Ю-Чжен Лю, Г. Гибсон

МИКРО-ПРОЦЕССОРЫ СЕМЕЙСТВА 8086/8088

MICROCOMPUTER SYSTEMS: THE 8086/8088 FAMILY

ARCHITECTURE, PROGRAMMING, AND DESIGN

YU-CHENG LIU
GLENN A. GIBSON

both of
Electrical Engineering Department
The University of Texas at El Paso

PRENTICE-HALL, INC., Englewood Cliffs, N.J. 07632

Ю-Чжен Лю, Г. Гибсон

МИКРОПРОЦЕССОРЫ СЕМЕЙСТВА 8086/8088

АРХИТЕКТУРА, ПРОГРАММИРОВАНИЕ И ПРОЕКТИРОВАНИЕ МИКРОКОМПЬЮТЕРНЫХ СИСТЕМ

Перевод с английского В.Л. Григорьева

Москва «Радио и связь» 1987

ББК 32.97
Л93
УДК 681.325.5-181.4

Редакция переводной литературы

Лю Ю-Чжен, Гибсон Г.

Л93 Микропроцессоры семейства 8086/8088. Архитектура, программирование и проектирование микрокомпьютерных систем: Пер. с англ. — М.: Радио и связь, 1987. — 512 с.; ил.

В книге американских авторов проведен детальный анализ архитектуры микропроцессоров семейства 8086/8088 фирмы Intel. Большое внимание уделено конструированию программ и модульному программированию. Подробно рассмотрены вопросы построения микрокомпьютерных систем на основе микропроцессоров 8086 и 8088. Проанализированы структура и работа микросхем, обслуживающих эти микропроцессоры.

Для инженеров-конструкторов микрокомпьютерных систем.

Л $\frac{2405000000\text{-}185}{046\,(01)\text{-}87}$ 116-87

ББК 32.97

ПРЕДИСЛОВИЕ К РУССКОМУ ИЗДАНИЮ

Начало 70-х годов ознаменовалось рождением нового и, как оказалось, весьма перспективного и беспрецедентного по своим последствиям направления в развитии вычислительной техники – в 1971 г. был выпущен первый в мире микропроцессор. С тех пор за короткое время появилось несколько поколений микропроцессоров, а для прогнозирования перспектив их будущих применений не хватает даже самой богатой фантазии.

Микропроцессоры, а в более общем плане – большие и сверхбольшие интегральные схемы, революционизируют вычислительную технику в том отношении, что она становится все более дешевой, массовой и надежной, а ее применение оказывается экономически эффективным практически во всех областях народного хозяйства. По существу, микропроцессорная техника является фундаментом грандиозной программы компьютеризации общества.

За прошедшие годы в микропроцессорной технике сформировались три крупных направления:

1. Разработка однокристальных микропроцессоров с фиксированными длиной слова и системой команд. Эти микропроцессоры представляют собой то, что традиционно называется центральным процессором, и для организации функционально законченной системы требуют памяти и средств ввода-вывода.

2. Создание однокристальных микроЭВМ (микрокомпьютеров), содержащих на кристалле все главные компоненты системы – центральный процессор (по-прежнему процессор имеет фиксированную длину слова и систему команд), память и средства ввода-вывода. Конечно, каждая из этих компонент обладает пока сравнительно ограниченными возможностями, но тем не менее на базе однокристальных компьютеров реализуются контроллеры, содержащие всего несколько микросхем.

3. Выпуск секционных микропроцессоров с микропрограммным управлением, рассчитанных на проектирование специализированных систем, в которых разработчик может определять оптимальные длину слова и систему внешних (машинных) команд.

Очевидно, не имеет смысла спорить о преобладающем значении того или иного направления, но в США семейства однокристальных микропроцессоров представлены наиболее широко. Достаточно сказать, что они применяются практически во всех персональных компьютерах, в частности, микропроцессор 8088 является ядром персонального компьютера фирмы IBM. Предлагаемая вниманию читателей книга известных американских авторов посвящена именно однокристальным микропроцессорам.

История развития однокристальных микропроцессоров показывает их эволюцию от первого 4-битного микропроцессора 4004 через 8- и 16-битные устройства к новейшим 32-битным процессорам, функциональные возможности которых превосходят возможности процессоров крупных компьютеров недавнего прошлого. Сейчас в электронной продукции США и других стран важную роль играют 16-битные микропроцессоры. Об этом свидетельствует хотя бы динамика их мирового объема продажи (по данным журнала "Электроника"): 1984 г. – 440 млн дол., 1985 – 550 млн дол. и 1986 г. – 690 млн дол. К 1990 г. он достигнет 2 – 3 млрд дол. Поэтому ориентация настоящей книги на 16-битные микропроцессоры 8086/8088 фирмы Intel, которая оказывает заметное влияние на прогресс зарубежной микропроцессорной техники, вполне обоснована. Наличие отечественного аналога микропроцессора 8086 делает книгу достаточно актуальной и для советских читателей.

Интересующиеся микропроцессорной техникой, очевидно, знакомы с предыдущей книгой Г. А. Гибсона и Ю-Ч. Лю "Аппаратные и программные средства микроЭВМ", выпущенной издательством "Финансы и статистика" в 1983 г. и посвященной 8-битному микропроцессору 8080. Настоящая же книга является своеобразным шагом вперед и отражает новые достижения в микропроцессорной технике.

Среди достоинств книги отметим высокий методический уровень изложения материала, широту охвата многочисленных вопросов, возникающих при переходе на новый уровень микросистем, практическую направленность содержания и наличие множества программ и схемных конфигураций. Завершающие каждую главу упражнения помогают закрепить изучаемый материал.

Мы не будем давать обзора содержания книги, так как он имеется в предисловии авторов. Отметим все же, что в ней рассмотрены не только собственно микропроцессоры 8086/8088, но и множество программируемых интерфейсных микросхем, которые представляют самостоятельный интерес. Кроме того, большое практическое значение имеют мультипроцессорные конфигурации с арифметическим сопроцессором и процессором ввода-вывода. Но, разумеется, невозможность "объять необъятное" сказалась и в данной книге – часть вопросов изложена недостаточно подробно, и авторы неоднократно отсылают читателя к фирменным руководствам.

Круг читателей, на который рассчитана книга, достаточно широк. Много полезного найдут в ней инженерно-технические работники, занятые разработкой аппаратных и программных средств микросистем. Книга вполне доступна (за исключением, возможно, отдельных вопросов, связанных со схемотехникой) инженерам многих смежных специальностей. Ее можно рекомендовать студентам-старшекурсникам высших учебных заведений, специализирующимся в области вычислительной техники (собственно, книга и написана как учебное пособие).

При переводе книги мы стремились придерживаться максимального соответствия авторской терминологии. Например, в тексте фигурируют термины "программируемый связной интерфейс" и "параллельный периферийный интерфейс" вместо употребляемых в нашей литературе терминов "програм-

мируемый адаптер последовательного интерфейса" и "программируемый адаптер параллельного интерфейса". Вместо терминов "микропроцессорная система" и "микрокомпьютерная система" используется более короткий термин "микросистема". Наконец, мы решились отказаться от аббревиатуры ЭВМ и заменить ее термином "компьютер", от которого легко происходят такие слова, как "компьютеризация", "компьютерный" и др. Было решено также пользоваться в книге минимумом сокращений, которые при их чрезмерном употреблении затрудняют восприятие изучаемого материала. Наиболее широко применяются следующие сокращения: ЦП – центральный процессор, ЗУПВ – запоминающее устройство с произвольной выборкой, ПДП – прямой доступ к памяти и ПВВ – процессор ввода-вывода.

В заключение выражаю надежду, что хорошая книга Ю-Ч. Лю и Г. Гибсона будет с интересом (и, возможно, даже с удовольствием) воспринята советскими читателями.

В. Л. Григорьев

ПРЕДИСЛОВИЕ

Предлагаемая вниманию читателей книга написана как учебник для студентов высших учебных заведений, рассчитанный на односеместровый курс; однако она окажется полезной также для инженеров и техников. Предполагается, что читатель знаком хотя бы с одним языком программирования высокого уровня, например Фортраном, Бейсиком или Паскалем, а также с логическими основами компьютеров. В гл. 1 приведены общие принципы работы компьютеров и знающие их читатели могут лишь бегло познакомиться с этой главой. Каждая глава заканчивается упражнениями, которые помогают закрепить изучаемый материал. Упражнения упорядочены в соответствии с содержанием главы и должны помочь читателю проверить понимание им пройденного материала. Некоторые упражнения содержат дополнительные сведения, расширяющие кругозор читателя.

Книга ориентирована на изучение аппаратных и программных средств микросистем. Рассматриваемые в ней общие принципы иллюстрируются конкретными микропроцессорами 8086/8088 фирмы Intel, а также их вспомогательными микросхемами и примерами программ. Приводимые программы опираются на язык ассемблера ASM-86 фирмы Intel; однако подробное описание этого языка в книге отсутствует, так как, по нашему мнению, оно уведет читателей в сторону от принципиальных моментов.

Мы решили посвятить книгу таким областям применений микропроцессоров, в которых возможностей простых контроллеров и 8-битных процессоров недостаточно; именно поэтому был выбран 16-битный процессор. Из наиболее популярных однокристальных 16-битных процессоров мы остановились на 8086/8088, так как они имеют обширную систему команд, включающую в себя много арифметических операций и манипуляций цепочками, а также несколько архитектурных новинок, рассчитанных на организацию мультипрограммных и мультипроцессорных систем. Кроме того, фирма Intel разработала такие микросхемы, как, например, процессор числовых данных 8087 и процессор ввода-вывода 8089, которые позволяют сравнительно легко реализовать в системе некоторые важные мультипроцессорные функции. Опираясь на микропроцессор 8086, можно с единых позиций рассмотреть основы мультипрограммных и мультипроцессорных систем. В книге отражены такие вопросы, как системное программное обеспечение, способы обработки цепочек и модульное программирование, которые обычно отсутствуют в литературе по 8-битным микропроцессорам. Значение их становится все более важным по мере усложнения применений микропроцессоров.

После обзора вычислительных систем и введения в микропроцессоры, содержащихся в гл. 1, в следующей главе рассматриваются внутренняя архи-

тектура и машинные команды микропроцессора 8086. Хотя микропроцессоры 8086 и 8088 реализуют одну и ту же систему команд, их внутренние архитектуры несколько отличаются; об этих отличиях речь идет в последнем параграфе гл. 2. Глава 3 посвящена основам программирования на языке ассемблера. В ней читатель знакомится с языком ассемблера, а затем изучает основные команды микропроцессора 8086, включая арифметические и логические команды и команды передачи управления. Применения различных команд иллюстрируются многочисленными примерами. Рассмотрены также директивы ассемблера, необходимые в одномодульных программах, и процесс ассемблирования. Глава 4 касается структур многомодульных ассемблерных программ и усовершенствованных методов программирования. В ней имеется раздел, в котором показано, как правильно разрабатывать многомодульные программы.

Глава 5 содержит описание возможностей микропроцессора 8086 для обработки цепочек. В ней же рассмотрены вопросы преобразований кодов, которые необходимо решать при разработке эффективных процедур ввода-вывода. Глава 6 посвящена способам программирования ввода-вывода, включая программный ввод-вывод, ввод-вывод по прерываниям и прямой доступ к памяти. Изложение последовательности прерывания и применения векторов прерываний сопровождается примерами. Чтобы читатель разобрался в том, каким образом ввод-вывод реализуется операционными системами реального времени, мы привели минимальные сведения по буферированию данных. Глава 7 представляет собой введение в мультипрограммирование. Здесь довольно кратко изложены наиболее важные вопросы мультипрограммирования и их значение для проектирования микросистем. К таким вопросам мы отнесли разделение данных, синхронизацию процессов, реентрантные программы и управление памятью.

Остальная часть книги касается аппаратных аспектов микросистем. В гл. 8 рассмотрены интерфейс между системной шиной и микропроцессором, управление прерываниями, действия шины и временные диаграммы работы. Глава 9 посвящена интерфейсам, через которые к системной шине подключаются устройства ввода-вывода и внешней памяти. В ней описаны последовательный и параллельный интерфейсы, контроллеры прямого доступа к памяти, таймеры и контроллеры внешней памяти. Подсистемы памяти обсуждаются в гл. 10, а гл. 11 знакомит читателей с мультипроцессорными конфигурациями, которые совместимы с микросхемами фирмы Intel. Здесь же рассматриваются процессор числовых данных 8087 и процессор ввода-вывода 8089. Заключительная гл. 12 представляет собой обзор новых микросхем фирмы Intel: процессоры 80130, 80186 и 80286. В приложении дана подробная система команд микропроцессоров 8086/8088.

Авторы благодарны сотрудникам факультета электроники Университета штата Техас (Эль-Пасо) и признательны многим людям, которые помогали в работе над книгой. Особенно мы благодарим Р. Брунса из фирмы Intel за передачу нам необходимой информации и М. Л. Гибсон за перепечатку и проверку рукописи, а также решение многих упражнений.

Ю-Ч. Лю
Гленн А. Гибсон

1. ВВЕДЕНИЕ

Основными сферами применения микроэлектроники являются управление и обработка данных. На *низшей ступени* находятся простые контроллеры, а на *высшей* – сложные устройства, такие как роботы, авиационное и военное оборудование, а также системы обработки данных широкого назначения. Поскольку использование сложной схемы в простом случае не экономично, большее не всегда является лучшим. Хотя определенные усовершенствования воэможны и на низшей ступени, новая технология смещается в область реализации все более сложных систем.

По степени интеграции микропроцессорных приборов различают:

малую – менее 10 вентилей,
среднюю – от 10 до 100 вентилей,
большую – от 100 до нескольких тысяч вентилей,
сверхбольшую – десятки тысяч вентилей.

Современная микроэлектроника успешно осваивает сверхбольшую степень интеграции, последним крупным достижением которой являются однокристальные 16-битные процессоры и их вспомогательные приборы. Наиболее иэвестны процессоры 8086 фирмы Intel, 8000 фирмы Zilog и M68000 фирмы Motorola. Настоящая книга, в основном, посвящена процессору 8086, но значительная часть ее материала относится и к другим 16-битным процессорам.

В данной главе приводятся основные определения и другая информация обзорного плана. В § 1.1. содержится обзор вычислительных систем, а в следующих трех параграфах рассматриваются способы хранения данных в компьютере, варианты доступа к хранимой информации и общие действия компьютера при выполнении программы; § 1.5 посвящен критериям, используемым при проектировании цифровых систем, и эволюции микропроцессоров на примерах разработок фирмы Intel.

1.1. ОБЗОР МИКРОСИСТЕМ

Микросистема, как и любая другая вычислительная система, состоит из двух главных компонент – аппаратных и программных средств. К *аппаратным средствам* относятся схемы, стойки и др., а *программные средства* представляют собой программы, управляющие компьютером при выполнении им различных задач.

1.1.1. АППАРАТНЫЕ СРЕДСТВА

Под *архитектурой* компьютера понимаются общая конфигурация основных компонент, их главные возможности и характеристики, а также взаимосвязи. Общая архитектура типичной микросистемы показана на рис. 1.1, где видны такие компоненты, как центральный процессор (ЦП), схемы синхронизации, память, подсистема ввода-вывода, логика управления шиной и системная шина. В микросистеме центральным процессором служит *микропроцессор.* Он дешифрирует команды и управляет всеми действиями в системе; он же выполняет все арифметические и логические операции. *Генератор синхронизации* формирует одну или несколько последовательностей равномерно расположенных импульсов, которые необходимы для синхронизации действий в микропроцессоре и логике управления шиной. Выходные импульсы генератора имеют одну и ту же частоту, но смещены во времени, т. е. имеют различные фазы. В микросистемах применяются одно- — четырехфазные сигналы синхронизации, причем многофазные сигналы требовались в первых микропроцессорах. В большинстве современных микропроцессоров схема синхронизации, за исключением осциллятора (кварца), размещается на кристалле самого микропроцессора.

Память предназначена для хранения данных и команд, которые выполняет ЦП. Обычно она состоит из набора модулей, каждый из которых содержит несколько тысяч ячеек. Каждая ячейка хранит часть или все данное или команду и с ней ассоциируется идентификатор, называемый *адресом памяти*

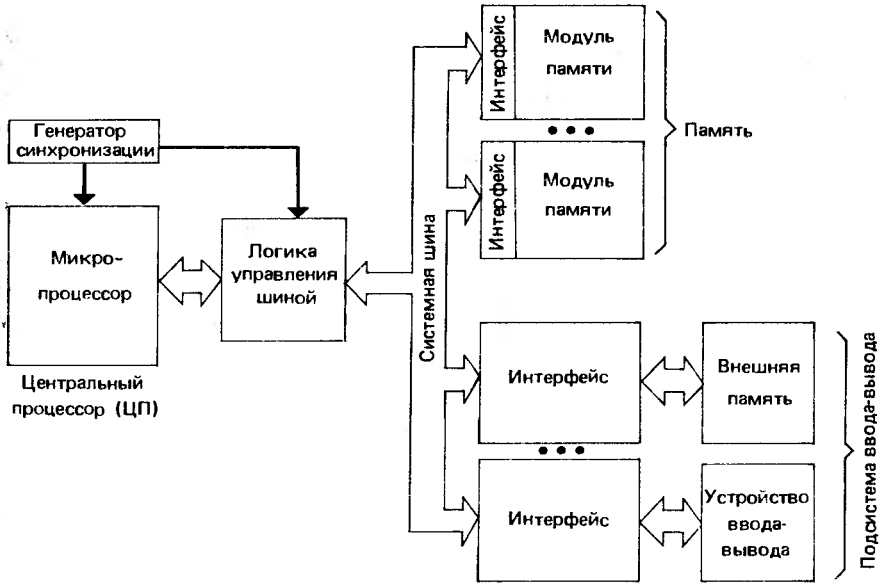


Рис. 1.1. Архитектура типичной микросистемы

(или просто *адресом*). Центральный процессор последовательно вводит (или выбирает) команды из памяти и выполняет определяемые ими задачи.

Подсистема ввода-вывода состоит из разнообразных устройств, предназначенных для взаимодействия с "внешним миром" и хранения больших объемов информации. Примерами устройств ввода служат карточные считыватели, фотосчитывающие ленточные устройства, аналого-цифровые преобразователи, а устройств вывода – строчные принтеры, графопостроители, карточные и ленточные перфораторы и цифро-аналоговые преобразователи. Некоторые устройства, например терминалы, обладают возможностями и ввода, и вывода. Компоненты компьютера, осуществляющие постоянное хранение программ и данных, называются *внешней* (*массовой*) памятью. Наиболее распространены сейчас ленточные и дисковые накопители, но в недавнем прошлом появились устройства на цилиндрических магнитных доменах (ЦМД-память) и приборах с зарядовой связью (ПЗС-память). Для выполнения программы ее необходимо передать из внешней памяти в основную.

Системную шину образует совокупность проводников, соединяющих ЦП с памятью и устройствами ввода-вывода. По этим проводникам, оформленным в виде кабеля или соединений на печатной плате, передается любая информация. Обычно проводники шины объединяются в три группы:

линии данных для передачи информации;

линии адреса, показывающие откуда или куда передается информация;

линии управления, регулирующие действия на шине.

Сигналы на шине должны быть скоординированы с сигналами, генерируемыми подключенными к шине разнообразными компонентами. Схемы для подключения шины к устройству называются *интерфейсом*, а *логика управления шиной* образует интерфейс ЦП. Проектирование интерфейсов и логики управления шиной упрощают разнообразные интерфейсные микросхемы. В зависимости от сложности системы логика управления шиной частично или полностью размещается на кристалле ЦП.

Интерфейсы памяти в основном образуют схемы для дешифрирования адреса целевой ячейки и буферирования данных на (с) шину(ы), а также схемы выполнения операций считывания и записи. Интерфейсы ввода-вывода варьируются от очень простых до очень сложных. Все интерфейсы ввода-вывода должны буферировать данные на(с) системную(ой) шину(ы), принимать приказы от ЦП и передавать в ЦП информацию о состоянии подключенного устройства. Кроме того, интерфейсы внешней памяти должны взаимодействовать непосредственно с памятью, а для этого требуется управление системной шиной. Взаимодействие между интерфейсом ввода-вывода и шиной данных осуществляется через регистры, называемые *портами ввода-вывода.*

1.1.2. ПРОГРАММНЫЕ СРЕДСТВА

Программные средства компьютера подразделяются на два больших класса – системные и пользовательские (прикладные). *Системные программные средства* (или *системное программное обеспечение*) образуют программы,

которые необходимы для создания, подготовки и выполнения других программ. *Пользовательские программные средства* (или *прикладное программное обеспечение*) включают в себя программы, разработанные пользователями в их попытках применить компьютер для решения своих задач.

Объем программных средств в конкретной вычислительной системе зависит от назначения. Компьютер, функция которого не изменяется весь срок его службы (например, контроллер, периодически выполняющий одну и ту же задачу), выполняет всего одну программу. С другой стороны, компьютер, предназначенный для обработки данных в широком смысле, требует обширного набора системных программ. В основном мы будем касаться компьютеров широкого назначения, в которых программы обычно, но не всегда, образуют четкую иерархическую структуру, представленную на рис. 1.2.

Операционная система – это совокупность системных программ, обеспечивающих взаимодействие пользователя с компьютером и эффективное использование ресурсов компьютера. Термин "операционная система" не имеет строгого определения, поэтому различные авторы относят к операционной системе различные системные программы. Наиболее важной частью операционной системы является *резидентный монитор*, постоянно находящийся в памяти компьютера. Резидентный монитор должен воспринимать приказы пользователей и инициировать выполнение операционной системой соответствующих действий.

В операционную систему входят также *драйверы ввода-вывода*, предназначенные для выполнения операций ввода-вывода, и *процедуры управления файлами*, т. е. большими наборами данных, которые хранятся во внешней памяти. Когда пользовательской или другой системной программе требуется обращение к устройству ввода-вывода, она не выполняет эту операцию сама, а запрашивает операционную систему, которая вызывает драйвер ввода-вывода. При таком подходе обеспечивается лучшее управление компьютером и отпадает необходимость введения подпрограмм ввода-вывода в пользовательские программы. Процедуры управления файлами применяются вместе

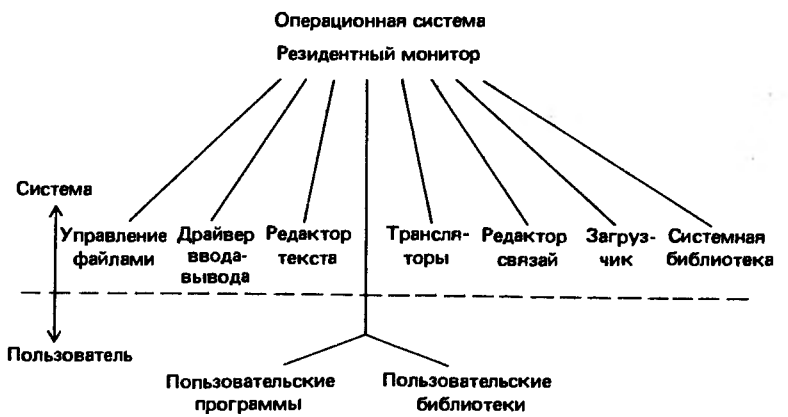


Рис. 1.2. Иерархия программных средств

с драйверами ввода-вывода внешней памяти для доступа, копирования и других операций с файлами.

В системные программные средства входят также трансляторы с языков высокого уровня, ассемблер, редактор текста и программы, помогающие в разработке других программ. Имеются три уровня программирования:

машинный язык;

язык ассемблера;

язык высокого уровня.

Программы на машинном языке — это такие, которые компьютер может непосредственно "понимать" и выполнять. Ассемблерные команды более или менее однозначно соответствуют машинным командам, но они написаны в виде символьных цепочек, которые более понятны человеку. Команды на языке высокого уровня намного ближе к английскому языку и структурированы так, что они близко соответствуют мышлению программиста. Но, в конце концов, программу на языке ассемблера или языке высокого уровня необходимо преобразовать на машинный язык, для чего применяются программы, называемые *трансляторами*. Транслятор программ, написанных на языке ассемблера, называется *ассемблером*, а на языке высокого уровня — *компилятором* или *интерпретатором*.

Редактор текста — это программа для ввода или модификации текста (букв, цифр, знаков пунктуации и др.), который необходимо запомнить или который уже хранится во внешней памяти. Текст может представлять собой программу на языке ассемблера или языке высокого уровня, набор данных или отчет. (Данная книга была написана с помощью редактора текста.) Редакторы текста применяются для различных целей, но нас будет интересовать использование их для создания программ.

При подготовке программ к выполнению требуются еще две системные программы. Часто возникает ситуация, когда одну и ту же задачу должны выполнять несколько программ. Поэтому в большинстве операционных систем предусматривается возможность образования наборов подпрограмм (называемых *библиотеками*), которые можно подключать к любой системной или пользовательской программам. Обычно имеется системная библиотека широкого назначения, но пользователи могут создавать и личные библиотеки. Вспомогательная программа, объединяющая выполняемую программу с библиотекой и другими ранее транслированными подпрограммами, называется *редактором связей* или *компоновщиком*. Еще одна служебная программа, предназначенная для размещения выполняемой программы в памяти, называется *загрузчиком*. Иногда функции редактирования связей и загрузки реализует одна программа. Весь процесс создания и выполнения программ подробно рассматривается в гл. 3.

1.2. ПРЕДСТАВЛЕНИЕ ДАННЫХ

Чтобы сделать компьютеры более надежными и простыми, в них применяют схемы, которые могут находиться только в двух состояниях; одно из них обозначается 0, а другое — 1. С помощью комбинаций из нескольких 0 и 1

можно представить любое число различных объектов. Комбинация, состоящая из одного 0 или одной 1, называется *битом*. В общем случае n бит могут представлять 2^n различных объектов и добавление еще одного бита удваивает число возможных комбинаций.

В компьютерах цепочки бит представляют собой числа, буквы, знаки пунктуации и любую другую информацию. Числа ассоциируются с двоичными комбинациями в соответствии с *числовыми форматами*. Имеются три основных формата:

двоичный (или целый);

плавающая точка (или вещественный);

двоично-кодированный десятичный (BCD или десятичный).

Форматы целого и плавающей точки соответствуют типам целых и вещественных чисел, которые применяют в Фортране и других языках высокого уровня.

Символьный код устанавливает соответствие букв и других символов двоичным комбинациям. Так как символьные коды включают в себя соответствия двоичных комбинаций десятичным цифрам, эти коды можно использовать для хранения и обработки чисел. (В языках высокого уровня символьные цепочки представляются символьными кодами.) Рассмотрим числовые форматы и символьные коды несколько подробнее (оставив формат плавающей точки до гл. 11).

1.2.1. ДВОИЧНЫЙ ФОРМАТ

Числа являются абстрактными объектами, которые обозначаются с помощью разнообразных правил и значков. Неотрицательные целые числа обычно представляются путем выбора числа x, называемого *основанием*, и x различных значков, называемых *цифрами*, и записи цепочки цифр вида

$$a_n a_{n-1} \ldots a_1 a_0.$$

Эта цепочка обозначает число

$$a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0.$$

Если, например, основание равно 10, цепочка 65308 представляет собой число

$$6 \times 10^4 + 5 \times 10^3 + 3 \times 10^2 + 0 \times 10 + 8.$$

Хотя мы обычно пользуемся основанием 10, им может быть любое целое число, большее 1. Так как компьютеры построены из схем с двумя состояниями, в них применяется основание 2. Тогда цепочка 10110 обозначает число

$$1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2 + 0.$$

В общении с компьютерами часто встречаются основания 8 и 16. Системы счисления, соответствующие основаниям 2, 8, 10 и 16, называются соответственно *двоичной, восьмеричной, десятичной* и *шестнадцатеричной системами счисления*. Значки, обозначающие в этих системах счисления цифры, приведены на рис. 1.3. Обычно основание системы (если оно не очевидно из кон-

Двоичная (Основание 2)	Восьмеричная (Основание 8)	Десятичная (Основание 10)	Шестнадцатеричная (Основание 16)
0	0 (000)	0 (0000)	0 (0000)
1	1 (001)	1 (0001)	1 (0001)
	2 (010)	2 (0010)	2 (0010)
	3 (011)	3 (0011)	3 (0011)
	4 (100)	4 (0100)	4 (0100)
	5 (101)	5 (0101)	5 (0101)
	6 (110)	6 (0110)	6 (0110)
	7 (111)	7 (0111)	7 (0111)
		8 (1000)	8 (1000)
		9 (1001)	9 (1001)
			A (1010)
			B (1011)
			C (1100)
			D (1101)
			E (1110)
			F (1111)

Рис. 1.3. Наиболее важные системы счисления
Примечание. В скобках даны двоичные эквиваленты

текста) обозначается индексом, например 1110_2 (двоичное число четырнадцать) или 1110_{10} (десятичное число тысяча сто десять).

Преобразование числа из системы счисления с основанием x в десятичную систему заключается в вычислении цифр d_i из соотношения

$$a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0 = d_m \times 10^m + \ldots + d_1 \times 10 + d_0$$

при заданных a_i. Наиболее просто это осуществить, если представить x и a_i в виде десятичных чисел и выполнить требуемые арифметические операции, например:

$$10111011_2 = 1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2 + 1 = 128 + 0 + 32 + 16 + 8 + 0 + 2 + 1 = 187_{10}$$

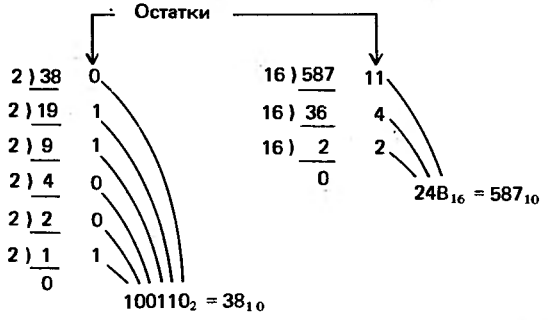
и

$$51A_{16} = 5 \times 16^2 + 1 \times 16 + 10 = 1306_{10}.$$

Такие же преобразования можно выполнить по правилу Горнера:

$$a_n x_n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0 = (\ldots (a_n x + a_{n-1}) x \ldots) x + a_0.$$

С помощью правила Горнера и последовательных делений на x десятичное число можно преобразовать в систему счисления с основанием x, как показывают следующие примеры:



Преобразование двоичного числа в 16-ричное осуществляется путем объединения двоичных цифр в группе по 4 и замены каждой группы ее 16-ричным эквивалентом, например

$$\underbrace{0110}_{6}\quad\underbrace{1011}_{B}\quad\underbrace{0111}_{7}$$

Обратное преобразование заключается в замене каждой 16-ричной цифры ее двоичным эквивалентом:

$$\underbrace{A}_{1010}\quad\underbrace{1}_{0001}\quad\underbrace{9}_{1001}$$

Преобразование двоичного числа в 8-ричное и обратное ему выполняют аналогично, но каждая группа содержит 3 двоичных цифры, а не 4.

Хотя сами компьютеры работают только с двоичными числами, в фирменных руководствах и книгах для записи чисел широко применяются 8- и 16-ричная системы счисления. Объясняется это более коротким представлением двоичных чисел, например 16-битное двоичное число 1011010100111010 имеет 16-ричное представление B53A. Мы будем широко пользоваться представлением чисел в 16-ричной системе счисления.

Арифметические операции в любой системе счисления выполняются по тем же алгоритмам, что и в десятичной системе. Однако, не зная таблиц 8- и 16-ричного сложения и умножения, трудно выполнять соответствующие арифметические операции. При необходимости операнды преобразуются в десятичные числа, над ними выполняется операция, а результат преобразуется в исходную систему счисления. Примеры арифметических операций над двоичными числами показаны на рис. 1.4.

Для представления отрицательных целых чисел можно ввести дополнительный бит, обозначающий знак. В этом случае 0 обычно соответствует знаку плюс, 1 – знаку минус, а получающийся формат называется *прямым кодом*. Однако этот формат на практике используется редко. В компьютерах почти всегда применяется *дополнительный код*, в котором отрицательные целые числа представляются в виде

$$-b = 2^n - b,$$

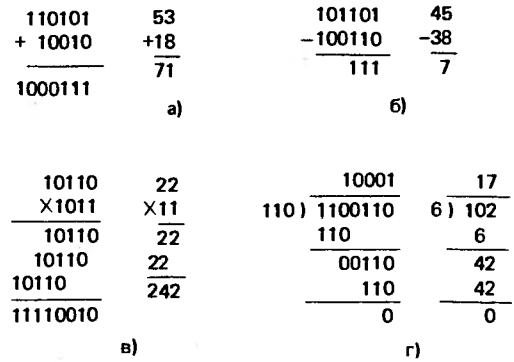


Рис. 1.4. Типичные двоичные арифметические операции

где b — абсолютное значение целого числа, а n — число бит, используемых для его представления. Если $n = 16$ и $b = 1234_{10} = 0000010011010010_2 = 04D2_{16}$, то $-b$ в дополнительном коде равно $2^{16} - 1234 = 1111101100101110_2 = FB2E_{16}$. При использовании n бит допускается однозначное представление целых чисел в диапазоне $-2^{n-1} \ldots 2^{n-1} - 1$. Когда $n = 16$, целые числа из диапазона $-32768 \ldots 32767$ записываются в следующем виде:

$$\begin{aligned}
1000000000000000 &= 8000\\
1000000000000001 &= 8001\\
&\vdots\\
1111111111111111 &= FFFF\\
0000000000000000 &= 0000\\
0000000000000001 &= 0001\\
&\vdots\\
0111111111111111 &= 7FFF
\end{aligned}$$

Для упрощения вычислений дополнительный код числа b можно образовать, заменив нули на единицы и единицы на нули, что дает $(2^n - 1) - b$ (*обратный код* b), и прибавив 1. Пусть $n = 8$ и $b = 46$:

$$\begin{aligned}
2^8 - 1 &= 11111111\\
b &= 00101110\\
\hline
(2^8 - 1) - b &= 11010001 \quad \text{обратный код}\\
&\quad +1\\
\hline
2^8 - b &= 11010010 \quad \text{дополнительный код}
\end{aligned}$$

Можно показать, что при обычном сложении знаковых целых чисел в дополнительном коде результат имеет правильное представление, если сохранять только младшие n бит. Вычитание выполняется путем прибавления дополнительного кода вычитаемого. Например, для $n = 8$:

$$
\begin{array}{rcl}
72 & = & 01001000 \\
-35 & = & -00100011 \\
\hline
37 & & \\
\end{array}
\qquad
\begin{array}{l}
\ 01001000 \\
+\ \underline{\ 11011101\ } \\
1\ \ 00100101 = 37 \\
\uparrow\ \text{Пропадает}
\end{array}
$$

$$
\begin{array}{rcl}
-29 & = & 11100011 \\
-(-90) & = & -10100110 \\
\hline
61 & & \\
\end{array}
\qquad
\begin{array}{l}
\ 11100011 \\
+\ \underline{\ 01011010\ } \\
1\ \ 00111101 = 61 \\
\uparrow\ \text{Пропадает}
\end{array}
$$

1.2.2. ДВОИЧНО-КОДИРОВАННЫЙ ДЕСЯТИЧНЫЙ ФОРМАТ

В *двоично-кодированном десятичном формате* (или BCD-формате) десятичные цифры хранятся в виде 4-битных двоичных эквивалентов. Имеются две основные разновидности этого формата: упакованный и неупакованный. В упакованном BCD-формате цепочка десятичных цифр хранится в виде последовательности 4-битных групп, например число 9502 – в виде 1001 0101 0000 0010. В неупакованном BCD-формате каждая цифра находится в младшей тетраде 8-битной группы, а содержимое старшей тетрады несущественно. Число 9502 будет храниться в виде

*шшш*1001 *шшш*0101 *шшш*0000 *шшш*0010

В отличие от двоичного дополнительного кода, который применяется для представления отрицательных целых чисел в двоичном формате, при отсутствии в системе специальных схем соглашение о знаке для BCD-формата устанавливается программистом (и после введения такого соглашения его необходимо неукоснительно соблюдать). Отрицательные целые числа допускают представления в десятичном обратном или в десятичном дополнительном коде. При выборе десятичного обратного кода для знаков плюс и минус берутся две из неиспользуемых 4-битных комбинаций, например 1100 обозначает плюс, а 1101 – минус. Знак допускается размещать до или после цепочки цифр.

Десятичный n-разрядный дополнительный код произвольного целого числа d определяется как $10^n - d$. При заданном n диапазон целых чисел, которые однозначно представляются в десятичном дополнительном коде, составляет от $-5 \times 10^{n-1}$ до $5 \times 10^{n-1} - 1$. В упакованном BCD-формате при n = = 8 для хранения целых чисел из диапазона –50000000 . . . 49999999 потребуются 32 бита. Десятичный дополнительный код обладает такими же свойствами, как и двоичный дополнительный, и сложение дает правильный результат в десятичном дополнительном коде, например для $n = 4$ имеем:

$$
\begin{array}{rcl}
252 & = & 0252 \\
+\ (-485) & = & 9515 \\
\hline
-233 & & 9767
\end{array}
$$

Вычитание сводится к прибавлению к уменьшаемому отрицательного вычитаемого.

Символ ASCII	16-ричный код	Управляющий символ
NUL	00	Пустой символ
SOH	01	Начало заголовка
STX	02	Начало текста
ETX	03	Конец текста
EOT	04	Конец передачи
ENQ	05	Запрос
ACK	06	Подтверждение
BEL	07	Звонок
BS	0B	Возврет на шаг
HT	09	Горизонтальная табуляция
LF	0A	Перевод строки
VT	0B	Вертикальная табуляция
FF	0C	Перевод формы (страницы)
CR	0D	Возврат каретки
SO	0E	Выдвигание
SI	0F	Вдвигание
DLE	10	Переключение линии
DC1	11	Управление устройством 1
DC2	12	Управление устройством 2
DC3	13	Управление устройством 3
DC4	14	Управление устройством 4
NAK	15	Отрицательное подтверждение
SYN	16	Синхронный/холостой
ETB	17	Конец передачи блока
CAN	1B	Уничтожение данных
EM	19	Конец носителя
SUB	1A	Начало специальной последовательности
ESC	1B	Переключение
FS	1C	Разделитель файлов
GS	1D	Разделитель групп
RS	1E	Разделитель записей
US	1F	Разделитель единиц
SP	20	Пробел
!	21	
"	22	
#	23	
$	24	
%	25	
&	26	
'	27	
(	28	
)	29	
*	2A	
+	2B	
,	2C	
–	2D	
.	2E	
/	2F	
0	30	
1	31	
2	32	
3	33	
4	34	
5	35	
6	36	
7	37	
8	38	
9	39	
:	3A	
;	3B	
<	3C	
=	3D	
>	3E	
?	3F	
@	40	
A	41	
B	42	
C	43	
D	44	
E	45	
F	46	
G	47	
H	48	
I	49	
J	4A	
K	4B	
L	4C	
M	4D	
N	4E	
O	4F	
P	50	
Q	51	
R	52	
S	53	
T	54	
U	55	
V	56	
W	57	
X	58	
Y	59	
Z	5A	
[	5B	
\	5C	
]	5D	
^	5E	
_	5F	
`	60	
a	61	
b	62	
c	63	
d	64	
e	65	
f	66	
g	67	
h	68	
i	69	
j	6A	
k	6B	
l	6C	
m	6D	
n	6E	
o	6F	
p	70	
q	71	
r	72	
s	73	
t	74	
u	75	
v	76	
w	77	
x	78	
y	79	
z	7A	
{	7B	
|	7C	
}	7D	
~	7E	
DEL	7F	забой

Рис. 1.5. Код ASCII

1.2.3. БУКВЕННО-ЦИФРОВЫЕ КОДЫ

Среди буквенно-цифровых (символьных) кодов преобладают код EBCDIC, используемый фирмой IBM, и код ASCII, который применяется почти всеми остальными производителями компьютеров. На рис. 1.5 показано кодирование символов в коде ASCII. Символьные коды являются основным средством осуществления ввода-вывода при взаимодействии с "внешним миром". Как показано на рис. 1.6, при нажатии клавиши на терминале производится

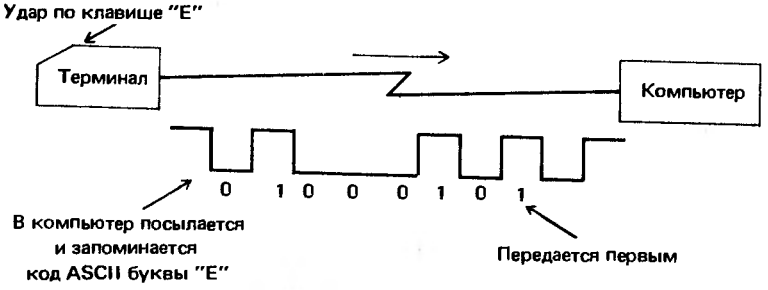


Рис. 1.6. Передача в компьютер кода ASCII символа

формирование и передача в компьютер соответствующего кода ASCII. Если же компьютер посылает в терминал двоичную цепочку кода ASCII, терминал должен дешифрировать эти биты и отреагировать соответствующим образом. Отметим, что печатаются не все символы кода ASCII; некоторые из них осуществляют возврат на шаг, пробел, перевод строки, возврат каретки и т. д. Кроме печатных и управляющих символов, в коде ASCII имеются такие символы, как EOF (конец файла) и EOT (конец передачи), которые служат маркерами при передаче и хранении данных.

Например, символьная цепочка

DOE,
JOHN P.–50

соответствует следующей цепочке двоичных комбинаций (которые даются в 16-ричной записи):

Возврат каретки Перевод строки Пробел

44	4F	45	2C	0D	0A	4A	4F	48	4E	20	50	2E	2D	35	30
D	O	E				J	D	H	N		P			5	0

Специально подчеркнем, что ни цифра 0, ни пробел не соответствуют нулевой комбинации. Двоичная комбинация, состоящая из нулей, называется *пустым символом* и не вызывает никаких действий. Она применяется в основном для выделения времени на возврат каретки или перевод строки в механическом терминале.

Число бит, которое необходимо в коде для представления символа, называется длиной. Код длиной n допускает идентификацию 2^n символов. Из

рис. 1.5 видно, что код ASCII является 7-битным и содержит 128 символов. Кроме 7 бит собственно кода, обычно к каждому символу в качестве старшего бита присоединяется бит паритета, поэтому в передаче участвуют 8 бит. Обычно в компьютере дополнительный бит после приема символа сохраняется, но устанавливается равным нулю. Следовательно, символы в компьютере хранятся в виде 8-битных групп.

Численная последовательность символов в коде называется *сравнительной последовательностью кода.* Важно, что числа, представляющие собой цифры, следуют в возрастающем порядке, так как при этом для сравнения значений применимы арифметические действия непосредственно над кодовыми числами. Кроме того, когда числа, ассоциируемые с буквами, следуют в возрастающем порядке, для алфавитного упорядочивания символьных цепочек можно воспользоваться арифметическими операциями.

Числа передаются в (из) компьютер(а) в виде последовательностей цифр, представленных в коде ASCII. Например, число 7902 передается как

37	39	30	32
7	9	0	2

Компьютер, принимая число, может запомнить его без модификации, что соответствует неупакованному BCD-формату; может удалить старшие тетрады и упаковать младшие по две, что соответствует упакованному BCD-формату; наконец, он может преобразовать число в двоичный формат. Выбор того или иного способа зависит от выполняемой программы. Неупакованные BCD-числа не требуется преобразовывать для операций ввода-вывода, но они занимают в памяти больше места. Беззнаковое целое число 7902 требует 32 бита памяти в неупакованном BCD-формате, 16 бит в упакованном и всего 13 бит в двоичном. Кроме того, компьютер выполняет арифметические операции с числами в двоичном формате быстрее, чем в BCD-формате. Подробнее преобразования чисел рассматриваются в гл. 5.

1.3. АДРЕСА

Конечно, все ячейки памяти и регистры ввода-вывода состоят из бит, но так как отдельные биты содержат очень мало информации, они группируются в *байты* и *слова*. Поскольку символы обычно имеют длину 7 или 8 бит и поскольку компьютеры более легко работают со степенями 2, байты почти всегда состоят из 8 бит. Слова же состоят из 2, 3 или 4 байт в зависимости от компьютера и структуры его системной шины. Так как 16-битные однокристальные микропроцессоры имеют в своих системных шинах 16 линий данных, в них термин "слово" обозначает 2 байта (16 бит).

С каждым байтом ассоциируется идентифицирующий его адрес и при обращении к байту его адрес передается в соответствующий интерфейс по линиям адреса. Адреса образованы двоичными комбинациями и множество всех возможных комбинаций в данной ситуации называется *адресным пространством*. В некоторых компьютерах имеется два адресных пространства, а в других для обращения ко всем ячейкам памяти и регистрам ввода-вывода используется единое адресное пространство. При наличии отдельных адрес-

ных пространств памяти и ввода-вывода для указания нужного адресного пространства вместе с линиями адреса необходимо использовать некоторые линии управления. Так как память разделена на модули, несколько старших бит адреса памяти применяют для выбора модуля, а остальные (младшие) биты идентифицируют байт (или слово) в модуле. Аналогично интерфейс идентифицируется старшими битами адреса ввода-вывода, а регистр в интерфейсе выбирается двумя или тремя младшими битами. Общая организация и адресация памяти и регистров ввода-вывода показаны на рис. 1.7.

Допустимое число бит адреса определяет размер адресного пространства. Если адрес содержит n бит, получается 2^n возможных адресов от 0 до $2^n - 1$. Число линий адреса в системной шине диктует размер пространства памяти (или, возможно, объединенного пространства памяти и ввода-вывода). При наличии n линий адреса максимальная емкость памяти (или памяти и ввода-вывода) составляет 2^n байт. Двадцать линий адреса обеспечивают емкость $2^{20} = (2^{10})^2 \approx (10^3)^2 = 1$ млн байт.

Когда слово состоит из двух байт, возникает вопрос, адрес какого байта использовать для идентификации слова. Кроме того, иногда требуется указывать конкретный бит в байте или в слове. В настоящей книге (как и в руководствах фирмы Intel) адресом слова считается адрес его младшего байта

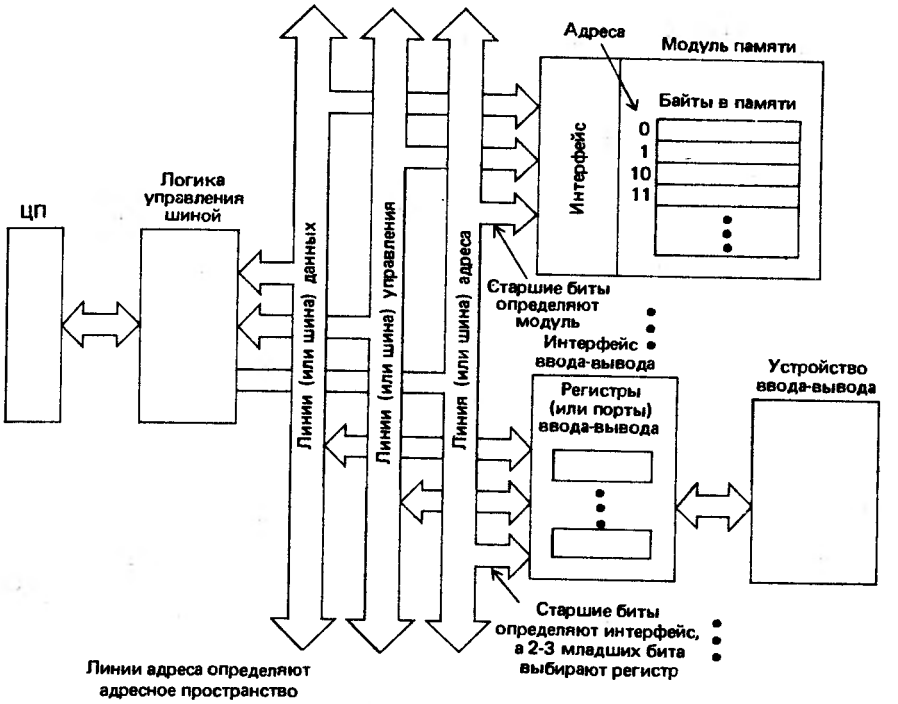


Рис. 1.7. Организация регистров памяти и ввода-вывода

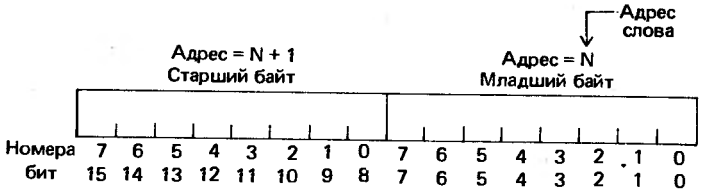


Рис. 1.8. Соглашения об адресах и нумерации бит

(или меньший адрес). Биты нумеруются с нуля, назначаемого младшему биту. В байте старший бит имеет номер 7, а в слове – номер 15. Рассмотренные соглашения иллюстрирует рис. 1.8.

В некоторых компьютерах требуется, чтобы слова начинались по адресам, кратным числу байт в слове, и фиксируется ошибка выравнивания, если это правило не соблюдается. В других же, в том числе и в микропроцессоре 8086, слова могут начинаться по любому адресу; однако, обращения к "невыравненным" словам требуют нескольких обращений к памяти (например, в микропроцессоре 8086 обращение к слову, начинающемуся по нечетному адресу, требует двух обращений к памяти).

1.4. ПРИНЦИПЫ ДЕЙСТВИЯ КОМПЬЮТЕРА

Опыт работы на языках высокого уровня показывает, что ЦП должен упрощать действия со следующими объектами:

присваивания и арифметические выражения;
безусловные переходы;
условные переходы, логические выражения и отношения;
циклы;
массивы и другие структуры данных;
подпрограммы;
ввод-вывод.

На рис. 1.9 показана типичная архитектура ЦП, обладающего такими возможностями. В нем имеются устройство управления для дешифрирования и исполнения команд, набор рабочих регистров, предназначенных для адресации и производства вычислительных операций, арифметико-логическое устройство (АЛУ) для выполнения арифметических и логических операций и секция управления вводом-выводом.

Как и программы на языках высокого уровня, программа на машинном языке выполняется последовательно до тех пор, пока не встретится команда перехода. *Регистр команды* IR содержит текущую команду на время ее дешифрации и выполнения, а *программный счетчик* PC предназначен для хранения адреса следующей команды. Когда текущая команда завершена, адрес из PC выдается на шину адреса, память помещает следующую команду на шину данных и ЦП вводит команду в IR. Пока дешифрируется эта команда, определяется ее длина в байтах и производится инкремент PC на эту длину и PC адресует следующую команду. Когда выполнение данной команды заканчивается, содержимое PC помещается на шину адреса и цикл повторяется.

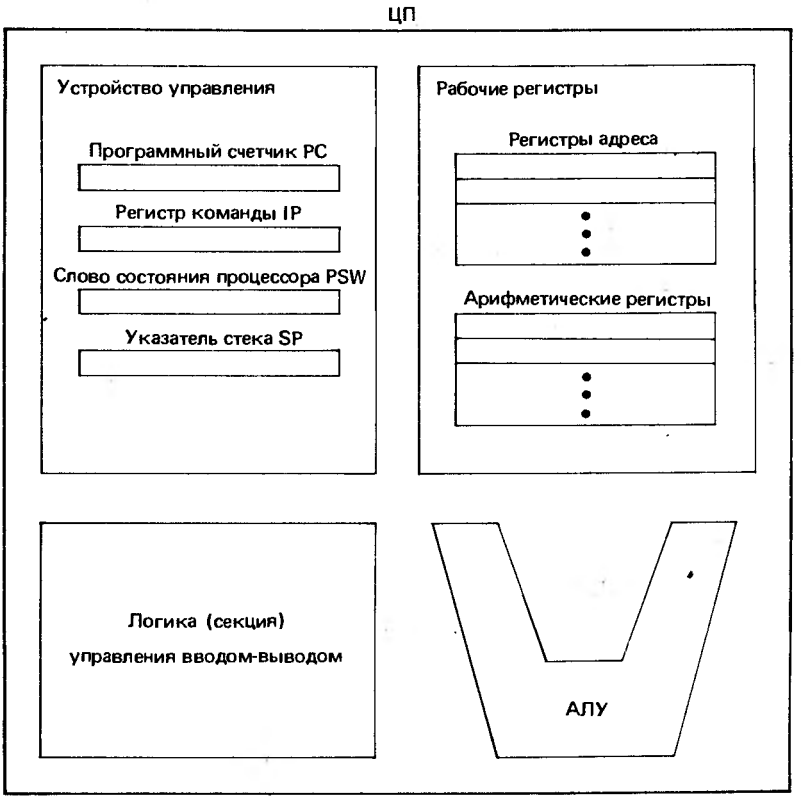


Рис. 1.9. Типичная архитектура центрального процессора

Команды безусловного перехода позволяют изменить естественный порядок следования команд путем замещения содержимого PC (т. е. адреса следующей по порядку команды) адресом, определяемым самой командой перехода. Команды условных переходов замещают или не замещают содержимое PC в зависимости от результатов предыдущих команд, т. е. текущего состояния процессора, определяемого предыдущими командами. Текущее состояние процессора находится в регистре, называемом *словом состояния процессора* PSW. В этом регистре имеются биты, показывающие такие условия, как получение в предыдущих операциях положительного, отрицательного или нулевого результата. Если после команды "вычитания" находится команда "перехода по нулю", переход осуществляется, если PSW показывает получение при вычитании нулевого результата. Если же PSW фиксирует ненулевой результат вычитания, переход не производится. Когда реализован переход, начинается новая последовательность команд с адреса, к которому осуществлен переход. Схема выполнения команд в компьютере приведена на рис. 1.10.

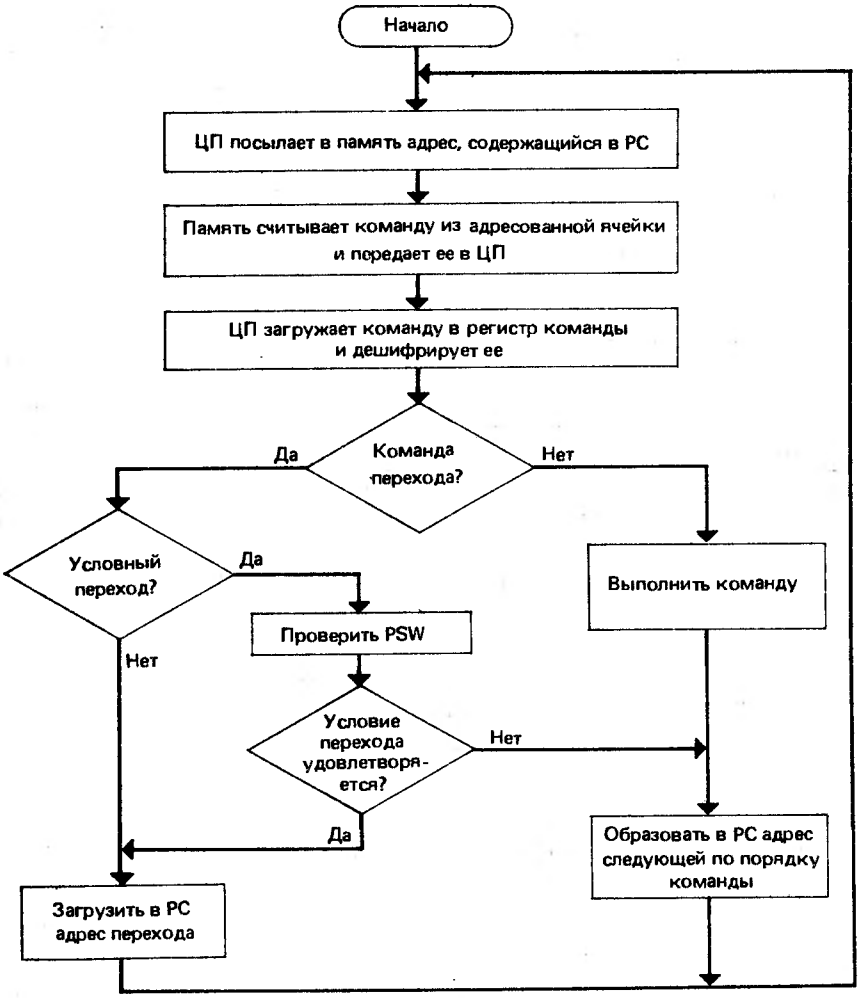


Рис. 1.10. Последовательное выполнение команд

Циклы обычно реализуются с помощью команд условных переходов, хотя в некоторых микропроцессорах имеются команды, которые объединяют счет и (или) проверку с условным переходом. В большинстве циклов, например в циклах DO языка Фортран, осуществляется инкремент или декремент счетчика и цикл повторяется до тех пор, пока счетчик не достигает предела. При каждом изменении счетчика результат сравнивается с пределом, соответственно устанавливается PSW и в зависимости от его содержимого переход производится или нет.

Действия, связанные с вызовом подпрограммы, требуют специальной разновидности перехода (см. рис. 1.11). Как и в других переходах, вызов подпрограммы также заменяет содержимое PC на адрес перехода, но при этом запоминается текущее содержимое PC, образующее *адрес возврата.* Команда возврата должна восстановить в PC адрес возврата, чтобы после завершения подпрограммы продолжалось последовательное выполнение основной программы. При вызове подпрограммы кроме запоминания адреса возврата обычно требуется временно сохранить и другую информацию, например содержимое рабочих регистров. Это объясняется тем, что подпрограмма может разрушить первоначальное содержимое этих регистров, которое потребуется при возврате в основную программу. Обычно такая информация запоминается в специальной области памяти, называемой *стеком*. Адрес ячейки стека, к которой производилось последнее обращение (или, как в некоторых компьютерах, к которой будет происходить следующее обращение), находится в регистре *указателя стека* SP. Стеки и подпрограммы подробно рассматриваются в гл. 4.

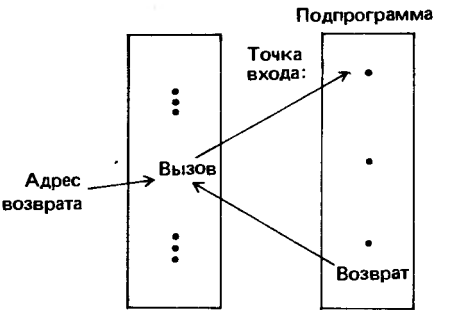


Рис. 1.11. Вызов подпрограммы

Рабочие регистры предназначены для временного хранения информации, которая помогает при адресации и в вычислительном процессе. Их можно разделить на две группы: адресную и арифметическую, хотя некоторые регистры можно отнести к обеим.

Адресная группа применяется для гибкой адресации данных. Обрабатываемое командой данное может быть частью команды, его адрес может быть частью команды, оно может находиться в регистре, его адрес может быть в регистре или адрес данного может быть суммой части команды и содержимого одного или нескольких регистров. Иногда при использовании регистра команда просто идентифицирует регистр, содержащий адрес, но чаще адрес определяется более сложно. Например, при обращении к элементам массива адрес элемента состоит из двух частей — *базового адреса* (т. е. адреса первого элемента массива) и *смещения*. Так как часто приходится обрабатывать весь массив, удобна возможность легкого инкремента смещения. Поэтому адрес элемента массива часто вычисляется суммированием содержимого двух регистров, один из которых содержит базовый адрес и называется *базовым*, а второй содержит смещение и называется *индексным*. Двумерный массив несколько усложняет ситуацию, требуя сложения базы, смещения в столбце и смещения. При этом обычно суммируются часть команды (смещение), базовый регистр и индексный регистр. Базовые регистры применяются также для перемещения в памяти программ и блоков данных, о чем речь пойдет в гл. 2 и 4.

Арифметические регистры предназначены для временного хранения операндов и результатов арифметических операций. Так как основным фактором, ограничивающим быстродействие, являются передачи по системной шине, обращение к регистру осуществляется намного быстрее, чем обращение к памяти. Поэтому при необходимости выполнения нескольких операций над набором данных лучше ввести данные в арифметические регистры, произвести требуемые вычисления и поместить результаты в память, чем обрабатывать данные непосредственно из памяти. Обычно чем больше арифметических регистров находится в составе ЦП, тем выше его быстродействие.

Арифметико-логическое устройство реализует арифметические, логические, сдвиговые и другие операции. Секция управления вводом-выводом содержит схемы ЦП, которые управляют операциями ввода-вывода. Какие схемы находятся в ЦП, а какие образуют внешнюю логику управления шиной, зависит от системы.

Хотя и имеется множество разновидностей рассмотренной архитектуры ЦП, мы касались общих принципов работы и основных узлов микропроцессоров. Приведенный материал должен помочь читателю в изучении конкретных микропроцессоров. В следующей главе более подробно описаны архитектура и действия микропроцессоров 8086/8088, выбранных в качестве базовых для данной книги.

1.5. МИКРОПРОЦЕССОРЫ В ПРОЕКТИРОВАНИИ ЦИФРОВЫХ СИСТЕМ

С начала 70-х годов микропроцессоры, интерфейсные микросхемы и полупроводниковая память произвели революцию в проектировании цифровых систем. Объясняется это программируемостью микропроцессоров и тем фактом, что однокристальный микропроцессор по своим возможностям эквивалентен сотням микросхем с малой и средней степенью интеграции. Микропроцессоры применяют во все большем числе цифровых систем, реализованных ранее на так называемой "жесткой логике", и эта тенденция сохраняется. По мере снижения стоимости микропроцессоров их используют даже в простых системах типа контроллеров, в которых реализуются не все возможности микропроцессоров. Причины указанной тенденции станут понятными, если разобраться в основных критериях, которые учитываются при проектировании цифровых систем.

Стоимость. Кроме стоимости собственно микросхем, на единичную стоимость системы влияют следующие факторы: приобретение, хранение и контроль микросхем, оплата монтажа, пайки и внешнего оформления, расходы на приобретение печатных плат, блоков питания, корпусов и т. д. Расходы на приобретение микросхем обычно не превышают 10 % общей стоимости системы, однако единичная стоимость пропорциональна числу микросхем, а не их внутренней сложности. Следовательно, экономичнее применять более дорогие БИС и СБИС, если они заменяют достаточное число микросхем с малой и средней степенью интеграции.

Гибкость. После запуска системы в производство могут потребоваться ее модификации или усовершенствования, что объясняется выявлением по ре-

зультатам эксплуатации просчетов проектирования или необходимостью введения новых функций. Если система реализована на основе "жесткой логики", ее придется заново проектировать, модифицировать и проверять, на что затрачивается много времени и средств. С другой стороны, благодаря программируемости микропроцессора изменения касаются в основном управляющей программы, а не аппаратных средств.

Надежность. Так как интенсивность отказов системы пропорциональна числу микросхем, надежность системы увеличивается по мере роста сложности микросхем и уменьшения их числа. Кроме того, сокращение числа микросхем ведет к уменьшению межсоединений с соответствующим упрощением решения проблем отказов, помех и синхронизации.

Время проектирования. Процесс традиционного проектирования систем на основе "жесткой логики" по своей природе последовательный, т. е. следующий его этап невозможно начинать до завершения предыдущего. Проектирование же микросистемы возможно разделить на разработки аппаратных и программных средств, которые можно вести параллельно.

Быстродействие. Основное преимущество системы на основе "жесткой логики" над микросистемой заключается в том, что она намного быстрее реагирует на входные воздействия. Однако во многих применениях высокого быстродействия не требуется и определяющим фактором оказывается гибкость программируемой системы. Кроме того, производительность новых микропроцессоров увеличивается и расхождение в быстродействии сокращается.

Эра микропроцессоров началась с тех пор, когда технология позволила реализовать в одной микросхеме все необходимые функции центрального процессора. Совершенствование микропроцессоров шло параллельно с развитием микроэлектронной технологии, которая позволяла размещать на кристалле все больше и больше логических схем. Хотя по мере усложнения микропроцессора растет и его стоимость, она все же остается намного меньше стоимости эквивалентной системы, реализованной на микросхемах с меньшими функциональными возможностями. Кроме уменьшения числа микросхем, необходимых для реализации данной функции, сокращается и общее число контактов, что позволяет уменьшить расходы на монтаж.

По мере перехода технологии от БИС к СБИС микропроцессоры стали иметь не 4 линии данных и 12 линий адреса (4 из которых были к тому же линиями данных), а 16 линий данных и 20 . . . 24 независимые линии адреса. Первые микропроцессоры подходили только для калькуляторов и простых контроллеров, а современные микропроцессоры можно использовать в качестве ЦП сложных компьютеров широкого назначения. Новейшие 16-битные однокристальные микропроцессоры допускают адресацию памяти до 24М байт и имеют средства мультипрограммирования и организации мультипроцессорных систем. Вместе с новыми микропроцессорами выпускаются разнообразные микросхемы, предназначенные для реализации памяти, интерфейсов и управления шиной.

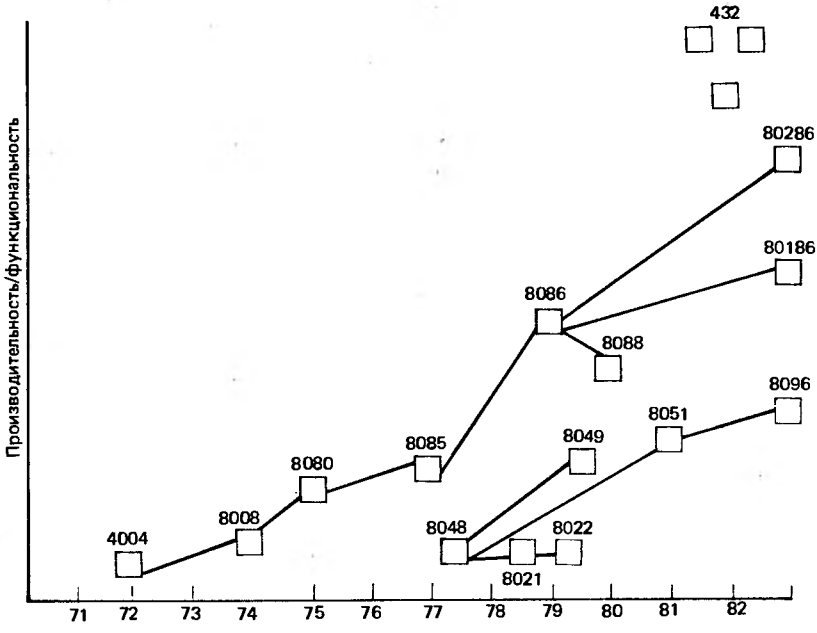


Рис. 1.12. Основные микропроцессорные семейства фирмы Intel

Рассмотрим для иллюстрации развитие базового семейства микропроцессоров фирмы Intel, показанное на рис. 1.12. Это семейство началось с первого 4-битного микропроцессора 4004 и семейства 4000 и пришло к 16-битному микропроцессору 8086 с рассчитанными на него многочисленными вспомогательными микросхемами. Микропроцессоры 8008, 8080 и 8085 относятся к классу 8-битных, причем каждый следующий из них оказывался более сложным и гибким. Микропроцессор 8088 представляет собой 8-битный вариант микропроцессора 8086; он имеет меньше линий данных, но сохраняет все функциональные возможности микропроцессора 8086. Микропроцессоры 80186 и 80286 являются однокристальными расширениями микропроцессора 8086, обладающими более высоким быстродействием и дополнительными вычислительными возможностями, которые особенно важны при проектировании сложных микросистем. На рис. 1.12 отражены также семейство однокристальных микрокомпьютеров 8048 и многокристальная система 432. Производительность семейства микропроцессоров фирмы Intel за прошедшие годы увеличилась в тысячи раз.

Упражнения

1. Преобразуйте следующие десятичные числа в двоичные:
а) 19, б) 79, в) 463, г) 1209, д) 11355.
2. Преобразуйте следующие двоичные числа в десятичные:
а) 10110, б) 1011011, в) 10010010, г) 1011010001110110.

3. Преобразуйте следующие числа в 16-ричную систему счисления:

а) 35_{10}, б) 298_{10}, в) 852_{10}, г) 162023_{10}, д) 11010_2, е) 11010110_2, ж) 0110000100111011_2.

4. Найдите двоичные и десятичные эквиваленты следующих 16-ричных чисел:

а) D5, б) 9A26, в) 7BF52A, г) 2A01BF57.

5. Пусть

$$A = 1011,\ B = 11001,\ C = 100110,\ D = 1000010011.$$

Выполните следующие операции в двоичной системе и проверьте ответы, пользуясь десятичной арифметикой:

а) A + B, б) C − B − A, в) AB, г) D/A, д) AC/B, е) D −AB.

6. Пусть A = 4F5A20 и B = 2FF609. Найдите результаты:

а) A + B, б) A − B, в) 2B − A, г) A/4.

7. Оцените приближенные значения 2^{24}, 2^{32} и 2^{64} (напомним: $2^{10} \approx 10^3$).

8. Определите, сколько различных комбинаций можно представить последовательностями из 12 бит? 24 бит? 32 бит?

9. Найдите 8-битные дополнительные коды чисел:

а) 01101001, б) 11010010.

10. Пусть

$$A = 00110010,\ B = 01001010,\ C = 11101001,\ D = 10111010.$$

Выполните следующие операции в дополнительном коде:

а) A + B, б) A + C, в) C + B, г) C + D, д) A − B, е) C − A, ж) D − C, з) A + D − C.

11. Найдите 4-разрядные десятичные дополнительные коды чисел 0232 и 9644.

12. Пусть

$$A = 0561,\ B = 0352,\ C = 9863,\ D = 9179.$$

Выполните следующие операции в 4-разрядном десятичном дополнительном коде:

а) A + B, б) A + C, в) C + D, г) A − B, д) C − D, е) B − C.

13. Найдите упакованные BCD-эквиваленты (двоичные и 16-ричные) следующих чисел:

а) 3251, б) 12907.

14. Представьте следующие символьные цепочки в коде ASCII (пользуясь 16-ричными кодами):

а) SAM JONES, б) −75.61, в) Hello, how are you?

г) G.A. SMITH
281 DOVE
EL PASO, TEXAS

15. Покажите, что двоичное число, представляющее собой символы GREEN, ANNE, меньше числа, представляющего собой символы GREEN, MARY.

16. Предположим, что имеются 20 линий адреса и 16 линий данных.

а) Определите адресное пространство памяти, если адресные пространства памяти и ввода-вывода различны.

б) Каков диапазон целых чисел в дополнительном коде, которые можно передавать по линиям данных?

17. Повторите упр. 16, считая, что имеется 16 линий адреса и 8 линий данных.

2. АРХИТЕКТУРА МИКРОПРОЦЕССОРА 8086

16-битный микропроцессор 8086 фирмы Intel содержит на кристалле около 29 000 транзисторов и производится по высококачественной МОП-технологии. Производительность его значительно выше 8-битного предшественни-

ка — микропроцессора 8080. Хотя и имеется определенная совместимость микропроцессора 8086 с архитектурой ЦП 8080, разработчики не ставили перед собой цели достичь ее полностью. Число линий адреса увеличено с 16 до 20, что позволяет адресовать память 1М байт вместо 64К байт. Увеличение емкости памяти обеспечивает переход к мультипрограммированию, поэтому в микропроцессоре 8086 предусмотрено несколько мультипрограммных возможностей. Кроме того, в микропроцессор 8086 встроены некоторые средства, упрощающие реализацию мультипроцессорных систем, что позволяет применять его с другими процессорами, например с процессором числовых данных 8087.

На рис. 2.1 приведена разводка контактов корпуса микропроцессора 8086. Он имеет 20 линий адреса, 16 из которых используются и как линии данных. Это обстоятельство приводит к тому, что на системную шину нельзя одновременно выдавать адреса и данные. Мультиплексирование адресов и данных во времени сокращает число контактов корпуса, но и замедляет скорость передачи данных. Однако благодаря тщательно разработанной временной диа-

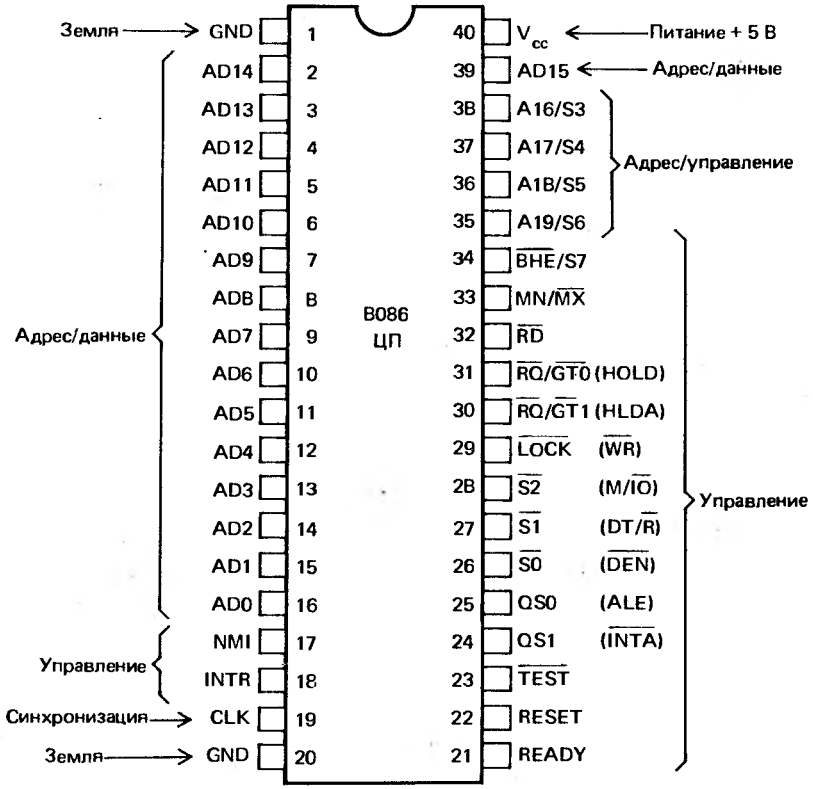


Рис. 2.1. Разводка контактов корпуса микропроцессора 8086

грамме работы скорость передачи уменьшается не столь значительно, как этого следовало бы ожидать. Микропроцессор имеет 16 линий управления, предназначенных для сигналов квитирования во время передач данных и внешнего управления ЦП. Он рассчитан на одно напряжение питания +5 В и однофазную синхронизацию, частота которой достигает 5 МГц. (Модель 8086-2 имеет частоту синхронизации до 8 МГц, а модель 8086-1 – до 10 МГц.) Два контакта *1* и *20* подключаются на землю.

Настоящая глава начинается с описания архитектуры ЦП 8086 и его внутренних операций. В § 2.3 обсуждаются режимы адресации данных, переходы и форматы машинных команд. В § 2.4 речь идет о времени выполнения команд, а § 2.5 посвящен 8-битному микропроцессору 8088, который программно совместим с микропроцессором 8086. Микропроцессор 8088 предназначен для перевода аппаратных конфигураций на базе микропроцессоров 8080/8085 на программную среду микропроцессора 8086.

2.1. АРХИТЕКТУРА ЦЕНТРАЛЬНОГО ПРОЦЕССОРА

Внутренняя архитектура микропроцессора 8086 представлена на рис. 2.2. За исключением регистра команд, которым фактически служит 6-байтная очередь, рассмотренные в § 1.4 регистры управления и рабочие регистры разделены на три группы в соответствии с выполняемыми ими функциями. Имеются группа регистров данных, представляющая собой, по существу, набор арифметических регистров; указательная группа, содержащая базовые

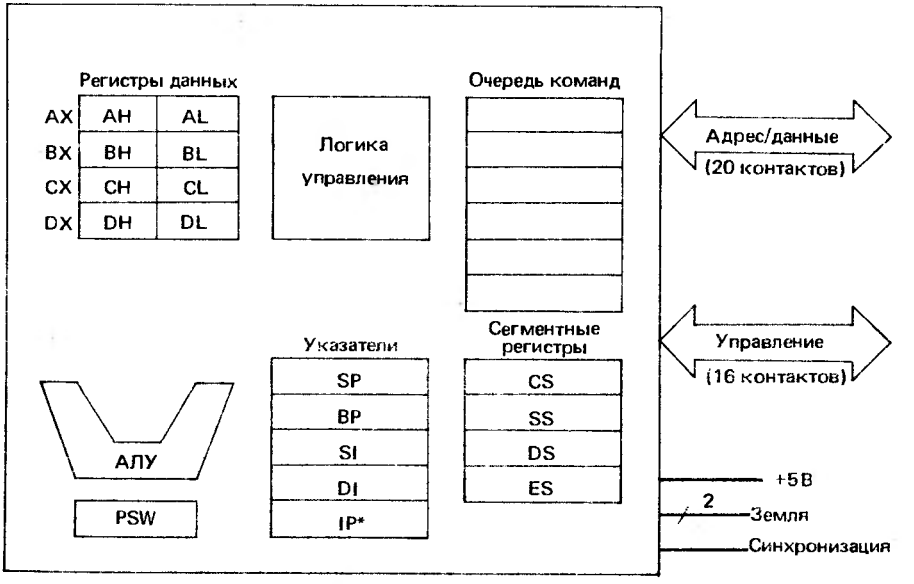


Рис. 2.2. Внутренняя конфигурация микропроцессора 8086

* В микропроцессоре 8086 программный счетчик называется указателем команды IP

и индексные регистры, а также программный счетчик и указатель стека, и сегментная группа, в состав которой входят специальные базовые регистры. Все регистры имеют длину 16 бит.

В группу регистров данных входят регистры AX, BX, CX и DX. Они предназначены для хранения операндов и результатов операций и допускают адресацию не только целых регистров, но и их младшей и старшей половин. Например, допускается использовать два байта в регистре AX вместе, а также указывать отдельные байты AL (младший) и AH (старший). С точки зрения программной совместимости с микропроцессором 8080 имеется следующее соответствие:

Микропроцессор 8086	Микропроцессор 8080
AL	A
BH	H
BL	L
CH	B
CL	C
DH	D
DL	E

Регистры BX, CX и DX кроме арифметических функций имеют и специальные назначения:

BX служит базовым регистром в вычислениях адреса,

CX в некоторых командах выступает неявным счетчиком,

DX в некоторых операциях ввода-вывода содержит адрес порта ввода-вывода.

Указательная и индексная группы представлены регистрами IP, SP, BP, SI и DI. Указатель команды IP и регистр SP фактически являются программным счетчиком и указателем стека, но полные адреса команды и стека образуются суммированием содержимого этих регистров и содержимого сегментных регистров CS и SS, которые рассматриваются далее. Регистр BP является базовым при обращении к стеку и может использоваться с другими регистрами и (или) смещением, которое является частью команды. Регистры SI и DI предназначены для индексирования. Хотя их можно использовать сами по себе, они часто комбинируются с регистрами BX и BP и (или) смещением. За исключением регистра IP, любой из указателей может хранить операнд, но допускает обращения только к 16-битному регистру.

Чтобы обеспечить гибкую базовую адресацию и индексирование, адрес данных формируют путем сложения содержимого регистров BX или BP, содержимого регистров SI или DI и смещения. Результат вычислений адреса называется *эффективным адресом* EA или *смещением*. (В руководствах фирмы Intel термин "эффективный адрес" применяется в контексте машинного языка, а термин "смещение" (*offset*) – в контексте языка ассемблера. Слово "смещение" (*displacement*) означает величину, которая прибавляется к содержимому регистра(ов) для образования EA.) Однако окончательный адрес данных определяется EA и соответствующим сегментным регистром – DS, ES или SS.

Сегментную группу образуют регистры CS, SS, DS и ES. Как говорилось ранее, участвующие в формировании адреса регистры BX, IP, SP, BP, SI и DI имеют длину всего 16 бит, поэтому эффективный адрес имеет такую же длину. Но, с другой стороны, выдаваемый на шину адреса *физический адрес* должен содержать 20 бит. Дополнительные 4 бита образуются при сложении эффективного адреса с содержимым одного из сегментных регистров, как показано на рис. 2.3. Перед сложением к содержимому сегментного регистра справа добавляются четыре нуля, что дает 20-битный результат. Если, например, (CS) = 123A и (IP) = 341B, следующая команда будет выбираться по адресу 157BB:

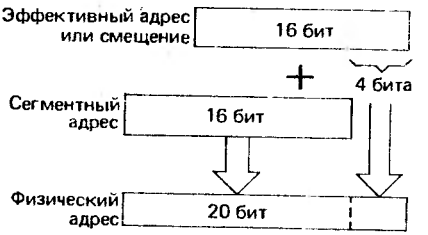


Рис. 2.3. Формирование физического адреса

	341B	эффективный адрес
+	123A0	начальный адрес сегмента
	157BB	физический адрес команды

(Отметим, что круглые скобки подразумевают слово "содержимое", например запись (IP) означает содержимое IP. Кроме того, все адреса даются в 16-ричной системе.)

