР ТИП   СОДЕРЖАНИЕ ПРИМЕЧАНИЯ */
/* ---          ------ ---   ---------- ---------- */
/* ЗАПИСЬ                                      1      */
/*   ПОРЯДКОВЫЙ   1    АДРЕС ЧИСЛО
     НОМЕР                                          */
/*   ИМЯ          20   БАЙТ  АЛФАВИТНЫЕ
                           КОДЫ                     */
/*   ДОМАШНИЙ     20   БАЙТ  АЛФАВИТНО-
     АДРЕС                 ЦИФРОВЫЕ
                           КОДЫ                     */
/*                                                    */
/* ПРИМЕЧАНИЕ 1: ЗАПИСЬ БАЗИРОВАНА В СООТВЕТСТВИИ С  */
```

```
/*                  УКАЗАТЕЛЕМ ЗАПИСИ                              */
                        DECLARE (RECORD BASED
                        RECORD$POINTER) STRUCTURE
                        (SERIAL$NUMBER ADDRESS, NAME (20)
                        BYTE, STREET$ADDRESS (20) BYTE);
                                  .
                                  .
                                  .
/* КОНЕЦ ПРОЦЕДУРЫ                                                 */
                        END PRINT;
```

Описание процедуры со структурой данных в качестве выходного параметра.

```
/* ПРОЦЕДУРА: ПОИСК (КЛЮЧ ПОИСКА, ЗАПИСЬ)                         */
                        SEARCH: PROCEDURE (SEARCH$KEY)
                        ADDRESS PUBLIC;
/* ************************************************************** */
                                  .
                                  .
                                  .
/* ВЫХОДНОЙ ПАРАМЕТР: ЗАПИСЬ                                      */
/*                                                                */
/* ИМЯ              РАЗМЕР ТИП    СОДЕРЖАНИЕ ПРИМЕЧАНИЯ           */
   ---              ------ ---    ---------- ----------            */
/* ЗАПИСЬ                                                         */
/*   ПОРЯДКОВЫЙ        1    АДРЕС ЧИСЛО
     НОМЕР                                                        */
/*   ИМЯ              20    БАЙТ  АЛФАВИТНЫЕ
                                  КОДЫ                          */
/*   ДОМАШНИЙ         20    БАЙТ  АЛФАВИТНО-
     АДРЕС                        ЦИФРОВЫЕ
                                  КОДЫ                          */
                        DECLARE RECORD STRUCTURE
                        (SERIAL$NUMBER ADDRESS, NAME (20)
                        BYTE, STREET$ADDSESS (20) BYTE);
/* ЛОКАЛЬНЫЙ ПАРАМЕТР: УКАЗАТЕЛЬ ЗАПИСИ                           */
/*                                                                */
/* ИМЯ              РАЗМЕР ТИП    СОДЕРЖАНИЕ ПРИМЕЧАНИЯ           */
/* ---              ------ ---    ---------- ----------            */
/* УКАЗАТЕЛЬ           1    АДРЕС УКАЗАТЕЛЬ                      */
   ЗАПИСИ                    DECLARE RECORD$POINTER ADDRESS;
/* ************************************************************** */
                                  .
                                  .
                                  .
/*     УСТАНОВИТЬ ЗНАЧЕНИЕ ЗАПИСИ РАВНЫМ ЗНАЧЕНИЮ
       ВЫБРАННОЙ ЗАПИСИ                                          */
                                  .
                                  .
                                  .
```

```
/*    ВОЗВРАТ                                                    */
                        RECORD$POINTER = .RECORD;
                        RETURN RECORD$POINTER;
                          .
                          .
                          .
```

Управляющие конструкции.

```
/* ВЫЗОВ: ОСТАНОВКА ТАЙМЕРА (;)                                  */
                        CALL STOP$TIMER;
/* ВЫЗОВ: ДОБАВЛЕНИЕ К ВХОДНОЙ ЗАПИСИ (ЗАПИСЬ;)                  */
                        CALL ADD$TO$INPUT$RECORD (CHARACTER);
/* ВЫЗОВ: ПРОВЕРКА ЗАПОЛНЕНИЯ ВХОДНОЙ ЗАПИСИ
   (;СТАТУС)                                                     */
                        STATUS = TEST$IF$INPUT$RECORD$FULL;
/* ВОЗВРАТ                                                       */
                        RETURN;
```

(конструкция ВОЗВРАТ для процедуры с выходным параметром СТАТУС)

```
/* ВОЗВРАТ                                                       */
                        RETURN STATUS;
```

Конструкции присваивания.

```
/* ПЕРЕВЕСТИ СТАТУС В СОСТОЯНИЕ "ЗАПОЛНЕНО"                      */
                        STATUS = 1;
/* ПЕРЕВЕСТИ СТАТУС В СОСТОЯНИЕ "НЕ ЗАПОЛНЕНО"                   */
                        STATUS = 0;
/* УСТАНОВИТЬ ПРОДОЛЖЕНИЕ                                        */
                        CONTINUOUS = 1;
/* СБРОСИТЬ ПРОДОЛЖЕНИЕ                                          */
                        CONTINUOUS = 0;
/* УСТАНОВИТЬ ВРЕМЯ В СЕКУНДАХ                                   */
                        TIME = 3600 * HOURS + 60 * MINUTES + SECONDS;
```

Конструкции цикла.

```
/* ВЫПОЛНИТЬ                                                     */
                                        DO;
                                          .
                                          .
                                          .
/* КОНЕЦ                                                         */
                                        END;
/* ВЫПОЛНЯТЬ, ПОКА ПЕРЕКЛЮЧАТЕЛЬ ВКЛЮЧЕН                         */
                                        DO WHILE KEYSWITCH <> 0;
                                          .
                                          .
                                          .
/* КОНЕЦ                                                         */
                                        END;
/* ВЫПОЛНЯТЬ, ПОКА ПЕРЕКЛЮЧАТЕЛЬ ВЫКЛЮЧЕН                        */
                                        DO WHILE KEYSWITCH = 0;
```

```
                                    .
                                    .
                                    .
/* КОНЕЦ                                                        */
                                END
/* ВЫПОЛНЯТЬ ДЛЯ КАЖДОГО КОНТАКТНОГО ДЕТЕКТОРА                  */
                                DO INDEX = 1 TO 5;
                                    .
                                    .
                                    .
/* КОНЕЦ                                                        */
                                END;
/* ВЫПОЛНЯТЬ НЕПРЕРЫВНО                                         */
                                DO WHILE 1 = 1;
                                    .
                                    .
                                    .
/* КОНЕЦ                                                        */
                                END;
```

Условные конструкции.

```
/* ЕСЛИ ПЕРЕКЛЮЧАТЕЛЬ ВКЛЮЧЕН                                   */
                    IF KEYSWITCH < > 0
/* ТО ВОЗВРАТ                                                   */
                      THEN RETURN;
/* ЕСЛИ ПЕРЕКЛЮЧАТЕЛЬ ВЫКЛЮЧЕН                                  */
                    IF KEYSWITCH = 0
/* ТО ВОЗВРАТ                                                   */
                      THEN RETURN;
/* ЕСЛИ ПЕРЕКЛЮЧАТЕЛЬ ВКЛЮЧЕН И ТАЙМЕР В СОСТОЯНИИ
   ПОКОЯ                                                        */
                    IF KEYSWITCH < > 0 AND TIMER$STATE = 0
/* ТО ВОЗВРАТ                                                   */
                      THEN RETURN;
/* ЕСЛИ ИНДИКАТОР ЗАПОЛНЕНИЯ ПОКАЗЫВАЕТ ЧТО ВХОДНАЯ
   ЗАПИСЬ ЗАПОЛНЕНА                                            */
/*                                                             */
                    IF FULL$INDICATOR = 10
/* ТО ПЕРЕВЕСТИ СТАТУС В СОСТОЯНИЕ "ЗАПОЛНЕНО'                 */
                      THEN STATUS = 1;
/* ИНАЧЕ ПЕРЕВЕСТИ СТАТУС В СОСТОЯНИЕ "НЕЗАПОЛНЕНО"            */
                      ELSE STATUS = 0;
/* ЕСЛИ СИМВОЛ СОДЕРЖИТ ЦИФРОВОЙ КОД                           */
                    IF CHARACTER >= '0' AND CHARACTER <= '9'
/*   ТО ВЫПОЛНИТЬ                                              */
                      THEN DO;
/*     ЕСЛИ СТАТУС УКАЗЫВАЕТ ЧТО ЗАПИСЬ НЕ ЗАПОЛНЕНА          */
                        IF STATUS = 0
```

```
/*      ТО ВЫЗОВ: ДОБАВЛЕНИЕ К ВХОДНОЙ ЗАПИСИ
        (СИМВОЛ;)                                          */
                        THEN CALL ADD$TO$INPUT$RECORD
                        (CHARACTER);
/*      ИНАЧЕ ВЫЗОВ: СИГНАЛ (;)                           */
                        ELSE CALL BEEP;
/*    КОНЕЦ                                                */
```

Другие высокоуровневые языки, используемые в микрокомпьютерах

В приложении содержатся примеры конвертирования конструкций языка проектирования в предложения языков Бейсик, Фортран и Паскаль. Все три языка используются в микрокомпьютерных системах, при этом Бейсик чаще применяется в персональных микрокомпьютерах, Фортран широко используется для научных и инженерных расчетов в ЭВМ любых типов, а Паскаль получил распространение как проблемно-ориентированный язык для микрокомпьютеров.

Бейсик

Комментарии в Бейсике обозначаются словом REMARK, которое должно быть первым в строке. Поскольку значимыми являются только первые три символа, для обозначения комментария достаточно использования REM. Каждой строке, включая строку комментария, должен предшествовать порядковый номер строки. Как уже говорилось в гл. 7, это затрудняет создание и модификацию исходных текстов программы. Строка без номера воспринимается как команда. Команда RUN, например, вызывает немедленное выполнение программы.

Некоторые версии Бейсика ограничивают число символов в слове шестью (иногда и менее) знаками. В таких случаях многие имена необходимо заменять короткими мнемоническими наименованиями. Например, имя структуры данных CONTROL MATRIX (УПРАВЛЯЮЩАЯ МАТРИЦА), которое будет использовано ниже, можно сократить до CNTRLM или даже до CM.

Описание модуля. Бейсик не является модульным языком. Большинство версий Бейсика предусматривает разбиение программ на процедуры, называемые *подпрограммами.* Однако многие из этих версий требуют при этом, чтобы все подпрограммы обрабатывались одновременно с целью уменьшения потоков информации между подпрограммами. Таким образом, программа состоит из единственного *модуля* и все данные являются *модульными* по определению. Следовательно, описания модуля не требуется.

Описание данных. Во многих версиях Бейсика не требуется явного определения переменных. Каждая переменная может принимать численные значения, состоящие из целой и дробной частей (например, число 25,478). Исключением является *строковая* переменная, которая представляет последовательность алфавитно-цифровых символов. Строковая переменная обозначается знаком доллара в конце имени переменной. Например, переменная FILE может содержать числовое значение, представляющее номер файла, в то время как строковая переменная FILE$ содержит строку символов, которая может представлять имя файла. Из этого следует, что FILE и FILE$ представляют различные переменные и могут быть одновременно использованы в одной и той же программе. В некоторых версиях Бейсика используются другие типы данных. В этом случае требуется описание данных.

Массивы в Бейсике описываются следующим образом:

```
10 REM — СТРУКТУРА ДАННЫХ: ПОСЛЕДОВАТЕЛЬНОСТЬ ЧИСЕЛ
20 REM
30 REM — ИМЯ        РАЗМЕР ТИП    СОДЕРЖАНИЕ ПРИМЕЧАНИЯ
40 REM — ---        ------ ---    ---------- ----------
50 REM — ПОСЛЕДО-     20    ЦЕЛОЕ ЧИСЛО
60 REM   ВАТЕЛЬ-
70 REM   НОСТЬ
80 REM   ЧИСЕЛ
90                                  DIM SERNO (20)
```

Конструкция Бейсика DIM (сокращенное от DIMENSION) описывает массив из двадцати последовательных чисел. В зависимости от версии Бейсика индекс, который определяет конкретный элемент массива, может принимать значения в диапазоне от нуля или от единицы до величины, определяющей размер массива в описании данных. Так, в приведенном выше примере двадцать элементов массива определяются в зависимости от используемой версии либо как SERNO (0), ..., SERNO (19), либо как SERNO (1), ..., SERNO (20).

Многие версии Бейсика позволяют описать двумерные и даже трехмерные массивы, что позволяет хранить и обрабатывать информацию в матричной форме:

```
10 REM — СТРУКТУРА ДАННЫХ: УПРАВЛЯЮЩАЯ МАТРИЦА
20 REM
30 REM — ИМЯ        РАЗМЕР ТИП    СОДЕРЖАНИЕ ПРИМЕЧАНИЯ
40 REM — ---        ------ ---    ---------- ----------
50 REM — УПРА-      10×2   ЦЕЛОЕ ПАРАМЕТРЫ
60 REM   ВЛЯЮЩАЯ                  ДВИГАТЕЛЯ
70 REM — МАТРИЦА
80                                  DIM CNTRLM (10, 2)
```

Описание данных с инициализацией. Для инициализации структуры данных необходимо использовать цикл, содержащий оператор READ, и предусмотреть в одном или нескольких операторах DATA значения величин, которые должны быть *прочитаны*. Циклическая конструкция Бейсика будет рассмотрена ниже.

```
 10 REM — СТРУКТУРА ДАННЫХ: ПОСЛЕДОВАТЕЛЬНОСТЬ ЧИСЕЛ
 20 REM
 30 REM — ИМЯ        РАЗМЕР ТИП    СОДЕРЖАНИЕ ПРИМЕЧАНИЯ
 40 REM — ---        ------ ---    ---------- ----------
 50 REM — ПОСЛЕДО-    20     ЦЕЛОЕ ЧИСЛО
 60 REM   ВАТЕЛЬ-
 70 REM   НОСТЬ
 80 REM   ЧИСЕЛ
 90               DIM SERNO (20)
100               FOR INDEX = 1 TO 20
110                 READ SERNO (INDEX)
120               NEXT I
130               DATA 110, 115, 125, 140, 160, 176, 188, 198, 206, 210
140               DATA 217, 222, 234, 238, 251, 263, 271, 284, 289, 295
```

Описание процедур. В Бейсике используется два типа процедур, называемых *подпрограммами*. В более общем типе подпрограмм не разрешается передача входных и выходных параметров между вызывающей и вызываемой процедурами. Начало подпрограммы определяется только номером строки, никаких описаний процедуры не требуется, конец подпрограммы определяется оператором RETURN:

```
1000 REM — ПРОЦЕДУРА: СИГНАЛ (;)
1010 REM — НАЧАЛО ПРОЦЕДУРЫ
1020 REM —   (первая строка процедуры)
1030                                  (первый оператор процедуры)
  •
  •
  •
1080 REM —   ВОЗВРАТ
1090                                  RETURN
1110 REM — КОНЕЦ ПРОЦЕДУРЫ
```

В другом типе подпрограмм Бейсика передача параметров между вызывающей и вызываемой процедурами разрешена, но не обязательна. Использование такой процедуры без передачи параметров иллюстрируется следующим примером:

```
2000 REM — ПРОЦЕДУРА: СИГНАЛ (;)
2010                                  SUB BEEP
  •
  •
  •
```

```
2080 REM — КОНЕЦ ПРОЦЕДУРЫ
2090                                    SUBEND
```

Описание процедуры с входным и выходным параметрами.

```
3000 REM — ПРОЦЕДУРА: ПОИСК (КЛЮЧ ПОИСКА; ЗАПИСЬ)
3010                                    SUB SEARCH (SRCHKY, RECORD)
  •
  •
  •
3130 REM — КОНЕЦ ПРОЦЕДУРЫ
3140                                    SUBEND
```

При вызове процедуры SEARCH указатели переменных, определенные в вызывающей процедуре, пересылаются в вызываемую процедуру. Эти указатели приравнивают переменные SRCHKY и RECORD переменным, определенным в вызывающей процедуре. Следовательно, при изменении RECORD меняется соответствующее данное и в вызывающей процедуре, и таким образом информация, содержащаяся в выходных параметрах, передается в вызывающую процедуру. Необходимо отметить, что, если в вызываемой процедуре меняется значение входного параметра, соответствующее данное в вызывающей процедуре также меняется. Поэтому, если в вызываемой процедуре входной параметр должен обрабатываться, необходимо вначале скопировать его в локальный параметр, чтобы устранить возможность возникновения ошибки.

Описание процедуры со структурами данных в качестве входного и выходного параметров.

```
4000 REM — ПРОЦЕДУРА: ПЕЧАТЬ (ВХОДНАЯ ЗАПИСЬ;)
4010                                    SUB PRINT (INREC ( ))
  •
  •
  •
4080 REM — КОНЕЦ ПРОЦЕДУРЫ
4090                                    SUBEND
```

Пустые скобки после INREC означают, что в процедуру PRINT передается в качестве входного параметра одномерный массив. Скобки с запятыми внутри, например (,) и (,,), означают, что в процедуру передается двумерный или трехмерный массив. Размерность массива в описании процедуры не указывается. Поскольку в Бейсике нет различия между входными и выходными параметрами, передача массивов, являющихся выходными параметрами, осуществляется таким же способом.

Управляющие конструкции. Для вызова процедур первого типа используется конструкция Бейсика GOSUB. Как указывалось в разделе описания процедур, в этом случае требуется конструкция RETURN. Для вызова процедур второго типа

используется конструкция Бейсика CALL. В данном случае конструкция RETURN не требуется, она заменяется конструкцией SUBEND, которая обозначает конец процедуры и вызывает завершение ее выполнения.

```
200 REM — ВЫЗОВ: СИГНАЛ (:)
210                         GOSUB 1000
300 REM — ВЫЗОВ: СИГНАЛ (;)
310                         CALL BEEP
400 REM — ВЫЗОВ: ПОИСК (КЛЮЧ ПОИСКА; ЗАПИСЬ)
410                         CALL SEARCH (SRCHKY, RECORD)
500 REM — ВЫЗОВ: ПОИСК (ИД НОМЕР ЗАПИСЬ)
510                         CALL FIND (SERNO ( ), RECORD)
1080 REM — ВОЗВРАТ
1090                        RETURN
1110 REM — КОНЕЦ ПРОЦЕДУРЫ
1180 REM — КОНЕЦ ПРОЦЕДУРЫ
1190                        SUBEND
```

Конструкции присваивания. Использование слова LET в конструкциях присваивания не обязательно, что иллюстрируется первыми двумя примерами:

```
310 REM — ПЕРЕВЕСТИ СТАТУС В СОСТОЯНИЕ «ЗАПОЛНЕНО»
320             LET STATUS = 1
310 REM — ПЕРЕВЕСТИ СТАТУС В СОСТОЯНИЕ «ЗАПОЛНЕНО»
320             STASUS = 1
370 REM — ПЕРЕВЕСТИ СТАТУС В СОСТОЯНИЕ «НЕ ЗАПОЛНЕНО»
380             LET STATUS = 0
510 REM — УСТАНОВИТЬ ПРОДОЛЖЕНИЕ
520             LET CONTS = 1
570 REM — СБРОСИТЬ ПРОДОЛЖЕНИЕ
580             LET CONTS = 0
800 REM — УСТАНОВИТЬ ВРЕМЯ В СЕКУНДАХ
810             LET TIME = 3600 * HOURS + 60 * MINUTS + SECOND
```

Конструкции цикла. В Бейсике не имеется конструкций типа ВЫПОЛНИТЬ... КОНЕЦ или ВЫПОЛНЯТЬ ПОКА... КОНЕЦ. В некоторых версиях Бейсика конструкция ВЫПОЛНИТЬ... КОНЕЦ при использовании в условных конструкциях может быть реализована путем размещения операций, которые должны находиться внутри конструкции ВЫПОЛНИТЬ... КОНЕЦ, в одной строке исходного текста Бейсика. Поскольку это не всегда возможно, рекомендуется использовать команду *безусловного перехода* или GOTO, как было показано в гл. 6 при конвертировании выражений на языке проектирования в

язык ассемблера. Конструкция ВЫПОЛНЯТЬ ПОКА ... КОНЕЦ может быть смоделирована путем использования условной конструкции с командой *безусловного перехода* или GOTO. Примеры этих реализаций приводятся в разделе условных конструкций. Ниже показаны примеры конструкций ВЫПОЛНИТЬ ДЛЯ КАЖДОГО... КОНЕЦ и ВЫПОЛНЯТЬ НЕПРЕРЫВНО... КОНЕЦ.

```
900 REM — ВЫПОЛНИТЬ ДЛЯ КАЖДОГО КОНТАКТНОГО ДЕТЕКТОРА
910                                   FOR INDEX = 1 TO 5
 •
 •
 •
980 REM — КОНЕЦ
990                                   NEXT INDEX
```

Как видно из примера, в Бейсике для обозначения конца цикла ВЫПОЛНИТЬ ДЛЯ КАЖДОГО... КОНЕЦ используется фраза NEXT... .

```
1000 REM — ВЫПОЛНЯТЬ НЕПРЕРЫВНО
 •
 •
 •
1070 REM — КОНЕЦ
1080                                  GOTO 1000
```

Отметим, что в операторе Бейсика GOTO в качестве метки используется ссылка на строку, содержащую комментарий. При выполнении комментарий игнорируется и выполняется следующий оператор Бейсика.

Условные конструкции.