Применение сегментных регистров, по существу, разделяет пространство памяти на перекрывающиеся сегменты, каждый из которых имеет размер 64К байт и начинается на 16-байтной границе (называемой границей параграфа), т. е. начинается с адреса, кратного 16. Далее мы будем называть содержимое сегментного регистра *сегментным адресом*, а сегментный адрес, умноженный на 16, – *начальным физическим сегментным адресом* или просто *на-*

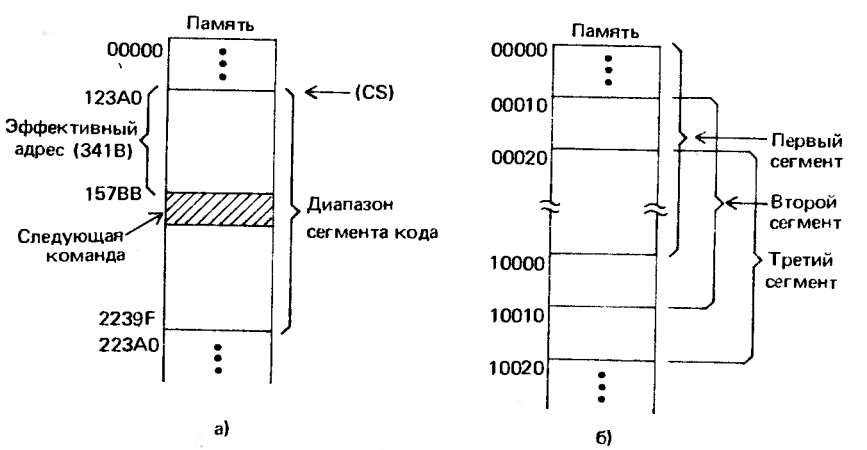


Рис. 2.4. Вычисление адреса (*а*) и сегментация памяти (*б*)

чальным сегментным адресом. Иллюстрацией рассмотренного примера служит рис. 2.4, *а*, а общий принцип сегментации памяти показан на рис. 2.4, *б*.

Наличие сегментных регистров обеспечивает следующие преимущества:

емкость памяти может доходить до 1М байт, хотя команды оперируют 16-битными адресами;

секции кода, данных и стека могут иметь длину более 64К байт благодаря использованию нескольких сегментов кода, данных или стека;

упрощается использование отдельных областей памяти для программы, ее данных и стека;

при каждом выполнении программы она сама и(или) ее данные могут размещаться в различных областях памяти.

На рис. 2.5 показано, каким образом можно распределить в памяти программный код, данные и стек. Самый простой и удобный вариант – разместить код и данные в единой области памяти, а стек – в некоторой фиксированной области, которая начинается, например, с адреса 08000. Такое решение приемлемо, когда в памяти всегда находится только одна программа, но в мультипрограммной среде в памяти одновременно находятся несколько программ. При мультипрограммировании лучше всего делать смежные блоки памяти как можно меньше, поэтому выгоднее отделить код от данных. Вариант смежного размещения кода и данных существует всегда, так как в регистры CS, DS и ES всегда можно загрузить одно и то же содержимое. Мультипрограммирование подробно рассматривается в гл. 7.

Четвертое преимущество также связано с мультипрограммированием. Рассмотрим, например, систему с разделением времени, в которой несколько

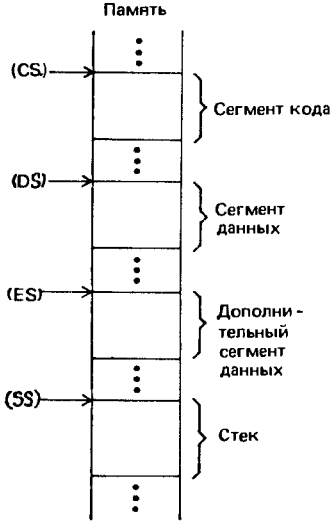


Рис. 2.5. Распределение кода программы, ее данных и стека

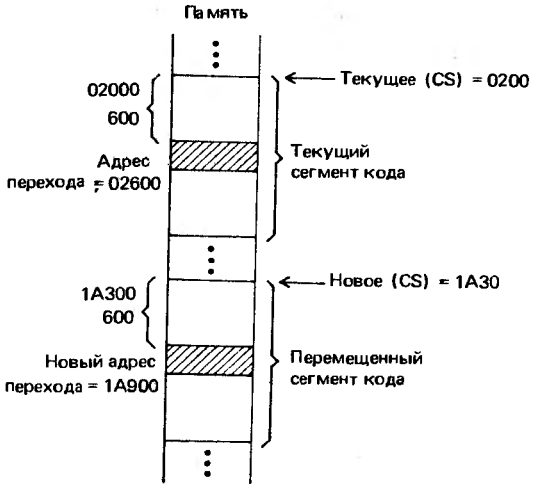


Рис. 2.6. Перемещение программы с помощью регистра CS

пользователей выполняют различные программы. Если в памяти места недостаточно, приходится обменивать программы между основной памятью и внешней. Следовательно, программы динамически размещаются в различных областях памяти. Например, при данном выполнении программа может начинаться с физического адреса 02000, а при следующей загрузке в память она может начинаться с адреса 1A300. Возникает вопрос: если в программе есть команда перехода, которая при данном выполнении вызывает переход по адресу 02600, будет ли осуществлен правильный переход по адресу 1A900, когда программа перемещается в область с начальным адресом 1A300? Ответ оказывается утвердительным, если при перемещении программы изменить содержимое CS на 1A300. Такая ситуация отражена на рис. 2.6. Аналогичные утверждения относятся и к перемещению блоков данных или стека. К перемещению мы вернемся в гл. 7.

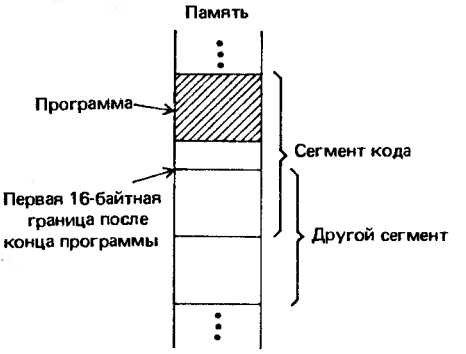


Рис. 2.7. Перекрывающиеся сегменты

Хотя сегменты имеют длину 64К байт, для лучшего использования памяти они могут перекрываться. Если, как показано на рис. 2.7, программа занимает только часть сегмента, то другой сегмент кода, данных или стека может перекрываться сегментом программы и начальный адрес второго сегмента может находиться в пределах 16 байт от конца программы. Могут "пропасть" только несколько байт между концом программы и следующей 16-байтной границей.

Слово состояния процессора PSW микропроцессора 8086 содержит 16 бит, но семь из них не используются. *Флажки* микропроцессора (см. рис. 2.8) 8086 разделяются на *условные* (или *флажки условий*), отражающие результат предыдущей операции АЛУ, и *управляющие* (или *флажки управления*), от которых зависит выполнение специальных функций. Младший байт в PSW соответствует 8-битному PSW микропроцессора 8080 и содержит все флажки условий, кроме OF.

Флажками условий являются:

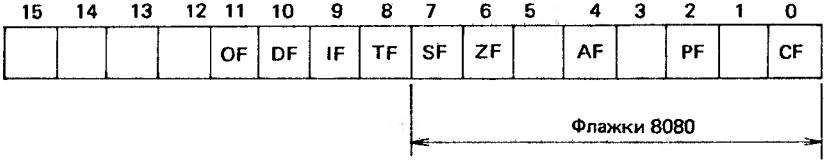


Рис. 2.8. Слово состояния микропроцессора 8086

Флажок знака SF. Равен старшему биту результата. Так как в дополнительном коде старший бит отрицательных чисел содержит 1, а у положительных чисел он равен 0, флажок SF показывает знак предыдущего результата.

Флажок нуля ZF. Устанавливается в 1 при получении нулевого результата и сбрасывается в 0, если результат отличается от нуля.

Флажок паритета PF. Устанавливается в 1, если младшие 8 бит результата содержат четное число единиц; в противном случае он сбрасывается в 0.

Флажок переноса CF. При сложении (вычитании) устанавливается в 1, если возникает перенос (заем) из старшего бита. Другие команды также воздействуют на этот флажок, что будет отмечено при их обсуждении.

Флажок вспомогательного переноса AF. Устанавливается в 1, если при сложении (вычитании) возникает перенос (заем) из бита 3. Флажок предназначен только для двоично-десятичной арифметики.

Флажок переполнения OF. Устанавливается в 1, если возникает переполнение, т. е. получение результата вне допустимого диапазона. При сложении этот флажок устанавливается, если имеется перенос в старший бит и нет переноса из старшего бита или наоборот. При вычитании он устанавливается, когда возникает заем из старшего бита, но заем в старший бит отсутствует, или наоборот.

Пусть, например, предыдущая команда производила следующее сложение:

$$\begin{array}{r} +0010\ \ 0011\ \ 0100\ \ 0101 \\ 0011\ \ 0010\ \ 0001\ \ 1001 \\ \hline 0101\ \ 0101\ \ 0101\ \ 1110 \end{array}$$

Тогда после ее выполнения получаются такие состояния флажков:

$$SF = 0,\quad ZF = 0,\quad PF = 0,\quad CF = 0,\quad AF = 0,\quad OF = 0.$$

Если в предыдущей команде выполнялось сложение

$$\begin{array}{r} +0101\ \ 0100\ \ 0011\ \ 1001 \\ 0100\ \ 0101\ \ 0110\ \ 1010 \\ \hline 1001\ \ 1001\ \ 1010\ \ 0011 \end{array}$$

флажки принимают следующие состояния:

$$SF = 1,\quad ZF = 0,\quad PF = 1,\quad CF = 0,\quad AF = 1,\quad OF = 1.$$

Флажки управления микропроцессора 8086:

Флажок направления DF. Применяется в командах манипуляций цепочками. Если он сброшен, цепочка обрабатывается с первого элемента, имеющего наименьший адрес. В противном случае цепочка обрабатывается от наибольшего адреса к наименьшему.

Флажок разрешения прерываний IF. Когда установлен этот флажок, ЦП распознает маскируемые прерывания; в противном случае эти прерывания игнорируются (подробнее см. гл. 4 и 6).

Флажок прослеживания (трассировки) TF. Когда этот флажок установлен, после выполнения каждой команды генерируется внутреннее прерывание (см. гл. 4).

2.2. ВНУТРЕННИЕ ОПЕРАЦИИ МИКРОПРОЦЕССОРА

В общем виде действия компьютера включают в себя следующие шаги (см. § 1.4):

1. Выборка следующей команды по адресу из PC.
2. Загрузка ее в регистр команды и дешифрирование с одновременным инкрементом PC для адресации следующей по порядку команды.
3. Выполнение команды, а в случае перехода загрузка в PC адреса перехода.
4. Повторение шагов 1 – 3.

Действия микропроцессора 8086 соответствуют этому общему шаблону, но имеются некоторые различия и совмещение нескольких операций.

Адрес следующей команды равен сумме (IP) и (CS) × 16_{10}, а регистр команды представлен 6-байтной очередью FIFO ("первый пришел – первый ушел"), которая непрерывно заполняется, когда системная шина не требуется для других операций. Такое "опережение" значительно увеличивает про-

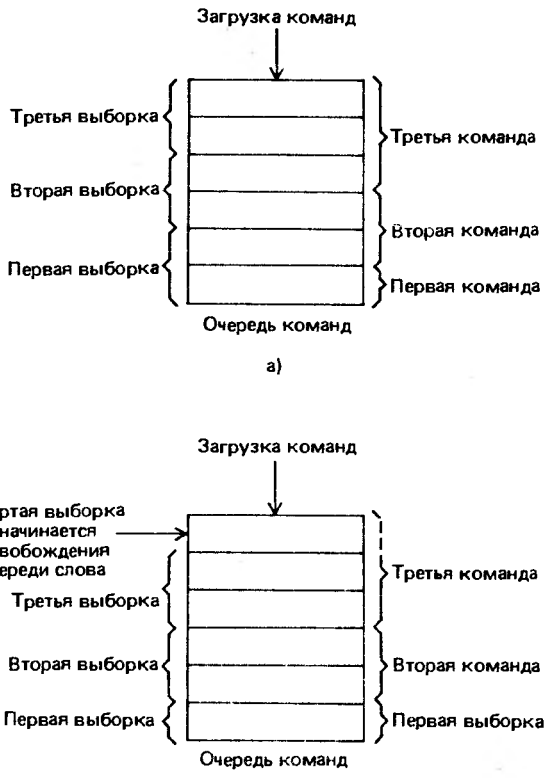


Рис. 2.9. Заполнение очереди команд после перехода, когда первая команда имеет четный (*а*) и нечетный (*б*) адрес

пускную способность ЦП, так как к моменту завершения текущей команды следующая команда чаще всего уже находится в ЦП. В случае перехода очередь сбрасывается и не дает экономии времени, но в среднем это происходит нечасто. Чтобы показать эффект опережающей загрузки очереди, рассмотрим команду 8-битного умножения. Эта команда выполняется минимум 71 такт синхронизации, но для считывания слова из памяти требуется всего 4 такта. Следовательно, при выполнении команды оказывается много тактов синхронизации, во время которых шина свободна для заполнения очереди команд.

Хотя микропроцессор 8086 может обращаться к слову по любому адресу, при нечетном адресе требуются два обращения к памяти: для младшего и для старшего байтов. Следовательно, возможна некоторая экономия времени, если хранить слова только по четным адресам. Длина команд составляет от одного до шести байт, но очередь позволяет считывать команды словами по четным адресам. Имеется только одно исключение, связанное с переходом по нечетному адресу. В этом случае ЦП считывает сначала один байт, а затем продолжает считывать снова по четным адресам. На рис. 2.9, *а* показано заполнение очереди такой последовательностью команд: 1-байтная команда, 2-байтная команда и 3-байтная команда. Предполагается, что эта последовательность начинается по четному адресу. На рис. 2.9, *б* показана эта же последовательность, но начинающаяся по нечетному адресу. Отметим, что во втором случае последний байт третьей команды не передается в очередь до тех пор, пока в ней не образуется пустое слово.

2.3. МАШИННЫЕ КОМАНДЫ

Команда разделяется на группы бит или *поля*, причем поле *кода операции* (*КОП*) показывает, что должен делать компьютер, а остальные поля, называемые *операндами*, идентифицируют требуемую команде информацию. Операнд может содержать данное, часть адреса данного, косвенный указатель данного или другую информацию, относящуюся к обрабатываемым командой данным. Общий формат команды представлен на рис. 2.10.

Команды могут содержать несколько операндов, но чем больше операндов и чем они длиннее, тем больше места занимает команда в памяти и тем больше времени требуется для передачи ее в ЦП. Чтобы минимизировать общее число бит в команде, большинство команд, особенно в 16-битных компьютерах, имеют один или два операнда, причем минимум одним из операндов в двухоперандной команде является регистр. Так как пространства памяти и ввода-вывода относительно велики, адреса памяти и ввода-вывода требуют сравнительно много бит, а из-за ограниченного числа регистров для определения регистра требуются всего несколько бит. Следовательно, для экономии длины команды следует максимально пользоваться регистрами. Ограничение двумя операндами, конечно, уменьшает гибкость многих команд,

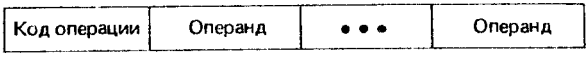


Рис. 2.10. Общий формат команды

но в действительности излишняя гибкость и не нужна. Например, команда сложения, в которой приходится указывать два слагаемых и результат, приводится к двум операндам посредством загрузки суммы на место одного из слагаемых. При этом оно теряется, но обычно это не играет роли. Если же слагаемое потребуется в дальнейшем, его приходится дублировать (запоминать где-то еще) до выполнения сложения.

2.3.1. РЕЖИМЫ АДРЕСАЦИИ

Способ определения операнда называется *режимом адресации*. Рассмотрим наиболее типичные режимы адресации микропроцессора 8086. Они разделяются на два класса – режимы адресации данных и режимы адресации переходов. На рис. 2.11 графически показаны определения операндов для различных режимов адресации данных. Различают следующие режимы адресации.

Непосредственный. Данное длиной 8 или 16 бит является частью команды.

Прямой. 16-битный эффективный адрес данного является частью команды.

Регистровый. Данное содержится в определяемом командой регистре. 16-битный операнд может находиться в регистрах AX, BX, CX, DX, SI, DI, SP или BP, а 8-битный – в регистрах AL, AH, BL, BH, CL, CH, DL или DH.

Регистровый косвенный. Эффективный адрес данного находится в базовом регистре BX или индексном регистре, определяемых командой:

$$EA = \left\{ \begin{matrix} (BX) \\ (SI) \\ (DI) \end{matrix} \right\}.$$

Регистровый относительный. Эффективный адрес равен сумме 8- или 16-битного смещения и содержимого базового или индексного регистров:

$$EA = \left\{ \begin{matrix} (BX) \\ (BP) \\ (SI) \\ (DI) \end{matrix} \right\} + \left\{ \begin{matrix} \text{8-битное смещение} \\ \text{16-битное смещение} \end{matrix} \right\}.$$

Базовый индексный. Эффективный адрес равен сумме содержимого базового и индексного регистров, определяемых командой:

$$EA = \left\{ \begin{matrix} (BX) \\ (BP) \end{matrix} \right\} + \left\{ \begin{matrix} (SI) \\ (DI) \end{matrix} \right\}.$$

Относительный базовый индексный. Эффективный адрес равен сумме 8- или 16-битного смещения и базово-индексного адреса:

$$EA = \left\{ \begin{matrix} (BX) \\ (BP) \end{matrix} \right\} + \left\{ \begin{matrix} (SI) \\ (DI) \end{matrix} \right\} + \left\{ \begin{matrix} \text{8-битное смещение} \\ \text{16-битное смещение} \end{matrix} \right\}.$$

Предположим, что (BX) = 0158, (DI) = 10A5, смещение = 1B57, (DS) = = 2100 и что в качестве сегментного регистра применяется DS. Тогда различные режимы адресации дают следующие эффективные и физические адреса:

Прямой: EA = 1B57,
Физический адрес = 1B57 + 21000 = 22B57.

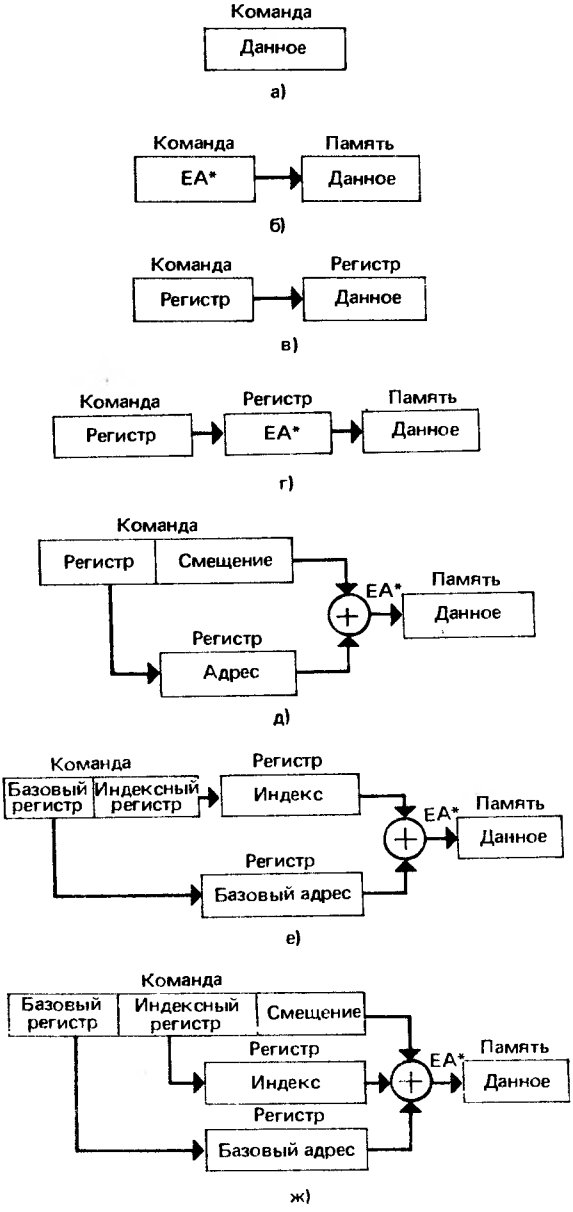
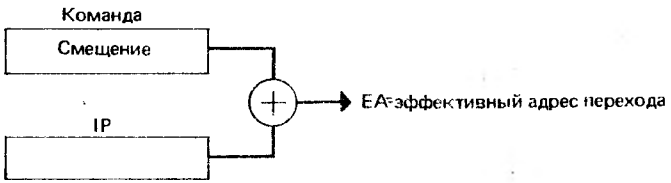


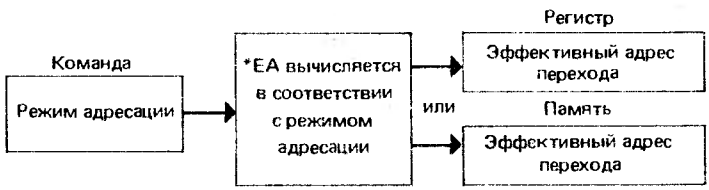
Рис. 2.11. Непосредственный (*а*), прямой (*б*), регистровый (*в*), регистровый косвенный (*г*), регистровый относительный (*д*), базовый индексный (*е*), относительный базовый индексный (*ж*) режимы адресации данных

* EA суммируется с умножением на 16 содержимым соответствующего сегментного регистра

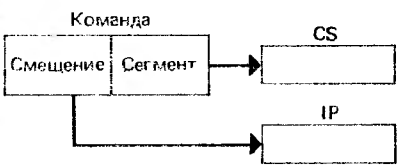
Регистровый: EA нет – данное в указанном регистре.
Регистровый косвенный (с участием регистра BX):
EA = 0158,
Физический адрес = 0158 + 21000 = 21158.
Регистровый относительный (с участием регистра BX):
EA = 0158 + 1B57 = 1CAF;
Физический адрес = 1CAF + 21000 = 22CAF.



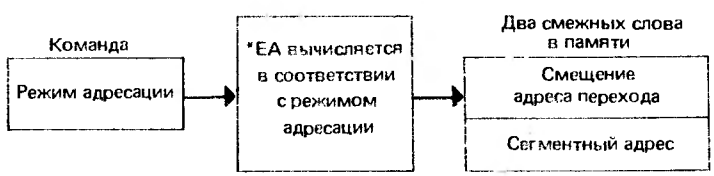
а)



б)



в)



г)

Рис. 2.12. Внутрисегментный прямой (*а*), внутрисегментный косвенный (*б*), межсегментный прямой (*в*), межсегментный косвенный (*г*) режимы адресации переходов
* EA суммируется с умноженным на 16 содержимым соответствующего сегментного регистра

Базовый индексный (с участием регистров BX и DI):

EA = 0158 + 10A5 = 11FD,

Физический адрес = 11FD + 21000 = 221FD.

Относительный базовый индексный (с участием регистров BX и DI):

EA = 0158 + 10A5 + 1B57 = 2D54,

Физический адрес = 2D54 + 21000 = 23D54.

Режимы для указания адресов перехода графически определены на рис. 2.12. Рассмотрим их подробнее.

Внутрисегментный прямой. Эффективный адрес перехода равен сумме 8- или 16-битного смещения и текущего содержимого IP. Когда смещение имеет длину 8 бит, этот режим называется коротким переходом. Внутрисегментную прямую адресацию в большинстве книг называют относительной адресацией, так как смещение вычисляется "относительно" IP. Этот режим допустим в условных и безусловных переходах, но в команде условного перехода может быть только смещение длиной 8 бит.

Внутрисегментный косвенный. Эффективный адрес перехода есть содержимое регистра или ячейки памяти, которые указываются в любом режиме (кроме непосредственного) адресации данных. Содержимое IP заменяется эффективным адресом перехода. Данный режим допустим только в командах безусловного перехода.

Межсегментный прямой. Заменяет содержимое IP одной частью команды, а содержимое CS – другой частью команды. Назначение данного режима адресации – обеспечить переход из одного сегмента кода в другой.

Межсегментный косвенный. Заменяет содержимое IP и CS содержимым двух смежных слов из памяти, которые определяются в любом режиме адресации данных кроме непосредственного и регистрового.

Отметим, что физический адрес перехода равен сумме нового содержимого IP и содержимого CS, умноженного на 16_{10}. Межсегментный переход может быть только безусловным.

Для иллюстрации косвенного перехода с некоторыми режимами адресации данных предположим, что (BX) = 1256, (SI) = 528F, смещение = 20A1. Тогда при прямой адресации эффективный адрес перехода равен 20A1 + + (DS) × 16_{10}. При регистровой косвенной адресации (с участием регистра BX) эффективный адрес перехода 1256 + 20A1 + (DS) × 16_{10}. Наконец, в базовой индексной адресации (с участием регистров BX и SI) эффективный адрес перехода 1256 + 528F + (DS) × 16_{10}.

2.3.2. ФОРМАТЫ КОМАНД

Несколько типичных форматов команд микропроцессора 8086 приведены на рис. 2.13. Длина команд варьируется от одного до 6 байт, а полный перечень команд дается в § 3.12. Длина смещений и непосредственных данных может быть 8 или 16 бит в зависимости от команды. В первых одном или двух байтах команды находятся код операции и указание режима адресации. После них могут находиться:

ни одного дополнительного байта;

двухбайтный EA (только для прямой адресации);

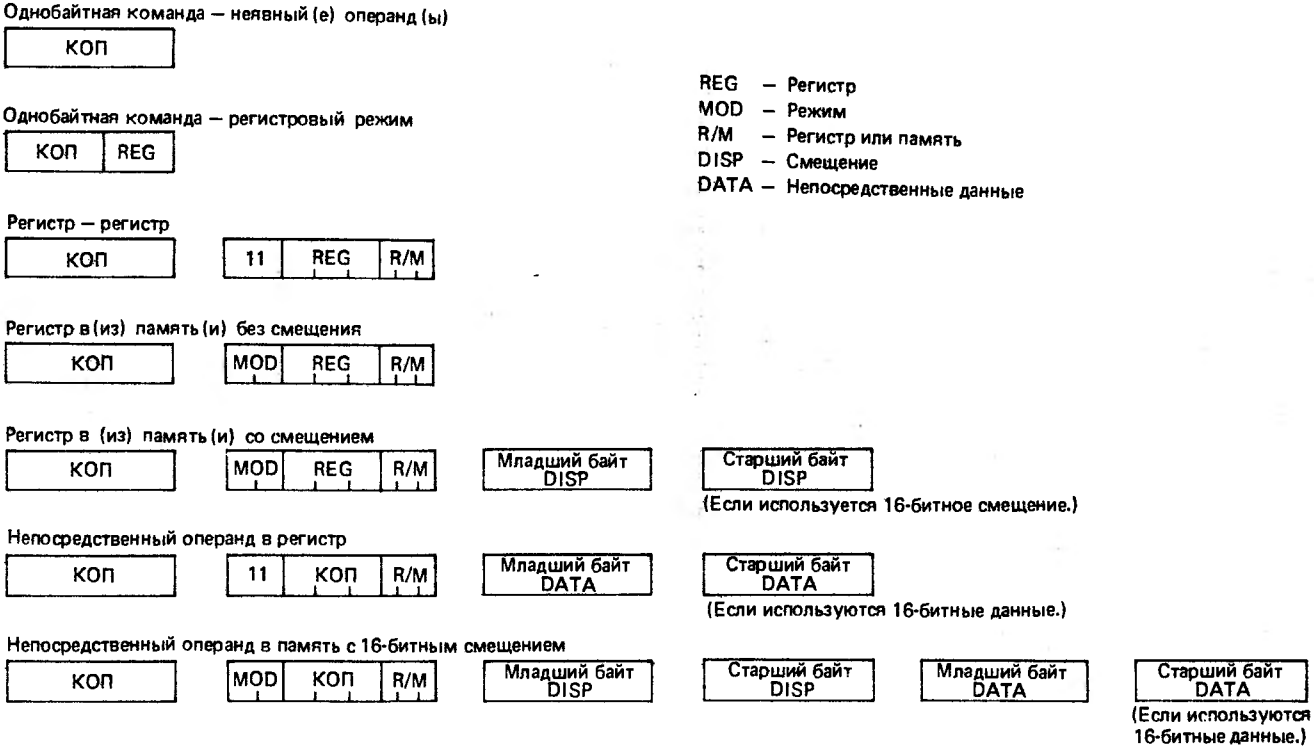


Рис. 2.13. Примеры форматов команд микропроцессора 8086

одно- или двухбайтное смещение;

одно- или двухбайтный непосредственный операнд;

одно- или двухбайтное смещение с последующим одно- или двухбайтным непосредственным операндом;

двухбайтное смещение и двухбайтный сегментный адрес (только для прямой межсегментной адресации).

Применение одной из перечисленных возможностей определяется кодом операции и режимом адресации. Если длина смещения или непосредственного операнда составляет два байта, первым всегда следует младший байт.

Обычно код операции занимает первый байт команды, но в некоторых командах в первом же байте указывается регистр, а в нескольких других командах три бита кода операции находятся во втором байте. В большинстве кодов операций имеются следующие однобитные индикаторы:

Бит W. Если команда может оперировать байтом и словом, в коде операции имеется бит W, который определяет операцию с байтом (W = 0) или словом (W = 1).

Бит D. Содержится в двухоперандных командах (за исключением команд с непосредственным операндом и цепочечных команд, которые рассматриваются в гл. 5). Одним из операндов должен быть регистр, определяемый полем REG. В таких командах бит D показывает, чем является регистр: операндом-источником (D = 0) или операндом-получателем (D = 1).

Бит S. 8-битное число в дополнительном коде можно расширить до 16-битного в дополнительном коде, если сделать все биты старшего байта равными старшему биту младшего байта. Такая операция называется расширением знака (или расширением со знаком). Бит S появляется вместе с битом W в командах сложения, вычитания и сравнения с непосредственным операндом и расшифровывается следующим образом:

8-битная операция – S:W = 00;

16-битная операция с 16-битным непосредственным операндом – S:W = 01;

16-битная операция с 8-битным операндом, который расширяется со знаком до 16 бит, – S:W = 11.

При небольших числах последний вариант допускает использовать однобайтный непосредственный операнд.

Бит V. Применяется в командах сдвигов для определения числа сдвигов (см. гл. 3).

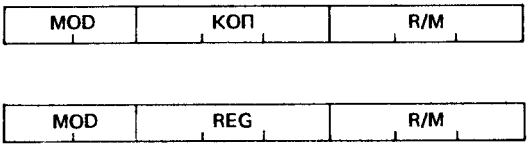
Адрес регистра	Регистры		Адрес регистра	Сегментный регистр
	W = 1	W = 0		
000	AX	AL	00	ES
001	CX	CL	01	CS
010	DX	DL	10	SS
011	BX	BL	11	DS
100	SP	AH		
101	BP	CH		
110	SI	DH		
111	DI	BH		

Рис. 2.14. Адреса регистров

Бит Z. Используется в команде REP (см. гл. 5).

Сегментный регистр определяется двумя битами, а любой другой регистр — тремя. Адреса регистров (т.е. двоичные обозначения их) приведены на рис. 2.14. Неоднозначности использования, например, 00 для указания сегментного регистра ES и 000 для указания регистров AX и AL не возникает, так как нужный регистр подразумевается кодом операции.

Если на код операции и режим адресации отводятся два байта, второй байт имеет одну из следующих двух форм:



Первая из них предназначена для однооперандных команд (или двухоперандных, в которых один из операндов неявно определяется кодом операции). Вторая форма характерна для двухоперандных команд, причем поле REG определяет регистр, который в зависимости от значения бита D является операндом-источником или операндом-получателем.

Операнд, указываемый полями MOD и R/M, определяется в соответствии с таблицей, приведенной на рис. 2.15. Если MOD ≠ 11, эффективный адрес вычисляется согласно таблице. Отметим, что MOD = 00 означает отсутствие

MOD / R/M	00	01	10	11 W = 0	11 W = 1
000 Сегментный регистр	(BX) + (SI) DS	(BX) + (SI) + DB DS	(BX) + (SI) + D16 DS	AL	AX
001 Сегментный регистр	(BX) + (DI) DS	(BX) + (DI) +D8 DS	(BX) + (DI) + D16 DS	CL	CX
010 Сегментный регистр	(BP) + (SI) SS	(BP) + (SI) + DB SS	(BP) + (SI) + D16 SS	DL	DX
011 Сегментный регистр	(BP) + (DI) SS	(BP) + (DI) + D8 SS	(BP) + (DI) + D16 SS	BL	BX
100 Сегментный регистр	(SI) DS	(SI) + DB DS	(SI) + D16 DS	AH	SP
101 Сегментный регистр	(DI) DS	(DI) + DB DS	(DI) + D16 DS	CH	BP
110 Сегментный регистр	D16 DS	(BP) + DB SS	(BP) + D16 SS	DH	SI
111 Сегментный регистр	(BX) DS	(BX) + D8 DS	(BX) + D16 DS	BH	DI

DB: 8-битное смещение (расширяется со знаком)

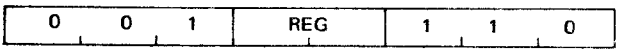
D16: 16-битное смещение

Рис. 2.15. Режимы адресации и сегментные регистры по умолчанию для различных комбинаций полей MOD и R/M

смещения, за исключением случая R/M = 110, который обозначает прямую адресацию. Комбинация MOD = 01 означает, что третий байт команды содержит 8-битное смещение, которое до вычисления эффективного адреса автоматически расширяется со знаком до 16 бит. Если MOD = 10, третий и четвертый байты команды содержат 16-битное смещение. Наконец, в случае MOD = 11 операндом является регистр, адрес которого определяется полем R/M.

На рис. 2.15 также показаны сегментные регистры, используемые в каждой из комбинаций полей MOD и R/M. Эффективный адрес операнда в памяти определяется полями MOD и R/M, но 20-битный физический адрес равен сумме эффективного адреса и содержимого сегментного регистра, умноженного на 16. В режимах адресации с привлечением регистра BP с эффективным адресом суммируется содержимое сегментного регистра SS, а в остальных режимах адресации участвует регистр DS.

Чтобы изменить используемые в соответствии с рис. 2.15 сегментные регистры, предусмотрена специальная однобайтная команда, называемая *префиксом замены сегмента*. Она имеет следующий формат:



Если команде предшествует префикс замены сегмента, при обращении к данным в процессе ее выполнения участвует сегментный регистр REG. Для приведенных на рис. 2.15 режимов адресации регистр DS можно заменить на CS, SS или ES, а регистр SS при участии в адресации регистра BP – на DS, CS или ES. Замену нельзя производить в следующих специальных случаях:

при вычислении адреса следующей выполняемой команды в качестве сегментного регистра всегда применяется CS;

при участии в адресации регистра SP сегментным регистром всегда служит SS (о стековых операциях см. гл. 4);

в цепочечных командах в качестве сегментного регистра операнда-получателя всегда используется ES (подробнее см. гл. 5).

Чтобы глубже разобраться в машинных командах микропроцессора 8086, рассмотрим команду сложения ADD. В команде ADD содержимое операнда-источника прибавляется к содержимому операнда-получателя и сумма заменяет содержимое операнда-получателя. В зависимости от режима адресации команда ADD принимает одну из трех форм, представленных на рис. 2.16.

На рис. 2.17 показан машинный код двух команд ADD, каждая из которых выполняет следующую операцию:

$$(CL) \leftarrow (BH) + (CL).$$

В первой команде бит D = 1 указывает, что сумма помещается в регистр REG = 001 = CL. Комбинация MOD = 11, поэтому R/M обозначает регистр 111 = BH. Во второй команде бит D = 0, поэтому регистр REG = 111 = BH служит источником. По-прежнему MOD = 11 и R/M обозначает регистр, которым здесь является 001 = CL.

На рис. 2.18, *а* показана команда, в которой для сложения содержимого ячейки памяти и регистра применяется относительный базовый индексный

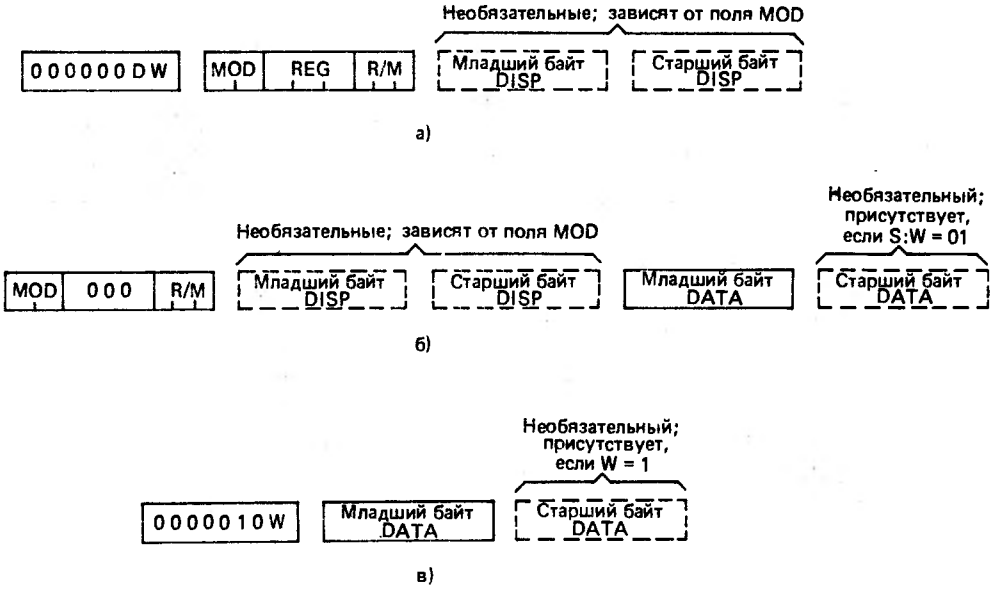


Рис. 2.16. Форматы команды сложения ADD в следующих операциях:
а – прибавить регистр к регистру или памяти и запомнить результат в регистре или памятн; *б* – прибавить непосредственный операнд к регистру (памяти) и поместить результат в регистр (память); *в* – прибавить непосредственный операнд к AX (AL) и запомнить результат в AX (AL) – специальный случай для аккумулятора

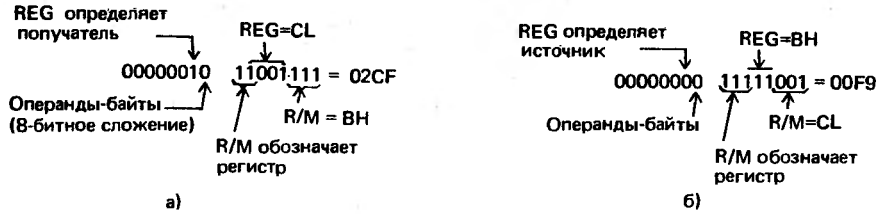


Рис. 2.17. Две эквивалентные команды прибавления содержимого регистра BH к содержимому CL: когда REG определяет получатель (*а*) и когда REG определяет источник (*б*)

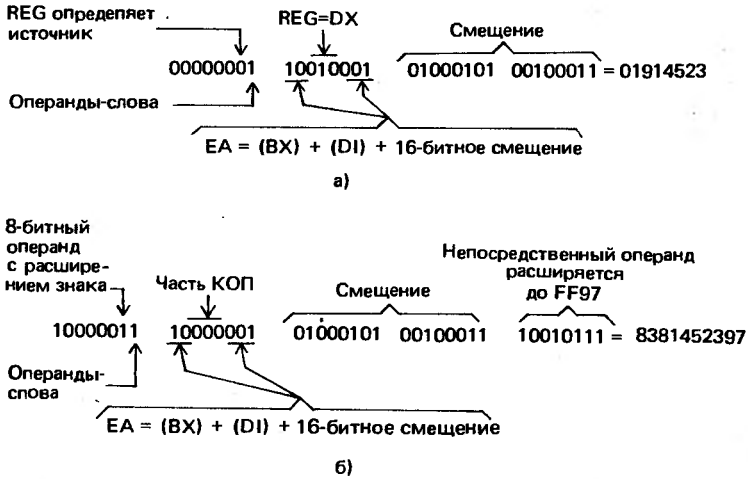


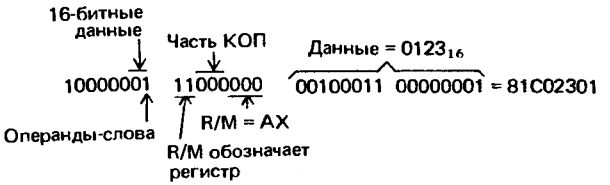
Рис. 2.18. Два примера команды ADD с относительным базовым режимом адресации: *а* — сложение регистра с памятью, *б* — сложение непосредственного операнда с памятью

режим адресации. Состояние D = 0 показывает, что сумма помещается в ячейку памяти, а бит W = 1 означает 16-битное сложение. Эффективный адрес равен сумме (BX), (DI) и 16-битного смещения, которое равно 2345. Если (BX) = 0892 и (DI) = 59A3, то

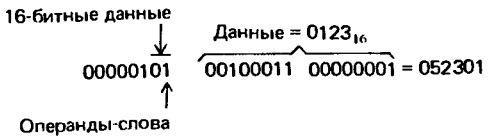
$$EA = 0892 + 59A3 + 2345 = 857A.$$

На рис. 2.18, *б* показана похожая команда, которая прибавляет к содержимому ячейки памяти непосредственный операнд.

Некоторые команды имеют длинную и короткую формы. Эти формы команды, которая прибавляет непосредственный операнд к регистру AX, показаны на рис. 2.19. Так как S:W = 01, команда содержит два байта данных. В длинной форме регистр AX явно указан полем R/M, а в короткой форме AX неявно определяется кодом операции. Короткие формы более подробно рассматриваются в § 3.12.



а)



б)

Рис. 2.19. Длинная (*а*) и короткая (*б*) формы прибавления непосредственного операнда к регистру AX

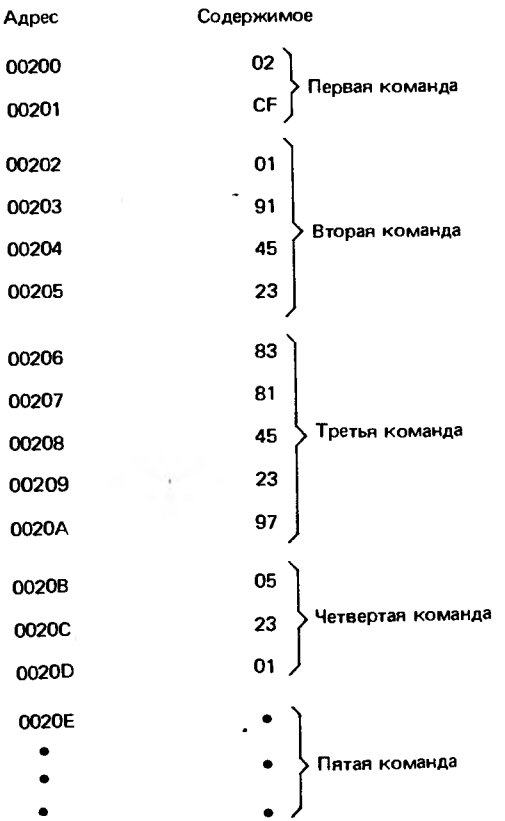


Рис. 2.20. Последовательность команд в памяти

На рис. 2.20 показано размещение команд из рис. 2.17, *а*, 2.18 и 2.19, *б* в памяти, начиная с адреса 00200. Если содержимое CS равно 0018, при выполнении этих команд будет произведен инкремент IP от 0080 до 008E.

2.4. ВРЕМЯ ВЫПОЛНЕНИЯ КОМАНД

Время выполнения команды можно определить, умножая число тактов синхронизации, необходимых для выполнения команды, на период синхронизации. Это время можно выразить в виде суммы базового времени выполнения (которое зависит от команды и режима адресации) и времени вычисления эффективного адреса, если привлекается операнд из памяти. Базовое время выполнения предполагает, что выполняемая команда уже выбрана и находится в очереди команд; в противном случае требуется учесть дополнительные такты синхронизации, необходимые для выборки команды. Базовые времена выполнения некоторых типичных команд микропроцессоров 8086/8088 приведены на рис. 2.21, а полная информация содержится в приложении. Время вычисления эффективного адреса зависит от режима адресации, что показано на рис.2.22.

Третий столбец на рис. 2.21 показывает число обращений к памяти, необходимых для выполнения команд. Чтобы определить время выполнения команды, в которой осуществляется обращение к полному слову памяти, следует учесть выравнивание операнда. Если слово имеет нечетный адрес, микропроцессор 8086 считывает его за два цикла шины, длящихся по 4 такта синхронизации. Следовательно, каждое обращение к слову с нечетным адресом требует 4 дополнительных такта синхронизации. В микропроцессоре 8088 на передачу каждого слова необходимо прибавить 4 такта синхронизации, так как в цикле шины передается только один байт.

	КОМАНДА	ЧИСЛО ТАКТОВ	ЧИСЛО ПЕРЕДАЧ
ADD	(СЛОЖИТЬ) ИЛИ SUB (ВЫЧЕСТЬ)		
	РЕГИСТР - РЕГИСТР	3	0
	ПАМЯТЬ - РЕГИСТР	9 + EA	1
	РЕГИСТР - ПАМЯТЬ	16 + EA	2
	НЕПОСРЕДСТВЕННЫЙ - РЕГИСТР	4	0
	НЕПОСРЕДСТВЕННЫЙ - ПАМЯТЬ	17 + EA	2
MOV	(ПЕРЕДАТЬ)		
	АККУМУЛЯТОР - ПАМЯТЬ	10	1
	ПАМЯТЬ - АККУМУЛЯТОР	10	1
	РЕГИСТР - РЕГИСТР	2	0
	ПАМЯТЬ - РЕГИСТР	8 + EA	1
	РЕГИСТР - ПАМЯТЬ	9 + EA	1
	НЕПОСРЕДСТВЕННЫЙ - РЕГИСТР	4	0
	НЕПОСРЕДСТВЕННЫЙ - ПАМЯТЬ	10 + EA	1
	РЕГИСТР - СЕГМЕНТНЫЙ РЕГИСТР	2	0
	ПАМЯТЬ - СЕГМЕНТНЫЙ РЕГИСТР	8 + EA	1
	СЕГМЕНТНЫЙ РЕГИСТР - РЕГИСТР	2	0
	СЕГМЕНТНЫЙ РЕГИСТР - ПАМЯТЬ	9 + EA	1
MUL	(УМНОЖИТЬ БЕЗ ЗНАКА)		
	МНОЖИТЕЛЬ - 8 БИТ, РЕГИСТР	70 - 77	0
	МНОЖИТЕЛЬ - 16 БИТ, РЕГИСТР	118 - 133	0
	МНОЖИТЕЛЬ - 8 БИТ, ПАМЯТЬ	(76 - 83) + EA	1
	МНОЖИТЕЛЬ - 16 БИТ, ПАМЯТЬ	(124 - 139) + EA	1

IMUL	(УМНОЖИТЬ СО ЗНАКОМ)		
	МНОЖИТЕЛЬ – 8 БИТ, РЕГИСТР	80 – 98	0
	МНОЖИТЕЛЬ – 16 БИТ, РЕГИСТР	128 – 154	0
	МНОЖИТЕЛЬ – 8 БИТ, ПАМЯТЬ	(86 – 104) + EA	1
	МНОЖИТЕЛЬ – 16 БИТ, ПАМЯТЬ	(134 – 160) + EA	1
DIV	(РАЗДЕЛИТЬ БЕЗ ЗНАКА)		
	ДЕЛИТЕЛЬ – 8 БИТ, РЕГИСТР	80 – 90	0
	ДЕЛИТЕЛЬ – 16 БИТ, РЕГИСТР	144 – 162	0
	ДЕЛИТЕЛЬ – 8 БИТ, ПАМЯТЬ	(86 – 96) + EA	1
	ДЕЛИТЕЛЬ – 16 БИТ, ПАМЯТЬ	(150 – 168) + EA	1
IDIV	(РАЗДЕЛИТЬ СО ЗНАКОМ)		
	ДЕЛИТЕЛЬ – 8 БИТ, РЕГИСТР	101 – 112	0
	ДЕЛИТЕЛЬ – 16 БИТ, РЕГИСТР	165 – 184	0
	ДЕЛИТЕЛЬ – 8 БИТ, ПАМЯТЬ	(107 – 118) + EA	1
	ДЕЛИТЕЛЬ – 16 БИТ, ПАМЯТЬ	(171 – 190) + EA	1
КОМАНДЫ СДВИГОВ И ЦИКЛИЧЕСКИХ СДВИГОВ			
	РЕГИСТР, НА 1 БИТ	2	0
	РЕГИСТР, НА ПЕРЕМЕННОЕ ЧИСЛО БИТ	8 + 4/БИТ	0
	ПАМЯТЬ, НА 1 БИТ	15 + EA	2
	ПАМЯТЬ, НА ПЕРЕМЕННОЕ ЧИСЛО БИТ	20 + EA + 4/БИТ	2
JMP	(БЕЗУСЛОВНЫЙ ПЕРЕХОД)		
	КОРОТКИЙ	15	0
	ВНУТРИСЕГМЕНТНЫЙ ПРЯМОЙ	15	0
	МЕЖСЕГМЕНТНЫЙ ПРЯМОЙ	15	0
	ВНУТРИСЕГМЕНТНЫЙ КОСВЕННЫЙ (РЕГИСТРОВЫЙ РЕЖИМ)	11	0
	ВНУТРИСЕГМЕНТНЫЙ КОСВЕННЫЙ	18 + EA	1
	МЕЖСЕГМЕНТНЫЙ КОСВЕННЫЙ	24 + EA	2
КОМАНДЫ УСЛОВНОГО ПЕРЕХОДА			
	JCXZ	6 (НЕТ ПЕРЕХОДА) 18 (ПЕРЕХОД)	0
	ОСТАЛЬНЫЕ КОМАНДЫ УСЛОВНОГО ПЕРЕХОДА	4 (НЕТ ПЕРЕХОДА) 16 (ПЕРЕХОД)	0

Рис. 2.21. Времена выполнения некоторых команд

Отметим, что некоторые команды имеют несколько отличающихся базовых времен выполнения в зависимости от режимов адресации. Примеры на рис. 2.21 показывают, что для данной двухоперандной команды операция регистр — регистр выполняется быстрее, чем с остальными режимами адресации, а операция регистр — память выполняется медленнее операции память — регистр из-за дополнительного времени на запоминание результата в памяти. Базовое время выполнения некоторых команд зависит также от данных. Типичными примерами служат команды умножения, деления, сдвига и цикли-

ЭФФЕКТИВНЫЙ АДРЕС	ЧИСЛО ТАКТОВ
ПРЯМОЙ	6
РЕГИСТРОВЫЙ КОСВЕННЫЙ	5
РЕГИСТРОВЫЙ ОТНОСИТЕЛЬНЫЙ	9
БАЗОВЫЙ ИНДЕКСНЫЙ	
(BP) + (DI) ИЛИ (BX) + (SI)	7
(BP) + (SI) ИЛИ (BX) + (DI)	8
ОТНОСИТЕЛЬНЫЙ БАЗОВЫЙ ИНДЕКСНЫЙ	
(BP) + (DI) + DISP ИЛИ	11
(BX) + (SI) + DISP	
(BP) + (SI) + DISP ИЛИ	12
(BX) + (DI) + DISP	

Рис. 2.22. Время, необходимое для вычисления эффективного адреса

ческого сдвига. Для команд условного перехода приведено два времени – меньшее соответствует случаю, когда условие не удовлетворяется и переход не производится, а большее соответствует реализации перехода. Во втором случае учитывается нарушение очереди команд и выборка следующей команды.

Пусть частота синхронизации равна 5 МГц (период 0,2 мкс). Тогда времена выполнения различных форм команды ADD вычисляются следующим образом.

Сложение регистр – регистр (результат в регистре) требует трех тактов синхронизации (0,6 мкс) для операнда байта или слова.

Сложение память – регистр (результат в регистре) с использованием относительной базовой индексной адресации требует

9 (базовое) + 12 (вычисление адреса) = 21 такт синхронизации (4,2 мкс)

для операции над байтами или словами, причем слово находится по четному адресу, или

9 + 12 + 4 (дополнительные такты) = 25 тактов (5,0 мкс),

если слово находится по нечетному адресу.

Сложение регистр – память (результат в памяти) с использованием базовой индексной адресации требует

16 (базовое) + 8 (вычисление адреса) = 24 такта синхронизации (4,8 мкс)

для операции над байтами или словами, причем слово находится по четному адресу, или

16 + 8 + 4 (считывание слова) + 4 (запоминание слова) = 32 такта синхронизации (6,4 мкс),

если слово находится по нечетному адресу.

2.5. МИКРОПРОЦЕССОР 8088

Микропроцессор 8088 представляет собой 8-битный микропроцессор, который полностью совместим с микропроцессором 8086 (т. е. имеет такую же систему команд) и который можно применять в аппаратуре системы, реализованной на базе микропроцессоров 8080 или 8085. Он имеет 8 линий данных, как и микропроцессоры 8080/8085, но его архитектура аналогична архитектуре ЦП 8086. Разводка контактов корпуса микропроцессора 8088 такая же, как и у микропроцессора 8086, но линии адреса A15 – A8 используются только для адресов, а линия $\overline{\text{BHE}}$ заменена линией состояния, так как микропроцессор 8088 может обращаться только к байтам. Поскольку линии управления микропроцессоров 8088 и 8080/8085 различаются, при введении микропроцессора 8088 в систему на базе микропроцессоров 8080/8085 требуется значительно изменить логику управления шиной. Особенно важно учесть тот факт, что в микропроцессоре 8088, как и в микропроцессоре 8085, адреса и данные мультиплексируются. В подсистемах памяти и ввода-вывода потребуются небольшие изменения, если не увеличивать емкость памяти благодаря 20 линиям адреса.

В отличие от 6-байтной очереди команд микропроцессора 8086 в микропроцессоре 8088 длина очереди команд составляет 4 байта. Причина такого

уменьшения длины очереди заключается в том, что микропроцессор 8088 может считывать только байты и увеличение времени выборки не позволяет процессору полностью использовать 6-байтную очередь. Алгоритм опережающей выборки отличается тем, что микропроцессор 8088 инициирует выбор ку команды, когда в очереди оказывается один свободный байт, а не два, как в микропроцессоре 8086. В микропроцессоре 8088 сохранены сегментные регистры, адрес длиной 20 бит и средства поддержки мультипроцессорных систем. Сохранена также возможность обработки 16-битных операндов.

Упражнения

1. Если физический адрес перехода равен 5A230, когда (CS) = 5200, каким он будет при изменении (CS) на 7800?

2. Пусть EA данного равен 2359 и (DS) = 490B. Чему равен физический адрес данного?

3. Пусть (BX) = 632D, (SI) = 2A9B, смещение = C237. Определите эффективный адрес (если это возможно) для следующих режимов адресации:

а) непосредственный;

б) прямой;

в) регистровый с участием BX;

г) регистровый косвенный с участием BX;

д) регистровый относительный с использованием BX;

е) базовый индексный;

ж) относительный базовый индексный.

4. Пусть задано (IP) = 2BC0, (CS) = 0200, смещение = 5119, (BX) = 1200, (DS) = = 212A, (224A0) = 0600, (275B9) = 098A. Найдите адрес перехода в команде перехода, использующей:

а) внутрисегментную прямую адресацию;

б) внутрисегментную косвенную адресацию, в которой фигурируют регистр BX и регистровая адресация;

в) внутрисегментную косвенную адресацию с привлечением регистра BX и регистровой относительной адресации.

5. Найдите сумму и состояния флажков AF, SF, ZF, CF, OF и PF после прибавления 62A0 к следующим числам: а) 1234, б) 4321, в) CFA0, г) 9D60.

6. Найдите разность и состояния флажков SF, ZF, CF, OF и PF после вычитания 4AE0 из следующих чисел: а) 1234, б) 5D90, в) 9090, г) EA04.

7. Воспользуйтесь примерами из раздела 2.3.2 для нахождения машинного кода команды, которая:

а) прибавляет содержимое BX к содержимому DX и помещает результат в DX:

б) использует регистры BX и SI в базовой индексной адресации для прибавления байта из памяти к (AL) и помещения результата в AL;

в) использует регистр BX и смещение B2 в регистровой косвенной адресации для прибавления содержимого ячейки памяти к (CX) и загрузки результата в ячейку памяти;

г) использует смещение 0524 и прямую адресацию для прибавления числа 2A59 к содержимому ячейки памяти и помещения суммы в ячейку памяти;

д) прибавляет число B5 к (AL) и загружает сумму в AL (дайте длинную и короткую формы этой команды).

8. Воспользуйтесь рис. 2.21 для определения времени выполнения команды, которая:

а) прибавляет (BX) к (CX);

б) использует регистровую относительную адресацию для вычитания слова в памяти

из (AX) и загружает результат в AX (сначала считайте, что слово имеет четный адрес, а затем – нечетный адрес);

в) выполняет условный переход (сначала считайте, что переход реализуется, а затем – не реализуется).

Частота синхронизации равна 5 МГц.

3. ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ АССЕМБЛЕРА

За исключением программы на машинном языке, состоящей из комбинаций нулей и единиц, которые непосредственно дешифрирует компьютер, программа на языке ассемблера является простейшей. В языке ассемблера имеются операторы двух видов: *команды*, которые ассемблер транслирует в машинные команды, и *директивы*, которые служат указаниями ассемблеру во время процесса ассемблирования, но не транслируются в машинные команды.

Хотя каждая ассемблерная команда порождает только одну машинную команду, ее проще писать благодаря тому, что аббревиатуры, называемые *мнемониками*, показывают тип команды, а символьные цепочки, называемые *идентификаторами*, представляют собой адреса и, возможно, числа. Типичная ассемблерная команда

```
ADD AX,COST
```

прибавляет содержимое ячейки памяти, ассоциируемой с идентификатором COST, к регистру AX. Аббревиатура ADD является мнемоникой команды. Директива

```
COST DW ?
```

заставляет ассемблер зарезервировать слово и ассоциировать с ним идентификатор COST, но не порождает машинной команды.

Поскольку ассемблер оказывается просто транслирующей программой, формат и синтаксис команд и директив определяются не компьютером, а тем, как написан ассемблер. В данной книге рассматривается ассемблер ASM-86, разработанный фирмой Intel. Основные его характеристики аналогичны и для других ассемблеров. Мы не будем обсуждать некоторые редко используемые возможности ассемблера ASM-86, а детали программирования следует изучить по руководству языка ассемблера имеющейся у пользователя системы.

Достоинства языков высокого уровня сохранены и в языке ассемблера, несмотря на то, что для реализации одного оператора языка высокого уровня могут потребоваться несколько ассемблерных операторов. Кроме обязательных способов выполнения условных переходов, циклов, ввода-вывода, арифметических операций и присваиваний, в

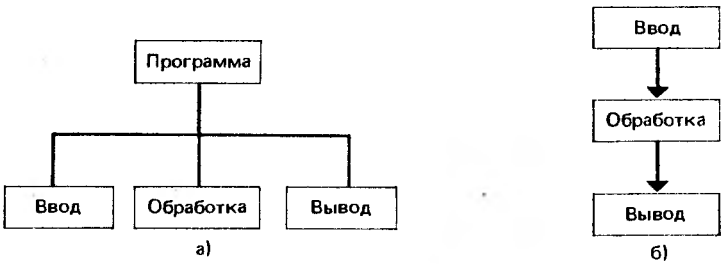


Рис. 3.1. Общие структуры программы:
а – чередование ввода-вывода с обработкой; *б* – последовательные ввод-обработка-вывод

ассемблере требуются средства инициализации, распределения памяти, именования констант, комментирования, структурирования данных, вызовов процедур, операторов-функций и глобальных меток.

Независимо от применяемого языка, программы состоят из секций ввода, обработки и вывода. В больших программах наблюдается сложное переплетение этих функций (рис. 3.1, *а*), а в простых программах они выстраиваются последовательно (рис. 3.1, *б*). Данная глава касается в основном простых программ, в которых ввод, обработка и вывод реализуются последовательно. Поскольку программирование ввода-вывода рассматривается в гл. 6, секции ввода и вывода в программах не приводятся, но их наличие указывается в комментариях. Далее будет показано, что директивы определения и распределения памяти обычно помещаются в начале исходной программы, что также подчеркивается в комментариях. Цель настоящей главы – осветить все вопросы, необходимые для разработки относительно сложных законченных одномодульных программ вида "ввод-обработка-вывод", в которых отсутствуют только секции ввода и вывода.

Вначале анализируется формат ассемблерных программ, а затем команды передач данных и арифметических операций. В § 3.3 арифметические команды служат основой для разработки структуры законченной программы. В § 3.4 и 3.5 описаны команды переходов и циклов, которые позволяют разрабатывать значительно более сложные программы. В § 3.6 – 3.9 обсуждаются логические команды, команды сдвигов и команды управления микропроцессором. Затем рассматриваются директивы и общая структура программы. Процессу ассемблирования программы посвящен § 3.11, а преобразованию ассемблерных команд в машинные – § 3.12.

3.1. ФОРМАТ АССЕМБЛЕРНЫХ КОМАНД

Общий формат ассемблерной команды имеет следующий вид:

Метка: Мнемоника Операнд,Операнд ;Комментарий

Пробелы вводятся произвольно, но минимум один пробел должен быть в тех местах, где его отсутствие ведет к неоднозначности (например, между мнемоникой и первым операндом) Кроме того, пробелы не допускаются в мнемониках и идентификаторах, а в цепочках-константах и в комментариях они должны вводиться специальными символами. *Метка* – это идентификатор, присваиваемый первому байту команды, у которой она появляется. Наличие метки в команде не обязательно, но если она есть, метка становится символическим именем, которое применяется в командах переходов для передачи управления отмеченной команде. При отсутствии метки двоеточия быть не должно. Во всех командах необходимо наличие мнемоники. (Мнемоники допускается модифицировать префиксами, которые вводятся по мере необходимости.) Наличие операндов зависит от команды – некоторые команды не имеют операндов, в других командах требуется один операнд, а в некоторых – два операнда. В случае двух операндов они разделяются запятой. Поле комментария предназначено для пояснения программы и может содержать любую комбинацию символов. Оно не обязательно, и при отсутствии комментария точка с запятой не нужна. Комментарием может быть целая строка и в этом случае первым символом в строке должна быть точка с запятой. Команду допускается переносить на следующие строки, помещая в начале каждой строки-продолжения символ амперсанда.

Ассемблерная команда должна иметь операнд для каждого операнда машинной команды и обозначение каждого операнда должно идентифицировать режим его адресации. При двух операндах первым указывается операнд-получатель, а вторым – операнд-источник. На рис. 3.2 показана типичная ассемблерная команда, которая прибавляет содержимое ячейки памяти к регистру AX. В ней используются регистровый относитель-

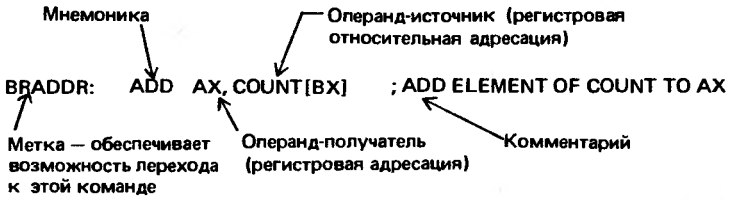


Рис. 3.2. Типичная ассемблерная команда

ный и регистровый режимы адресации. Когда операндом является слово в памяти, младший байт слова имеет меньший (младший) адрес, а старший байт — больший. Если, например, идентификатор COST обозначает операнд-слово, то COST ассоциируется с младшим байтом, а COST + 1 — со старшим байтом.

На рис. 3.3 показан синтаксис операндов различных типов. Пока нам достаточно этих основных форм (возможные модификаторы операндов рассматриваются далее). *Переменной* называется идентификатор, который ассоциируется с первым байтом данного.

РЕЖИМ АДРЕСАЦИИ	ФОРМАТ	ПРИМЕРЫ
ДАННЫХ		
НЕПОСРЕДСТВЕННЫЙ	ВЫРАЖЕНИЕ-КОНСТАНТА	10110B 329 0A9H 'AB' YY-XX+5
ПРЯМОЙ	ПЕРЕМЕННАЯ ± ВЫРАЖЕНИЕ-КОНСТАНТА	WQT CNT-5 ARRAY+5
РЕГИСТРОВЫЙ	РЕГИСТР	AX DH
РЕГИСТРОВЫЙ КОСВЕННЫЙ	[РЕГИСТР]	[BX]
РЕГИСТРОВЫЙ ОТНОСИТЕЛЬНЫЙ	ПЕРЕМЕННАЯ + [РЕГИСТР ± ВЫРАЖЕНИЕ-КОНСТАНТА] ИЛИ [РЕГИСТР ± ВЫРАЖЕНИЕ-КОНСТАНТА]	VAR_X[BX] MESG[SI+10H] [BX-1]
БАЗОВЫЙ ИНДЕКСНЫЙ	[БАЗОВЫЙ РЕГИСТР][ИНДЕКСНЫЙ РЕГИСТР]	[BP][DI]
ОТНОСИТЕЛЬНЫЙ БАЗОВЫЙ ИНДЕКСНЫЙ	ПЕРЕМЕННАЯ[БАЗОВЫЙ РЕГИСТР ± ВЫРАЖЕНИЕ-КОНСТАНТА] [ИНДЕКСНЫЙ РЕГИСТР ± ВЫРАЖЕНИЕ-КОНСТАНТА] ИЛИ [БАЗОВЫЙ РЕГИСТР ± ВЫРАЖЕНИЕ-КОНСТАНТА] [ИНДЕКСНЫЙ РЕГИСТР ± ВЫРАЖЕНИЕ-КОНСТАНТА]	E[BX+5][SI-2] DATA[BX][SI] [BP+2][SI]
ПЕРЕХОДОВ		
ВНУТРИСЕГМЕНТНЫЙ ПРЯМОЙ	МЕТКА ± ВЫРАЖЕНИЕ-КОНСТАНТА	OVFL+7
ВНУТРИСЕГМЕНТНЫЙ КОСВЕННЫЙ	КАК И АДРЕСАЦИЯ ДАННЫХ, КРОМЕ НЕПОСРЕДСТВЕННОЙ	
МЕЖСЕГМЕНТНЫЙ ПРЯМОЙ	МЕТКА ± ВЫРАЖЕНИЕ-КОНСТАНТА	BRANCH_EXIT
МЕЖСЕГМЕНТНЫЙ КОСВЕННЫЙ	КАК И АДРЕСАЦИЯ ДАННЫХ, КРОМЕ НЕПОСРЕДСТВЕННОЙ ИЛИ РЕГИСТРОВОЙ	

ПРИМЕЧАНИЕ. ВО ВСЕХ РЕЖИМАХ, КРОМЕ НЕПОСРЕДСТВЕННОГО, "ВЫРАЖЕНИЕ-КОНСТАНТА" НЕ ОБЯЗАТЕЛЬНО.

Рис. 3.3. Форматы операндов для различных режимов адресации

Выражение представляет собой конкатенацию (сцепление) имен, называемых *термами*, а выражение-константа — это выражение, которое можно вычислить с получением числа. В качестве термов применяются идентификаторы переменных, а также следующие конструкции:

Константа. Число, система счисления которого указывается такими суффиксами: B – двоичное, D – десятичное, O – восьмеричное, H – шестнадцатеричное. По умолчанию принимается десятичная система. Первая цифра в 16-ричном числе должна быть 0 – 9; если старшая цифра оказывается буквой (A – F), то перед ней вводится нуль. Примеры чисел:

$$1011_2 = 1011B \quad 223_{10} = 223D = 223 \quad B25A_{16} = 0B25AH$$

Константа-цепочка. Символьная цепочка, заключенная в апострофы (').

Арифметические операторы. Операторы "+", "–", "*" и "/".

Логические операторы. Операторы "AND", "OR", "NOT" и "XOR" (исключающее ИЛИ). Логические операции выполняются посредством представления операндов в двоичной форме и обработки соответствующих пар разрядов.

Подвыражение. Выражение, являющееся частью другого выражения и ограниченное от порождающего выражения круглыми скобками.

Имя. Идентификатор, представляющий собой константу, константу-цепочку или выражение.

Ассемблер ASM-86 допускает также операторы отношений, но мы их касаться не будем. Старшинство операторов такое же, как в языках высокого уровня. Идентификатор, будь он меткой, переменной или именем, представляет собой цепочку длиной до 31 символа (она может быть и более длинной, но ассемблер ASM-86 распознает только первые 31 символ). Символами могут быть буквы, цифры, вопросительные знаки, значки амперсанда и подчеркивания, но первым символом не должна быть цифра. Идентификаторами не могут быть ключевые слова ассемблера, например мнемоники команд, имена регистров и специальные операторы. Несколько законченных команд ADD и машинные команды, в которых они транслируются, представлены на рис. 3.4.

```
МАШИННЫЙ КОД                    АССЕМБЛЕРНЫЙ КОД

03 C3                ADD    AX,BX          ;(BX) ПРИБАВЛЯЕТСЯ К AX
81 C3 6D 02          ADD    BX,621         ;ПРИБАВИТЬ 621 К BX
01 11                ADD    [BX][DI],DX    ;ПРИБАВИТЬ (DX) К ЯЧЕЙКЕ,
                                           ;СМЕЩЕНИЕ КОТОРОЙ РАВНО СУММЕ
                                           ;(BX) И (DI)
02 B6 2E 02          ADD    AL,NUM[BP+11]  ;ПРИБАВИТЬ К AL СОДЕРЖИМОЕ
                                           ;ЯЧЕЙКИ, СМЕЩЕНИЕ КОТОРОЙ
                                           ;РАВНО СУММЕ СМЕЩЕНИЯ
                                           ;NUM, (BP) И 11

ПРЕДПОЛАГАЕТСЯ, ЧТО СМЕЩЕНИЕ NUM РАВНО 0223, ПОЭТОМУ
      EA = (BP) + 0223 + 000B = (BP) + 022E
```

Рис. 3.4. Типичные ассемблерные команды сложения

Следует подчеркнуть, что сегментный регистр в качестве операнда-получателя допустим только в некоторых командах. Сегментный регистр разрешается явно определять только в командах MOV, PUSH и POP. Первая из этих команд рассматривается в § 3.2, а остальные две – в гл. 4. Однако сегментные регистры могут неявно фигурировать и в других командах, например командах переходов. За исключением цепочечных команд, в двухоперандной команде один из операндов должен быть регистром (кроме команд, в которых операндом-источником служит непосредственное значение). В качестве операнда нельзя определять регистр IP, так как содержимое этого регистра модифицируют только команды переходов и другие команды передачи данных управления.

Имеется несколько модификаторов, называемых *атрибутными операторами*, которые обсуждаются по ходу изложения. Один из таких операторов – оператор указателя (PTR) – необходимо пояснить уже сейчас. Многие команды оперируют байтами или словами в зависимости от состояния бита W. Обычно, как минимум, один из операндов

такой, что операция с байтом или словом подразумевается его обозначением или тем, как определен операнд в памяти. Ассемблер соответственно устанавливает или сбрасывает бит W, например, операнд-получатель в команде ADD AX, [BX] предполагает операцию над словом, а в команде ADD AL, [BX] – операцию с байтом. Если с помощью соответствующих директив переменная COST определена как слово, а переменная COUNT как байт, то команда INC COST оперирует словом, а команда INC COUNT – байтом. Однако в некоторых ситуациях ассемблер не может определить тип операнда. Команда INC [BX] производит инкремент величины, адрес которой находится в регистре BX, но является ли эта величина байтом или словом? Одна из функций оператора PTR заключается в указании длины величины в такой и других неоднозначных ситуациях. Он употребляется в виде записи требуемого типа с последующим оператором PTR. В приведенной команде INC оператор PTR применяется для модификации операнда следующим образом:

```
INC BYTE PTR [BX]
```

в случае инкремента байта или

```
INC WORD PTR [BX]
```

когда производится инкремент слова.

3.2. КОМАНДЫ ПЕРЕДАЧ ДАННЫХ

Во всех компьютерах необходимы команды, предназначенные просто для пересылок данных, адресов и непосредственных операндов в регистры или в ячейки памяти. Рассмотрим подобные команды микропроцессора 8086, но вначале приведем обозначения, употребляемые нами в определениях команд. Используемые имена и аббревиатуры показаны на рис. 3.5 (многие из них уже встречались); с их помощью осуществляются сокращенные определения команд и других дескриптивных операторов. Используя приведенные обозначения, оператор

Содержимое регистра AX заменяется содержимым регистра AX
плюс непосредственный операнд

СОКРАЩЕНИЕ	СМЫСЛОВОЕ ЗНАЧЕНИЕ	ОБОЗНАЧЕНИЕ	СМЫСЛОВОЕ ЗНАЧЕНИЕ
OPR	ОПЕРАНД	←	ЗАМЕНЯЕТСЯ НА
SRC	ОПЕРАНД-ИСТОЧНИК	↔	ОБМЕНИВАЕТСЯ С
DST	ОПЕРАНД-ПОЛУЧАТЕЛЬ	(X)	СОДЕРЖИМОЕ X
REG	РЕГИСТР	∧	ЛОГИЧЕСКОЕ "И"
RSRC	РЕГИСТР-ИСТОЧНИК	∨	ЛОГИЧЕСКОЕ "ИЛИ"
RDST	РЕГИСТР-ПОЛУЧАТЕЛЬ	⊻	ЛОГИЧЕСКОЕ "ИСКЛЮЧАЮЩЕЕ ИЛИ"
CNT	СЧЕТЧИК	$\overline{X}$	ИНВЕРСИЯ X
DISP	СМЕЩЕНИЕ (ОБЩЕЕ ОБОЗНАЧЕНИЕ)		
D8	8-БИТНОЕ СМЕЩЕНИЕ		
D16	16-БИТНОЕ СМЕЩЕНИЕ		
ADDR	АДРЕС		
EA	ЭФФЕКТИВНЫЙ АДРЕС		
SEG	СЕГМЕНТ		
DATA	НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД (ОБЩЕЕ ОБОЗНАЧЕНИЕ)		

```
DATA8          8-БИТНЫЙ НЕПОСРЕДСТВЕННЫЙ
                  ОПЕРАНД
DATA16         16-БИТНЫЙ НЕПОСРЕДСТВЕННЫЙ
                  ОПЕРАНД
AX,BX,CX,      РЕГИСТРЫ МИКРОПРОЦЕССОРА 8086
DX,AL,AH,
BL,BH,CL,
CH,DL,DH,
IP,BP,SP,
SI,DI

CS,SS,DS,      СЕГМЕНТНЫЕ РЕГИСТРЫ
ES             МИКРОПРОЦЕССОРА 8086

DF,IF,TF       УПРАВЛЯЮЩИЕ ФЛАЖКИ
               МИКРОПРОЦЕССОРА 8086

OF,SF,ZF,      ФЛАЖКИ УСЛОВИЙ
AF,PF,CF       МИКРОПРОЦЕССОРА 8086
```

Рис. 3.5. Список сокращений и условных обозначений

можно записать в виде

$$(AX) \leftarrow (AX) + DATA.$$

Аналогичным образом оператор

Содержимое BX заменяется содержимым ячейки памяти, эффективный адрес которой равен содержимому BP плюс смещение

допускает следующую запись:

$$(BX) \leftarrow ((BP) + DISP).$$

В микропроцессоре 8086 имеются 4 базовые команды MOV, LEA, LDS и LES (рис. 3.6) для передач значений в(из) регистры(ов) и память(и). Команда MOV предназначена для

НАЗВАНИЕ	МНЕМОНИКА	И ФОРМАТ	ОПИСАНИЕ
ПЕРЕДАТЬ	MOV	DST,SRC	(DST) ⟵ (SRC)
ЗАГРУЗИТЬ ЭФФЕКТИВНЫЙ АДРЕС	LEA	REG,SRC	(REG) ⟵ SRC
ЗАГРУЗИТЬ В DS УКАЗАТЕЛЬ	LDS	REG,SRC	(REG) ⟵ (SRC) (DS) ⟵ (SRC+2)
ЗАГРУЗИТЬ В ES УКАЗАТЕЛЬ	LES	REG,SRC	(REG) ⟵ (SRC) (ES) ⟵ (SRC+2)
ОБМЕНЯТЬ	XCHG	OPR1,OPR2	(OPR1) ⟷ (OPR2)

ФЛАЖКИ. НИ ОДИН ИЗ ФЛАЖКОВ НЕ ИЗМЕНЯЕТСЯ.

РЕЖИМЫ АДРЕСАЦИИ. ПОЛУЧАТЕЛЬ НЕ МОЖЕТ БЫТЬ НЕПОСРЕДСТВЕННЫМ И НЕ МОЖЕТ БЫТЬ CS. В КОМАНДАХ LEA, LDS, LES ОПЕРАНД REG НЕ МОЖЕТ БЫТЬ СЕГМЕНТНЫМ РЕГИСТРОМ, А ИСТОЧНИК НЕ МОЖЕТ ИМЕТЬ НЕПОСРЕДСТВЕННЫЙ ИЛИ РЕГИСТРОВЫЙ РЕЖИМ. В КОМАНДЕ MOV ОДИН ИЗ ОПЕРАНДОВ ДОЛЖЕН БЫТЬ РЕГИСТРОМ (ЗА ИСКЛЮЧЕНИЕМ НЕПОСРЕДСТВЕННОГО ОПЕРАНДА-ИСТОЧНИКА). В КОМАНДЕ XCHG ХОТЯ БЫ ОДИН ИЗ ОПЕРАНДОВ ДОЛЖЕН БЫТЬ РЕГИСТРОМ, НО НИ ОДИН ИЗ ОПЕРАНДОВ НЕ МОЖЕТ БЫТЬ СЕГМЕНТНЫМ РЕГИСТРОМ.

Рис. 3.6. Команды передач данных

пересылок байта или слова внутри ЦП или между ЦП и памятью. В зависимости от режима адресации она может передавать информацию:

из регистра в регистр,

из непосредственного операнда в регистр,

из непосредственного операнда в ячейку памяти,

из ячейки памяти в регистр,
из регистра в ячейку памяти,
из регистра (ячейки памяти) в сегментный регистр (кроме CS),
из сегментного регистра в регистр/ячейку памяти.

При выполнении команд передачи не изменяется ни один из флажков. Несколько примеров команды MOV приведены на рис. 3.7.

На рис. 3.8, *а* показана программа для обмена содержимого двух байт в ячейках памяти, представленных OPR1 и OPR2, а на рис. 3.8, *б* – выполняемые этой программой

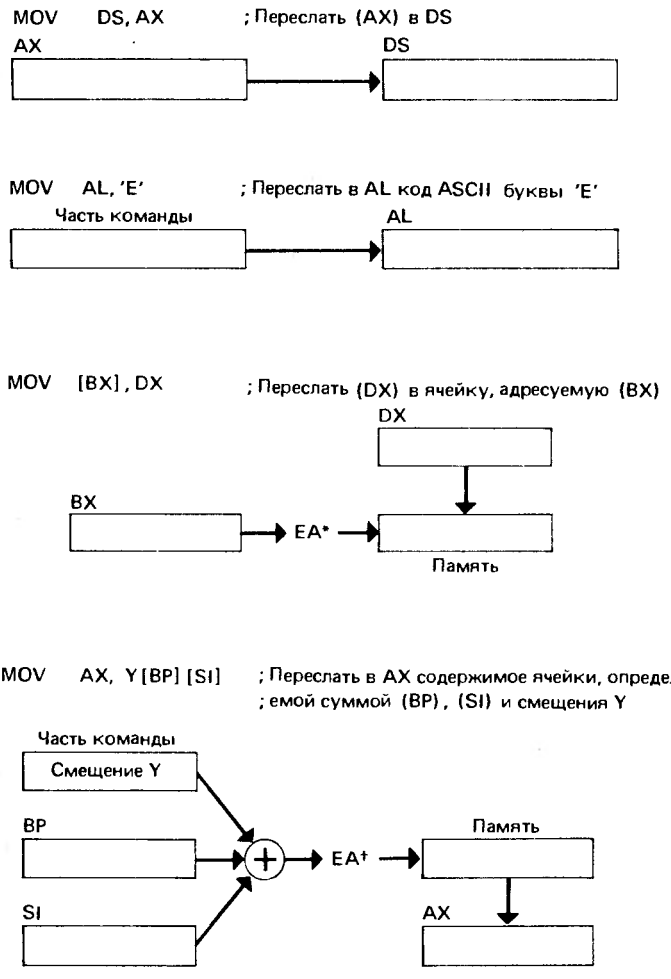


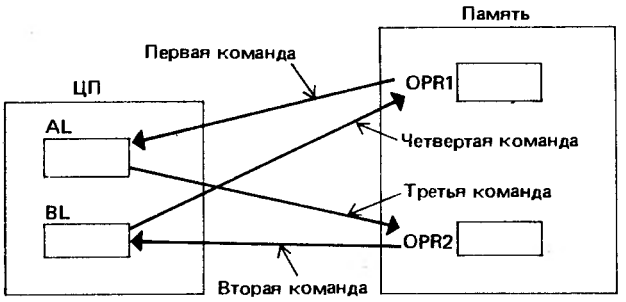
Рис. 3.7. Примеры команды MOV

```
Директивы определения данных, команды ввода и др.
OPR1 и OPR2 определены как переменные-байты

        MOV     AL,OPR1         ; (OPR1) в AL
        MOV     BL,OPR2         ; (OPR2) в BL
        MOV     OPR2,AL         ; (AL) в OPR2
        MOV     OPR1,BL         ; (BL) в OPR1

Вывод и другие команды
```

а)



б)

Рис. 3.8. Программный фрагмент для обмена содержимого двух ячеек памяти: *а* – код; *б* – предпринимаемые действия

действия. Рис. 3.9 иллюстрирует, каким образом транслируется и запоминается в памяти машинный код данного фрагмента. Приведенные адреса команд предполагают, что первая команда начинается в 0230 и (CS) = 0C30, а смещения OPR1 и OPR2 считаются равными 1200 и 1C20.