```
1100 REM — ЕСЛИ ПЕРЕКЛЮЧАТЕЛЬ ВКЛЮЧЕН
1110 REM —   ТО УСТАНОВИТЬ СТАТУС
1120               IF KEYSW < > 0 THEN LET STATUS = 1
1200 REM — ЕСЛИ ПЕРЕКЛЮЧАТЕЛЬ ВЫКЛЮЧЕН
1210 REM —   ТО СБРОСИТЬ СТАТУС
1220               IF KEYSW = 0 THEN LET STATUS = 0
1300 REM — ЕСЛИ ПЕРЕКЛЮЧАТЕЛЬ ВКЛЮЧЕН И ТАЙМЕР
1310 REM   В СОСТОЯНИИ ПОКОЯ
1320 REM —   ТО УСТАНОВИТЬ СТАТУС
1330               IF KEYSW < > 0 AND TIMST = 0
1340                                 THEN LET STATUS = 1
1400 REM — ЕСЛИ ИНДИКАТОР ЗАПОЛНЕНИЯ ПОКАЗЫВАЕТ ЧТО
1410 REM   ВХОДНАЯ ЗАПИСЬ ЗАПОЛНЕНА
1420 REM —   ТО ПЕРЕВЕСТИ СТАТУС В СОСТОЯНИЕ «ЗАПОЛНЕНО»
1430 REM —   ИНАЧЕ ПЕРЕВЕСТИ СТАТУС В СОСТОЯНИЕ
1440 REM     «НЕ ЗАПОЛНЕНО»
1450               IF FIND = 0 THEN STATUS = 1 ELSE STATUS = 0
```

В некоторых версиях Бейсика не разрешается использовать часть ELSE условной конструкции. В таком случае для реализации конструкции ЕСЛИ... ТО... ИНАЧЕ необходимо использовать команды *безусловного перехода* или GOTO, как показано ниже.

```
1500 REM — ЕСЛИ ИНДИКАТОР ЗАПОЛНЕНИЯ ПОКАЗЫВАЕТ ЧТО
1510  REM  ВХОДНАЯ ЗАПИСЬ ЗАПОЛНЕНА
1520                                IF FINDR = 10 THEN GOTO 1540
1530                                  GOTO 1570
1540 REM —   ТО ПЕРЕВЕСТИ СТАТУС В СОСТОЯНИЕ «ЗАПОЛНЕНО»
1550                                  STATUS = 1
1560                                  GOTO 1600
1570 REM —   ИНАЧЕ ПЕРЕВЕСТИ СТАТУС В СОСТОЯНИЕ
1580 REM     «НЕ ЗАПОЛНЕНО»
1590                                  STATUS = 0
1600 (следующая строка процедуры)
```

Следующий пример иллюстрирует использование GOTO для реализации фразы ИНАЧЕ и конструкции ВЫПОЛНИТЬ ... КОНЕЦ.

```
1600 REM — ЕСЛИ СИМВОЛ СОДЕРЖИТ ЦИФРОВОЙ КОД
1610                IF CHAR$ >= «0» OR CHAR$ <= «9»
1620          THEN  GOTO 1640
1630                GOTO 1750
1640 REM —   ТО ВЫПОЛНИТЬ
1650 REM —      ЕСЛИ СТАТУС УКАЗЫВАЕТ ЧТО ЗАПИСЬ
1660 REM        НЕ ЗАПОЛНЕНА
1670                IF STATUS = 0 THEN GOTO 1690
1680                  GOTO 1730
1690 REM —        ТО ВЫЗОВ: ДОБАВЛЕНИЕ К ВХОДНОЙ ЗАПИСИ
1700 REM          (СИМВОЛ;)
1710                  CALL ADDREC (CHAR$)
1720                  GOTO 1750
1730 REM —        ИНАЧЕ ВЫЗОВ: СИГНАЛ (;)
1740                  CALL BEEP
1750 REM —    КОНЕЦ
1760 (следующая строка процедуры)
```

Имитация конструкции ВЫПОЛНЯТЬ ПОКА... КОНЕЦ с использованием условных конструкций иллюстрируется на следующих примерах:

```
1800 REM — ВЫПОЛНЯТЬ ПОКА ПЕРЕКЛЮЧАТЕЛЬ ВКЛЮЧЕН
1810                              IF KEYSW < > 0 THEN GOTO 1830
1820                                GOTO 1890
1830 (первая строка блока)
  •
  •
  •
1870 REM — КОНЕЦ
```

```
1880                                    GOTO 1810
1890 (следующая строка процедуры)
2000 REM — ВЫПОЛНЯТЬ ПОКА ПЕРЕКЛЮЧАТЕЛЬ ВЫКЛЮЧЕН
2010                                    IF KEYSW = 0 THEN GOTO 2030
2020                                    GOTO 2090
2030 (первая строка блока)
  •
  •
  •
2070 REM — КОНЕЦ
2080                                    GOTO 2010
2090 (следующая строка процедуры)
```

Фортран

В этом разделе обсуждается последняя версия Фортрана, известная под официальным названием Фортран 77. Читатель должен заметить, что между старыми версиями Фортрана (Фортран II и Фортран IV) и новой версией обычно не делается различия, для всех трех используется общее наименование Фортран.

Комментарии в Фортране обозначаются буквой С на месте первого символа строки. Позиции 1—5 каждой строки, не являющейся комментарием, могут использоваться в качестве метки, состоящей из чисел от 1 до 99999. Использование меток в Фортране будет проиллюстрировано далее. Шестая позиция каждой строки обычно оставляется пустой. Если предложение Фортрана содержит столько символов, что они не умещаются в одной строке, часть предложения может быть размещена на следующей строке. Для обозначения того, что две или несколько строк должны интерпретироваться как единое предложение, шестая позиция второй и последующих строк не должна быть пустой.

Компиляторы Фортрана часто ограничивают число символов в слове до шести. В таком случае многие имена необходимо заменять короткими мнемоническими сочетаниями. Так, имя структуры данных CONTROL MATRIX (УПРАВЛЯЮЩАЯ МАТРИЦА), которое уже использовалось в примерах, может быть сокращено до CNTRLM. Если возможно использовать более длинные слова и имена, часто используется символ подчеркивания для объединения слов в одно слово или имя, например CONTROL_MATRIX. Оба этих соглашения будут использоваться в примерах данного раздела.

Описание модуля. Фортран не является модульным языком в том смысле, что модуль должен состоять из набора процедур, имеющих доступ к модульным структурам данных. Однако в

нем допускается разбиение программ на процедуры, называемые *подпрограммами*. Программист может организовать группы этих процедур в модули с целью проектирования и документирования, но язык сам по себе не поддерживает модульность. Поэтому в описании модулей нет необходимости.

Описание данных. Многие версии Фортрана не требуют явного описания переменных. Если переменная начинается с одной из букв от I до N, она рассматривается как положительное или отрицательное целое (INTEGER). Число битов, используемое для представления целого, зависит от конкретной ЭВМ и версии компилятора. Переменная, начинающаяся с любой другой буквы, рассматривается как положительное или отрицательное действительное (REAL) число, т. е. такое, которое содержит целую и дробную части, например 25.478. Если компилятор Фортрана допускает использование других *типов* переменных, кроме INTEGER или REAL, описание данных обязательно.

```
C — ВЫХОДНОЙ ПАРАМЕТР: СТАТУС
C —
C — ИМЯ       РАЗМЕР   ТИП    СОДЕРЖАНИЕ    ПРИМЕЧАНИЯ
C — ---       ------   ---    ----------    ----------
C — СТАТУС      1      БАЙТ   ФЛАЖОК
                                        BYTE STATUS
```

Массивы в Фортране должны быть *обязательно* описаны.

```
C — СТРУКТУРА ДАННЫХ: ЗАПИСЬ
C —
C — ИМЯ       РАЗМЕР ТИП      СОДЕРЖАНИЕ      ПРИМЕЧАНИЯ
C — ---       ------ ---      ----------      ----------
C — ЗАПИСЬ      10   СИМВОЛ   ЦИФРОВЫЕ КОДЫ
                                    CHARACTER RECORD (10)
```

Это описание определяет массив размером десять байт, который может быть использован для хранения только символьных кодов. Индекс, используемый в Фортране для спецификации элементов массива, обычно меняется в диапазоне от единицы до величины размера массива, указанной в описании массива. Существуют и другие способы определения диапазона набора индексов, но мы не будем останавливаться на этом. Таким образом, будем считать, что элементы массива в нашем примере обозначаются как RECORD (1), ..., RECORD (10).

В некоторых версиях Фортрана допускается другой способ описания массива, например:

```
BYTE RECORD (10)
```

В этом случае тип хранимой информации не ограничивается символьными кодами, допускается хранение любой 8-битовой величины.

Массивы Фортрана могут быть описаны как двумерные и трехмерные, что означает, что информация может храниться в них в матричной форме.

```
C — СТРУКТУРА ДАННЫХ: УПРАВЛЯЮЩАЯ МАТРИЦА
C —
C — ИМЯ              РАЗМЕР ТИП    СОДЕРЖАНИЕ ПРИМЕЧАНИЯ
C — ---              ------ ---    ---------- ----------
C — УПРАВЛЯЮЩАЯ  10 × 2  ЦЕЛОЕ ПАРАМЕТРЫ
C — МАТРИЦА                     ДВИГАТЕЛЯ
                      INTEGER CONTROL_MATRIX (10 × 2)
```

Описание данных с инициализацией.

```
C — СТРУКТУРА ДАННЫХ: УПРАВЛЯЮЩАЯ МАТРИЦА
C —
C — ИМЯ              РАЗМЕР ТИП    СОДЕРЖАНИЕ ПРИМЕЧАНИЯ
C — ---              ------ ---    ---------- ----------
C — УПРАВЛЯЮЩАЯ  10 × 2  ЦЕЛОЕ ПАРАМЕТРЫ
    МАТРИЦА                     ДВИГАТЕЛЯ
                  INTEGER CONTROL_MATRIX (10 × 2)
                  DATA CONTROL_MATRIX (10, 11, 12, 15, 17, 22, 25,
     2            34, 38, 40, 51, 60, 63, 71, 76, 84, 88, 89, 95, 98)
```

Обратите внимание на число 2 внизу слева, которое находится в шестой позиции второй строки оператора DATA. Оно указывает, что две строки вмещают одно предложение Фортрана, о чем указывалось выше.

Описание процедуры.

```
C — ПРОЦЕДУРА: СИГНАЛ (;)
                                SUBROUTINE BEEP
                    •
                    •
                    •
C — КОНЕЦ ПРОЦЕДУРЫ
                                END
```

Описание процедуры с входным параметром.

```
C — ПРОЦЕДУРА: ДОБАВЛЕНИЕ К ВХОДНОЙ ЗАПИСИ (СИМВОЛ;)
                                SUBROUTINE ADDREC (CHR)
C — КОНЕЦ ПРОЦЕДУРЫ
                                END
```

Описание процедуры с выходным параметром. Процедура с единственным выходным параметром может быть реализована

путем использования процедуры FUNCTION, как показано на следующем примере:

```
C — ПРОЦЕДУРА: ПРОВЕРКА ЗАПОЛНЕНИЯ ВХОДНОЙ ЗАПИСИ
C — (;СТАТУС)
                                        FUNCTION TSTFUL
                                        INTEGER TSTFUL
      •                                      •
      •                                      •
      •                                      •
                                        TSTFUL = STATUS
                                        RETURN
                                             •
                                             •
                                             •
C — КОНЕЦ ПРОЦЕДУРЫ
                                        END
```

Имя самой процедуры должно быть описано в процедуре FUNCTION как переменная Фортрана. Этой переменной до того, как завершится выполнение процедуры, должно быть присвоено значение выходного параметра. Описание выходного параметра STATUS, являющегося внешним параметром для процедуры FUNCTION, не показано. Показано только то, как его значение присваивается переменной TSTFUL.

В Фортране не делается различий между входными и выходными параметрами в процедуре SUBROUTINE. Ниже показан пример использования процедуры SUBROUTINE с двумя выходными параметрами:

```
C — ПРОЦЕДУРА: ПРИМЕР С ВХОДОМ И ВЫХОДОМ (ВХОД;
C — ВЫХОД1, ВЫХОД2)
                        SUBROUTINE INOUTX (INPUT, OUTPT1, OUTPT2)
C — **********************************************************************
C — ВХОДНОЙ ПАРАМЕТР: ВХОД
C —
C — ИМЯ     РАЗМЕР  ТИП      СОДЕРЖАНИЕ      ПРИМЕЧАНИЯ
C — ---     ------  ---      ----------      ----------
C — ВХОД      1     СИМВОЛ   ЦИФРОВЫЕ КОДЫ
                                CHARACTER INPUT
C —
C — ВЫХОДНЫЕ ПАРАМЕТРЫ: ВЫХОД1, ВЫХОД2
C —
C — ИМЯ     РАЗМЕР  ТИП      СОДЕРЖАНИЕ      ПРИМЕЧАНИЯ
C — ---     ------  ---      ----------      ----------
C — ВЫХОД1    1     СИМВОЛ   ЦИФРОВЫЕ КОДЫ
C — ВЫХОД2    1     ЦЕЛОЕ    ЦИФРОВЫЕ
C —                          ВЕЛИЧИНЫ
                                CHARACTER OUTPT1
                                INTEGER OUTPT2
```

При вызове процедуры SUBROUTINE INOUTX указатели переменных, определенных в вызывающей процедуре, передаются в вызываемую процедуру. Эти указатели приравнивают переменные INPUT, OUTPT1 и OUTPT2 переменным, определенным в вызывающей процедуре. Поэтому при изменении OUTPT1 и OUTPT2 меняются также соответствующие данные в вызывающей процедуре. Таким образом, информация, содержащаяся в выходных параметрах, может быть передана в вызывающую процедуру. Необходимо заметить, что если меняется значение входного параметра в вызываемой программе, то соответствующая информация в вызывающей программе также меняется. Поэтому если предусмотрена обработка входного параметра в вызываемой программе, то чтобы избежать возможных ошибок, входной параметр необходимо скопировать в локальный параметр.

Описание процедур со структурами данных в качестве входных и выходных параметров. Массивы в Фортране передаются от процедуры к процедуре так же легко, как и отдельные входные или выходные параметры. С целью обеспечения совместимости массив должен быть описан как в вызывающей, так и в вызываемой процедурах. В рассматриваемых ниже примерах приводится описание процедуры с массивом, передаваемым в качестве входного и выходного параметра. Поскольку в Фортране для процедуры SUBROUTINE не делается различий между входными и выходными параметрами, передача выходных параметров, являющихся массивами, осуществляется так же, как и входных.

```
C — ПРОЦЕДУРА: ПЕЧАТЬ (ЗАПИСЬ;)
                    SUBROUTINE PRINT (RECORD)
```

В Фортране существует другой способ доступа из процедур к содержимому структур данных, описанных в другой процедуре, без передачи этих структур в качестве параметров. Это может быть выполнено посредством описания данных как COMMON (ОБЩИЕ данные). Этот способ позволяет обрабатывать структуры данных так, как если бы они были модульными структурами. Но так как модульность в Фортране не поддерживается, ответственность за ограничение доступа к структурам данных, описанных как COMMON, возлагается на программиста, а не на компилятор. Модульное определение структур данных с помощью языка проектирования может помочь программисту контролировать доступ к *модульным* структурам данных при использовании Фортрана.

```
C — ПРОЦЕДУРА: СТИРАНИЕ ВХОДНОЙ ЗАПИСИ (;)
C — ********************************************************************** **
```

```
C — МОДУЛЬНАЯ СТРУКТУРА ДАННЫХ: ВХОДНАЯ ЗАПИСЬ
C —
C — ИМЯ              РАЗМЕР ТИП  СОДЕРЖАНИЕ ПРИМЕЧАНИЯ
C — ---              ------ ---  ---------- ----------
C — ВХОДНАЯ ЗАПИСЬ     10     БАЙТ ЦИФРОВЫЕ
C —                                КОДЫ
                               BYTE INPUT_RECORD (10)
C —  ИНДИКАТОР          1     БАЙТ СЧЕТЧИК
C —  ЗАПОЛНЕНИЯ
                      BYTE FULL_INDICATOR
                      COMMON/INPUT_RECORD/INPUT_RECORD,
                      FULL_INDICATOR
C — **********************************************************************
C — НАЧАЛО ПРОЦЕДУРЫ
C —  ВЫПОЛНИТЬ ДЛЯ КАЖДОГО ЭЛЕМЕНТА ВО ВХОДНОЙ
C —  ЗАПИСИ
C —    УСТАНОВИТЬ ЭЛЕМЕНТ В КОД НУЛЯ
                      INPUT_RECORD (INDEX) = 0
C —  КОНЕЦ
C —  СБРОСИТЬ ИНДИКАТОР ЗАПОЛНЕНИЯ
                      FULL_INDICATOR = 0
C —  ВОЗВРАТ
C — КОНЕЦ ПРОЦЕДУРЫ
C — ПРОЦЕДУРА: ПРОВЕРКА ЗАПОЛНЕНИЯ ВХОДНОЙ ЗАПИСИ
C — (;СТАТУС)
C — **********************************************************************
C — МОДУЛЬНАЯ СТРУКТУРА ДАННЫХ: ВХОДНАЯ ЗАПИСЬ
C —
C — ИМЯ              РАЗМЕР ТИП  СОДЕРЖАНИЕ ПРИМЕЧАНИЯ
C — ---              ------ ---  ---------- ----------
C — ВХОДНАЯ ЗАПИСЬ     10     БАЙТ ЦИФРОВЫЕ
                                   КОДЫ
                      BYTE INPUT_RECORD (10)
C —  ИНДИКАТОР          1     БАЙТ СЧЕТЧИК
C —  ЗАПОЛНЕНИЯ       BYTE FULL_INDICATOR
                      COMMON/INPUT_RECORD/INPUT_RECORD,
                      FULL_INDICATOR
```

Управляющие конструкции.

```
C — ВЫЗОВ: ОСТАНОВКА ТАЙМЕРА (;)
                      CALL STOP_TIMER
C — ВЫЗОВ: ДОБАВЛЕНИЕ К ВХОДНОЙ ЗАПИСИ (СИМВОЛ;)
                      CALL ADDREC (CHR)
C — ВЫЗОВ: ПРОВЕРКА ЗАПОЛНЕНИЯ ВХОДНОЙ ЗАПИСИ
C — (;СТАТУС)
                      STATUS = TSTFUL
C — ВЫЗОВ: ПРИМЕР С ВХОДОМ И ВЫХОДОМ (ВХОД; ВЫХОД1,
```

```
C — ВЫХОД2)
                        CALL INOUTX (INPUT, OUTPT1, OUTPT2)
C — ВОЗВРАТ
                        RETURN
```

Конструкции присваивания.

```
C — ПЕРЕВЕСТИ СТАТУС В СОСТОЯНИЕ «ЗАПОЛНЕНО»
                    STATUS = 1
C — ПЕРЕВЕСТИ СТАТУС В СОСТОЯНИЕ «НЕ ЗАПОЛНЕНО»
                    STATUS = 0
C — УСТАНОВИТЬ ПРОДОЛЖЕНИЕ
                    CONTINUOUS = 1
C — СБРОСИТЬ ПРОДОЛЖЕНИЕ
                    CONTINUOUS = 0
C — УСТАНОВИТЬ ВРЕМЯ В СЕКУНДАХ
                    TIMF = 3600 * HOURS + 60 * MINUTES + SECONDS
```

Конструкции цикла. В Фортране нет конструкций типа ВЫПОЛНИТЬ... КОНЕЦ или ВЫПОЛНЯТЬ ПОКА... КОНЕЦ. В качестве эквивалента конструкции ВЫПОЛНИТЬ... КОНЕЦ в условных конструкциях могут служить конструкции THEN и ELSE вместе с конструкцией ENDIF. Конструкция ВЫПОЛНЯТЬ ПОКА... КОНЕЦ может быть смоделирована с использованием условной конструкции с командой *безусловного перехода* или GOTO. Примеры этих конструкций приводятся в разделе условных конструкций. Ниже показаны примеры конструкций ВЫПОЛНИТЬ ДЛЯ КАЖДОГО... КОНЕЦ и ВЫПОЛНЯТЬ НЕПРЕРЫВНО... КОНЕЦ.

```
C — ВЫПОЛНИТЬ ДЛЯ КАЖДОГО КОНТАКТНОГО ДЕТЕКТОРА
                                        DO 100 INDEX = 1,5
                    .
                    .
                    .
C — КОНЕЦ
100                                     CONTINUE
```

Как видно из примера, метка в Фортране (в данном случае 100) используется для обозначения конца цикла ВЫПОЛНИТЬ ДЛЯ КАЖДОГО... КОНЕЦ. Конструкция Фортрана CONTINUE не обозначает каких-либо действий. Вместе с меткой она обозначает некоторую конкретную точку в процедуре.