Типичная команда LEA имеет вид

```
LEA SI,COL [BX]
```

Она загружает в регистр SI эффективный адрес, вычисленный сложением содержимого BX и смещения COL. Команда LEA не влияет на флажки. Ее операнд-источник должен давать эффективный адрес и именно этот адрес, а не содержимое по этому адресу, загружается в регистр-получатель.

Команда XCHG аналогична команде MOV, но вместо дублирования источника в получатель она производит обмен двух величин. Так как непосредственные операнды обменять невозможно, ни один из операндов команды XCHG не может быть константой. В примере на рис. 3.8 команда XCHG может заменить две команды MOV (рис. 3.10).

Отметим, что при использовании команды XCHG потребовались всего три команды и один регистр. Следующие команды осуществляют передачу адреса ARRAY в регистр BX, загрузку нуля в регистр SI и пересылку слова, начинающегося в ARRAY, в регистр AX:

```
LEA   BX,ARRAY
MOV   SI,0
MOV   AX,[BX][SI]
```

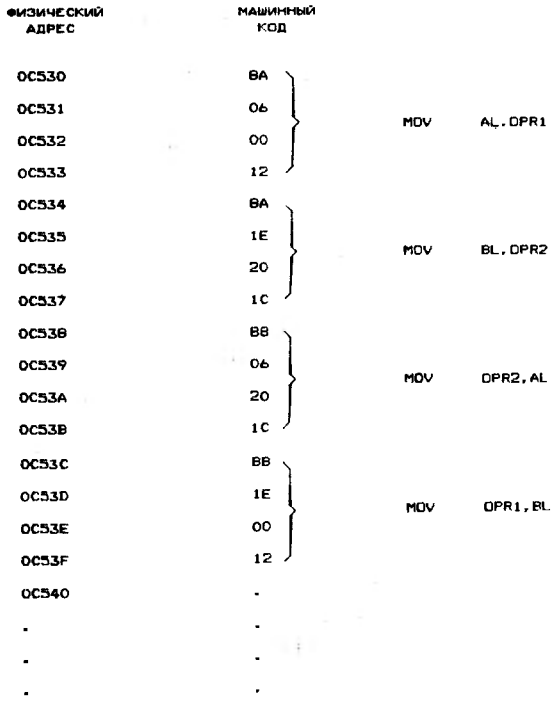


Рис. 3.9. Машинный код программного фрагмента, приведенного на рис. 3.8, *а*

```
ДИРЕКТИВЫ ОПРЕДЕЛЕНИЯ ДАННЫХ, КОМАНДЫ ВВОДА И ДР.
OPR1 И OPR2 ОПРЕДЕЛЕНЫ КАК ПЕРЕМЕННЫЕ-БАЙТЫ.

        MOV     AL,OPR1         ;ЗАГРУЗИТЬ (OPR1) В AL
        XCHG    AL,OPR2         ;ОБМЕНЯТЬ (AL) С (OPR2)
        MOV     OPR1,AL         ;ПЕРЕДАТЬ (AL) В OPR1

КОМАНДЫ ВЫВОДА И ДР.
```

Рис. 3.10. Обмен байт в памяти с помощью команды XCHG

Команды LDS и LES аналогичны, но первая загружает из памяти регистр DS, а вторая – регистр ES. Обе команды также загружают из памяти второй несегментный регистр и не модифицируют флажки. Типичный вид команд LDS и LES:

```
LDS  SI,STRING_SOURCE_POINTER
LES  DI,TABLE[BX]
```

Здесь STRING_SOURCE_POINTER и TABLE являются переменными типа двойного слова. Данные команды упрощают коммутацию сегментов данных посредством одновременной загрузки базового или индексного регистра и сегментного регистра. Удобство наличия таких команд будет очевидно в дальнейшем.

3.3. АРИФМЕТИЧЕСКИЕ КОМАНДЫ

Система команд микропроцессора 8086 включает в себя команды для выполнения арифметических операций над двоичными, упакованными и неупакованными двоично-кодированными десятичными числами (или BCD-числами). Разновидности реализуемых командами арифметических операций отражены на рис. 3.11, из которого видно, что с

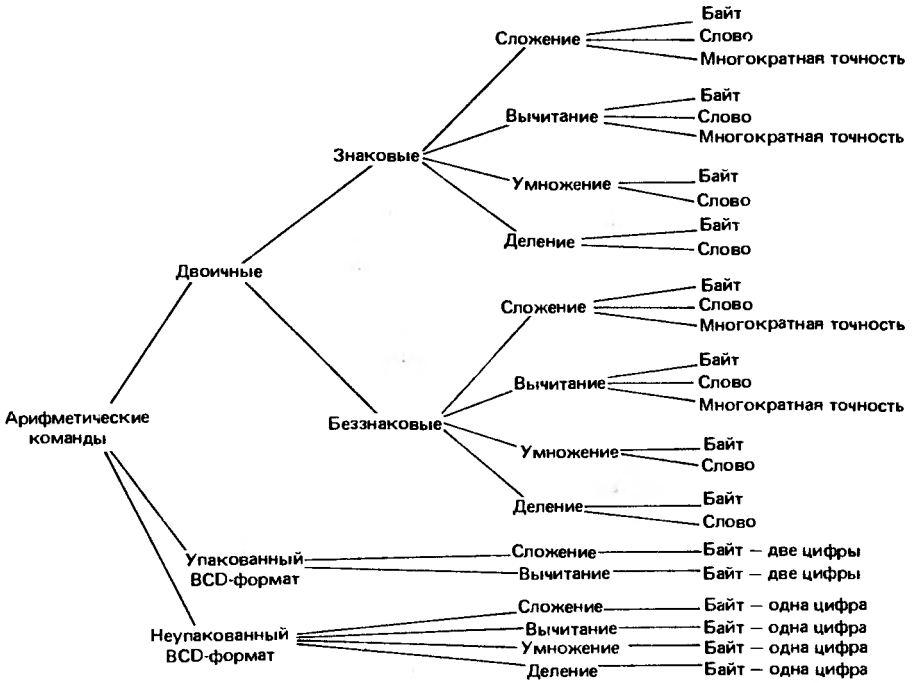


Рис. 3.11. Арифметические операции, непосредственно реализуемые командами микропроцессора 8086

Рис. 3.12. Последовательность преобразований, необходимых для вычислений в двоичном формате

двоичными и неупакованными BCD-числами можно выполнять все 4 арифметические операции, а с упакованными – только сложение и вычитание.

Как упоминалось в § 1.2, двоичные операции реализуются быстрее BCD-операций; кроме того, в двоичном формате в заданном числе разрядов обеспечивается хранение чисел из большего диапазона. В 16 разрядах можно хранить любое из 64К двоичных чисел, но только одно из 10000 упакованных BCD-чисел и одно из 100 неупакованных BCD-чисел. С другой стороны, неупакованные BCD-числа можно непосредственно вводить и выводить – не требуются дополнительные преобразования. На рис. 3.12 показан полный процесс преобразования входных BCD-чисел в двоичные, производства вычислений в двоичной форме, преобразования результатов в код ASCII и последующего вывода результатов. Более подробно этот процесс рассматривается в § 5.5.

3.3.1. ДВОИЧНАЯ АРИФМЕТИКА

Команды двоичных сложения и вычитания определены на рис. 3.13. Все команды воздействуют на арифметические флажки. В них допускается использовать любой режим адресации для одного из операндов, но (за исключением непосредственного операнда-источника) один из двух операндов должен быть регистром. Приведенные команды,

НАЗВАНИЕ	МНЕМОНИКА И ФОРМАТ		ОПИСАНИЕ
СЛОЖИТЬ	ADD	DST,SRC	(DST) ← (SRC)+(DST)
СЛОЖИТЬ С ПЕРЕНОСОМ	ADC	DST,SRC	(DST) ← (SRC)+(DST)+(CF)
ВЫЧЕСТЬ	SUB	DST,SRC	(DST) ← (DST)-(SRC)
ВЫЧЕСТЬ С ЗАЕМОМ	SBB	DST,SRC	(DST) ← (DST)-(SRC)-(CF)

ФЛАЖКИ. МОДИФИЦИРУЮТСЯ ВСЕ ФЛАЖКИ УСЛОВИЙ.

РЕЖИМЫ АДРЕСАЦИИ. ОДИН ИЗ ОПЕРАНДОВ ДОЛЖЕН НАХОДИТЬСЯ В РЕГИСТРЕ (ЗА ИСКЛЮЧЕНИЕМ НЕПОСРЕДСТВЕННОГО ОПЕРАНДА-ИСТОЧНИКА). ДРУГОЙ ОПЕРАНД МОЖЕТ ИМЕТЬ ЛЮБОЙ РЕЖИМ АДРЕСАЦИИ.

Рис. 3.13. Команды двоичных сложения и вычитания

как и другие команды двоичной арифметики, оперируют байтами и словами. Программный фрагмент, иллюстрирующий команды ADD и SUB, приведен на рис. 3.14. В этом фрагменте суммируется содержимое ячеек X и Y, к сумме прибавляется 24, затем вычи-

```
ДИРЕКТИВЫ ОПРЕДЕЛЕНИЯ ДАННЫХ, КОМАНДЫ ВВОДА И ДР.
X, Y, W И Z ОПРЕДЕЛЕНЫ КАК ПЕРЕМЕННЫЕ-СЛОВА.

        MOV     AX,X            ;ПЕРЕДАТЬ  (X) В AX
        ADD     AX,Y            ;ПРИБАВИТЬ (Y) К AX
        ADD     AX,24           ;ПРИБАВИТЬ 24 К СУММЕ
        SUB     AX,Z            ;ВЫЧЕСТЬ (Z) ИЗ (X)+(Y)+24
        MOV     W,AX            ;ЗАПОМНИТЬ РЕЗУЛЬТАТ В W

КОМАНДЫ ВЫВОДА И ДР.
```

Рис. 3.14. Пример операций с одинарной точностью

тается содержимое ячейки Z и результат запоминается в W; другими словами, этот фрагмент реализует оператор

$$W \leftarrow X + Y + 24 - Z.$$

Поскольку 16-битное слово может хранить всего одно из 64К чисел, арифметические операции над одиночными словами оказываются довольно ограниченными. Для расширения диапазона чисел необходимо сцеплять слова, образуя группы, кратные 16 разря-

дам. Арифметика, оперирующая одиночными операндами-словами, называется *арифметикой одинарной точности*, оперирующая двумя словами – *арифметикой двойной точности* и т. д. В общем, все, большее одинарной точности, называется *многократной точностью*. Арифметики двойной точности, допускающей $2^{32} \approx$ 4 млрд различных чисел, обычно достаточно при работе с целыми числами; операнды, состоящие из двух слов, называются операндами *двойной длины*. (В принципе, данное положение применимо и к байтам, однако операции многократной точности над байтами применяются редко.)

Команды двоичного сложения и вычитания одновременно оперируют словами, но, переходя от команд ADD и SUB к командам ADC и SBB, можно автоматически учесть переносы и заемы. Как показано далее, с помощью команд ADC и SBB относительно легко реализуется арифметика многократной точности.

Если 16 бит в слове считаются числом в дополнительном коде, число называется *знаковым*, а если все 16 бит рассматриваются как неотрицательное число, оно называется *беззнаковым*. Например, 16-битное число C2A3 как знаковая величина равно -15709_{10}, а как беззнаковая величина равно 49827_{10}. Очевидно, любое число с нулевым старшим битом оказывается одним и тем же при рассмотрении его как знакового или беззнакового. Логические схемы в АЛУ выполняют все сложения так, как если бы они были беззнаковыми. Знаковое сложение в дополнительном коде осуществляется просто путем игнорирования переноса из 16-го бита. Переполнение в беззнаковом сложении возникает, если сумму нельзя представить в 16 битах, а переполнение в знаковом сложении возникает, если абсолютное значение суммы слишком велико для 15 бит (за исключением числа -2^{15}). Однако, как отмечалось в гл. 2, ЦП устанавливает или сбрасывает флажок OF так, как будто выполняется знаковое сложение. Аналогичные положения относятся и к вычитанию, если учесть, что флажок переноса устанавливается при возникновении заема.

Чтобы показать, каким образом два 16-битных сложения могут дать 32-битную сумму, рассмотрим следующий пример:

Перенос = 0		Перенос = 1	
0015	25A0	0015	65A0
+ 0021	B79E	+ 0021	B79E
0036	DD3E	0037	1D3E

В обоих случаях получен правильный ответ при сложении младших 16 бит и последующем сложении старших 16 бит с учетом переноса из младших 16 бит. Хотя это и не очевидно, такая процедура оказывается правильной и при допущении отрицательных чисел с представлением их в дополнительном коде (см. упр. 10). На рис. 3.15 приведен фрагмент вычисления

DPSUM ← DP1 + DP2

с двойной точностью. В нем предполагается, что младшие слова операндов и суммы находятся соответственно в DP1, DP2 и DPSUM, а старшие слова – в DP1 + 2, DP2 + 2 и DPSUM + 2. Отметим, что младшие слова складываются командой ADD, а старшие –

```
ДИРЕКТИВЫ ОПРЕДЕЛЕНИЯ ДАННЫХ, КОМАНДЫ ВВОДА И ДР.
DP1, DP2 И DPSUM ОПРЕДЕЛЕНЫ КАК ПЕРЕМЕННЫЕ-СЛОВА.

        MOV     AX,DP1          ;СЛОЖИТЬ МЛАДШИЕ СЛОВА
        ADD     AX,DP2          ;ИЗ DP1 И DP2 И ПОМЕСТИТЬ
        MOV     DPSUM,AX        ;СУММУ В DPSUM
        MOV     AX,DP1+2        ;СЛОЖИТЬ СТАРШИЕ СЛОВА ИЗ DP1+2
        ADC     AX,DP2+2        ;И DP2+2 С УЧЕТОМ ПЕРЕНОСА
        MOV     DPSUM+2.AX      ;И ПОМЕСТИТЬ СУММУ В DPSUM+2

КОМАНДЫ ВЫВОДА И ДР.
```

Рис. 3.15. Сложение с двойной точностью

командой ADC, поэтому в их сумме учитывается и перенос. Аналогичным образом при операции с большей точностью операнды следует хранить в последовательных словах, причем младшее слово имеет наименьший адрес, следующее – адрес больше на 2 и т. д. Кроме пары младших слов, остальные слова суммируются командой ADC.

Для иллюстрации вычитания рассмотрим следующие 32-битные операции:

	0006	A92F		0006	A92F
−	0004	9837	−	0004	B837
	0002	10F8		0002	F0F8
Заем	−0		Заем	−1	
	0002	10F8		0001	F0F8

По-прежнему правильные результаты получаются при обработке операндов по 16 бит, если заем из старших 16 бит вычитается из старшей разности. Напомним, что флажок CF устанавливается в 1, если беззнаковое вычитаемое больше беззнакового уменьшаемого (т. е. возникает заем). Это правило установки флажка CF относится и к вычитанию операндов, представленных в дополнительном коде. На рис. 3.16 показано вычисление с двойной точностью выражения

$$W \leftarrow X + Y + 24 - Z.$$

```
ДИРЕКТИВЫ ОПРЕДЕЛЕНИЯ ДАННЫХ, КОМАНДЫ ВВОДА И ДР.
X, Y, W И Z ОПРЕДЕЛЕНЫ КАК ПЕРЕМЕННЫЕ-СЛОВА.

        MOV     AX,X            ;СЛОЖИТЬ ЧИСЛА С ДВОЙНОЙ
        MOV     DX,X+2          ;ТОЧНОСТЬЮ X,X+2 И Y,Y+2
        ADD     AX,Y
        ADC     DX,Y+2
        ADD     AX,24           ;ПРИБАВИТЬ К РЕЗУЛЬТАТУ ЧИСЛО 24
        ADC     DX,0            ;С ДВОЙНОЙ ТОЧНОСТЬЮ
        SUB     AX,Z            ;ВЫЧЕСТЬ ИЗ СУММЫ ЧИСЛО Z,Z+2
        SBB     DX,Z+2          ;С ДВОЙНОЙ ТОЧНОСТЬЮ
        MOV     W,AX            ;ЗАПОМНИТЬ ВЕСЬ РЕЗУЛЬТАТ
        MOV     W+2,DX          ; В W,W+2

КОМАНДЫ ВЫВОДА И ДР.
```

Рис. 3.16. Вычисление выражения с операндами, имеющими двойную точность

Знаковую смешанную арифметику с операндами различной точности можно выполнять, расширяя знак более короткого числа до выравнивания длин чисел. Для этого предназначены команды CBW и CWD, определение которых приведено на рис. 3.17.

НАЗВАНИЕ	МНЕМОНИКА И ФОРМАТ	ОПИСАНИЕ
ПРЕОБРАЗОВАТЬ БАЙТ В СЛОВО	CBW	РАСШИРЯЕТ ЗНАК AL В AH
ПРЕОБРАЗОВАТЬ СЛОВО В ДВОЙНОЕ СЛОВО	CWD	РАСШИРЯЕТ ЗНАК AX В DX

ФЛАЖКИ. НИ ОДИН ИЗ ФЛАЖКОВ НЕ ИЗМЕНЯЕТСЯ.

РЕЖИМЫ АДРЕСАЦИИ. ОПЕРАНД ДОЛЖЕН НАХОДИТЬСЯ В AL ИЛИ AX.

Рис. 3.17. Команды расширения знака

Команда CBW расширяет знак байта из регистра AL, образуя в регистре AX эквивалентное слово в дополнительном коде. Если, например, (AL) = B4, то после выполнения команды CBW в регистре AX получается FFB4. Аналогично команда CWD расширяет знак слова из AX в регистр DX, формируя двойное слово в DX:AX. В обеих командах

операнд неявно указывается кодом операции и флажки не модифицируются. Далее приведен фрагмент для прибавления одинарного числа из BX к двойному числу в ячейках DP и DP + 2 с загрузкой результата в DP и DP + 2:

```
MOV     AX,BX
CWD                         ;РАСШИРЕНИЕ ЗНАКА В DX
ADD     AX,DP
MOV     DP,AX
ADC     DX,DP+2             ;УЧИТЫВАЕТСЯ ЗНАК В BX
MOV     DP+2,DX
```

После выполнения этих команд старое содержимое DP и DP + 2 теряется.

На рис. 3.18 определены еще четыре команды двоичной арифметики. Эти команды воздействуют на все арифметические флажки, но команды INC и DEC не изменяют состояние флажка CF. Команды INC, DEC и NEG имеют один операнд: команда INC производит инкремент операнда на 1, команда DEC – декремент операнда на 1, а команда NEG изменяет знак операнда. В них допускается любой режим адресации за исключе-

НАЗВАНИЕ	МНЕМОНИКА И ФОРМАТ		ОПИСАНИЕ
ИНКРЕМЕНТ	INC	OPR	(OPR) ← (OPR)+1
ДЕКРЕМЕНТ	DEC	OPR	(OPR) ← (OPR)-1
ИЗМЕНИТЬ ЗНАК	NEG	OPR	(OPR) ← -(OPR)
СРАВНИТЬ	CMP	OPR1,OPR2	(OPR1) - (OPR2)

ФЛАЖКИ. ВСЕ ФЛАЖКИ УСЛОВИЙ МОДИФИЦИРУЮТСЯ, НО КОМАНДЫ INC И DEC НЕ ВОЗДЕЙСТВУЮТ НА ФЛАЖОК CF.

РЕЖИМЫ АДРЕСАЦИИ. В КОМАНДАХ INC, DEC И NEG НЕ ДОПУСКАЕТСЯ НЕПОСРЕДСТВЕННЫЙ РЕЖИМ. ОДИН ИЗ ОПЕРАНДОВ В КОМАНДЕ CMP ДОЛЖЕН БЫТЬ РЕГИСТРОМ (ЗА ИСКЛЮЧЕНИЕМ НЕПОСРЕДСТВЕННОГО ОПЕРАНДА OPR2); ДРУГОЙ ОПЕРАНД МОЖЕТ ИМЕТЬ ЛЮБОЙ РЕЖИМ АДРЕСАЦИИ, НО OPR1 НЕ МОЖЕТ БЫТЬ НЕПОСРЕДСТВЕННЫМ ЗНАЧЕНИЕМ.

Рис. 3.18. Однооперандные команды двоичной арифметики и команда сравнения

нием непосредственного. Команды INC и DEC предназначены главным образом для счета и индексирования и широко применяются в циклах. Команда сравнения CMP идентична команде SUB, но результат нигде не запоминается. Она используется только для установки флажков на основе отношения между операндами и обычно помещается перед командой условного перехода. Выполнение перехода зависит от состояний флажков, т. е. от значений сравниваемых операндов. Функции команды CMP рассмотрены вместе с переходами.

На рис. 3.19 определены команды двоичных умножения и деления. Команды умножения осуществляют умножение байт с получением результата-слова и умножение слов, давая в результате двойное слово. Если в знаковом умножении старший байт (слово) результата не является расширением знака младшего байта (слова), флажки CF и OF устанавливаются в 1, а в противном случае они содержат 0. В беззнаковом умножении флажки CF и OF устанавливаются в 1, если в произведении получен ненулевой старший байт (слово), иначе оба они будут нулевыми. Проверяя состояния этих флажков, можно определить, размещается ли произведение в одном байте (слове). Остальные флажки могут модифицироваться, но их состояния не определены, т. е. не предсказуемы. В команде IMUL знакового умножения знак произведения определяется по обычным алгебраическим правилам.

Команда MUL беззнакового умножения интерпретирует операнды как беззнаковые числа и дает беззнаковое произведение. Чтобы показать различия между командами

НАЗВАНИЕ	МНЕМОНИКА И ФОРМАТ		ОПИСАНИЕ
УМНОЖИТЬ СО ЗНАКОМ	IMUL	SRC	ОПЕРАНДЫ-БАЙТЫ (AX) ← (AL)*(SRC) ОПЕРАНДЫ-СЛОВА (DX:AX) ← (AX)*(SRC) ПРОИЗВЕДЕНИЕ СО ЗНАКОМ
УМНОЖИТЬ БЕЗ ЗНАКА	MUL	SRC	АНАЛОГИЧНА КОМАНДЕ IMUL, НО ОПЕРАНДЫ И ПРОИЗВЕДЕНИЕ БЕЗЗНАКОВЫЕ
РАЗДЕЛИТЬ СО ЗНАКОМ	IDIV	SRC	ДЕЛИТЕЛЬ-БАЙТ (AL) ← ЧАСТНОЕ(AX)/(SRC) (AH) ← ОСТАТОК(AX)/(SRC) ДЕЛИТЕЛЬ-СЛОВО (AX) ← ЧАСТНОЕ(DX:AX)/(SRC) (DX) ← ОСТАТОК(DX:AX)/(SRC) ЧАСТНОЕ И ОСТАТОК ИМЕЮТ ЗНАКИ, ЗНАК ОСТАТКА РАВЕН ЗНАКУ ДЕЛИМОГО
РАЗДЕЛИТЬ БЕЗ ЗНАКА	DIV	SRC	АНАЛОГИЧНА КОМАНДЕ IDIV, НО ОПЕРАНДЫ, ЧАСТНОЕ И ОСТАТОК БЕЗЗНАКОВЫЕ

ФЛАЖКИ. КОМАНДЫ IMUL И MUL УСТАНАВЛИВАЮТ OF И CF В СОСТОЯНИЕ 1, ЕСЛИ ДЛЯ РЕЗУЛЬТАТА ТРЕБУЮТСЯ ДВА БАЙТА (СЛОВА); В ПРОТИВНОМ СЛУЧАЕ ЭТИ ФЛАЖКИ НУЛЕВЫЕ. ОСТАЛЬНЫЕ ФЛАЖКИ УСЛОВИЙ НЕ ОПРЕДЕЛЕНЫ. В КОМАНДАХ IDIV И DIV ВСЕ ФЛАЖКИ УСЛОВИЙ НЕ ОПРЕДЕЛЕНЫ.

РЕЖИМЫ АДРЕСАЦИИ. ОПЕРАНДЫ-ИСТОЧНИКИ В ЭТИХ КОМАНДАХ НЕ МОГУТ БЫТЬ НЕПОСРЕДСТВЕННЫМИ ЗНАЧЕНИЯМИ, А ВСЕ ДРУГИЕ РЕЖИМЫ АДРЕСАЦИИ ДОПУСТИМЫ. ПОЛУЧАТЕЛЕМ ДОЛЖНЫ БЫТЬ AX ИЛИ DX:AX.

Рис. 3.19. Команды двоичных умножения и деления

IMUL и MUL, предположим, что (AL) = B4 (т. е. -76_{10} как 8-битное знаковое целое число и 180_{10} как беззнаковое целое число) и (BL) = 11 (или 17_{10} как знаковое и беззнаковое целое число). Команда IMUL BL формирует (AX) = FAF4 = -1292_{10} и (CF) = = (OF) = 1 (результат не помещается в 8 бит). Команда MUL BL дает (AX) = 0BF4 = = 3060_{10} и (CF) = (OF) = 1 (т. е. результат по-прежнему слишком велик для 8 бит). Отметим, что в обеих командах IMUL и MUL переполнение невозможно, так как для произведения отводится достаточное место.

Команда MUL беззнакового умножения предназначена в основном для операций умножения с многократной точностью. Для иллюстрации беззнакового умножения двойной точности рассмотрим два неотрицательных числа с двойной точностью $a2^{16} + b$ и $c2^{16} + d$, где a, b, c и d – коэффициенты, соответствующие основанию 2^{16}. Умножение с основанием 2^{16} реализуется следующим образом:

$$\begin{array}{r} a2^{16} + b \\ \times\ c2^{16} + d \\ \hline ad2^{16} + bd \\ ac2^{32} + bc2^{16} \\ \hline ac2^{32} + (ad + bc)\,2^{16} + bd \end{array}$$

Так как $ad2^{16}$ и $bc2^{16}$ эквивалентны ad и bc с последующими 16 нулями, а $ac2^{32}$ равно ac с 32 нулевыми битами, произведение можно найти следующей процедурой:

1. Вычислить bd и запомнить младшее слово в качестве младшего слова произведения.

2. Вычислить ad, прибавить старшее слово bd к младшему слову ad и прибавить перенос к старшему слову ad.

3. Вычислить bc и прибавить его к результату шага 2, пользуясь сложением с двойной точностью. Запомнить перенос для учета его на шаге 5.

4. Запомнить младшее слово результата шага 3 как следующее младшее слово произведения.

5. Вычислить *ac*, прибавить старшее слово результата шага 3 к младшему слову *ac* и прибавить к старшему слову *ac* переносы, включая и перенос шага 3.

6. Запомнить двойное слово результата шага 5 как два старших слова произведения.

На рис. 3.20 приведен фрагмент вычисления с двойной точностью

DPZ ← DPX * DPY

```
ДИРЕКТИВЫ ОПРЕДЕЛЕНИЯ ДАННЫХ, КОМАНДЫ ВВОДА И ДР.
DPX, DPY И DPZ ОПРЕДЕЛЕНЫ КАК ПЕРЕМЕННЫЕ-СЛОВА.

        MOV     AX,DPX          ;УМНОЖИТЬ НА (DPY)
        MUL     DPY             ;ЗНАЧЕНИЕ DPX И ЗАПОМНИТЬ
        MOV     DPZ,AX          ;МЛАДШИЙ РЕЗУЛЬТАТ В DPZ,
        MOV     CX,DX           ;А СТАРШИЙ РЕЗУЛЬТАТ В CX
        MOV     AX,DPX+2        ;УМНОЖИТЬ (DPX+2)
        MUL     DPY             ;НА (DPY) И ЗАПОМНИТЬ
        MOV     BX,DX           ; СТАРШИЙ РЕЗУЛЬТАТ В BX
        ADD     CX,DX           ;ПРИБАВИТЬ МЛАДШИЙ РЕЗУЛЬТАТ К CX
        ADC     BX,0            ;ПРИБАВИТЬ ПЕРЕНОС К BX
        MOV     AX,DPX          ;УМНОЖИТЬ (DPX)
        MUL     DPY+2           ;НА (DPY+2)
        ADD     CX,AX           ;СЛОЖИТЬ С ДВОЙНОЙ ТОЧНОСТЬЮ
        ADC     BX,DX           ;(DX:AX) И (BX:CX)
        MOV     DPZ+2,CX        ;ЗАПОМНИТЬ МЛАДШИЙ РЕЗУЛЬТАТ В DPZ+2
        MOV     CX,0            ;И ПЕРЕНОС В CX
        ADC     CX,0            ;
        MOV     AX,DPX+2        ;УМНОЖИТЬ (DPX+2)
        MUL     DPY+2           ;НА (DPY+2)
        ADD     AX,BX           ;ПРИБАВИТЬ МЛАДШУЮ ЧАСТЬ ЭТОГО
                                ;РЕЗУЛЬТАТА К СТАРШЕЙ ЧАСТИ
                                ;ПРЕДВАРИТЕЛЬНОГО РЕЗУЛЬТАТА
        ADC     DX,CX           ;СЛОЖИТЬ СТАРШУЮ ЧАСТЬ
                                ;ЭТОГО РЕЗУЛЬТАТА
                                ;С ПРОМЕЖУТОЧНЫМИ ПЕРЕНОСАМИ
        MOV     DPZ+4,AX        ;ЗАПОМНИТЬ В DPZ+4 И DPZ+6
        MOV     DPZ+6.DX        ;

КОМАНДЫ ВЫВОДА И ДР.
```

Рис. 3.20. Умножение с двойной точностью

в предположении, что DPX и DPY – неотрицательные числа. Умножение, допускающее отрицательные числа, выполняется в два этапа: вначале по приведенному выше алгоритму умножаются абсолютные значения сомножителей, а затем изменяется знак результата, если сомножители имеют разные знаки (см. упр. 11).

Команды деления IDIV и DIV дают неопределенные состояния флажков и, как команды умножения, не допускают непосредственных операндов. При делении необходимо получать частное и остаток. Поскольку делимое вдвое длиннее частного, может возникнуть переполнение, однако флажок OF в этом случае не показывает переполнения, а частное и остаток не определены. (Переполнение вызывает прерывание – см. гл. 4.) В команде IDIV знакового деления знак частного определяется по алгебраическому правилу, а остатку присваивается знак делимого. (Следовательно, независимо от знаков операндов произведение делителя и частного плюс остаток всегда равно делимому.) Для сравнения действий команд IDIV и DIV предположим, что (AX) = 0400 и (BL) = B4 (т. е. -76_{10} со знаком и 180_{10} без знака). Тогда команда IDIV BL формирует остаток (AH) = 24 = 36_{10} и частное (AL) = F3 = -13_{10}, а команда DIV BL – остаток (AH) = 7C = 124_{10} и частное (AL) = 05 = 5_{10}.

Деление с многократной точностью зависит от длин делителя и делимого. Для иллюстрации алгоритма беззнакового деления трех слов на одно предположим, что делитель равен a, а делимое $b2^{32} + c2^{16} + d$, где a, b, c и d – коэффициенты по основанию 2^{16}. Ниже показан обычный алгоритм деления по основанию 2^{16}:

$$
\begin{array}{r}
q_2 2^{32} + q_1 2^{16} + q_0 \\
\hline
a)\ b2^{32} + c2^{16} + d \\
\hline
r_2 2^{32} + c2^{16} \\
\hline
r_1 2^{16} + d \\
\hline
r_0
\end{array}
$$

Здесь q_2 и r_2 – частное и остаток, получаемые при делении b/a; q_1 и r_1 – частное и остаток от деления $(r_2 2^{16} + c)$ на a; q_0 и r_0 – частное и остаток от деления $(r_1 2^{16} + d)/a$. В начале процесса b необходимо загрузить в AX, а нуль – в регистр DX. Так как b имеет длину слова, то q_2 умещается в одном слове и переполнения при первом делении не возникает. Поскольку r_1 и r_2, находящиеся в регистре DX после второго и третьего деления, должны быть меньше a, то в двух последних операциях деления переполнения также невозможно. Данную процедуру можно легко распространить на деление n слов на одно слово. Реализация алгоритма для беззнаковых целых чисел вынесена в упр. 12. Деление знаковых чисел с многократной точностью реализуется делением абсолютных значений чисел и определением знака по алгебраическому правилу (см. упр. 13).

Так как в команде DIV допускается делитель, максимальная длина которого одно слово, этой командой трудно воспользоваться, если делитель длиннее слова. Возможно, что такие операции придется выполнять на разрядном уровне, пользуясь командами сдвигов, которые рассматриваются в § 3.9. Однако в упр. 22 дан вариант процедуры деления двойного слова на двойное слово с помощью команды DIV.

```
ДИРЕКТИВЫ ОПРЕДЕЛЕНИЯ ДАННЫХ, КОМАНДЫ ВВОДА И ДР.
X, Y, Z И V ОПРЕДЕЛЕНЫ КАК ПЕРЕМЕННЫЕ-СЛОВА.

          MOV     AX,X            ;УМНОЖИТЬ (X) НА Y
          IMUL    Y               ;И ЗАПОМНИТЬ ПРОИЗВЕДЕНИЕ В BX:CX
          MOV     CX,AX
          MOV     BX,DX
          MOV     AX,Z            ;ПРИБАВИТЬ К ПРОИЗВЕДЕНИЮ В BX:CX
          CWD                     ;РАСШИРЕННОЕ СО ЗНАКОМ (Z)
          ADD     CX,AX
          ADC     BX,DX
          SUB     CX,540          ;ВЫЧЕСТЬ 540 ИЗ BX:CX
          SBB     BX,0
          MOV     AX,V            ;ВЫЧЕСТЬ (BX:CX) ИЗ (V) С РАСШИРЕННЫМ
          CWD                     ;ЗНАКОМ И РАЗДЕЛИТЬ НА (X),
          SUB     AX,CX           ;ЧАСТНОЕ ОБРАЗОВАТЬ В AX,
          SBB     DX,BX           ;А ОСТАТОК В DX
          IDIV    X

КОМАНДЫ ВЫВОДА И ДР.
```

Рис. 3.21. Пример, содержащий все арифметические операции

Пример программного фрагмента, содержащего несколько операций двоичной арифметики одинарной точности, приведен на рис. 3.21. В нем реализованы следующие вычисления:

(AX) ← Частное ((V – (X * Y + Z – 540))/X)
(DX) ← Остаток

3.3.2. АРИФМЕТИКА УПАКОВАННЫХ BCD-ЧИСЕЛ

Упакованные BCD-числа хранятся по две цифры в байте в виде 4-битных групп, называемых тетрадами. Арифметико-логическое устройство может выполнять только двоичные сложения и вычитание, но с помощью коррекции суммы или разности получается правильный результат в упакованном BCD-формате. Правило коррекции для сложения гласит:

Если в результате сложения любых двух цифр получается двоичное число в диапазоне 1010 ... 1111, не являющееся допустимой BCD-цифрой, или если возник перенос в следующий десятичный разряд, то к полученной сумме прибавляется 6 (0110).

Приведем примеры:

Перенос из предыдущего разряда	0	0	
	+ 7	+0111	
	6	0110	
	13	1101	
		110	Прибавить 6, так как 1101 –
Перенос в следующий разряд → 1		0011	запрещенная тетрада
Перенос из предыдущего разряда	1	1	
	+ 9	+1001	
	9	1001	
	19 1	0011	
		110	Прибавить 6, так как
Перенос в следующий разряд → 1		1001	возник перенос

Фактически, это правило необходимо для пропуска шести наборов, запрещенных в BCD-формате, когда такой пропуск оправдан.

В микропроцессоре 8086 для реализации BCD-сложения допускается только операция сложения с байтами. Поскольку байт содержит две тетрады (цифры), для фиксации переноса из бита 3 требуется флажок AF (это перенос из старшего бита младшей тетрады). Перенос же из старшей тетрады фиксирует флажок CF. Осуществляя двоичное сложение двухбайтных операндов и сопровождая его командой, которая анализирует состояния флажков AF и CF и тетрады на запрещенные комбинации, а затем прибавляет 6 для коррекции суммы, можно получить правильный результат в упакованном BCD-формате.

Нетрудно вывести аналогичное правило для вычитания: фиксируются заемы вместо переносов, а для получения правильного упакованного BCD-результата применяется специальная команда коррекции разностей, которая действует после команд вычитания. Специальные команды коррекции сложения DAA и вычитания DAS представлены на рис. 3.22. Если команда DAA прибавляет шестерки в нужные тетрады, то команда DAS вычитает их. Эти команды требуют, чтобы сумма или разность находились в регистре AL.

НАЗВАНИЕ	МНЕМОНИКА И ФОРМАТ	ОПИСАНИЕ
ДЕСЯТИЧНАЯ КОРРЕКЦИЯ ДЛЯ СЛОЖЕНИЯ	DAA	(AL) ← СУММА В AL КОРРЕКТИРУЕТСЯ В УПАКОВАННЫЙ BCD-ФОРМАТ
ДЕСЯТИЧНАЯ КОРРЕКЦИЯ ДЛЯ ВЫЧИТАНИЯ	DAS	(AL) ← РАЗНОСТЬ В AL КОРРЕКТИРУЕТСЯ В УПАКОВАННЫЙ BCD-ФОРМАТ

ФЛАЖКИ. ФЛАЖОК OF НЕ ОПРЕДЕЛЕН, А ВСЕ ОСТАЛЬНЫЕ ФЛАЖКИ УСЛОВИЙ МОДИФИЦИРУЮТСЯ.

РЕЖИМЫ АДРЕСАЦИИ. ОПЕРИРУЮТ ТОЛЬКО РЕГИСТРОМ AL.

Рис. 3.22. Команды коррекции упакованного BCD-формата

На рис. 3.23 показан фрагмент вычисления выражения

BCD3 ← BCD1 + BCD2.

Здесь BCD1 и BCD2 являются переменными-словами, которые содержат 4-разрядные

КОМАНДА		ДЕЙСТВИЕ		СОДЕРЖИМОЕ AL	CF	AF
MOV	AL,BCD1	(AL) ←	34	34	--	--
ADD	AL,BCD2	(AL) ←	34 + 89	BD	0	0
DAA			СКОРРЕКТИРОВАТЬ	23	1	*
MOV	BCD3,AL	(BCD3) ←	23	23	1	*
MOV	AL,BCD1+1	(AL) ←	18	18	1	*
ADC	AL,BCD2+1	(AL) ←	18 + 27 + (CF)	40	0	1
DAA			СКОРРЕКТИРОВАТЬ	46	0	*
MOV	BCD3+1,AL	(BCD3+1) ←	46	46	0	*

* НЕ ВАЖЕН

Рис. 3.23. Сложение в упакованном BCD-формате

BCD-числа в десятичном допонпительном коде, причем старшие цифры находятся в старших байтах слов. В правых столбцах, показывающих содержимое AL, CF и AF после выполнения каждой команды, предполагается, что (BCD1) = 1834 и (BCD2) = 2789. Фрагмент на рис. 3.24 реализует вычитание

$$BCD3 \leftarrow BCD1 - BCD2$$

4-разрядных чисел в десятичном дополнительном коде. Столбцы справа показывают содержимое AL, CF и AF в предположении, что (BCD1) = 1234 и (BCD2) = 4612.

КОМАНДА		ДЕЙСТВИЕ		СОДЕРЖИМОЕ AL	CF	AF
MOV	AL,BCD1	(AL) ←	34	34	--	--
SUB	AL,BCD2	(AL) ←	34 - 12	22	0	0
DAS			СКОРРЕКТИРОВАТЬ	22	0	*
MOV	BCD3,AL	(BCD3) ←	22	22	0	*
MOV	AL,BCD1+1	(AL) ←	12	12	0	*
SBB	AL,BCD2+1	(AL) ←	12 - 46 - (CF)	CC	1	1
DAS			СКОРРЕКТИРОВАТЬ	66	1	*
MOV	BCD3+1,AL	BCD3+1 ←	(AL)	66	1	*

* НЕ ВАЖЕН

РЕЗУЛЬТАТ = 6622 = -3378 В ДЕСЯТИЧНОМ ДОПОЛНИТЕЛЬНОМ КОДЕ.

Рис. 3.24. Вычитание в упакованном BCD-формате

Команды умножения и деления упакованных BCD-чисел отсутствуют, поскольку здесь байт содержит две цифры, а простых алгоритмов обработки сразу по две цифры нет. Если известно, что множитель (частное) невелик(о), можно применить циклические сложения (вычитания), но, в общем, более рационально преобразовать операнды в двоичные эквиваленты и воспользоваться двоичной арифметикой.

3.3.3. АРИФМЕТИКА НЕУПАКОВАННЫХ BCD-ЧИСЕЛ

В данном формате байт содержит всего одну десятичную цифру, благодаря чему можно реализовать умножение и деление неупакованных BCD-чисел. Операции сложения, вычитания и умножения с неупакованными BCD-числами выполняются примерно так же, как и операции сложения и вычитания над упакованными BCD-числами: сначала двоичные операции воздействуют на байты операндов, а затем полученный результат корректируется. Коррекция для деления осуществляется до операции двоичного деления. Во всех четырех арифметических операциях старшие тетрады операндов должны быть нулевыми (в сложении и вычитании ошибочные результаты могут и не получиться, но старшая тетрада результата может оказаться бессмысленной). На рис. 3.25 показаны команды коррекции для четырех арифметических операций, а на рис. 3.26 – программный фрагмент для вычисления выражения

$$(DX) \leftarrow UP1 + UP2 - UP3$$

НАЗВАНИЕ	МНЕМОНИКА И ФОРМАТ		ОПИСАНИЕ
КОРРЕКЦИЯ НЕУПАКОВАННОГО BCD-СЛОЖЕНИЯ	AAA	(AL) ←	СУММА В AL КОРРЕКТИРУЕТСЯ В НЕУПАКОВАННЫЙ BCD-ФОРМАТ
		(AH) ←	(AH) + ПЕРЕНОС ОТ КОРРЕКЦИИ
КОРРЕКЦИЯ НЕУПАКОВАННОГО BCD-ВЫЧИТАНИЯ	AAS	(AL) ←	РАЗНОСТЬ В AL КОРРЕКТИРУЕТСЯ В НЕУПАКОВАННЫЙ BCD-ФОРМАТ
		(AH) ←	(AH) - ЗАЕМ ОТ КОРРЕКЦИИ
КОРРЕКЦИЯ НЕУПАКОВАННОГО BCD-УМНОЖЕНИЯ	AAM	(AX) ←	ПРОИЗВЕДЕНИЕ В AL КОРРЕКТИРУЕТСЯ В НЕУПАКОВАННЫЙ BCD-ФОРМАТ, А AH СОДЕРЖИТ СТАРШУЮ ЦИФРУ
КОРРЕКЦИЯ НЕУПАКОВАННОГО BCD-ДЕЛЕНИЯ	AAD	(AL) ←	10*(AH)+(AL)
		(AH) ←	0

ПРИМЕЧАНИЕ. СТАРШАЯ ТЕТРАДА ОПЕРАНДОВ ДОЛЖНА БЫТЬ НУЛЕВОЙ.

ФЛАЖКИ. В КОМАНДАХ AAA И AAS ФЛАЖКИ AF И CF УСТАНАВЛИВАЮТСЯ, ЕСЛИ ПРОИСХОДИТ КОРРЕКЦИЯ, А ОСТАЛЬНЫЕ ФЛАЖКИ НЕ ОПРЕДЕЛЕНЫ. КОМАНДЫ AAM И AAD УСТАНАВЛИВАЮТ ФЛАЖКИ PF, SF И ZF ПО СООТВЕТСТВУЮЩИМ ПРАВИЛАМ, А ФЛАЖКИ OF, AF И CF ОСТАВЛЯЮТ НЕ ОПРЕДЕЛЕННЫМИ.

РЕЖИМЫ АДРЕСАЦИИ. ОПЕРАНД НАХОДИТСЯ В РЕГИСТРАХ AX ИЛИ AL.

Рис. 3.25. Команды коррекции неупакованного BCD-формата

с двухразрядными числами в десятичном дополнительном коде. Расширение операций сложения и вычитания на многоразрядные числа очевидно (см. упр. 18).

В операциях умножения и деления старшие тетрады результатов должны содержать 0. Команды AAM заменяет (AH) на частное (AL)/10, а в (AL) образуется остаток. Команда AAD умножает (AH) на 10, прибавляет произведение к (AL), а затем сбрасы-

```
ДИРЕКТИВЫ ОПРЕДЕЛЕНИЯ ДАННЫХ, КОМАНДЫ ВВОДА И ДР.
UP1, UP2 И UP3 ОПРЕДЕЛЕНЫ КАК ПЕРЕМЕННЫЕ-БАЙТЫ.

        MOV     AL,UP1          ;СЛОЖИТЬ МЛАДШИЕ ЦИФРЫ
        ADD     AL,UP2          ;И СКОРРЕКТИРОВАТЬ
        AAA
        MOV     DL,AL           ;ПЕРЕДАТЬ СУММУ В DL
        MOV     AL,UP1+1        ;СЛОЖИТЬ СТАРШИЕ ЦИФРЫ
        ADC     AL,UP2+1        ;С УЧЕТОМ ПЕРЕНОСА
        AAA                     ;И СКОРРЕКТИРОВАТЬ
        XCHG    AL,DL           ;ОБМЕНЯТЬ AL И DL
        SUB     AL,UP3          ;ВЫЧЕСТЬ МЛАДШУЮ ЦИФРУ ИЗ СУММЫ
        AAS                     ;И СКОРРЕКТИРОВАТЬ
        XCHG    AL,DL           ;ОБМЕНЯТЬ AL И DL
        SBB     AL,UP3+1        ;ВЫЧЕСТЬ СТАРШУЮ ЦИФРУ ИЗ СУММЫ
        AAS                     ;И СКОРРЕКТИРОВАТЬ
        MOV     DH,AL           ;ПЕРЕДАТЬ AL В DH

КОМАНДЫ ВЫВОДА И ДР.
```

Рис. 3.26. Пример сложения и вычитания в неупакованном BCD-формате

вает AH. (В правильности этих действий читателю рекомендуется убедиться самостоятельно – см. упр. 19 и 20.)

Беззнаковое умножение двухразрядных чисел осуществляется по следующему алгоритму:

$$\begin{array}{r} \times\ a_1\ a_0 \\ b_1\ b_0 \\ \hline c_{02}c_{01}c_{00} \\ c_{12}c_{11}c_{10} \\ \hline c_3\ c_2\ c_1\ c_0 \end{array}$$

Здесь A, B и C определены как переменные-байты. Реализация этого алгоритма для вычисления выражения

C ← A * B

показана на рис. 3.27 вместе с последовательностью действий и результатами, соответствующими (A) = 34 и (B) = 56.

```
        КОМАНДА                 ДЕЙСТВИЕ                    СОДЕРЖИМОЕ AX
MOV     AL,A            (AL) <-- 4                          -- 04
MUL     B               (AX) <-- 4*6                        00 18
AAM                        СКОРРЕКТИРОВАТЬ                  02 04
MOV     WORD PTR C0,AX  (C0+1:C0) <-- 24                    02 04
MOV     AL,A+1          (AL) <-- 3                          02 03
MUL     B               (AX) <-- 3*6                        00 12
AAM                        СКОРРЕКТИРОВАТЬ                  01 08
ADD     AL,C0+1         (AL) <-- 2+8                        01 0A
AAA                        СКОРРЕКТИРОВАТЬ                  02 00
MOV     WORD PTR C0+1,AX (C0+2:C0+1) <-- 20                 02 00
MOV     AL,A            (AL) <-- 4                          02 04
MUL     B+1             (AX) <-- 4*5                        00 14
AAM                        СКОРРЕКТИРОВАТЬ                  02 00
MOV     WORD PTR C1,AX  (C1+1:C1) <-- 20                    02 00
MOV     AL,A+1          (AL) <-- 3                          02 03
MUL     B+1             (AX) <-- 3*5                        00 0F
AAM                        СКОРРЕКТИРОВАТЬ                  01 05
ADD     AL,C1+1         (AL) <-- 5+2                        01 07
AAA                        СКОРРЕКТИРОВАТЬ                  01 07
MOV     WORD PTR C1+1,AX (C1+2:C1+1) <-- 17                 01 07
MOV     AL,C0           (AL) <-- 4                          01 04
MOV     C,AL            (C) <-- 4                           01 04
MOV     AL,C0+1         (AL) <-- 0                          01 00
ADD     AL,C1           (AL) <-- 0+0                        01 00
AAA                        СКОРРЕКТИРОВАТЬ                  01 00
MOV     C+1,AL          (C+1) <-- 0                         01 00
MOV     AL,C0+2         (AL) <-- 2                          01 02
ADC     AL,C1+1         (AL) <-- 2+7+(CF)                   01 09
AAA                        СКОРРЕКТИРОВАТЬ                  01 09
MOV     C+2,AL          (C+2) <-- 9                         01 09
MOV     AL,0            (AL) <-- 0                          01 00
ADC     AL,C1+2         (AL) <-- 0+1+(CF)                   01 01
AAA                        СКОРРЕКТИРОВАТЬ                  01 01
MOV     C+3,AL          (C+3) <-- 1                         01 01
```

Рис. 3.27. Умножение двухразрядных чисел в неупакованном BCD-формате

Деление на один разряд осуществляется по обычным правилам и сводится к вычислению

C ← Частное B/A
R ← Остаток

```
        КОМАНДА                 ДЕЙСТВИЕ                    СОДЕРЖИМОЕ AX
MOV     AH,0            (AH) <-- 0                          00 --
MOV     AL,B+1          (AL) <-- 5                          00 05
DIV     A                          РАЗДЕЛИТЬ                02 01
MOV     C+1,AL          (C1+1) <-- 1                        02 01
MOV     AL,B            (AL) <-- 3                          02 03
AAD                        СКОРРЕКТИРОВАТЬ                  00 17
DIV     A                          РАЗДЕЛИТЬ                02 07
MOV     C,AL            (C) <-- 7                           02 07
MOV     R,AH            (R) <-- 2                           02 07
```

Рис. 3.28. Деление в неупакованном BCD-формате

Предполагается, что A – одноразрядное и B – двухразрядное числа (см. рис. 3.28, где показаны точные действия и последовательное содержимое регистра AX при условии, что (A) = 3 и (B) = 53). Подробные вычисления имеют вид:

```
Первое частное ------\    /-- Второе частное
                      \  /
                       17
                    3 ) 53
Первый остаток  -->    23
                       2 <-- Второй (окончательный) остаток
```

Операция начинается с деления 5 на 3 командой DIV, которая помещает в AH остаток, в AL – частное. После запоминания частного, являющегося старшей цифрой неупакованного BCD-частного, и пересылки следующей цифры делимого в AL, команда AAD загружает в регистр AL значение 10*(AH) + (AL). Затем вновь действует команда DIV которая образует младшую BCD-цифру и окончательный остаток. С помощью повторяющихся команд AAD и DIV и запоминания частного можно разделить на одну цифру делимое, содержащее произвольное число цифр. Если делитель содержит более одной цифры, алгоритм значительно усложняется и целесообразнее преобразовать операнды в двоичную форму и осуществить двоичное деление.

3.4. КОМАНДЫ ПЕРЕХОДОВ

Команда перехода нарушает естественный порядок выполнения команд посредством загрузки в программный счетчик адреса команды, к которой осуществляется переход. В условном переходе замена содержимого программного счетчика адресом перехода зависит от состояний флажков. В микропроцессоре 8086 физический адрес состоит из смещения и сегмента, поэтому одни команды перехода модифицируют оба регистра IP и CS, а другие – только регистр IP. (Отметим, что мнемоники команд перехода начинаются с буквы J – от jump.) Команды, изменяющие содержимое регистров CS и IP, называются межсегментными переходами, а модифицирующие только содержимое IP – внутрисегментными. Межсегментные передачи управления реализуются только командами безусловных переходов.

3.4.1. КОМАНДЫ УСЛОВНЫХ ПЕРЕХОДОВ

Все команды условных переходов имеют следующий машинный формат:

Второй байт содержит 8-битное знаковое смещение (в дополнительном коде) относительно следующей команды в программе. Эффективный адрес перехода находится путем расширения знака D8 и прибавления его к содержимому IP после инкремента IP для адресации следующей по порядку команды. Следовательно, отрицательное смещение D8 означает переход назад, а положительное смещение D8 – переход вперед относительно следующей команды. Длина D8 ограничивает расстояние между адресом следующей команды и адресом перехода до диапазона –128 . . . + 127 байт. На рис. 3.29 показаны значения D8 для различных расстояний перехода.

РАССТОЯНИЕ (ДЕСЯТИЧНОЕ ЧИСЛО)	D8 (ШЕСТНАДЦАТЕРИЧНОЕ)	АДРЕС ПЕРЕХОДА
–128	80	(IP)–128
.	.	.
.	.	.
.	.	.
–2	FE	(IP)–2
–1	FF	(IP)–1
0	00	(IP)
1	01	(IP)+1
2	02	(IP)+2
.	.	.
.	.	.
.	.	.
127	7F	(IP)+127

Рис. 3.29. Соответствие между расстояниями перехода, значениями D8 и адресами перехода

Чтобы проанализировать, как ассемблер определяет значение D8, рассмотрим следующий фрагмент:

```
0050   AGAIN:   INC   CX
0052            ADD   AX,[BX]
0054            JNS   AGAIN
0056   NEXT:    MOV   RESULT,CX
```

Столбец слева показывает эффективный адрес первого байта каждой команды. После вычисления

```
  0050   Эффективный адрес перехода
- 0056   (IP), когда реализуется JNS
------
   -6
```

ассемблер установит значение D8, равное FA.

Ассемблерная команда условного перехода имеет следующий формат:

Операция Операнд

Здесь операнд определяет команду, к которой осуществляется переход. Операндом может быть метка или метка плюс (минус) выражение, вычисление которого дает константу . Мнемоника команды начинается с буквы J, а остальные буквы показывают условие, при котором осуществляется переход. Команды условных переходов приведены на

НАЗВАНИЕ	МНЕМОНИКА И ФОРМАТ		АЛЬТЕРНАТИВНАЯ МНЕМОНИКА	ПРОВЕРЯЕМОЕ УСЛОВИЕ
ПЕРЕЙТИ, ЕСЛИ НУЛЬ ИЛИ РАВНО	JZ	OPR	JE	ZF = 1
ПЕРЕЙТИ, ЕСЛИ НЕ НУЛЬ ИЛИ НЕ РАВНО	JNZ	OPR	JNE	ZF = 0
ПЕРЕЙТИ, ЕСЛИ ЗНАК УСТАНОВЛЕН	JS	OPR		SF = 1
ПЕРЕЙТИ, ЕСЛИ ЗНАК СБРОШЕН	JNS	OPR		SF = 0
ПЕРЕЙТИ, ЕСЛИ ЕСТЬ ПЕРЕПОЛНЕНИЕ	JO	OPR		OF = 1
ПЕРЕЙТИ, ЕСЛИ НЕТ ПЕРЕПОЛНЕНИЯ	JNO	OPR		OF = 0
ПЕРЕЙТИ, ЕСЛИ ПАРИТЕТ УСТАНОВЛЕН (ЧЕТНЫЙ ПАРИТЕТ)	JP	OPR	JPE	PF = 1
ПЕРЕЙТИ, ЕСЛИ ПАРИТЕТ СБРОШЕН (НЕЧЕТНЫЙ ПАРИТЕТ)	JNP	OPR	JPO	PF = 0
ПЕРЕЙТИ, ЕСЛИ НИЖЕ/НЕ ВЫШЕ ИЛИ РАВНО (БЕЗ ЗНАКА)	JB	OPR	JNAE, JC	CF = 1
ПЕРЕЙТИ, ЕСЛИ НЕ НИЖЕ/ВЫШЕ ИЛИ РАВНО (БЕЗ ЗНАКА)	JNB	OPR	JAE, JNC	CF = 0
ПЕРЕЙТИ, ЕСЛИ НИЖЕ ИЛИ РАВНО/НЕ ВЫШЕ (БЕЗ ЗНАКА)	JBE	OPR	JNA	CF∨ ZF = 1
ПЕРЕЙТИ, ЕСЛИ НЕ НИЖЕ ИЛИ РАВНО/ВЫШЕ (БЕЗ ЗНАКА)	JNBE	OPR	JA	CF∨ ZF = 0
ПЕРЕЙТИ, ЕСЛИ МЕНЬШЕ/НЕ БОЛЬШЕ ИЛИ РАВНО (СО ЗНАКОМ)	JL	OPR	JNGE	SF∀ OF = 1
ПЕРЕЙТИ, ЕСЛИ НЕ МЕНЬШЕ/БОЛЬШЕ ИЛИ РАВНО (СО ЗНАКОМ)	JNL	OPR	JGE	SF∀ OF = 0
ПЕРЕЙТИ, ЕСЛИ МЕНЬШЕ ИЛИ РАВНО/НЕ БОЛЬШЕ (СО ЗНАКОМ)	JLE	OPR	JNG	(SF∀ OF)∨ ZF = 1
ПЕРЕЙТИ, ЕСЛИ НЕ МЕНЬШЕ ИЛИ РАВНО/БОЛЬШЕ (СО ЗНАКОМ)	JNLE	OPR	JG	(SF∀ OF)∨ ZF = 0

ЕСЛИ ПРОВЕРЯЕМОЕ УСЛОВИЕ УДОВЛЕТВОРЯЕТСЯ, (IP)← (IP) + DB С РАСШИРЕНИЕМ ЗНАКА;
В ПРОТИВНОМ СЛУЧАЕ IP НЕ ИЗМЕНЯЕТСЯ И ПРОГРАММА ПРОДОЛЖАЕТСЯ В ЕСТЕСТВЕННОМ ПОРЯДКЕ.

ФЛАЖКИ. НИКАКИЕ ФЛАЖКИ НЕ МОДИФИЦИРУЮТСЯ.

РЕЖИМЫ АДРЕСАЦИИ. РЕЖИМ ОТНОСИТЕЛЬНО IP. ОПЕРАНД OPR ДОЛЖЕН БЫТЬ МЕТКОЙ В ПРЕДЕЛАХ -128...127 БАЙТ ОТ КОМАНДЫ, НАХОДЯЩЕЙСЯ ЗА КОМАНДОЙ ПЕРЕХОДА.

Рис. 3.30. Команды условных переходов

рис. 3.30. Некоторые из них имеют альтернативные мнемоники. Например, JNL (перейти, если не меньше) эквивалентна JGE (перейти, если больше или равно).

В первых десяти командах на рис. 3.30 условием перехода служит состояние одного флажка. Пять из них осуществляют переход, если соответствующий флажок находится в 1, и продолжают естественный порядок, если флажок содержит 0. Другие пять команд действуют "наоборот". Эти десять команд можно использовать в любой ситуации, когда в зависимости от состояния одного флажка необходимо предпринять одно из двух действий. Например, команду JZ можно применить после команды ADD, чтобы вызвать переход в случае нулевой суммы, а команду JNZ – для перехода при иенулевой сумме. Две эквивалентные структуры с привлечением команд JZ и JNZ показаны на рис. 3.31.

Последние восемь команд на рис. 3.30 предназиачены для использования после команд сравнения CMP, в которых второй операнд вычитается из первого. Путем сравнения двух операндов и перехода в зависимости от состояний соответствующих флажков реализуются решения, опирающиеся на операторы отношений $<$, $>$, $\leqslant$, $\geqslant$, $\neq$ и $=$. Команды

```
CMP  TOTAL,100
JGE  EXIT
```

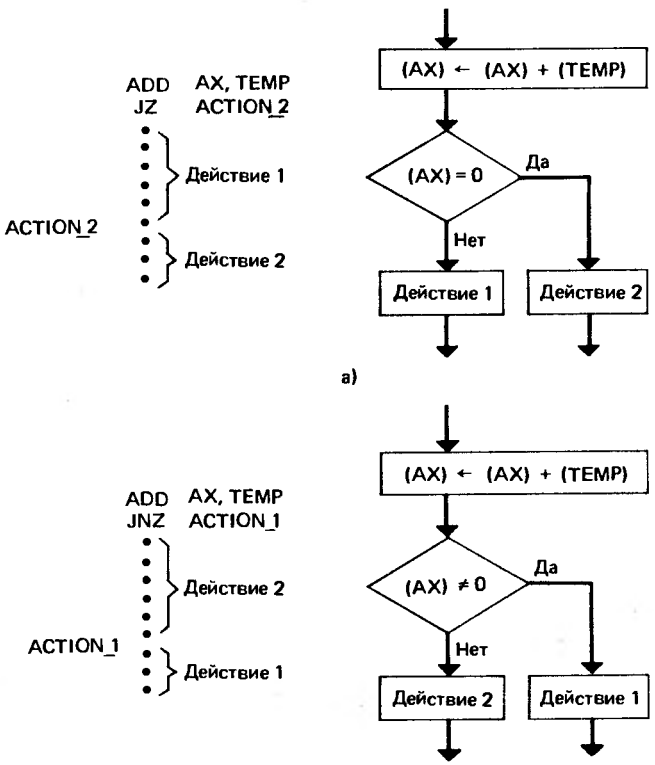


Рис. 3.31. Условные переходы по флажку ZF:
а – переход по ZF = 1; *б* – переход по ZF = 0

вызывают переход к метке EXIT, если содержимое TOTAL больше или равно 100. Рассматриваемые восемь команд можно разделить на две группы по четыре, причем первые из них соответствуют беззнаковым сравнениям, а вторые – знаковым. С командами беззнаковых сравнений ассоциируются термины "ниже" (B – below) и "выше" (A – above), а с командами знаковых сравнений – термины "меньше" (L – less) и "больше" (G – greater). Команды JE и JNE (эквивалентные соответственно командам JZ и JNZ) применяются вместе с командами сравнения для проверки операндов на равенство и неравенство. В этом случае нет различий в сравнении знаковых и беззнаковых величин.

Тот факт, что CF = 1 является правильным условием перехода в команде JB (перейти, если ниже), оказывается очевидным, так как команда CMP производит вычитание, а при этом флажок CF устанавливается, если и только если необходим беззнаковый заем (см. двоичную арифметику в разделе 3.3.1). Ясно, что проверяемым условием в команде JNB (перейти, если не ниже) является CF = 0, так как оно противоположно условию перехода в команде JB. Легко показать, что условиями переходов "если ниже или равно" и "если не ниже или равно" оказываются CF $\vee$ ZF = 1 и CF $\vee$ ZF = 0. Условия переходов в командах знаковых переходов проверить не так легко, но их можно доказать, рассматривая все возможные комбинации сравниваемых операндов. Мы этого делать не будем.

```
ДИРЕКТИВЫ ОПРЕДЕЛЕНИЯ ДАННЫХ, КОМАНДЫ ВВОДА И ДР.
X, Y И REZULT ОПРЕДЕЛЕНЫ КАК ПЕРЕМЕННЫЕ-СЛОВА.

                MOV     AX,X            ;ПЕРЕДАТЬ (X) В AX И ПЕРЕЙТИ
                CMP     AX,50           ; К TOO_HIGH, ЕСЛИ (X) БОЛЬШЕ 50
                JG      TOO_HIGH
                SUB     AX,Y            ;ИНАЧЕ ВЫЧЕСТЬ (Y) И ПЕРЕЙТИ
                JO      OVERFLOW        ;ПО ПЕРЕПОЛНЕНИЮ
                JNS     NONNEG          ;ЕСЛИ ПЕРЕПОЛНЕНИЯ НЕТ,
                NEG     AX              ;НАЙТИ АБСОЛЮТНОЕ ЗНАЧЕНИЕ
NONNEG:         MOV     RESULT,AX       ;И ЗАПОМНИТЬ В RESULT

КОМАНДЫ ВЫВОДА И ДР.
```

Рис. 3.32. Фрагмент с несколькими условными переходами

На рис. 3.32 показан программный фрагмент, содержащий несколько команд условных переходов. В нем осуществляется переход к TOO_HIGH, если X > 50, и к OVERFLOW, если знаковое вычитание X – Y вызывает переполнение. В противном случае вычисляется и загружается в RESULT значение |X – Y|.

3.4.2. КОМАНДЫ БЕЗУСЛОВНЫХ ПЕРЕХОДОВ

Имеется пять команд безусловного перехода. Их машинные форматы показаны на рис. 3.33, а ассемблерные команды – на рис. 3.34. Три команды осуществляют переход к точке в текущем сегменте кода, в две команды – в другой сегмент. Первый тип перехода называется *внутрисегментным*, а второй – *межсегментным*. Все пять команд имеют одинаковую мнемонику JMP и содержат один операнд.

Внутрисегментные переходы реализуются с помощью 8-битного смещения от текущего содержимого IP, 16-битного смещения от (IP) или получения адреса перехода из ячейки памяти, адресуемой командой. Вычисление 8-битного смещения производится так же, как в командах условных переходов, и имеет те же ограничения. Диапазон адресов переходов составляет –128 . . . +127 байт относительно адреса следующей команды. Вычисление 16-битного смещения выполняется аналогично, но теперь переход производится в любую точку текущего сегмента кода. Благодаря игнорированию переноса из 16-го бита текущий сегмент интерпретируется как кольцо. Следовательно, хотя ассемблер вычисляет смещение как число в дополнительном коде, диапазон составляет от – (IP) до 2^{16} – (IP), а не от -2^{15} до $2^{15} - 1$. Если, например, как показано на рис. 3.35, (IP) =

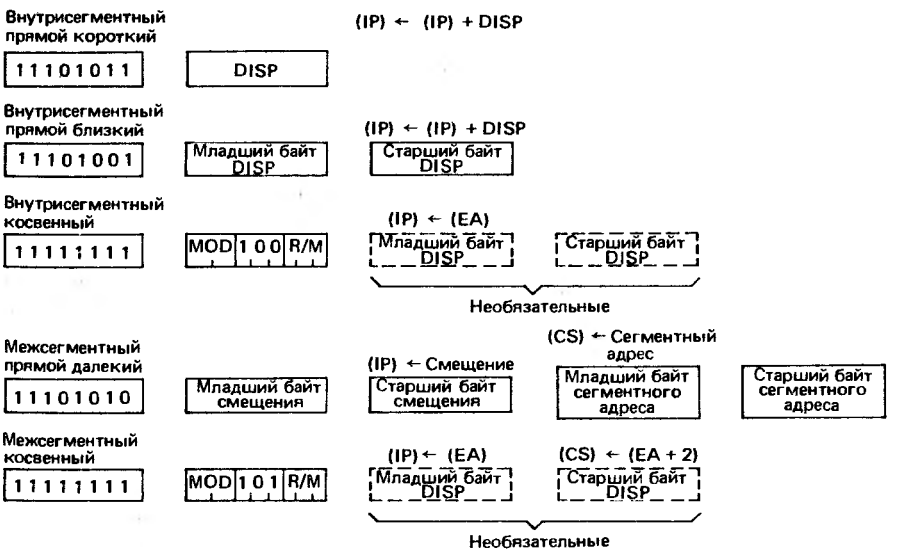


Рис. 3.33. Форматы машинного кода команд безусловных переходов

= 1000 и смещение равно EFFF или меньше, оно вызовет переход вперед, но если смещение равно F000 или больше, будет выполнен переход назад.

Межсегментные переходы модифицируют содержимое регистров CS и IP. Они могут быть прямыми, т. е. новое содержимое CS и IP является частью команды, или косвенными, причем в первом слове операнда находится новое содержимое IP, а в следующем слове – новое содержимое CS. В межсегментных косвенных переходах операнд должен указывать на ячейку памяти.

НАЗВАНИЕ	МНЕМОНИКА И ФОРМАТ	ОПИСАНИЕ
ВНУТРИСЕГМЕНТНЫЙ ПРЯМОЙ КОРОТКИЙ ПЕРЕХОД	JMP SHORT OPR	(IP)← (IP) ПЛЮС 8-БИТНОЕ СМЕЩЕНИЕ, ОПРЕДЕЛЯЕМОЕ OPR
ВНУТРИСЕГМЕНТНЫЙ ПРЯМОЙ БЛИЗКИЙ ПЕРЕХОД	JMP NEAR PTR OPR	(IP)← (IP) ПЛЮС 16-БИТНОЕ СМЕЩЕНИЕ, ОПРЕДЕЛЯЕМОЕ OPR
ВНУТРИСЕГМЕНТНЫЙ ПРЯМОЙ КОСВЕННЫЙ ПЕРЕХОД	JMP OPR *	(IP)← (EA), ГДЕ EA ОПРЕДЕЛЯЕТСЯ OPR
МЕЖСЕГМЕНТНЫЙ ПРЯМОЙ ДАЛЕКИЙ ПЕРЕХОД	JMP FAR PTR OPR	(IP)← СМЕЩЕНИЕ OPR В СЕГМЕНТЕ (CS)← СЕГМЕНТНЫЙ АДРЕС СЕГМЕНТА, СОДЕРЖАЩЕГО OPR
МЕЖСЕГМЕНТНЫЙ КОСВЕННЫЙ ПЕРЕХОД	JMP OPR *	(IP)← (EA), ГДЕ EA ОПРЕДЕЛЯЕТСЯ OPR (CS)← (EA+2), ГДЕ EA ОПРЕДЕЛЯЕТСЯ OPR

* ТИП ПЕРЕХОДА ОПРЕДЕЛЯЕТСЯ ТИПОМ ОПЕРАНДА.

ФЛАЖКИ. НИКАКИЕ ФЛАЖКИ НЕ МОДИФИЦИРУЮТСЯ.

РЕЖИМЫ АДРЕСАЦИИ. ВО ВНУТРИСЕГМЕНТНЫХ ПРЯМЫХ ПЕРЕХОДАХ ПРИМЕНЯЕТСЯ ОТНОСИТЕЛЬНЫЙ РЕЖИМ, В МЕЖСЕГМЕНТНЫХ ПРЯМЫХ ПЕРЕХОДАХ – ПРЯМОЙ РЕЖИМ. В КОСВЕННЫХ ПЕРЕХОДАХ НЕ ДОПУСКАЕТСЯ НЕПОСРЕДСТВЕННЫЙ РЕЖИМ, А В МЕЖСЕГМЕНТНЫХ КОСВЕННЫХ ПЕРЕХОДАХ ДОЛЖНА АДРЕСОВАТЬСЯ ПАМЯТЬ.

Рис. 3.34. Команды безусловных переходов

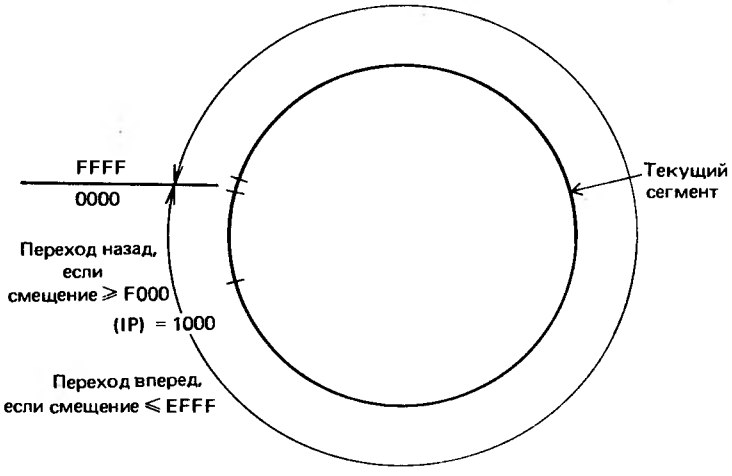


Рис. 3.35. Вычисление адреса перехода с использованием 16-битного смещения

Поскольку все команды безусловного перехода имеют одну и ту же мнемонику, ассемблеру необходимы какие-то средства для определения одного из пяти типов переходов. Хотя имеется несколько способов определения ассемблером типа перехода, для лучшего понимания программ целесообразно установить и соблюдать соглашение об указании типа команды. В книге принято соглашение о том, что в командах внутрисегментного прямого короткого перехода, внутрисегментного прямого близкого перехода и межсегментного прямого перехода содержатся соответственно атрибутные операторы SHORT, NEAR PTR и FAR PTR. Если такие префиксы применяются для этих типов команд всегда, отсутствие атрибутного оператора означает косвенный переход. В косвенном переходе тип операнда всегда определяет, каким будет переход: внутрисегментным или межсегментным. Если операнд является словом, предполагается внутрисег-

```
ДИРЕКТИВЫ ОПРЕДЕЛЕНИЯ ДАННЫХ, КОМАНДЫ ВВОДА И ДР.
N, И ARRAY ОПРЕДЕЛЕНЫ КАК ПЕРЕМЕННЫЕ-СЛОВА.

          MOV     CX,N            ;ЗАГРУЗИТЬ РАЗМЕР МАССИВА В CX
          MOV     BX,0            ;СБРОСИТЬ BX
          MOV     DI,BX           ;DI ДЛЯ ПОДСЧЕТА ЭЛЕМЕНТОВ БОЛЬШИХ 0
          MOV     SI,BX           ;SI ДЛЯ ПОДСЧЕТА ЭЛЕМЕНТОВ РАВНЫХ 0
AGAIN:    CMP     ARRAY[BX],0     ;ПЕРЕЙТИ К LES_OR_EQ,
          JLE     LES_OR_EQ       ;ЕСЛИ ЭЛЕМЕНТ МЕНЬШЕ ИЛИ РАВЕН 0,
          INC     DI              ;ИНАЧЕ ИНКРЕМЕНТ DI
          JMP     SHORT NEXT      ;И ПЕРЕХОД К NEXT
LES_OR_EQ: JL     NEXT            ;ЕСЛИ ЭЛЕМЕНТ МЕНЬШЕ ИЛИ РАВЕН 0,
          INC     SI              ;ПЕРЕЙТИ К NEXT, ИНАЧЕ ИНКРЕМЕНТ SI
NEXT:     ADD     BX,2            ;ИНКРЕМЕНТ СМЕЩЕНИЯ
          DEC     CX              ;ДЕКРЕМЕНТ СЧЕТЧИКА ЦИКЛА
          JNZ     AGAIN           ;И ПОВТОРЕНИЕ, ЕСЛИ ОН НЕ РАВЕН 0
          MOV     AX,N            ;ВЫЧЕСТЬ DI И SI ИЗ N
          SUB     AX,DI           ;ДЛЯ ПОЛУЧЕНИЯ ЧИСЛА
          SUB     AX,SI           ;ЭЛЕМЕНТОВ, МЕНЬШИХ 0
          JZ      SKIP            ;ЕСТЬ ОТРИЦАТЕЛЬНЫЕ ВЕЛИЧИНЫ?
          JMP     NEAR PTR NEG_VAL;ЕСЛИ ЕСТЬ, ПЕРЕЙТИ К NEG_VAL
SKIP:     .
          .
          .

КОМАНДЫ ВЫВОДА И ДР.
```

Рис. 3.36. Пример с командами условных и безусловных переходов

ментный переход, а если двойным словом (необходимым для хранения сегмента и смещения), – межсегментный. (Это означает, что тип операнда должен быть известным, когда ассемблер встречает команду перехода; об этом речь пойдет в § 3.11, а типы операндов рассматриваются в разделе 3.10.1.)

Для иллюстрации условных и безусловных переходов обратимся к программному фрагменту на рис. 3.36, который подсчитывает число положительных, нулевых и отрицательных значений в наборе из N слов, начинающемся в ARRAY. Вначале определяется число положительных значений (в регистре SI) и число нулевых значений (в регистре DI). Затем находится число отрицательных значений вычитанием (DI) и (SI) из N и образованием результата в AX. При наличии хотя бы одного отрицательного значения осуществляется близкий переход к NEG_VAL.

Так как условные переходы имеют ограниченный диапазон, для реализации их в удаленную точку приходится объединять команды условного и безусловного переходов. Такой прием иллюстрирует рис. 3.36. Условный переход JZ SKIP вызывает пропуск команды безусловного перехода JMP NEAR PTR NEG_VAL при отсутствии отрицательных величин. Если же метка NEG_VAL находится в другом сегменте кода, то NEAR следует заменить на FAR.

Действие межсегментного перехода показано на рис. 3.37. В этом примере команда JMP вызывает загрузку в IP смещения CONTINUE в сегменте кода CODE2 и загрузку в CS сегментного адреса CODE2.

```
CODE1       SEGMENT
                .
                .
                .
                JMP       FAR PTR CONTINUE
                .
                .
                .
CODE1       ENDS
                .
                .
                .
CODE2       SEGMENT
                .
                .
                .
COMTINUE:       MOV       AX,BX
                .
                .
                .
CODE2       ENDS
```

Рис. 3.37. Пример межсегментного перехода

Косвенные переходы не содержат адреса перехода в самой команде, а должны считывать его из ячейки памяти или из регистра. В команде внутрисегментного косвенного перехода для определения местоположения адреса перехода допускается применять любой режим адресации данных (конечно, за исключением непосредственной адресации). Межсегментный косвенный переход оказывается более ограниченным, так как в нем требуются смещение и сегмент; следовательно, операнд должен указывать на двойное слово в памяти и не указывать на регистр. Например, команда JMP TABLE[SI] считывает адрес перехода из ячейки памяти с адресом TABLE + (SI). Эту команду можно заставить выбрать адрес перехода из массива адресов и использовать ее как часть перехода по нескольким направлениям, аналогично оператору вычисляемого GO TO в языке Фортран.

3.5. КОМАНДЫ ЦИКЛОВ

На рис. 3.38 показан наиболее общий вид циклов с постпроверкой . Счетчик COUNT инициализируется на число повторений и на каждом проходе осуществляется его декремент. Когда счетчик достигает нуля, осуществляется выход из цикла. Если в качестве

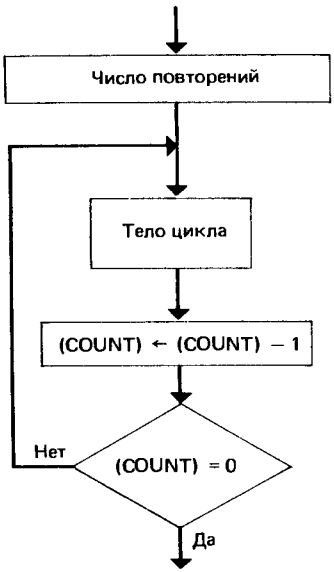


Рис. 3.38. Типичная структура цикла с постпроверкой

счетчика используется регистр CX и N содержит число повторений, то цикл с постпроверкой реализуется следующим образом:

```
                MOV     CX,N
BEGIN:          .       }
                .       }  ТЕЛО ЦИКЛА
                .       }
                DEC     CX
                JNZ     BEGIN
```

Команды циклов предназначены для упрощения действий декремента, проверки и перехода. В приведенном фрагменте эти действия реализуют две команды, но в более сложных ситуациях может потребоваться большее число их.

Команды циклов имеют следующий формат:

Здесь D8 представляет собой однобайтное смещение от текущего содержимого IP. Сходство команд циклов с командами условных переходов очевидно и они подчиняются тем же ограничениям, в частности, диапазон адреса перехода равен –128 . . . +127 байт от следующей команды. Команды циклов показаны на рис. 3.39. Из определения команды

НАЗВАНИЕ	МНЕМОНИКА И ФОРМАТ		АЛЬТЕРНАТИВНАЯ МНЕМОНИКА	ПРОВЕРЯЕМОЕ УСЛОВИЕ
ЗАЦИКЛИТЬ	LOOP	OPR		(CX) ≠ 0
ЗАЦИКЛИТЬ, ПОКА НУЛЬ ИЛИ РАВНО	LOOPZ	OPR	LOOPE	ZF = 1 И (CX) ≠ 0
ЗАЦИКЛИТЬ, ПОКА НЕ НУЛЬ ИЛИ НЕ РАВНО	LOOPNZ	OPR	LOOPNE	ZF = 0 И (CX) ≠ 0
ПЕРЕЙТИ ПО CX	JCXZ	OPR		(CX) = 0

ЗА ИСКЛЮЧЕНИЕМ КОМАНДЫ JCXZ, КОТОРАЯ НЕ ИЗМЕНЯЕТ (CX), ПРОИЗВОДИТСЯ (CX)⟵(CX)-1. ЗАТЕМ, ЕСЛИ ПРОВЕРЯЕМОЕ УСЛОВИЕ УДОВЛЕТВОРЯЕТСЯ, (IP)⟵(IP)+D8 С РАСШИРЕНИЕМ ЗНАКА; В ПРОТИВНОМ СЛУЧАЕ (IP) НЕ ИЗМЕНЯЕТСЯ И ПРОГРАММА ПРОДОЛЖАЕТСЯ В ЕСТЕСТВЕННОМ ПОРЯДКЕ.

ФЛАЖКИ. НИКАКИЕ ФЛАЖКИ НЕ МОДИФИЦИРУЮТСЯ.

РЕЖИМЫ АДРЕСАЦИИ. РЕЖИМ ОТНОСИТЕЛЬНО IP. ОПЕРАНД OPR ДОЛЖЕН БЫТЬ МЕТКОЙ, КОТОРАЯ НАХОДИТСЯ В ДИАПАЗОНЕ -128...127 БАЙТ ОТ КОМАНДЫ, СЛЕДУЮЩЕЙ ЗА КОМАНДОЙ ЦИКЛА.

Рис. 3.39. Команды циклов

LOOP следует, что реализация цикла с постпроверкой упрощается следующим образом:

```
        MOV   CX,N
        .
BEGIN:  .
        .
        LOOP  BEGIN
```

В общем, команды на рис. 3.39 являются командами переходов, но, поскольку они специально предназначены для реализации циклов, их называют командами циклов (или зацикливания).