```
C — ВЫПОЛНЯТЬ НЕПРЕР .ВНО
200                                     CONTINUE
                    .
                    .
                    .
C — КОНЕЦ                               GOTO 200
```

Условные конструкции. Ниже в таблице перечислены отношения и логические операторы, используемые в условных конструкциях Фортрана:

Оператор	Значение
.EQ.	равно
.LT.	меньше, чем
.GT.	больше, чем
.LE.	меньше или равно
.GE.	больше или равно
.NE.	не равно
.NOT.	не
.AND.	и
.OR.	или

```
C — ЕСЛИ ПЕРЕКЛЮЧАТЕЛЬ ВКЛЮЧЕН
                                    IF (KEYSW.NE.0) THEN
C —   ТО  УСТАНОВИТЬ СТАТУС
                                      STATUS = 1
                                    ENDIF
C — ЕСЛИ ПЕРЕКЛЮЧАТЕЛЬ ВЫКЛЮЧЕН
                                    IF (KEYSW.EQ.0) THEN
C —   ТО  ВЫПОЛНИТЬ
C —       СБРОСИТЬ СТАТУС
                                      STATUS = 0
C —       ВОЗВРАТ
                                      RETURN
C —       КОНЕЦ
                                    ENDIF
```

Так как в Фортране нет конструкции ВЫПОЛНИТЬ... КОНЕЦ, конструкции THEN и ENDIF определяют пару *скобок*, выполняющих роль ВЫПОЛНИТЬ... КОНЕЦ для условной конструкции ЕСЛИ... ТО. Когда в Фортране применяется конструкция ЕСЛИ... ТО, использование конструкции ENDIF обязательно, даже если после THEN следует единственная операция, как показано в первом из вышеприведенных примеров. Другой способ реализации конструкции ЕСЛИ... ТО с единственной операцией после THEN показан в первом примере ниже.

```
C — ЕСЛИ ПЕРЕКЛЮЧАТЕЛЬ ВКЛЮЧЕН
C —   ТО УСТАНОВИТЬ СТАТУС
                                    IF (KEYSW.NE.0) STATUS = 1
```

```
C — ЕСЛИ ИНДИКАТОР ЗАПОЛНЕНИЯ ПОКАЗЫВАЕТ ЧТО ВХОДНАЯ
C — ЗАПИСЬ ЗАПОЛНЕНА
                                    IF (FINDR.EQ.10) THEN
C —   ТО ПЕРЕВЕСТИ СТАТУС В СОСТОЯНИЕ «ЗАПОЛНЕНО»
                                      STATUS = 1
C —   ИНАЧЕ ПЕРЕВЕСТИ СТАТУС В СОСТОЯНИЕ
C —   «НЕ ЗАПОЛНЕНО»
                                    ELSE
                                      STATUS = 0
                                    ENDIF
```

В условной конструкции ЕСЛИ... ТО... ИНАЧЕ (IF... THEN... ELSE) различают две части: THEN и ELSE определяют пару *скобок* для части ТО (THEN) условной конструкции, а ELSE и ENDIF — части ИНАЧЕ (ELSE). *Скобки* THEN, ELSE и ENDIF необходимы даже в том случае, если в соответствующей конструкции языка проектирования *скобки* ВЫПОЛНИТЬ... КОНЕЦ не используются.

```
C — ЕСЛИ СИМВОЛ СОДЕРЖИТ ЦИФРОВОЙ КОД
C —   ТО ВЫПОЛНИТЬ
                              IF (CHR.GE.'0'.AND.CHR.LE.'9') THEN
C —     ЕСЛИ СТАТУС УКАЗЫВАЕТ ЧТО ВХОДНАЯ ЗАПИСЬ
C —     НЕ ЗАПОЛНЕНА
                                IF (STATUS.EQ.0) THEN
C —       ТО ВЫЗОВ: ДОБАВЛЕНИЕ К ВХОДНОЙ ЗАПИСИ
C —       (СИМВОЛ;)
                                  CALL ADDREC (CHR)
C —       ИНАЧЕ ВЫЗОВ: СИГНАЛ (;)
                                ELSE
                                  CALL BEEP
                                ENDIF
C —   КОНЕЦ
                              ENDIF
```

Имитация конструкции ВЫПОЛНЯТЬ ПОКА ... КОНЕЦ с использованием условной конструкции показана на нижеследующих примерах.

```
C — ВЫПОЛНЯТЬ ПОКА ПЕРЕКЛЮЧАТЕЛЬ ВКЛЮЧЕН
300                                 IF (KEYSW.NE.0) GOTO 320
310                                   GOTO 400
320 (ПЕРВАЯ СТРОКА БЛОКА)
                 •
                 •
                 •
C — КОНЕЦ
                                    GOTO 300
400                                 CONTINUE
C — ВЫПОЛНЯТЬ ПОКА ПЕРЕКЛЮЧАТЕЛЬ ВЫКЛЮЧЕН
500                                 IF (KEYSW.EQ.0) GOTO 520
```

```
510                                    GOTO 600
520 (ПЕРВАЯ СТРОКА БЛОКА)
                  •
                  •
                  •
С — КОНЕЦ
                                       GOTO 500
600                                    GONTINUE
```

Паскаль

Комментарии в Паскале обозначаются фигурными скобками { и }. Хотя обычно на число символов в слове или в имени не накладывается никаких ограничений, в некоторых версиях Паскаля воспринимаются только первые восемь символов, и поэтому они должны быть уникальными. Компиляторы Паскаля не делают различия между прописными и строчными символами, однако между пользователями Паскаля существуют некоторые *соглашения* на этот счет. Чтобы не отступать от правил, мы будем использовать прописные символы для конструкций языка проектирования и для большинства ключевых слов Паскаля, например PROCEDURE. Для имен Паскаля будут использоваться строчные символы, однако каждое имя должно начинаться с прописной буквы. Так имя на языке проектирования ОБРАБОТКА ВХОДНОЙ ЗАПИСИ (MAINTAIN INPUT RECORD) конвертируется в имя Паскаля MaintainInputRecord. Таким же образом кодируются идентификаторы Паскаля, за исключением того, что первый символ идентификатора должен быть строчной буквой. Например, идентификатор языка проектирования INPUT RECORD конвертируется в идентификатор Паская inputRecord.

В языке PL/M точка с запятой используется для *завершения* конструкции. В Паскале же точка с запятой используется не столько для завершения, сколько для *разделения* конструкций. Так, если условная конструкция языка проектирования

```
ЕСЛИ
  ТО ВЫПОЛНИТЬ
     КОНЕЦ
  ИНАЧЕ
```

конвертируется в Паскаль, будет неправильным поместить точку с запятой после конструкции, соответствующей КОНЕЦ в языке проектирования. Это приведет к тому, что часть ELSE будет рассматриваться компилятором как отдельная, не имеющая смысла конструкция. Следует отметить, что в некоторых случаях наличие точки с запятой не является обязательным и не приводит к каким-либо трудностям при компиляции. В на-

ших примерах точки с запятой вставляются только там, где это требуется или где существует большая вероятность, что они потребуются в соответствии с контекстом. В остальных случаях они опускаются. Мы отсылаем читателя к одному из описаний или руководств по Паскалю, перечисленных в списке литературы, для ознакомления с синтаксисом Паскаля и деталями использования точки с запятой в программах Паскаля.

Описание модуля.

```
{МОДУЛЬ: ИСПОЛНИТЕЛЬНЫЙ                    }
                                PROGRAM Executive;
                .
                .
                .
{КОНЕЦ МОДУЛЯ                               }
                                END.
```

Отметим, что если слово END обозначает конец модуля, то после него должна стоять точка.

В стандартном Паскале допускается единственный модуль, которым является PASCAL PROGRAM. Однако в Паскале-80 для микрокомпьютера Intel 8085 предусмотрена компиляция отдельных модулей с последующим объединением объектных модулей в результирующую программу. Более того, в системе Intel предусмотрено установление связей между объектными модулями Паскаля, PL/M и Фортрана, поскольку для каждого языка компиляторами Intel используются одни и те же соглашения по передаче параметров. В отличие от стандартного Паскаля с единственным модулем PROGRAM в мультимодульной системе описание ИСПОЛНИТЕЛЬНОГО модуля имеет следующий вид:

```
{МОДУЛЬ: ИСПОЛНИТЕЛЬНЫЙ                    }
                        PARTITIONED PROGRAM Executive;
                .
                .
                .
{КОНЕЦ МОДУЛЯ                               }
                        END.
```

Описание для остальных модулей имеет вид:

```
{МОДУЛЬ: ОБРАБОТКА ВХОДНОЙ ЗАПИСИ          }
                        COMPONENT MaintainInputRecord;
                .
                .
                .
{КОНЕЦ МОДУЛЯ                               }
                        END.
```

Описание данных.

```
{ВЫХОДНОЙ ПАРАМЕТР: СТАТУС                  }
```

```
{                                                              }
{ИМЯ      РАЗМЕР  ТИП      СОДЕРЖАНИЕ ПРИМЕЧАНИЯ}
{---      ------  ---      ---------- ----------}
{СТАТУС     1     ЦЕЛОЕ    ФЛАЖОК                }
                        VAR status: integer;

{МОДУЛЬНАЯ СТРУКТУРА ДАННЫХ: ВХОДНАЯ ЗАПИСЬ    }
{                                                              }
{ИМЯ       РАЗМЕР ТИП      СОДЕРЖАНИЕ ПРИМЕЧАНИЯ}
{---          10    СИМВОЛ ЦИФРОВЫЕ КОДЫ        }
{ВЫХОДНАЯ ЗАПИСЬ  10  СИМВОЛ ЦИФРОВЫЕ КОДЫ }
                  VAR inputRecord: ARRAY [1 ... 10] OF char;
```

Спецификация [1... 10] в примере указывает, что используемый для определения элементов массива индекс меняется в диапазоне от 1 до 10. В описании данных Паскаля может быть определен любой диапазон. Например, выражение

```
VAR inputRecord: ARRAY [ —5 ... 5] OF char;
```

описывает структуру данных inputRecord, содержащую одиннадцать символьных кодов, элементы которой обозначаются как inputRecord(—5), ..., inputRecord(0), ..., inputRecord(5).

Паскаль допускает определение многомерных массивов, что позволяет хранить информацию в матричной форме.

```
{СТРУКТУРА ДАННЫХ: УПРАВЛЯЮЩАЯ МАТРИЦА          }
{                                                              }
{ИМЯ              РАЗМЕР ТИП    СОДЕРЖАНИЕ ПРИМЕЧАНИЯ}
{---              ------ ---    ---------- ----------}
{УПРАВЛЯЮЩАЯ  10×2  ЦЕЛОЕ ПАРАМЕТРЫ                }
{МАТРИЦА                    ДВИГАТЕЛЯ                }
                VAR controlMatrix: ARRAY [1 .. 10, 1 .. 2] OF integer;
```

Описание данных с инициализацией. В Паскале не предусмотрено простых способов инициализации целых структур данных. Необходимо для инициализации каждого элемента массива применять отдельную конструкцию присваивания, как показано в следующем примере:

```
{СТРУКТУРА ДАННЫХ: УПРАВЛЯЮЩАЯ МАТРИЦА          }
{                                                              }
{ИМЯ              РАЗМЕР ТИП    СОДЕРЖАНИЕ ПРИМЕЧАНИЯ}
{---              ------ ---    ---------- ----------}
{УПРАВЛЯЮЩАЯ  10×2  ЦЕЛОЕ ПАРАМЕТРЫ                }
{МАТРИЦА                    ДВИГАТЕЛЯ                }
              VAR controlMatrix: ARRAY [1 .. 10, 1 .. 2] OF integer;
                •
                •
                •
              BEGIN
                controlMatrix [1, 1] := 5;
```

```
controlMatrix [10, 2] := 3
controlMatrix [2, 1] := 7;
controlMatrix [10, 2] := 4
•
•
•
```

Описание процедур.