На рис. 3.40 показан программный фрагмент, в котором команда LOOP применяется для сложения M слов, начинающихся с адреса ARRAY, и запоминания результата в

```
ДИРЕКТИВЫ ОПРЕДЕЛЕНИЯ ДАННЫХ, КОМАНДЫ ВВОДА И ДР.
M, ARRAY И TOTAL ОПРЕДЕЛЕНЫ КАК ПЕРЕМЕННЫЕ-СЛОВА.

                MOV     CX,M            ;ПЕРЕДАТЬ СЧЕТЧИК В CX
                MOV     AX,0            ;СБРОСИТЬ AX И SI
                MOV     SI,AX
START_LOOP:     ADD     AX,ARRAY[SI]    ;ПРИБАВИТЬ СЛЕДУЮЩИЙ ЭЛЕМЕНТ К AX
                ADD     SI,2            ;ИНКРЕМЕНТ ИНДЕКСА НА 2
                LOOP    START_LOOP      ;ПОВТОРИТЬ ЦИКЛ, ЕСЛИ (CX) НЕ РАВНО 0
                MOV     TOTAL,AX        ;ЗАПОМНИТЬ РЕЗУЛЬТАТ В TOTAL

КОМАНДЫ ВЫВОДА И ДР.
```

Рис. 3.40. Программа сложения массива двоичных чисел

TOTAL. Второй фрагмент, приведенный на рис. 3.41, осуществляет сложение 16-разрядных упакованных BCD-чисел. Предполагается, что каждое из чисел размещается в 8 смежных байтах, причем две младшие цифры находятся в байте с меньшим адресом. Сумма также запоминается в 8 байтах.

Команда JCXZ является по существу командой условного перехода, в которой решение принимается в зависимости от содержимого регистра CX, а не состояний флажков. Она отнесена к командам циклов, так как наиболее часто применяется для обхода цикла, если начальное значение счетчика равно 0.

Остальные команды циклов LOOPZ (или LOOPE) и LOOPNZ (или LOOPNE) в принятии решения о повторении цикла проверяют два условия. (Команды LOOPE и LOOPNE обычно применяются, когда команды циклов находятся после команд сравнений.) Клас-

```
ДИРЕКТИВЫ ОПРЕДЕЛЕНИЯ ДАННЫХ, КОМАНДЫ ВВОДА И ДР.
AUGEND, ADDEND И SUM ОПРЕДЕЛЕНЫ КАК ПЕРЕМЕННЫЕ-БАЙТЫ.

          MOV     CX,8            ;ПЕРЕДАТЬ СЧЕТЧИК В CX
          MOV     SI,0            ;СБРОСИТЬ ИНДЕКС
          CMP     AX,AX           ;СБРОСИТЬ ПЕРЕНОС
BEGIN:    MOV     AL,AUGEND[SI]   ;СЛОЖИТЬ ДВЕ
          ADC     AL,ADDEND[SI]   ;ЦИФРЫ
          DAA                     ;И ПОМЕСТИТЬ
          MOV     SUM[SI],AL      ;РЕЗУЛЬТАТ В SUM
          INC     SI              ;ИНКРЕМЕНТ ИНДЕКСА
          LOOP    BEGIN           ;ПОВТОРИТЬ, ЕСЛИ (CX) НЕ РАВНО 0

КОМАНДЫ ВЫВОДА И ДР.
```

Рис. 3.41. Программа сложения двух 16-разрядных упакованных BCD-чисел

сический пример проверки двух условий до повторения цикла связан с поиском в массиве заданного элемента. Пусть ASCII_STR является переменной, ассоциируемой с началом цепочки из L символов, и в цепочке необходимо найти символ пробела (код ASCII пробела равен 20_{16}.). Цикл заканчивается, когда обнаружен пробел или когда просмотрена вся цепочка. Если пробел не найден, осуществляется переход к NOT_FOUND. Фрагмент поиска в массиве показан на рис. 3.42.

```
ДИРЕКТИВЫ ОПРЕДЕЛЕНИЯ ДАННЫХ, КОМАНДЫ ВВОДА И ДР.
L ОПРЕДЕЛЕНА КАК ПЕРЕМЕННАЯ-СЛОВО, ASCII_STR - КАК ПЕРЕМЕННАЯ-БАЙТ.

          MOV     CX,L            ;ПЕРЕДАТЬ РАЗМЕР МАССИВА В CX
          MOV     SI,-1           ;ИНИЦИАЛИЗИРОВАТЬ ИНДЕКС
          MOV     AL,20H          ;ЗАГРУЗИТЬ В AL КОД ПРОБЕЛА
NEXT:     INC     SI              ;ИНКРЕМЕНТ ИНДЕКСА
          CMP     AL,ASCII_STR[SI] ;ПРОВЕРИТЬ НА ПРОБЕЛ
          LOOPNE  NEXT            ;ЗАЦИКЛИТЬ, ЕСЛИ НЕ ПРОБЕЛ
                                  ;И (CX) НЕ РАВНО 0
          JNZ     NOT_FOUND       ;ПЕРЕЙТИ К NOT_FOUND,
                                  ;ЕСЛИ ПРОБЕЛ НЕ НАЙДЕН

КОМАНДЫ ВЫВОДА И ДР.
```

Рис. 3.42. Пример поиска в массиве с применением команды LOOPNE

Циклы могут вкладываться друг в друга и переходы осуществляются либо внутри цикла, либо во вложенных циклах. Для иллюстрации более сложной ситуации рассмотрим программный фрагмент сортировки массива чисел в убывающем порядке. Схема алгоритма *пузырьковой сортировки* приведена на рис. 3.44, где N – число элементов в массиве, A – имя массива и I – индекс в массиве. Пузырьковая сортировка запускается с начала массива и перестраивает пары элементов в убывающем порядке. После первого прохода по массиву наименьший элемент оказывается последним; следовательно, на

```
ДИРЕКТИВЫ ОПРЕДЕЛЕНИЯ ДАННЫХ, КОМАНДЫ ВВОДА И ДР.
N И A ОПРЕДЕЛЕНЫ КАК ПЕРЕМЕННЫЕ-СЛОВА.

          MOV     CX,N            ;УСТАНОВИТЬ COUNT1
          DEC     CX              ;НА N-1
LOOP1:    MOV     DI,CX           ;ЗАПОМНИТЬ COUNT1 В DI
          MOV     BX,0            ;СБРОСИТЬ BX
LOOP2:    MOV     AX,A[BX]        ;ЗАГРУЗИТЬ A(I) В AX
          CMP     AX,A[BX+2]      ;И СРАВНИТЬ С A(I+1)
          JGE     CONT            ;ОБМЕНЯТЬ, ЕСЛИ A(I) < A(I+1),
          XCHG    AX,A[BX+2]      ;И ЗАПОМНИТЬ
          MOV     A[BX],AX        ;БОЛЬШЕЕ ЧИСЛО
CONT:     ADD     BX,2            ;ИНКРЕМЕНТ ИНДЕКСА
          LOOP    LOOP2           ;ЕСЛИ НЕ КОНЕЦ ПРОХОДА, ПОВТОРИТЬ
          MOV     CX,DI           ;ВОССТАНОВИТЬ COUNT1 ВНЕШНЕГО ЦИКЛА
          LOOP    LOOP1           ;ЕСЛИ НЕ ПОСЛЕДНИЙ ПРОХОД, ПОВТОРИТЬ

КОМАНДЫ ВЫВОДА И ДР.
```

Рис. 3.43. Программа пузырьковой сортировки

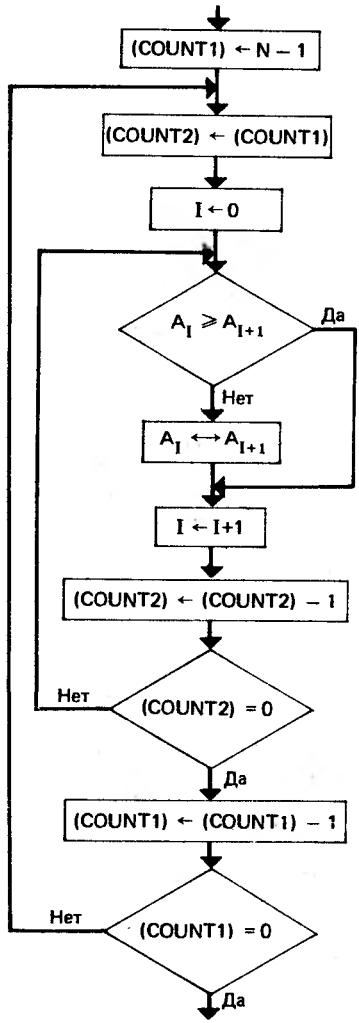


Рис. 3.44. Схема алгоритма пузырьковой сортировки

втором проходе необходимо анализировать только первые N – 1 элементов. На третьем проходе остаются N – 2 элементов и т. д. Программа пузырьковой сортировки представлена на рис. 3.43. Более эффективный метод сортировки, называемый сортировкой-вставкой, описан в упр. 34.

3.6. ХОЛОСТАЯ КОМАНДА И КОМАНДА ОСТАНОВА

С командами переходов довольно часто применяется холостая (пустая) команда, которая для микропроцессора 8086 имеет мнемонику NOP (рис. 3.45). Холостая команда выполняет ту же функцию, что и оператор CONTINUE в Фортране и других языках вы-

НАЗВАНИЕ	МНЕМОНИКА И ФОРМАТ	ОПИСАНИЕ
ХОЛОСТАЯ ОПЕРАЦИЯ	NOP	НЕ ВЫЗЫВАЕТ НИКАКИХ ДЕЙСТВИЙ
ОСТАНОВИТЬ	HLT	ПРЕКРАЩАЕТ ДЕЙСТВИЯ КОМПЬЮТЕРА

ФЛАЖКИ. НИКАКИЕ ФЛАЖКИ НЕ МОДИФИЦИРУЮТСЯ.

РЕЖИМЫ АДРЕСАЦИИ. НЕТ.

Рис. 3.45. Команды NOP и HLT

сокого уровня: служит меткой точки перехода, которая ничего не делает. Наличие такой команды особенно важно при разработке программы, когда неизбежны частые ее модификации. Введение метки перехода для команды, предпринимающей некоторое действие, в виде

```
        JZ      EXIT
        .
        .
        .
EXIT:   MOV   AX,SUM[BX]
```

оказывается недостаточно гибким, так как при необходимости введения новых команд в точке с меткой EXIT требуется повторить команду пересылки. Если же использовать такую последовательность

```
        JZ      EXIT
        .
        .
        .
EXIT:   NOP
        MOV   AX,SUM[BX]
```

то добавления осуществляются без переделки имеющейся программы. Данная возможность особенно важна на этапе отладки, когда в ключевых точках программы приходится временно вводить команды печати сообщений. Обычно ключевыми являются точки переходов. Наконец, как и в циклах DO языка Фортран, которые часто заканчиваются операторами CONTINUE, целесообразно помещать команды NOP в конце циклов с постпроверкой, особенно во вложенных циклах.

На рис. 3.45 приведена команда останова HLT, которая иногда также вводится в ключевых точках при разработке программы. Команда HLT заставляет компьютер прекратить действия. Анализируя его состояние при останове, можно получить информацию о предыдущих действиях компьютера. Команды HLT часто встречаются в диагностических программах, предназначенных для упрощения эксплуатации вычислительной системы.

3.7. КОМАНДЫ МАНИПУЛЯЦИЙ ФЛАЖКАМИ

Как уже говорилось, многие команды устанавливают или сбрасывают флажки в соответствии с полученным результатом. Однако иногда необходимо прямо управлять флажками. Желательно иметь возможность устанавливать, сбрасывать или инвертировать флажок CF, а далее будет показана необходимость прямого воздействия на управляющие флажки DF и IF. В примере на рис. 3.41 до входа в цикл необходимо сбросить флажок CF, так как в цикле все сложения, включая и сложение младших байт, реализуются командой ADC. Хотя в данном случае можно сбросить флажок CF командой CMP AX,AX, удобнее и эффективнее иметь команду, которая явно сбрасывает флажок CF. Команды манипуляций флажками микропроцессора 8086 представлены на рис. 3.46.

НАЗВАНИЕ	МНЕМОНИКА И ФОРМАТ	ОПИСАНИЕ
СБРОСИТЬ ПЕРЕНОС	CLC	CF ← 0
ИНВЕРТИРОВАТЬ ПЕРЕНОС	CMC	CF ← $\overline{CF}$
УСТАНОВИТЬ ПЕРЕНОС	STC	CF ← 1
СБРОСИТЬ НАПРАВЛЕНИЕ	CLD	DF ← 0
УСТАНОВИТЬ НАПРАВЛЕНИЕ	STD	DF ← 1
СБРОСИТЬ ПРЕРЫВАНИЕ	CLI	IF ← 0
УСТАНОВИТЬ ПРЕРЫВАНИЕ	STI	IF ← 1
ЗАГРУЗИТЬ АН ИЗ ФЛАЖКОВ *	LAHF	(AH) ← (МЛАДШИЙ БАЙТ PSW)
ЗАПОМНИТЬ АН ВО ФЛАЖКАХ *	SAHF	(МЛАДШИЙ БАЙТ PSW) ← (AH)

* БИТЫ 0, 2, 4, 6 И 7 ПЕРЕДАЮТСЯ В СООТВЕТСТВИИ С РИС. 2.8.
БИТЫ 1, 3 И 5 НЕ ОПРЕДЕЛЕНЫ.

ФЛАЖКИ. МОДИФИЦИРУЮТСЯ ТОЛЬКО УКАЗАННЫЕ ФЛАЖКИ.

РЕЖИМЫ АДРЕСАЦИИ. НЕТ.

Рис. 3.46. Команды манипуляций флажками

Команды STC, CLC и CMC, которые соответственно устанавливают, сбрасывают и инвертируют флажок переноса CF, обычно применяют в арифметике многократной точности. Команды STD и CLD устанавливают и сбрасывают флажок направления DF, который влияет на действие цепочечных команд (см. гл. 5), а команды STI и CLI — флажок прерывания IF, который управляет маскируемыми прерываниями (см. гл. 4 и 6). Последние две команды на рис. 3.46 предназначены для упрощения преобразования программ микропроцессора 8080 в программы микропроцессора 8086. Команда LAHF (SANF) передает младший байт PSW в(из) регистр(а) AH. Этот байт содержит все арифметические флажки за исключением флажка переполнения, которого нет в микропроцессоре 8080. Команды LAHF и SAHF можно использовать с логическими командами для инициализации или сравнения флажков SF, ZF, AF, PF и CF с заданным двоичным набором.

3.8. ЛОГИЧЕСКИЕ КОМАНДЫ

Команды микропроцессора 8086, реализующие логические операции, определены на рис. 3.47. Все команды обрабатывают операнды (байты или слова) поразрядно. Это означает, что команда NOT инвертирует каждый бит, а в остальных командах логическая

НАЗВАНИЕ	МНЕМОНИКА И ФОРМАТ		ОПИСАНИЕ
ИНВЕРТИРОВАТЬ	NOT	OPR	(OPR) ← ($\overline{OPR}$)
ОБ'ЕДИНИТЬ ПО "ИЛИ"	OR	DST,SRC	(DST) ← (DST) ∨ (SRC)
ОБ'ЕДИНИТЬ ПО "И"	AND	DST,SRC	(DST) ← (DST) ∧ (SRC)
СЛОЖИТЬ ПО MOD2 ("ИСКЛЮЧАЮЩЕЕ ИЛИ")	XOR	DST,SRC	(DST) ← (DST) ⊻ (SRC)
ПРОВЕРИТЬ	TEST	OPR1,OPR2	OPR1 ∧ OPR2

ФЛАЖКИ. КОМАНДА NOT НЕ ВОЗДЕЙСТВУЕТ НА ФЛАЖКИ. ОСТАЛЬНЫЕ КОМАНДЫ СБРАСЫВАЮТ CF И OF, ОСТАВЛЯЮТ AF НЕ ОПРЕДЕЛЕННЫМ И УСТАНАВЛИВАЮТ SF, ZF И PF ПО ОБЫЧНЫМ ПРАВИЛАМ.

РЕЖИМЫ АДРЕСАЦИИ. В КОМАНДЕ NOT НЕ ДОПУСКАЕТСЯ НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД. В ОСТАЛЬНЫХ КОМАНДАХ ОДИН ИЗ ОПЕРАНДОВ ДОЛЖЕН БЫТЬ РЕГИСТРОМ (ЗА ИСКЛЮЧЕНИЕМ НЕПОСРЕДСТВЕННОГО ОПЕРАНДА-ИСТОЧНИКА). ДРУГОЙ ОПЕРАНД МОЖЕТ ИМЕТЬ ЛЮБОЙ РЕЖИМ АДРЕСАЦИИ.

Рис. 3.47. Логические команды

операция выполняется над каждой парой соответствующих бит. Если, например, (DL) = = 10001010 и выполняется команда NOT DL, то новое (DL) = 01110101. Если (AL) = = 10010110 и (LOG_DATA) = 01011010, то команда OR AL,LOG_DATA образует (AL) = 11011110, команда AND AL,LOG_DATA дает (AL) = 00010010, а команда XOR AL,LOG_DATA формирует результат (AL) = 11001100.

Команда TEST аналогична команде AND, но она никуда не загружает результат. Как и команда сравнения CMP, она применяется только для установки флажков.

Команда NOT не воздействует на флажки. Остальные логические команды сбрасывают CF и OF, оставляют AF в неопределенном состоянии и воздействуют на SF, ZF и PF по обычным правилам. Если в двухоперандных командах источником является непосредственное значение, а получателем AL или AX, ассемблер генерирует короткую форму машинной команды (длиной 2 или 3 байта).

Логические команды предназначены для разнообразных целей, но наиболее часто их используют для селективных установки, инвертирования, сброса и проверки бит в операнде-получателе в соответствии с двоичным набором операнда-источника. Такие действия часто встречаются в операциях над битами регистров и данных ввода-вывода (см. гл. 6). При этом операнд-источник называется *маской*, а сама операция – *маскированием*. Биты селективно устанавливаются операцией OR следующим образом:

```
          ↓↓↓ ↓          Биты, установленные в 1
          01101001       Маска
OR        10100001       Получатель
          11101001       Новое содержимое получателя
          ↑ ↑ ↑↑         Состояния не изменились
```

Так как биты 0, 3, 5 и 6 маски содержат 1, то эти же биты в получателе будут установлены в 1, а так как биты 1, 2, 4 и 7 маски содержат 0, соответствующие биты в получателе остаются неизменными. Маскирование селективно устанавливает биты 0, 3, 5 и 6 и сохраняет остальные биты. Читателю предлагается самостоятельно убедиться в том, что команду XOR можно аналогичным образом использовать для селективного инвертирования бит.

Для селективного сброса бит применяется команда AND, в которой биты маски, соответствующие сбрасываемым битам, содержат 0, а остальные биты установлены в 1. Чтобы сбросить биты 2, 5 и 7, сохранив неизменными остальные, выполняется следующая операция:

```
          ↓ ↓ ↓          Сбрасываемые биты
          01011011       Маска
AND       10010101       Получатель
          00010001       Новое содержимое получателя
          ↑ ↑↑ ↑↑        Состояния не изменились
```

В селективной проверке, если любой из единичных бит маски соответствует единичным битам получателя, команда TEST формирует ZF = 0, а в противном случае ZF = 1. Обычно после команды TEST следуют команды JZ или JNZ, т. е. по результату проверки принимается решение. Проверка того, что все биты получателя, которые соответствуют единичным битам в маске, находятся в состоянии 1, реализуется следующим образом:

```
                01101101    Получатель
                10010010    Инверсия получателя
TEST (AND)      01100100    Маска
                --------
                00000000    Результат
                 ↑↑ ↑
                 └──┴──────Проверяемые биты
```

Если любой из проверяемых бит в исходном получателе содержит 0, результат окажется ненулевым. Когда получатель находится в AL, такую проверку осуществляют команды:

```
NOT   AL
TEST  AL,MASK_PAT
JZ    ALL_BITS_SET
```

Команда XOR применяется для проверки тождества двоичных наборов ее операндов. Команды

```
XOR  AX,TEST_PAT
JZ   MATCH
```

осуществляют переход к MATCH, когда двоичный набор в TEST_PAT точно соответствует набору в AX.

На рис. 3.48 показан программный фрагмент, в котором селективно устанавливаются биты 0 и 2 в регистре DL, инвертируются биты 1 и 6, сбрасываются биты 3 и 4, а за-

```
ДИРЕКТИВЫ ОПРЕДЕЛЕНИЯ ДАННЫХ. КОМАНДЫ ВВОДА И ДР.

        OR      DL,00000101B    ;УСТАНОВИТЬ БИТЫ 2 И 0
        XOR     DL,01000010B    ;ИНВЕРТИРОВАТЬ БИТЫ 6 И 1
        AND     DL,11100111B    ;СБРОСИТЬ БИТЫ 3 И 4
        MOV     AL,DL           ;ПРОДУБЛИРОВАТЬ РЕЗУЛЬТАТ В AL
        NOT     AL              ;И ПЕРЕЙТИ НА МЕТКУ
        TEST    AL,10000010B    ;EXIT, ЕСЛИ ОБА БИТА 7 И 1
        JZ      EXIT            ;УСТАНОВЛЕНЫ

КОМАНДЫ ВЫВОДА И ДР.
```

Рис. 3.48. Пример селективных операций над битами

```
ДИРЕКТИВЫ ОПРЕДЕЛЕНИЯ ДАННЫХ, КОМАНДЫ ВВОДА И ДР.

        NOT     AX              ;ПЕРЕЙТИ К TASK1, ЕСЛИ БИТЫ
        TEST    AX,4002H        ;14 И 1 УСТАНОВЛЕНЫ,
        JZ      TASK1           ;ИЛИ ЕСЛИ
        TEST    AX,280H         ;БИТЫ 9 И 7
        JZ      TASK1           ;УСТАНОВЛЕНЫ
        NOT     AX              ;ПЕРЕЙТИ К TASK2, ЕСЛИ
        TEST    AX,18H          ;БИТЫ 3 И 4
        JNZ     TASK2           ;УСТАНОВЛЕНЫ
TASK3:                          ;ИНАЧЕ ВЫПОЛНЯТЬ TASK3
          .
          .     } ЗАДАЧА 3
          .
TASK1:
          .
          .     } ЗАДАЧА 1
          .
TASK2:
          .
          .     } ЗАДАЧА 2
          .

КОМАНДЫ ВЫВОДА И ДР.
```

Рис. 3.49. Применение состояний бит для программного управления

тем проверяются биты 1 и 7 с переходом к EXIT, если оба они установлены. Рис. 3.49 представляет собой фрагмент, в котором осуществляются переходы к TASK1, TASK2 или TASK3 в результате анализа бит AX по следующим правилам:

1. Если биты 1 и 14 или 7 и 9 установлены, перейти к TASK1.

2. Если не было перехода к TASK1 и установлен бит 3 или 4, перейти к TASK2.

3. В противном случае выполнять TASK3.

Логические команды можно применять для вычислеиия логических выражений. Программный фрагмент на рис. 3.50, в котором биты 7, 6, . . . , 1, 0 регистра AL представля-

```
ДИРЕКТИВЫ ОПРЕДЕЛЕНИЯ ДАННЫХ, КОМАНДЫ ВВОДА И ДР.

                MOV     BL,00110101B    ;СОЗДАТЬ МАСКИ ДЛЯ КАЖДОГО
                OR      BL,AL           ;ИЗ ТРЕХ МИНТЕРМОВ
                MOV     BH,10101000B    ;ПУТЕМ УСТАНОВКИ В 1
                OR      BH,AL           ;БЕЗРАЗЛИЧНЫХ БИТ
                MOV     CL,01010101B
                OR      CL,AL
                XOR     BL,10111111B    ;ПЕРЕЙТИ К TRUE,
                JZ      TRUE            ;ЕСЛИ ЛЮБОЙ ИЗ МИНТЕРМОВ
                XOR     BH,11111011B    ;РАВЕН 1
                JZ      TRUE
                XOR     CL,01111101B
                JZ      TRUE
                XOR     AH,AH           ;ИНАЧЕ СБРОСИТЬ АН
                JMP     CONT
TRUE:           MOV     AH,1            ;УСТАНОВИТЬ АН В 1
CONT:           .
                .
                .

КОМАНДЫ ВЫВОДА И ДР.
```

Рис. 3.50. Фрагмент программы вычисления булева выражения

ют собой значения логических переменных $X_7, \ldots, X_0$, вычисляет следующее булево выражение:

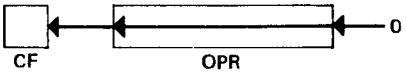
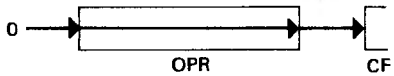
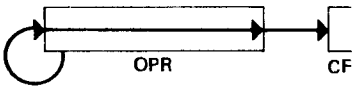
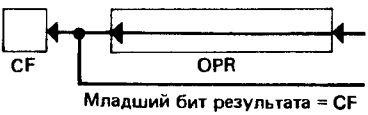
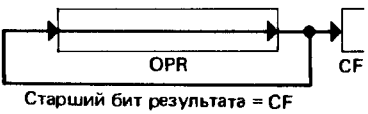
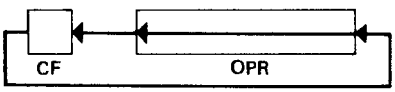
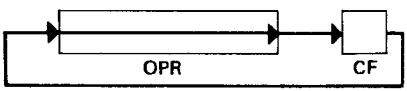
$$f(X_7, \ldots, X_0) = X_7\overline{X}_6X_3X_1 + X_6X_4\overline{X}_2X_1X_0 + \overline{X}_7X_5X_3\overline{X}_1$$

Значение f загружается в регистр АН. Первые шесть команд формируют маски для трех термов, объединяя по ИЛИ содержимое AL с байтами, которые содержат 1 в "неопределенных" позициях и 0 в остальных. Затем наборы термов сравниваются с масками и при наличии соответствия АН устанавливается в 1. В противном случае АН сбрасывает в 0 путем выполнения исключающего ИЛИ регистра АН с самим собой. Применение команды XOR для сброса регистра эффективнее команды MOV с непосредственным операндом, так как команда XOR требует всего 2 байта и выполняется быстрее (хотя при этом иесколько ухудшается читабельность программы).

3.9. КОМАНДЫ СДВИГОВ

Команды сдвигов и циклических сдвигов (ротаций) микропроцессора 8086, которые перемещают все биты операнда влево или вправо на указанное CNT число, приведены на рис. 3.51. В командах сдвига влево с правой стороны операнда "вдвигаются" нули, а старшие бнты выдвигаются с левой стороны и теряются, но последний из них сохраняется во флажке CF. Команды сдвига вправо аналогичным образом сдвигают биты вправо; однако команда SAR (арифметический сдвиг вправо) не помещает слева нули, а дублнрует в старшие биты знак операнда. Команды циклического сдвига отличаются от команд сдвнга тем, что операнд считается "кольцом", в котором выдвигаемые с одной стороны биты вдвигаются с другой стороны. В командах RCL н RCR кольцо включает в себя флажок переноса CF, а в командах ROL и ROR не включает, хотя флажок переноса участвует во всех случаях.

Команды сдвигов воздействуют на все арифметнческие флажки, а команды циклическнх сдвигов влияют только на флажки CF и OF. Однако во всех командах сдвигов состоянне флажка AF не определелио. Во всех командах флажок OF содержит полезную

Название	Мнемоника и формат	Описание*
Сдвинуть логически влево	SHL OPR, CNT	
Сдвинуть арифметически влево	SAL OPR, CNT	Аналогична SHL
Сдвинуть логически вправо	SHR OPR, CNT	
Сдвинуть арифметически вправо	SAR OPR, CNT	
Сдвинуть циклически влево	ROL OPR, CNT	
Сдвинуть циклически вправо	ROR OPR, CNT	
Сдвинуть циклически влево через перенос	RCL OPR, CNT	
Сдвинуть циклически вправо через перенос	RCR OPR, CNT	

*Число сдвигов определяется CNT.

Флажки. Перенос CF устанавливается в соответствии с рисунками. Флажки PF, SF и ZF модифицируются командами сдвигов, но команды циклических сдвигов на них не воздействуют. Флажок OF имеет смысл, если только счетчик равен 1. Команды сдвигов влияют на состояние флажка AF, но оно определенного смысла не имеет

Режимы адресации. OPR может иметь любой режим, кроме непосредственного. CNT должен быть 1 или CL.

Рис. 3.51. Команды сдвигов и циклических сдвигов

информацию, если только счетчик равен 1. В этом случае он устанавливается по правилу: если два старших бита операнда равны, то OF = 0, а в противном случае OF = 1. Большей частью наиболее важно состояние CF, даже когда остальные флажки устанавливаются по определенным правилам.

Операнд-получатель OPR определяется в любом режиме адресации, за исключением непосредственного. Счетчик CNT может быть 1, выражением-константой, значением которого является 1, или регистром CL. Если им указан CL, то число позиций, на которое осуществляется сдвиг, определяется содержимым CL. Команды SHR QUAN[SI],1 и SHR QUAN[SI],ONE одинаковы, если ONE является идентификатором, ассоциируемым с числом 1 (см. раздел 3.10.4). Команды

```
MOV  CL,6
SAL  DATA[BX][DI],CL
```

вызывают сдвиг содержимого ячейки, адресуемой EA, влево на 6 разрядов. Команда SHR AL,3 оказывается недействительной, так как константа CNT может быть равна только 1. Команда SHR AL,BL также недействительна, поскольку в качестве счетчика разрешается указывать только регистр CL. На рис. 3.52 показаны действия нескольких

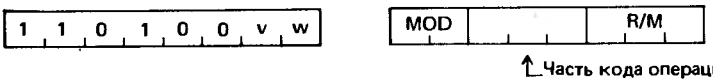
КОМАНДА			РЕЗУЛЬТАТЫ	
SHL	DL, CL	CF = 0	01101000	
SAL	DL, CL	CF = 0	01101000	
SHR	DL, CL		00010001	CF = 1
SAR	DL, CL		11110001	CF = 1
ROL	DL, CL	CF = 0	01101100	
ROR	DL, CL		10110001	CF = 1
RCL	DL, CL	CF = 0	01101110	
RCR	DL, CL		01110001	CF = 1

В ИСХОДНОМ СОСТОЯНИИ
(DL) = 10001101
(CL) = 00000011
(CF) = 1

Рис. 3.52. Примеры команд сдвигов и циклических сдвигов

команд сдвигов и циклических сдвигов при следующих начальных значениях: (DL) = = 8D, (CL) = 3 и CF = 1.

Машинный формат команд сдвигов и циклических сдвигов имеет следующий вид:



Бит W по-прежнему определяет размер операнда – байт или слово. Бит V установлен в 0, если счетчик сдвигов равен 1, и в 1, когда счетчик определяется регистром CL. Три средних бита во втором байте идентифицируют одну из семи команд.

Команды сдвигов и циклических сдвигов применяют для изменения формата данных, управления программой в некоторых ситуациях и в других целях. Отметим, что сдвиг влево на n позиций эквивалентен умножению на 2^n, например

$$6 \times 2 = 00000110 \times 100 = 00011000.$$

Сдвиг вправо аналогичен делению на 2^n. Логический сдвиг вправо соответствует беззнаковому делению, а арифметический – знаковому.

На рис. 3.53 приведена программа преобразования 16-разрядного неупакованного BCD-числа, начинающегося в UNPACKED, в упакованное BCD-число с запоминанием его в PACKED. Команды сдвига применяются здесь для удаления лишних бит. В упр. 35 читателю предлагается написать программу обратного преобразования.

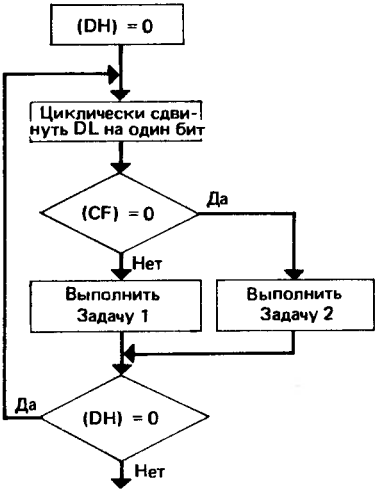
```
ДИРЕКТИВЫ ОПРЕДЕЛЕНИЯ ДАННЫХ, КОМАНДЫ ВВОДА И ДР.
UNPACKED И PACKED ОПРЕДЕЛЕНЫ КАК ПЕРЕМЕННЫЕ-БАЙТЫ.

            MOV     DX,8            ;СЧЕТЧИК ЦИКЛА В DX
            MOV     CL,4            ;СЧЕТЧИК СДВИГОВ В CL
            MOV     SI,0            ;СБРОСИТЬ ИНДЕКСЫ
            MOV     DI,SI           ; В SI И DI
CONVERT:    MOV     AX, WORD PTR UNPACKED[SI]  ;ПЕРЕДАТЬ
            SHL     AL,CL           ;ДВЕ ЦИФРЫ В AX
            SHR     AX,CL           ;И УПАКОВАТЬ ИХ
            MOV     PACKED[DI],AL   ;ЗАПОМНИТЬ РЕЗУЛЬТАТ
            ADD     SI,2            ;ИНКРЕМЕНТ ИНДЕКСОВ
            INC     DI
            DEC     DX              ;ДЕКРЕМЕНТ СЧЕТЧИКА ЦИКЛА
            JNZ     CONVERT         ;ПОВТОРИТЬ, ЕСЛИ СЧЕТЧИК НЕ РАВЕН 0

КОМАНДЫ ВЫВОДА И ДР.
```

Рис. 3.53. Преобразование 16-разрядного неупакованного BCD-числа в упакованное BCD-число

Для иллюстрации команд циклического сдвига на рис. 3.54, *а* приведена схема программного управления, а на рис. 3.54, *б* – соответствующая программа. Программа содержит цикл, выход из которого осуществляется, когда предварительно сброшенный регистр DH оказывается ненулевым. При каждом проходе по циклу содержимое DL циклически сдвигается влево на один разряд. В зависимости от состояния бита DL, выдвинутого во флажок CF, решается одна из двух задач. Предполагается, что при этом в



а)

Директивы определения данных, команды ввода и др.

```
            MOV     DH,0
REPEAT:     ROL     DL,1
            JB      TASK1
            JMP     NEAR PTR TASK2
TASK1:      .
            .       Задача 1
            .
            JMP     NEAR PTR NEXT
TASK2:      .
            .       Задача 2
            .
NEXT:       CMP     DH,0
            JNE     DONE
            JMP     NEAR PTR REPEAT
DONE:       .
            .
            .
```

Команды вывода и др.

б)

Рис. 3.54. Пример программного управления с применением команды ROL: *а* – схема алгоритма; *б* – код

определенных условиях в регистр DH загружается ненулевое значение. Чтобы отвести достаточно места для задач, в некоторых точках применяются близкие условные переходы (типа NEAR) вместо коротких условных (типа SHORT).

3.10. ДИРЕКТИВЫ И ОПЕРАТОРЫ

Ассемблерные команды транслируются в машинные и в программах на языках высокого уровня соответствуют исполняемым операторам. Но в этих программах должны быть и неисполненные операторы, предназначенные для инициализации значений, резервирования памяти, назначения имен константам, формирования структур данных и окончания компилирования. Аналогичным образом в ассемблерных программах должны присутствовать директивы, выполняющие такие же функции. Поскольку важную роль в работе микропроцессора 8086 играют сегментные регистры, в ассемблере потребуются директивы, показывающие ассемблеру предполагаемое содержимое сегментных регистров при различных обстоятельствах в ходе ассемблирования. Более того, так как программирование на языке ассемблера довольно близко к фактическим действиям компьютера, в ассемблерах имеются директивы, предоставляющие программисту большую свободу в точном размещении данных и сегментации программы, чем при программировании на языке высокого уровня. Рассмотрим директивы, необходимые при разработке одномодульных программ (директивы объединения модулей обсуждаются в гл. 4).

3.10.1. ОПРЕДЕЛЕНИЕ ДАННЫХ И РАСПРЕДЕЛЕНИЕ ПАМЯТИ

Операторы инициализации данных и резервирования памяти имеют следующий формат:

Переменная Мнемоника Операнд. . .,Операнд ;Комментарий

Здесь переменная не обязательна, но при ее наличии ей назначается смещение первого байта, резервируемого директивой. Отметим, что в отличие от метки переменная должна заканчиваться пробелом, а не двоеточием. Мнемоника определяет длину каждого операнда:

DB (определить байт). Каждый операнд-данное занимает один байт.

DW (определить слово). Каждый операнд-данное занимает одно слово, причем его младшая половина находится в первом байте, а старшая – во втором байте.

DD (определить двойное слово). Каждый операнд-данное имеет длину в два слова, первым из которых следует младшее слово.

Операнды показывают инициализируемые данные или объем резервируемого пространства. Комментарии поясняют инициализацию данных или распределение, но могут отсутствовать. После переменной и мнемоники должны быть один или несколько пробелов; кроме этого условия размещение пробелов произвольно (конечно, их нельзя вводить в мнемоники, идентификаторы и т. д.).

Директивы BD, DW и DD применяются для резмещения данных в конкретных ячейках или просто для резервирования пространства без инициализации. Кроме того, директивы DW и DD используются для формирования смещений или полных адресов меток и переменных. Для инициализации данных операнд должен быть константой, выражением, вычисление которого дает константу, или цепочкой. Например, директивы

```
DATA_BYTE   DB   104, 10H
DATA_WORD   DW   100,100H,-5
DATA_DW     DD   3*20,0FFFDH
```

вызывают инициализацию байт в соответствии с рис. 3.55. Первый оператор инициализирует три смежных байта на константы и ассоциирует с первым байтом переменную

DATA_BYTE. Второй оператор инициализирует три смежных слова (6 байт) на константы и ассоциирует переменную DATA_WORD с первым байтом первого слова. Последний оператор резервирует два двойных слова и инициализирует их на значения выражений 3*20 и FFFD. Он также ассоциирует DATA_DW с первым байтом первого двойного слова.

8-разрядное число 12345678 определяется в упакованном BCD-формате директивой

```
PACKED   DB  78H,56H,34H,12H
```

а в неупакованном BCD-формате – директивой

```
UNPACKED  DB  8H,7H,6H,5H,4H,3H,2H,1H
```

Младшие байты находятся по меньшим адресам, представляемым PACKED и UNPACKED.

Символьную цепочку в коде ASCII можно инициализировать, используя в качестве операнда цепочку-константу. Оператор

```
MESSAGE  DB  'H','E','L','L','O'
```

загружает коды ASCII букв H(48), E(45), L(4C), L(4C) и O(4F) в смежные байты, начинающиеся с байта, адрес которого ассоциируется с переменной MESSAGE. Эквивалентная форма этого оператора имеет вид

```
MESSAGE  DB  'HELLO'
```

Отметим, что первый символ цепочки помещается в первый байт, второй – во второй и т. д. Директивы DW и DD также можно использовать для инициализации символь-

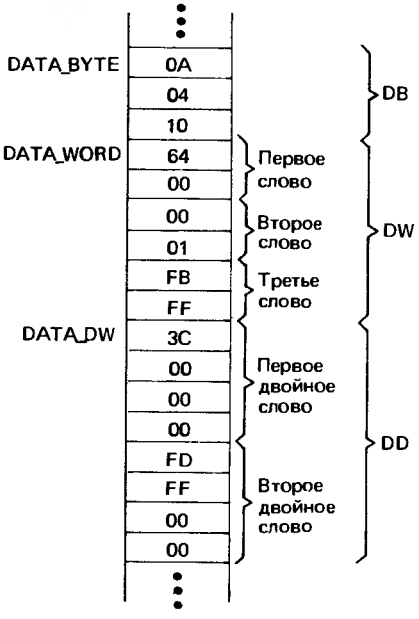


Рис. 3.55. Типичная инициализация данных с помощью директив DB, DW и DD

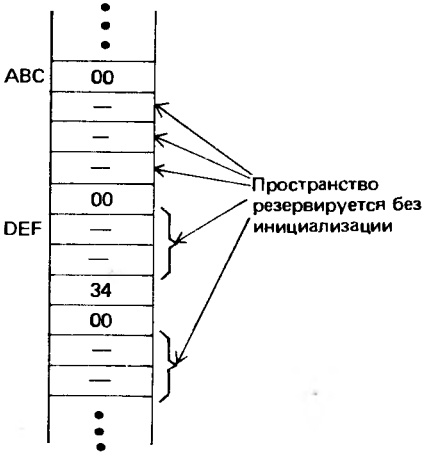


Рис. 3.56. Применение вопросительного знака в качестве операнда

ных цепочек, но для этого их применяют довольно редко, так как байты меняются местами. Например, директива DB 'AB' помещает 'A' в первый байт и 'B' во второй, а директива DW 'AB' помещает 'B' в первый байт и 'A' во второй.

Когда операндом служит вопросительный знак, инициализация не производится, но резервируется соответствующее пространство (один байт в директиве DB, два байта в директиве DW и четыре байта в операторе DD). Операторы

```
ABC   DB   0,?,?,?,0
DEF   DW   ?,52,?
```

генерируют последовательность байт, показанную на рис. 3.56.

Инициализация путем перечисления содержимого отдельных ячеек оказывается удовлетворительной при заполнении небольшого числа ячеек, но при наличии нескольких операндов такой подход будет громоздким. Поэтому ASM-86 допускает применение оператора дублирования DUP, аналогичного коэффициентам дублирования в операторах инициализации языков высокого уровня. Несколько операндов или наборов операндов разрешается заменять следующей конструкцией:

Выражение DUP (Операнд, . . . , Операнд)

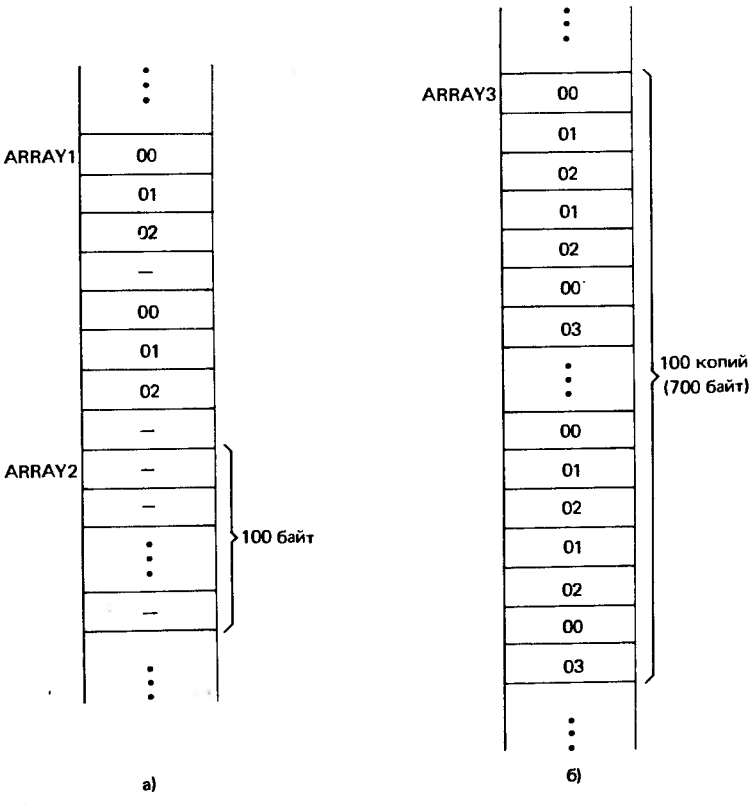


Рис. 3.57. Применение оператора дублирования без вложения (*а*) и с вложением (*б*)

Здесь выражение вычисляет положительное целое число, которое определяет число повторений набора операндов. Операторы

```
ARRAY1  DB  2 DUP(0,1,2,?)
ARRAY2  DB  100 DUP(?)
```

вызывают инициализацию и распределение, показанные на рис. 3.57, *а*. Первый оператор резервирует 8 байт для ARRAY1, а второй – 100 байт для ARRAY2. Первый оператор эквивалентен следующему оператору:

```
ARRAY1  DB  0,1,2,?,0,1,2,?
```

Допускается вложение операторов DUP, например оператор

```
ARRAY3  DB  100 DUP(0,2DUP(1,2),0,3)
```

вызывает инициализацию данных в соответствии с рис. 3.57, *б*.

Возможно также инициализировать смещение или полный адрес переменной или метки, указывая в поле операнда операторов DW или DD выражение, которое вычисляет смещение или адрес, т. е. имеет вид

Переменная ± Выражение-константа

или

Метка ± Выражение-константа

Если операнд находится в операторе DW, запоминается только смещение, а если в операторе DD, запоминаются смещение и сегментный адрес, причем смещение располагается в первом слове. Оператор DB нельзя применять для запоминания адреса, так как он резервирует всего 8 бит. Операторы

```
PARAMETER_TABLE  DW  PAR1
                 DW  PAR2
                 DW  PAR3
```

запоминают смещения PAR1, PAR2, PAR3 в соответствии с рис. 3.58, *а*. Отметим, что PAR1, PAR2 и PAR3 могут быть переменными или метками. Операторы вида

```
INTERSEG_DATA  DD  DATA1
               DD  DATA2
```

применяются для запоминания смещений и сегментных адресов (см. рис. 3.58, *б*).

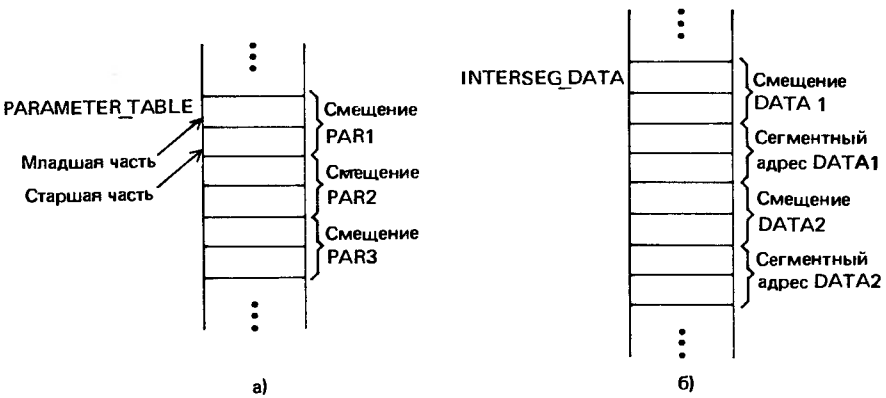


Рис. 3.58. Применение операторов DW (*а*) и DD (*б*) для инициализации адресов

Переменная, ассоциируемая с оператором определения памяти, представляет собой смещение первого байта первого элемента данных в текущем сегменте и имеет атрибут типа, показывающего длину каждого элемента данных в операторе. Длина равна 1 байту в операторе DB, 2 байтам в операторе DW и 4 байтам в операторе DD. Выражению

Переменная ± Выражение-константа

придается тот же атрибут типа, что и у переменной. Ассемблер использует атрибут типа для определения того, оперирует ли машинная команда байтом или словом (т. е. следует устанавливать или сбрасывать бит W в команде). Например, в командах

```
        MOV   OPER1, 0
        MOV   OPER2, 0
          .
          .
          .
OPER1   DB    ?,?
OPER2   DW    ?,?
```

бит W = 0 в первой команде MOV и W = 1 во второй команде.

Так как все переменные типизируются в соответствии с оператором, в котором они появляются, ассемблер может обнаружить ошибки кодирования, контролируя, чтобы оба операнда в двухоперандной команде имели один и тот же атрибут типа. Если в следующем фрагменте

```
        MOV   OPER1 + 1, AX
        MOV   OPER2,AL
          .
          .
          .
OPER1   DB    1,2
OPER2   DW    1010,2020
```

программистом указана первая команда, она заменит содержимое первого байта OPER2 на (AH). Так как команда наверняка не предназначена для этого действия, ассемблер не ассемблирует ее, а печатает сообщение об ошибке. Вторая команда также содержит операнды разных типов и вызовет сообщение об ошибке.

Однако допускается замена ("перевешивание") неявного типа, придаваемого операнду, для чего предназначен атрибутный оператор PTR со следующим форматом:

Тип PTR Переменная ± Выражение-константа

Здесь типом может быть BYTE, WORD и DWORD (двойное слово). В приведенном примере команда

```
MOV WORD PTR OPER1 + 1,AX
```

будет оперировать словом и ассемблируется без формирования сообщения об ошибке. Команда

```
MOV BYTE PTR OPER2,AL
```

заменит первый байт первого слова OPER2 на содержимое AL, а команда

```
MOV BYTE PTR OPER2 + 1,AL
```

загружает (AL) в старший байт первого слова OPER2.

Поскольку неудобно пользоваться оператором PTR для частого обращения к переменным, имеющим как бы два различных типа, предусмотрен способ ассоциирования ячейки с двумя различными переменными. Директива LABEL со следующим форматом

Переменная LABEL Тип

вызывает типизирование переменной и назначение ей текущего смещения. Директивы

```
BYTE_ARRAY  LABEL BYTE
WORD_ARRAY DW     50 DUP(?)
```

ассоциируют BYTE_ARRAY и WORD_ARRAY с одной и той же ячейкой – первым байтом 100-байтного блока. Команда

```
MOV WORD_ARRAY + 2,0
```

загружает 0 в третий и четвертый байты блока, а команда

```
MOV BYTE_ARRAY + 2,0
```

загружает 0 в третий байт.

3.10.2. СТРУКТУРЫ

Подобно массиву в языке Фортран, все элементы, распределяемые одним оператором определения памяти, должны быть одного и того же типа. Желательно, особенно при обработке экономической информации, чтобы переменная имела несколько полей с индивидуальными типами. Например, на рис. 3.59 показана структура данных записи о работнике. Чтобы в программе можно было определить структуры данных и обращаться к ним в целом, на многих языках высокого уровня, например Паскаль и PL/1, предусмотрены специальные неисполняемые операторы. В ассемблере ASM-86 для этого введена директива STRUC, которая позволяет модифицировать операнды команды таким образом, что они могут адресовать конкретные поля в структуре.

Определение структуры задает шаблон структуры и в упрощенном формате представляется в виде

```
Имя структуры STRUC
    .
    .  Последовательность директив DB, DW и DD
    .
Имя структуры ENDS
```

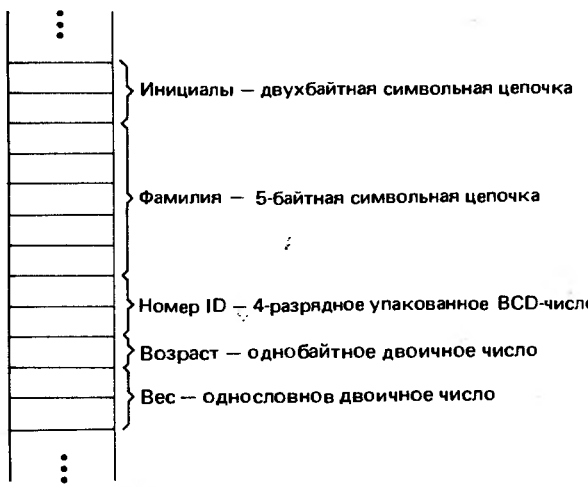


Рис. 3.59. Поля в типичной структуре данных (на примере записи о работнике)

Если операторы DB, DW и DD содержат идентификатор переменной, он обозначает начало поля и называется *идентификатором поля*. Структуру записи о работнике, показанную на рис. 3.59, можно определить в виде

```
PERSONNEL_DATA  STRUC
    INITIALS    DB      'XX'
    LAST_NAME   DB      5 DUP(?)
    ID          DB      0,0
    AGE         DB      ?
    WEIGHT      DW      ?
PERSONNEL_DATA  ENDS
```

Определение структуры не резервирует памяти и не инициализирует значений – оно просто определяет шаблон. Следовательно, для резервирования требуемого пространства определение необходимо сопроводить оператором для вызова структуры. Операторы вызова имеют следующий формат:

Переменная Имя структуры ‹Спецификации инициализации›

Здесь имя является именем структуры в директиве STRUC, а переменная ассоциируется с началом структуры. Переменная также используется с именами полей для обращения к различным полям в структуре.

Определение структуры может показывать начальные значения, которые считаются значениями по умолчанию и могут быть заменены спецификациями инициализации (за исключением случая, когда в одном операторе осуществляется несколько инициализаций). В приведенном примере можно заменить все поля, кроме LAST_NAME и ID. Спецификация инициализации состоит из последовательности выражений-констант (по одному выражению для каждого поля), разделенных запятыми. Если последовательность начинается с одной или нескольких запятых или в ней имеется несколько запятых подряд, этим указывается отсутствие одного или нескольких выражений. В данном случае необходимо использовать соответствующие поля инициализируемых значений в определении структуры. Допускается большее число полей, чем подразумевается выражениями и запятыми, и в этом случае заменяются только инициализации определения структуры для указанных полей. При отсутствии выражений и запятых (т. е. при наличии только ‹ ›) используются все значения инициализации из определения структуры.

Вместе со структурой PERSONNEL_DATA допускается применять операторы

```
EMPLOYEE_1  PERSONNEL_DATA  ‹'JR',,,35›
EMPLOYEE_2  PERSONNEL_DATA  ‹ ›
```

Эти операторы резервируют память и инициализируют значения в соответствии с рис. 3.60, *а*. Шаблон из определения структуры используется дважды: сначала для распределения пространства переменной EMPLOYEE_1, а затем для распределения пространства переменной EMPLOYEE_2. В первом случае символьная цепочка 'JR' заменяет цепочку 'XX' и 35 заменяет неопределенное значение в AGE, но для EMPLOYEE_2 все инициализации осуществляются в соответствии с определением структуры.

Спецификации инициализации разрешается модифицировать оператором DUP. Например, для структуры PERSONNEL_DATA оператор

```
EMPLOYEES  PERSONNAL_DATA  100 DUP(‹ ›)
```

реализует распределение, показанное на рис. 3.60, *б*. В этом примере структура дублируется 100 раз, образуя массив структурированных элементов.

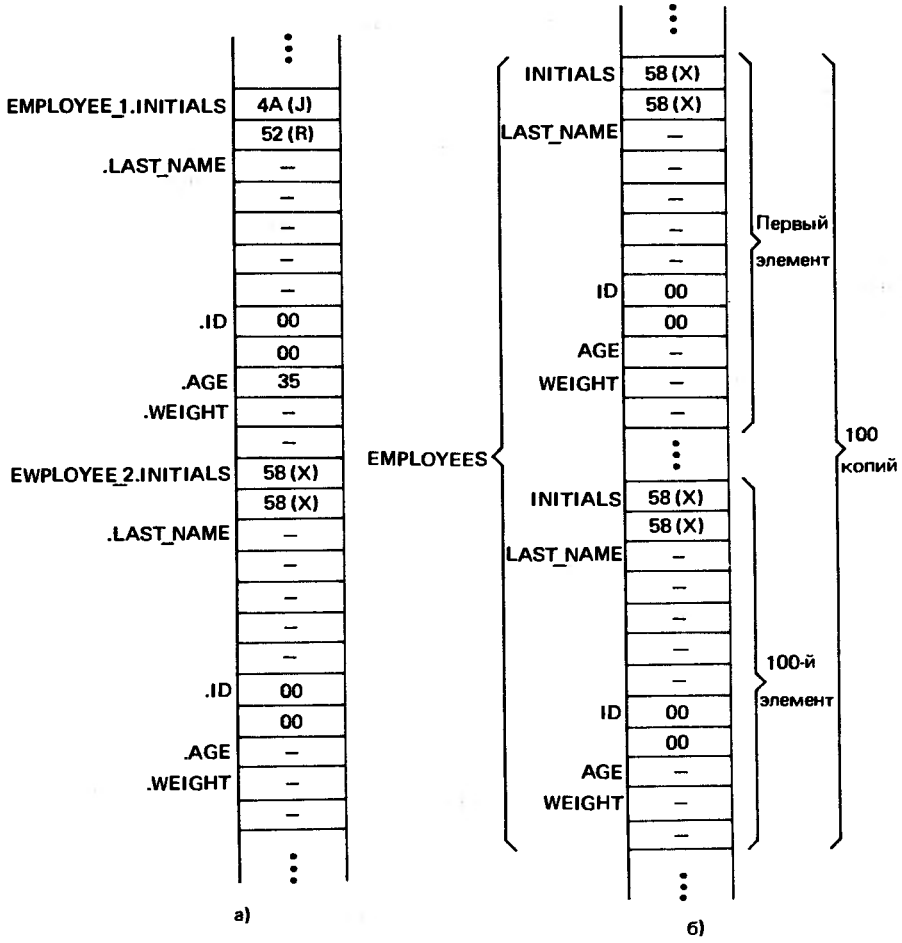


Рис. 3.60. Распределение и инициализация структур без дублирования (*а*) и с дублированием (*б*)

Команда обращается к ассоциированной со структурой переменной посредством следующей модификации операнда, определяющего обращение к памяти:

Обозначение смещения. Имя поля

Если (SI) = 2, третий элемент поля LAST_NAME в структуре EMPLOYEE_1, определяемой структурой PERSONNEL_DATA, пересылается в регистр AL командой

```
MOV AL,EMPLOYEE_1.LAST_NAME[SI]
```

Когда регистр BX содержит смещение EMPLOYEE_1, эту же команду можно переписать в виде

```
MOV AL,[BX].LAST_NAME[SI]
```

Команда

```
MOV AL,EMPLOYEES + 4*12.LAST_NAME[SI]
```

передает в регистр AL третий элемент поля LAST_NAME пятой копии в структуре EMPLOYEES (см. рис. 3.60, *б*).

3.10.3. ЗАПИСИ

Директива RECORD предназначена для определения двоичного набора в байте или слове. Ее применение аналогично директиве STRUC в том отношении, что RECORD только формирует шаблон, а резервирование и инициализацию памяти осуществляют другие операторы. Директива RECORD имеет следующий формат:

Имя записи RECORD Спецификация поля, . . . , Спецификация поля

Каждая спецификация поля имеет форму:

Имя поля: Длина = Начальное значение

причем начальное значение не обязательно. Например, директива

```
PATTERN RECORD OPCODE:5, MODE:3, OPR1:4 = 8, OPR2:4
```

разделяет слово на четыре поля и придает им имена OPCODE, MODE, OPR1 и OPR2. Длины полей составляют 5, 3, 4 и 4 бита соответственно. Оператор

```
INSTRUCTION PATTERN <,,,5>
```

резервирует одно слово, ассоциирует его с переменной INSTRUCTION и инициализирует OPR1 на 8, а OPR2 на 5, как показано на рис. 3.61. Оператор

```
INSTRUCTION_ARRAY PATTERN 100 DUP(< >)
```

резервирует 100 слов и ассоциирует первое из них с переменной INSTRUCTION_ARRAY. В данном случае третье поле в каждом слове инициализируется иа 8, а остальные поля не инициализируются.

Рис. 3.61. Типичное применение директивы RECORD для разделения слова

Имена полей могут появляться в командах сами по себе или с операторами MASK или WIDTH. Оператор MASK формирует операнд-маску с единицами в битах указанного поля и нулями в остальных битах. Оператор WIDTH образует операнд, равный длине указанного поля. Если имя поля появляется само по себе, операндом является число бит, необходимых для правого выравнивания поля. Последовательность команд

```
MOV  AX,INSTRUCTION
AND  AX,MASK OPR1
MOV  CL,OPR1
SHR  AX,CL
```

осуществляет передачу в регистр AX слова INSTRUCTION и выравнивает вправо поле OPR1.

3.10.4. НАЗНАЧЕНИЕ ИМЕН ВЫРАЖЕНИЯМ

Если выражение появляется в программе несколько раз, удобно присвоить ему имя, которым и пользоваться затем при обращении. Это позволит не только заменить коротким именем длинное выражение, но и воспользоваться содержательным именем, которое проще запомнить. Оператор, который назначает имя выражению, имеет формат:

Имя выражения EQU Выражение

Здесь именем выражения может быть любой допустимый идентификатор, а выражение может иметь формат любого допустимого операнда, быть любым выражением, вычисление которого дает константу (тогда имя выражения будет именем константы), или быть

```
                ДИРЕКТИВА                          ОПИСАНИЕ
                                                     .
CONSTANT    EQU     256             ПРИСВАИВАЕТ ЧИСЛУ 256 ИМЯ CONSTANT

DATA        EQU     HEIGHT+12       ПРИСВАИВАЕТ ПРЯМОМУ ОПЕРАНДУ HEIGHT+12
                                        ИМЯ DATA

FIELD       EQU     [BX].ETA[SI]    ПРИСВАИВАЕТ ИМЯ FIELD ОПЕРАНДУ
                                        [BX].ETA[SI], ОБРАЩАЮЩЕМУСЯ
                                        К СТРУКТУРЕ

ALPHA       EQU     7               ЭТА КОМБИНАЦИЯ ПРИДАЕТ 7 ИМЯ ALPHA,
BETA        EQU     ALPHA-2           7-2=5  - ИМЯ BETA, ПРЯМОМУ ОПЕРАНДУ
ADDR        EQU     VAR+BETA        VAR+BETA = VAR+5  - ИМЯ ADDR
```

Рис. 3.62. Примеры директивы EQU

любой допустимой мнемоникой. Несколько примеров выражений показаны на рис. 3.62. Обычно операторы EQU группируются в начале программы. Команда MOV в такой последовательности

```
INDXD_CHAR  EQU  CHAR_ARRAY [SI + 10]
                      .
                      .
                      .
            MOV  AL,INDXD_CHAR
                      .
                      .
                      .
```

эквивалентна команде

```
MOV  AL,CHAR_ARRAY [SI + 10]
```

Если выражение в операторе EQU содержит идентификатор (переменную, метку, имя константы и др.), то либо он должен представлять собой все выражение, либо он должен определяться в программе до оператора EQU. Оператор

```
AB  EQU  DATA_ONE
```

допустим независимо от того, где в программе определяется DATA_ONE, а оператор

```
AB EQU  DATA_ONE + 2
```

вызовет ошибку, если DATA_ONE ранее не определено. По этой причине рекомендуется помещать все определения данных и структур до операторов EQU. Однако оператор EQU следует указать до того, как применить его назначения; поэтому целесообразный порядок заключается в том, чтобы первыми разместить операторы определения данных, затем операторы EQU, а после них команды и связанные с ними директивы.

Другая важная причина использования имен вместо выражений заключается в более простой модификации программы. Предположим, например, что в программе несколь-

ко раз встречается команда ADD BX,6 и известно, что в некоторых применениях программы число 6 требуется заменить на другое число. Регистр BX может содержать адрес, а назначение команды заключается в инкременте этого адреса между полями в структуре. Если программу потребуется переписать для приспособления под другую длину, придется изменять команды. Но если поместить в начале программы оператор

```
NUM  EQU  6
```

и записать команды ADD в виде

```
ADD  BX,NUM
```

то значение инкремента можно изменить везде, например, на 8, изменив оператор EQU на

```
NUM  EQU  8
```

Такое применение оператора EQU особенно удобно при адресации портов ввода-вывода, так как при изменении конфигурации системы эти адреса могут изменяться.

3.10.5. ОПРЕДЕЛЕНИЯ СЕГМЕНТОВ

Напомним, что физический адрес формируется посредством сложения смещения и увеличенного в 16 раз сегментного адреса, содержащегося в сегментном регистре. Одна из задач ассемблера при трансляции команд в машинный код заключается в назначении смещений меткам и переменным. Ассемблер должен также передавать редактору связей (через объектные модули) всю информацию, необходимую ему для объединения различных сегментов и модулей в законченную программу. В ASM-86 предусмотрены несколько директив, сообщающих ассемблеру о том, как выполнять эти функции.

Чтобы назначить смещения переменным и меткам, ассемблер должен знать точную структуру каждого сегмента. Сегменты данных, дополнительных данных и стека обычно имеют следующую структуру:

```
Имя сегмента SEGMENT
                Директивы определения памяти,
                распределения и выравнивания
Имя сегмента ENDS
```

Структура сегмента кода имеет вид

```
Имя сегмента SEGMENT
                Команды и относящиеся
                к ним директивы
Имя сегмента ENDS
```

Именем сегмента может быть любой допустимый идентификатор. Директивы и команды, находящиеся между директивами SEGMENT и ENDS, считаются содержащимися в сегменте. Полная структура программы, имеющей два сегмента данных и один сегмент кода, приведена на рис. 3.63. (В ассемблере ASM-86 директива SEGMENT может содержать некоторые модификаторы, часть которых обсуждается в гл. 4.)

Кроме конструкции сегментов, ассемблер в ходе трансляции команд должен знать точное соответствие между сегментами и сегментными регистрами. Знание такого соответствия позволяет проконтролировать определенные виды ошибок и несовместимостей, например, определена ли переменная в нужном сегменте данных или сегменте стека. Назначения сегментов сегментным регистрам осуществляются директивами

```
ASSUME  Назначение, . . . ,Назначение
```

где каждое назначение имеет следующий формат:

```
Имя сегментного регистра:Имя сегмента
```

Оператор

```
ASSUME  CS:CODE_SEG,DS:SEG1,ES:SEG2
```

информирует ассемблер о необходимости считать, что сегментный адрес CODE_SEG находится в CS, SEG1 в DS и SEG2 в ES. Назначение для SS не сделано либо потому, что стек не используется, либо потому, что назначение для SS находится в отдельном операторе ASSUME. Важно подчеркнуть, что директива ASSUME не загружает сегментные адреса в соответствующие сегментные регистры. Для всех сегментных регистров, кроме CS, который обычно загружается командой межсегментного перехода при инициировании сегмента кода, загрузка должна производиться явными передачами, обычно командами MOV. Директива ASSUME дает как бы "обещание" ассемблеру и должна сопровождаться командами MOV, которые выполняют это "обещание".

В соответствии со структурой программы на рис. 3.63 сегмент кода обычно начинается следующим образом:

```
CODE_SEG    SEGMENT
 ASSUME  CS:CODE_SEG, DS:DATA_SEG1,ES:DATA_SEG2
       START:  MOV     AX,DATA_SEG1
               MOV     DS,AX
               MOV     AX,DATA_SEG2
               MOV     ES,AX
                 .
                 .
                 .
CODE_SEG    ENDS

        DATA_SEG1       SEGMENT
                 .
                 .  } ДИРЕКТИВЫ
                 .
        DATA_SEG1       ENDS
        DATA_SEG2       SEGMENT
                 .
                 .  } ДИРЕКТИВЫ
                 .
        DATA_SEG2       ENDS
        CODE_SEG        SEGMENT
           START:
                         .
                         .  } КОМАНДЫ И ДИРЕКТИВЫ
                         .
        CODE_SEG        ENDS
                        END  START
```

Рис. 3.63. Типичная структура программы

Поскольку команда MOV не может передавать непосредственный операнд в сегментный регистр, для каждого регистра требуются две команды. Имена DATA_SEG1 и DATA_SEG2 являются именами сегментов, а не переменными, поэтому ассемблер транслирует первую и третью команды MOV в команды, у которых операнды-источники являются непосредственными значениями. Операнды-источники должны быть сегмент-

ными адресами, но так как размещение сегментов в памяти осуществляет редактор связей, секцин данных этих команд фактически формирует редактор связей.

Один и тот же сегмент допускается назначать двум сегментным регистрам, например директива

```
ASSUME CS:CODE_SEG, DS:CODE_SEG
```

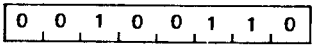
назначает один и тот же сегмент регистрам CS и DS. Операторы ASSUME можно использовать также для переназначения сегментных регистров. Если в приведенном выше примере позднее в программе появляются операторы

```
ASSUME  DS:DATA_SEG3
    MOV AX,DATA_SEG3
    MOV DS,AX
```

ассемблер при трансляции последующих операторов будет использовать DATA_SEG3 вместо DATA_SEG1, но назначения для остальных сегментных регистров останутся неизменными.

Когда ассемблер встречает в операнде команды переменную, он обычно предполагает сегментный регистр по умолчанию в соответствии с режимом адресации, как показано на рис. 2.15. Однако, если регистром по умолчанию принимается DS, а переменная не находится в сегменте, назначенном регистру DS, ассемблер просматривает сегменты, назначенные регистром CS, ES и SS. Если переменная найдена, ассемблер вводит перед командой, содержащей переменную, префикс замены сегмента на CS, ES или SS. При выполнении команды однобайтный префикс заставит ее использовать вместо регистра DS регистры CS, ES или SS.

Сегментные регистры по умолчанию разрешается заменять явно, помещая перед переменной их имена с последующим двоеточием. Команда ADD AX,ES:ALT заставит ассемблер ввести перед машинным кодом команды ADD префикс замены сегмента



В этом случае ассемблеру не нужно вначале отыскивать сегмент, назначенный регистру DS. Если имя сегментного регистра явно не появляется во всех операндах, которые используют его, оно должно фигурировать в операторе ASSUME, в противном случае возникает ошибка. Вместо имени сегментного регистра допускается указывать имя сегмента, но при условии, что оператор ASSUME связал эти два имени, например пара

```
ASSUME ES:EXTRA_SEG
        .
        .
        .
    ADD AX,EXTRA_SEG:ALT
```

эквивалентна команде ADD AX,ES:ALT. Замена сегментного регистра должна определяться явно, когда определяемый позднее в программе операнд находится в сегменте, не адресуемом сегментным регистром по умолчанию (см. § 3.11).

3.10.6. ОКОНЧАНИЕ ПРОГРАММЫ

В программах на языках высокого уровня их окончание отмечается оператором END. Аналогичным образом конец ассемблерной программы указывает директива END со следующим форматом:

```
END  Метка
```

В программной структуре метка помещается только в конце основной программы, хотя все отдельно ассемблируемые программные модули должны завершаться директивой END. Эта же метка должна находиться у первой выполняемой команды, т. е. в точке, с которой начинается программа. Другими словами, ассоциируемый с этой меткой адрес является тем адресом, по которому переходит загрузчик после того как программа загружена и готова к исполнению.

На рис. 3.64 показана законченная, хотя и простая, программа. Она суммирует содержимое OPER1 и OPER2 и помещает в RESULT абсолютное значение суммы. Программа

```
DATA_SEG        SEGMENT
    OPER1           DW      12
    OPER2           DW      230
    RESULT          DW      ?
DATA_SEG        ENDS
CODE_SEG        SEGMENT
  ASSUME      CS:CODE_SEG, DS:DATA_SEG
    START:          MOV     AX,DATA_SEG
                    MOV     DS,AX
                    MOV     AX,OPER1
                    ADD     AX,OPER2
                    JGE     STORE
                    NEG     AX
    STORE:          MOV     RESULT,AX
                    HLT
CODE_SEG        ENDS
                    END   START
```

Рис. 3.64. Законченная программа

бесполезна, так как в ней отсутствуют ввод и вывод, но она иллюстрирует структуру законченных одномодульных программ и применение некоторых директив.

На рис. 3.65 показано, как структурировать эту же программу, когда сегменты данных и кода совмещены. Вид этой программы более похож на программы тех компью-

```
EXAMPLE         SEGMENT
    OPER1           DW      12
    OPER2           DW      230
    RESULT          DW      ?
  ASSUME      CS:EXAMPLE, DS:EXAMPLE
    START:          MOV     AX,CS
                    MOV     DS,AX
                    MOV     AX,OPER1
                    ADD     AX,OPER2
                    JGE     STORE
                    NEG     AX
    STORE:          MOV     RESULT,AX
                    HLT
EXAMPLE         ENDS
                    END   START
```

Рис. 3.65. Законченная программа с совмещенными сегментами данных и кода

теров, в которых отсутствуют сегментные регистры. Для микропроцессора 8086 предпочтительнее структура, приведенная на рис. 3.64, так как она обеспечивает большую гибкость в распределении памяти (т. е. сегменты кода и данных назначаются независимо).

3.10.7. ДИРЕКТИВЫ ВЫРАВНИВАНИЯ

В языке ассемблера предусмотрены две директивы, предназначенные для выравнивания. Директива

EVEN

превращает адрес следующего байта в четное число. Отметим, что микропроцессор 8086 осуществляет обращения к словам быстрее, если они начинаются с четных адресов. Сле-

довательно, перед резервированием массива слов целесообразно поместить директиву EVEN, например:

```
DATA_SEG        SEGMENT
                  .
                  .
                  .
                  EVEN
WORD_ARR          DW      100 DUP(?)
                  .
                  .
                  .
DATA_SEG        ENDS
```

Однако в микропроцессоре 8088, в котором обращение к операнду-слову требует два цикла шины независимо от адреса операнда, выравнивание по четным адресам не дает преимуществ.

Директива

ORG Выражение-константа

заставляет следующий байт быть n-м байтом в сегменте, где n – значение выражения-константы. Например, код

```
VECTORS     SEGMENT
                ORG     10
VECT1           DW      47A5H
                ORG     20
VECT2           DW      0C596H
                  .
                  .
                  .
VECTORS     ENDS
```

ассоциирует VECT1 и VECT2 с десятым и двадцатым байтами сегмента VECTORS.

3.10.8. АТРИБУТНЫЕ ОПЕРАТОРЫ, ВОЗВРАЩАЮЩИЕ ЗНАЧЕНИЕ

В языках ассемблера обычно предусматривается несколько операторов, которые не нужны для правильного ассемблирования программы, но которые облегчают действия программиста, обеспечивая гибкость, улучшая читабельность и заставляя сам ассемблер подставлять некоторые числа. В этот класс попадают следующие атрибутные операторы, возвращающие значение: LENGTH, SIZE, OFFSET, SEG и TYPE. Все эти операторы возвращают значение, относящееся к находящимся в операндах выражениям.

Оператор LENGTH возвращает число единиц, назначенных переменной. Оператор

```
FEES DW 100 DUP(0)
```

ассоциирует с переменной FEES сто слов, а оператор

```
MOV CX,LENGTH FEES
```

эквивалентен команде MOV CX,100. Целесообразнее применять оператор LENGTH, так как команда

```
MOV CX,LENGTH FEES
```

будет правильной, даже если директиву DW изменить на резервирование 150 слов, а "стандартную" машинную команду придется заменить на MOV CX,150.

Оператор SIZE аналогичен оператору LENGTH, но вместо числа единиц он возвращает число байт. Предполагая наличие предыдущей директивы DW, команда

```
MOV CX,SIZE FEES
```

вызывает загрузку в регистр CX числа 200.

Оператор OFFSET возвращает значение смещения метки или переменной. Команда

```
MOV BX,OFFSET OPER_ONE
```

заставляет смещение OPER_ONE ассемблироваться как непосредственный операнд, а во время выполнения смещение OPER_ONE загружается в регистр BX. Данная команда эквивалентна команде LEA BX,OPER_ONE.

Аналогично оператор SEG вызывает введение как непосредственного операнда сегментного адреса переменной или метки (хотя фактическое введение осуществляет редактор связей). Если DATA_SEG ассоциирован с блоком памяти, начинающимся по адресу 05000, и OPER1 находится в DATA_SEG, то команда

```
MOV BX,SEG OPER1
```

загружает в BX значение 0500.

Оператор TYPE применяется в основном с именами переменных и структур и возвращает число байт, ассоциируемых с переменной или структурой. Команда

```
ADD SI,TYPE ARRAY
```

прибавляет единицу к регистру SI, если ARRAY определен в операторе DB, два, если ARRAY определен в операторе DW, четыре, если ARRAY определен в операторе DD, и число байт в структуре, если ARRAY является именем структуры. Оператор TYPE редко используется с метками, но, когда это сделано, он возвращает −1 для метки NEAR и −2 для метки FAR.

3.11. ПРОЦЕСС АССЕМБЛИРОВАНИЯ

Мы рассмотрим процесс ассемблирования, чтобы пояснить структуру ассемблерных программ и показать необходимость некоторых правил. Как видно из рис. 3.66, входом для ассемблера является исходный модуль, а выходом – объектный модуль и листинг. Исходный модуль обычно представлен в виде файла на магнитных ленте или диске; файл создается вводом с перфокарт (на каждой карте находится один оператор) или редактором текста посредством печати по одному оператору в строке. Объектный модуль, содержащий машинный код и информацию для объединения модуля с другими объектными модулями, запоминается в виде файла.

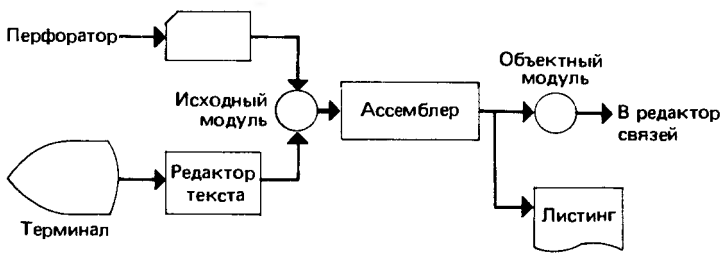


Рис. 3.66. Вход и выход ассемблера

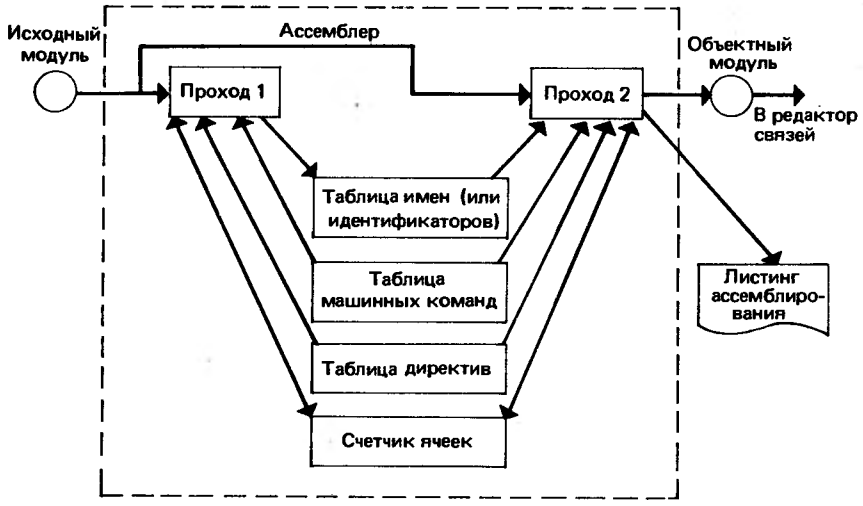


Рис. 3.67. Двухпроходовый ассемблер

На рис. 3.67 показана внутренняя структура ассемблера. (Мы рассматриваем гипотетический и значительно упрощенный ассемблер, чтобы показать существо его основных действий.) Рассматриваемый ассемблер, называемый двухпроходовым, как и большинство ассемблеров, сканирует исходный код два раза. Цель первого прохода – получить информацию о местоположении идентификаторов, а второго – генерировать машинный код. Для локализации идентификаторов в ассемблере предусмотрена переменная, называемая *счетчиком ячеек* (*адресов*). При сканировании программы производится инкремент счетчика ячеек на число байт, занимаемых операторами. Когда программа переходит из одного сегмента в другой, счетчик ячеек сбрасывается в 0. Счетчик ячеек можно считать указателем, который в ходе ассемблирования динамически фиксирует относительные позиции (т. е. смещения) внутри каждого сегмента. На первом проходе ассемблер с помощью счетчика ячеек строит таблицу, называемую *таблицей имен* (или *таблицей идентификаторов*), которая позволяет на втором проходе использовать смещения идентификаторов для генерирования адресов операндов. Покажем это на примере:

```
                                          СЧЕТЧИК ЯЧЕЕК   ДЛИНА В БАЙТАХ
CODE_SEG        SEGMENT                       0               0
  ASSUME     CS:CODE_SEG, DS:DATA_SEG         0               0
      START: MOV     AX,DATA_SEG              0               3
             MOV     DS,AX                    3               2
             MOV     CX,COUNT                 5               4
    REPEAT:  DEC     CX                       9               1
             .                               10               .
             .                                .               .
             .                                .               .
```

Как обычно, первоначально счетчик ячеек содержит 0. При сканировании первых двух операторов, которые являются директивами и не занимают места в машинном коде, счетчик ячеек не изменяется. Следующие четыре оператора являются командами, которые транслируются в машинные команды длиной 3, 2, 4 и 1 байт соответственно. Следовательно, они вызывают инкремент счетчика ячеек, значениями которого становятся

3, 5, 9, 10 и т. д. В следующем фрагменте показан инкремент счетчика ячеек в соответствии с числом байт, резервируемых каждой директивой:

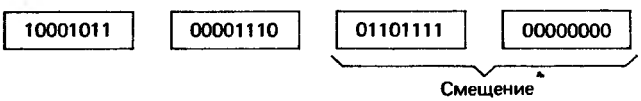
```
                                        СЧЕТЧИК     ЧИСЛО
                                        ЯЧЕЕК       РАСПРЕДЕЛЯЕМЫХ
                                                    БАЙТ
DATA_SEG    SEGMENT                     0           0
            ORG   10                    0           10
NUM         DB    -29                   10          1
ARRAY       DB    100 DUP(?)            11          100
COUNT       DW    5                     111         2
MASKS       DB    82H,04H,2AH           113         3
            .                           116         .
            .                           .           .
            .                           .           .
```

Для приведенных фрагментов на первом проходе с помощью счетчика ячеек формируется следующая таблица имен:

```
                        СЕГМЕНТ, В КОТОРОМ
ИМЯ         СМЕЩЕНИЕ    ИМЯ ОПРЕДЕЛЕНО      ТИП

DATA_SEG    00H                             СЕГМЕНТ
NUM         0AH         DATA_SEG            ПЕРЕМЕННАЯ (БАЙТ)
ARRAY       0BH         DATA_SEG            ПЕРЕМЕННАЯ (БАЙТ)
COUNT       6FH         DATA_SEG            ПЕРЕМЕННАЯ (СЛОВО)
MASKS       71H         DATA_SEG            ПЕРЕМЕННАЯ (БАЙТ)
  .
  .
  .
CODE_SEG    00H                             СЕГМЕНТ
START       00H         CODE_SEG            МЕТКА (БЛИЗКАЯ)
REPEAT      09H         CODE_SEG            МЕТКА (БЛИЗКАЯ)
  .
  .
  .
```

Элемент таблицы имен ассоциируется с каждым идентификатором и содержит тип и имя сегмента, в котором определяется идентификатор. На втором проходе эта информация используется при генерировании команд, операнды которых содержат идентификаторы. Например, при построении машинного кода команды MOV CX,COUNT ассемблер контролирует соответствие типов источника и получателя и доступность COUNT через DS. Затем ассемблер находит, что смещение COUNT равно $006F (= 111_{10})$, и формирует машинную команду



Ассемблер также имеет две таблицы, называемые *таблицами фиксированных имен*, которые содержат всю необходимую ассемблеру информацию о командах и директивах. В этих таблицах находятся мнемоники, коды операций и форматы, информация о длине, необходимая для инкремента счетчика ячеек, и другие сведения.

Основные функции первого и второго проходов ассемблера показаны на рис. 3.68 и 3.69. Когда в самом левом поле оператора встречается метка или переменная, говорят, что она определяется. В этот момент метка или переменная ассоциируется со смещением, для чего в таблицу имен помещается текущее содержимое счетчика ячеек, символическое представление метки или переменной и другие атрибуты. Если одна и та же метка или переменная определяются в исходном модуле дважды, фиксируется ошибка. Ошибка также возникает, если мнемоника команды или директивы не находится ни в

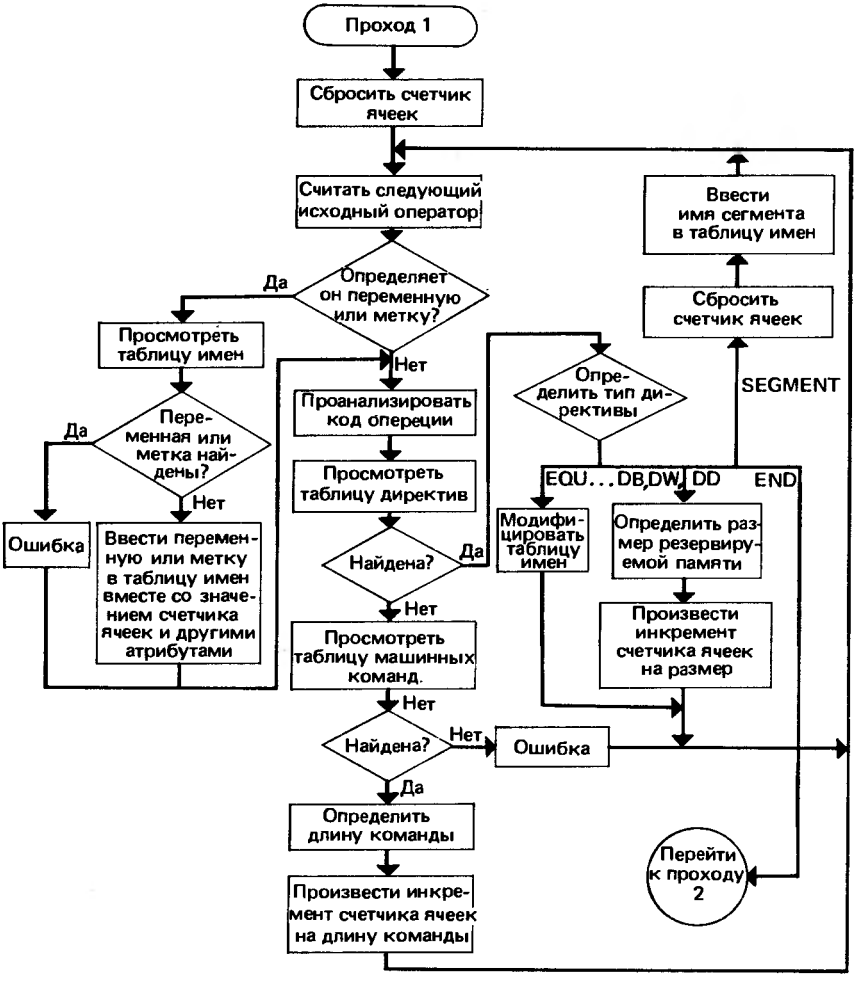
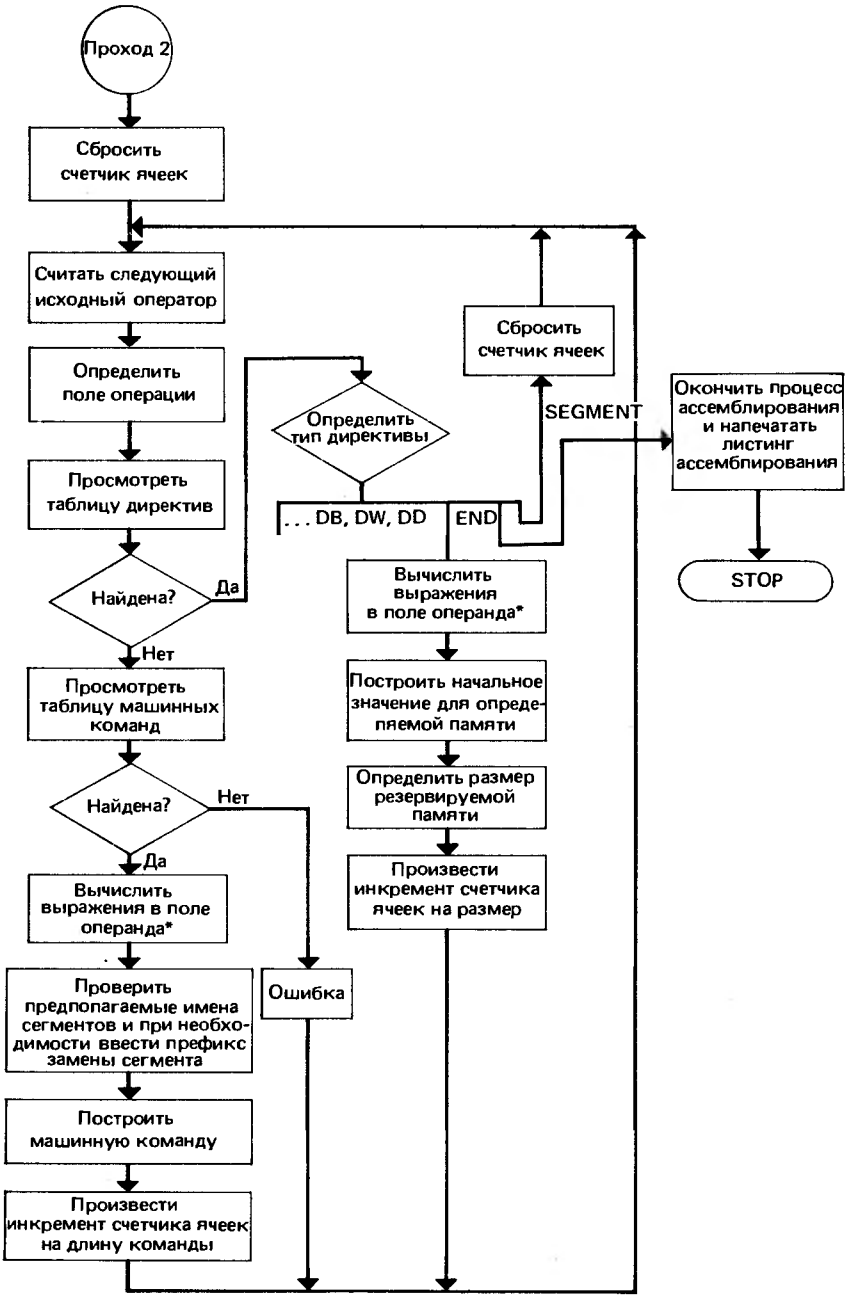


Рис. 3.68. Основные действия на первом проходе

одной из таблиц фиксированных имен. Когда на первом проходе встречается директива END, он заканчивается и начинается второй проход.

Кроме ассемблирования машинных команд, на втором проходе необходимо вводить константы инициализации из директив определения данных и подготавливать требуемую редактору связей информацию. Поскольку операнды команд и директив могут содержать выражения, вычисление смещения оказывается более сложным, чем поиск элемента в таблице имен. Такие вычисления выражений также реализуются на втором проходе. Когда на втором проходе встречается директива END, объектный модуль и листинг выводятся на соответствующие периферийные устройства и процесс ассемблирования заканчивается.



* Требует обращения к таблице имен, если в выражении есть метка или переменная

Рис. 3.69. Основные действия на втором проходе

Если в момент появления в операнде переменной или метки она уже определена (т.е. была в самом левом поле одного из предыдущих операторов), это появление называют *обращением назад*; в противном случае имеет место *обращение вперед*. Обрабатывать обращения вперед ассемблеру несколько сложнее. Причина заключается в том, что длина команды может зависеть от атрибутов операнда и, если атрибуты не указаны явно, ассемблеру придется предполагать, на какое значение выполнять инкремент счетчика ячеек. Неправильное предположение либо вызовет ошибки ассемблирования, либо заставит второй проход компенсировать эти ошибки с ассемблированием неоптимального кода. Допускается явное указание атрибутов с помощью атрибутных операторов и префиксов сегментов, например SHORT, PTR, ES: и др.

Рассмотрим, к примеру, команду JMP EXIT. Если EXIT является обращением вперед, в момент встречи команды ассемблер не может узнать, какой атрибут имеет переход: короткий, близкий или далекий. По умолчанию ассемблер считает, что EXIT является близкой меткой и резервирует для команды 3 байта. С другой стороны, нет никаких сомнений в том, что команда

```
JMP FAR PTR EXIT
```

имеет длину 5 байт.

Второй пример показывает другую, но тоже неоднозначную ситуацию. Рассмотрим команду MOV AX,OPERAND и предположим, что OPERAND является обращением вперед к сегменту, отличающемуся от сегмента DS (например, OPERAND определен в сегменте, адресуемом SS). Из-за отсутствия дополнительных указаний на первом проходе ассемблер предполагает, что должен использоваться сегментный регистр DS, и не учитывает необходимый префикс замены сегмента. Такое предположение вызовет ошибку на втором проходе. Однако команда MOV AX,SS:OPERAND или комбинация

```
ASSUME    SS:DATA_SEG2
.
.
.
MOV AX,DATA_SEG2:OPERAND
```

явно информирует ассемблер о необходимости введения префикса замены сегмента.

Чтобы избежать проблем и получающихся ошибок, связанных с обращениями вперед, рекомендуются следующие меры:

1. Применять атрибутные операторы для явного определения типов операндов во всех командах JMP.

2. Помещать все директивы определения данных перед командами.

3. Все операторы EQU помещать между директивами, относящимися к данным, и командами. Если выражение в одном операторе EQU обращается к имени в другом операторе EQU, обращение должно быть обращением назад.

Правило 2 подразумевает, что в текущем исходном модуле все относящиеся к данным сегменты должны находиться над сегментами кода. Правило 3 позволяет снять сложности обращений вперед при расшифровке операторов EQU.

Приведенные рекомендации соответствуют следующему правилу языка Фортран и других языков высокого уровня: операторы распределения памяти, инициализации и определения параметров должны предшествовать испопняемым операторам.

В ассемблере ASM-86, как и в большинстве других ассемблеров, допускаются значения по умолчанию, в которых по возможности учтены рассмотренные вопросы. Однако чрезмерное употребление значений по умолчанию часто ведет к плохому программированию и к программам, в которых трудно разбираться. Поэтому лучше всего не пользоваться теми значениями по умолчанию, которые не могут быть поняты любым другим

человеком, пользующимся программой. Мы поясним только те значения по умолчанию, которые считаем полезными.

Многие ассемблеры разработаны с возможностью восприятия имени, которое позволяет программисту непосредственно обращаться к счетчику ячеек. В ассемблере ASM-86 таким именем является значок "$". Команда JNE $ + 6 заставит ассемблер вычислить смещение адреса перехода посредством прибавления 6 к текущему содержимому счетчика ячеек. Хотя имя счетчика ячеек применяется редко, его наличие заслуживает упоминания. Одна из трудностей использования таких имен связана с тем, что при изменении кода между командой JNE и точкой через 4 байта от этой команды адрес перехода, вычисляемый из $ + 6, окажется неправильным. Имя счетчика ячеек удобнее применять в директивах вида

```
ORG OFFSET $+100
```

Эта директива служит только для инкремента счетчика ячеек на 100. Даже на втором проходе ничего не ассемблируется в 100 байт, которые пропускаются этой директивой. Приведенная выше директива ORG смещает ассемблирование точно так же, как и директива

```
DB 100 DUP(?)
```

Кроме объектного модуля ассемблер выводит листинг, содержащий информацию о программе и ее трансляции. В зависимости от вида приказа вызова ассемблера листинг содержит либо только список ошибок, обнаруженных при ассемблировании, либо подробный листинг программы, либо что-то промежуточное между этими крайними случаями. Полный листинг с машинным кодом обычно содержит листинг программы, сообщения об ошибках и таблицу имен перекрестных обращений.

```
LOC  OBJ                  LINE     SOURCE

----                         1     DATA_SEG        SEGMENT
0000 0C00                    2         OPER1           DW        12
0002 E600                    3         OPER2           DW        230
0004 ????                    4         RESULT          DW        ?
----                         5     DATA_SEG        ENDS
-----                        6     CODE_SEG        SEGMENT
                             7       ASSUME    CS:CODE_SEG,DS:DATA_SEG
0000 B8----          R       8         START:          MOV       AX,DATA_SEG
0003 8ED8                    9                         MOV       DS,AX
0005 A10000                 10                         MOV       AX,OPER1
0008 03060200               11                         ADD       AX,OPER2
000C 7D02                   12                         JGE       STORE
000E F7D8                   13                         NEG       AX
0010 A30400                 14         STORE:          MOV       RESULT,AX
0013 F4                     15                         HLT
----                        16     CODE_SEG        ENDS
  0000                      17                         END       START


XREF SYMBOL TABLE LISTING
---- ------ ----- --------


NAME        TYPE     VALUE   ATTRIBUTES, XREFS

??SEG . .  SEGMENT           SIZE=0000H PARA PUBLIC
CODE_SEG.  SEGMENT           SIZE=0014H PARA      6# 7 16
DATA_SEG.  SEGMENT           SIZE=0006H PARA      1# 5 7 8
OPER1 . .  V WORD    0000H   DATA_SEG   2# 10
OPER2 . .  V WORD    0002H   DATA_SEG   3# 11
RESULT. .  V WORD    0004H   DATA_SEG   4# 14
START . .  L NEAR    0000H   CODE_SEG   8# 17
STORE . .  L NEAR    0010H   CODE_SEG   12 14#
```

Рис. 3.70. Пример ассемблерного листинга

Типичный листинг приведен на рис. 3.70. Первый столбец в программной части листинга дает значение счетчика ячеек непосредственно перед ассемблированием соответствующего оператора. Второй столбец показывает машинный код, в который ассемблируется оператор, третий столбец – просто номер строки в исходном коде, а остальная часть строки – исходный код в том виде, как его встретил ассемблер. Если обнаруживается ошибка, в следующей строке выводятся идентификационный номер ошибки и сообщение.

Отметим, что команда MOV в строке 8 построена не полностью. Как показано в разделе 3.10.5, если второй операнд оказывается именем сегмента, он представляет собой сегментный адрес, а не содержимое ячейки памяти. Поскольку начальные адреса сегментов не известны до связывания программы, ассемблер не может полностью построить эту команду MOV. Буква "R" в данной строке означает "переместимый" и показывает, что адрес должен определить редактор связей. Термин "переместимый" поясняется в гл. 4 при обсуждении связывания программы.

Таблица перекрестных обращений суммирует всю информацию, относящуюся к идентификаторам в программе. Для каждого имени сегмента эта таблица показывает размер сегмента (как 16-ричное число) и номера тех строк, в которых появляется имя. Номер строки директивы сегмента, содержащей имя, отмечается знаком "#". Переменные и метки сопровождаются буквами, обозначающими переменную (V) или метку (L), атрибутом типа, смещением, именем сегмента, в котором они определяются, и номерами строк, в которых они появляются. Строки, в которых они определяются, отмечены значком "#". Например, OPER2 – это переменная с типом слова, которая определяется в сегменте DATA_SEG. Ее смещение в DATA_SEG равно 0002. Она появляется в строках 3 и 11 листинга, а определяется в строке 3. Аналогично, STORE является близкой меткой, которая определяется в сегменте CODE_SEG и имеет в нем смещение 0010H. Она появляется в строках 12 и 14, а определяется в строке 14. Читателю рекомендуется внимательно проверить все элементы в таблице имен и машинный код листинга.

3.12. ТРАНСЛЯЦИЯ АССЕМБЛЕРНЫХ КОМАНД

Трансляция ассемблерных команд в машинные в большинстве случаев оказывается довольно простой. Все команды микропроцессора 8086 состоят из 1 или 2 байт кода операции и определения режима адресации, к которым добавляются 0 – 4 байта непосредственного операнда,смещения или сегментного адреса. Число добавляемых байт зависит от режима адресации. Если режим адресации требует смещения и непосредственного операнда, то первым следует смещение, а при наличии смещения и сегментного адреса первым размещается смещение.

Форматы машинного кода всех команд микропроцессоров 8086/8088 приведены на рис. 3.71. Для удобства здесь же показаны адреса регистров, режимы адресации и др. Все двухоперандные команды (MOV, ADD, ADC, SUB, SBB, CMP, AND, OR, XOR и TEST), кроме одной, в которых операндом-источником является непосредственное значение, а другим операндом служат регистры AX или AL, имеют короткий и длинный форматы. В коротком формате AX или AL подразумеваются кодом операции. Кроме того, команда MOV имеет короткий формат для передачи непосредственного значения в любой регистр. Команда XCHG имеет короткий формат для обмена между регистром и AX или AL. Несколько однооперандных команд (INC, DEC, PUSH и POP) имеют короткие форматы, когда операндом является регистр.

```
AL - 8-БИТНЫЙ АККУМУЛЯТОР
AX - 16-БИТНЫЙ АККУМУЛЯТОР
CX - РЕГИСТР СЧЕТЧИК
DS - СЕГМЕНТ ДАННЫХ
ES - ДОПОЛНИТЕЛЬНЫЙ СЕГМЕНТ
ВЫШЕ/НИЖЕ ОТНОСИТСЯ К БЕЗЗНАКОВОМУ ЗНАЧЕНИЮ
БОЛЬШЕ - БОЛЕЕ ПОЛОЖИТЕЛЬНОЕ
МЕНЬШЕ - МЕНЕЕ ПОЛОЖИТЕЛЬНОЕ
        (БОЛЕЕ ОТРИЦАТЕЛЬНОЕ) ЗНАКОВОЕ ЗНАЧЕНИЕ
ЕСЛИ D=1, ТО "В" РЕГИСТР;  ЕСЛИ D=0, ТО "ИЗ" РЕГИСТРА
ЕСЛИ W=1, КОМАНДА ОПЕРИРУЕТ СЛОВОМ;
          ЕСЛИ W=0, КОМАНДА ОПЕРИРУЕТ БАЙТОМ

ЕСЛИ MOD=11,  ТО R/M ИНТЕРПРЕТИРУЕТСЯ ТАК ЖЕ, КАК ПОЛЕ REG
ЕСЛИ MOD=00,  ТО DISP=0 *, А МЛАДШИЙ И СТАРШИЙ БАЙТЫ СМЕЩЕНИЯ ОТСУТСТВУЮТ
ЕСЛИ MOD=01,  ТО DISP РАВНО МЛАДШЕМУ БАЙТУ СМЕЩЕНИЯ,
              РАСШИРЕННОМУ СО ЗНАКОМ ДО 16 БИТ,
              А СТАРШИЙ БАЙТ СМЕЩЕНИЯ ОТСУТСТВУЕТ
ЕСЛИ MOD=10,  ТО DISP РАВНО СТАРШЕМУ И МЛАДШЕМУ БАЙТАМ СМЕЩЕНИЯ
ЕСЛИ R/M=000, ТО EA=(BX)+(SI)+DISP
ЕСЛИ R/M=001, ТО EA=(BX)+(DI)+DISP
ЕСЛИ R/M=010, ТО EA=(BP)+(SI)+DISP
ЕСЛИ R/M=011, ТО EA=(SP)+(DI)+DISP
ЕСЛИ R/M=100, ТО EA=(SI)+DISP
ЕСЛИ R/M=101, ТО EA=(DI)+DISP
ЕСЛИ R/M=110, ТО EA=(BP)+DISP *
ЕСЛИ R/M=111, ТО EA=(BX)+DISP
DISP НАХОДИТСЯ ПОСЛЕ ВТОРОГО БАЙТА КОМАНДЫ (ПЕРЕД ДАННЫМИ,
               ЕСЛИ ОНИ ТРЕБУЮТСЯ В КОМАНДЕ)

* ЗА ИСКЛЮЧЕНИЕМ СЛУЧАЯ MOD=00 И R/M=110, КОГДА EA РАВЕН СТАРШЕМУ
  И МЛАДШЕМУ БАЙТАМ СМЕЩЕНИЯ
```

```
ЕСЛИ S:W=01, ТО ОПЕРАНД ОБРАЗУЮТ 16 БИТ НЕПОСРЕДСТВЕННЫХ ДАННЫХ
ЕСЛИ S:W=11, ТО БАЙТ НЕПОСРЕДСТВЕННЫХ ДАННЫХ РАСШИРЯЕТСЯ СО ЗНАКОМ
             ДЛЯ ОБРАЗОВАНИЯ 16-БИТНОГО ОПЕРАНДА
ЕСЛИ V=0, ТО "СЧЕТЧИК" РАВЕН 1;
          ЕСЛИ V=1, ТО "СЧЕТЧИК" НАХОДИТСЯ В РЕГИСТРЕ CL
X - НЕОПРЕДЕЛЕННОСТЬ
Z - ПРИМЕНЯЕТСЯ В ЦЕПОЧЕЧНЫХ ПРИМИТИВАХ ДЛЯ СРАВНЕНИЯ С ФЛАЖКОМ ZF

        ПРЕФИКС ЗАМЕНЫ СЕГМЕНТА
          +---------------+
          !0 0 1 REG 1 1 0!
          +---------------+
```

REG ИНТЕРПРЕТИРУЕТСЯ В СООТВЕТСТВИИ СО СЛЕДУЮЩЕЙ ТАБЛИЦЕЙ

16 БИТ (W=1)		8 БИТ (W=0)		СЕГМЕНТ	
000	AX	000	AL	00	ES
001	CX	001	CL	01	CS
010	DX	010	DL	10	SS
011	BX	011	BL	11	DS
100	SP	100	AH		
101	BP	101	CH		
110	SI	110	DH		
111	DI	111	BH		

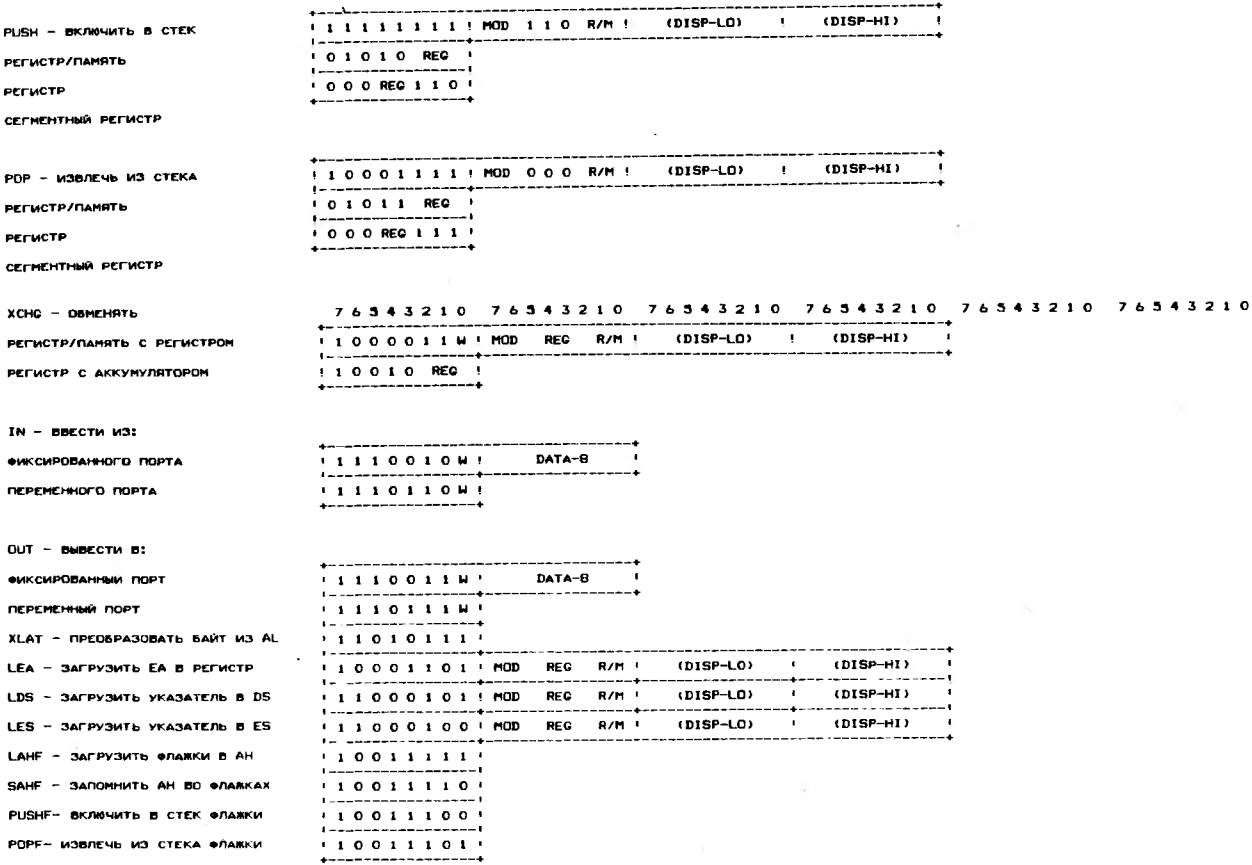
В КОМАНДАХ С ОБРАЩЕНИЕМ К РЕГИСТРУ ФЛАЖКОВ, КАК К 16-БИТНОМУ ОБ'ЕКТУ, РЕГИСТР ОБОЗНАЧАЕТСЯ FLAGS

ФЛАЖКИ (FLAGS) = X:X:X:(DF):(DF):(IF):(TF):(SF):(ZF):X:(AF):X:(PF):X:(CF)

КОДИРОВАНИЕ КОМАНД МИКРОПРОЦЕССОРОВ 8086/8088

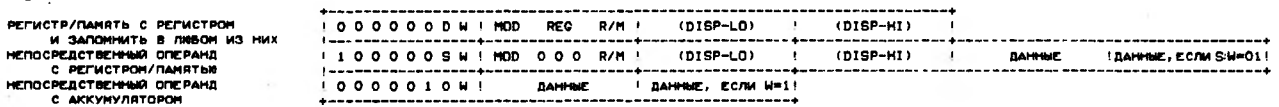
```
КОМАНДЫ ПЕРЕДАЧ ДАННЫХ                 7 6 5 4 3 2 1 0   7 6 5 4 3 2 1 0   7 6 5 4 3 2 1 0   7 6 5 4 3 2 1 0   7 6 5 4 3 2 1 0   7 6 5 4 3 2 1 0
                                       +-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+
MOV - ПЕРЕДАТЬ                         ! 1 0 0 0 1 0 D W ! MOD  REG  R/M !    (DISP-LO)    !    (DISP-HI)    !                                       !
                                       +-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+
РЕГИСТР/ПАМЯТЬ                         ! 1 1 0 0 0 1 1 W ! MOD 0 0 0 R/M !    (DISP-LO)    !    (DISP-HI)    !       ДАННЫЕ      ! ДАННЫЕ, ЕСЛИ W=1!
     В/ИЗ РЕГИСТРА                     +-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+
НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД               ! 1 0 1 1 W  REG  !       ДАННЫЕ    ! ДАННЫЕ, ЕСЛИ W=1!
     В РЕГИСТР/ПАМЯТЬ                  +-----------------+-----------------+-----------------+
НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД               ! 1 0 1 0 0 0 0 W ! МЛ. БАЙТ АДРЕСА ! СТ. БАЙТ АДРЕСА !
     В РЕГИСТР                         +-----------------+-----------------+-----------------+
ПАМЯТЬ В АККУМУЛЯТОР                   ! 1 0 1 0 0 0 1 W ! МЛ. БАЙТ АДРЕСА ! СТ. БАЙТ АДРЕСА !
                                       +-----------------+-----------------+-----------------+
АККУМУЛЯТОР В ПАМЯТЬ                   ! 1 0 0 0 1 1 1 0 ! MOD 0 SR  R/M !    (DISP-LO)    !    (DISP-HI)    !
                                       +-----------------+-----------------+-----------------+-----------------+
РЕГИСТР/ПАМЯТЬ                         ! 1 0 0 0 1 1 0 0 ! MOD 0 SR  R/M !    (DISP-LO)    !    (DISP-HI)    !
     В СЕГМЕНТНЫЙ РЕГИСТР              +-----------------+-----------------+-----------------+-----------------+
СЕГМЕНТНЫЙ РЕГИСТР
     В РЕГИСТР/ПАМЯТЬ
```

Рис. 3.71. Машинные коды команд микропроцессора 8086

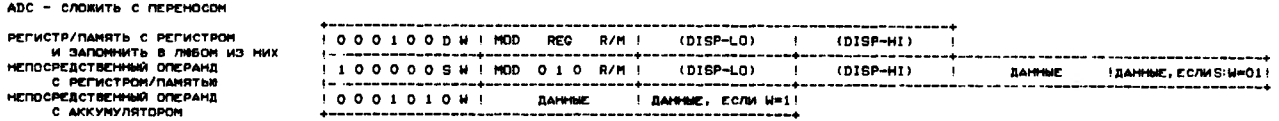

PUSH – ВКЛЮЧИТЬ В СТЕК
1 1 1 1 1 1 1 1 | MOD 1 1 0 R/M | (DISP-LO) | (DISP-HI)
РЕГИСТР/ПАМЯТЬ
0 1 0 1 0 REG
РЕГИСТР
0 0 0 REG 1 1 0
СЕГМЕНТНЫЙ РЕГИСТР
POP – ИЗВЛЕЧЬ ИЗ СТЕКА
1 0 0 0 1 1 1 1 | MOD 0 0 0 R/M | (DISP-LO) | (DISP-HI)
РЕГИСТР/ПАМЯТЬ
0 1 0 1 1 REG
РЕГИСТР
0 0 0 REG 1 1 1
СЕГМЕНТНЫЙ РЕГИСТР
XCHG – ОБМЕНЯТЬ
7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0
РЕГИСТР/ПАМЯТЬ С РЕГИСТРОМ
1 0 0 0 0 1 1 W | MOD REG R/M | (DISP-LO) | (DISP-HI)
РЕГИСТР С АККУМУЛЯТОРОМ
1 0 0 1 0 REG
IN – ВВЕСТИ ИЗ:
ФИКСИРОВАННОГО ПОРТА
1 1 1 0 0 1 0 W | DATA-8
ПЕРЕМЕННОГО ПОРТА
1 1 1 0 1 1 0 W
OUT – ВЫВЕСТИ В:
ФИКСИРОВАННЫЙ ПОРТ
1 1 1 0 0 1 1 W | DATA-8
ПЕРЕМЕННЫЙ ПОРТ
1 1 1 0 1 1 1 W
XLAT – ПРЕОБРАЗОВАТЬ БАЙТ ИЗ AL
1 1 0 1 0 1 1 1
LEA – ЗАГРУЗИТЬ EA В РЕГИСТР
1 0 0 0 1 1 0 1 | MOD REG R/M | (DISP-LO) | (DISP-HI)
LDS – ЗАГРУЗИТЬ УКАЗАТЕЛЬ В DS
1 1 0 0 0 1 0 1 | MOD REG R/M | (DISP-LO) | (DISP-HI)
LES – ЗАГРУЗИТЬ УКАЗАТЕЛЬ В ES
1 1 0 0 0 1 0 0 | MOD REG R/M | (DISP-LO) | (DISP-HI)
LAHF – ЗАГРУЗИТЬ ФЛАЖКИ В AH
1 0 0 1 1 1 1 1
SAHF – ЗАПОМНИТЬ AH ВО ФЛАЖКАХ
1 0 0 1 1 1 1 0
PUSHF– ВКЛЮЧИТЬ В СТЕК ФЛАЖКИ
1 0 0 1 1 1 0 0
POPF– ИЗВЛЕЧЬ ИЗ СТЕКА ФЛАЖКИ
1 0 0 1 1 1 0 1

АРИФМЕТИЧЕСКИЕ КОМАНДЫ

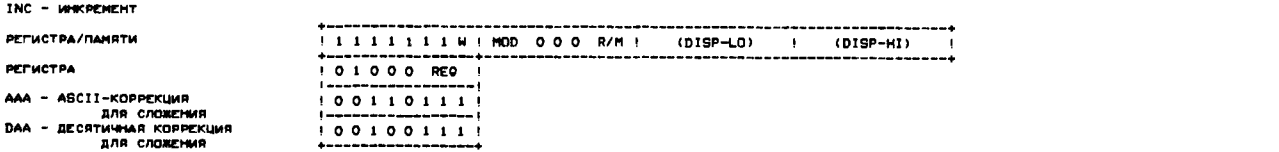
ADD - СЛОЖИТЬ



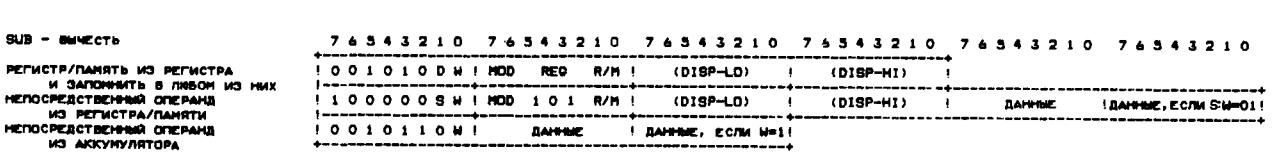
ADC - СЛОЖИТЬ С ПЕРЕНОСОМ



INC - ИНКРЕМЕНТ



SUB - ВЫЧЕСТЬ



SBB - ВЫЧЕСТЬ С ЗАЕМОМ

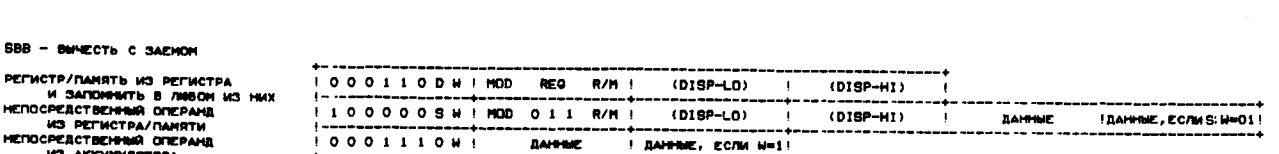
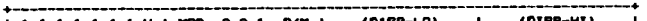


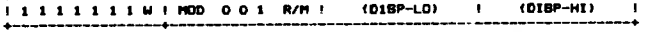
Рис. 3.71 (продолжение)

DEC – ДЕКРЕМЕНТ

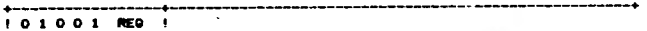
РЕГИСТРА/ПАМЯТИ



РЕГИСТРА

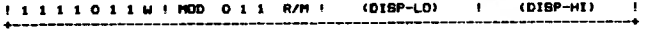


NEG – ИЗМЕНИТЬ ЗНАК

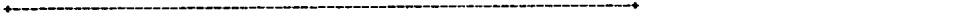


CMP – СРАВНИТЬ

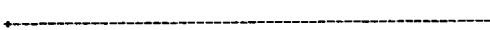
РЕГИСТР/ПАМЯТЬ И РЕГИСТР



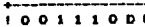
НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД И РЕГИСТР/ПАМЯТЬ



НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД И АККУМУЛЯТОР

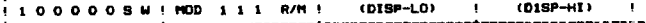


AAS – ASCII-КОРРЕКЦИЯ ДЛЯ ВЫЧИТАНИЯ

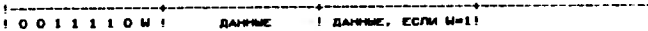


DAS – ДЕСЯТИЧНАЯ КОРРЕКЦИЯ ДЛЯ ВЫЧИТАНИЯ

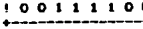
MUL – УМНОЖИТЬ (БЕЗ ЗНАКА)



IMUL – УМНОЖИТЬ (СО ЗНАКОМ)



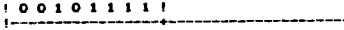
AAM – ASCII-КОРРЕКЦИЯ ДЛЯ УМНОЖЕНИЯ



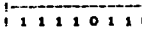
DIV – РАЗДЕЛИТЬ (БЕЗ ЗНАКА)

IDIV – РАЗДЕЛИТЬ (СО ЗНАКОМ)

AAD – ASCII-КОРРЕКЦИЯ ДЛЯ ДЕЛЕНИЯ



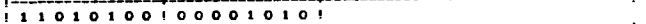
CBW – ПРЕОБРАЗОВАТЬ БАЙТ В СЛОВО



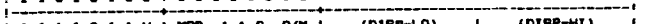
CWD – ПРЕОБРАЗОВАТЬ СЛОВО В ДВОЙНОЕ СЛОВО

ЛОГИЧЕСКИЕ КОМАНДЫ

NOT – ИНВЕРТИРОВАТЬ



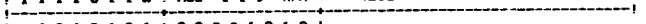
SHL/SAL – СДВИНУТЬ ЛОГИЧЕСКИ/АРИФМЕТИЧЕСКИ ВЛЕВО



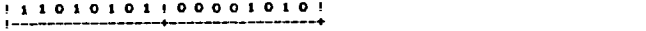
SHR – СДВИНУТЬ ЛОГИЧЕСКИ ВПРАВО

SAR – СДВИНУТЬ АРИФМЕТИЧЕСКИ ВПРАВО

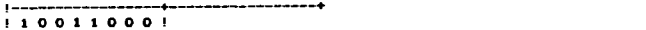
ROL – СДВИНУТЬ ЦИКЛИЧЕСКИ ВЛЕВО



ROR – СДВИНУТЬ ЦИКЛИЧЕСКИ ВПРАВО



RCL – СДВИНУТЬ ЦИКЛИЧЕСКИ ВЛЕВО ЧЕРЕЗ ПЕРЕНОС



RCR – СДВИНУТЬ ЦИКЛИЧЕСКИ ВПРАВО ЧЕРЕЗ ПЕРЕНОС

AND – ОБ'ЕДИНИТЬ ПО "И"

РЕГИСТР/ПАМЯТЬ С РЕГИСТРОМ И ЗАПОМНИТЬ В ЛЮБОМ ИЗ НИХ	0 0 1 0 0 0 D W	MOD REG R/M	(DISP-LO)	(DISP-HI)		
НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД С РЕГИСТРОМ/ПАМЯТЬЮ	1 0 0 0 0 0 0 W	MOD 1 0 0 R/M	(DISP-LO)	(DISP-HI)	ДАННЫЕ	ДАННЫЕ, ЕСЛИ W=01
НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД С АККУМУЛЯТОРОМ	0 0 1 0 0 1 0 W	ДАННЫЕ	ДАННЫЕ, ЕСЛИ W=1			

TEST – ПРОВЕРИТЬ (ФУНКЦИЯ "И" БЕЗ ЗАПОМИНАНИЯ РЕЗУЛЬТАТА)

РЕГИСТР/ПАМЯТЬ И РЕГИСТР	1 0 0 0 0 1 0 W	MOD REG R/M	(DISP-LO)	(DISP-HI)		
НЕПОСРЕДСТВЕННЫЕ ДАННЫЕ И РЕГИСТР/ПАМЯТЬ	1 1 1 1 0 1 1 W	MOD 0 0 0 R/M	(DISP-LO)	(DISP-HI)	ДАННЫЕ	ДАННЫЕ, ЕСЛИ W=01
НЕПОСРЕДСТВЕННЫЕ ДАННЫЕ И АККУМУЛЯТОР	1 0 1 0 1 0 0 W	ДАННЫЕ	ДАННЫЕ, ЕСЛИ W=1			

OR – ОБ'ЕДИНИТЬ ПО "ИЛИ"

РЕГИСТР/ПАМЯТЬ С РЕГИСТРОМ И ЗАПОМНИТЬ В ЛЮБОМ ИЗ НИХ	0 0 0 0 1 0 D W	MOD REG R/M	(DISP-LO)	(DISP-HI)		
НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД С РЕГИСТРОМ/ПАМЯТЬЮ	1 0 0 0 0 0 0 W	MOD 0 0 1 R/M	(DISP-LO)	(DISP-HI)	ДАННЫЕ	ДАННЫЕ, ЕСЛИ W=01
НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД С АККУМУЛЯТОРОМ	0 0 0 0 1 1 0 W	ДАННЫЕ	ДАННЫЕ, ЕСЛИ W=1			

XOR – СЛОЖИТЬ ПО МОДУЛЮ 2 (ИСКЛЮЧАЮЩЕЕ-ИЛИ)

РЕГИСТР/ПАМЯТЬ С РЕГИСТРОМ И ЗАПОМНИТЬ В ЛЮБОМ ИЗ НИХ	0 0 1 1 0 0 D W	MOD REG R/M	(DISP-LO)	(DISP-HI)		
НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД С РЕГИСТРОМ/ПАМЯТЬЮ	1 0 0 0 0 0 0 W	MOD 1 1 0 R/M	(DISP-LO)	(DISP-HI)	ДАННЫЕ	ДАННЫЕ, ЕСЛИ W=01
НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД С АККУМУЛЯТОРОМ	0 0 1 1 0 1 0 W	ДАННЫЕ	ДАННЫЕ, ЕСЛИ W=1			

КОМАНДЫ МАНИПУЛЯЦИИ ЦЕПОЧКАМИ

REP – ПОВТОРИТЬ	1 1 1 1 0 0 1 Z
MOVS – ПЕРЕДАТЬ БАЙТ/СЛОВО	1 0 1 0 0 1 0 W
CMPS – СРАВНИТЬ БАЙТ/СЛОВО	1 0 1 0 0 1 1 W
SCAS – СКАНИРОВАТЬ БАЙТ/СЛОВО	1 0 1 0 1 1 1 W
LODS – ЗАГРУЗИТЬ БАЙТ/СЛОВО В AL/AX	1 0 1 0 1 1 0 W
STOS – ЗАПОМНИТЬ БАЙТ/СЛОВО ИЗ AL/AX	1 0 1 0 1 0 1 W

КОМАНДЫ ПЕРЕДАЧИ УПРАВЛЕНИЯ

CALL – ВЫЗВАТЬ (ВЫЗОВ)	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
ПРЯМОЙ ВНУТРИСЕГМЕНТНЫЙ	1 1 1 0 1 0 0 0	IP-INC-LO	IP-INC-HI			
КОСВЕННЫЙ ВНУТРИСЕГМЕНТНЫЙ	1 1 1 1 1 1 1 1	MOD 0 1 0 R/M	(DISP-LO)	(DISP-HI)		
ПРЯМОЙ МЕЖСЕГМЕНТНЫЙ	1 0 0 1 1 0 1 0	IP-LO	IP-HI			
		CS-LO	CS-HI			
КОСВЕННЫЙ МЕЖСЕГМЕНТНЫЙ	1 1 1 1 1 1 1 1	MOD 0 1 1 R/M	(DISP-LO)	(DISP-HI)		

Рис. 3.71 (продолжение)

JMP – БЕЗУСЛОВНО ПЕРЕЙТИ (ПЕРЕХОД)

ПРЯМОЙ ВНУТРИСЕГМЕНТНЫЙ

ПРЯМОЙ ВНУТРИСЕГМЕНТНЫЙ КОРОТКИЙ

КОСВЕННЫЙ ВНУТРИСЕГМЕНТНЫЙ

ПРЯМОЙ МЕЖСЕГМЕНТНЫЙ

КОСВЕННЫЙ МЕЖСЕГМЕНТНЫЙ

1 1 1 0 1 0 0 1	IP-INC-LO	IP-INC-HI		
1 1 1 0 1 0 1 1	IP-INC8			
1 1 1 1 1 1 1 1	MOD 1 0 0 R/M	(DISP-LO)	(DISP-HI)	
1 1 1 0 1 0 1 0	IP-LO	IP-HI		
	CS-LO	CS-HI		
1 1 1 1 1 1 1 1	MOD 1 0 1 R/M	(DISP-LO)	(DISP-HI)	

RET – ВОЗВРАТИТЬСЯ ИЗ ВЫЗОВА (ВОЗВРАТ)

ВНУТРИСЕГМЕНТНЫЙ

ВНУТРИСЕГМЕНТНЫЙ С ПРИБАВЛЕНИЕМ НЕПОСРЕДСТВЕННОГО ОПЕРАНДА К SP

МЕЖСЕГМЕНТНЫЙ

МЕЖСЕГМЕНТНЫЙ С ПРИБАВЛЕНИЕМ НЕПОСРЕДСТВЕННОГО ОПЕРАНДА К SP

1 1 0 0 0 0 1 1		
1 1 0 0 0 0 1 0	МЛ. БАЙТ ДАННЫХ	СТ. БАЙТ ДАННЫХ
1 1 0 0 1 0 1 1		
1 1 0 0 1 0 1 0	МЛ. БАЙТ ДАННЫХ	СТ. БАЙТ ДАННЫХ

JE/JZ – ПЕРЕЙТИ, ЕСЛИ РАВНО/НУЛЬ

JL/JNGE – ПЕРЕЙТИ, ЕСЛИ МЕНЬШЕ/НЕ БОЛЬШЕ ИЛИ РАВНО

JLE/JNG – ПЕРЕЙТИ, ЕСЛИ МЕНЬШЕ ИЛИ РАВНО/НЕ БОЛЬШЕ

JS/JNAE – ПЕРЕЙТИ, ЕСЛИ НИЖЕ/НЕ ВЫШЕ ИЛИ РАВНО

JBE/JNA – ПЕРЕЙТИ, ЕСЛИ НИЖЕ ИЛИ РАВНО/НЕ ВЫШЕ

JP/JPE – ПЕРЕЙТИ, ЕСЛИ ПАРИТЕТ УСТАНОВЛЕН (ЧЕТНЫЙ ПАРИТЕТ)

JO – ПЕРЕЙТИ, ЕСЛИ ЕСТЬ ПЕРЕПОЛНЕНИЕ

JS – ПЕРЕЙТИ, ЕСЛИ ЗНАК УСТАНОВЛЕН

JNE/JNZ – ПЕРЕЙТИ, ЕСЛИ НЕ РАВНО/НЕ НУЛЬ

JNL/JGE – ПЕРЕЙТИ, ЕСЛИ НЕ МЕНЬШЕ/БОЛЬШЕ ИЛИ РАВНО

JNLE/JG – ПЕРЕЙТИ, ЕСЛИ НЕ МЕНЬШЕ ИЛИ РАВНО/БОЛЬШЕ

JNB/JAE – ПЕРЕЙТИ, ЕСЛИ НЕ НИЖЕ/ВЫШЕ ИЛИ РАВНО

JNBE/JA – ПЕРЕЙТИ, ЕСЛИ НЕ НИЖЕ ИЛИ РАВНО/ВЫШЕ

JNP/JPO – ПЕРЕЙТИ, ЕСЛИ ПАРИТЕТ СБРОШЕН (НЕЧЕТНЫЙ ПАРИТЕТ)

JNO – ПЕРЕЙТИ, ЕСЛИ НЕТ ПЕРЕПОЛНЕНИЯ

0 1 1 1 0 1 0 0	IP-INC8
0 1 1 1 1 1 0 0	IP-INC8
0 1 1 1 1 1 1 0	IP-INC8
0 1 1 1 0 0 1 0	IP-INC8
0 1 1 1 0 1 1 0	IP-INC8
0 1 1 1 1 0 1 0	IP-INC8
0 1 1 1 0 0 0 0	IP-INC8
0 1 1 1 1 0 0 0	IP-INC8
0 1 1 1 0 1 0 1	IP-INC8
0 1 1 1 1 1 0 1	IP-INC8
0 1 1 1 1 1 1 1	IP-INC8
0 1 1 1 0 0 1 1	IP-INC8
0 1 1 1 0 1 1 1	IP-INC8
0 1 1 1 1 0 1 1	IP-INC8
0 1 1 1 0 0 0 1	IP-INC8

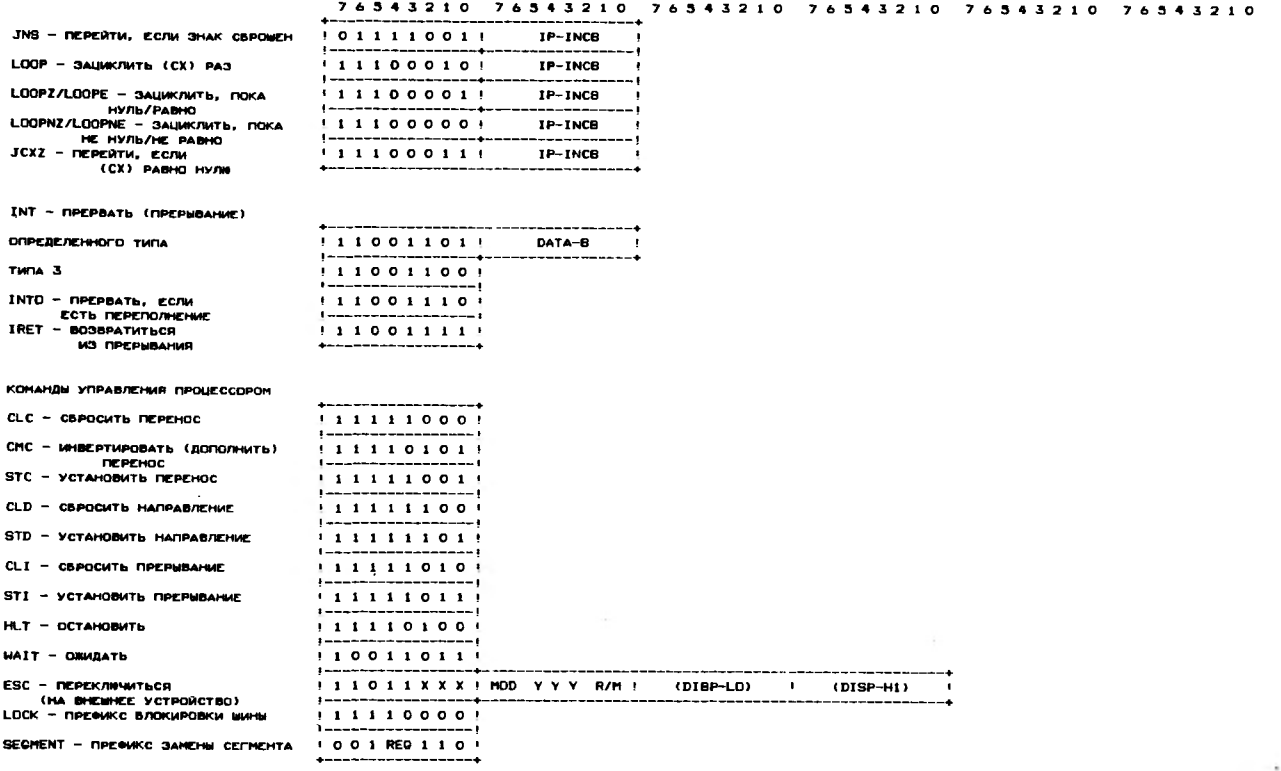
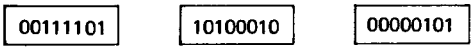


Рис. 3.71 (окончание)

Во всех случаях короткий формат на один байт короче соответствующего длинного формата и иногда команда имеет длину всего один байт. Как правило, ассемблер всегда генерирует короткий формат. Например, команда CMP AX,5A2H ассемблируется в формате



а не в длинном формате

Упражнения

1. Считайте, что LAB является меткой, VAR – переменной, CON – именем константы, и идентифицируйте все возможные режимы адресации для каждого из следующих операндов:

а) VAR [BX] б) CON + 63H в) LAB г) VAR
д) VAR [BX + 6] е) VAR [BX] [SI] ж) VAR [BX + 3] [DI + 9]

2. Поясните, почему иногда необходим атрибутный оператор PTR.

3. Напишите программу, которая загружает 75_{10} в регистр DX, а затем передает (DX) в каждое из трех слов, начинающихся с ячейки, ассоциируемой со словом BLOCK. Приведите рисунок, аналогичный рис. 3.8, *б*, который показывает действия при выполнении этой программы.

4. Напишите программу, которая изменяет порядок содержимого 4 байт от LIST до LIST + 3.

5. Какие из приведенных ниже ассемблерных команд являются недопустимыми? Укажите ошибку в каждой недопустимой команде. Считайте все идентификаторы переменными, которые определены как слова:

```
MOV BP,AL
MOV WORD_OP [BX + 4 * 3] [DI],SP
MOV WORD_OP1,WORD_OP2
MOV AX,WORD_OP1 [DX]
MOV CS,AX
MOV DS,BP
MOV SAVE_WORD,DS
MOV SP,SS:DATA_WORD [BX] [SI]
MOV [BX] [SI],2
MOV AX,WORD_OP1 + WORD_OP2
MOV AX,WORD_OP1 – WORD_OP2 + 100
MOV WORD_OP1,WORD_OP1 – WORD_OP2
```

6. Поясните различия между командами

```
MOV AX,TABLE_ADDR  и  LEA AX,TABLE_ADDR
```

7. Дайте рисунок, аналогичный рис. 3.8, *б*, показывающий действия при выполнении программы, приведенной на рис. 3.14.

8. Для каждой из следующих команд приведите рисунок, аналогичный рис. 3.7:

а) MOV BX,CX б) XCHG OPR [SI],AX
в) ADD DX,67H г) LEA CX,X [BX] [DI]

9. Напишите программы, которые выполняют следующие двоичные операции:

а) Z ← W + (X − Z) б) Z ← W − (X + 6) − (Y + 9)

в) Z ← (W * X)/(Y + 6) R ← остаток г) Z ← ((W − X)/10 * Y) * * 2

10. Покажите, что перенос всегда устанавливается правильно при выполнении сложения с многократной точностью чисел в дополнительном коде. Покажите также, что он правильно устанавливается и при вычитании.

11. Воспользуйтесь командами условного перехода для модификации программы на рис. 3.10 так, чтобы она умножала знаковые числа двойной точности в дополнительном коде.

12. Напишите программу беззнакового двоичного деления числа из n слов на однословное число.

13. Воспользуйтесь командами условного перехода для модификации ответа в упр. 12 так, чтобы можно было делить знаковые числа.

14. Напишите программы для реализации следующих операций над двухразрядными упакованными BCD-числами:

а) U ← V + (S − 6) б) U ← (X + W) − (Z − W)

15. Повторите упр. 14 для 4-разрядных упакованных BCD-чисел.

16. Повторите упр. 14 для одноразрядных неупакованных BCD-чисел.

17. Повторите упр. 14 для двухразрядных неупакованных BCD-чисел. Напишите также команды, реализующие операцию

U ← X * Y/Z (игнорировать остаток)

где Z – одноразрядное число.

18. Модифицируйте пример на рис. 3.26 для трехразрядных чисел в десятичном дополнительном коде.

19. Покажите с помощью математических выкладок, что неупакованное BCD-произведение правильно корректируется посредством деления (AL) на 10 с загрузкой частного в AH и остатка в AL.

20. Предположим, что последовательность неупакованных BCD-цифр делится на неупакованную BCD-цифру. Покажите, что для каждой цифры правильное неупакованное BCD-частное получается путем умножения (AH) (остатка от предыдущего деления) на 10 и прибавления его к (AL) (следующей цифре) до выполнения двоичного деления.

21. Напишите циклическую программу для сложения двух 16-разрядных упакованных BCD-чисел в дополнительном коде. Повторите упражнение для неупакованных BCD-чисел.

22. Алгоритм беззнакового деления с двойной точностью числа $c2^{16} + d$ на число $a2^{16} + b$ заключается в следующем. Вычислить такие q и r, что $c = qa + r$ (q и r можно получить делением с однократной точностью). Но равенство $c = qa + r$ означает, что

$$c2^{16} + d = qa2^{16} + r2^{16} + d = \\ = q(a2^{16} + b) + r2^{16} + d - qb$$

Величина $r2^{16} + d - qb$ считается остатком первоначального деления; если она отрицательна, следует производить повторяющийся декремент q до тех пор, пока она не станет положительной.

Реализуйте данную процедуру, пользуясь командой DIV.

23. Разработайте алгоритм неупакованного BCD-деления, когда делимое имеет n разрядов, а делитель – m разрядов.

24. Найдите ошибки в следующих командах, считая VAR1 и VAR2 переменными-словами, а LAB – меткой:

а) AAA DX б) ADD VAR1,VAR2

в) SUB AL,VAR1 г) JMP LAB[SI]

д) JNZ VAR1 е) JMP NEAR LAB

25. Постройте машинный код каждой из следующих команд, полагая, что ADDR имеет смещение 10B0 и что команда имеет смещение 10A0:

а) JNZ ADDR б) JB ADDR–2

в) JMP SHORT ADDR г) JMP NEAR PTR ADDR

Повторите упражнение, считая, что команда имеет смещенне 10C0.

26. Предположим, что BR_ADDR имеет смещение 0900 в сегменте CODE2 и что сегментный адрес CODE2 равен 0100. Образуйте машинный код команды

JMP FAR BR_ADDR

27. Пусть TAB имеет смещение 1C00. Постройте машинный код следующих команд внутрисегментных косвенных переходов:

а) JMP TAB б) JMP TAB[SI] в) JMP WORD PTR [BX][SI]

28. Напишите программу, реалнзующую действия схемы на рис. 3.72.

29. Напишите программу, которая упорядочивает в алфавитном порядке буквы, коды ASCII которых находятся в CHAR, CHAR + 1 н CHAR + 2.

30. Напишите программу, которая пересылает информацию из области с начальным адресом LIST и конечным адресом LIST + 100 в область с соответствующими адресами BLK и BLK + 100.

31. Напишите программу, которая изменяет порядок следования байт от ARRAY до ARRAY + N – 1, где N – содержимое NUM.

32. Модифицируйте программу пузырьковой сортировки на рис. 3.44, предусмотрев проверку более раннего окончания сортировки, что повышает ее эффективность.

33. Воспользуйтесь алгоритмом, предложенным в конце раздела 3.3.1, для того, чтобы в упр. 22 допускалось делимое 8-кратной точности.

34. Сортировка вставкой размещенння чисел в убывающем порядке начинается с упорядочивания первых двух элементов масснва. Затем в нужное место вставляется третий элемент по результату его сравнения с текущим вторым элементом и, возможно, с текущим первым элементом. После чего в правильное место по результатам необходимых сравнений вставляется четвертый элемент н т. д. Напишите программу упорядочивания

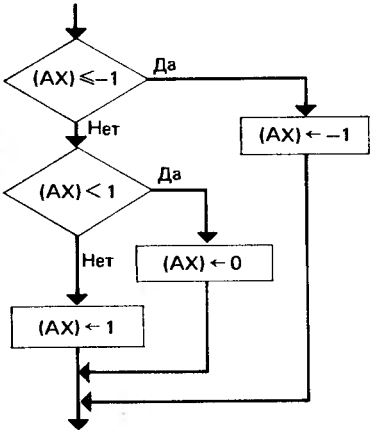


Рис. 3.72. Схема программы для упр. 28

по методу сортировки вставкой массива слов LIST, содержащего N элементов, причем N находится в LEN.

35. Напишите программу преобразования 16-разрядного упакованного BCD-числа, начинающегося в PACKED, в неупакованное BCD-число и запоминания результата, начиная с UNPACKED. Модифицируйте ее так, чтобы неупакованное число было цепочкой в коде ASCII.

36. Напишите программу, которая проверяет байт STATUS и переходит к ROUTINE_1, если биты 1, 3 или 5 содержат 1. В противном случае она переходит к ROUTINE_2, если оба бита 1 и 3 содержат единицу, и к ROUTINE_3, если биты 1 и 3 содержат 0. Во всех остальных случаях программа переходит к ROUTINE_4. Считайте, что стандартные программы (routine) имеют длину более 128 байт. Постройте схему программы.

37. Пусть в области от STRING до STRING + 99 находится цепочка символов. Предположим, что необходимо установить в 1 бит 5 регистра DL, если цепочка содержит цифру; в противном случае этот бит сбрасывается в 0. В любом случае воздействие отиосится только к биту 5. Постройте схему решения задачи, а затем напишите ассемблерную программу.

38. Оцените время выполнения следующих эквивалентных программных фрагментов (частота синхронизации 5 МГц, времена выполнения команд даны в приложении):

а)
```
        MOV   CX,5
NEXT:   ADD   AX,[SI]
        INC   SI
        LOOP  NEXT
```

б)
```
        MOV   BX,5
NEXT:   ADD   AX,[SI]
        INC   SI
        DEC   BX
        JNZ   NEXT
```

39. Четыре бита в кодах операции команд условных переходов определяют производимую командой проверку. Что это за биты? (См. рис. 3.71).

40. Считайте содержимое регистров DX:AX 32-битной величиной и напишите программу, загружающую в регистр DL номер младшего бита, который содержит 1.

41. Напишите программу, которая упаковывает четыре 12-битных величины из четырех смежных слов в три смежных слова.

42. Команды сдвигов и циклических сдвигов различаются только 3 битами. Какими битами эти команды отличаются друг от друга? (См. рис. 3.71).

43. Покажите распределяемое пространство и инициализируемые данные для следующих директив:

а) BYTE_VAR DB 'BYTE',12,-12H,3 DUP(0,?,2 DUP(1,2),?)

б) WORD_VAR DW 5 DUP(0,1,2),?,-5,'BY','TE',256H

44. Пусть DECIMAL_STRING и BINARY_WORD определяются в сегменте DATA_SEG. Напишите законченный сегмент кода, который выполняет следующие преобразования:

а) положительного BCD-числа, четыре неупакованные BCD-цифры которого хранятся в DECIMAL_STRING, в 16-битное двоичное число с запоминанием его в BINARY_WORD;

б) двоичного числа, хранимого в BINARY_WORD, в эквивалентное число в неупакованном BCD-формате с запоминанием результата в DECIMAL_STRING.

45. Определите структуру INVEN_REC, которая начинается со слова с именем PART (без инициализации), после которого следуют 3 байта с именем TYPE, содержащие 05H, 0A2H и 52H, а в конце находится двойное слово с именем SIZE (без инициализации). Приведите оператор, который вызывает INVEN_REG и инициализирует поле PART на 6257H, не изменяя других полей. Затем определите сегмент данных для склада. Сегмент начинается с 9-байтной цепочки с именем NAME и начальным значением 'INVENTORY', продолжается байтом LEN без инициализации и заканчивается 100 копиями INVEN_REC (без инициализации).

46. Пусть 8 булевых переменных $X_0, X_1, \ldots, X_7$ хранятся в переменной INPUT со следующим назначением бит:

7	6	5	4	3	2	1	0
X_7	X_6	X_5	X_4	X_3	X_2	X_1	X_0

а) Определите RECORD для такого назначения бит и резервирования памяти для INPUT.

б) Напишите программу вычисления булева выражения

$$\overline{X}_7 X_2 X_1 \overline{X}_0 + \overline{X}_3 X_6 + \overline{X}_5 \overline{X}_4 X_3$$

и запоминания результата (0 или 1) в регистре DX.

в) Напишите программу, в которой применяется таблица переходов BR_TAB для передачи управления ROUTINE_i, если $X_i = 1$, $i = 0 \ldots 7$. Поиск единичного бита осуществляется справа налево, а при отсутствии единичных бит переход не предпринимается.

47. Пусть переменные-слова X, Y и Z определены в сегментах X_SEG, Y_SEG и Z_SEG соответственно, и регистры DS, ES и SS зарезервированы для X_SEG, Y_SEG и Z_SEG. Напишите программу вычисления

$$X \leftarrow X + Y + Z$$

в которой имеются необходимые операторы ASSUME и команды для загрузки сегментных регистров.

48. Опишите законченный сегмент данных DATA_SEG, в котором целое число 5 назначено байту NUM и целые числа –1, 0, 2, 5 и 4 назначены первым пяти элементам массива DATA_LIST из 10 слов. Затем напишите законченный сегмент кода, который выполняет следующие действия:

а) помещает наибольшее и наименьшее из пяти первых чисел DATA_LIST в регистры BX и DX;

б) вычисляет сумму и произведение первых пяти чисел DATA_LIST и запоминает результаты в SUM и PRODUCT соответственно.

49. Напишите законченную программу, которая прибавляет AUGEND из сегмента D_SEG к ADDEND из сегмента E_SEG и помещает результат в SUM сегмента D_SEG. Переменные AUGEND, ADDEND и SUM являются двоичными числами двойной точности, причем AUGEND инициализирована на 99251 и ADDEND – на –15962. Код должен находиться в C_SEG и сегменты D_SEG, E_SEG и C_SEG ассоциируются с регистрами DS, ES и CS соответственно.

50. Постройте содержимое таблицы имен и машинный код, формируемые ассемблером для следующей программы:

```
PROG          SEGMENT
 ASSUME    CS:PROG, DS:PROG
  DATA_ARRAY   DW     10 DUP(?)
  SUM          DW     ?
  START:       MOV    AX,CS
               MOV    DS,AX
               XOR    AX,AX
               MOV    BX,AX
               MOV    CX,10
  NEXT:        ADD    AX,DATA_ARRAY[BX]
               ADD    BX,2
               LOOP   NEXT
               MOV    SUM,AX
               HLT
PROG          ENDS
               END    START
```

51. Модифицируйте программу в упр. 50 так, чтобы данные находились в отдельном сегменте NUMBERS, а затем повторите упражнение.

52. Предположите, что смещение DATA_WORD равно 123E, и приведите все возможные машинные команды, эквивалентные командам:

а) MOV AX,DATA_WORD
б) MOV BX,34
в) MOV DATA_WORD,34

53. Пусть переменные AMT и QUAN имеют смещения 0300 и 1A00 соответственно. Постройте машинный код каждой из команд:

а) ADD AMT,–32H
б) ADC DX,AMT + 4
в) SUB AX,QUAN
г) SBB QUAN[SI + 2],AX
д) CMP AX,QUAN
е) DAA
ж) ADD AX,DX
з) CMP DX,0
и) INC CX
к) NEG QUAN[BX][SI]
л) ADD DX,AMT[BX][SI + 2]
м) CWD

4. МОДУЛЬНОЕ ПРОГРАММИРОВАНИЕ

В гл. 3 речь шла об отдельных командах и об объединении их для выполнения простых задач. Но вопросы построения законченных программ, которые там обсуждались, относились к простым одномодульным программам. Ясно, что большинство практических программ оказываются гораздо сложнее и включают в себя несколько задач, подобно рассмотренным в примерах из гл. 3. В данной главе обсуждается построение сложных программ из нескольких небольших фрагментов, называемых *программными модулями* (или просто *модулями*) и выполняющих четко определенные задачи. Такое построение машинного кода называется *модульным программированием.*

Этапы разработки ассемблерных программ аналогичны соответствующим этапам разработки программ на языках высокого уровня. Они включают в себя:

1) точную формулировку функций программы;
2) разделение общей проблемы на отдельные задачи;
3) точное определение того, что должна делать каждая задача, и взаимодействие ее с другими задачами;
4) превращение задач в ассемблерные модули и объединение модулей в законченную программу;
5) отладку и тестирование программы;
6) документирование программы.

В данной главе мы коснемся в основном этапов 2, 3 и 4. Они аналогичны этапам действия разработчика аппаратных средств, который вначале разделяет систему на схемные модули, обычно соответствующие печатным платам, проектирует модули, а затем объединяет их в систему.

Основным средством для разделения программы на модули оказывается *иерархическая диаграмма*, т. е. схема, показывающая взаимосвязи между модулями (задачами) и подмодулями (подзадачами). Типичная иерархическая диаграмма приведена на рис. 4.1. Такие диаграммы похожи на организа-

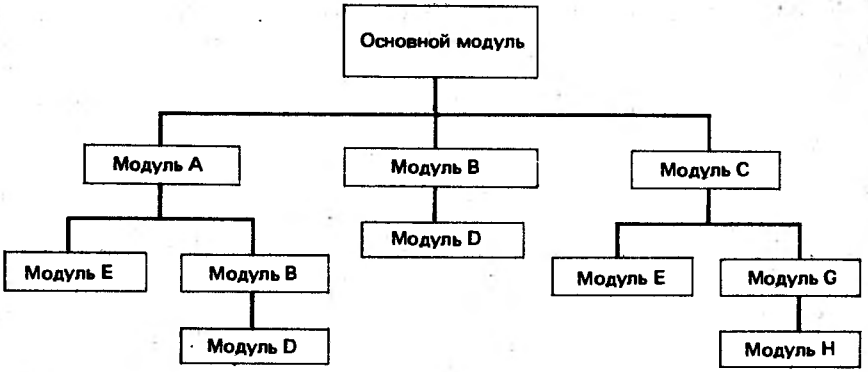


Рис. 4.1. Типичная иерархическая диаграмма

ционную структуру корпораций и служат тем же целям. Основной модуль соответствует президенту корпорации, главные подмодули A, B и C соответствуют вице-президентам и т. д. Иерархическая диаграмма показывает существующие между модулями и их подмодулями связи подчиненности подобию тому, как организационная структура показывает субординацию персонала корпорации. Диаграмма помогает программисту и другим людям визуализировать функциональную структуру программы. Отметим, что подмодуль может быть подчинен нескольким модулям.

Имеются следующие причины разделения программы на небольшие части:

в отдельных модулях проще разобраться;

различные модули можно поручить разрабатывать нескольким программистам;

отладку и тестирование можно осуществить более упорядоченным образом;

проще разбираться в документации;

модификации в программе оказываются локализованными;

часто требуемые задачи можно оформить в модули, которые хранятся в библиотеках и используются несколькими программами.

Относительно первой части заметим, что человек одновременно может сконцентрировать свое внимание только на ограниченном объеме материала. Следовательно, нам приходится "разделять" свое мышление независимо от вида решаемой проблемы. Модульное программирование оказывается естественным распространением этого положения на программирование.

При разработке программных модулей необходимо рассмотреть, как и при каких обстоятельствах модули инициируются и завершаются (*сопряжение по управлению*) и какая информация передается между модулями (*сопряжение по данным*). Связи между модулями зависят от нескольких факторов, например организации данных, или от того, как ассемблируются модули: вместе или отдельно. В общем, выбор задач и разработка модулей должны обеспечивать простые связи по управлению и минимальные связи по данным.

В большинстве языков ассемблеров предусматриваются три средства, помогающие реализовать модульность программ. Первое из них позволяет структурировать данные таким образом, что к ним могут легко обращаться несколько модулей. Второе заключается в применении процедур (или подпрограмм). Наконец, третье средство позволяет вводить секции кода (*макрокоманды*) с помощью одного оператора, содержащего имя и набор аргументов. Структурирование данных обсуждалось в § 3.10 и рассматривается далее в первых двух параграфах данной главы. В § 4.1 показано также, как объединяются отдельно ассемблированные модули и как программы готовятся для выполнения. В § 4.2 и 4.3 рассматриваются стеки и процедуры и их реализация в микропроцессоре 8086. Следующий параграф посвящен прерываниям, а § 4.5 — макрокомандам. Наконец, в завершающих § 4.6 и 4.7 речь идет о разработке ассемблерных программ и особенно программ, предназначенных для микропроцессора 8086.

4.1. РЕДАКТИРОВАНИЕ СВЯЗЕЙ И ПЕРЕМЕЩЕНИЕ

При разработке программы некоторые ее модули могут находиться в одном и том же исходном модуле и ассемблируются вместе, а другие оформлены в разных исходных модулях и ассемблируются отдельно. Если модули ассемблируются отдельно, основной модуль, в котором находится первая выполняемая программа, должен заканчиваться оператором END с указанием точки входа, а остальные модули должны заканчиваться безоперандными операторами END. В любом случае получающиеся объектные модули, некоторые из которых могут быть сгруппированы в библиотеки, должны быть связаны до выполнения программы в загрузочный модуль. Кроме вывода загрузочного модуля, редактор связей обычно печатает *карту памяти*, показывающую, в какие области памяти загружаются связанные объектные модули. После образования загрузочного модуля загрузчик размещает его в память компьютера и начинается выполнение программы. Хотя ввод-вывод и может осуществляться модулями в программе, обычно его реализуют драйверы ввода-вывода, являющиеся частью операционной системы. В пользовательской программе находятся только обращения к драйверам ввода-вывода, которые и заставляют операционную систему выполнить их.

Общий процесс создания и выполнения программы представлен на рис. 4.2. Конечно, в конкретной системе может не быть точного соответствия рисунку, но общие принципы оказываются одними и теми же. Стрелки показывают, что после любого из основных этапов могут потребоваться коррективы. В программном обеспечении MCS-86 редактор связей состоит из двух компонент: собственно редактора связей и распределителя, а загрузка осуществляется операционной системой. Однако независимо от системы комбинация редактора связей и загрузчика должна осуществлять все назначения сегментов и адресов, чтобы программа выполнялась правильно. Если говорить конкретнее, редактор связей и загрузчик должны:

найти подлежащие связыванию объектные модули;

построить загрузочный модуль, указывая положения всех сегментов во всех связываемых объектных модулях;

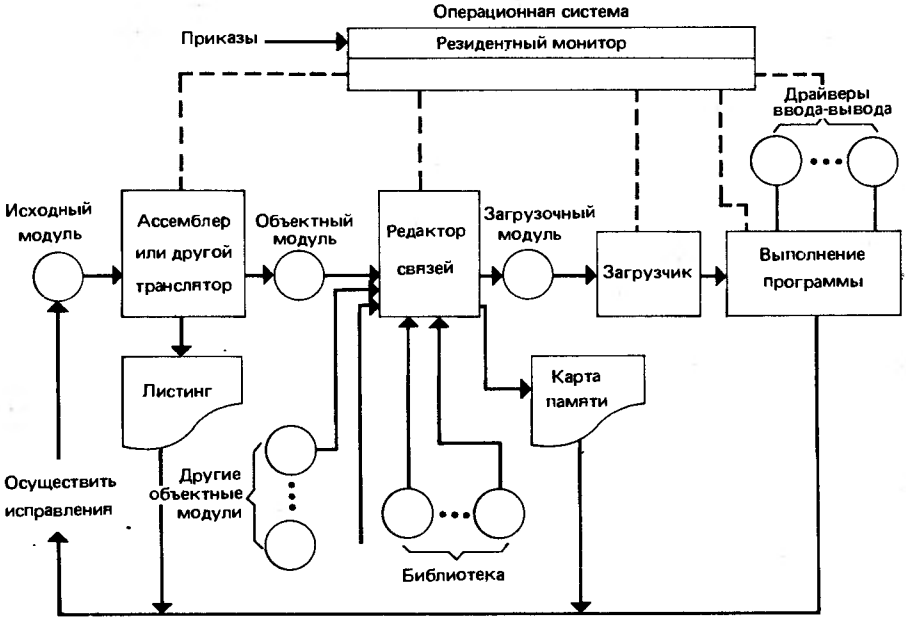


Рис. 4.2. Создание и выполнение программы

заполнить все те смещения, которые не смог определить ассемблер;
заполнить все сегментные адреса;
загрузить программу для выполнения.

Связываемые объектные модули определяются указанием их имен в приказе редактору связей и операционной системой, осуществляющей поиск в библиотеках. Формат приказа редактору связей зависит от системы, но обычно он содержит командное слово и последующий список объектных модулей и библиотек. Как правило, системную библиотеку не нужно указывать в списке, так как в ней автоматически отыскиваются необходимые объектные модули, которых нет в списке приказа.

Порядок следования объектных модулей в списке приказа может обусловливать порядок объединения их в загрузочный модуль, но конструкцию загрузочного модуля может определять и операционная система в соответствии с набором фиксированных правил. Первый вариант дает пользователю возможность управлять построением загрузочного модуля, но обычно пользователю точное упорядочивание в загрузочном модуле безразлично и действия по обработке деталей он возлагает на операционную систему. Объем и способ управления, предоставляемые пользователю, зависят от системы. В программном обеспечении MCS-86 пользователю разрешается подробно определять упорядочивание модулей и сегментов.

4.1.1. ОБЪЕДИНЕНИЕ СЕГМЕНТОВ

Кроме приказов, ассемблер ASM-86 позволяет редактору связей управлять организацией сегментов из различных объектных модулей. Иногда сегменты с одним и тем же именем сцепляются, а иногда перекрываются (накладываются). Объединением одноименных сегментов управляют модификаторы, содержащиеся в директивах SEGMENT. Директива SEGMENT имеет следующий формат:

Имя сегмента SEGMENT Тип объединения

Здесь тип объединения показывает, каким образом сегмент размещается в загрузочном модуле. Сегменты с различными именами объединять нельзя, а сегменты с одним и тем же именем, но без типа объединения заставляют редактор связей зафиксировать ошибку. Предусмотрены следующие типы объединений:

PUBLIC. Если сегменты из различных объектных модулей имеют одно и то же имя и тип объединения PUBLIC, в загрузочном модуле они сцепляются в один сегмент. Порядок сцепления (конкатенации) определяется приказом редактору связей.

COMMON. Если сегменты в различных объектных модулях имеют одно и то же имя и тип объединения COMMON, они перекрываются таким образом, что имеют один начальный адрес. Длина общего сегмента равна максимальной из длин объединяемых сегментов.

STACK. Когда сегменты в различных объектных модулях имеют одно и то же имя и тип объединения STACK, они превращаются в один сегмент, длина которого равна сумме длин отдельных сегментов. По существу, здесь сегменты объединяются для образования одного большого стека.

AT. Тип объединения AT сопровождается выражением, вычисление которого дает константу — сегментный адрес. Следовательно, пользователь может явно определить точное размещение сегмента в памяти.

MEMORY. Данный тип объединения вызывает размещение сегмента в конце загрузочного модуля. Если связываются несколько сегментов с типом объединения MEMORY, только первый из них считается имеющим тип объединения MEMORY, а остальные перекрываются так, как будто они имеют тип объединения COMMON.

(В директивах SEGMENT могут быть также модификаторы выравнивания, которые определяют выравнивание начального адреса сегмента. Однако в книге предполагается применение только выравнивания по умолчанию, когда сегмент начинается с адреса, кратного 16. Чтобы познакомиться с другими типами выравнивания, следует обратиться к соответствующим руководствам.)

На рис. 4.3 показано, как типы объединения PUBLIC и COMMON влияют на формирование загрузочного модуля. При необходимости объединения нескольких сегментов кода в один тип PUBLIC снимает необходимость изменения содержимого регистра CS при передаче управления в сегменте кода, т. е. позволяет заменить межсегментные переходы на внутрисегментные. Использование типа объединения PUBLIC для сегментов данных позво-

```
        ИСХОДНЫЙ МОДУЛЬ  1

DATA        SEGMENT         COMMON
              .
              .
              .
DATA        ENDS
CODE        SEGMENT         PUBLIC
              .
              .
              .
CODE        ENDS


        ИСХОДНЫЙ МОДУЛЬ  2

DATA        SEGMENT         COMMON
              .
              .
              .
DATA        ENDS
CODE        SEGMENT         PUBLIC
              .
              .
              .
CODE        ENDS
```

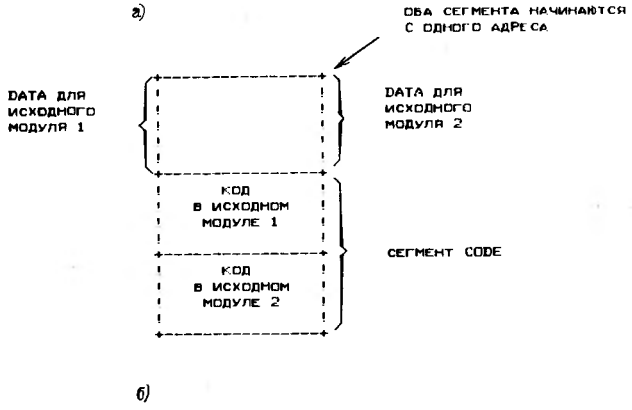


Рис. 4.3. Код (*а*) и загрузочный модуль (*б*) объединений сегментов, порождаемых типами объединения PUBLIC и COMMON

ляет объединить несколько наборов данных в единый бóльший набор. Тип объединения COMMON применяется для создания пула ячеек, распределяемых несколькими модулями (подробнее см. раздел 4.3.3).

Тип объединения STACK формирует один стек, разделяемый несколькими модулями, причем длина общего стека учитывает спецификации каждого модуля. Пример использования модификатора STACK приведен на рис. 4.4. Стеки подробно рассматриваются в § 4.2.

Тип объединения AT дает пользователю возможность определить точное начало сегмента кода или данных. Окончательные физические адреса переменных и меток, определяемых в сегментах без типа объединения AT, устанавливаются в процессе редактирования связей и загрузки. Такие переменные и метки называются *перемещаемыми*. С другой стороны, адреса перемен-

```
        исходный модуль  1

STACK_SEG       SEGMENT         STACK
                DW                  20 DUP (?)
    TOP_OF_STACK        LABEL       WORD
STACK_SEG       ENDS
                .
                .
                .
                END

        исходный модуль  2

STACK_SEG       SEGMENT         STACK
                DW                  30 DUP (?)
STACK_SEG       ENDS
                .
                .
                .
                END
```

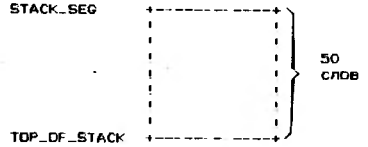


Рис. 4.4. Объединение стека из двух сегментов

ных и меток, определенных в сегментах с типом объединения AT, задаются программистом и ассемблером – они называются *абсолютными* переменными и метками. Модификатор AT часто применяется при программировании ввода-вывода, где участвуют специальные подпрограммы, буферы данных и другие области данных, к которым должны иметь непосредственный доступ множество программ.