```
{ПРОЦЕДУРА: СТИРАНИЕ ВХОДНОЙ ЗАПИСИ (;)                    }
                                    PROCEDURE ClearInputRecord;
{***************************************************************}
{НАЧАЛО ПРОЦЕДУРЫ                                           }
                                    BEGIN
        •
        •
{КОНЕЦ ПРОЦЕДУРЫ                                           }
                                    END;
```

Описание процедуры с входным параметром.

```
{ПРОЦЕДУРА: ДОБАВЛЕНИЕ К ВХОДНОЙ ЗАПИСИ (СИМВОЛ;)          }
{***************************************************************}
{ВХОДНОЙ ПАРАМЕТР: СИМВОЛ                                   }
{                                                           }
{ИМЯ     РАЗМЕР ТИП      СОДЕРЖАНИЕ      ПРИМЕЧАНИЯ        }
{---     ------ ---      ----------      ----------        }
{СИМВОЛ    1    СИМВОЛ ЦИФРОВЫЕ КОДЫ                        }
                    PROCEDURE AddToInputRecord (character: char);
{***************************************************************}
{НАЧАЛО ПРОЦЕДУРЫ                                           }
                    BEGIN
        •
        •
        •
{КОНЕЦ ПРОЦЕДУРЫ                                           }
                    END·
```

Как показано в примере, описание входного параметра процедуры объединено с описанием самой процедуры. Отдельного описания входного параметра не требуется.

Описание процедур с выходными параметрами. Процедура с единственным выходным параметром может быть реализована в Паскале с использованием процедуры FUNCTION, как показано в следующем примере:

```
{ПРОЦЕДУРА: ПРОВЕРКА ЗАПОЛНЕНИЯ ВХОДНОЙ ЗАПИСИ            }
{(;СТАТУС)                                                  }
                                    FUNCTION TestIfInputRecordFull: integer;
{***************************************************************}
{НАЧАЛО ПРОЦЕДУРЫ
```

```
                        BEGIN
                          TestIfInputRecordFull := status;
                            .
                            .
                            .
{КОНЕЦ ПРОЦЕДУРЫ                                          }
                        END;
```

При описании процедуры как FUNCTION имя процедуры автоматически объявляется переменной Паскаля (в данном случае — как целое). Эта переменная должна быть приравнена величине выходного параметра до завершения выполнения процедуры. Описание выходного параметра STATUS, являющегося локальным параметром, не показано. Показано только, как используется STATUS, когда его значение приравнивается переменной TestIfInputRecordFull.

Ниже показан пример использования процедуры типа FUNCTION с входным и выходным параметрами.

```
{ПРОЦЕДУРА: ПОИСК (КЛЮЧ ПОИСКА; ЗАПИСЬ)                   }
{*********************************************************}
{ВХОДНОЙ ПАРАМЕТР: КЛЮЧ ПОИСКА                            }
{                                                         }
{ИМЯ            РАЗМЕР ТИП    СОДЕРЖАНИЕ   ПРИМЕЧАНИЯ}
{---            ------ ---    ----------   ----------}
{КЛЮЧ ПОИСКА      1    ЦЕЛОЕ ЦИФРОВОЙ КОД              }
                      FUNCTION Search (searchKey: integer): integer;
{*********************************************************}
{НАЧАЛО ПРОЦЕДУРЫ                                         }
                      BEGIN
                          .
                          .
                          .
                        Search := match;
                          .
                          .
                          .
{КОНЕЦ ПРОЦЕДУРЫ                                          }
                      END;
```

Как и в предыдущем примере, описание выходного параметра MATCH опущено, так как он является локальным параметром. Показано только приравнивание MATCH переменной Search.

Процедура с несколькими выходными параметрами может быть реализована с помощью описания процедуры как PROCEDURE и объявления выходных параметров с помощью конструкции VAR, как показано в следующем примере:

```
{ПРОЦЕДУРА: ПРИМЕР С ВХОДОМ И ВЫХОДОМ (ВХОД; ВЫХОД1 }
{ВЫХОД2)
{*************************************************************************}
{ВХОДНОЙ ПАРАМЕТР: ВХОД                                                   }
{                                                                         }
{ИМЯ     РАЗМЕР ТИП     СОДЕРЖАНИЕ     ПРИМЕЧАНИЯ                        }
{---     ------ ---     ----------     ----------                        }
{ВХОД         1 СИМВОЛ ЦИФРОВЫЕ КОДЫ                                      }
{                                                                         }
{ВЫХОДНЫЕ ПАРАМЕТРЫ: ВЫХОД1, ВЫХОД2                                       }
{                                                                         }
{ИМЯ     РАЗМЕР ТИП     СОДЕРЖАНИЕ     ПРИМЕЧАНИЯ                        }
{---     ------ ---     ----------     ----------                        }
{ВЫХОД1       1 СИМВОЛ ЦИФРОВЫЕ КОДЫ                                      }
{ВЫХОД2       1 ЦЕЛОЕ  ЦИФРОВЫЕ КОДЫ                                      }
                        PROCEDURE InputOutputExample
                        (input: char; VAR output1: char;
                                      VAR output2: integer);
```

В этом примере при вызове процедуры InputOutputExample значение входного параметра передается от переменной, определенной в вызывающей процедуре, переменной input в вызываемой процедуре. Поэтому изменение значения переменной input в вызываемой процедуре не влияет на значение переменной в вызывающей процедуре. Использование конструкции VAR для обоих выходных параметров означает, что в вызываемую процедуру пересылаются указатели переменных, определенных в вызывающей процедуре. Эти указатели приравнивают значения переменных output1 и output2 переменным, определенным в вызывающей процедуре. Поэтому при изменении переменных output1 и output2 соответствующие данные в вызывающей процедуре также меняются. В Паскале информация, содержащаяся в нескольких выходных параметрах, может быть передана в вызывающую процедуру.

Описание процедур со структурами данных в качестве входных и выходных параметров.

```
{ПРОЦЕДУРА: ПЕЧАТЬ (ВХОДНАЯ ЗАПИСЬ;)                                     }
{*************************************************************************}
{ВХОДНОЙ ПАРАМЕТР: ВХОДНАЯ ЗАПИСЬ                                         }
{                                                                         }
{ИМЯ              РАЗМЕР ТИП     СОДЕРЖАНИЕ ПРИМЕЧАНИЯ                   }
{---              ------ ---     ---------- ----------                   }
{ВХОДНАЯ ЗАПИСЬ       10 СИМВОЛ ЦИФРОВЫЕ                                 }
{                                КОДЫ                                     }
      PROCEDURE Print (VAR inputRecord: ARRAY [1 .. 10] OF char);
```

Пример иллюстрирует передачу в процедуру массива в качестве входного параметра. Поскольку соглашение о вызове процедур, описанных как PROCEDURE с использованием конструкции VAR для описания данных, не делает различия между входными и выходными параметрами, передача массивов в качестве выходных параметров осуществляется таким же способом.

Описание процедур для мультимодульной системы. В системах, скомпонованных из нескольких модулей, описание процедур в компонентных модулях, вызываемых процедурами из других модулей, принимает следующий вид:

```
{ПРОЦЕДУРА: СТИРАНИЕ ВХОДНОЙ ЗАПИСИ (;)          }
                    SEGMENT PROCEDURE ClearInputRecord;
```

В каждом модуле, содержащем процедуры, вызывающие SEGMENT PROCEDURE, необходимо внешнее описание SEGMENT PROCEDURE. Внешнее описание процедур в Паскале-80 имеет следующий вид:

```
                    SEGMENT PROCEDURE ClearInputRecord;
                    SEPARATE;
```

Управляющие конструкции.

```
{ВЫЗОВ: ОСТАНОВКА ТАЙМЕРА (;)                      }
                    StopTimer;
{ВЫЗОВ: ДОБАВЛЕНИЕ К ВХОДНОЙ ЗАПИСИ (СИМВОЛ;)     }
                    AddToInputRecord (character);
{ВЫЗОВ: ПРОВЕРКА ЗАПОЛНЕНИЯ ВХОДНОЙ ЗАПИСИ (;СТАТУС) }
                    status := TestIfInputRecordFull;
```

В Паскале нет конструкции RETURN. Выполнение процедуры завершается при достижении конструкции END, соответствующей по смыслу конструкции END PROCEDURE. Поэтому необходимо предусмотреть завершение всех путей в конце процедуры. Для этих целей может быть использована конструкция GOTO, как показано ниже в процедуре ОЖИДАНИЕ ВКЛЮЧЕНИЯ ПЕРЕКЛЮЧАТЕЛЯ. В этом примере конвертируется только часть конструкций языка проектирования.

```
{ПРОЦЕДУРА: ОЖИДАНИЕ ВКЛЮЧЕНИЯ ПЕРЕКЛЮЧАТЕЛЯ (;)  }
{НАЧАЛО ПРОЦЕДУРЫ                                 }
  ВЫПОЛНЯТЬ НЕПРЕРЫВНО                            }
{   ВЫЗОВ: СЧИТЫВАНИЕ ПЕРЕКЛЮЧАТЕЛЯ (;ПЕРЕКЛЮЧАТЕЛЬ)}
{   ЕСЛИ ПЕРЕКЛЮЧАТЕЛЬ ВКЛЮЧЕН                    }}
{     ТО ВОЗВРАТ
                                    THEN GOTO 100;
```

```
{ КОНЕЦ                                                    }
                                                      END;
{КОНЕЦ ПРОЦЕДУРЫ                                           }
                                                   100: END
```

Отметим, что меткой в Паскале является число, меняющееся в диапазоне от 1 до 9999. Метка должна быть описана до ее использования таким же образом, как описываются переменные и структуры данных. Описание метки имеет следующий вид:

```
LABEL 100;
```

Если использование конструкции GOTO нежелательно, можно построить процедуру таким образом, что все пути будут оканчиваться в конце процедуры. Примером такого подхода является другая версия процедуры ОЖИДАНИЯ ВКЛЮЧЕНИЯ ПЕРЕКЛЮЧАТЕЛЯ, показанная ниже и рассмотренная ранее (см. рис. 4.11). Поскольку конструкция ВОЗВРАТ непосредственно предшествует конструкции КОНЕЦ ПРОЦЕДУРЫ, то для правильного выполнения процедуры нет необходимости в конвертировании этой конструкции в Паскаль. В остальном конвертирование процедуры не представляет трудности и оставляется читателю в качестве упражнения после того, как будет описано конвертирование циклических и условных конструкций.