Иногда программисту необходимо расположить один из сегментов по большим адресам, чем остальные сегменты. В таким случаях применяется тип объединения MEMORY.

4.1.2. ОБРАЩЕНИЯ К ВНЕШНИМ ИДЕНТИФИКАТОРАМ

Очевидно, связываемые объектные модули должны иметь возможность обращаться друг к другу, т. е. модуль может обращаться к некоторым переменным и(или) меткам из других модулей. Когда идентификатор определяется в объектном модуле, он называется *локальным* (или *внутренним*) относительно модуля; если же идентификатор определен не в данном модуле, а в одном из других связываемых модулей, он называется *внешним* (или *глобальным*) относительно модуля.

В одномодульных программах все идентификаторы, к которым имеются обращения, должны быть определенными локально, иначе ассемблер зафиксирует ошибку. В многомодульных программах ассемблер необходимо заранее информировать обо всех внешне определенных идентификаторах, которые появляются в модуле, чтобы ассемблер не считал их не определенными. Кроме того, чтобы разрешить другим объектным модулям обращаться к некоторым идентификаторам в данном модуле, он должен содержать список идентификаторов, к которым разрешено обращение. Таким образом, в каждом модуле появляются два списка: один содержит идентификаторы, к которым обращается данный модуль, а другой – локально определенные идентификаторы, к которым обращаются другие модули. Списки реализуются директивами EXTRN и PUBLIC со следующими форматами:

EXTRN Идентификатор:Тип, . . . , Идентификатор:Тип
PUBLIC Идентификатор, . . . , Идентификатор

Здесь идентификаторы представляют собой переменные или метки, объявляемые как внешние или как доступные другим модулям. Так как ассемблер для генерирования правильного машинного кода должен знать типы всех внешних идентификаторов, с каждым идентификатором в операторе EXTRN должен быть ассоциирован спецификатор типа. Типом переменной может быть BYTE, WORD или DWORD, а типом метки – NEAR или FAR. Если в операторе INC VAR1 переменная VAR1 типа WORD является внешней, модуль с этим оператором должен содержать директиву вида

EXTRN . . . , VAR1:WORD, . . .

а модуль, в котором определяется VAR1, должен иметь оператор вида

PUBLIC . . . , VAR1, . . .

Одна из главных задач редактора связей заключается в проверке того, что каждому идентификатору в операторе EXTRN соответствует точно один идентификатор в операторе PUBLIC. Если этого нет, существует не определенное внешнее обращение и редактор связей фиксирует ошибку. На рис. 4.5

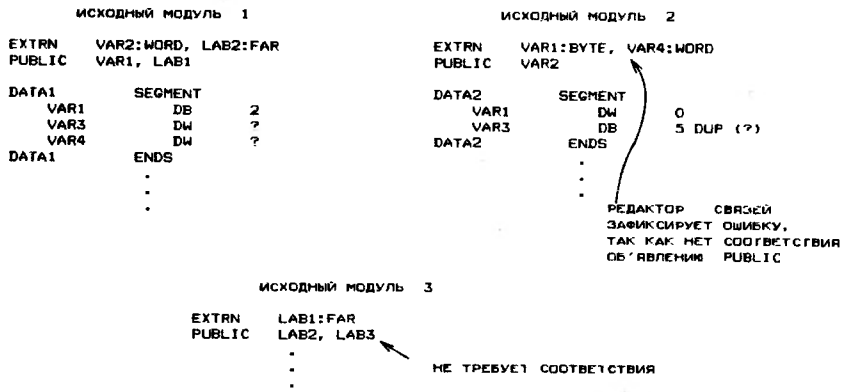


Рис. 4.5. Иллюстрация соответствия, проверяемого редактором связей

показаны три модуля и действия редактора связей по установлению соответствия при их объединении. В модуле 1 идентификаторы VAR1 и LAB1 являются локальными и объявлены глобальными, а VAR3 и VAR4 являются локальными, но не объявлены глобальными. Следовательно, редактор связей игнорирует VAR3 и VAR4. В модуле 1 имеются обращения к VAR2 и LAB2, которые не являются локально определенными, но доступны модулю 1, так как они объявлены PUBLIC в одном из других модулей. Модуль 2 содержит определение VAR2, а в модуле 3 определяется LAB2. Модуль 2 обращается к VAR1 и VAR4, которые определяются в модуле 1, но, так как VAR4 отсутствует в операторе PUBLIC, редактор связей зафиксирует ошибку. Неодно-

значности из-за определения VAR3 в модуле 2 и в модуле 1 не возникает, так как нет перекрестного обращения с привлечением VAR3. Определенная в модуле 2 метка LAB3 объявлена PUBLIC, но не требует соответствия; она просто не используется другими модулями, с которыми сейчас связывается модуль 3.

Как уже говорилось, каждый адрес состоит из двух частей: смещения и сегментного адреса. Смещения локальных идентификаторов может ввести и вводит ассемблер, но смещения внешних идентификаторов и все сегментные адреса должны вводиться в процессе редактирования связей. Смещения, ассоциируемые со всеми внешними обращениями, могут быть назначены, когда найдены все объектные модули и просмотрены их таблицы внешних имен. Назначение сегментных адресов называется перемещением и осуществляется после того, как в процессе редактирования точно определено место каждого сегмента в памяти. (В системе MCS-86 перемещение реализует распределитель LOC-86.)

Обратимся к программе на рис. 4.6. Первое слово в директиве DD и смещения VAR1 в команде DEC и POINTER в команде LES вводит ассемблер.

```
  EXTRN   LAB:FAR
DATA_SEG  SEGMENT
   POINTER     DD      VAR2
    VAR1       DW      ?
DATA_SEG  ENDS
EXTRA     SEGMENT
    VAR2       DW      ?
EXTRA     ENDS
CODE_SEG  SEGMENT
  ASSUME  CS:CODE_SEG, DS:DATA_SEG, ES:EXTRA
          MOV     AX,DATA_SEG
          MOV     DS,AX
                   .
                   .
                   .
          DEC     VAR1
          LES     DI,POINTER
          ADD     BX,ES:[DI]
                   .
                   .
                   .
          JMP     FAR PTR LAB
                   .
                   .
                   .
CODE_SEG  ENDS
```

Рис. 4.6. Программа, иллюстрирующая процесс введения адресов

Так как DATA_SEG является именем сегмента, режим адресации в команде MOV оказывается непосредственным, причем операндом служит сегментный адрес DATA_SEG. Этот адрес и второе слово в директиве DD должен вводить редактор связей. Кроме того, редактор связей должен заполнить смещение и сегментный адрес внешнего обращения в команде JMP.

Хотя информация ассемблеру о том, какие переменные являются внешними, необходима, одной ее недостаточно. Как и в случае локальных переменных, во время выполнения программы требуется динамически загружать правильные сегментные адреса. Если VAR – внешняя переменная, то до ис-

пользования ее командой в соответствующий сегментный регистр следует загрузить сегментный адрес VAR. Такое действие реализуется следующим образом:

```
        MOV     AX,SEG VAR      ;ПОМЕСТИТЬ АДРЕС СЕГМЕНТА
        MOV     ES,AX           ;ПЕРЕМЕННОЙ VAR В ES
                .
                .
                .
        ADD     DX,ES:VAR       ;КОМАНДА ИМЕЕТ ДОСТУП К VAR
```

Если до этих команд был оператор ASSUME, который ассоциировал сегментный адрес VAR с ES, префикс "ES:" в команде ADD можно удалить.

В качестве более наглядного примера рассмотрим три исходных модуля, представленные на рис. 4.7. Первые две команды MOV загружают в DS сегментный адрес LOCAL_DATA. Такая загрузка требуется даже в одномодульной программе. Следующие две команды MOV помещают в ES адрес сегмента, содержащего VAR1, поэтому следующая команда ADD правильно адресу-

```
                    ИСХОДНЫЙ МОДУЛЬ  1

  EXTRN   VAR1:WORD, OUTPUT:FAR
  EXTRN   VAR2:WORD
LOCAL_DATA  SEGMENT
                .
                .
                .
LOCAL_DATA  ENDS
CODE        SEGMENT
  ASSUME  CS:CODE, DS:LOCAL_DATA
    START:      MOV     AX,LOCAL_DATA
                MOV     DS,AX           ;ДОСТУП К LOCAL_DATA ЧЕРЕЗ DS
                    .
                    .
                    .
                MOV     AX,SEG VAR1     ;УСТАНОВИТЬ ES
                MOV     ES,AX           ;ДЛЯ ДОСТУПА К VAR1
                ADD     BX,ES:VAR1      ;ИСПОЛЬЗОВАТЬ VAR1
                    .
                    .
                    .
                MOV     AX,SEG VAR2     ;УСТАНОВИТЬ ES
                MOV     ES,AX           ;ДЛЯ ДОСТУПА К VAR2
                SUB     ES:VAR2,50      ;ИСПОЛЬЗОВАТЬ VAR2
                    .
                    .
                    .
                JMP     OUTPUT          ;ПОСЛЕДНЕЕ СЛОВО КОМАНДЫ
                    .                   ;СОДЕРЖИТ СЕГМЕНТНЫЙ АДРЕС ROUTINE
                    .
                    .
CODE        ENDS
                END  START
```

```
                    ИСХОДНЫЙ МОДУЛЬ  2

  PUBLIC  VAR1
EXTDATA1    SEGMENT
                .
                .
                .
    VAR1        DW      ?
                .
                .
                .
EXTDATA1    ENDS
                .
                .
                .
                END
```

```
                    ИСХОДНЫЙ МОДУЛЬ  3

  PUBLIC    VAR2
EXTDATA2    SEGMENT
                .
                .
                .
    VAR2:       DW      ?
                .
                .
                .
EXTDATA2    ENDS
                END
  PUBLIC    OUTPUT
ROUTINE     SEGMENT
  ASSUME    CS:ROUTINE
                .
                .
                .
    OUTPUT      LABEL   FAR
                .
                .
                .
ROUTINE     ENDS
                END
```

Рис. 4.7. Пример загрузки сегментных регистров из внешних обращений

ет VAR1. Третья пара команд MOV определяет (ES), что позволяет в команде SUB правильно адресовать VAR2. До выполнения команды JMP не нужны дополнительные команды для установки CS, так как новое содержимое CS является частью любой команды межсегментного перехода. Редактор связей помещает в последнее слово команды JMP сегментный адрес сегмента ROUTINE, который содержит метку OUTPUT, и при выполнении команды JMP происходит загрузка регистра CS. Отметим, что объявления внешних и глобальных идентификаторов в исходном модуле можно разнести по нескольким операторам EXTRN и PUBLIC.

Когда модуль обращается к нескольким переменным во внешнем сегменте и имя внешнего сегмента известно, имеется другой метод связывания, совместимый с установкой сегментных адресов локальных переменных с помощью оператора ASSUME. Этот метод заключается в определении сегмента с тем же именем, что и внешний сегмент, и указании его в операторах EXTRN с именами внешних переменных. Такой сегмент должен состоять только из операторов EXTRN, а сами операторы EXTRN должны содержать только имена переменных, обращения к которым находятся во внешнем сегменте. Кроме того, в его операторе SEGMENT необходимо указать тип объединения PUBLIC. Когда такой сегмент определен, имя сегмента и перечисленные в его операторах EXTRN переменные можно считать локальным именем сег-

```
                        ИСХОДНЫЙ МОДУЛЬ  1

GLOBAL      SEGMENT     PUBLIC
  EXTRN   VAR1:WORD, VAR2:WORD
GLOBAL      ENDS
LOCAL_DATA  SEGMENT
                .
                .
                .
LOCAL_DATA  ENDS
CODE        SEGMENT
  ASSUME  CS:CODE, DS:LOCAL_DATA, ES:GLOBAL
                .
                .
                .
                MOV     AX,GLOBAL       ;ИСПОЛЬЗОВАТЬ ES ДЛЯ ДОСТУПА
                MOV     ES,AX           ;К СЕГМЕНТУ GLOBAL
                MOV     BX,VAR1
                MOV     VAR2,BX
                .
                .
                .
CODE        ENDS
              END


                        ИСХОДНЫЙ МОДУЛЬ  2

GLOBAL      SEGMENT     PUBLIC
  PUBLIC  VAR1, VAR2
    VAR1        DW      ?
    VAR2        DW      ?
                .
                .
                .
GLOBAL      ENDS
                .
                .
                .
              END
```

Рис. 4.8. Доступ к нескольким внешним именам, определенным в одном и том же сегменте

мента и локальными переменными. Данный прием показан на рис. 4.8 для двух модулей. Отметим, что при указании ES:GLOBAL в операторе ASSUME префикс "ES:" в третьей и четвертой командах MOV не нужен.

4.2. СТЕКИ

Во многих ситуациях программе требуется временно запомнить информацию, а затем считывать ее в обратном порядке. Одним из примеров может служить запоминание и восстановление счетчиков при организации вложенных циклов. В микропроцессоре 8086 для реализации необходимых в циклах счета, проверки и перехода удобно применять регистр CX и команды циклов. Но так как команды циклов предназначены для использования только регистра CX, при вложении циклов возникает проблема "нехватки регистров". Если имеются эффективные средства запоминания счетчиков циклов по порядку с последующим восстановлением их путем извлечения последнего запомненного счетчика первым, то хотя бы часть указанных трудностей снимается (см. рис. 4.9).

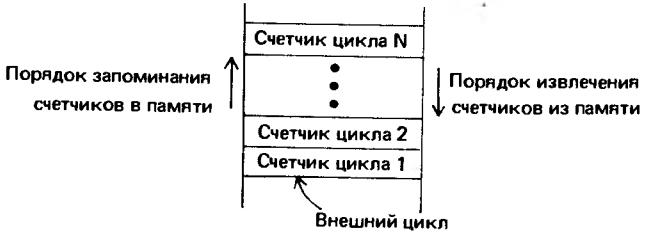


Рис. 4.9. Запоминание и восстановление счетчиков во вложенных циклах

Наиболее часто проблема временного хранения данных решается посредством реализации в компьютере *стека LIFO* ("последний пришел – первый ушел"), называемого также *стеком включения/извлечения*. Стеки применяются и для других манипуляций, но наиболее важное их использование связано с процедурами, которые рассматриваются в § 4.3. Стек обычно рассчитан на косвенную адресацию через специальный регистр – указатель стека; при включении элементов в стек производится автоматический декремент указателя стека, а при извлечении – инкремент. Помещение чего-либо в стек называется *включением* (push), а обратное действие – *извлечением* (pop). Адрес последнего включенного в стек элемента называется *вершиной стека* (TOS).

Команды микропроцессора 8086, предназначенные для прямого включения в стек и извлечения из стека, показаны на рис. 4.10. Стек оперирует только словами и не допускает непосредственных данных, но все остальные режимы адресации в командах PUSH и POP разрешены. На флажки воздействует только команда POPF, которая извлекает содержимое вершины стека в PSW. Отметим, что команды PUSH и POP являются единственными, кроме коман-

НАЗВАНИЕ	МНЕМОНИКА И ФОРМАТ		ОПИСАНИЕ
ВКЛЮЧИТЬ В СТЕК	PUSH	SRC	(SP) ← (SP)-2 ((SP)+1:(SP)) ← (SRC)
ИЗВЛЕЧЬ ИЗ СТЕКА	POP	DST	(DST) ← ((SP)+1:(SP)) (SP) ← (SP)+2
ВКЛЮЧИТЬ В СТЕК ФЛАЖКИ	PUSHF		(SP) ← (SP)-2 ((SP)+1:(SP)) ← (PSW)
ИЗВЛЕЧЬ ИЗ СТЕКА ФЛАЖКИ	POPF		(PSW) ← ((SP)+1:(SP)) (SP) ← (SP)+2

ФЛАЖКИ. ФЛАЖКИ МОДИФИЦИРУЕТ ТОЛЬКО КОМАНДА POPF.

РЕЖИМЫ АДРЕСАЦИИ. В КОМАНДАХ PUSH И POP ОПЕРАНД ДОЛЖЕН БЫТЬ СЛОВОМ И НЕ МОЖЕТ БЫТЬ НЕПОСРЕДСТВЕННЫМ ЗНАЧЕНИЕМ. КАК ОПЕРАНД ДОПУСКАЕТСЯ УКАЗЫВАТЬ СЕГМЕНТНЫЙ РЕГИСТР, НО В КОМАНДЕ POP РЕГИСТР CS УКАЗЫВАТЬ НЕЛЬЗЯ.

Рис. 4.10. Стековые команды

ды MOV, в которых допускается явно определять сегментный регистр как операнд. Однако в команде POP регистр CS указывать нельзя. Как показано на рис. 4.11, указатель стека SP показывает на вершину стека и слово включается в стек путем декремента (SP) на 2, так что он адресует следующее слово в памяти с меньшим адресом, и последующего запоминания операнда-источника по адресу, содержащемуся в SP. Извлечение из стека состоит в загрузке содержимого вершины стека в операнд-получатель с последующим инкрементом (SP) на 2. Следовательно, после одинакового числа включений и извлечений вершина стека оказывается в первоначальном положении. На рис. 4.12 показана организация вложенных циклов с помощью команд микропроцессора 8086.

Рис. 4.11. Стек микропроцессора 8086

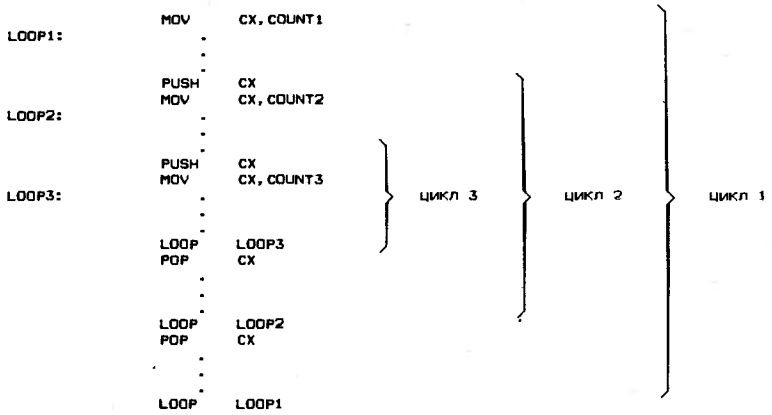


Рис. 4.12. Применение стека для вложения циклов

Физический адрес стека формируется из (SP) и (SS) или (BP) и (SS), причем SP служит неявным указателем стека для всех операций включения и извлечения, а SS – сегментным регистром стека. Содержимое SS является самым младшим адресом (т. е. границей) области стека и называется *базой стека*. Первоначальное содержимое SP считается наибольшим смещением, которого может достигать стек. Таким образом, стек занимает ячейки памяти от 16 × (SS) до 16 × (SS) плюс первоначальное (SP). Конечно, местоположение стека должно быть вначале задано программой – операционной системой или пользовательской программой, что реализуется загрузкой (SS) и (SP) в соответствии с рис. 4.13. Регистр BP предназначен, главным образом, для произвольных обращений к стеку. Например, команда MOV AX, [BP] [SI]

```
STACK_SEG   SEGMENT
              DW      30 DUP(?)
    TOS       LABEL   WORD
STACK_SEG   ENDS
CODE_SEG    SEGMENT
  ASSUME  CS:CODE_SEG, SS:STACK_SEG
    START:    MOV     AX,STACK_SEG
              MOV     SS,AX             ;ИНИЦИАЛИЗИРУЕТ SS
              MOV     SP,OFFSET TOS     ;ИНИЦИАЛИЗИРУЕТ SP
                .
                .
                .
CODE_SEG    ENDS
```

Рис. 4.13. Задание стека посредством загрузки регистров SS и SP

загружает в AX содержимое ячейки из сегмента стека, смещение которой равно сумме (BP) и (SI). С другой стороны, в команде MOV AX, [BX] [SI] операнд-источник находится в сегменте данных.

Стек эффективнее обычной памяти в двух отношениях. Команды PUSH и POP короче стандартных команд, так как один из операндов косвенно адресуется через регистр SP, а инкремент или декремент (SP) с образованием нового адреса производится автоматически. Например, регистры CX и DX мож-

но запомнить в памяти, начиная с SAVE, следующими командами, которые занимают 8 байт:

```
MOV  SAVE,CX
MOV  SAVE + 2,DX
```

Если смещение SAVE находится в SI, запоминание осуществляет последовательность

```
MOV  [SI],CX
INC  SI
INC  SI
MOV  [SI],DX
```

которая занимает 6 байт, но состоит из 4 команд. Для запоминания CX и DX в стеке требуются только команды

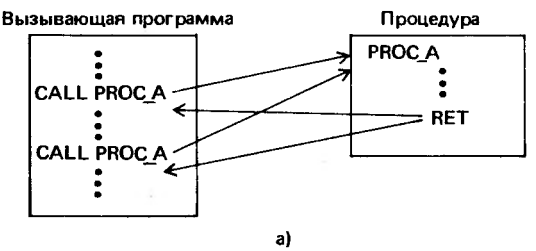
```
PUSH CX
PUSH DX
```

занимающие всего 2 байта. Восстановление CX и DX осуществляют команды

```
POP DX
POP CX
```

4.3. ПРОЦЕДУРЫ

Процедурой (или *подпрограммой*) называется код, к которому можно перейти и из которого можно возвратиться так, как будто код вводится в ту точку, из которой осуществляется переход. Переход к процедуре называется *вызовом*, а соответствующий переход назад называется *возвратом*. Возврат



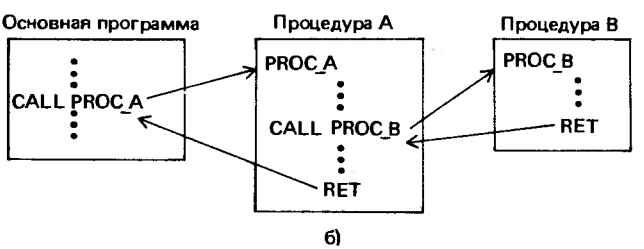


Рис. 4.14. Многократные вызовы процедур (*а*) и вложенные процедуры (*б*)

всегда производится к команде, находящейся сразу после вызова, независимо от того, где находится вызов. Если, как показано на рис. 4.14, *а*, к одной и той же процедуре делается несколько вызовов, возврат после каждого вызова осуществляется к команде, находящейся после данного вызова. Следовательно, в памяти нужно хранить только одну копию процедуры, даже если процедура вызывается несколько раз. При вложении процедур (см. рис. 4.14, *б*) каждый возврат производится в соответствующую вызывающую, а не в старшую по иерархии процедуру.

Процедуры обеспечивают основное средство разделения кода программы на модули. Хотя и не все модули являются процедурами, большинство из них все же будут ими, так как процедуры можно легко по отдельности разрабатывать, тестировать и документировать. Процедуры можно хранить в библиотеках и использовать множеством программ. Короче говоря, процедуры обеспечивают все удобства модульного программирования и делают это достаточно гибко. Основной недостаток процедур заключается в необходимости дополнительного кода для объединения их так, чтобы они могли взаимодействовать друг с другом. Этот дополнительный код называется *связыванием* процедур и о нем пойдет речь в данном разделе.

При вызове процедуры необходимо удовлетворить следующие три требования:

в отличие от других команд переходов вызов процедуры должен запомнить адрес следующей команды, чтобы можно было осуществить возврат в нужное место вызывающей программы;

используемые процедурой регистры необходимо запомнить до изменения их содержимого, а перед самым выходом из процедуры восстановить;

процедура должна иметь средства взаимодействия или разделения данных с вызвавшей ее программой и другими процедурами.

4.3.1. ВЫЗОВЫ, ВОЗВРАТЫ И ОПРЕДЕЛЕНИЯ ПРОЦЕДУР

Первое из приведенных выше требований удовлетворяется при наличии специальных команд вызова и возврата. Эти команды микропроцессора 8086 приведены на рис. 4.15. Команда CALL не только осуществляет переход по указанному адресу, но и включает в стек адрес возврата. Команда RET просто извлекает адрес возврата из стека. Предполагается, конечно, что, если процедура обращалась к стеку, указатель стека возвращен на правильное значение и действия со стеком не разрушили адрес возврата. Очевидно, при работе с процедурами программист должен внимательно следить за стеком, в противном случае возможны возвраты к бессмысленным ячейкам.

Режимы адресации команды CALL такие же, что и для команды JMP, но короткий вызов отсутствует. Вызов может быть прямым или косвенным, внутрисегментным или межсегментным. Если оператор вызова оказывается обращением вперед, перед операндом следует ввести атрибутный оператор NEAR PTR или FAR PTR, чтобы указать внутрисегментный прямой вызов или межсегментный прямой вызов соответственно. Если команда CALL является внутрисегментным (межсегментным) вызовом, то и команда RET

НАЗВАНИЕ	МНЕМОНИКА И ФОРМАТ		ОПИСАНИЕ
ВНУТРИСЕГМЕНТНЫЙ ПРЯМОЙ ВЫЗОВ	CALL	DST	(SP) ← (SP)-2 ((SP)+1:(SP)) ← (IP) (IP) ← (IP)+D16 *
ВНУТРИСЕГМЕНТНЫЙ КОСВЕННЫЙ ВЫЗОВ	CALL	DST	(SP) ← (SP)-2 ((SP)+1:(SP)) ← (IP) (IP) ← (EA)
МЕЖСЕГМЕНТНЫЙ ПРЯМОЙ ВЫЗОВ	CALL	DST	(SP) ← (SP)-2 ((SP)+1:(SP)) ← (CS) (SP) ← (SP)-2 ((SP)+1:(SP)) ← (IP) (IP) ← D16 (CS) ← СЕГМЕНТНЫЙ АДРЕС (ПОСЛЕДНЕЕ СЛОВО КОМАНДЫ)
МЕЖСЕГМЕНТНЫЙ КОСВЕННЫЙ ВЫЗОВ	CALL	DST	(SP) ← (SP)-2 ((SP)+1:(SP)) ← (CS) (SP) ← (SP)-2 ((SP)+1:(SP)) ← (IP) (IP) ← (EA) (CS) ← (EA+2)
ВНУТРИСЕГМЕНТНЫЙ ВОЗВРАТ	RET		(IP) ← ((SP)+1:(SP)) (SP) ← (SP)+2
ВНУТРИСЕГМЕНТНЫЙ ВОЗВРАТ С НЕПОСРЕДСТВЕННЫМИ ДАННЫМИ	RET	EXP***	АНАЛОГИЧНА ПРЕДЫДУЩЕЙ И ЕЩЕ (SP) ← (SP)+D16
МЕЖСЕГМЕНТНЫЙ ВОЗВРАТ	RET		(IP) ← ((SP)+1:(SP)) (SP) ← (SP)+2 (CS) ← ((SP)+1:(SP)) (SP) ← (SP)+2
МЕЖСЕГМЕНТНЫЙ ВОЗВРАТ С НЕПОСРЕДСТВЕННЫМИ ДАННЫМИ	RET	EXP **	АНАЛОГИЧНА ПРЕДЫДУЩЕЙ И ЕЩЁ (SP) ← (SP)+D16

* СМЕЩЕНИЕ МЕЖДУ НАЗНАЧЕНИЕМ И КОМАНДОЙ, НАХОДЯЩЕЙСЯ ПОСЛЕ КОМАНДЫ CALL.
** EXP - ВЫРАЖЕНИЕ, ВЫЧИСЛЕНИЕ КОТОРОГО ДАЕТ КОНСТАНТУ, ПРЕВРАЩАЮЩУЮСЯ В ЧАСТЬ D16 КОМАНДЫ.

ФЛАЖКИ. НИКАКИЕ ФЛАЖКИ НЕ МОДИФИЦИРУЮТСЯ.

РЕЖИМЫ АДРЕСАЦИИ. ЛЮБОЙ РЕЖИМ АДРЕСАЦИИ ПЕРЕХОДА, КРОМЕ КОРОТКОГО CALL.

Рис. 4.15. Команды вызовов и возвратов

должна быть внутрисегментным (межсегментным) возвратом. Это объясняется тем, что внутрисегментный вызов включает в стек только (IP), т. е. смещение адреса возврата, а межсегментный вызов должен включать в стек (IP) и (CS). Соответственно возврат должен извлекать из стека одно или два слова. В команде RET допускается необязательный 16-битный непосредственный операнд. Этот операнд, если он имеется, прибавляется к (SP) после извлечения из стека адреса возврата. Этим обеспечивается модификация вершины стека, применение которой рассматривается далее.

На рис. 4.16 показано действие стека при выполнении такой последовательности вызовов и возвратов:

1. Основная программа вызывает процедуру PRO_A типа FAR.
2. PRO_A вызывает процедуру PRO_B типа NEAR.
3. PRO_B вызывает процедуру PRO_C типа NEAR.
4. Осуществляется возврат в PRO_B.
5. Осуществляется возврат в PRO_A.
6. PRO_A вызывает процедуру PRO_C типа NEAR.

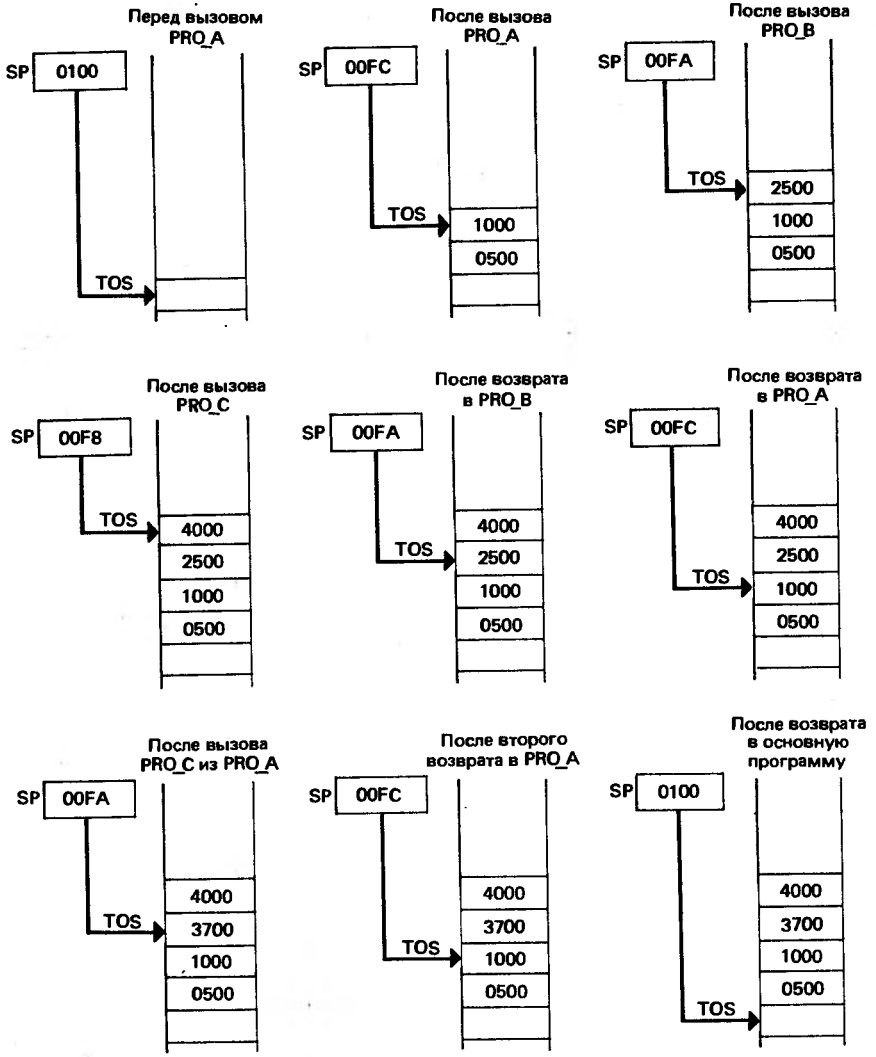


Рис. 4.16. Действия стека в типичной последовательности вызовов и возвратов

7. Осуществляется возврат в PRO_A.
8. Осуществляется возврат в основную программу.

Предполагается, что на рис. 4.16 стековые операции реализуются только для вызовов и возвратов и адреса возвратов имеют следующие значения:

Адрес возврата основной программы при вызове PRO_A (CS) = 0500 (IP) = 1000

Адрес возврата PRO_A при вызове PRO_B	(IP) = 2500
Адрес возврата PRO_B при вызове PRO_C	(IP) = 4000
Адрес возврата PRO_A при вызове PRO_C	(IP) = 3700

Отметим, что при извлечении из стека информация не разрушается, но она стирается последующими включениями, например когда PRO_A вызывает PRO_C.

Процедуры ограничиваются в исходном коде с помощью оператора PROC со следующим форматом

Имя процедуры PROC Атрибут

в начале процедуры и завершением процедуры оператором

Имя процедуры ENDP

Имя процедуры является идентификатором, используемым для вызова процедуры, а атрибутом служит NEAR или FAR. Атрибут необходим для определения типа ассемблируемой команды RET. В случае атрибута NEAR команда RET извлекает из стека одно слово в регистр IP, а когда указан атрибут FAR при возврате извлекается также слово в регистр CS.

Процедура может находиться:

в том же сегменте кода, что и вызывающий ее оператор;

в сегменте кода, отличающемся от сегмента, содержащего вызывающий ее оператор, но в том же исходном модуле, что и вызывающий оператор;

в исходном модуле и сегменте, которые отличаются от содержащих вызывающий оператор.

В первом случае атрибутом может быть NEAR, но при условии, что все вызовы находятся в том же сегменте кода, что и процедура. В последних двух случаях атрибутом должен быть FAR. Если процедуре придан атрибут FAR, все ее вызовы должны быть межсегментными, даже если вызов осуществляется из того же сегмента кода. В противном случае возврат извлекает из сте-

```
BEGX    SEGMENT
          .
          .
          .
SUBT    PROC    FAR
          .
          .
          .
          RET
SUBT    ENDP
          .
          .
          .
        CALL    FAR PTR SUBT
          .
          .
          .
BEGX    ENDS
SEGY    SEGMENT
          .
          .
          .
        CALL    FAR PTR SUBT
          .
          .
          .
SEGY    ENDS
```

Рис. 4.17. Объявление процедуры

ка два слова, хотя вызов включал в стек одно слово. Наконец, в третьем случае имя процедуры необходимо объявить в операторах EXTRN и PUBLIC.

На рис. 4.17 показана ситуация, в которой процедура SUBT определена в сегменте кода SEGX, а вызывается из двух сегментов кода SEGX и SEGY. Так как SUBT вызывается из обоих сегментов SEGX и SEGY, ей необходимо придать атрибут FAR. Поскольку SUBT придан атрибут FAR, вызовы ее даже в сегменте SYGX должны включать в стек (IP) и (CS), как и вызов из SEGY. Наконец, так как SUBT определена до ее вызовов, ассемблер может правильно транслировать вызовы, не пользуясь атрибутным оператором FAR PTR, но для большей ясности программы этот оператор указан.

4.3.2. ЗАПОМИНАНИЕ И ВОССТАНОВЛЕНИЕ РЕГИСТРОВ

Так как вызывающая программа и процедура разделяют одни и те же регистры, при вызове процедуры необходимо запомнить регистры, а перед возвратом в вызывающую программу восстановить их. Для запоминания и восстановления содержимого регистров, модифицируемого процедурой, применяют разнообразные способы, но при наличии стека именно его почти всегда используют для этой цели. На рис. 4.18, *а* приведен код, предназначенный для запоминания и восстановления содержимого регистров AX, BX, CX и DX, а на рис. 4.18, *б* показано, как ведет себя стек и указатель стека. Отметим, что содержимое регистров извлекается в обратном порядке.

Код на рис. 4.18, *а* осуществляет запоминание и восстановление внутри процедуры. Запоминание возможно произвести непосредственно перед вызовом процедуры, а восстановление — после возврата, но данный способ обычно не применяется. Когда эти задачи реализует процедура, необходимый для их выполнения код записывается только один раз. Если бы такой код был частью вызывающей программы, его пришлось бы повторять при каждом вызове.

Необходимо запоминать не все регистры, а только те из них, которые модифицирует процедура. Однако, если процедура не слишком мала и не под-

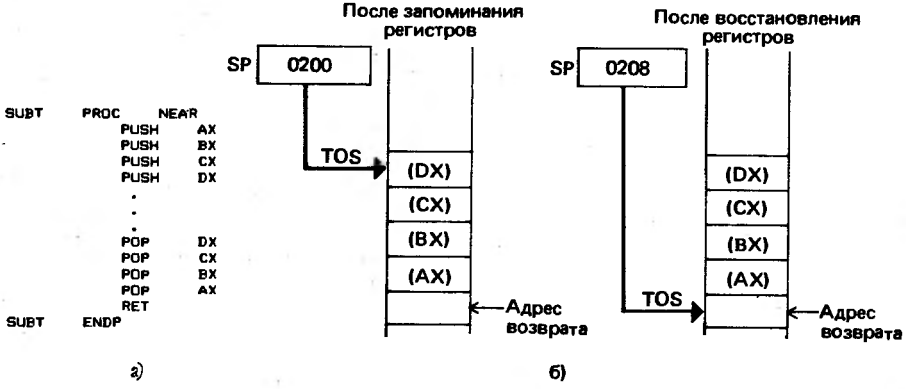


Рис. 4.18. Запоминание и восстановление регистров:
а — код; *б* — поведение стека

вержена изменениям, целесообразно автоматически запоминать все регистры из группы данных и, возможно, указательные, индексные и сегментные регистры. Если, например, регистр DX не запоминался, так как он не требовался, а позднее процедура была модифицирована и потребовался регистр DX, в процедуру пришлось вводить команды PUCH DX и POP DX. В противном случае после возврата в вызывающую программу содержимое DX будет неправильным. Очевидным исключением является случай, когда DX специально предназначен для возврата результата; конечно, здесь не нужно изменять (DX) командой POP.

4.3.3. ВЗАИМОДЕЙСТВИЕ ПРОЦЕДУР

Имеются два общих вида процедур, которые обрабатывают один и тот же набор данных и которые при каждом вызове обрабатывают различные наборы данных. Например, процедуру можно написать так, что она всегда суммирует COUNT элементов в массиве ARY и возвращает результат как переменную SUM, но процедуру можно написать и так, что она сможет суммировать элементы в произвольном массиве и возвратить результат как произвольную переменную.

Если процедура относится к первому виду и находится в том же исходном модуле, что и вызывающая программа, она может непосредственно обращаться к переменным, как показано в программе на рис. 4.19. Когда же процедура находится в отдельном исходном модуле, она может все-таки не-

```
DATA        SEGMENT
    ARY         DW      100 DUP(?)
    COUNT       DW      ?
    SUM         DW      ?
            .
            .
            .
DATA        ENDS
CODE        SEGMENT
            .
            .
            .
                CALL    NEAR PTR PROADD
            .
            .
            .
PROADD      PROC    NEAR
                PUSH    AX              ;ЗАПОМНИТЬ РЕГИСТРЫ
                PUSH    CX
                PUSH    SI
                LEA     SI,ARY          ;ПЕРЕДАТЬ АДРЕС ARY В SI
                MOV     CX,COUNT        ;ПЕРЕДАТЬ COUNT В CX
                XOR     AX,AX           ;СБРОСИТЬ AX
    NEXT:       ADD     AX,[SI]         ;СЛОЖИТЬ ПЕРВЫЕ COUNT
                ADD     SI,2            ;ЧИСЕЛ ИЗ ARY
                LOOP    NEXT
                MOV     SUM,AX          ;ЗАПОМНИТЬ РЕЗУЛЬТАТ В SUM
                POP     SI              ;ВОССТАНОВИТЬ РЕГИСТРЫ
                POP     CX
                POP     AX
                RET                     ;ВОЗВРАТ
PROADD      ENDP
            .
            .
            .
CODE        ENDS
```

Рис. 4.19. Процедура с прямым обращением к обрабатываемым переменным

посредственно обращаться к переменным при условии, что в вызывающей программе имеется директива

```
PUBLIC ARY,COUNT,SUM
```

а в исходном модуле процедуры – директива

```
EXTRN  ARY:WORD, COUNT:WORD, SUM:WORD
```

В другом варианте ARY, COUNT и SUM можно разместить в общей области, пользуясь кодом

```
DATA      SEGMENT  COMMON
   ARY            DW  100 DUP(?)
   COUNT          DW  ?
   SUM            DW  ?
DATA      ENDS
```

в начале обоих исходных модулей. Такой прием позволяет обоим исходным модулям разделять сегмент DATA. Если в вызывающей программе применяются другие имена переменных, например NUM вместо ARY, N вместо COUNT и TOTAL вместо SUM, сегмент DATA в ее исходном модуле необходимо определить следующим образом:

```
DATA        SEGMENT  COMMON
      NUM           DW  100 DUP(?)
      N             DW  ?
      TOTAL         DW  ?
DATA        ENDS
```

В любом случае при редактировании связей сегменты DATA совмещаются и первые 100 слов DATA содержат суммируемые числа, следующее слово содержит счетчик, а последнее слово предназначено для результата.

Независимо от использования прямого обращения к переменным или общей области, оба рассмотренные подхода взаимодействия с процедурой PROADD ограничены суммированием чисел из одной и той же области памяти при каждом вызове процедуры. Единственный способ сложения чисел из другого массива WEIGHTS заключается в передаче их в ARY (или NUM) и счетчика в COUNT (или N) до вызова процедуры, а после выполнения процедуры результат придется передать из SUM (или TOTAL) в нужную ячейку. На эти действия затрачивается значительное число команд и выполняются они долго, так как передаются 102 числа.

Решение заключается в том, чтобы передать процедуре адреса массива, счетчика и результата; тогда при каждом вызове процедура работает прямо с различными ячейками памяти. Несмотря на участие в обработке 102 чисел, необходимо передать всего три адреса, так как элементы массива размещены последовательно. Переменные, адреса которых передаются процедуре, называются *параметрами*. Имеется два основных способа передачи адресов параметров: построить таблицу (или массив) адресов параметров и передать адрес таблицы через регистр или включить адреса параметров в стек.

На рис. 4.20, *а* показана программа построения таблицы адресов параметров для процедуры PROADD, а на рис. 4.20, *б* — модифицированная версия PROADD, в которой необходимые адреса считываются из таблицы. Непосредственно перед вызовом PROADD смещения ARY, COUNT и SUM помещаются

```
PROG_SEG    SEGMENT
   ARY          DW      100 DUP(?)          ;ЗАРЕЗЕРВИРОВАТЬ 100 СЛОВ ДЛЯ ARY,
   COUNT        DW      ?                   ;ПО ОДНОМУ СЛОВУ ДЛЯ COUNT
   SUM          DW      ?                   ;И SUM
   TABLE        DW      3 DUP(?)            ;ЗАРЕЗЕРВИРОВАТЬ 3 СЛОВА
                                            ;ДЛЯ АДРЕСОВ ПАРАМЕТРОВ
                DW      100 DUP(?)          ;ЗАРЕЗЕРВИРОВАТЬ ОБЛАСТЬ
   TOS          LABEL   WORD                ;ДЛЯ СТЕКА
 ASSUME  CS:PROG_SEG, SS:PROG_SEG, DS:PROG_SEG
   START:       MOV     AX,CS
                MOV     DS,AX               ;ИНИЦИАЛИЗИРОВАТЬ DS
                MOV     SS,AX               ;ИНИЦИАЛИЗИРОВАТЬ SS
                MOV     SP,OFFSET TOS       ;ИНИЦИАЛИЗИРОВАТЬ SP
                 .
                 .
                 .
                MOV     TABLE,OFFSET ARY        ;ПЕРЕДАТЬ АДРЕСА ARY,
                MOV     TABLE+2,OFFSET COUNT    ;COUNT И SUM
                MOV     TABLE+4,OFFSET SUM      ;В ТАБЛИЦУ ПАРАМЕТРОВ
                MOV     BX,OFFSET TABLE         ;АДРЕС ТАБЛИЦЫ В BX
                CALL    FAR PTR PROADD          ;ВЫЗВАТЬ PROADD
                 .
                 .
                 .
PROG_SEG    ENDS
```

а)

```
PROADD      PROC    FAR
                PUSH    AX              ;ЗАПОМНИТЬ РЕГИСТРЫ
                PUSH    CX
                PUSH    SI
                PUSH    DI
                MOV     SI,[BX]         ;ЗАГРУЗИТЬ В РЕГИСТРЫ
                MOV     DI,[BX+2]       ;АДРЕС МАССИВА,
                MOV     CX,[DI]         ;ЗНАЧЕНИЕ СЧЕТЧИКА
                MOV     DI,[BX+4]       ;И АДРЕС РЕЗУЛЬТАТА
                XOR     AX,AX           ;СБРОСИТЬ AX
   NEXT:        ADD     AX,[SI]         ;СЛОЖИТЬ ЭЛЕМЕНТЫ МАССИВА
                ADD     SI,2            ;В AX
                LOOP    NEXT
                MOV     [DI],AX         ;ВОЗВРАТИТЬ РЕЗУЛЬТАТ
                POP     DI              ;ВОССТАНОВИТЬ РЕГИСТРЫ
                POP     SI
                POP     CX
                POP     AX
                RET                     ;ВОЗВРАТ
PROADD      ENDP
```

б)

Рис. 4.20. Вызывающая программа (*а*) и процедура (*б*) при передаче параметров через таблицу адресов

в массиве TABLE, а адрес TABLE загружается в BX. После вызова PROADD адреса ARY и SUM берутся из TABLE и передаются в SI и DI, а значение COUNT передается в CX. Прежнее содержимое этих регистров и AX, который служит аккумулятором, запоминается в стеке в начале процедуры и восстанавливается перед выходом из нее. Процедура может находиться в том же сегменте кода, что и вызов, в другом сегменте кода или в другом исходном модуле. В приведенном примере данные, стек и вызывающая программа объединены в один и тот же сегмент. Отметим, что до оператора вызова регистр DS должен адресовать сегмент данных, в котором находятся параметры и таблица адресов параметров.

Передача адресов параметров через стек иллюстрируется на рис. 4.21, причем на рис. 4.21, *а* приведена вызывающая последовательность, а на рис. 4.21, *б* — процедура. Чтобы показать другой подход, в вызывающей программе данные, стек и код размещены в отдельных сегментах. Как и в предыдущем примере, до вызова PROADD регистр DS должен адресовать сег-

```
PARM_SEG    SEGMENT
    ARY         DW      100 DUP(?)
    COUNT       DW      ?
    SUM         DW      ?
PARM_SEG    ENDS
STACK_SEG   SEGMENT
            DW      100 DUP(?)
    TOS     LABEL   WORD
STACK_SEG   ENDS
CODE1       SEGMENT
  ASSUME  CS:CODE1, DS:PARM_SEG, SS:STACK_SEG
    START:  MOV     AX,PARM_SEG
            MOV     DS,AX           ;УСТАНОВИТЬ DS
            MOV     AX,STACK_SEG
            MOV     SS,AX           ;УСТАНОВИТЬ SS
            MOV     SP,OFFSET TOS   ;УСТАНОВИТЬ SP
              .
              .
              .
            MOV     BX,OFFSET ARY           ;ВКЛЮЧИТЬ АДРЕС ARY В СТЕК
            PUSH    BX
            MOV     BX,OFFSET COUNT         ;ВКЛЮЧИТЬ АДРЕС COUNT В СТЕК
            PUSH    BX
            MOV     BX,OFFSET SUM           ;ВКЛЮЧИТЬ АДРЕС SUM В СТЕК
            PUSH    BX
            CALL    FAR PTR PROADD
              .
              .
              .
CODE1       ENDS
```

а)

```
CODE2       SEGMENT
  ASSUME  CS:CODE2
    STACK_STRC  STRUC
        SAVE_BP     DW      ?
        SAVE_CS_IP  DW      2 DUP(?)
        PAR3_ADDR   DW      ?
        PAR2_ADDR   DW      ?
        PAR1_ADDR   DW      ?
    STACK_STRC  ENDS
PROADD      PROC    FAR
            PUSH    BP                  ;ЗАПОМНИТЬ BP
            MOV     BP,SP               ;ДОСТУП К ПАРАМЕТРАМ ЧЕРЕЗ BP
            PUSH    AX                  ;ЗАПОМНИТЬ ДРУГИЕ РЕГИСТРЫ
            PUSH    CX
            PUSH    SI
            PUSH    DI
            MOV     SI,[BP].PAR1_ADDR   ;ВЗЯТЬ АДРЕС PAR1 ИЗ СТЕКА
            MOV     DI,[BP].PAR2_ADDR   ;ЗАГРУЗИТЬ В CX
            MOV     CX,[DI]             ;ЗНАЧЕНИЕ PAR2
            MOV     DI,[BP].PAR3_ADDR   ;ВЗЯТЬ АДРЕС PAR3 ИЗ СТЕКА
            XOR     AX,AX               ;СБРОСИТЬ AX
    NEXT:   ADD     AX,[SI]             ;ВЫЧИСЛИТЬ СУММУ
            ADD     SI,2
            LOOP    NEXT
            MOV     [DI],AX             ;ЗАПОМНИТЬ РЕЗУЛЬТАТ
            POP     DI                  ;ВОССТАНОВИТЬ РЕГИСТРЫ
            POP     SI
            POP     CX
            POP     AX
            POP     BP
            RET     6                   ;СКОРРЕКТИРОВАТЬ СТЕК
PROADD      ENDP
CODE2       ENDS
```

б)

Рис. 4.21. Вызывающая программа (*а*) и процедура (*б*) при передаче параметров через стек

мент, в котором находятся параметры. Для лучшего доступа к параметрам в модуле с процедурой верхняя часть стека определена как структура и в операндах применяются имена элементов структуры. (Поскольку вызывающая программа резервирует пространство, структуру не требуется вызывать. Она используется только как средство придания содержательных имен ее элементам в исходном коде.) Для индексирования элементов в структуре применяется регистр BP. Отметим, что команда RET корректирует указатель стека, поэтому после возврата в вызывающую программу TOS будет в том же месте, что и перед включением в стек адресов параметров. На рис. 4.22 показано содержимое стека при выполнении процедуры PROADD. После извлечения содержимого регистров TOS содержит старое (IP), а после выполнения команды RET она будет той же самой, что и TOS непосредственно перед началом вызывающей последовательности.

Хотя ранее предполагалось, что вместо параметров всегда передаются адреса параметров, это не всегда так. Когда входной параметр (в противоположность результату) имеет длину в один байт или одно слово, проще поместить в таблицу параметров или в стек сам параметр. В приведенных примерах можно передавать значение COUNT, а не его адрес. Однако, так как программист процедуры должен точно знать, как передается необходимая информация, целесообразно установить среди программистов соглашение и следовать ему всегда, даже если соглашение ведет к менее эффективному коду. Соблюдение всегда одного и того же метода часто оказывается небольшой жертвой в пользу большей ясности всей программы. При программировании

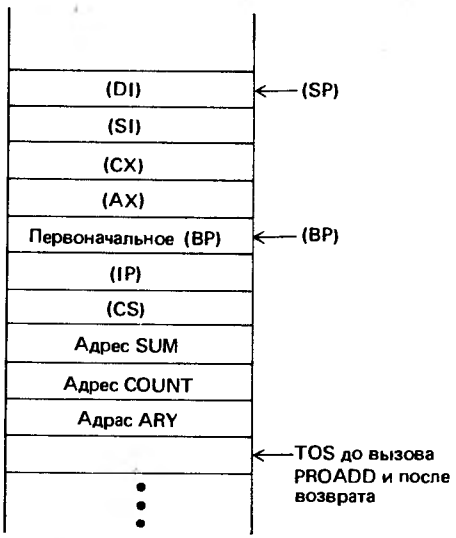


Рис. 4.22. Содержимое стека при выполнении процедуры PROADD

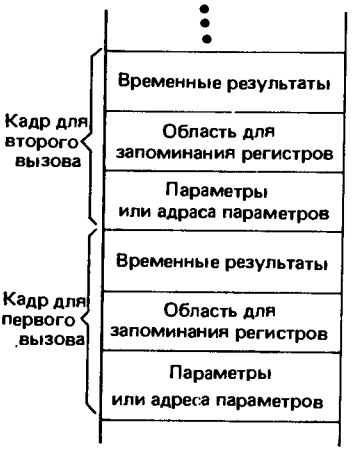


Рис. 4.23. Применение стека для организации динамической памяти в рекурсивных вызовах

процедур широкого назначения выполнение соглашения является обязательным.

4.3.4. РЕКУРСИВНЫЕ ПРОЦЕДУРЫ

Иногда алгоритм определяется в собственных терминах; другими словами, алгоритм вкладывается сам в себя таким образом, что вычислительная процедура вызывает саму себя. Например, алгоритм Горнера вычисления полинома

$$((\ldots(a_n x + a_{n-1})x + a_{n-2})x + \ldots)x + a_1)x + a_0$$

включает в себя перемножение x и a_n, прибавление a_{n-1} к результату, умножение на x полученного результата, прибавление a_{n-2} к новому результату и т. д. Следующий результат получается путем умножения предыдущего результата на x и прибавления очередного коэффициента. Однократное применение алгоритма состоит в умножении и сложении, а окончательный результат получается вложением простого алгоритма умножения-сложения в самого себя. Такие алгоритмы называются *рекурсивными* и иногда требуются в программных системах для поиска, декодирования синтаксиса языка высокого уровня и решения других задач.

Рекурсивные алгоритмы допускают реализацию в виде процедуры, которая сама себя вызывает, но при этом необходимо обеспечить, чтобы каждый последовательный вызов не разрушал параметров и результатов, полученных предыдущим вызовом, и чтобы процедура не модифицировала сама себя. Это означает, что каждый вызов должен запоминать свой набор параметров, регистры и все промежуточные результаты в различных областях памяти. Чтобы гарантировать использование таких отдельных областей памяти, обычно применяется стек. При включении данных в стек, как показано на рис. 4.23, их можно удобно считывать в обратном порядке по мере того, как процесс выходит из вложений. Запоминаемые при организации процедуры данные называются *кадром*, и простой доступ к отдельным элементам в кадре реализуется через регистр BP. Обычно кадр хранит запомненное содержимое регистров, адреса параметров и рабочую (временную) область памяти. Правильное запоминание информации в стеке обеспечивает упорядоченный выход. Очевидно, для предотвращения бесконечного вложения рекурсивный алгоритм должен включать в себя условный переход, который в конце концов прекращает вложение.

Тривиальным примером рекурсивной процедуры служит процедура вычисления факториала. Оно реализуется рекурсивной процедурой FACT следующим образом:

```
BEGIN FACT(N,RESULT)
        SAVE РЕГИСТРЫ В СТЕКЕ
IF N=0 THEN
        RESULT ⟵ 1
      ELSE
        PUSH АДРЕС RESULT В СТЕК
        PUSH N-1 В СТЕК
        CALL FACT(N-1,RESULT)
        RESULT ⟵ N*RESULT
ENDIF
        RESTORE РЕГИСТРЫ ИЗ СТЕКА
        DELETE ПАРАМЕТРЫ ИЗ СТЕКА
RETURN
```

Параметры N и RESULT представляют собой отыскиваемый факториал и результат. Программа, реализующая данный алгоритм, приведена на рис. 4.24. Хотя в описании ранее указано только включение и извлечение параметров, в программе имеются и другие необходимые стековые операции. Для определения текущего стекового кадра применяется структура и доступ к кадру организован через регистр BP.

Хотя использование рекурсивной процедуры в простой задаче вычисления факториала малоэффективно и не оправданно, пример показывает сущность рекурсивного программирования. Более обоснованные применения можно

```
CODE        SEGMENT
    FRAME       STRUC
        SAVE_BP     DW      ?
        SAVE_CS_IP  DW      2 DUP(?)
        N           DW      ?
        RESULT_ADDR DW      ?
    FRAME       ENDS
  ASSUME  CS:CODE
FACT        PROC    FAR
                PUSH    BP                      ;ЗАПОМНИТЬ BP
                MOV     BP,SP                   ;BP АДРЕСУЕТ FRAME
                PUSH    BX                      ;ЗАПОМНИТЬ ИСПОЛЬЗУЕМЫЕ
                PUSH    AX                      ;РЕГИСТРЫ
                MOV     BX,[BP].RESULT_ADDR     ;АДРЕС РЕЗУЛЬТАТА В BX
                MOV     AX,[BP].N
                CMP     AX,0                    ;N=0?
                JE      DONE
                PUSH    BX                      ;АДРЕС ДЛЯ СЛЕДУЮЩЕГО ВЫЗОВА
                DEC     AX                      ;ВКЛЮЧИТЬ В СТЕК N-1
                PUSH    AX                      ;ДЛЯ СЛЕДУЮЩЕГО ВЫЗОВА
                CALL    FAR PTR FACT
                MOV     BX,[BP].RESULT_ADDR
                MOV     AX,[BX]
                MUL     [BP].N                  ;(AX) = N*RESULT
                JMP     SHORT RETURN
    DONE:       MOV     AX,1                    ;(AX) = 1
    RETURN:     MOV     [BX],AX                 ;RESULT = (AX)
                POP     AX
                POP     BX
                POP     BP
                RET     4
FACT        ENDP
CODE2       ENDS
```

Рис. 4.24. Рекурсивная процедура вычисления факториала

найти в программных системах. Кроме того, многие из рассмотренных понятий потребуются при обсуждении в гл. 7 реентрантных процедур.

4.4. ПРЕРЫВАНИЯ И ПРОЦЕДУРЫ ПРЕРЫВАНИЙ

Иногда необходимо заставить компьютер автоматически выполнять одну из набора специальных программ, когда в программе или в вычислительной системе возникают определенные условия. Действие, стимулирующее выполнение одной из таких программ, называется *прерыванием*, а выполняемая программа — *процедурой прерывания*. Существуют два общих класса прерываний и связанных с ними программ: *внутренние* инициируются состоянием ЦП или командой, а *внешние* — сигналом, подаваемым в ЦП от других компонент системы. Типичные внутренние прерывания инициируются при делении на нуль или при выполнении специальной команды, а типичные внешние — когда устройство ввода-вывода требует обслуживания от ЦП. Так как

внешние прерывания в основном связаны с вводом-выводом, они подробно рассмотрены в гл. 6.

Процедура прерывания аналогична процедуре в том отношении, что переход к ней осуществляется из любой другой программы, а после выполнения процедуры прерывания возврат происходит в прерванную программу. Запись процедуры прерывания должна быть такой, чтобы прерванная программа продолжалась так, как будто ничего не произошло (кроме, разумеется, потери времени). Это означает, что необходимо запоминать и восстанавливать PSW и регистры, используемые процедурой прерывания, и что возврат должен происходить к команде, следующей за последней командой, выполненной до прерывания. Процедура прерывания отличается от процедуры тем, что вместо связывания с конкретной программой, она иногда размещается в фиксированной области памяти (т. е. размещена абсолютно). Не связанная с остальными сегментами, процедура прерывания может использовать для взаимодействия с другими программами только общие области, также размещенные абсолютно. Так как некоторые виды прерываний инициируются внешними событиями, они возникают в произвольных точках прерываемой программы. При этом процедуре прерывания невозможно передать параметры или адреса параметров и взаимодействие по данным должно происходить через переменные, которые непосредственно доступны обеим программам.

Независимо от вида прерывания возникающие при этом действия одинаковы и называются *последовательностью прерывания*. Последовательность прерывания в микропроцессоре 8086 показана на рис. 4.25. Некоторыми видами прерываний управляют флажки IF и TF, которые для восприятия прерываний должны быть правильно установлены. Если условия для прерывания удовлетворяются и необходимые флажки установлены, микропроцессор завершает текущую команду, а затем реализует последовательность прерывания: текущее содержимое PSW, CS и IP включается в стек, в IP и CS помещается новое содержимое из двойного слова, адрес которого определяется

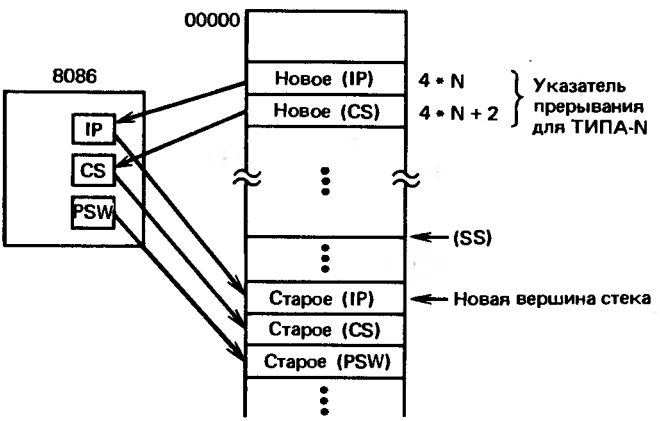


Рис. 4.25. Последовательность прерывания

типом прерывания, и флажки IF и TF сбрасываются. Новое содержимое IP и CS определяет начальный адрес выполняемой процедуры прерывания. После обслуживания прерывания возврат в прерванную программу осуществляется командой, которая извлекает из стека IP, CS и PSW.

Двойное слово, в котором находится новое содержимое IP и CS, называется *указателем прерывания* (или *вектором*). Каждому типу прерывания назначено число из диапазона 0 . . . 255 и адрес указателя прерывания находится путем умножения типа на 4. Если, например, тип равен 9, то указатель прерывания содержится в байтах 00024 . . . 00027. Для хранения двойного слова требуются 4 байта, поэтому указатели прерываний могут занимать первые 1024 байта памяти и их никогда нельзя использовать для других целей. Некоторые из 256 типов прерываний резервируются операционной системой и должны инициализироваться после включения компьютера. Пользователи

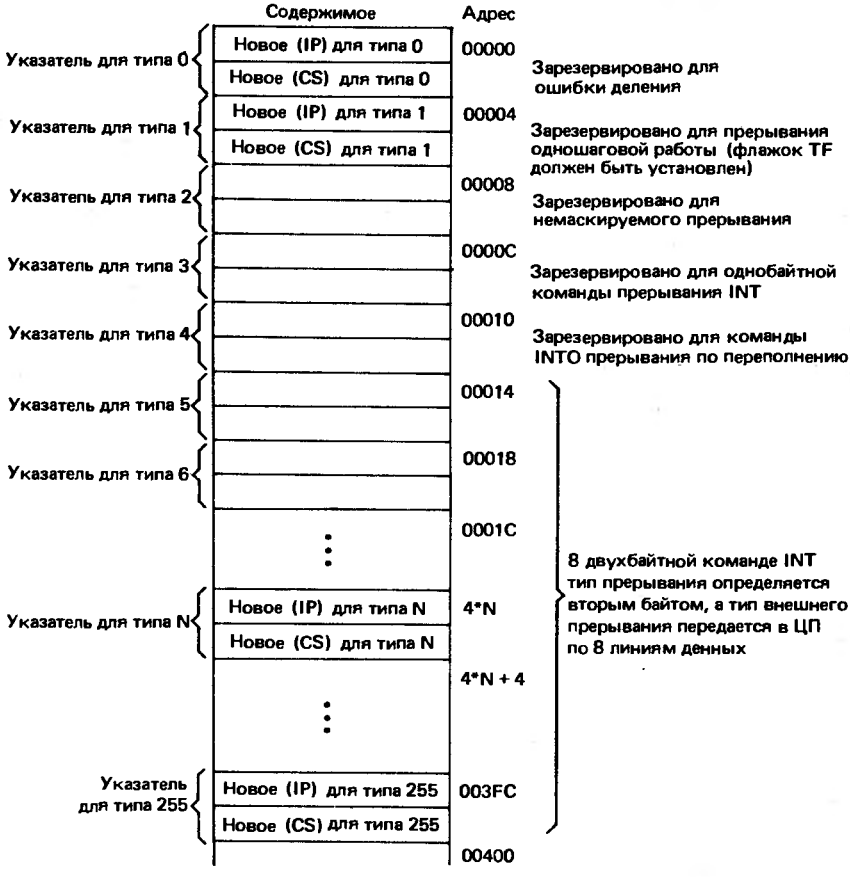


Рис. 4.26. Организация указателей прерываний

приспосабливают остальные типы прерываний в соответствии со своими требованиями.

Виды прерываний и соответствующие им типы представлены на рис. 4.26, показывающем размещение указателей в памяти. Только первые пять типов определены явно, а остальные отведены для команд прерываний или внешних прерываний. Из рисунка видно, что прерывание типа 0 связано с ошибкой деления. Следовательно, при попытке деления на нуль компьютер включает текущее содержимое PSW, CS и IP в стек, загружает регистры IP и CS из адресов 00000 и 00002 и продолжает выполнение с адреса, определяемого новым содержимым IP и CS. Прерывание из-за ошибки деления возникает, когда в командах DIV или IDIV частное превышает диапазон независимо от состояния флажков IF и TF.

Прерывание типа 1 обеспечивает одношаговую работу и только им управляет флажок TF. Если флажок TF разрешен, по окончании следующей команды возникает прерывание, которое вызывает переход к ячейке, адресуемой содержимым 00004 ... 00007. Так как в последовательности прерывания флажок TF сбрасывается, прерывания в процедуре прерывания не возникают, но при возврате восстанавливается исходное PSW (включая и разрешенный флажок TF) и сразу после команды, выполненной за возвратом, появится прерывание. Следовательно, до тех пор, пока флажок TF установлен, прерывание возникает после каждой команды программы. Одношаговое прерывание используется главным образом при отладке, причем процедура прерывания печатает после каждой команды содержимое важных регистров и ячеек памяти. Программист получает при этом "фотоснимок" программы после выполнения каждой команды. Флажок TF можно разрешить посредством включения в стек, объединения по ИЛИ вершины стека с 0100 и извлечения из стека в PSW. Флажок TF запрещается аналогично, но путем объединения по И содержимого PSW и FEFF.

Прерывание типа 2 относится к немаскируемому внешнему прерыванию и воспринимается независимо от состояния флажка IF. Они инициируется сигналом, подаваемым на вход NMI, и рассматривается в гл. 6.

Остальные типы прерываний соответствуют либо командам прерываний, включенным в прерываемую программу, либо внешним прерываниям. Команды прерываний приведены на рис. 4.27 и их прерываниями флажок IF не управляет. Маскируемые внешние прерывания инициируют последовательность прерывания, если только IF = 1 (подробнее см. гл. 6). Инициирующие их команды иногда называются *программными прерываниями*. На рис. 4.27 дано также определение команды IRET, которая осуществляет возврат из процедуры прерывания. Она аналогична команде RET, но кроме адреса возврата извлекает из стека исходное содержимое PSW.

Команда INT имеет два формата: INT или INT *Тип*. Если *Тип* отсутствует, команда оказывается однобайтной и имеет тип 3; в противном случае длина команды 2 байта и указатель прерывания начинается по адресу 4 * *Тип*.

Команда INT также часто применяется при отладке, когда одношаговая работа дает больше информации, чем требуется. Помешая команды INT в

НАЗВАНИЕ	МНЕМОНИКА И ФОРМАТ		ОПИСАНИЕ
ПРЕРЫВАНИЕ С ТИПОМ *	INT	TYPE	(SP) ← (SP)-2 ((SP)+1:(SP)) ← (PSW) (SP) ← (SP)-2 ((SP)+1:(SP)) ← (CS) (SP) ← (SP)-2 ((SP)+1:(SP)) ← (IP) (IP) ← (TYPE*4) (CS) ← (TYPE*4+2)
ОДНОБАЙТНОЕ ПРЕРЫВАНИЕ *	INT		АНАЛОГИЧНА ПРЕДЫДУЩЕЙ, КРОМЕ TYPE = 3
ПРЕРЫВАНИЕ ПРИ ПЕРЕПОЛНЕНИИ	INTO		ЕСЛИ (OF)=1, (SP) ← (SP)-2 ((SP)+1:(SP)) ← (PSW) (SP) ← (SP)-2 ((SP)+1:(SP)) ← (CS) (SP) ← (SP)-2 ((SP)+1:(SP)) ← (IP) (IP) ← (10H) (CS) ← (12H)
ВОЗВРАТ ИЗ ПРЕРЫВАНИЯ	IRET		(IP) ← ((SP)+1:(SP)) (SP) ← (SP)+2 (CS) ← ((SP)+1:(SP)) (SP) ← (SP)+2 (PSW) ← ((SP)+1:(SP)) (SP) ← (SP)+2

* ЭТИ ПРЕРЫВАНИЯ НЕ УПРАВЛЯЮТСЯ ФЛАЖКАМИ IF И TF.

ФЛАЖКИ. ЗА ИСКЛЮЧЕНИЕМ КОМАНДЫ IRET ФЛАЖКИ IF И TF СБРАСЫВАЮТСЯ ПОСЛЕДОВАТЕЛЬНОСТЬЮ ПРЕРЫВАНИЯ, А ОСТАЛЬНЫЕ ФЛАЖКИ НЕ МОДИФИЦИРУЮТСЯ. КОМАНДА IRET ВОЗДЕЙСТВУЕТ НА ВСЕ ФЛАЖКИ ПОСРЕДСТВОМ ЗАГРУЗКИ ИХ ИЗ СТЕКА.

РЕЖИМЫ АДРЕСАЦИИ. ОТСУТСТВУЮТ ЗА ИСКЛЮЧЕНИЕМ КОМАНДЫ INT TYPE, В КОТОРОЙ TYPE ДОЛЖЕН БЫТЬ ВЫРАЖЕНИЕМ-КОНСТАНТОЙ СО ЗНАЧЕНИЕМ ИЗ ДИАПАЗОНА 0...255.

Рис. 4.27. Команды прерываний

ключевых точках (называемых *контрольными*) программы, программист может использовать процедуру прерывания для печати в этих точках сообщений и другой информации. Если во всех контрольных точках выводится одна и та же информация, целесообразно воспользоваться однобайтной командой INT, но если информация различна, в команде придется указывать тип. Применение двухбайтной команды INT для отладки показано на рис. 4.28.

Команда INTO имеет тип 4 и вызывает прерывание, если только флажок OF = 1. Ее часто помещают сразу после арифметической команды и при нали-

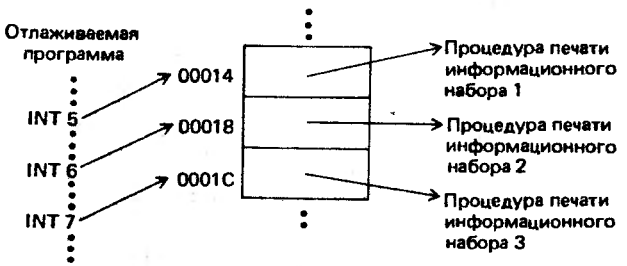


Рис. 4.28. Применение команды INT для отладки программы

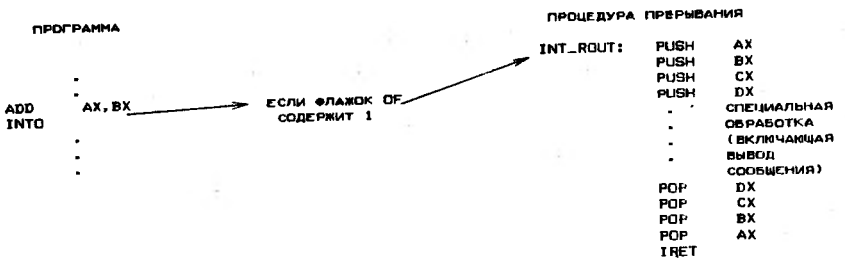


Рис. 4.29. Применение команды прерывания для переполнения

чии переполнения осуществляется специальная обработка. В отличие от деления на нуль переполнение не вызывает автоматического прерывания; прерывание необходимо явно определять командой INTO. Типичное применение команды INTO показано на рис. 4.29. Отметим, что PSW запоминать не нужно, так как его включает в стек последовательность прерывания.

4.5. МАКРОКОМАНДЫ

Достоинства процедур заключаются в том, что они экономят память и время программирования благодаря многократному использованию кода и обеспечивают модульность, упрощающую отладку и модификацию программы. Недостаток процедур объясняется необходимостью их обязательного связывания; иногда для программирования связей процедуры требуется более длинный код, чем для решения задачи. В этом случае процедура не экономит память, а время выполнения значительно увеличивается. Для ситуаций, в которых аналогичные, но короткие фрагменты кода появляются в нескольких местах программы, необходимо средство, обеспечивающее простоту программирования процедур, но позволяющее избежать сложностей связей. Таким средством оказываются макрокоманды.

Макрокомандой называется фрагмент кода, который необходимо записать только один раз, но базовую структуру которого можно повторить в исходном модуле несколько раз, помещая в точке каждого повторения всего один оператор. Макрокоманда отличается от процедуры тем, что машинные команды повторяются при каждом ее вызове; следовательно, макрокоманды не экономят память, но сокращают время программирования (связи не требуются) и обеспечивают некоторую степень модульности.

Повторяемый код называется *кодом прототипа*, а код прототипа вместе с операторами для его вызова и завершения называется *макроопределением*. В первом операторе макроопределения находится имя макрокоманды. Сущность использования макрокоманд заключается в построении макроопределения и последующем введении макрокоманды в различных точках программы с помощью оператора, содержащего имя макрокоманды. Эти операторы называются *макровызовами*. Когда ассемблер встречает макровызов, он заменяет вызов кодом макрокоманды, а само действие замены называется *макрорасширением*.

Чтобы обеспечить применение кода прототипа во многих ситуациях, части операторов в этом коде (например, операнды) представлены так, что их при каждом расширении макрокоманды можно заменить различными символьными цепочками. Такие части оператора называются *фиктивными параметрами* и указываются в первом операторе макроопределения. Оператор вызова макрокоманды имеет соответствующий список *фактических параметров*, являющихся символьными цепочками; они заменяют фиктивные параметры при расширении макрокоманды. Во время макрорасширения первый фактический параметр заменяет первый фиктивный параметр в коде прототипа, второй фактический параметр заменяет второй фиктивный параметр и т. д. Соответствие фактических параметров фиктивным аналогично соответствию аргументов параметрам в вызовах подпрограммы в языках высокого уровня, например в Фортране.

4.5.1. МАКРОСРЕДСТВА ASM-86

В ассемблере ASM-86 макроопределение образуется следующим образом:

```
%*DEFINE  (Имя макрокоманды (Список фиктивных параметров) )
(       .
        .       Код прототипа
        .
)
```

Здесь имя макрокоманды (которое должно быть идентификатором, начинающимся с буквы и содержащим только буквы, цифры и символы подчеркивания) применяется для вызова макрокоманды, а фиктивные параметры в списке параметров разделяются запятыми. ASM-86 требует, чтобы каждому фиктивному параметру в коде прототипа предшествовал символ %. Макровызов имеет следующий формат:

% Имя макрокоманды (Список фактических параметров)

Фактические параметры также разделяются запятыми.

Как пример на рис. 4.30 показано определение и применение макрокоманды для умножения операндов длиной в слово и запоминания результата (по предположению, не превышающего 16 бит) в третьем операнде. Из макроопределения видно, что макрокоманда вначале запоминает в стеке содержимое регистров AX и DX, выполняет умножение, а затем восстанавливает AX и DX. Фиктивными параметрами являются OPR1, OPR2 и RESULT. В первом макровызове они заменяются на CX, VAR и XYZ [BX] соответственно, а во втором — на 240, BX и SAVE. Если фиктивный параметр находится в операторе, соответствующий фактический параметр должен быть таким, чтобы получился допустимый оператор, иначе ассемблер зафиксирует ошибку. Для макрокоманды, определенной на рис. 4.30, вызов

%MULTIPLY (BX,240,SAVE)

приведет к ошибке, так как в расширении четвертый оператор превращается в IMUL 240, а в команде IMUL непосредственный операнд не допускается.

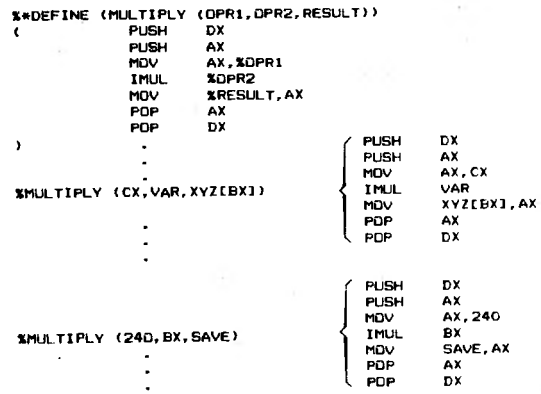


Рис. 4.30. Пример использования макрокоманды

Ассемблер ASM-86 допускает сцепление (конкатенацию) фиктивных параметров с образованием одной символьной цепочки. Для макроопределения

```
%*DEFINE (MSGGEN (LAB,NUM,XYZ))
(%LAB%NUM  DB 'HELLO MR.%XYZ'
)
```

вызов

```
%MSGGEN (MSG,1,TAYLOR)
```

приводит к расширению

```
MSG1 DB 'HELLO MR.TAYLOR'
```

Допускается определять макрокоманду без фиктивных параметров, но в этом случае и вызов не должен содержать никаких параметров. Полезным примером макрокоманды без параметров служит макрокоманда включения в стек содержимого регистров в начале процедуры. Если в исходном модуле имеется макроопределение

```
%*DEFINE  (SAVEREG)
(         PUSH   AX
          PUSH   BX
          PUSH   CX
          PUSH   DX
          PUSH   SI
          PUSH   DI
)
```

то вызов

```
%SAVEREG
```

можно использовать для запоминания регистров в начале каждой процедуры, находящейся в исходном модуле. Аналогичную макрокоманду можно применить и для восстановления регистров в конце каждой процедуры.

Имеются два основных способа встраивания в ассемблер макросредств. Один из них заключается во введении макросредств в первый проход ассемблера, а другой, реализованный в ASM-86, связан с расширением ассемблера путем введения предварительного прохода. Предварительный проход (называемый *макропроцессором*) отыскивает в исходном коде макроопределения и макровызовы. Когда обнаруживается макроопределение, имя макрокоманды и последовательность команд помещаются в таблицу макроопределений. После обнаружения макровызова макропроцессор просматривает таблицу макроопределений и подставляет соответствующее определение в оператор вызова с заменой фиктивных параметров на фактические. Такая конструкция требует, чтобы макрокоманда определялась до ее вызова (данное ограничение снимается в двухпроходовом макропроцессоре). Затем выход макропроцессора, представляющий собой расширенный исходный код, транслируется в машинный код во время процесса ассемблирования (см. § 3.11).

4.5.2. ЛОКАЛЬНЫЕ МЕТКИ

Довольно часто макроопределение должно содержать метки. Рассмотрим, к примеру, макрокоманду ABSOL, которая заменяет операнд его абсолютным значением. Определение этой макрокоманды имеет вид

```
%*DEFINE    (ABSOL   (OPER))
(           CMP      %OPER,0
            JGE      NEXT
            NEG      %OPER
NEXT:       NOP
)
```

После первого вызова ABSOL метка NEXT появится в программе и, следовательно, станет определенной. Любой последующий вызов ABSOL приведет к новому определению NEXT. При этом в процессе ассемблирования будет зафиксирована ошибка, так как NEXT ассоциируется более чем с одной ячейкой. Одно из решений задачи заключается в том, чтобы заменить NEXT фиктивным параметром, а при каждом вызове использовать для метки различный фактический параметр. Но такое решение заставляет программиста следить за тем, какие метки он уже применял, и вводить только неиспользованные.

Чтобы избежать трудностей, связанных с включением метки в список параметров, в некоторых ассемблерах допускаются специальные метки, называемые локальными; они имеют суффиксы, и при каждом вызове макрокоманды осуществляется инкремент суффикса. В ассемблере ASM-86 суффиксами служат двухразрядные начинающиеся с нуля числа, которые увеличиваются на 1; при этом для каждого расширения формируется своя метка. Метки объявляются локальными записью в конце первого оператора макроопределения:

```
LOCAL  Список локальных меток
```

В приведенном ранее примере макрокоманду ABSOL можно переопределить следующим образом:

```
%*DEFINE  (ABSOL  (OPER))  LOCAL  NEXT
(         CMP     %OPER,0
          JGE     %NEXT
          NEG     %OPER
%NEXT:    NOP
```

При этом первые два вызова %ABSOL (VAR) и ABSOL (BX) приведут к следующим расширениям:

```
          CMP   VAR,0
          JGE   NEXT00
          NEG   VAR
NEXT00:   NOP
            .
            .
            .
          CMP   BX,0
          JGE   NEXT01
          NEG   BX
NEXT01:   NOP
```

4.5.3. ВЛОЖЕННЫЕ МАКРОКОМАНДЫ

Допускается появление макровызова внутри макроопределения. Такая ситуация называется *вложением макрокоманд* и имеет ограничение: все макрокоманды, находящиеся в определении данной макрокоманды, должны быть известны до ее вызова. Любую макрокоманду, если ее фиктивные параметры правильно определены и используются, можно вызвать прямо или из одной или нескольких других макрокоманд. Пример вложения макрокоманд приведен на рис. 4.31, где внешняя макрокоманда DIFSQR вычисляет $(OPR1 - OPR2)^2$ и использует для вычисления разности макрокоманду DIF.

```
%*DEFINE (DIFSQR (OPR1,OPR2,RESULT))
(         PUSH    DX
          PUSH    AX
%DIF (%OPR1,%OPR2)
          IMUL    AX
          MOV     %RESULT,AX
          POP     AX
          POP     DX
)
%*DEFINE (DIF (X,Y))
(         MOV     AX,%X
          SUB     AX,%Y
)
            .                          PUSH    DX
            .                          PUSH    AX
            .                          MOV     AX,VAR1
%DIFSQR (VAR1,VAR2,ERROR)              SUB     AX,VAR2
            .                          IMUL    AX
            .                          MOV     ERROR,AX
            .                          POP     AX
                                       POP     DX
```

Рис. 4.31. Пример вложенных макрокоманд

Можно также включать макроопределения внутрь макроопределений. Способ определения макрокоманд для выполнения различных бинарных опера-

ций и запоминания результата в произвольной ячейке показан на рис. 4.32. Макрокоманда DEFMAC содержит определение макрокоманды, имя которой является фиктивным параметром MACNAM. Следовательно, при вызове

```
%*DEFINE (DEFMAC (MACNAM,OPERATOR))
(%*DEFINE (%MACNAM (X,Y,Z))
(           PUSH    AX
            MOV     AX,%X
            %OPERATOR AX,%Y
            MOV     %Z,AX
            POP     AX
)
)

                .
                .
                .
;ОПРЕДЕЛЕНИЕ МАКРОКОМАНДЫ ADDITION
%DEFMAC (ADDITION,ADD)
                .
                .
                .
;ОПРЕДЕЛЕНИЕ МАКРОКОМАНДЫ SUBTRACT
%DEFMAC (SUBTRACT,SUB)
                .
                .
                .
;ОПРЕДЕЛЕНИЕ МАКРОКОМАНДЫ LOGOR
%DEFMAC (LOGOR,OR)
                .
                .
                .
;ВЫЗОВ МАКРОКОМАНДЫ ADDITION
%ADDITION (VAR1,VAR2,VAR3)
                .
                .
                .
```

Рис. 4.32. Применение макрокоманды для определения макрокоманд

DEFMAC она будет определять макрокоманду, имя которой является фактическим параметром в операторе вызова и которое соответствует MACNAM в первом операторе DEFMAC. Оператор

```
%DEFMAC   (ADDITION,ADD)
```

вызывает определение макрокоманды ADDITION, которая суммирует содержимое двух ячеек и помещает результат в третью ячейку. Аналогично определяются макрокоманды SUBTRACT и LOGOR для вычитания и дизъюнкции. Вызов

```
%ADDITION   (VAR1,VAR2,VAR3)
```

приводит к расширению

```
PUSH  AX
MOV   AX,VAR1
ADD   AX,VAR2
MOV   VAR3,AX
POP   AX
```

Так как применение макрокоманд для определения макрокоманд порождает код, в котором трудно разбираться, данную возможность следует использовать только в специальных случаях. Обычно и вложение макрокоманд также приводит к труднопонимаемому коду, но иногда оказывается весьма полезным, и этот недостаток перевешивают достоинства, чего нет при вложении определений.

4.5.4. УПРАВЛЯЕМОЕ РАСШИРЕНИЕ И ДРУГИЕ ФУНКЦИИ

Часто требуется написать исходный модуль, который ассемблируется в различных ситуациях по-разному. Например, модуль может быть предназначен для обслуживания двух типов терминалов и некоторые части исходного кода необходимо написать точно в соответствии с используемым терминалом, хотя значительная часть исходного кода одинакова для обоих типов терминалов. В другой ситуации желательно, чтобы расширение кода прототипа зависело от типов фактических параметров, передаваемых вызовом. Это позволяет вызову определять, игнорировать или нет часть кода прототипа, или выбирать какой-то из двух кодов прототипа. Возможность выбирать ассемблируемый код называется *управляемым расширением* (или *условным ассемблированием*), и в ассемблере ASM-86 такая возможность встроена в предпроцессор.

В ASM-86 выбор ассемблируемого кода определяется управляющей функцией %IF (см. рис. 4.33). На рис. 4.33 определены также управляющие функции %REPEAT и %WHILE, которые позволяют дублировать исходный код до его ассемблирования. В функции %REPEAT выражение должно быть арифметическим и значение его задает число повторений блока кода в исходном коде. В функциях %IF и %WHILE выражение может быть арифметическим или

ФОРМАТ	ОПИСАНИЕ
%IF (ВЫРАЖЕНИЕ) THEN (. } БЛОК КОДА 1) ELSE (. } БЛОК КОДА 2) FI	ЕСЛИ ВЫЧИСЛЕНИЕ ВЫРАЖЕНИЯ ДАЕТ НЕЧЕТНОЕ ЧИСЛО ИЛИ TRUE, В РАСШИРЕНИЕ ВКЛЮЧАЕТСЯ БЛОК КОДА 1; В ПРОТИВНОМ СЛУЧАЕ ВКЛЮЧАЕТСЯ БЛОК КОДА 2
%IF (ВЫРАЖЕНИЕ) THEN (. } БЛОК КОДА) FI	ЕСЛИ ВЫЧИСЛЕНИЕ ВЫРАЖЕНИЯ ДАЕТ НЕЧЕТНОЕ ЧИСЛО ИЛИ TRUE, БЛОК КОДА ВКЛЮЧАЕТСЯ В РАСШИРЕНИЕ; В ПРОТИВНОМ СЛУЧАЕ БЛОК КОДА ИГНОРИРУЕТСЯ
%REPEAT (ВЫРАЖЕНИЕ) (. } БЛОК КОДА)	БЛОК КОДА ДУБЛИРУЕТСЯ ЧИСЛО РАЗ, РАВНОЕ ЗНАЧЕНИЮ ВЫРАЖЕНИЯ
%WHILE (ВЫРАЖЕНИЕ) (. } БЛОК КОДА)	БЛОК КОДА ПОВТОРЯЕТСЯ ДО ТЕХ ПОР, ПОКА ПРИ ВЫЧИСЛЕНИИ ВЫРАЖЕНИЯ НЕ ПОЛУЧИТСЯ ЧЕТНОЕ ЧИСЛО ИЛИ FALSE
%EXIT	ЗАВЕРШАЕТ ДЕЙСТВИЕ МАКРОРАСШИРЕНИЯ

ПРИМЕЧАНИЕ. ЕСЛИ ВЫРАЖЕНИЕ В ФУНКЦИЯХ %IF И %WHILE ЯВЛЯЕТСЯ АРИФМЕТИЧЕСКИМ, НЕЧЕТНОЕ ЧИСЛО ИНТЕРПРЕТИРУЕТСЯ КАК TRUE, А ЧЕТНОЕ ЧИСЛО — КАК FALSE. ЕСЛИ ВЫРАЖЕНИЕ ЯВЛЯЕТСЯ ЛОГИЧЕСКИМ, TRUE ПРЕДСТАВЛЯЕТСЯ -1H, А FALSE - 00H.

Рис. 4.33. Управляющие функции

двумя арифметическими, объединенными одним из следующих операторов отношений:

EQ – Равно	NE – Не равно
LT – Меньше	LE – Меньше или равно
GT – Больше	GE – Больше или равно

Кроме того, оно может быть полным логическим выражением, которое содержит комбинацию логических операторов NOT, AND, OR и XOR. Во всех случаях значением выражения является TRUE (истина) или FALSE (ложь). Для арифметического выражения нечетное число интерпретируется как TRUE, а четное – как FALSE; когда же выражение не является арифметическим, значение –1H считается TRUE, а 00H считается FALSE. (Так как выражения вычисляет макропроцессор, в нем нельзя использовать имена, определяемые директивами ассемблера, например директивой EQU.)

Следующий код

```
%IF   (%VECTYPE  EQ 0)  THEN
(                       INC     VECT0
)
ELSE
(                       INC     VECT1
)FI
```

вызывает ассемблирование команды INC VECT0, если значение VECTYPE равно нулю; в противном случае ассемблируется INC VECT1. Код

```
%IF   (%REG_NUM)   THEN
(     %REPEAT   (3)
(     ADD   AX,INCREMENT
)
)FI
```

вызывает прибавление (INCREMENT) к AX три раза, если REG_NUM нечетное число; в противном случае команда ADD не ассемблируется.

Чтобы блок кода, ассоциированный с управляющей функцией %WHILE, не расширялся бесконечно, он должен содержать операторы, которые изменяют значение выражения в операторе %WHILE. Обычно зто осуществляется

ФОРМАТ	ОПИСАНИЕ
%SET (ИМЯ,ЗНАЧЕНИЕ)	ИМЯ АССОЦИИРУЕТСЯ СО ЗНАЧЕНИЕМ
%LEN (СИМВОЛЬНАЯ ЦЕПОЧКА)	ЕСЛИ %LEN ЯВЛЯЕТСЯ ЧАСТЬЮ ВЫРАЖЕНИЯ В ФУНКЦИЯХ %IF, %WHILE ИЛИ %REPEAT, ТО КАК ЧИСЛО ВОЗВРАЩАЕТСЯ ДЛИНА ЦЕПОЧКИ; В ПРОТИВНОМ СЛУЧАЕ ВОЗВРАЩАЕМЫМ ЗНАЧЕНИЕМ ЯВЛЯЕТСЯ ПРЕДСТАВЛЕНИЕ ДЛИНЫ В КОДЕ ASCII С ДОБАВЛЕННОЙ БУКВОЙ "H" (В КОДЕ ASCII)

Рис. 4.34. Функции %SET и %LEN

с помощью функции %SET, приведенной на рис. 4.34, которая изменяет значение имени, содержащегося в выражении оператора %WHILE. Например, код

```
%SET      (X, 0)
%WHILE      (%X LT 10 AND %VAR GT 0)
(CONS%X       DB  %X
%SET      (X,%X + 1)
)
```

вызывает ассемблирование

```
CONS00H  DB  00H
CONS01H  DB  01H
CONS02H  DB  02H
              .
              .
              .
CONS09H  DB  09H
```

если %VAR больше нуля; в противном случае блок кода пропускается. Отметим, что значение %X, которое является представлением X в коде ASCII, сцепляется с цепочкой CONS путем добавления %X к CONS.

Последнее определение на рис. 4.33 касается функции %EXIT. Эта управляющая функция предназначена для более раннего окончания макроопределения, например из блока кода %IF.

На рис. 4.34 дано также определение функции %LEN, которая возвращает длину своего аргумента. Если %LEN появляется в выражении управляющих функций %IF, %REPEAT или %WHILE, возвращаемое %LEN значение является числом, но если %LEN применяется где-то еще, возвращается представление значения в коде ASCII с добавленной буквой "H" (в коде ASCII). Если макрокоманда

```
%*DEFINE  (MSGGEN  (X, Y, Z))
(%X    DW     %LEN (%Z)
%Y     DB     '%Z'
)
```

вызывается оператором

```
%MSGGEN  (NUM1,MSG1, GOOD  MORNING)
```

получается расширение

```
NUM1  DW  0CH
MSG1  DB  'GOOD  MORNING'
```

Кроме операторов отношений, которые сравнивают численные значения, ASM-86 имеет набор функций для лексического сравнения символьных цепочек. Каждая из функций, определенных на рис. 4.35, дает результат TRUE (−1H) или FALSE (00H).

Рассмотрим, каким образом можно расширить макрокоманду MULTIPLY на рис. 4.30 так, чтобы результат запоминался в AX. В первоначальном определении AX нельзя указывать как третий аргумент, так как в этом случае

ФОРМАТ	ОПИСАНИЕ *
%EQS (цепочка 1,цепочка 2)	TRUE(-1H), если две цепочки идентичны
%NES (цепочка 1,цепочка 2)	TRUE(-1H), если две цепочки не идентичны
%LTS (цепочка 1,цепочка 2)	TRUE(-1H), если цепочка 1 алфавитно предшествует цепочке 2
%LES (цепочка 1,цепочка 2)	TRUE(-1H), если цепочка 1 алфавитно равна или предшествует цепочке 2
%GES (цепочка 1,цепочка 2)	TRUE(-1H), если цепочка 1 алфавитно равна или следует за цепочкой 2
%GTS (цепочка 1,цепочка 2)	TRUE(-1H), если цепочка 1 алфавитно следует за цепочкой 2

* FALSE представляется как 00H

Рис. 4.35. Функции сравнения цепочек

команда POP AX разрушит произведение. Изменим определение MULTIPLY в следующей форме:

```
%*DEFINE  (MULTIPLY   (OPR1,OPR2,RESULT))
(         PUSH    DX
          %IF   (%NES  (%RESULT,AX)) THEN
(         PUSH    AX
)FI
          MOV     AX,OPR1
          IMUL    %OPR2
          %IF     (%EQS (%RESULT,AX)) THEN
(         POP     DX
)         ELSE
(         MOV     %RESULT,AX
          POP     AX
          POP     DX
)FI
)
```

В этом случае оператор

```
%MULTIPLY  (CX,VAR,AX)
```

порождает следующее расширение:

```
PUSH  DX
MOV   AX,CX
IMUL  VAR
POP   DX
```

4.6. ПРОЕКТИРОВАНИЕ ПРОГРАММЫ

Почти все программы имеют иерархическую структуру, наверху которой находится управляющий программный модуль, называемый *основной программой* и предназначенный для управления выполнением модулей, находя-

щихся под ним. В свою очередь эти главные подмодули управляют использованием своих подмодулей и т. д. Как показано на рис. 4.1, подмодули могут появляться в нескольких местах иерархической диаграммы, например модуль для ввода конкретного набора данных или для реализации BCD-деления.

Проектирование программы сильно связано с ее иерархической структурой. После точной формулировки программируемой проблемы первый этап состоит в определении основных задач. Обычно они включают в себя задачу ввода, задачу вывода и несколько задач обработки. Соответствующие основным задачам модули необходимо тщательно определить и продумать их взаимодействия друг с другом. Затем специальные задания из основных модулей назначаются подмодулям и определяются взаимосвязи полученных подструктур. Этот процесс, называемый *проектированием сверху-вниз*, продолжается до разделения программы на такие части, в которых можно легко разобраться и которые можно легко реализовать.

При проектировании сверху-вниз важно разобраться в фундаментальной природе взаимодействия между модулями (*сопряжение по данным*) и в том способе, каким модули вызывают и управляют друг другом (*сопряжение по управлению*). При разделении программы на задачи и определении соответствующих модулей необходимо опираться на выполняемые функции. Обычно не рекомендуется разделять функцию (например, вычисление детерминанта) на модули одного и того же уровня, т. е. модули, которые не образуют подиерархическую структуру. Кроме функционального разделения, решение о расчленении программы должно опираться на сопряжения по данным и управлению. Четкие правила принятия решений отсутствуют, но в большинстве случаев справедливы следующие правила:

1. Если задача требует передачи необычного объема данных между ней и порождающей задачей, следует обе задачи запрограммировать в одном и том же модуле. Понятие "необычный" зависит от размера и важности подчиненной задачи, а также от того, насколько легко можно обращаться к данным. Применение общих областей сокращает передачу параметров, но у этого подхода имеются свои сложности.

2. Следует избегать в модуле нескольких точек входа и выхода. Связи между модулями должны быть максимально простыми.

Модули обычно программируются как процедуры, а сложных связей между процедурами необходимо избегать.

При разделении задач между модулями важным фактором оказывается его длина. Чем больше операторов содержит модуль, тем труднее разобраться в нем и отладить. Но, с другой стороны, очень короткий модуль, возможно, не оправдывает связей, необходимых для взаимодействия с ним (если он не является макрокомандой и не требует связей). В общем, процедурный модуль, состоящий из менее 20 или более 100 операторов, необходимо тщательно проанализировать и рассмотреть возможности объединения его с порождающим модулем или вынесения части его функций в подчиненные модули.

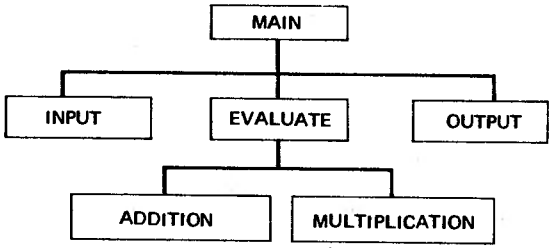


Рис. 4.36. Иерархическая диаграмма программы вычисления полинома

Когда иерархическая структура программы определена, необходимо разобраться со структурой обрабатываемых программой данных и четко сформулировать входы и выходы каждого модуля. Некоторые данные следует разместить в общих областях, к которым могут обращаться все модули, а другие данные придется локализовать и передавать их между модулями через стек или таблицу параметров. Помните, что локализованные данные можно передавать только тем модулям, которые подчинены модулю, содержащему определение данных.

Назначения входов и выходов можно осуществить с помощью описания модуля. Обычно такое описание просто перечисляет входы и выходы и содержит краткое пояснение функции модуля. Важно, чтобы описание было дано для каждого модуля и чтобы эти описания можно было легко соотнести с соответствующими блоками в иерархической диаграмме.

Как простой пример рассмотрим программу, которая вводит набор коэффициентов в массив A и числа в переменные X и N, производит вычисление

$$Y \leftarrow A_N X^N + \ldots + A_1 X + A_0$$

и выводит содержимое переменной Y. Предположим, что все входные, обрабатываемые и выходные величины являются 16-разрядными неупакованными BCD-числами. Иерархическая диаграмма программы приведена на рис. 4.36. Она состоит из модулей MAIN, INPUT, EVALUATE, ADDITION, MULTIPLICATION и OUTPUT. Модуль MAIN вызывает INPUT для ввода необходимых данных, использует модуль EVALUATE для производства вычислений, а затем переходит к модулю OUTPUT для печати значения Y. Модуль EVALUATE использует для выполнения арифметических операций подмодули ADDITION и MULTIPLICATION. Описания модулей INPUT, EVALUATE и ADDITION показаны на рис. 4.37.

После определений модулей, их входов и выходов, структуры данных, построения иерархической диаграммы и составления описаний модулей программист готов к разработке и кодированию отдельных модулей. К этому времени уже должны быть приняты алгоритмы работы модулей, так как при

```
ИМЯ:  INPUT

ВХОДЫ:  ВВОДЯТСЯ С ТЕРМИНАЛА
        N, X  И  A_N,...,A_0

ВЫХОДЫ:  В MAIN ЧЕРЕЗ ТАБЛИЦУ ПАРАМЕТРОВ:
         N, X  И  A_N,...,A_0

ФУНКЦИЯ:  ВВЕСТИ ВСЕ ДАННЫЕ НЕОБХОДИМЫЕ ДЛЯ ВЫЧИСЛЕНИЯ

                A_N X^N + ... + A_1 X + A_0

                а)

ИМЯ:  EVALUATE

ВХОДЫ:  ИЗ MAIN ЧЕРЕЗ ТАБЛИЦУ ПАРАМЕТРОВ:
        N, X  И  A_N,...,A_0

ВЫХОДЫ:  В MAIN ЧЕРЕЗ ТАБЛИЦУ ПАРАНЕТРОВ:
         ЗНАЧЕНИЕ ПОЛИНОМА В Y

ФУНКЦИЯ:  ВЫЧИСЛИТЬ

        Y <-- A_N X^N + ... + A_1 X + A_0

        С ПОМОЩЬЮ ПРАВИЛА ГОРНЕРА И ПОДМОДУЛЕЙ ADDITION
        И MULTIPLICATION ДЛЯ ВЫПОЛНЕНИЯ АРИФМЕТИЧЕСКИХ
        ОПЕРАЦИЙ НАД 16-РАЗРЯДНЫМИ НЕУПАКОВАННЫМИ BCD-ЧИСЛАМИ.

                б)

ИМЯ:  ADDITION

ВХОДЫ:  ИЗ EVALUATE ЧЕРЕЗ ТАБЛИЦУ ПАРАМЕТРОВ:
        СУММИРУЕМЫЕ ОПЕРАНДЫ

ВЫХОДЫ:  В EVALUATE ЧЕРЕЗ ТАБЛИЦУ ПАРАМЕТРОВ:
         СУММА

ФУНКЦИЯ:  СЛОЖИТЬ ДВА 16-РАЗРЯДНЫХ НЕУПАКОВАННЫХ BCD-ЧИСЛА.

                в)
```

Рис. 4.37. Описания нескольких модулей в программе вычисления полинома: *а* – ввод; *б* – вычисление; *в* – сложение

определении функций модулей обычно требуется некоторое знание алгоритмов. Однако детали алгоритмов пока отсутствуют.

Разработка и кодирование ассемблерного модуля, по существу, аналогичны этим же действиям для модуля на языке высокого уровня, хотя несколько внимательнее анализируется эффективность модуля. На рис. 4.38, *а* – *д* приведены элементарные программные структуры, называемые *конструкциями*. Модули следует формировать, размещая эти конструкции в простую последовательность или организуя их вложение. Выход реализуется как модификация конструкций IF-THEN-ELSE, DO-WHILE, DO-UNTIL или CASE. Выход из цикла DO-WHILE представлен на рис. 4.38, *е*.

На рис. 4.39 приведена схема типичной комбинации конструкций. Ветвь THEN содержит только вызов процедуры, а ветвь ELSE – простую последовательность (или простое следование). Первым элементом простой последовательности является цикл DO-WHILE с выходом, вторым – вызов процедуры, а третьим – конструкция IF-THEN-ELSE с пустой ветвью ELSE.

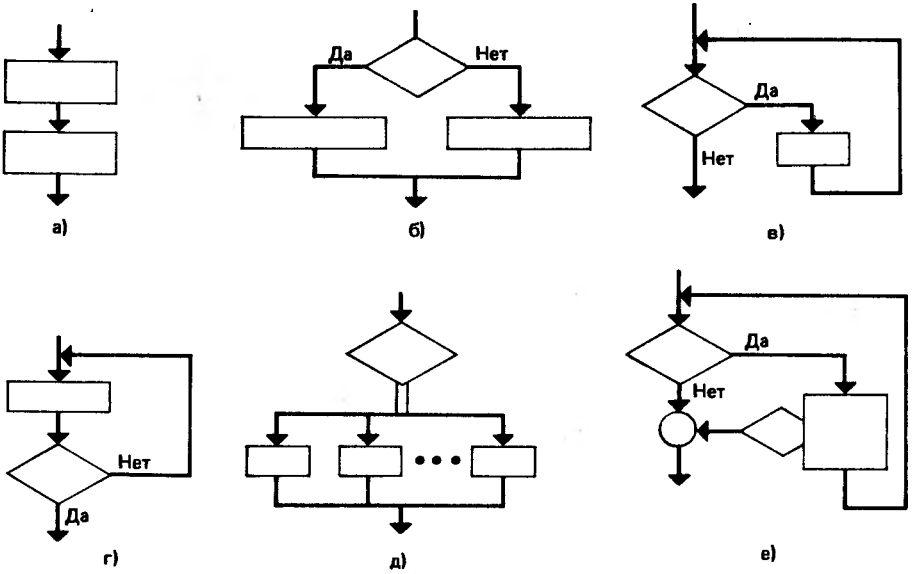


Рис. 4.38. Элементарные конструкции и пример выхода:
а – простая последовательность; *б* – IF-THEN-ELSE; *в* – DO-WHILE; *г* – DO-UNTIL; *д* – CASE; *е* – выход из DO-WHILE

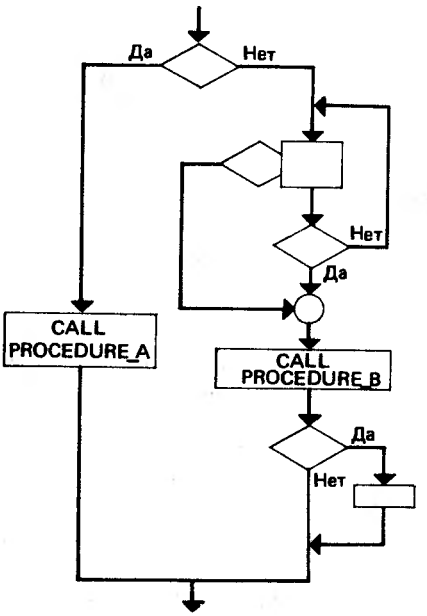


Рис. 4.39. Типичная комбинация элементарных конструкций

Необходимо избегать отклонений от стандартных конструкций, но, так как при программировании на языке ассемблера бóльшее внимание уделяется эффективности, приемлемы бóльшие отклонения, чем при программировании на языке высокого уровня. Преобладающим отклонением оказывается выход, который переходит к точке, находящейся через несколько операторов после окончания его конструкции. При программировании на языке ассемблера иногда удобно ввести небольшой фрагмент кода между концом конструкции и точкой перехода.

Подчеркивание важности элементарных конструкций обусловлено тем, что получаемые из них программы проще понимать и отлаживать. Переходы назад, кроме как в циклах, применять не следует, так как они могут быть запутанными и часто вносят логические ошибки, которые трудно находить. Нестандартных конструкций удается избежать благодаря интенсивному использованию процедур и программных флажков (т. е. переменных, которые управляют логикой программы).

Ассемблерные реализации всех конструкций, кроме конструкции простого следования, для микропроцессора 8086 приведены на рис. 4.40. Структура IF-THEN-ELSE на рис. 4.40, *а* предполагает, что в зависимости от равен-

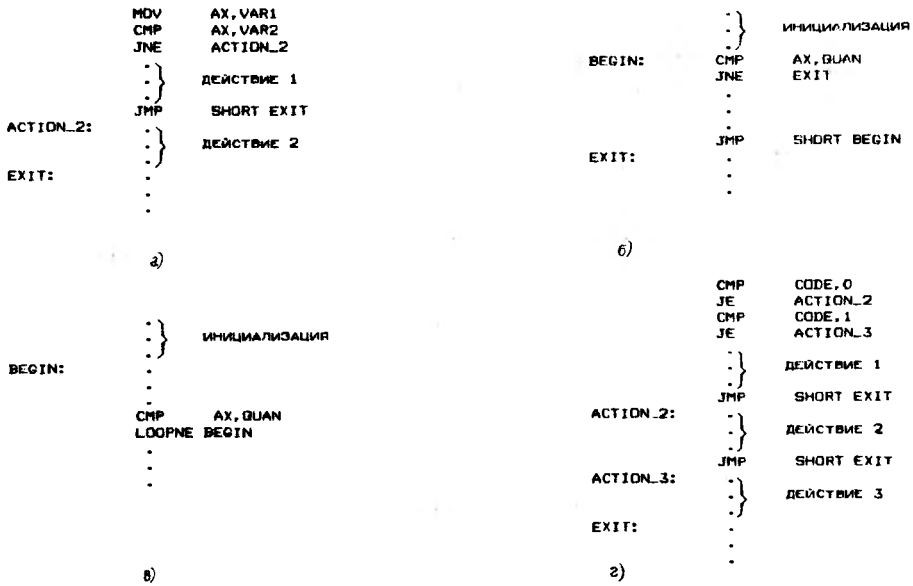


Рис. 4.40. Типичная ассемблерная реализация элементарных конструкций:
а – IF-THEN-ELSE; *б* – DO-WHILE; *в* – DO-UNTIL; *г* – CASE

ства или неравенства двух величин предпринимается одно из двух действий. Если действие программных последовательностей оказывается длинным, их можно оформить в процедуры, а без процедур команду JNE следует заменить на комбинацию

```
                JE    ACTION_1
                JMP   NEAR PTR ACTION_2
ACTION_1:       .
                .
                .
                .
```

и команду JMP SHORT EXIT придется заменить на команду

```
JMP NEAR PTR EXIT
```

Реализация конструкции DO-WHILE на рис. 4.40, *б* опирается только на сравнение двух величин, но проверка в цикле DO-UNTIL на рис. 4.40, *в* основана на счете и сравнении. Структура CASE на рис. 4.40, *г* предполагает, что различные действия предпринимаются в зависимости от значения CODE. Если CODE = 0, то выполняется действие 2; если CODE = 1, выполняется действие 3, а когда CODE равно любому другому числу, выполняется действие 1. Как и в примере для IF-THEN-ELSE, если действия в реализациях циклов и CASE

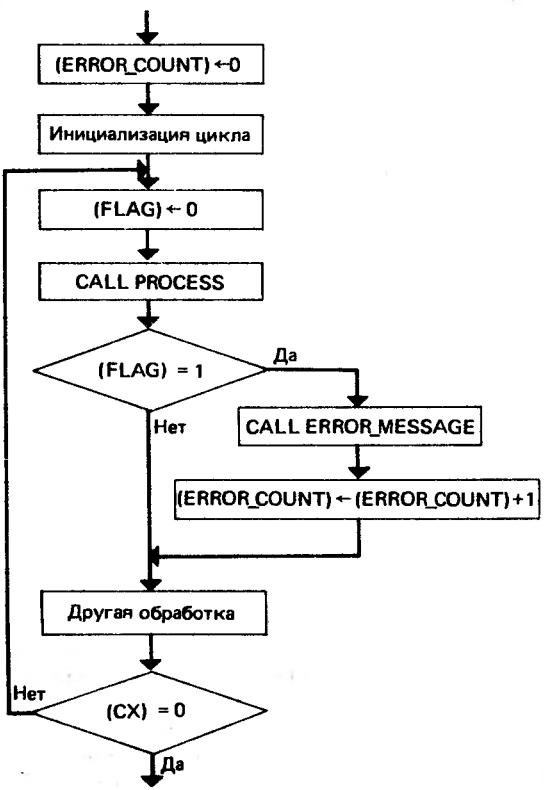


Рис. 4.41. Реализация комбинации элементарных конструкций

слишком длинные, код следует модифицировать так, чтобы в нем применялись безусловные переходы типа NEAR.

Чтобы показать кодирование комбинации конструкций, рассмотрим схему на рис. 4.41 и соответствующий код на рис. 4.42. Программа входит в цикл со значением ERROR_COUNT = 0. Внутри цикла программа устанавливает FLAG = 0 и вызывает процедуру PROCESS. Если при выполнении PROCESS возникает некоторая ошибка, флажок FLAG устанавливается в 1. Затем сразу после выхода из PROCESS проверяется содержимое FLAG и, если оно равно 1, вызывается процедура ERROR_MESSAGE и производится инкремент ERROR_COUNT.

```
        MOV     ERROR_COUNT,0
        .
        .       } ИНИЦИАЛИЗАЦИЯ ЦИКЛА
        .
BEGIN:  MOV     FLAG,0
        .
        .       } СВЯЗЫВАНИЕ ДЛЯ PROCESS
        .
        CALL    PROCESS
        CMP     FLAG,1
        JNE     OTHER
        .
        .       } СВЯЗЫВАНИЕ ДЛЯ ERROR_MESSAGE
        .
        CALL    ERROR_MESSAGE
        INC     ERROR_COUNT
OTHER:  .
        .       } ДРУГАЯ ОБРАБОТКА
        .
        LOOP    BEGIN
```

Рис. 4.42. Программа реализации схемы, показанной на рис. 4.41

Этот пример показывает также использование процедур для разделения и сокращения модуля. Можно включить одну или обе процедуры, вызываемые в цикле, прямо в код (т. е. как часть простого следования), но при введении процедур модуль, содержащий цикл, не "запутывается" функциями PROCESS и ERROR_MESSAGE.

Псевдокод и схемы помогают в процессе кодирования, а затем служат для целей документирования, как и при программировании на языках высокого уровня. Преобразование псевдокода в ассемблерные операторы оказывается не тривиальным из-за отсутствия соответствия между ключевыми словами псевдокода и ассемблерных операторов. Обычно для каждого оператора псевдокода требуется несколько ассемблерных операторов. Кроме подробной схемы программы, рекомендуется использовать системные схемы, показывающие передачи данных из(в) файлов и устройств ввода-вывода.

При программировании на языке ассемблера необходимо тщательно обосновывать выбор между процедурой или циклом и прямым кодированием. Связи процедуры или инициализация цикла могут потребовать больше команд, чем их экономится при использовании процедуры или цикла. Например, иногда проще записать 4 команды ADD в простой последовательности, чем реализовать цикл. Однако 4 команды ADD оказываются негибкими и позволяют просуммировать точно пять чисел. При организации процедур приходится учитывать несколько факторов: объем связей, длину процедуры,

необходимость размещать процедуру в том же исходном модуле, который вызывает ее, или в библиотеке. Иногда целесообразнее поместить часто используемую процедуру в библиотеку, даже если вызов ее потребует большего объема связей, чем требуется для выполнения задачи.

Важную роль в разработке модульных программ играют макрокоманды, поскольку они, хотя и не сокращают машинного кода, уменьшают размер исходного кода, что необходимо учитывать при оценке длины модулей.

При программировании на языке ассемблера важное значение имеет время выполнения. По существу, одной из причин предпочтения языка ассемблера языкам высокого уровня является большая степень управления процессом, что позволяет уменьшить время выполнения. Именно в этом случае необходимо тщательно проанализировать применение циклов и процедур. На инициализацию, модификацию, проверку и переход в циклах и на связи в процедурах расходуется время.

4.7. ПРИМЕР ПРОЕКТИРОВАНИЯ ПРОГРАММЫ

Предположим, что во внешней памяти хранятся записи о персонале и оглавление, показывающее местонахождение каждой записи. Необходимо разработать программу для модификации записи, т. е. для введения записей о новых работниках, удаления записей об уволившихся работниках и изменения данных в записях имеющихся работников. Программа должна реализовать следующие действия:

1. Ввести оглавление.

2. Ввести приказ "S" (stop – остановиться) или приказ, начинающийся с букв "I" (insert – ввести), "D" (delete – удалить) или "C" (change – изменить) и заканчивающийся набором модифицирующих данных.

3. Если буква приказа "S", программа выводит модифицированное оглавление, очищает системные файлы и завершается; в противном случае программа переходит к шагу 4.

4. Установить код ошибки 0.

5. Если приказом являются буквы "I", "D" или "C", перейти к шагу 6; иначе установить код ошибки 1 и перейти к шагу 7.

6. Осуществить запрошенную модификацию. Если в данных имеется ошибка, установить код ошибки 2.

7. При наличии ошибки напечатать идентифицирующее ее сообщение.

8. Ввести новый набор модифицирующих данных и перейти к шагу 3.

На рис. 4.43 приведены схема программы и системная схема, а на рис. 4.44 – иерархическая диаграмма, показывающая разделение программы. Модули должны выполнять следующие функции:

MAIN. Управляет общим действием, вызывая по мере необходимости другие модули.

INPUTM. Вводит оглавление и записи о работниках с устройства внешней памяти.

OUTPUTM. Выводит оглавление и записи о работниках в устройство внешней памяти.

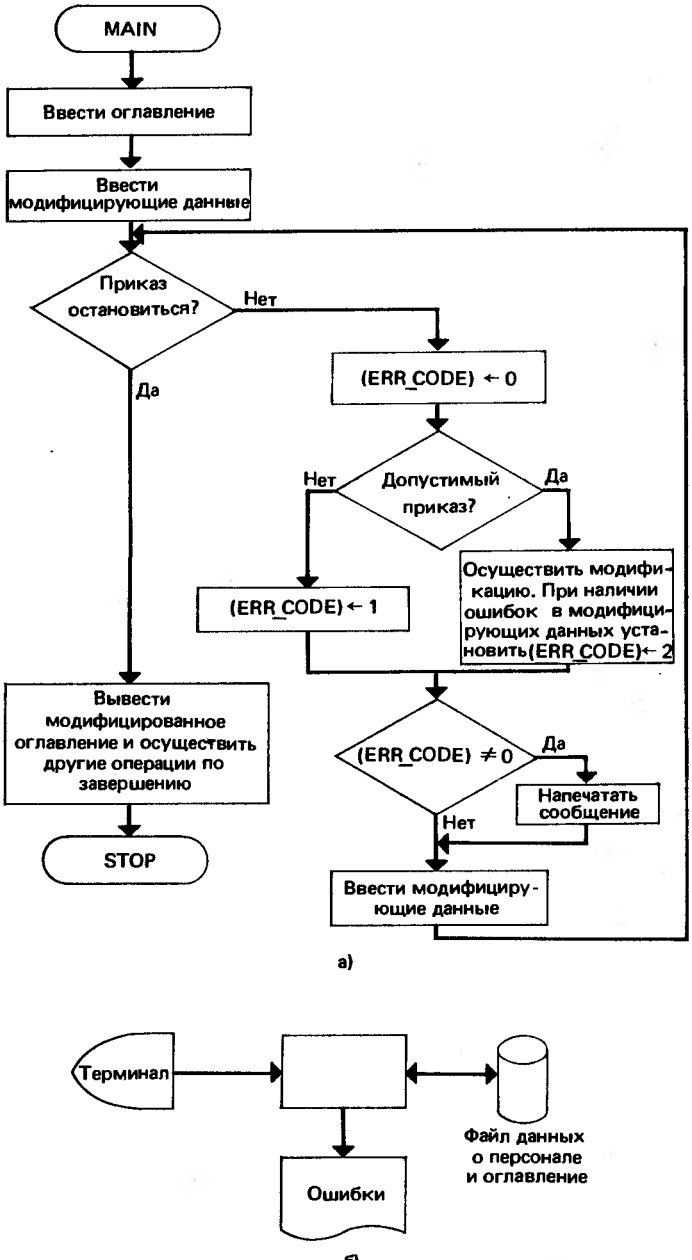


Рис. 4.43. Схема программы (*а*) и системная схема (*б*) обработки записей о персонале

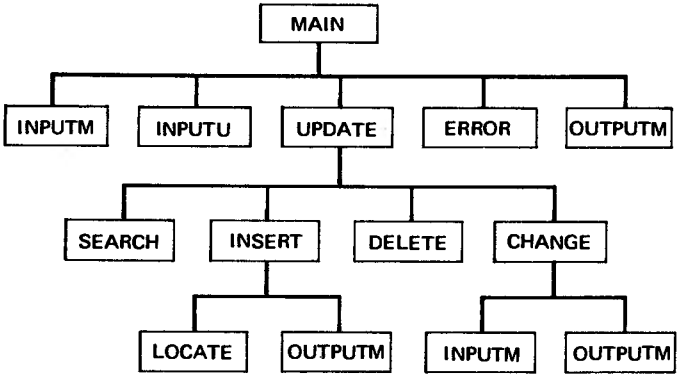


Рис. 4.44. Иерархическая диаграмма для примера обработки записей о персонале

INPUTU. Вводит набор данных, необходимых для выполнения модификации.

UPDATE. Определяет тип модификации и управляет действием модификации.

SEARCH. Отыскивает в оглавлении элемент, относящийся к заданной записи.

INSERT. Помещает в оглавление элемент о новой записи и выводит новую запись во внешнюю память.

LOCATE. Определяет информацию, необходимую системе для запоминания новой записи.

DELETE. Удаляет из оглавления элемент о записи.

CHANGE. Вводит запись из внешней памяти, осуществляет запрошенное оглавление и выводит модифицированную запись.

ERROR. Определяет тип ошибки и печатает соответствующее сообщение.

Отметим, что сообщения об ошибках печатает отдельный модуль. Модуль ERROR получает значение кода через переменную ERR_CODE, которую устанавливают другие модули при обнаружении ошибки, и печатает соответствующее сообщение. Такой способ снимает необходимость наличия вызовов программы печати сообщения в разных местах программы и сокращает модули, которые должны фиксировать ошибки. Конечно, при этом появляется дополнительная переменная ERR_CODE, но с точек зрения эстетики и отладки программы выигрыш очевиден.

Предполагается, что оглавление структурировано в виде связанного списка, упорядоченного по возрастанию номеров работников. Как показано на рис. 4.45, *связанный список* представляет собой набор элементов, причем одно из полей в каждом элементе служит указателем местоположения следующего элемента списка. В каждом элементе имеются еще два поля; одно из них является ключом идентификации, а второе — информационной частью элемента. В нашей проблеме список представляет собой оглавление, а каж-

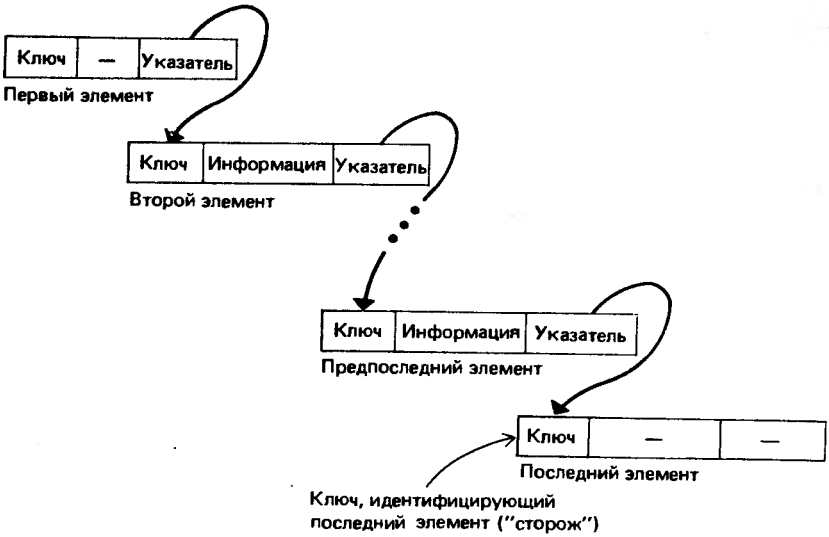


Рис. 4.45. Организация связанного списка

дый элемент списка относится к одному работнику. Ключ служит идентификационным номером работника, а информационное поле содержит информацию для процедур ввода и вывода внешней памяти, которая необходима для локализации записи о работнике в устройстве внешней памяти. Для микропроцессора 8086 список можно определить как набор структур, каждая из которых является элементом списка. Указатель на следующий элемент может быть физической позицией в массиве структур относительно начального (базового) адреса массива. Следовательно, если поместить базовый адрес в BX и указатель в SI, начальный адрес элемента сразу находится с помощью базовой индексной адресации.

Предполагается, что идентификационные номера работников имеют по 4 разряда и находятся в диапазоне 1000 . . . 9999. Первый элемент в массиве просто адресует элемент первого работника. Он должен иметь ключ 1 и никогда не удаляется. Поля ключей элементов в массиве, которые в данный момент не связаны в список, должны быть установлены в 0. Последний элемент в массиве всегда является последним элементом в списке. Он содержит в поле ключа 10000 и никогда не удаляется. Число 10000 сигнализирует о достижении конца списка (такой элемент называется *сторожем*). В элементах одно слово отведено для ключа, три слова для информации и одно слово для указателя.

Для введения элемента в упорядоченный связанный список вначале необходимо найти место, где должно осуществляться введение, и неиспользуемый элемент в массиве списка. Затем в неиспользуемый элемент помещается новый вводимый элемент. Наконец, указатель из элемента списка, который предшествует новому элементу, помещается в поле указателя нового

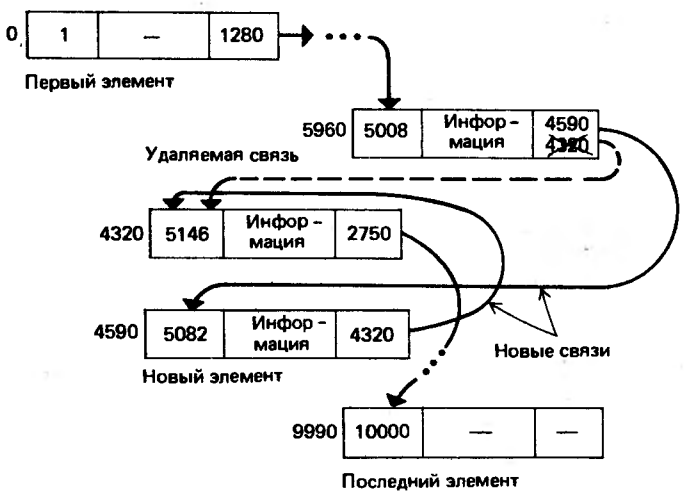
элемента, а затем он (т. е. указатель предшествующего элемента) изменяется для адресации нового элемента. Этот процесс иллюстрируется на рис. 4.46, *а*. Если ключ вводимого элемента равен 5082, элементами с ключами, соседними 5082, являются

Ячейка 5960 5008 Информация 4320
Ячейка 4320 5146 Информация 2750

и новый элемент помещается в ячейку 4590 в массиве списка, то новым элементом будет

Ячейка 4590 5082 Информация 4320

а указатель в элементе, ключ которого равен 5008, изменяется на 4590. Удаление произвольного элемента показано на рис. 4.46, *б*. Отметим, что упорядочивание осуществляется в соответствии с ключами, а не с ячейками в мас-



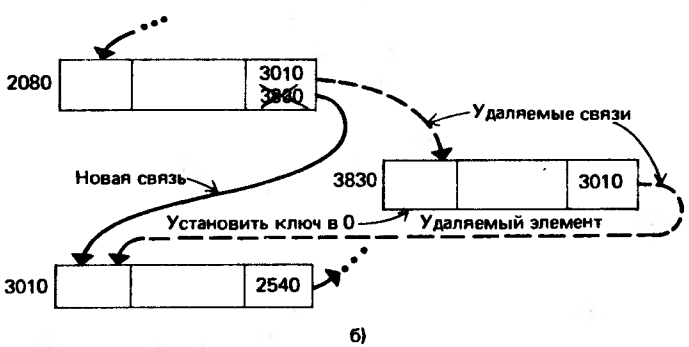


Рис. 4.46. Введение (*а*) и удаление (*б*) элемеитов из связанного списка

сиве списка. Независимость ключей от положения в массиве является основным достоинством связанного списка.

Предположим, что модули INPUTM, OUTPUTM и LOCATE являются библиотечными программами и что INPUTM и OUTPUTM должны получать из таблицы параметров число вводимых или выводимых байт, три слова о местонахождении записи во внешней памяти и адрес памяти, указывающий, где хранятся вводимые или выводимые данные. Модулю LOCATE передается адрес блока из трех слов, в котором он должен запомнить доступное местонахождение во внешней памяти. Модули MAIN, INPUTU, UPDATE и ERROR ассемблируются отдельно, а SEARCH, INSERT, DELETE и CHANGE включены как отдельные процедуры в исходный модуль UPDATE. Следовательно, модуль UPDATE и указанные подмодули могут использовать одни и те же имена переменных и передача параметров не нужна.

Приведем список элементов данных, которые используются более чем одним модулем:

DATA_BUFF (256 байт). Область памяти, предназначенная для хранения обрабатываемой записи. Ее местонахождение передается через таблицу параметров.

NO_OF_BYTES (одно слово). Число байт, вводимых INPUTM или выводимых OUTPUTM. Его местонахождение передается с помощью таблицы параметров.

NEW_REG_INFO (три слова). Информационная часть элемента в DIRECTORY, которая заполняется LOCATE. Ее адрес передается через таблицу параметров.

DIRECTORY (5000 слов, 1000 элементов). Упорядоченный связанный список оглавления записей о работниках. Находится в общей области.

UPDATE_DATA (259 байт). Область памяти для хранения модифицирующих данных, введенных модулем INPUTU. Находится в общей области.

ERR_CODE (1 байт). Код, показывающий, возникла или нет ошибка, и если возникла, то какого вида. Находится в общей области.

SRCH_CODE (1 байт). Код, который показывает результат поиска SEARCH. Определение этого кода: 0 – ключ не найден, 1 – ключ найден. Является локальным для UPDATE и его непосредственных подмодулей.

PRIOR_ELE (одно слово). Для временного хранения индекса элемента, являющегося предыдущим для последнего элемента, проверенного в ходе поиска. Является локальным для UPDATE и его непосредственных подмодулей.

CUR_ELE (одно слово). Для временного хранения индекса последнего элемента, проверенного в ходе поиска. Является локальным для UPDATE и его непосредственных подмодулей.

KEY (одно слово). Для хранения ключа, отыскиваемого SEARCH. Является локальным для UPDATE и его непосредственных подмодулей.

Рекомендуемые описания модулей UPDATE и SEARCH приведены на рис. 4.47.

```
ИМЯ:  UPDATE

ОБЩИЕ ОБЛАСТИ:  ОБЩИЕ ОБЛАСТИ, КОТОРЫЕ ИСПОЛЬЗУЮТСЯ ИЛИ МОДИФИЦИРУЮТСЯ

          DIRECTORY, UPDATE_DATA, ERR_CODE

ФУНКЦИЯ:  ОПРЕДЕЛИТЬ БУКВУ КОДА ПРИКАЗА "I" (ВВЕСТИ), "D" (УДАЛИТЬ) ИЛИ
      "C" (ИЗМЕНИТЬ) И ВЫЗВАТЬ ПОДМОДУЛИ SEARCH, INSERT, DELETE ИЛИ
      CHANGE ДЛЯ ВЫПОЛНЕНИЯ УКАЗАННОГО ДЕЙСТВИЯ. (ПРОВЕРКА НА ПРИКАЗ
      "S" (ОСТАНОВИТЬСЯ) ОСУЩЕСТВЛЯЕТСЯ В MAIN.)  МОДУЛЬ ВНАЧАЛЕ
      ЗАГРУЖАЕТ ERR_CODE,  А ЗАТЕМ,  ЕСЛИ НЕ МОЖЕТ ИДЕНТИФИЦИРОВАТЬ
      БУКВУ ПРИКАЗА,  ВОЗВРАЩАЕТСЯ В MAIN,  ЗАГРУЖАЯ 1 В ERR_CODE.
      ПОДМОДУЛИ UPDATE МОГУТ ИЗМЕНИТЬ ERR_CODE НА 2,  ЕСЛИ БУДЕТ
      ОБНАРУЖЕНА ОШИБКА.

          а)

ИМЯ:  SEARCH

ВХОД:  ИЗ UPDATE - ЛОКАЛЬНАЯ ПЕРЕМЕННАЯ

          KEY

ВЫХОД:  ИЗ UPDATE - ЛОКАЛЬНЫЕ ПЕРЕМЕННЫЕ

          PRIOR_ELE, CUR_ELE, SRCH_CODE

ОБЩИЕ ОБЛАСТИ:  ИСПОЛЬЗУЮТСЯ ОБЩИЕ ОБЛАСТИ

          ,DIRECTORY, ERR_CODE

ФУНКЦИЯ:  УСТАНАВЛИВАЕТ ERR_CODE НА 2 И ВОЗВРАЩАЕТСЯ, ЕСЛИ (KEY)
      НАХОДИТСЯ ВНЕ ДИАПАЗОНА 1000...9999.  В ПРОТИВНОМ СЛУЧАЕ
      ОТЫСКИВАЕТ В DIRECTORY КЛЮЧ ИЗ KEY И ВОЗВРАЩАЕТ В UPDATE
      ИНДЕКСЫ ТАКИХ ЭЛЕМЕНТОВ, У КОТОРЫХ:

          КЛЮЧ ЯВЛЯЕТСЯ НАИБОЛЬШИМ КЛЮЧОМ, МЕНЬШИМ (KEY),
          КЛЮЧ ЯВЛЯЕТСЯ НАИМЕНЬШИМ КЛЮЧОМ, БОЛЬШИМ ИЛИ РАВНЫМ (KEY).

      ЗАГРУЖАЕТ В SRCH_CODE ЕДИНИЦУ,  ЕСЛИ СООТВЕТСТВИЕ (KEY) НАЙДЕНО;
      В ПРОТИВНОМ СЛУЧАЕ ПОМЕЩАЕТ В SRCH_CODE НУЛЬ.

          б)
```

Рис. 4.47. Описания модулей UPDATE (*а*) и SEARCH (*б*) для примера обработки записей о персонале

Приказ модификации, который содержит информацию, введенную INPUTU и UPDATE_DATA, должен иметь однобайтный тип приказа (буквы "I", "D", "C" или "S") с последующими ключом (одно слово) и остальной информацией, необходимой для выполнения приказа. Эта информация зависит от приказа и определяется следующим образом:

Ввести. Полная вводимая запись, включая ее ключ.

Удалить. Ключ удаляемой записи.

Изменить. Ключ изменяемой записи и пара номер поля-данные поля, где номер поля показывает изменяемое поле, а данные поля содержит данные для замены.

Остановить. Не требуется никакой дополнительной информации.

На рис. 4.48 представлен исходный модуль, содержащий UPDATE, который основывается на рассмотренном формате приказа. Приведены полный код только для модулей UPDATE и SEARCH, а также их сегменты данных.

```
  PUBLIC  UPDATE
  EXTRN   LOCATE:FAR, INPUTM:FAR, OUTPUTM:FAR
DATA_SEG    SEGMENT
    SRCH_CODE   DB      ?
    PRIOR_ELE   DW      ?
    CUR_ELE     DW      ?
    KEY         DW      ?
DATA_SEG    ENDS
COM_SEG     SEGMENT     COMMON
    D_ELEMENT   STRUC
        EMPNUM      DW      ?
        INFO        DW      3 DUP(?)
        POINTER     DW      ?
    D_ELEMENT   ENDS
    U_DATA      STRUC
        COMMAND     DB      ?
        COM_KEY     DW      ?
        COM_DATA    DB      256 DUP(?)
    U_DATA      ENDS
    DIRECTORY   D_ELEMENT 1000 DUP(<>)
    ERR_CODE    DB      ?
    UPDATE_DATA U_DATA  <>
COM_SEG     ENDS
UPDATE_SEG  SEGMENT
  ASSUME  CS:UPDATE_SEG, DS:DATA_SEG, ES:COM_SEG
UPDATE      PROC        FAR
                PUSH    AX                      ;ЗАПОМНИТЬ РЕГИСТРЫ В СТЕКЕ
                PUSH    BX
                PUSH    CX
                PUSH    DX
                PUSH    SI
                PUSH    DI
                MOV     ERR_CODE,0              ;СБРОСИТЬ ERR_CODE
                MOV     AX,UPDATE_DATA.COM_KEY  ;ПЕРЕДАТЬ КЛЮЧ ПРИКАЗА
                MOV     KEY,AX                  ;В AX И KEY
                CALL    NEAR PTR SEARCH         ;ИСКАТЬ КЛЮЧ
                MOV     DL,UPDATE_DATA.COMMAND  ;ПЕРЕДАТЬ ПРИКАЗ В DL
                CMP     DL,'I'                  ;ЕСЛИ БУКВА ПРИКАЗА НЕ I,
                JNZ     NOT_I                   ;ПЕРЕЙТИ К NOT_I,
                CALL    NEAR PTR INSERT         ;ИНАЧЕ ДЕЛАТЬ ВВЕДЕНИЕ
                JMP     SHORT EXIT              ;И ПЕРЕЙТИ К EXIT
    NOT_I:      CMP     DL,'D'                  ;ЕСЛИ БУКВА ПРИКАЗА НЕ D,
                JNZ     NOT_D                   ;ПЕРЕЙТИ К NOT_D
                CALL    NEAR PTR DELETE         ;ИНАЧЕ ДЕЛАТЬ УДАЛЕНИЕ
                JMP     SHORT EXIT              ;И ПЕРЕЙТИ К EXIT
    NOT_D:      CMP     DL,'C'                  ;ЕСЛИ БУКВА ПРИКАЗА НЕ C,
                JNZ     NOT_C                   ;ПЕРЕЙТИ К NOT_C
                CALL    NEAR PTR CHANGE         ;ИНАЧЕ ДЕЛАТЬ ИЗМЕНЕНИЕ
                JMP     SHORT EXIT              ;И ПЕРЕЙТИ К EXIT
    NOT_C:      MOV     ERR_CODE,1              ;УСТАНОВИТЬ ERR_CODE В 1
    EXIT:       POP     DI                      ;ВОССТАНОВИТЬ РЕГИСТРЫ
                POP     SI
                POP     DX
                POP     CX
                POP     BX
                POP     AX
                RET                             ;ВОЗВРАТ
UPDATE      ENDP

                .
                .
                .

SEARCH      PROC        NEAR
                MOV     SRCH_CODE,0             ;СБРОСИТЬ SRCH_CODE
                MOV     AX,KEY                  ;ПЕРЕДАТЬ KEY В AX
                CMP     AX,1000                 ;ПРОВЕРИТЬ НАХОЖДЕНИЕ КЛЮЧА
                JB      ERROR                   ;В ДИАПАЗОНЕ 1000...9999;
                CMP     AX,9999                 ;ЕСЛИ ОН ПРАВИЛЕН,
                JB      ERROR                   ;ПЕРЕЙТИ К OK
                JMP     SHORT OK
    ERROR:      MOV     ERR_CODE,2              ;ИНАЧЕ ЗАГРУЗИТЬ 2 В ERR_CODE
                JMP     SHORT NOT_FOUND         ;И ПЕРЕЙТИ К NOT_FOUND
    OK:         LEA     BX,DIRECTORY.EMPNUM     ;БАЗОВЫЙ АДРЕС В BX
                MOV     SI,ES:[BX+8]            ;УКАЗАТЕЛЬ В SI, ПРЕДЫДУЩИЙ
                MOV     PRIOR_ELE,0             ;ИНДЕКС В PRIOR_ELE
    AGAIN:      CMP     AX,ES:[BX][SI]          ;СРАВНИТЬ КЛЮЧИ И ПЕРЕЙТИ
                JE      FOUND                   ;К FOUND, ЕСЛИ ОНИ РАВНЫ
                JL      GREATER                 ;ЕСЛИ МЕНЬШЕ, ПЕРЕЙТИ К GREATER
                MOV     PRIOR_ELE,SI            ;ИНАЧЕ МОДИФИЦИРОВАТЬ
                MOV     SI,ES:[BX][SI+8]        ;PRIOR_ELE И SI
                JMP     SHORT AGAIN
```

```
    FOUND:      MOV     SRCH_CODE,1
    GREATER:    MOV     CUR_ELE,SI
    NOT_FOUND:  RET
SEARCH      ENDP
              .
              .
              .
UPDATE_SEG  ENDS
              END
```

Рис. 4.48. Код модулей UPDATE и SEARCH

Упражнения

1. Рассмотрите следующие исходные модули:

```
                    ИСХОДНЫЙ МОДУЛЬ  1

S_SEG       SEGMENT         STACK
              DW            100 DUP(?)
  TOP         LABEL         WORD
S_SEG       ENDS
D_SEG       SEGMENT         COMMON
  VAR         DB            50 DUP(?)
D_SEG       ENDS
E_SEG       SEGMENT         AT 1000H
  AREA        DW            70 DUP(0)
E_SEG       ENDS
C_SEG       SEGMENT         PUBLIC
              .
              .     }   500H БАЙТ
              .
C_SEG       ENDS

                    ИСХОДНЫЙ МОДУЛЬ  2

S_SEG       SEGMENT         STACK
              DW            50 DUP(?)
  TOP         LABEL         WORD
S_SEG       ENDS
D_SEG       SEGMENT         COMMON
  VECT        DW            30
D_SEG       ENDS
C_SEG       SEGMENT         PUBLIC
              .
              .     }   1000H БАЙТ
              .
C_SEG       ENDS
```

Предположим, что редактор связей размещает сегменты в порядке S_SEG, D_SEG и C_SEG и низ стека начинается по адресу 20000. Точно покажите, каким образом сегменты будут размещены в памяти загрузчиком.

2. Напишите операторы, которые объявляют слово с именем VAR1 и метку LAB1 типа NEAR как внешние, а перемениую VAR2 и метку LAB2 как локальные, но доступные другим исходным модулям.

3. При каких обстоятельствах метке в операторе EXTRN придается атрибут NEAR?

4. Предположим, что:

а) двойное слово с именем VAR1, массив байт с именем VAR2 и метка LAB1 типа NEAR определяются в исходном модуле 1, но используются исходными модулями 2 и 3;

б) слово с именем VAR3 и метка LAB2 типа FAR определяются в исходном модуле 2, причем VAR3 используется исходным модулем 1 и LAB2 – исходным модулем 3;

в) метка LAB3 типа FAR определяется в исходном модуле 3 и используется исходным модулем 2.

Дайте для каждого модуля необходимые операторы EXTRN и PUBLIC и покажите, каким образом редактор связей осуществляет соответствие переменных и меток (см. рис. 4.5).

5. Для приведенного далее кода покажите, какие сегментные адреса определяются ассемблером, а какие – редактором связей (ответы обоснуйте):

```
        EXTRN     COST:WORD,ROUTINE:FAR
        PUBLIC    BEGIN
DATA_1  SEGMENT
  TOTAL   DW      50 DUP(?)
  NUM     DW      ?
DATA_1  ENDS
DATA_2  SEGMENT
  PART    DW      100 DUP(?)
DATA_2  ENDS
CODE    SEGMENT
          .
          .
          .
          MOV     AX,DATA_1
          MOV     DX.AX
          .
          .
          .
          MOV     AX,SEG COST
          MOV     ES,AX
          MOV     AX,TOTAL
          ADD     AX,ES:COST
          .
          .
          .
          MOV     AX,DATA_2
          MOV     ES,AX
          INC     ES:PART[SI]
          MOV     CX,NUM
          CMP     TOTAL[DI],CX
          JE      NEXT
          JMP     FAR PTR ROUTINE
  NEXT:   .
          .
          .
CODE      ENDS
```

6. Приведите код, который загружает в регистр SI значение внешне определенной переменной COUNT.

7. Требуется ли межсегментный переход к внешне определенной метке предварять командами загрузки регистра CS? Почему?

8. Приведите код, позволяющий в исходном модуле 1 обращаться к переменным-словам NUM1, NUM2, NUM3 и NUM4, определенным в исходном модуле 2, так, как будто они определены в исходном модуле 1.

9. Пусть имеется код:

```
S_SEG     SEGMENT     AT      1000H
            DW        200 DUP(?)
  TOS       LABEL     WORD
S_SEG     ENDS
            .
            .
            .
C_SEG     SEGMENT
  ASSUME  CS:C_SEG,   SS:S_SEG
            MOV       AX,S_SEG
            MOV       SS,AX
            MOV       SP,OFFSET TOS
          .
          .
          .
            PUSH      T_ADDR
            PUSH      AX
            PUSHF
          .
          .
          .
            POPF
            POP       AX
            POP       T_ADDR
          .
          .
          .
C_SEG     ENDS
```

Дайте рисунок стека, показывающий его важные точки и адреса этих точек; опишите действия команд включений в стек и извлечений из стека.

10. Постройте последовательность команд, которые включают в стек смещения X, Y и Z.

11. Предположим, что программа повторно использует некоторый код и что этот код должен иметь стек, отличный от стека остальной части программы. Приведите команды, необходимые в начале и конце кода для коммутации стека. Дайте также код для определения двух стеков и инициализации (SS) и (SP).

12. Почему в операторе PROC должен указываться атрибут NEAR или FAR?

13. Рассмотрите следующий ряд вызовов:

а) MAIN вызывает процедуру SUBA типа NEAR и смещение возврата равно 0400;

б) SUBA вызывает процедуру SUBB типа NEAR и смещение возврата равно 0A00;

в) SUBB вызывает процедуру SUBC типа FAR; смещение возврата равно 0100 и сегментный адрес возврата равен B200;

г) возврат из SUBC в SUBB;

д) SUBB вызывает процедуру SUBD типа NEAR и смещение возврата равно 0C00;

е) возврат из SUBD в SUBB;

ж) возврат из SUBB в SUBA;

з) возврат из SUBA в MAIN;

и) MAIN вызывает процедуру SUBC; смещение возврата равно 0600 и сегментный адрес возврата равен 1000.

Считая, что стек действует только в вызовах и возвратах, приведите серию диаграмм, подобных показанным на рис. 4.16.

14. Напишите процедуру COMPUTE для вычисления

$$R \leftarrow X + Y - 3$$

полагая, что X, Y и R содержатся в словах, а COMPUTE находится в том же сегменте кода, что и вызывающая программа. Приведите также определения сегмента данных D_SEG, который содержит X и Y, сегмента данных E_SEG (содержащего R), начальную часть вызывающей программы и вызов COMPUTE. Модифицируйте код для случая, когда COMPUTE находится в том же исходном модуле, но в другом сегменте кода, чем вызывающая программа. Еще раз модифицируйте код, считая, что COMPUTE находится в другом исходном модуле и в другом сегменте, чем вызывающая программа, и что для взаимодействия применяется таблица параметров.

15. Напишите процедуру MULT для беззнакового двоичного умножения двойных слов A и B и размещения 64-битного результата в четырех смежных словах, начиная с PROD. Процедура MULT является единственным кодом в ее исходном модуле, а A, B и PROD находятся в общей области COM_SEG. Приведите определения COM_SEG в исходном модуле вызывающей программы, где переменные именуются X, Y и Z, и в исходном модуле процедуры. Наконец, реализуйте типичный вызов MULT.

16. Напишите процедуру SEARCH типа FAR, которая отыскивает в массиве байт заданный байт; в случае успеха параметр-слово содержит индекс элемента в массиве, а при неудаче в параметр загружается −1. Параметры передаются SEARCH через таблицу адресов параметров. Приведите код вызова SEARCH для поиска в ARY значения (ID) и загрузки индекса в IDX.

17. Перепишите процедуру и вызывающую последовательность в упр. 16 так, чтобы адреса параметров передавались через стек.

18. Для какой цели в поле операнда команды RET допускается употребление выражения?

19. Напишите набор процедур для выполнения беззнаковых двоичных арифметических операций над двоичными словами. Процедуры находят операнды включенными в

стек непосредственно перед вызовом и возвращают результаты в регистрах DX:AX. В набор входят процедуры:

а) ADDITION. Суммирует два операнда.

б) SUBTRACT. Вычитает два операнда. Уменьшаемое включается в стек первым.

в) MULTIPLY. Умножает два операнда и возвращает только младшие 32 бита.

г) DIVIDE. Делит два операнда и возвращает только частное. Делимое включается в стек первым.

20. Напишите процедуру EVALUATE, которая использует процедуры из упр. 19, для вычисления выражений:

$$A * X + B \qquad X * (X + Y - Z)$$

В каждом случае непосредственно перед вызовом процедуры параметры включаются в стек в том порядке, в каком они следуют в выражении.

21. Напишите процедуру BCD_ADDSUB, которая суммирует или вычитает два 16-разрядных BCD-числа. Процедура получает через стек начальные адреса операндов и адрес результата. Отрицательные числа представлены в десятичном дополнительном коде. Кроме того, через стек передается флажок, указывающий выполняемую операцию. Вычитание реализуется прибавлением к уменьшаемому десятичного дополнительного кода вычитаемого. Десятичный дополнительный код формируется второй процедурой TENS_COMP, которая получает начальный адрес числа в регистре BX. Процедура TENS_COMP заменяет переданный ей параметр результатом.

22. Напишите рекурсивную процедуру PEVAL для вычисления полинома по правилу Горнера:

$$Y \leftarrow A_0 + A_1 X + \ldots + A_N X^N.$$

Коэффициенты $A_0, \ldots, A_N$ находятся в смежных словах памяти и адреса всех параметров передаются через стек.

23. Дано содержимое регистров (SP) = 0100, (SS) = 0300, (PSW) = 0240 и ячеек памяти:

00020 0040
00022 0100

Далее, команда INT 8 имеет смещение 00A0 в сегменте с адресом 0900. Определите содержимое SP, SS, IP, CS, PSW и трех верхних слов стека после выполнения последовательности прерывания, инициированной командой INT 8.

24. Проанализируйте различия в отладке с помощью прерываний, вызываемых командами INT, INT *Тип*, а также прерываниями пошаговой работы. Покажите, когда каждое из этих прерываний предпочтительнее двух других.

25. Пусть сразу после выполнения команды INTO (смещение ее равно 0500, а сегментный адрес B100) PSW содержит 0180. Что будет находиться в вершине стека после завершения последовательности прерывания?

26. Дайте определение макрокоманды DADD, которая суммирует два операнда длиной в три слова из ячеек памяти и запоминает результат также в памяти. Фиктивные операнды ассоциируются с младшими словами операндов и результата. Постройте расширения, получающиеся от следующих вызовов:

```
%DADD  (OPR,PRICE[SI],TOTAL)
.
.
.
%DADD  (TOTAL,TOTAL,TAX + 4)
```

27. Дайте определение макрокоманды RESTORE, которая извлекает из стека содержимое регистров, включенное в стек в таком порядке: AX, BX, CX, DX, SI и DI.

28. Можно ли в макрокоманде ABSOL из раздела 4.5.2 заменить команду JGE %NEXT на команду JGE $ + 4 и избежать локальной метки? Ответ обоснуйте.

29. Напишите макрокоманду LINE, которая вычисляет

$$Y \leftarrow AX + B$$

с двойными словами посредством вызова макрокоманды DMULT для выполнения умножения и макрокоманды DADD для выполнения сложения.

30. Задан код:

```
%*DEFINE    (ABSDIF  (V1,V2,V3))  LOCAL  CONT
(                PUSH      AX
%DIF    (%V1,%V2)
                 CMP       AX,0
                 JGE       CONT
                 NEG       AX
%CONT:           MOV       %V3,AX
                 POP       AX
)
%*DEFINE    (DIF  (X,Y))
(                MOV       AX,%X
                 SUB       AX,%Y
)
```

Найдите расширения следующих вызовов и укажите недопустимые вызовы;

а) %ABSDIF (P1,P2,DISTANCE)

б) %ABSDIF ([BX][SI],X[DI],CX)

в) %ABSDIF ([BX][SI],X[BX][SI],240H)

г) %ABSDIF (AX,AX,AX)

31. Постройте набор макроопределений для выполнения логических операций OR, AND и XOR над двойными словами, местоположение которых указывается первыми двумя фиктивными параметрами. Результат помещается в регистры DX:AX. Сначала решите задачу без вложения макроопределений, а затем повторите решение с вложением определений.

32. Напишите фрагмент кода, который вызывает ассемблирование команды MOV TERMINAL,0, если фиктивный параметр X = 'VT55'; в противном случае ассемблируется команда MOV TERMINAL,1.

33. Напишите фрагмент кода, который вызывает ассемблирование команды ADD AX,AX десять раз, если длина цепочки, заданной именем X, больше 5.

34. Определите макрокоманду, которая формирует код для сложения двух двоичных N-байтных операндов и запоминания N-байтного результата, начиная с произвольной ячейки. N является именем константы и фигурирует как четвертый фиктивный параметр.

35. Определите макрокоманду для пересылки произвольной символьной цепочки, которая заканчивается символом EOF, из одной области памяти в другую.

36. По следующему псевдокоду постройте подробную схему, которая четко показывает элементарные конструкции:

```
CODE ⟵ 0
IF A=0  THEN
        IF B=0  THEN
                CODE ⟵ 1
            ELSE
                CALL DIVIDE  вычислить  X1  ⟵  -C/B
        ENDIF
    ELSE
        CALL EVAL  вычислить  E  ⟵  -B/2A
        CALL DISCRIM  вычислить  D  ⟵  (B/2A)^2  -C/A
        IF D<0  THEN
                CODE ⟵ 2
            ELSE IF D=0  THEN
                X1 ⟵ E
                CODE ⟵ 3
            ELSE
                CALL SQRDOT  вычислить  F  ⟵  √D
                CALL SUBTRACT  вычислить  X1  ⟵  E-F
                CALL ADDITION  вычислить  X2  ⟵  E+F
        ENDIF
ENDIF
```

37. Рассмотрите проблему использования детерминантов для решения уравнений:

$$A_{11}X_1 + A_{12}X_2 = B_1,$$
$$A_{21}X_1 + A_{22}X_2 = B_2.$$

Все величины являются 16-разрядными упакованными BCD-числами в десятичном дополнительном коде. Предположим, что ввод, решение и вывод реализуют отдельные модули и модуль решения использует подмодуль DETEVAL для вычисления детерминанта. Пусть арифметические операции и преобразования упакованного/неупакованного форматов выполняются отдельными модулями. Постройте иерархическую диаграмму программы, дайте описания модулей SOLUTION, DETEVAL и MULTIPLY, постройте псевдокод и схему модуля SOLUTION и, наконец, напишите ассемблерный код модуля SOLUTION.

38. Поясните, каким образом программные прерывания можно использовать для выполнения арифметических операций в ситуациях, показанных в упр. 37.

39. Напишите псевдокод для модуля DELETE (рис. 4.44) и преобразуйте его в ассемблерный код микропроцессора 8086.

5. МАНИПУЛЯЦИИ БАЙТАМИ И ЦЕПОЧКАМИ

Одно из важных применений компьютеров связано с *обработкой текстов*, которая представляет собой манипуляции последовательностями байт, содержащих коды символов, т. е. символьными цепочками. При разработке редактора текста или программы форматирования текста требуются программы пересылок и сравнений символьных цепочек, введения цепочек в другие цепочки и удаления цепочек из других цепочек. Часто необходимо отыскивать в цепочке заданную подцепочку или заменять подцепочку другой подцепочкой. Редактор текста требуется в любой вычислительной системе широкого назначения, и его возможности, улучшающие операции системы над символьными цепочками, имеют большое значение. Рассмотрим те возможности микропроцессора 8086, которые упрощают обработку символьных цепочек.

Хотя большинство приводимых команд ориентированы на операции с символьными цепочками, их можно применять и для других целей, например в файловой системе для поиска в массивах ключей. Эти команды не выпол-

няют функций, которые невозможно реализовать другими командами; они предназначены только для более эффективного выполнения некоторых действий, которые потенциально оказываются весьма длительными (например, поиск заданных байта, слова или подцепочки в цепочке или массиве, содержащих тысячи байт).

Иногда требуется переходить от одной системы кодирования к другой. Например, в системе может быть терминал, работающий в символьном коде EBCDIC, а вся система спроектирована на работу с кодом ASCII. В этом случае передаваемые и принимаемые от терминала символы необходимо преобразовывать. В микропроцессоре 8086 проблема преобразования кода решена введением команды XLAT.

В § 5.1 определяются цепочечные команды микропроцессора 8086. В § 5.2 обсуждается префикс повторения REP, который применяется вместе с цепочечными командами для ускорения некоторых операций, обычно реализуемых циклами; § 5.3 содержит пример редактирования текста, опирающийся на материал предыдущих параграфов; § 5.4 посвящен команде XLAT и ее применениям, а в § 5.5 речь идет о преобразованиях формата.

5.1. ЦЕПОЧЕЧНЫЕ КОМАНДЫ

Цепочечные команды приведены на рис. 5.1. Так как без префикса REP они оперируют только одним байтом или словом, их называют *примитивами*. Все команды имеют длину один байт, и бит 0 показывает операцию с байтом (бит 0 = 0) или словом (бит 0 = 1).

Любой из пяти базовых примитивов допускает один из трех следующих форматов:

Операция Операнд(ы)

или

ОперацияB

или

ОперацияW

В первом формате обработка байт или слов определяется неявно типом операнда (операндов). Второй и третий форматы явно указывают операцию над байтами или словами.

Независимо от формата примитива фактические операнды определяются содержимым регистров SI, DI, DS и ES. Адрес операнда-источника равен сумме (SI) и (DS) × 16_{10}, если только регистр DS не заменен явным определением ES, CS или SS как сегментного регистра. Адрес операнда-получателя всегда равен сумме (DI) и (ES) × 16_{10}. Такая регистровая косвенная адресация означает, что смещение(я) источника и(или) получателя следует загрузить в регистр(ы) SI и(или) DI до выполнения примитива. Необходимо уяснить, что, если в примитивах MOVS и CMPS цепочки находятся в одном и том же сегменте и DS используется в качестве сегментного адреса источника, регистры DS и ES должны содержать один и тот же сегментный адрес. Основные причины указания операнда(ов) в примитивах заключаются в том, что наличие идентификатора(ов) источника (и получателя) делает программу

НАЗВАНИЕ	МНЕМОНИКА И ФОРМАТ *		ОПИСАНИЕ **
ПЕРЕДАТЬ ЦЕПОЧКУ	MOVS	DST,SRC	((DI))<— ((SI)) ОПЕРАНДЫ-БАЙТЫ (SI)<— (SI)+1, (DI)<— (DI)+1 ОПЕРАНДЫ-СЛОВА (SI)<— (SI)+2, (DI)<— (DI)+2
ПЕРЕДАТЬ ЦЕПОЧКУ БАЙТ	MOVSB		
ПЕРЕДАТЬ ЦЕПОЧКУ СЛОВ	MOVSW		
СРАВНИТЬ ЦЕПОЧКИ	CMPS	SRC,DST	((SI)) - ((DI)) ОПЕРАНДЫ-БАЙТЫ (SI)<— (SI)+1, (DI)<— (DI)+1 ОПЕРАНДЫ-СЛОВА (SI)<— (SI)+2, (DI)<— (DI)+2
СРАВНИТЬ ЦЕПОЧКИ БАЙТ	CMPSB		
СРАВНИТЬ ЦЕПОЧКИ СЛОВ	CMPSW		
СКАНИРОВАТЬ ЦЕПОЧКУ	SCAS	DST	ОПЕРАНД-БАЙТ (AL) - ((DI)), (DI)<— (DI)+1 ОПЕРАНД-СЛОВО (AX) - ((DI)), (DI)<— (DI)+2
СКАНИРОВАТЬ ЦЕПОЧКИ БАЙТ	SCASB		
СКАНИРОВАТЬ ЦЕПОЧКИ СЛОВ	SCASW		
ЗАГРУЗИТЬ ЦЕПОЧКУ	LODS	SRC	ОПЕРАНД-БАЙТ (AL)<— ((SI)), (SI)<— (SI)+1 ОПЕРАНД-СЛОВО (AX)<— ((SI)), (SI)<— (SI)+2
ЗАГРУЗИТЬ ЦЕПОЧКУ БАЙТ	LODSB		
ЗАГРУЗИТЬ ЦЕПОЧКУ СЛОВ	LODSW		
ЗАПОМНИТЬ ЦЕПОЧКУ	STOS	DST	ОПЕРАНД-БАЙТ ((DI))<—(AL), (DI)<— (DI)+1 ОПЕРАНД-СЛОВО ((DI))<—(AX), (DI)<— (DI)+2
ЗАПОМНИТЬ ЦЕПОЧКУ БАЙТ	STOSB		
ЗАПОМНИТЬ ЦЕПОЧКУ СЛОВ	STOSW		

* СУФФИКС B ПОКАЗЫВАЕТ ОПЕРАНДЫ-БАЙТЫ, A W — ОПЕРАНДЫ-СЛОВА.
** ИНКРЕМЕНТ (+), ЕСЛИ DF=0; ДЕКРЕМЕНТ (-), ЕСЛИ DF=1.

ФЛАЖКИ. КОМАНДЫ CMPS И SCAS ВОЗДЕЙСТВУЮТ НА ВСЕ ФЛАЖКИ УСЛОВИЙ, А ОСТАЛЬНЫЕ НЕ МОДИФИЦИРУЮТ НИКАКИЕ ФЛАЖКИ.

РЕЖИМЫ АДРЕСАЦИИ. ОПЕРАНДЫ ПОДРАЗУМЕВАЮТСЯ (НЕЯВНЫЕ).

Рис. 5.1. Цепочечные примитивы

более понятной и позволяет ассемблеру проконтролировать адресуемость операнда(ов). Применение различных сегментных регистров для источника и получателя позволяет размещать две цепочки в разных сегментах. Но, чтобы избежать коммутации сегментных регистров, в небольшой программе можно определить все операнды в одном сегменте. В частности, этот прием используется, когда имеются операнды-цепочки для источника и получателя.

Кроме выполнения указанной операции, примитив вызывает автоматический инкремент или декремент регистров SI и (или) DI. Такое автоиндексирование допускает повторяющееся использование примитива для последовательной обработки смежных байт или слов. Осуществление автоинкремента или автодекремента определяется флажком DF в PSW: если DF = 1, производится автодекремент, а если DF = 0 — автоинкремент. Когда, например, DF = 0, команда MOVSB пересылает байт из (SI) + (DS) × 16_{10} в (DI) + + (ES) × 16_{10} и производит инкремент SI и DI на 1. Поэтому, если этот примитив находится в цикле, его новое выполнение вызывает пересылку следующего байта.

При работе с цепочками команды MOVS и CMPS имеют следующие преимущества по сравнению с командами MOV и CMP:

1. Длина команд MOVS и CMPS составляет всего один байт.

2. Оба операнда находятся в памяти.

3. Автоиндексирование делает ненужными явные команды инкремента или декремента, что повышает производительность.

Для примера рассмотрим пересылку цепочки из одной области памяти в другую. На рис. 5.2, *а* показано решение с командой MOV, которая не может осуществлять передачу память — память. Решение с применением команды

```
        MOV     SI,OFFSET STRING1   ;SI СЛУЖИТ ИНДЕКСОМ ИСТОЧНИКА
        MOV     DI,OFFSET STRING2   ;DI СЛУЖИТ ИНДЕКСОМ ПОЛУЧАТЕЛЯ
        MOV     CX,LENGTH STRING1   ;ЗАГРУЗИТЬ ДЛИНУ В CX
MOVE:   MOV     AL,[SI]             ;ПЕРЕДАТЬ БАЙТ ИЗ ИСТОЧНИКА
        MOV     [DI],AL             ;В ПОЛУЧАТЕЛЬ
        INC     SI                  ;ИНКРЕМЕНТ ИНДЕКСА ИСТОЧНИКА
        INC     DI                  ;ИНКРЕМЕНТ ИНДЕКСА ПОЛУЧАТЕЛЯ
        LOOP    MOVE
```

а)

```
        MOV     SI,OFFSET STRING1   ;ПРЕДПОЛАГАЕТСЯ, ЧТО (DS)=(ES)
        MOV     DI,OFFSET STRING2
        MOV     CX,LENGTH STRING1
        CLD                         ;СБРОСИТЬ ФЛАЖОК НАПРАВЛЕНИЯ
MOVE:   MOVS    STRING2,STRING1     ;ДЛЯ АВТОИНКРЕМЕНТА
        LOOP    MOVE
```

б)

Рис. 5.2. Фрагменты передачи блока данных с командами MOV (*а*) и MOVS (*б*)

MOVS представлено на рис. 5.2, *б*. Отметим, что вторая программа может пересылать байты или слова в зависимости от типа STRING1 и STRING2. В § 5.2 показано, что данную задачу можно реализовать эффективнее, пользуясь префиксом REP, который делает ненужным явный цикл.