```
ПРОЦЕДУРА: ОЖИДАНИЕ ВКЛЮЧЕНИЯ ПЕРЕКЛЮЧАТЕЛЯ (;)
НАЧАЛО ПРОЦЕДУРЫ
  ВЫЗОВ: СЧИТЫВАНИЕ ПЕРЕКЛЮЧАТЕЛЯ (;ПЕРЕКЛЮЧАТЕЛЬ)
  ВЫПОЛНЯТЬ ПОКА ПЕРЕКЛЮЧАТЕЛЬ ВЫКЛЮЧЕН
    ВЫЗОВ: СЧИТЫВАНИЕ ПЕРЕКЛЮЧАТЕЛЯ (;ПЕРЕКЛЮЧАТЕЛЬ)
  КОНЕЦ
  ВОЗВРАТ
КОНЕЦ ПРОЦЕДУРЫ
```

Конструкции присваивания.

```
{ПЕРЕВЕСТИ СТАТУС В СОСТОЯНИЕ "ЗАПОЛНЕНО"     }
                        status := 1;
{ПЕРЕВЕСТИ СТАТУС В СОСТОЯНИЕ "НЕ ЗАПОЛНЕНО"  }
                        status := 0;
{УСТАНОВИТЬ ПРОДОЛЖЕНИЕ                       }
                        continuous := 1;
{СБРОСИТЬ ПРОДОЛЖЕНИЕ                         }
                        continuous := 0;
{УСТАНОВИТЬ ВРЕМЯ В СЕКУНДАХ                  }
                        time := 3600 * hours + 60 * minutes + second;
```

Конструкции цикла.

```
{ВЫПОЛНИТЬ                                          }
                                          BEGIN
          .
          .
          .
{КОНЕЦ                                             }
                                          END:
{ВЫПОЛНЯТЬ ПОКА ПЕРЕКЛЮЧАТЕЛЬ ВКЛЮЧЕН              }
                                          WHILE keyswitch < > 0 DO
                                            BEGIN
          .
          .
          .
{КОНЕЦ                                             }
                                            END;
```

Если внутри конструкции ВЫПОЛНЯТЬ ПОКА... КОНЕЦ находится единственная конструкция, конструкция Паскаля BEGIN... END может быть опущена, как показано в следующем примере:

```
{ВЫПОЛНЯТЬ ПОКА ПЕРЕКЛЮЧАТЕЛЬ ВЫКЛЮЧЕН                  }
                                          WHILE keyswitch = 0 DO
{ВЫЗОВ: СЧИТЫВАНИЕ ПЕРЕКЛЮЧАТЕЛЯ (;ПЕРЕКЛЮЧАТЕЛЬ)}
                                            keyswitch := ReadKeyswitch
{КОНЕЦ                                                 }
{ВЫПОЛНИТЬ ДЛЯ КАЖДОГО КОНТАКТНОГО ДЕТЕКТОРА          }
                                          FOR index := 1 TO 5 DO
                                            BEGIN
                    .
                    .
                    .
{КОНЕЦ                                                 }
                                            END;
```

Так же как и в вышеприведенном примере, если внутри конструкции ВЫПОЛНИТЬ ДЛЯ КАЖДОГО.. КОНЕЦ находится единственная конструкция, конструкция BEGIN... END может быть опущена.

```
{ВЫПОЛНЯТЬ НЕПРЕРЫВНО          }
                          WHILE 1 = 1 DO
                            BEGIN
                    .
                    .
                    .
{КОНЕЦ                          }
                            END;
```

Конструкция ВЫПОЛНЯТЬ НЕПРЕРЫВНО... КОНЕЦ

может быть всегда реализована в Паскале с помощью процедуры EXECUTIVE, которая не имеет конструкции RETURN. Она может быть использована в других процедурах, если используется конструкция GOTO, как было показано ранее в данном разделе. Если использование конструкции GOTO нежелательно, конструкция ВЫПОЛНЯТЬ НЕПРЕРЫВНО... КОНЕЦ может быть заменена другими конструкциями, например конструкцией ВЫПОЛНЯТЬ ПОКА... КОНЕЦ

Условные конструкции.

```
{ЕСЛИ ПЕРЕКЛЮЧАТЕЛЬ ВКЛЮЧЕН                                    }
                                    IF keyswitch < > 0
{  ТО УСТАНОВИТЬ СТАТУС                                        }
                                    THEN status := 1;
{ЕСЛИ ПЕРЕКЛЮЧАТЕЛЬ ВЫКЛЮЧЕН                                   }
                                    IF keyswitch = 0
{  ТО СБРОСИТЬ СТАТУС                                          }
                                    THEN status := 0;
{ЕСЛИ ПЕРЕКЛЮЧАТЕЛЬ ВКЛЮЧЕН И ТАЙМЕР В СОСТОЯНИИ               }
{ПОКОЯ                                                         }
                                    IF (keyswitch < > 0) AND (timerState = 0)
{  ТО УСТАНОВИТЬ СТАТУС                                        }
                                    THEN status := 1
{ЕСЛИ ИНДИКАТОР ЗАПОЛНЕНИЯ ПОКАЗЫВАЕТ ЧТО ВХОДНАЯ              }
{ЗАПИСЬ ЗАПОЛНЕНА                                              }
                                    IF fullIndicator = 10
{  ТО ПЕРЕВЕСТИ СТАТУС В СОСТОЯНИЕ «ЗАПОЛНЕНО»                 }
                                    THEN status := 1;
{  ИНАЧЕ ПЕРЕВЕСТИ СТАТУС В СОСТОЯНИЕ «НЕ ЗАПОЛНЕНО»           }
                                    ELSE status := 0
{ЕСЛИ СИМВОЛ СОДЕРЖИТ ЦИФРОВОЙ КОД                             }
                                    IF (character >= '0') AND (character <= '9')
{  ТО ВЫПОЛНИТЬ                                                }
                                    THEN BEGIN
{     ЕСЛИ СТАТУС УКАЗЫВАЕТ ЧТО ВХОДНАЯ ЗАПИСЬ                 }
{     НЕ ЗАПОЛНЕНА                                             }
                                       IF status = 0
{       ТО ВЫЗОВ: ДОБАВЛЕНИЕ К ВХОДНОЙ ЗАПИСИ (СИМВОЛ;)       }
                                          THEN AddToInputRecord (character);
{       ИНАЧЕ ВЫЗОВ: СИГНАЛ (;)                                }
                                          ELSE Beep;
{   КОНЕЦ                                                      }
                                    END;
```

Отметим, что точка с запятой после ELSE Beep является необязательной и может быть опущена.

Библиография

Библиография включает список избранных книг и руководств, рекомендуемых для дальнейшего изучения.

Микропроцессоры/микрокомпьютеры — общие вопросы

Aumiaux M., The Use of Microprocessors, Wiley, New York, 1980.
Barna A., Porat D. I., Introduction to Microcomputers and Microprocessors, Wiley, New York, 1976. (Имеется перевод: Барна А., Порэт Д. И., Введение в микро-ЭВМ и микропроцессоры. — М.: Знание, 1978.)
Cooper J. A., Microprocessor Bakground for Management Personnel, Prentice-Hall, Englewood Cliffs, N. J., 1981.
Garland H., Introduction to Microprocessor System Design, McGraw-Hill, New York, 1979.
Gibson G. A., Liu Y. C., Microcomputers for Engineers and Scientists, Prentice-Hall, Englewood Cliffs, N. J., 1980. (Имеется перевод: Гибсон. Г., Лю Ю-Ч., Аппаратные и программные средства микро-ЭВМ. — М.: Финансы и статистика, 1983.)
Givone D. D., Roesser R. P. Microprocessors/Microcomputers — An Introduction, McGraw-Hill, New York, 1980. (Имеется перевод: Гивоне Д. Д., Россер Р. П., Микропроцессоры и микрокомпьютеры. — М.: Мир, 1983.)
Greenfield J. D., Wray W. C., Using Microprocessors and Microcomputers: The 6800 Family, Wiley, New York, 1981.
Grillo J. P., Robertson J. D. Microcomputer Systems — an applications approach, Brawn, Dubuque, Iowa, 1979.
Klingman E. E., Microprocessor Systems Design, Prentice-Hall, Englewood Cliffs, N. J., 1977. (Имеется перевод: Клингман Э., Проектирование микропроцессорных систем. — М.; Мир, 1980.)
Korn G. A., Microprocessors and Small Digital Computer Systems for Engineers and Scientists, McGraw-Hill, New York, 1977.
Lee S. C., ed., Microcomputer Design and Applications, Academic Press, New York, 1977.
Lenk J. D., Hadbook of Microprocessors, Microcomputers and Minicomputers, Prentice-Hall, Englewood Cliffs, N. J., 1979.
Ogdin C. A., Microcomputer Management and Programming, Prentice-Hall, Englewood Cliffs, N. J., 1980.
Olesky J. E., Rutkowski G. B., Microprocessor and Digital Computer Technology, Prentice-Hall, Englewood Cliffs, N. J., 1981.
Peatman J. B., Microcomputer-based Design, McGraw-Hill, New York, 1977.
Short K. L., Microprocessors and Programmed Logic, Prentice-Hall, Englewood Cliffs, N. J., 1980.
Sipll C. J., Microcomputer Handbook, Petrocelli/Charter, New York, 1977.
Streitmatter G. A., Fiore V., Microprocessors — Theory and Applications, Reston, Reston, Va., 1979.
Tocci R. J., Laskowski L. P., Microprocessors and Microcomputers: Hardware and Software, Prentice-Hall, Englewood Cliffs, N. J., 1982.

Wakerly J. F., Microcomputer Architecture and Programming, Wiley, New York, 1981. (Имеется перевод: Уокерли Дж. Ф., Архитектура и программирование микро-ЭВМ. — М.: Мир, 1983.)

Терминология

Sippl C. J., Kidd D. A., Microcomputer Dictionary and Guide, Matrix Publishers, Champaign, Ill., 1976.

Значение компьютеров в современном обществе

Kemeny J. G., Man and the Computer, Scribner's, New York, 1972.
Montagu A., Snyder S. S., Man and the Computer, Auerbach Publishers, Philadelphia, Pa., 1972.
Wise K. D., Chen K., Yokely R. E., Microcomputers: A Technology Forecast and Assessment to the Year 2000, Wiley, New York, 1980.

Человеческие факторы вычислительных систем

Meadows C. T., Man — Machine Communication, Wiley, New York, 1970.

Теория вычислительных систем

Katzan H., Jr., Introduction to Computer Sciences, Petrocelli/Charter, New Yok, 1975.

Технология программирования

Buckle J. K., Managing Software Projects, American Elsevier, New York, 1977.
Brooks F. P., The Mythical Man-Month: Essays on Software Engineering, Addison-Wesley, Reading, Mass., 1975. (Имеется перевод: Брукс Ф. П., Как проектируются и создаются программные комплексы. — М.: Наука, 1979.)
Goos G., Hartmanis J., eds., Software Engineering — An Advanced Course, Springer-Verlag, New York, 1975.
Gunther R. C., Management Methodology for Software Product Engineering, Wiley, New York, 1978. (Имеется перевод: Гантер Р., Методы управления проектированием программного обеспечения. — М.: Мир, 1981.)
Hughes J. K., Michtom J. I., A Structured Approach to Programming, Prentice-Hall, Englewood Cliffs, N. J., 1977. (Имеется перевод: Хьюз Дж., Мичтом Дж., Структурный подход к программированию. — М.: Мир, 1980.)
Jackson M. A., Principles of Program Design, Academic Press, New York, 1975.
Jensen R. W., Tonies C. C., Software Engineering, Prentice-Hall, Englewood Cliffs, N. J., 1979.
Linger R. C., Mills H. D., Witt B. I., Structured Programming: Theory and Practice, Addison-Wesley, Reading, Mass., 1979. (Имеется перевод: Лингер Р. и др., Теория и практика структурного программирования. — М.: Мир, 1982)
Maynard J., Modular Programming, Auerbach Publishers, Philadelphia, Pa., 1972.
McGowan C. L., Keily J. R., Top-Down Structured Programming Techniques, Petrocelli/Charter, New York, 1975.
Wirth N., Systematic Programming: An Introduction, Prentice-Hall, Englewood Cliffs, N. J., 1973. (Имеется перевод: Вирт Н., Систематическое программирование. Введение. — М.: Мир, 1977.)
Yourdon E., Structured Walkthroughs, Prentice-Hall, Englewood Cliffs, N. J., 1980.

Программное обеспечение микрокомпьютеров

Ogdin C. A., Software Desing for Microcomputers, Prentice-Hall, Englewood Cliffs, N. J., 1978.

Программирование на языке PL/M

Intel Corporation, PL/M Programming Manul, Intel Corporation, Santa Clara, Calif.
McCracken D. D., A Guide to PL/M Programming for Microcomputer Applications, Addison-Wesley, Reading, Mass., 1978

Архитектура компьютеров

Baer J. L., Computer Systems Architecture, Computer Science Press, Potomac, Md., 1980.
Stone H. S., ed., Introduction to Computer Architecture, Science Research Associates, Chicago, 1980

Архитектура микрокомпьютеров

Doty K. L., Fundamental Principles of Microcomputer Architecture, Matrix Publishers, Beaverton, Oreg., 1979.
Greenfield S. E., The Architecture of Microcomputers, Brown, Boston, 1980.
Lippiatt A. G., The Architecture of Small Computer Systems, Prentice-Hall International, London, 1978. (Имеется перевод: Липьят А., Архитектура малых вычислительных систем. — М.: Мир, 1981.)

Программирование на языке ассемблера

Duncan F. G., Microprocessor Programming and Software Development, Prentice-Hall International, London, 1979.
McGlynn D. R., Modern Microprocessor System Design, Wiley, New York, 1980.

Сопряжение микро-ЭВМ

Artwick B., Microcomputer Interfacing, Prentice-Hall, Englewood Cliffs, N. J., 1980. (Имеется перевод: Артвик Б. А., Сопряжение микро-ЭВМ с внешними устройствами. — М.: Машиностроение, 1983.)
Garrett P. H., Analog Systems for Microprocessors and Minicomputers, Reston, Reston, Va., 1978. (Имеется перевод: Гаррет П. Н., Аналоговые устройства для микропроцессоров и мини-ЭВМ. — М.: Мир, 1981.)
Lipovski G. J., Microcomputer Interfacing: Principles and Practices, Lexington Books, Lexington, Mass., 1980.

Операционные системы

Barron D. W., Computer Operating Systems, Chapman and Hall, London, 1971.

Операционные системы для микрокомпьютеров

Intel Corporation, ISIS-II Systems User's Guide, Intel Corporation, Santa Clara, Calif.

Intel Corporation, PL/M-80 Compiler Operator's Manual, Intel Corporation, Santa Clara, Calif.

Интеграция и отладка

Brown A. R., Sampson W. A., Program Debugging: The Prevention and Cure of Program Errors, American Elsevier, New York, 1973.
Bruce R., Software Debugging for Microcomputers, Reston, Reston, Va., 1980.
Coffron J. W., Practical Troubleshooting Techniques for Microprocessor Systems, Prentice-Hall, Englewood Cliffs, N. J., 1981.
Ghani N., Farrell E., Microprocessor System Debugging, Wiley, New York, 1980.
Myers G. J., The Art of Software Testing, Wiley, New York, 1979. (Имеется перевод: Майерс Г. Дж., Искусство тестирования программ. — М.: Финансы и статистика, 1982.)

Проектирование логических схем и аппаратного обеспечения для микрокомпьютеров

Bishop R., Basic Microprocessors and the 6800, Hayden Book Co., Rochelle Park, N. J., 1979.
Coffron J. W., Practical Hardware Details for 8080, 8085, Z80 and 6800 Microprocessor Systems, Prentice-Hall, Englewood Cliffs, N. J., 1981. (Имеется перевод: Коффрон Дж., Технические средства микропроцессорных систем. — М.: Мир, 1983.)
Intel Corporation, 8085 Microcomputer Systems User's Manual, Intel Corporation, Santa Clara, Calif.
Intel Corporation, Component Data Catalog, Inter Corporation, Santa Clara, Calif.
Intel Corporation, Systems Data Catalog. Intel Corporation, Santa Clara, Calif.
Kraft G. D., Toy W. N., Mini/Microcomputer Hardware Design, Prentice-Hall, Englewood Cliffs, N. J., 1978.
Mano M. M., Digital Logic and Computer Design, Prentice-Hall, Englewood Cliffs, N. J., 1979.
Peatman J. B., Digital Hardware Desingn, McGraw-Hill, New York, 1980.
Stout D. F., Kaufman M., Handbook of Microcircuit Design and Application, McGraw-Hill, New York, 1980.
Winkel D., Prosser F., The Art of Digital Design: An Introduction to Top-Down Design, Prentice-Hall, Englewood Cliffs, N. J., 1980.

Персональные компьютеры

Bunnell D., Personal Computing, A Beginner's Guide, Hawthorn Books, New York, 1978.
Freiberger S. J., Chew P., Jr., A Consumer's Guide to Personal Computing, Hayden Book Co., Rochelle Park, N. J., 1978.
Koff R. M., Home Computers, A Manual of Possibilities, Harcourt Brace Jovanovich, Inc., New York, 1979.
Solomon L., Veit S., Getting Involved with Your Own Computer, A Guide for Beginners, Ridley Enslow Publishers, Short Hills, N. J., 1977.
Wells B., Personal Computers: What They Are and How to Use Them, Trafalgar House Publishers, Englewood Cliffs, N J., 1978

Компьютеры для бизнеса

Silver G. A., Small Computer Systems for Business, McGraw-Hill, New York, 1978.

Программирование на языке Бейсик

Albrecht R. L., Finkel L., Brown J. R., BASIC, Wiley, New York, 1978.
Sass C. J., A Structured Approach to BASIC Programming, Allyn and Bacon, Newton, Mass., 1980.

Программирование на языке Фортран

Katzan H., Jr., FORTRAN 77, Van Nostrand Reinhold, New York, 1978. (Имеется перевод: Катцан Г., Язык Фортран 77. — М.: Мир, 1982.)
Page R., Didday R., FORTRAN 77 for Humans, West Publishing Co., St. Paul, Minn., 1980.

Программирование на языке Паскаль

Atkinson L., Pascal Programming, Wiley, New York, 1980.
Bowles K. L., Microcomputer Problem Solving Using Pascal, Springer-Verlag, New York, 1977.
Grogono P., Programming in Pascal, Addison-Wesley, Reading, Mass., 1978. (Имеется перевод: Грогоно П., Программирование на языке Паскаль. — М.: Мир, 1982.)
Moore L., Foundations of Programming with Pascal, Wiley, New York, 1980.
Wilson I. R., Addyman A. M., A Practical Introduction to Pascal, Springer-Verlag, New York, 1978. (Имеется перевод: Уилсон И. П., Эддиман А. М., Практическое введение в Паскаль. — М.: Мир, 1983.)

Предметный указатель

Содержание

УВАЖАЕМЫЙ ЧИТАТЕЛЬ!

Ваши замечания о содержании книги, ее оформлении, качестве перевода и другие просим присылать по адресу: 129820, Москва, И-110, ГСП, 1-й Рижский пер., д. 2, изд-во «Мир».

Фридмен М., Ивенс Л.

ПРОЕКТИРОВАНИЕ СИСТЕМ С МИКРОКОМПЬЮТЕРАМИ

Научный редактор Т. Н. Шестакова
Младший научный редактор Н. И. Сивилева
Художник В. Е. Карпов
Художественный редактор Н. М. Иванов
Технический редактор И. М. Кренделева
Корректор В. И. Киселева

ИБ № 5399

Сдано в набор 20.02.85. Подписано к печати 13.01.86. Формат 60×90¹/₁₆. Бумага типографская № 2. Печать высокая. Гарнитура литературная. Объем 12,75 бум. л. Усл. печ. л. 25,50. Усл. кр.-отт. 25,50. Уч.-изд. л. 24,82. Изд. № 20/3809. Тираж 30 000 экз. Зак. № 569. Цена 2 руб.

ИЗДАТЕЛЬСТВО «МИР»
129820, ГСП, Москва, И-110, 1-й Рижский пер., 2.

Ленинградская типография № 2 головное предприятие ордена Трудового Красного Знамени Ленинградского объединения «Техническая книга» им. Евгении Соколовой Союзполиграфпрома при Государственном комитете СССР по делам издательств, полиграфии и книжной торговли. 198052, г. Ленинград, Л-52, Измайловский проспект, 29.

Издательство «Мир» выпустило в свет

Клингман Э. Проектирование специализированных микропроцессорных систем: Пер. с англ. — М.: Мир, 1985, 21 л., 1 р. 90 к.

В книге американского автора рассматриваются вопросы проектирования цифровых систем с использованием устройств с микропрограммным управлением и разрядно-модульной организацией. Приведены примеры построения микропроцессорных систем, предназначенных для решения различных технических задач.

Книга предназначена для специалистов, связанных по роду деятельности с проектированием микропроцессорных систем. Она может быть также полезной студентам старших курсов и аспирантам, специализирующимся в области проектирования систем обработки данных на базе микропроцессоров.