Программа на рис. 5.3 показывает применение флажка DF при пересылке данных из одной области в перекрывающуюся область. Здесь предполагает-

```
        MOV     SI,SRCADDR          ;ПРЕДПОЛАГАЕТСЯ, ЧТО (DS)=(ES)
        MOV     DI,DSTADDR
        MOV     CX,N                ;ЗАГРУЗИТЬ ДЛИНУ В CX
        CLD                         ;СБРОСИТЬ ФЛАЖОК НАПРАВЛЕНИЯ.
        CMP     SI,DI               ;ЕСЛИ АДРЕС STRG1 БОЛЬШЕ АДРЕСА
        JA      MOVE                ;STRG2, НАЧАТЬ С ПЕРВОГО
        STD                         ;ЭЛЕМЕНТА; В ПРОТИВНОМ СЛУЧАЕ
        ADD     SI,CX               ;НАЧАТЬ С ПОСЛЕДНЕГО ЭЛЕМЕНТА
        DEC     SI
        ADD     DI,CX
        DEC     DI
MOVE:   MOVSB
        LOOP    MOVE
          .
          .
          .
```

Рис. 5.3. Передача блока данных между перекрывающимися областями

ся, что SRCADDR, DSTADDR и N хранят соответственно адрес источника, адрес получателя и число пересылаемых байт. Имеются две возможные ситуации (рис. 5.4): адрес источника больше адреса получателя и тогда первым необходимо пересылать первый элемент источника; адрес источника меньше адреса получателя, что требует передачи первым последнего элемента источника.

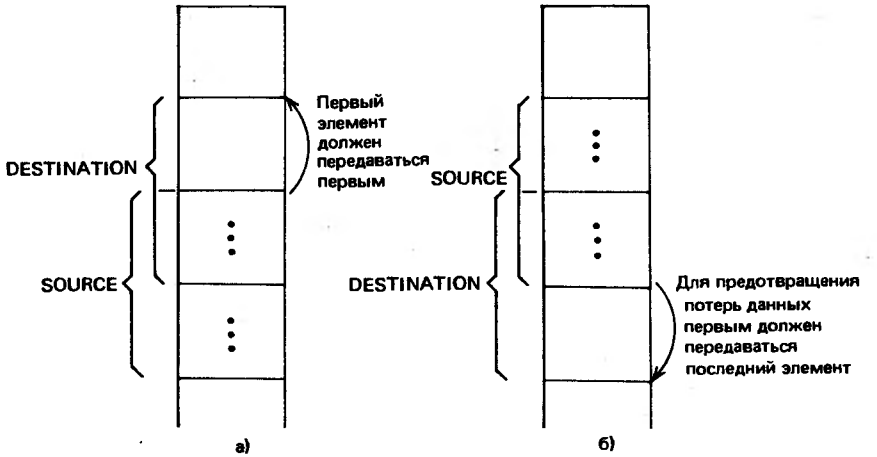


Рис. 5.4. Возможные случаи при передаче данных между перекрывающимися областями, когда начальный адрес получателя меньше (*а*) и больше (*б*) начального адреса источника

Примитив CMPS предназначен для сравнения цепочек байт или слов произвольной длины. Фрагмент на рис. 5.5 осуществляет переход к SAME, если

```
        MOV     SI,OFFSET STRG1
        MOV     DI,OFFSET STRG2
        MOV     CX,LENGTH STRG1
        CLD
NEXT:   CMPS    STRG1,STRG2         ;СРАВНИВАТЬ STRG1 И STRG2
        JNE     EXIT                ;ПЕРЕЙТИ К SAME, ЕСЛИ ОНИ РАВНЫ
        LOOP    NEXT
        JMP     NEAR PTR SAME
EXIT:   .
        .
        .
```

Рис. 5.5. Сравнение цепочек

две цепочки равны, и к EXIT в противном случае. Как и программу пересылки цепочки, эту программу можно значительно улучшить с помощью префикса REP.

Отметим, что при указании операндов в примитиве MOVS операнд, ассоциируемый с регистром SI, находится последним, а в примитиве CMPS — первым. Кроме того, MOVS не воздействует на флажки, а назначение CMPS — установить флажки для принятия последующих решений.

Оставшиеся примитивы SCAS, LODS и STOS имеют по одному операнду в памяти, причем только SCAS воздействует на флажки. Программа с исполь-

```
        MOV     DI,OFFSET LINE      ;ЗАГРУЗИТЬ АДРЕС LINE В DI
        MOV     CX,80               ;И СЧЕТЧИК В CX
        MOV     AL,20H              ;ЗАГРУЗИТЬ В AL КОД ПРОБЕЛА
        CLD                         ;НАЧАТЬ С ПЕРВОГО ЭЛЕМЕНТА
NEXT:   SCAS    LINE                ;ЕСЛИ В LINE ТОЛЬКО ПРОБЕЛЫ,
        LOOPE   NEXT                ;ПЕРЕЙТИ К NOT_FOUND, ИНАЧЕ
        JE      NOT_FOUND           ;ПЕРЕСЫЛАТЬ ЦЕПОЧКУ
        MOV     SI,DI               ;ПЕРЕДАТЬ В SI АДРЕС ПЕРВОГО
        DEC     SI                  ;СИМВОЛА - НЕ ПРОБЕЛА.
        MOV     DI,OFFSET SYMBOL    ;АДРЕС SYMBOL В DI
        MOV     CX,31               ;И СЧЕТЧИК В CX
```

```
FILL:   STOS    SYMBOL              ;ЗАПОМНИТЬ SYMBOL ПРОБЕЛАМИ
        LOOP    FILL
        MOV     DI,OFFSET SYMBOL    ;УСТАНОВИТЬ DI
        MOV     CX,31               ;И CX
        JMP     SHORT SCANE
MOVE:   STOS    SYMBOL
SCANE:  LODS    LINE                ;ПЕРЕДАВАТЬ ЦЕПОЧКУ В SYMBOL
        CMP     AL,20H              ;ДО ОБНАРУЖЕНИЯ ПРОБЕЛА
        LOOPNE  MOVE                ;ИЛИ ПЕРЕДАЧИ 31 СИМВОЛА
          .
          .
          .
```

Рис. 5.6. Пример использования примитивов SCAS, STOS и LODS

зованием этих примитивов приведена на рис. 5.6. В ней предполагается, что операнды определены в одном и том же сегменте, адресуемом DS и ES, и сканируется цепочка из 80 символов, заканчивающаяся пробелом. Если цепочка содержит только пробелы, осуществляется переход на NOT_FOUND; в противном случае, начиная с первого символа, отличающегося от пробела, подцепочка LINE длиной до 31 символа пересылается в SYMBOL. Подцепочка может заканчиваться пробелом или после передачи первых 31 символов.

5.2. ПРЕФИКС ПОВТОРЕНИЯ

Так как операции с цепочками предполагают зацикливание, в машинном языке микропроцессора 8086 предусмотрен префикс, значительно упрощающий реализацию циклов с примитивами. Машинный код префикса:

1	1	1	1	0	0	1	Z

Бит Z помогает управлять циклом с примитивами CMPS и SCAS. С префиксом REP (= 11110011) примитивы MOVS, LODS и STOS, не влияющие на флажки, повторяются число раз, определяемое содержимым регистра CX в соответствии со следующими этапами:

1. Если (CX) = 0, закончить операцию.
2. Декремент CX на 1.
3. Выполнить указанный примитив.
4. Повторить этапы 1 – 3.

В примитивах CMPS и SCAS, которые воздействуют на флажки, префикс вызывает их повторение число раз, определяемое содержимым регистра CX, или до тех пор, пока бит Z будет не соответствовать флажку ZF – что произойдет первым. Сравнение бита Z и флажка ZF осуществляется после каждого повторения. В любом случае при каждом повторении происходит декремент CX, и если несоответствие бита Z флажку ZF не вызывает прекращения, цикл закончится при достижении CX нуля. Следовательно, после повторяющегося выполнения примитивов CMPS и SCAS причину окончания можно проверить по состоянию флажка ZF.

В языке ассемблера префикс формируется при введении перед примитивом соответствующей мнемоники. Мнемоники префикса REP определены на рис. 5.7.

НАЗВАНИЕ	МНЕМОНИКА И ФОРМАТ	УСЛОВИЕ ОКОНЧАНИЯ
ПОВТОРЯТЬ ЦЕПОЧЕЧНУЮ ОПЕРАЦИЮ ДО CX = 0	REP ЦЕПОЧЕЧНЫЙ ПРИМИТИВ *	(CX)=0
ПОВТОРЯТЬ ЦЕПОЧЕЧНУЮ ОПЕРАЦИЮ, ПОКА РАВНО ИЛИ НУЛЬ	REPE ЦЕПОЧЕЧНЫЙ ПРИМИТИВ ** ИЛИ REPZ	(CX)=0 ИЛИ (ZF)=0
ПОВТОРЯТЬ ЦЕПОЧЕЧНУЮ ОПЕРАЦИЮ, ПОКА НЕ РАВНО ИЛИ НЕ НУЛЬ	REPNE ЦЕПОЧЕЧНЫЙ ПРИМИТИВ ** ИЛИ REPNZ	(CX)=0 ИЛИ (ZF) 1

* MOVS, LODS ИЛИ STOS
** CMPS ИЛИ SCAS

ПРИМЕЧАНИЕ. ВО ВСЕХ СЛУЧАЯХ ПРИ КАЖДОМ ПОВТОРЕНИИ (CX) ← (CX)-1.

Рис. 5.7. Префикс повторения

Чтобы проиллюстрировать применение префикса REP, вернемся к программе на рис. 5.2, *б*. При замене явного цикла

```
MOVE:  MOVS  STRING1,STRING2
       LOOP  MOVE
```

на

```
REP    MOVS  STRING1,STRING2
```

не только упрощается код, но и сокращается время выполнения с 18 + 17 = = 35 тактов синхронизации на итерацию до 9 + 17 = 26 тактов синхронизации для первой итерации и 17 тактов синхронизации на каждую последующую.

Далее, последние 4 команды в примере на сравнение цепочек (см. рис. 5.5)

```
NEXT:  CMPS  STR1,STR2
       JNE   EXIT
       LOOP  NEXT
       JMP   NEAR PTR SAME
```

можно заменить на команды

```
REPE   CMPS  STR1,STR2
       JNE   EXIT
       JMP   NEAR PTR SAME
```

Здесь время сокращается с 43 тактов синхронизации на итерацию до 22. Экономия времени оказывается значительной, если этот фрагмент используется для поиска подцепочки в очень длинной цепочке и находится, следовательно, внутри внешнего цикла.

Еще два примера с примитивами и префиксом REP приведены на рис. 5.8 и 5.9. Фрагмент на рис. 5.8 просматривает массив TABLE из 20-байтных цепочек до тех пор, пока первые 8 байт элемента массива не совпадут с цепочкой, начинающейся в SYMBOL. Смещения и сегментные адреса TABLE и SYMBOL находятся соответственно в TABLEPT и SYMBOLPT. Предполагается, что первые 8 байт элемента массива содержат имя, а остальные — информацию, относящуюся к этому имени. Фрагмент на рис. 5.9 сканирует цепочку, начинающуюся в ASCII_STG и заканчивающуюся точкой, и заменяет знак долла-

```
                MOV     BX,TABLE_SIZE       ;ЗАГРУЗИТЬ В BX ЧИСЛО ЭЛЕМЕНТОВ
                LES     DI,TABLEPT          ;ЗАГРУЗИТЬ СМЕЩЕНИЕ И СЕГМЕНТНЫЙ
                                            ;АДРЕС ТАБЛИЦЫ
                MOV     DX,DI               ;ЗАПОМНИТЬ СМЕЩЕНИЕ В DX
                LDS     SI,SYMBOLPT         ;ЗАГРУЗИТЬ СМЕЩЕНИЕ И СЕГМЕНТНЫЙ
                                            ;АДРЕС ОТЫСКИВАЕМОГО ИМЕНИ ДЛЯ
                CLD                         ;НАЧАЛА С ПЕРВОГО ЭЛЕМЕНТА
LOOP1:          MOV     CX,8
                REPE    CMPSB               ;СРАВНИВАТЬ ДВА СООТВЕТСТВУЮЩИХ
                JE      FOUND               ;СИМВОЛА ДО ИХ РАЗЛИЧИЯ
                ADD     DX,20               ;УКАЗАТЕЛЬ НА СЛЕДУЮЩИЙ ЭЛЕМЕНТ
                MOV     DI,DX
                MOV     SI,OFFSET SYMBOL    ;УКАЗАТЕЛЬ ПЕРВОГО СИМВОЛА
                DEC     BX
                JNE     LOOP1
NOT_FOUND:        .
                  .
                  .
FOUND:            .
                  .
                  .
                  .
```

Рис. 5.8. Поиск в таблице заданного 8-символьного имени

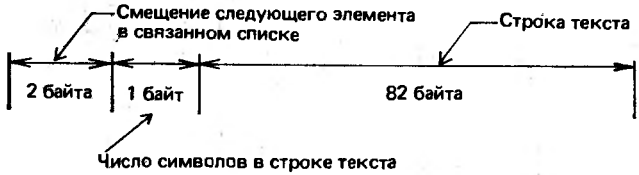
```
                MOV     CX,LENGTH ASCII_STG ;ИНИЦИАЛИЗИРОВАТЬ ПОИСК
                MOV     AL,'$'
                MOV     DI,OFFSET ASCII_STG
                CLD
LOOP1:          REPNZ   SCAS ASCII_STG      ;ОТЫСКИВАТЬ ЗНАК '$'
                JNZ     DONE                ;И ЗАМЕНЯТЬ ЕГО НА ЗНАК
                MOV     BYTE PTR [DI-1],'_' ;ПОДЧЕРКИВАНИЯ '_'
                JMP     SHORT LOOP1
DONE:             .
                  .
                  .
```

Рис. 5.9. Замена в символьной цепочке каждого знака доллара на значок подчеркивания

ра ($) на символ подчеркивания (_). Предполагается, что DS и ES содержат сегментный адрес, ассоциируемый с массивом ASCII_STG байт.

5.3. ПРИМЕР РЕДАКТОРА ТЕКСТА

Рассмотрим простой редактор текста, предназначенный для создания, модификации и запоминания до 400 строк текста, содержащих до 82 символов в строке. Текст хранится в виде связанного списка, все элементы которого имеют следующий формат:



Первый элемент в связанном списке должен содержать нулевые символы и его нельзя удалять. Последний элемент указывает на первый элемент, т. е. список организован в кольцо. Последние два символа в каждой строке, кроме первой, должны быть комбинацией возврата каретки и перевода строки.

Иерархическая диаграмма программы представлена на рис. 5.10. Модуль EDITOR выводит стимулирующий символ "?", запрашивая приказ. Он вызывает INPUT_COM для ввода приказа, а затем в зависимости от первой буквы

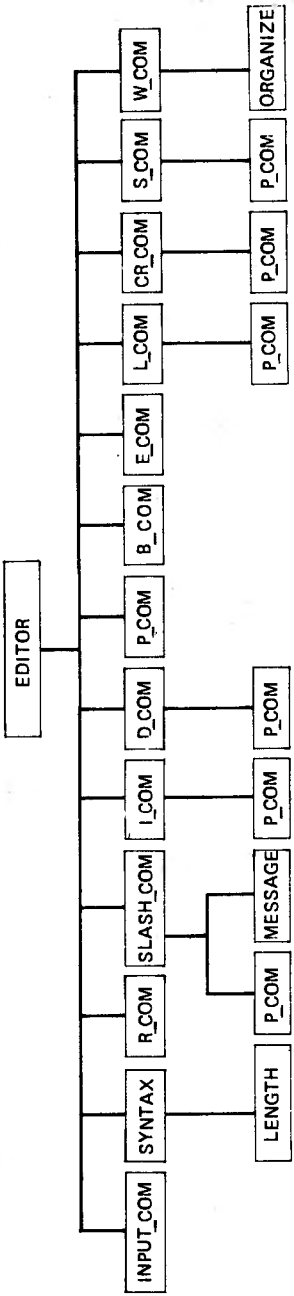


Рис. 5.10. Иерархическая диаграмма редактора текста

приказа выполняет один из других модулей. В любой момент времени одна из строк текста считается текущей, а адресующее эту строку смещение называется *курсором*. В редакторе текста реализованы следующие приказы:

R Имя файла. Считывает файл Имя файла в связанный список (т. е. буфер текста), находящийся в памяти. Файл должен иметь приведенную выше структуру связанного списка, а буфером текста является общая область с именем TEXT_BUFF. После выполнения данного приказа курсор адресует первую строку текста.

/ Цепочка. Просматривает текст, начиная с первой строки, до обнаружения подцепочки, которая точно соответствует Цепочке. Курсор адресует эту строку, и производится печать строки. Если Цепочка не найдена, курсор принимает исходное значение и подпрограмма MESSAGE печатает сообщение NOT FOUND (не найдена).

I Цепочка. Если длина Цепочки не больше 80, этот приказ помещает Цепочку (с добавленной комбинацией возврата каретки и перевода строки) в строку, находящуюся сразу после текущей строки. Курсор адресует новую строку, и эта строка выводится на печать. Если же длина Цепочки больше 80, приказ игнорируется.

D. Если текущая строка не является первым элементом списка, этот приказ удаляет ее, переводит курсор на следующую строку и печатает следующую строку; в противном случае приказ игнорируется.

P. Если текущая строка не является первым элементом списка, этот приказ печатает ее; в противном случае приказ игнорируется.

B. Заставляет курсор адресовать первый элемент списка.

E. Заставляет курсор адресовать последний элемент списка.

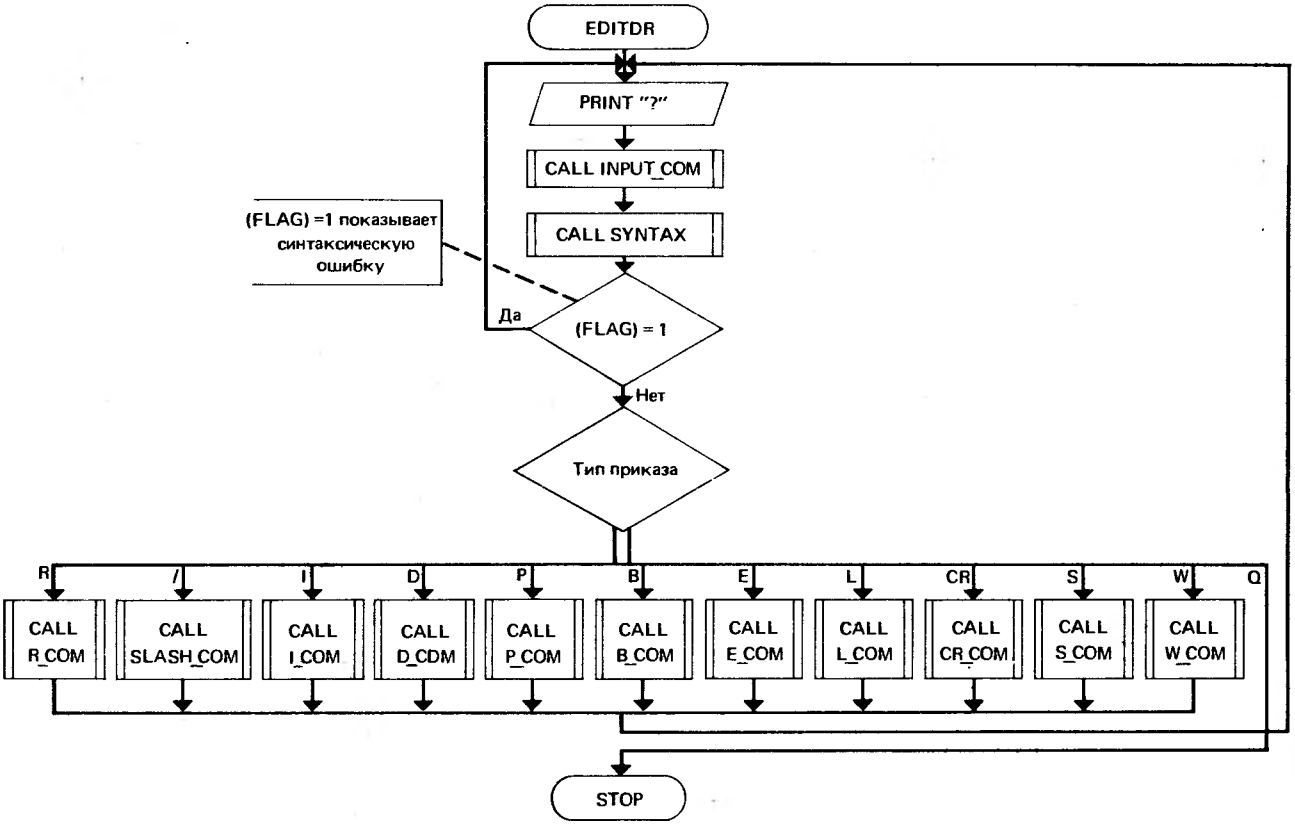


Рис. 5.11. Схема модуля EDITOR

L. Вызывает печать всех строк и перевод курсора на последнюю строку.

(Возврат каретки). Реализует перемещение курсора на следующую строку и печать этой строки.

S /Цепочка1/Цепочка 2. Если длина Цепочки 2 минус длина Цепочки 1 плюс длина текущей строки не больше 80, этот приказ отыскивает в текущей строке Цепочку 1 и, если она найдена, заменяет Цепочку 1 на Цепочку 2. Кроме того, измененная строка выводится на печать. Если получающаяся строка слишком длинна или Цепочка 1 не найдена, никаких действий не предпринимается. В любом случае курсор не модифицируется.

W Имя файла. Копирует связанный список в массив так, что строки следуют по порядку, а затем выводит упорядоченный массив в файл Имя файла.

Q. Вызывает окончание программы.

После символа приказа, содержащего имя файла или цепочку, должен следовать пробел. Синтаксис приказов контролирует модуль SYNTAX. Если синтаксис правилен и символом приказа является I, / или S, модуль SYNTAX вызывает модуль LENGTH для определения длины (длин) цепочки (цепочек), фигурирующей (их) в приказе. Модуль ORGANIZE копирует связанный список в упорядоченный массив до его записи в файл модулем W_COM. На рис. 5.11 показана схема модуля EDITOR.

Рис. 5.12 представляет собой код модуля S_COM. Предполагается, что

```
        PUBLIC  S_COM
        EXTRN   P_COM:FAR
CODE_SEG        SEGMENT
        ASSUME  CS:CODE_SEG
          LENGTH2       DB      ?
          ADLENG        DB      ?
S_COM           PROC    FAR
                PUSH    AX                      ;ЗАПОМНИТЬ РЕГИСТРЫ
                PUSH    CX
                PUSH    DX
                PUSH    SI
                PUSH    DI
                MOV     SI,[BX]                 ;ЗАГРУЗИТЬ АДРЕС ТЕКУЩЕГО
                MOV     DI,[SI]                 ;ЭЛЕМЕНТА ТЕКСТА В DI
                MOV     DL,ES:[DI+2]            ;ДЛИНА ТЕКУЩЕЙ СТРОКИ В DL
                MOV     SI,[BX+2]               ;ЗАГРУЗИТЬ ДЛИНУ STRING1
                MOV     DH,[SI]                 ;В DH
                MOV     SI,[BX+6]               ;ЗАГРУЗИТЬ ДЛИНУ STRING2
                MOV     AL,[SI]                 ;В AL
                MOV     CS:LENGTH2,AL           ;И LENGTH2
                SUB     AL,DH                   ;ПЕРЕЙТИ НА EXIT, ЕСЛИ LENGTH2
                ADD     AL,DL                   ;МИНУС LENGTH1 ПЛЮС ДЛИНА
                CMP     AL,82                   ;ТЕКУЩЕЙ СТРОКИ БОЛЬШЕ 82
                JNA     NEXT                    ;(EXIT СЛИШКОМ ДАЛЕКА ДЛЯ
                JMP     EXIT                    ;УСЛОВНОГО ПЕРЕХОДА)
        NEXT:   MOV     CS:ADLENG,AL            ;ЗАПОМНИТЬ СКОРРЕКТИРОВАННУЮ ДЛИНУ
                MOV     SI,[BX+4]               ;ЗАГРУЗИТЬ АДРЕС STRING1 В SI
                ADD     DI,3                    ;ЗАГРУЗИТЬ АДРЕС ТЕКУЩЕЙ СТРОКИ
                MOV     AX,DI                   ;В DI И AX
                XOR     CX,CX                   ;СБРОСИТЬ CX
                CLD                             ;И ФЛАЖОК НАПРАВЛЕНИЯ
                MOV     CL,DL                   ;ЗАГРУЗИТЬ В CL ДЛИНУ ТЕКУЩЕЙ
                SUB     CL,DH                   ;СТРОКИ МИНУС LENGTH1
        SEARCH: PUSH    CX                      ;ВКЛЮЧИТЬ В СТЕК СЧЕТЧИК,
                PUSH    DI                      ;УКАЗАТЕЛЬ СТРОКИ
                PUSH    SI                      ;И АДРЕС STRING1
                MOV     CL,DH                   ;ПЕРЕДАТЬ LENGTH1 В CL
                REPE    CMPSB                   ;СРАВНИВАТЬ ПОДЦЕПОЧКУ С STRING1
                POP     SI                      ;ИЗВЛЕЧЬ ИЗ СТЕКА СЧЕТЧИК,
                POP     DI                      ;УКАЗАТЕЛЬ СТРОКИ
                POP     CX                      ;И АДРЕС STRING1
                JE      FOUND                   ;ПРИ СООТВЕТСТВИИ ПЕРЕЙТИ К FOUND
                INC     DI                      ;ИНАЧЕ ИНКРЕМЕНТ УКАЗАТЕЛЯ СТРОКИ,
                LOOP    SEARCH                  ;ДЕКРЕМЕНТ CX И ПОВТОРЕНИЕ
                JMP     EXIT                    ;ПЕРЕЙТИ К EXIT, ЕСЛИ НЕ НАЙДЕНА
```

```
FOUND:      PUSH    DI              ;ЗАПОМНИТЬ DI И DS
            PUSH    DS
            MOV     SI,AX           ;ЗАГРУЗИТЬ АДРЕС ТЕКУЩЕЙ СТРОКИ В SI
            XOR     CX,CX           ;ЗАГРУЗИТЬ ЧИСЛО СДВИГАЕМЫХ
            MOV     CL,DL           ;СИМВОЛОВ В CX
            ADD     CX,AX
            SUB     CX,DI
            SUB     CL,DH
            MOV     AX,ES           ;СДЕЛАТЬ DS И ES
            MOV     DS,AX           ;ОДИНАКОВЫМИ
            CMP     DH,CS:LENGTH2   ;СРАВНИТЬ LENGTH1 И LENGTH2
            JA      ABOVE
            STD                     ;ЕСЛИ LENGTH1 НИЖЕ ИЛИ РАВНА
            XOR     AX,AX           ;LENGTH2, УСТАНОВИТЬ DF,
            MOV     DI,SI           ;А АДРЕСА ИСТОЧНИКА
            MOV     AL,DL           ;И ПОЛУЧАТЕЛЯ - НА СДВИГ
            ADD     SI,AX           ;ОСТАВШЕГОСЯ ТЕКСТА СТРОКИ
            DEC     SI              ;ВПРАВО
            MOV     AL,ADLENG
            ADD     DI,AX
            DEC     DI
            JMP     SHORT SHIFT
ABOVE:      CLD                     ;ИНАЧЕ СБРОСИТЬ DF
            MOV     SI,DI           ;И УСТАНОВИТЬ АДРЕСА
            XOR     AX,AX           ;НА СДВИГ ВЛЕВО
            MOV     AL,DH
            ADD     SI,AX
            MOV     AL,CS:LENGTH2
            ADD     DI,AX
SHIFT:      REP     MOVSB           ;СДВИНУТЬ ТЕКСТ
            POP     DS              ;ВОССТАНОВИТЬ DS И DI
            POP     DI
            MOV     CL,CS:LENGTH2   ;ВВЕСТИ STRING2
            MOV     SI,[BX+BI
            CLD
            REP     MOVSB
            MOV     SI,[BX]         ;ЗАГРУЗИТЬ АДРЕС КУРСОРА В SI
            MOV     DI,[SI]         ;И ТЕКУЩЕГО ЭЛЕМЕНТА ТЕКСТА В DI
            MOV     AL,CS:ADLENG    ;СКОРРЕКТИРОВАТЬ ДЛИНУ
            MOV     ES:[DI+2],AL    ;ТЕКУЩЕЙ СТРОКИ
            CALL    FAR P_COM       ;ПЕЧАТЬ НОВОЙ ТЕКУЩЕЙ СТРОКИ
EXIT:       POP     DI              ;ВОССТАНОВИТЬ РЕГИСТРЫ
            POP     SI
            POP     DX
            POP     CX
            POP     AX
            RET
S_COM       ENDP
CODE_SEG    ENDS
            END
```

Рис. 5.12. Код модуля S_COM

этот модуль является отдельно ассемблируемой процедурой. Ей передаются следующие пять адресов-параметров, находящихся в массиве, адрес которого передается в регистре BX.

CURSOR. Смещение текущего элемента текста в связанном списке.

LENGTH_1. Длина Цепочки 1.

STRING_1. Смещение первого байта Цепочки 1.

LENGTH_2. Длина Цепочки 2.

STRING_2. Смещение первого байта Цепочки 2.

Предполагается, что связанный список текста находится в общей области и при вызове процедуры сегментный адрес этой области содержится в регистре ES. Сегментный адрес остальных параметров находится в регистре DS; следовательно, к локальным переменным процедуры, которые находятся в CODE_SEG, необходимо обращаться с использованием CS. В упр. 7 рекомендуется написать код модулей EDITOR, SLASH_COM, I_COM, SYNTAX и LENGTH.

5.4. ТАБЛИЧНОЕ ПРЕОБРАЗОВАНИЕ

Во введении к данной главе упоминалась необходимость преобразования одного кода в другой. Например, терминал может взаимодействовать с компьютером в коде EBCDIC, а программное обеспечение рассчитано на работу в коде ASCII (или наоборот).

Преобразование кода длиной не более 8 бит (всего до 256 комбинаций) наиболее просто реализуется с помощью хранения выходных кодов в массиве длиной до 256 байт и использования исходного кода в качестве индекса в массиве требуемых значений кода. Если код EBCDIC преобразуется в код ASCII, то значение кода буквы "A" (в коде EBCDIC оно равно 11000001) необходимо прибавить к начальному адресу массива. Если поместить код ASCII буквы A, равный 01000001, в элемент массива, имеющий адрес массива плюс 00C1, необходимое преобразование реализуется очень просто.

НАЗВАНИЕ	МНЕМОНИКА И ФОРМАТ	ОПИСАНИЕ
ПРЕОБРАЗОВАТЬ	XLAT OPR XLATB	(AL) ← ((BX)+(AL))

ФЛАЖКИ. НИКАКИЕ ФЛАЖКИ НЕ МОДИФИЦИРУЮТСЯ.

РЕЖИМЫ АДРЕСАЦИИ. ОПЕРАНДЫ ОПРЕДЕЛЯЮТСЯ КОДОМ ОПЕРАЦИИ (НЕЯВНО).

Рис. 5.13. Команда преобразования XLAT

В микропроцессоре 8086 предусмотрена специальная команда для выполнения этих действий — это команда XLAT, определенная из рис. 5.13. В ассемблерных программах она может иметь две формы: XLATB и XLAT OPR, которые порождают одну и ту же машинную команду:

11010111

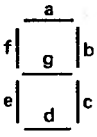
Во второй форме OPR — фиктивный операнд, который обычно является именем переменной, ассоциируемой с таблицей преобразования, и служит только для улучшения читабельности программы. В команде XLAT предполагается, что базовый адрес массива байт находится в регистре BX, а преобразуемый байт — в регистре AL. Преобразуемое значение кода берется из массива и помещается в регистр AL. Состояния флажков не изменяются.

Предположим, что цепочку неупакованных BCD-цифр с нулевыми старшими тетрадами необходимо преобразовать в двоичные комбинации, предназначенные для управления 7-сегментными индикаторами. Размещение сегментов и их обозначения приведены на рис. 5.14, *а*, а рис. 5.14, *б* показывает преобразование кода, необходимое для получения изображений цифр. В выходном коде бит 0 соответствует сегменту *a* индикатора, бит 1 — сегменту *b* и т. д. Очевидно, цепочка

05 07 09 00 00 01 03 04

соответствующая числу 57900134, вызывает формирование выходной цепочки

6D 07 6F 3F 3F 06 4F 66



а)

Десятичная цифра	Вид индикации	Неупакованный BCD-формат	g f e d c b a	16-ричный код
0	0	00000000	0 1 1 1 1 1 1	3F
1	1	00000001	0 0 0 0 1 1 0	06
2	2	00000010	1 0 1 1 0 1 1	5B
3	3	00000011	1 0 0 1 1 1 1	4F
4	4	00000100	1 1 0 0 1 1 0	66
5	5	00000101	1 1 0 1 1 0 1	6D
6	6	00000110	1 1 1 1 1 0 1	7D
7	7	00000111	0 0 0 0 1 1 1	07
8	8	00001000	1 1 1 1 1 1 1	7F
9	9	00001001	1 1 0 1 1 1 1	6F

б)

Рис. 5.14. Семисегментный индикатор (*а*) и эквиваленты семисегментного кода (*б*) при преобразовании BCD-цифр в код семисегментного индикатора

```
                .
                .
                .
;СЕМИСЕГМЕНТНЫЙ КОД ДЛЯ ДЕСЯТИЧНЫХ ЦИФР 0, 1, 2, 3, 4, 5, 6, 7, 8 И 9
    TABLE       DB      3FH
                DB      06H
                DB      5BH
                DB      4FH
                DB      66H
                DB      6DH
                DB      7DH
                DB      07H
                DB      7FH
                DB      6FH
                    .
                    .
                    .
                LEA     BX,TABLE            ;ЗАГРУЗИТЬ БАЗОВЫЙ АДРЕС ТАБЛИЦЫ
                LEA     SI,STG_BCD          ;УКАЗАТЕЛЬ ЦЕПОЧКИ-ИСТОЧНИКА
                LEA     DI,STG_DISP         ;УКАЗАТЕЛЬ ПОЛУЧАТЕЛЯ
                CLD                         ;ЗАДАТЬ АВТОИНКРЕМЕНТ
                MOV     CX,LENGTH STG_BCD
    CONVERT:    LODS    STG_BCD             ;ПЕРЕДАТЬ ЦИФРУ В AL
                CMP     AL,9                ;ОНА БОЛЬШЕ 9 ?
                JA      INVALID             ;ЕСЛИ ДА, НЕДЕЙСТВИТЕЛЬНАЯ ЦИФРА
                XLAT    TABLE               ;ПРЕОБРАЗОВАТЬ В КОД ИНДИКАТОРА
                STOS    STG_DISP            ;ЗАПОМНИТЬ РЕЗУЛЬТАТ
                LOOP    CONVERT             ;ПОВТОРЯТЬ ДО ЗАВЕРШЕНИЯ
                    .
                    .
                    .
```

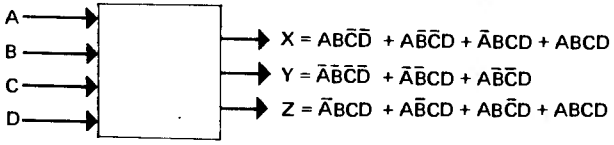
Рис. 5.15. Фрагмент преобразования неупакованных BCD-цифр в код семисегментного индикатора

На рис. 5.15 приведен фрагмент программы для преобразования цепочки неупакованных BCD-цифр, начинающейся в STG_BCD, в цепочку, выводимую на индикаторы, которая начинается в STG_DISP. Предусмотрен контроль недопустимых входных комбинаций с переходом к метке INVALID, если такая комбинация обнаруживается.

Преобразование кода можно рассмотреть и с более общей точки зрения. Оно, по существу, представляет собой функциональное отображение множества 8-битных комбинаций в само себя. Этими комбинациями не обязательно должны быть коды символов. Как в предыдущем примере, они могут быть элементами любого множества, число которых не превышает 256. Например, входные для процесса преобразования биты могут соответствовать входам логической схемы, а получающиеся биты – выходам схемы. Допускается обработка до 8 входов и 8 выходов. В качестве примера на рис. 5.16, *б* показан массив, необходимый для формирования выходов логической схемы, представленной на рис. 5.16, *а*. В таблице преобразования предполагается, что булева переменная D соответствует младшему входному биту, а булева функция Z – младшему выходному биту.

5.5. ПРЕОБРАЗОВАНИЯ ФОРМАТОВ ЧИСЕЛ

Терминалы, принтеры, считыватели перфокарт и другие устройства обычно взаимодействуют с вычислительной системой в коде ASCII. Однако внут-



а)

```
                    XYZ
LOG_TAB     DB      010B
            DB      000B
            DB      000B
            DB      010B
            DB      000B
            DB      000B
            DB      000B
            DB      101B
            DB      000B
            DB      110B
            DB      000B
            DB      001B
            DB      100B
            DB      001B
            DB      000B
            DB      101B
```

б)

Рис. 5.16. Логическая схема (*а*) и таблица преобразования кода (*б*) для реализации многовыходной логической схемы

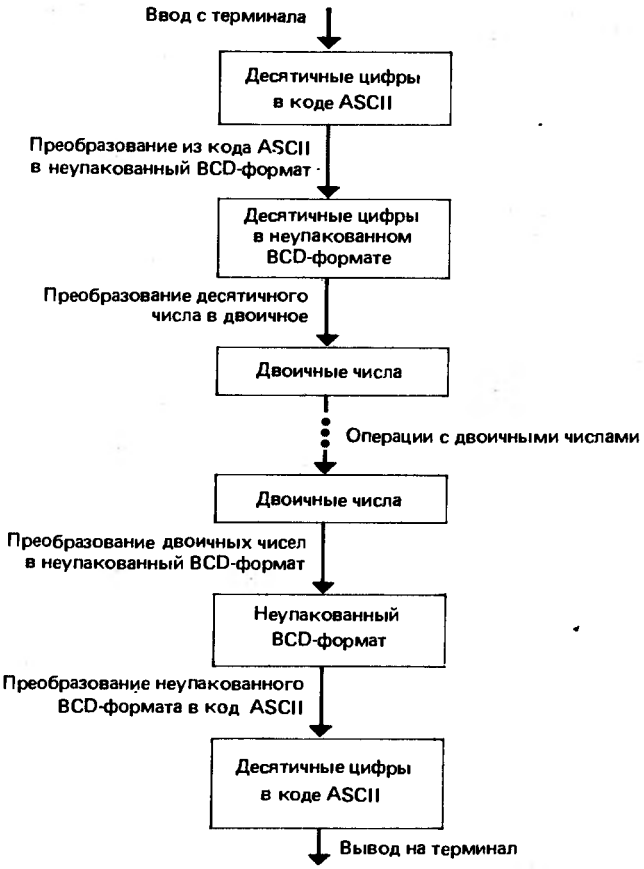


Рис. 5.17. Процесс преобразований, необходимый для ввода и вывода чисел

ренние арифметические операции чаще всего выполняются в двоичном или упакованном BCD-формате. Следовательно, появляется необходимость преобразования внешнего формата данных, в котором работает периферийное оборудование, во внутренний формат, на который рассчитаны машинные команды.

Несмотря на то, что микропроцессор 8086 способен выполнять арифметические операции с ASCII-числами, примеры в гл. 3 показывают сложность реализации всех действий, выходящих из рамки простых арифметических операций. Упакованный BCD-формат приемлем только для сложения и вычитания. Конечно, наилучшим вариантом с точки зрения быстродействия и простоты оказывается двоичная арифметика. К сожалению, для ее реализации все числа должны быть преобразованы из ASCII-формата в двоичный. Следовательно, приходится выбирать между дополнительным временем на пре-

образования и дополнительным временем на выполнение арифметических операций. В некоторых задачах приемлемым решением является упакованный BCD-формат, так как преобразование ASCII-чисел в BCD-числа оказывается относительно простым, а сложение и вычитание упакованных BCD-чисел выполняются довольно быстро. Решение о выборе формата определяется объемом ввода-вывода и сложностью вычислений.

На рис. 5.17 показан полный процесс ввода символьных цепочек в коде ASCII, преобразования их в двоичные числа, выполнения вычислений и обратного преобразования результатов в ASCII-цепочки. Преобразование в двоичный формат реализуется в два этапа. Сначала каждый ASCII-символ преобразуется в неупакованный BCD-формат (старшие тетрады нулевые), а затем цепочка неупакованных цифр — в двоичное число по правилу Горнера. Обозначим ASCII-цепочку через $d_n \ldots d_1 d_0$, где d_i – десятичные цифры. Эквивалентное двоичное число получается при вычислении полинома

$$((\ldots((10d_n + d_{n-1})10 + d_{n-2}) \ldots)10 + d_1)10 + d_0.$$

Программа преобразования ASCII-числа в двоичный эквивалент представлена на рис. 5.18. Предполагается, что пять символов (возможно, со старши-

```
        MOV     CX,5                    ;ПЕРЕДАТЬ ЧИСЛО ЦИФР В CX
        MOV     DI,10                   ;ПЕРЕДАТЬ ОСНОВАНИЕ 10 В DI
        MOV     BH,'9'                  ;КОД ASCII ЦИФРЫ 9
        MOV     BL,'0'                  ;КОД ASCII ЦИФРЫ 0
        MOV     SI,OFFSET ASCII_STG
        ADD     SI,LENGTH ASCII_STG-1
        STD                             ;ВЫЗЫВАЕТ АВТОДЕКРЕМЕНТ
                                        ;В ЦЕПОЧЕЧНЫХ ОПЕРАЦИЯХ
        XOR     AX,AX                   ;СБРОСИТЬ AX
AGAIN:  MUL     DI                      ;УМНОЖИТЬ ЧАСТИЧНУЮ СУММУ НА 10
        JC      OVERFLOW                ;ЧИСЛО БОЛЬШЕ 65535
        MOV     DX,AX                   ;ЗАПОМНИТЬ ЧАСТИЧНЫЙ РЕЗУЛЬТАТ
        LODS    ASCII_STG               ;ЗАГРУЗИТЬ СЛЕДУЮЩУЮ ЦИФРУ
        AND     AL,7FH                  ;СБРОСИТЬ БИТ ПАРИТЕТА
        CMP     AL,BL                   ;ЦИФРА МЕНЬШЕ 0 ?
        JL      INVALID                 ;НЕДЕЙСТВИТЕЛЬНАЯ ЦИФРА
        CMP     AL,BH                   ;ЦИФРА БОЛЬШЕ 9 ?
        JG      INVALID                 ;НЕДЕЙСТВИТЕЛЬНАЯ ЦИФРА
        SUB     AL,30H                  ;ПРЕОБРАЗОВАТЬ В НЕУПАКОВАННЫЙ
                                        ;BCD-ФОРМАТ
        CBW                             ;РАСШИРИТЬ ДО ПОЛНОГО СЛОВА
        ADD     AX,DX
        JC      OVERFLOW                ;ЧИСЛО БОЛЬШЕ 65535
        LOOP    AGAIN
          .
          .
          .
```

Рис. 5.18. Преобразование десятичного ASCII-числа в двоичный эквивалент

ми нулями) хранятся по адресу ASCII_STG, начиная с младшей цифры. Если встречается недопустимый символ (не цифра), осуществляется переход к INVALID. Так как для результата отведено одно слово, происходит переход к OVERFLOW, если результат больше $2^{16} - 1$. Программа обратного преобразования приведена на рис. 5.19. Когда результат содержит менее пяти цифр, в старшие разряды помещаются нули (30_{16} в коде ASCII); обе программы не воспринимают отрицательных чисел (см. упр. 15).

Иногда требуется вводить или выводить ASCII-цепочку, которая представляет собой 16-ричное число. Примером служит отладочная программа, осуществляющая модификацию машинных команд другой программы. От-

```
        MOV     CX,5
        XOR     DI,DI           ;DI СЛУЖИТ ИНДЕКСНЫМ РЕГИСТРОМ
        MOV     AX,BINNUM
        MOV     BX,10
AGAIN:  XOR     DX,DX           ;СБРОСИТЬ DX
        DIV     BX              ;РАЗДЕЛИТЬ DX:AX НА 10
        ADD     DL,30H          ;ПРЕОБРАЗОВАТЬ ОСТАТОК В ASCII
        MOV     ASCII_BCD[DI],DL ;ЗАПОМНИТЬ ДЕСЯТИЧНУЮ ЦИФРУ
        INC     DI              ;ПРОДВИНУТЬ ИНДЕКС
        LOOP    AGAIN
          .
          .
          .
```

Рис. 5.19. Преобразование двоичного числа в ASCII-число

ладочная программа должна допускать ввод ASCII-цепочки, которая представляет собой 16-ричный адрес, машинную команду или данное, и преобразовывать ее в двоичную форму, пригодную для непосредственной модификации машинного кода или содержимого памяти в отлаживаемой программе. Например, 16-ричные числа

01A25 1B3D

могут быть реакцией на запрос изменения содержимого адреса (первое число) на указанное значение (второе число). Отладочная программа должна преобразовать введенные с терминала ASCII-цепочки '01A25' и '1B3D' в эквивалентные 16-ричные числа. Может потребоваться и вывод 16-ричных адресов, команд и содержимого памяти.

Процедура ввода 16-ричных чисел включает в себя два этапа. На первом этапе 16-ричные цифры кода ASCII преобразуются в двоичные эквиваленты и помещаются в байты с нулевыми старшими тетрадами. На втором производится простая упаковка этих двоичных эквивалентов. Так как двоичный эквивалент 16-ричного числа получается при представлении каждой 16-ричной цифры ее двоичным эквивалентом, результатом процесса входного преобра-

```
        MOV     CL,4                    ;CL СЛУЖИТ СЧЕТЧИКОМ СДВИГОВ
        MOV     CH,CL                   ;CH СЛУЖИТ СЧЕТЧИКОМ РАЗРЯДОВ
        MOV     SI,OFFSET ASCII_STG     ;SI СЛУЖИТ УКАЗАТЕЛЕМ
        CLD                             ;ЗАДАТЬ АВТОИНКРЕМЕНТ
                                        ;В ЦЕПОЧЕЧНОЙ ОПЕРАЦИИ
        XOR     AX,AX                   ;СБРОСИТЬ AX
        XOR     DX,DX                   ;СБРОСИТЬ DX
AGAIN:  LODS    ASCII_STG               ;ЗАГРУЗИТЬ ASCII-ЦИФРУ
        AND     AL,7FH                  ;СБРОСИТЬ БИТ ПАРИТЕТА
        CMP     AL,'0'                  ;МЕНЬШЕ 30H ?
        JL      INVALID
        CMP     AL,'9'                  ;БОЛЬШЕ 39H ?
        JG      A_TO_F
        SUB     AL,30H                  ;ПРЕОВРАЗОВАТЬ В ЦИФРУ 0 - 9
        JMP     SHORT ROTATE
A_TO_F: CMP     AL,'A'                  ;МЕНЬШЕ 41H ?
        JL      INVALID
        CMP     AL,'F'                  ;БОЛЬШЕ 46H ?
        JG      INVALID
        SUB     AL,37H                  ;ПРЕОБРАЗОВАТЬ В ЦИФРУ A - F
ROTATE: OR      DL,AL                   ;ОБ'ЕДИНИТЬ С ПРЕДЫДУЩИМИ ЦИФРАМИ
        ROR     DX,CL                   ;В ДВОИЧНЫЙ ЭКВИВАЛЕНТ
        DEC     CH
        JNZ     AGAIN                   ;ПОВТОРИТЬ, ЕСЛИ НЕ ПОСЛЕДНЯЯ ЦИФРА
        MOV     INTER,DX                ;ЗАПОМНИТЬ ДВОИЧНЫЙ ЭКВИВАЛЕНТ
          .
          .
          .
```

Рис. 5.20. Преобразование 4-разрядного 16-ричного ASCII-числа в двоичный эквивалент

зования будет двоичный эквивалент входного 16-ричного числа. Обратное преобразование для вывода реализуется такими же этапами, но в противоположном порядке.

Программа на рис. 5.20 преобразует 4-разрядное 16-ричное число (в коде ASCII) в двоичный эквивалент. Предполагается, что 16-ричное число хранится по адресу ASCII_STG, начиная с младшего разряда. Каждая цифра сравнивается с кодами ASCII 0 . . . 9 и A . . . F для проверки допустимой 16-ричной цифры. При положительном результате проверки она преобразуется в двоичный эквивалент, а затем объединяется с другими цифрами, образуя двоичный эквивалент числа.

Упражнения

1. Оцените время выполнения программ на рис. 5.2, *а*, если длина STRING равна 10, и на рис. 5.2, *б*, если используется префикс REP. (Частота синхронизации равна 5 МГц.)

2. Перепишите программу на рис. 5.5, заменив команду CMPS командой CMP.

3. Напишите программу просмотра цепочки STRING и пересылки подцепочки цифр в NUM, не пользуясь префиксом REP. Когда встречается символ, не являющийся пробелом или цифрой, происходит переход к MESSG1. Цепочка STRING просматривается до обнаружения цифры, а затем цифры передаются в NUM до тех пор, пока не встретится пробел или общее число символов не превысит 20 – в этом случае программа переходит к EXIT.

4. Приведите программу, которая сравнивает первые 10 байт, начинающихся в CHAR_1, с первыми 10 байтами, начинающимися в CHAR_2. При равенстве программа переходит к MATCH, а при неравенстве продолжается.

5. Перепишите программу из упр. 3, пользуясь префиксом REP.

6. Постройте машинный код команд:

а)
```
AGAIN: MOVS STRING1,STRING2
       LOOP AGAIN
```

б)
```
       REP MOVS STRING1,STRING2
```

Здесь STRING1 и STRING2 – начальные адреса массивов слов.

7. Рассмотрите пример редактора текста из § 5.3 и предположите:

TEXT_BUFF. Общая область, используемая для хранения связанного списка текста. Эта область определяется как структура, каждый элемент которой соответствует строке текста.

CURSOR. Смещение текущей строки в TEXT_BUFF.

FLAG. Показывает синтаксическую ошибку. Устанавливается в 1, если модуль SYNTAX обнаружит ошибку.

FILENAME. Область для хранения *Имени файла*, находящегося в приказах R или W.

LENGTH_1. Хранит длину Цепочки в приказах / или I и длину Цепочки 1 в приказе S.

STRING_1. Смещение первого байта Цепочки в приказах / или I и Цепочки 1 в приказе S.

STRING_2. Смещение первого байта Цепочки 2 в приказе S.

LENGTH_2. Хранит длину Цепочки 2 в приказе S.

Напишите код следующих программных модулей, считая, что они ассемблируются раздельно:

а) EDITOR б) SLASH_COM в) I_COM г) SYNTAX и LENGTH ассемблируются вместе, но LENGTH находится в отдельном сегменте и вызывается как процедура.

8. Расширьте таблицу на рис. 5.14 для преобразования 16-ричных цифр в коды 7-сегментного индикатора. Цифры A, C, E и F индицируются как прописные, а цифры B и D – как строчные.

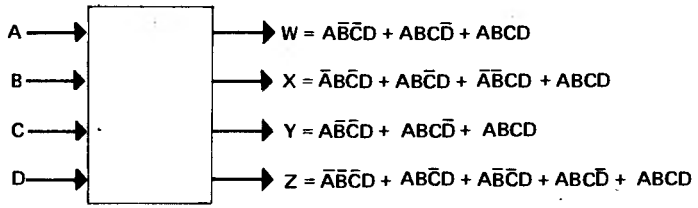


Рис. 5.21. Логическая схема для упр. 9

9. На рис. 5.21 приведена логическая схема. Определите содержимое массива IN_OUT, который образует правильные выходы для всех возможных входов. Пусть 10 последовательных входов хранятся в области от INPUT до INPUT + 9; напишите программу, которая помещает соответствующие выходы в область от OUTPUT до OUTPUT + 9.

10. Измените решение упр. 3.30, пользуясь цепочечными примитивами.

11. Измените решение упр. 4.35, пользуясь цепочечными примитивами.

12. Напишите программу, которая просматривает ASCII-цепочку из 100 символов и заменяет каждую десятичную цифру на %. Считайте, что цепочка начинается в STG.

13. Напишите программу, которая копирует все ASCII-символы в STG, заключенные в пару круглых скобок, в цепочку MESSAG и запоминает в COUNT число переданных символов.

14. Пусть таблица имен начинается в ячейке TABLE и состоит из 100 элементов; каждый элемент имеет 80 байт, причем первые 8 байт представляют собой поле имени, а остальные 72 байта являются информационным полем. Напишите программу поиска в таблице заданного имени из 8 символов, хранимого в NAME. Если имя найдено, необходимо скопировать соответствующую информацию в INFO; в противном случае INFO заполняется пустыми символами (нулевыми байтами).

15. Модифицируйте программы на рис. 5.18 и 5.19, так чтобы они были рассчитаны на старшие пробелы, а не на нули. Кроме того, модифицируйте программу на рис. 5.18 так, чтобы первый знак минус приводил к запоминанию дополнительного кода числа, содержащегося в цепочке. Помните, что теперь диапазон чисел составляет −32768 . . . . . 32767 и что необходимо соответствующим образом изменить контроль входных данных.

16. Напишите программу преобразования двоичного целого числа из BINARY в четыре 16-ричных цифры в коде ASCII и загрузки результата в ASCII_STG.

6. ПРОГРАММИРОВАНИЕ ВВОДА-ВЫВОДА

В предыдущих главах предполагалось, что программы и обрабатываемые компьютером данные уже находятся в памяти, и не рассматривалось, как они туда попадают. Во всех примерах информация передавалась только между ЦП и памятью. Теперь разберем способы передачи информации между периферийными устройствами или внешней памятью и ЦП или памятью.

На рис. 6.1 показана базовая архитектура одношинной вычислительной системы. Более сложные и разнообразные конфигурации многошинных систем рассматриваются в гл. 11. Как видно из рис. 6.1, все периферийные устрой-

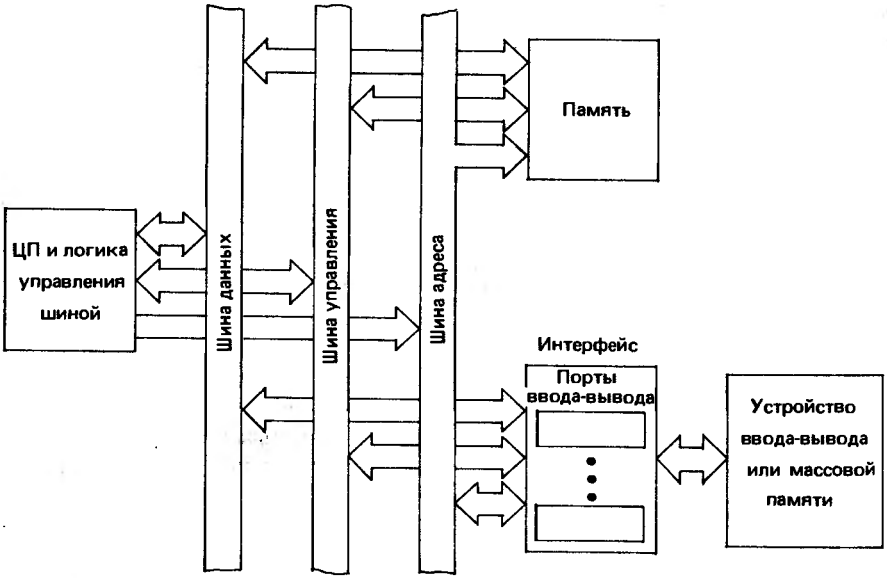


Рис. 6.1. Одношинная система

ства и внешняя память подключаются к системной шине через интерфейсы. Каждый интерфейс имеет набор регистров, называемых *портами ввода-вывода*, через которые ЦП и память взаимодействуют с внешним устройством. Одни порты предназначены для буферирования данных, другие – для хранения информации о состоянии устройства и интерфейса, которую может проверить ЦП, а третьи – для восприятия приказов от ЦП, управляющих действиями интерфейса и устройства. Все взаимодействие с внешним миром осуществляется через порты ввода-вывода в интерфейсе. Следовательно, в ЦП должны быть средства для передачи информации в (из) порты (ов), а также в (из) память (и).

В некоторых компьютерах адреса памяти и портов включены в единое адресное пространство и все команды с обращением к памяти могут обращаться и к портам. В других компьютерах, например в системах на базе микропроцессора 8086, допускается организация двух адресных пространств: памяти и ввода-вывода. Для этого в шине управления предусматриваются управляющие линии, показывающие, к какому пространству относится адрес на шине адреса. Чтобы выдать правильные сигналы на управляющие линии, система с раздельными пространствами памяти и ввода-вывода должна иметь специальные команды для взаимодействия с портами ввода-вывода.

Вывод из ЦП в управляющий или буферный порт осуществляется выдачей адреса на шину адреса и соответствующих сигналов на шину управления с последующей выдачей данных на шину данных. Ввод из входного порта реализуется выдачей адреса и управляющих сигналов на соответствующие шины и ожиданием реакции интерфейса путем выдачи содержимого адресованного

порта на шину данных. Следует подчеркнуть, что адреса ассоциируются с портами, а не с интерфейсами. Если интерфейс имеет четыре порта, он должен воспринимать четыре адреса. Однако допускается спроектировать интерфейс так, чтобы два или более регистров разделяли один и тот же порт, если схема интерфейса соответственно управляет внутренними передачами.

Данная глава начинается с обсуждения общих вопросов передач и программирования ввода-вывода. Рассматриваются основные виды ввода-вывода: программный ввод-вывод, ввод-вывод по прерываниям и блоковые передачи; § 6.1 служит кратким введением в несколько важных вопросов, которые затем обсуждаются более детально. В § 6.5 приведен законченный пример программирования ввода-вывода.

6.1. ОБЩИЕ ПРИНЦИПЫ ВВОДА-ВЫВОДА

Передачу данных в(из) порт(а) можно осуществить двумя способами. Первый способ – выполнить команду, которая передает один байт или одно слово, второй – выполнить последовательность команд, которая заставляет специальную компоненту, связанную с интерфейсом, передать совокупность байт или слов в(из) указанный(ого) блок(а) ячеек памяти. Второй способ называется *блоковой передачей* или *прямым доступом к памяти* (*ПДП*), а специальная компонента называется *контроллером ПДП*. Байт или слово передаются между портом и ЦП, а блоковые передачи осуществляются непосредственно в(из) память(и).

Программный ввод-вывод и ввод-вывод по прерываниям опираются на передачи байт и слов; данные от памяти к портам и наоборот передаются через регистр ЦП. Например, если необходимо ввести слово из порта в ячейку памяти, его приходится вначале ввести в регистр ЦП, а затем переслать в ячейку памяти. Чтобы передать блок байт или слов из периферийного устройства в память, программа должна последовательно вводить байты или слова в ЦП, а затем помещать их в смежные ячейки памяти. Эти операции реализуются программным циклом. Блоковым вводом управляют интерфейс и контроллер ПДП так, что каждые байт или слово по мере их появления передаются из порта непосредственно в память. Для инициирования передачи программа должна только выдать необходимые приказы в интерфейс и контроллер. Аналогичным образом интерфейс и его контроллер ПДП могут воспринимать последовательные байты или слова из памяти и передавать их во внешнее устройство.

Возникают два вопроса: как система узнает, когда интерфейс имеет данные или готов принимать данные, и какой интерфейс обслуживать первым, когда одновременно требуют внимания несколько интерфейсов. При программном вводе-выводе программа определяет требующие обслуживания интерфейсы, проверяя биты "готовность" в их регистрах состояния. Программная проверка разрядов или сигналов готовности называется *опросом* (*полингом*). При вводе-выводе по прерываниям интерфейс посылает в ЦП внешнее прерывание, когда он имеет данные для ввода или готов воспринимать их, а сама операция ввода-вывода реализуется *процедурой прерывания*.

В операции ПДП интерфейс запрашивает использование шины, выдавая сигнал по линии управления, и осуществляет необходимую передачу без помощи ЦП. Подробные действия во всех трех видах ввода-вывода рассматриваются в § 6.2 – 6.4.

Когда обслуживания запрашивают несколько интерфейсов, порядок удовлетворения запросов зависит от встроенного в систему механизма *приоритетов*. Известны несколько способов назначения интерфейсам приоритетов; некоторые из них реализуются программно, другие – аппаратно, а третьи являются комбинацией первых двух. Хотя механизм приоритетов может варьироваться в зависимости от вида ввода-вывода, необходим метод однозначного разрешения возникающих конфликтов. Так как приоритеты часто определяются аппаратными средствами, они рассматриваются не только в остальных разделах данной главы, но и в гл. 8.

В микропроцессоре 8086 программное взаимодействие с портами ввода-вывода осуществляется командами ввода IN и вывода OUT, определенными на рис. 6.2. Обе команды передают байт или слово и имеют длинный и корот-

НАЗВАНИЕ	МНЕМОНИКА И ФОРМАТ		ОПИСАНИЕ
ВВЕСТИ			
ДЛИННАЯ ФОРМА, БАЙТ	IN	AL, PORT	(AL) ← (PORT)
ДЛИННАЯ ФОРМА, СЛОВО	IN	AX, PORT	(AX) ← (PORT+1:PORT)
КОРОТКАЯ ФОРМА, БАЙТ	IN	AL, DX	(AL) ← ((DX))
КОРОТКАЯ ФОРМА, СЛОВО	IN	AX, DX	(AX) ← ((DX)+1:(DX))
ВЫВЕСТИ			
ДЛИННАЯ ФОРМА, БАЙТ	OUT	PORT, AL	(PORT) ← (AL)
ДЛИННАЯ ФОРМА, СЛОВО	OUT	PORT, AX	(PORT+1:PORT) ← (AX)
КОРОТКАЯ ФОРМА, БАЙТ	OUT	DX, AL	((DX)) ← (AL)
КОРОТКАЯ ФОРМА, СЛОВО	OUT	DX, AX	((DX)+1:(DX)) ← (AX)

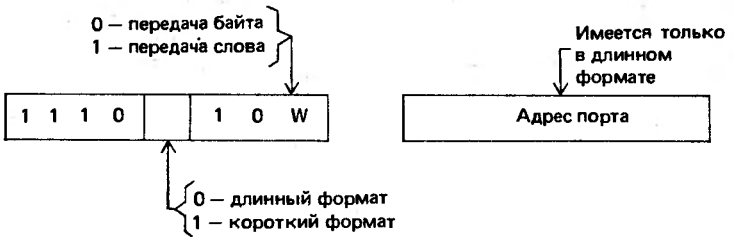
ПРИМЕЧАНИЕ. PORT ЯВЛЯЕТСЯ КОНСТАНТОЙ, НАХОДЯЩЕЙСЯ В ДИАПАЗОНЕ 0...255.

ФЛАЖКИ. НИКАКИЕ ФЛАЖКИ НЕ МОДИФИЦИРУЮТСЯ.

РЕЖИМЫ АДРЕСАЦИИ. ОПЕРАНДЫ ОПРЕДЕЛЯЮТСЯ, КАК ПОКАЗАНО ВЫШЕ.

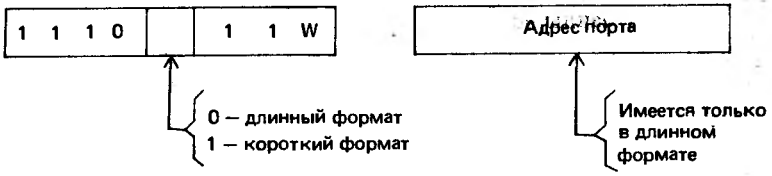
Рис. 6.2. Команды ввода и вывода

кий форматы. Первым операндом в команде IN, т. е. операндом-получателем, должен быть регистр AL (передается байт) или регистр AX (передается слово). Как и при обращении к памяти, слово передается из двух смежных адресов, причем младший байт передается в AX из порта с меньшим адресом. Если второй операнд в команде IN оказывается константой, она служит адресом порта, содержимое которого вводится в регистр. Команда в этом случае имеет длину 2 байта и адрес занимает второй байт. Когда вторым операндом указан DX, команда оказывается однобайтной, а в качестве адреса порта используется содержимое DX. В отличие от адресации памяти содержимое DX не модифицируется никаким сегментным регистром. Этот формат обеспечивает доступ к портам ввода-вывода с адресами в диапазоне 0 . . . 64К – 1. Команда IN имеет следующий машинный код:



Хотя в команде IN первым операндом неявно служит AL или AX, регистр необходимо указывать, чтобы ассемблер мог определить бит W.

Аналогичные положения относятся и к команде OUT, но в ней адрес порта является получателем и поэтому должен указываться первым операндом, а вторым операндом будут AL или AX. Машинный код команды OUT:



Отметим, что в длинном формате команд IN и OUT адрес порта должен находиться в диапазоне 0000 . . . 00FF, а в коротком формате — в диапазоне 0000 . . . FFFF (т. е. может быть любым адресом в пространстве ввода-вывода). Команды IN и OUT не воздействуют на флажки.

Команда IN применяется для ввода данных из буферного регистра данных или состояния из регистра состояния. Следующие команды

```
IN    AX,28H
MOV   DATA_WORD,AX
```

передают слово из портов с адресами 0028 и 0029 в ячейку памяти DATA_WORD. Команды

```
IN    AL,27H
TEST  AL,00000100B
JNZ   ERROR
```

реализуют ввод байта из порта 27H и переход к ERROR, если бит 2 этого байта содержит 1.

Команда OUT предназначена для вывода данных или выдачи приказа в адресуемый порт. Если бит 7 порта 26H инициирует блоковую передачу через соответствующий интерфейс, передачу запускают команды:

```
COM_REG   EQU  26H
BEGIN_BIT EQU  10000000B
```

```
MOV  AL,OTRCNTBITS
OR   AL,BEGIN_BIT
OUT  COM_REG,AL
```

Байт приказа OTRCNTBITS содержит и другие биты в приказе, выдаваемом в регистр управления. Иногда достаточно ввести всего один байт или слово и использовать или обрабатывать его, но чаще нужна последовательность байт или слов, прежде чем в данных появится содержательный смысл. Когда требуются несколько байт или слов, до обработки их необходимо передать в набор смежных ячеек памяти. Такой набор ячеек называется *буфером в памяти* или просто *буфером*. Например, символы в напечатанной строке могут быть приказом для операционной системы. Так как его невозможно интерпретировать до ввода всей строки, цепочку символов следует запоминать в буфере до появления комбинации возврат каретки/перевод строки.

Часто требуется временно запоминать входные данные до их преобразования в подходящий формат или производства других операций. В некоторых операциях удобно использовать более одного буфера. Пока в один из них осуществляется ввод, происходит обработка содержимого второго буфера; когда заполняемый буфер полон, его содержимое можно обрабатывать, а ввод производить в другой. Такой метод обработки называется *двойным буферированием*, а при наличии более двух буферов — *многократным буферированием.* Хотя мы рассмотрели буферирование при вводе, аналогичные положения относятся и к выводу. Если ввод, обработка и вывод должны осуществляться одновременно, потребуется тройное буферирование.

Важной составляющей процесса ввода-вывода является преобразование численных данных из входного или выходного форматов в формат, используемый компьютером. Простые преобразования обсуждались в гл. 5, но иногда преобразования оказываются намного сложнее. Например, может потребоваться преобразовать цепочку символов в формат с плавающей точкой или разделить строку символов на части и производить преобразование каждой из них. Обычно оператор ввода-вывода языка высокого уровня задает шаблон для упомянутого процесса разделения. Операторы языка Фортран

```
WRITE     (6,20) X,N
20 FORMAT (T2,F10.2,I5)
```

вызывают преобразование переменной X (в формате с плавающей точкой) и переменной N (в целочисленном формате) в правильные символьные цепочки и вывод их печатной строкой. Шаблон символов определяется оператором FORMAT. Первые 10 символов формируются из значения X, а следующие 5 — из значения N.

Важным моментом при вводе-выводе оказывается передача данных между процедурами. Обычно процедуры, реализующие ввод-вывод, отделены от процедур, обрабатывающих данные. Появляются две возможные ситуации. Простой микрокомпьютер в системе с низкой производительностью может выполнять всего одну программу. В этом случае программист может разместить компоненты программы, включая и области данных, в фиксированных областях памяти и самостоятельно осуществлять все управление связывани-

ем. Следовательно, программист полагает, что каждая процедура точно знает, где в памяти находятся все остальные компоненты программы. Это означает, что процедура ввода помещает входные данные в область, непосредственно доступную другим процедурам программы, а процедура вывода обращается за выводимыми данными в фиксированную область. Области программ и данных в памяти динамически не перемещаются. Именно такая статическая ситуация предполагается в настоящей главе.

Другой является ситуация, когда системой управляет сложная операционная система, работающая в мультипрограммной среде. При этом ввод-вывод осуществляется набором процедур ввода-вывода, называемых *драйверами ввода-вывода*, и процедурами прерываний. Драйверы и процедуры прерываний образуют подсистему ввода-вывода операционной системы. В отличие от приведенной простой ситуации в мультипрограммной среде требуются непрерывные перемещения областей программ и данных. Так как память обычно не может хранить сразу все программы, пользовательские программы "путешествуют" между основной и внешней памятью; по мере необходимости вызываются и выполняются драйверы ввода-вывода. Как и следует ожидать, взаимодействие между операционной системой, процедурами прерываний, драйверами ввода-вывода и пользовательскими программами оказывается очень сложным. В типичной системе необходимые пользовательской программе ввод и вывод в действительности реализуются операционной системой. У этого механизма имеются два очевидных достоинства. Во-первых, пользователь освобождается от необходимости знать детальные операции ввода-вывода. Так как драйверы ввода-вывода зависят от устройств, пользователь может не знать эти устройства столь подробно, чтобы самостоятельно разработать для них программы ввода-вывода. Включив эти программы в состав операционной системы, пользователи могут разделять их, что сокращает объем дублирования. Во-вторых, лучше используются системные ресурсы. Если ввод-вывод осуществляется через операционную систему, пользовательская программа информирует монитор о всех запросах ввода-вывода. Монитор может и инициировать ввод-вывод и освободить ЦП для другого задания, пока текущее задание ожидает завершения ввода-вывода.

Возможная последовательность событий, когда вводом-выводом управляет операционная система, представлена на рис. 6.3. Когда пользовательской

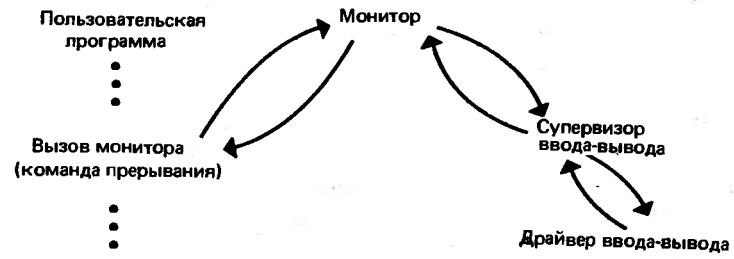


Рис. 6.3. Управление вводом-выводом в операционной системе

программе необходим ввод-вывод, она вызывает монитор с помощью программного прерывания. Оно передает управление монитору вместе с кодом функции, определяющим операцию ввода-вывода, и параметрами, необходимыми для выполнения требуемой операции. Анализируя код функции, монитор (который кроме ввода-вывода удовлетворяет и другие запросы на обслуживание) распознает запрос операции ввода-вывода и диспетчирует задачу супервизору ввода-вывода. Супервизор, далее, определяет участвующее в операции устройство ввода-вывода и инициирует соответствующий драйвер ввода-вывода, которым обычно является процедура прерывания или ПДП. Для временного хранения вводимых или выводимых данных драйвер пользуется буфером в системной области. Супервизор ввода-вывода пересылает выводимые или вводимые данные из пользовательской области в буферную или наоборот. По завершении операции ввода-вывода управление через супервизор ввода-вывода и монитор возвращается пользовательской программе.

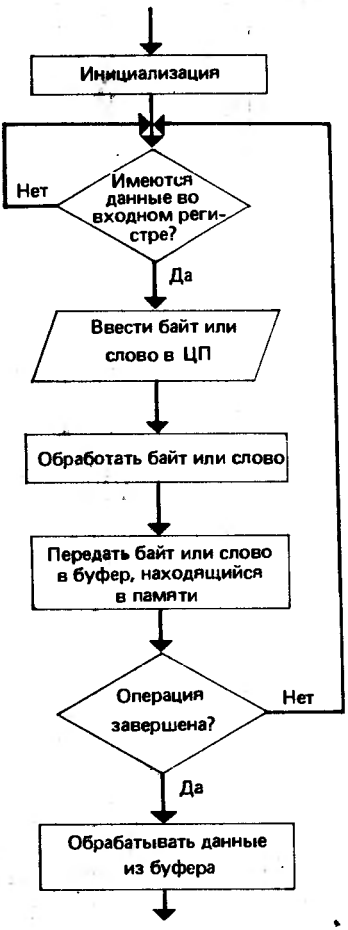


Рис. 6.4. Программный ввод-вывод

6.2. ПРОГРАММНЫЙ ВВОД-ВЫВОД

Программный ввод-вывод заключается в непрерывной проверке интерфейса и выполнении операции ввода-вывода с интерфейсом, когда его состояние показывает наличие данных для ввода или когда его буферный выходной регистр готов принимать данные от ЦП. Типичные операции программного ввода показаны на рис. 6.4. Предполагается, что вводится последовательность байт или слов; после ввода в ЦП каждые байт или слово модифицируются и передаются в буфер. После ввода всех данных и размещения в буфере начинается их обработка.

Как более сложный пример, рассмотрим ввод строки символов с терминала в 82-байтный массив, начинающийся с BUFFER. Ввод осуществляется до восприятия возврата каретки или получения более 80 символов. Если в первых 81 символе нет возврата каретки, на терминал выводится сообщение "BUFFER OVERFLOW" (переполнение буфера); в противном случае к возврату каретки автоматически добавляется перевод строки. Так как код ASCII является 7-битным, бит 7 при передаче с терминала часто служит битом паритета. Пусть бит 7 устанавливается в

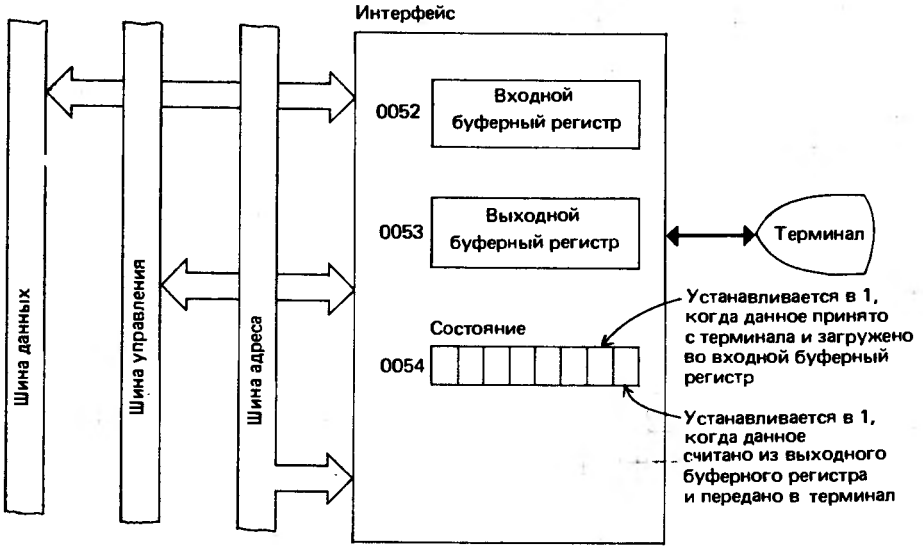


Рис. 6.5. Интерфейс для примера программного ввода-вывода

соответствии с четным паритетом, а при обнаружении байта с нечетным паритетом происходит переход к ERROR (ошибка). Если ошибки паритета нет, до передачи байта в буфер бит 7 сбрасывается. В интерфейсе терминала (рис. 6.5) адрес входного буферного регистра равен 0052, выходного буферного регистра – 0053 и регистра состояния – 0054. Единица в бите 1 регистра состояния показывает, что входной регистр содержит байт для ввода, а единица в бите 0 показывает, что выходной регистр пуст. Интерфейс спроектирован таким образом, что бит 1 устанавливается, когда байт вводится от устройства, и сбрасывается, когда байт вводится из интерфейса. Бит 0 сбрасывается, когда байт выводится в интерфейс, и устанавливается, когда байт выводится в устройство.

Программа для выполнения сформулированной задачи представлена на рис. 6.6. Регистры ES и DI являются сегментным и индексным для запоминания символов по мере их ввода, а CX используется как счетчик, ограничивающий зацикливание. Первые четыре исполняемые команды инициализируют регистры DS и ES, а следующие четыре загружают смещение BUFFER в регистр DI и COUNT, устанавливают CX на максимальную длину строки и сбрасывают флажок DF. Затем реализуется двухуровневое вложение циклов для ввода строки символов. Первые три команды образуют внутренний цикл, заставляющий компьютер ожидать до тех пор, пока в бите состояния входных данных не появится 1. Затем символ вводится в регистр AL, где контролируется его паритет. При четном паритете бит 7 сбрасывается. После сброса бита паритета введенный символ запоминается в буфере, а затем сравнивается с кодом возврата каретки. Команда LOOPNE вызывает ввод следу-

```
DATA_SEG    SEGMENT
    MESSAGE     DB      'BUFFER OVERFLOW',0DH,0AH
                .
                .
                .
DATA_SEG    ENDS
COM_SEG     SEGMENT     COMMON
    BUFFER      DB      82 DUP(?)           ;ЗАРЕЗЕРВИРОВАТЬ БУФЕРНУЮ
    COUNT       DW      ?                   ;ОБЛАСТЬ И СЧЕТЧИК
COM_SEG     ENDS
                .
                .
                .
     IN_BUF     EQU     52H                 ;НАЗНАЧИТЬ ИМЕНА АДРЕСАМ
     OUT_BUF    EQU     53H                 ;РЕГИСТРОВ ИНТЕРФЕЙСА
     STATUS     EQU     54H                 ;И БИТАМ ГОТОВНОСТИ
     RRDY       EQU     00000010B           ;В РЕГИСТРЕ СОСТОЯНИЯ
     TRDY       EQU     00000001B
                .
                .
                .
  ASSUME  DS:DATA_SEG, ES:COM_SEG
                MOV     AX,DATA_SEG         ;ИНИЦИАЛИЗИРОВАТЬ
                MOV     DS,AX               ;РЕГИСТРЫ DS И ES
                MOV     AX,COM_SEG
                MOV     ES,AX
                .
                .
                .
                MOV     DI,OFFSET BUFFER    ;ИНИЦИАЛИЗАЦИЯ, НЕОБХОДИМАЯ
                MOV     COUNT,DI            ;ДЛЯ ВВОДА
                MOV     CX,81
                CLD                         ;ЗАДАТЬ АВТОИНКРЕМЕНТ
    NEXT_IN:    IN      AL,STATUS           ;ОЖИДАТЬ ПОЯВЛЕНИЕ СИМВОЛА
                TEST    AL,RRDY             ;ВО ВХОДНОМ БУФЕРНОМ
                JZ      NEXT_IN             ;РЕГИСТРЕ
                IN      AL,IN_BUFF          ;ВВЕСТИ СИМВОЛ
                OR      AL,0                ;ПРОВЕРИТЬ ПАРИТЕТ И ПЕРЕЙТИ
                JPE     NO_ERROR            ;К ERROR ПРИ НЕЧЕТНОМ
                JMP     NEAR PTR ERROR      ;ПАРИТЕТЕ
    NO_ERROR:   AND     AL,7FH              ;СБРОСИТЬ БИТ ПАРИТЕТА
                STOSB                       ;ПЕРЕДАТЬ В БУФЕР
                CMP     AL,0DH              ;ПРОВЕРИТЬ ВОЗВРАТ
                LOOPNE  NEXT_IN             ;КАРЕТКИ И ПОВТОРИТЬ
                JNE     OVERFLOW            ;ПЕРЕПОЛНЕНИЕ
                MOV     AL,0AH              ;ДОБАВИТЬ ПЕРЕВОД СТРОКИ
                STOSB                       ;И ЗАПОМНИТЬ
                SUB     DI,COUNT            ;ЗАПОМНИТЬ ЧИСЛО
                MOV     COUNT,DI            ;СИМВОЛОВ
                .
                .
                .
    OVERFLOW:   MOV     SI,OFFSET MESSAGE   ;ИНИЦИАЛИЗАЦИЯ, НЕОБХОДИМАЯ
                MOV     CX,17               ;ДЛЯ ВЫВОДА
    NEXT_OUT:   IN      AL,STATUS           ;ОЖИДАТЬ ДОСТУПНОСТИ
                TEST    AL,TRDY             ;ВЫХОДНОГО БУФЕРНОГО
                JZ      NEXT_OUT            ;РЕГИСТРА
                LODSB                       ;ВЫВЕСТИ СИМВОЛ
                OUT     OUT_BUFF,AL
                LOOP    NEXT_OUT            ;ПОВТОРЯТЬ ДО КОНЦА СООБЩЕНИЯ
                .
                .
                .
```

Рис. 6.6. Пример программного ввода-вывода

ющего символа, если не обнаружен возврат каретки и счетчик в CX не содержит нуля. Если же последний символ является возвратом каретки, добавляется перевод строки и общее число символов помещается в COUNT. Когда счетчик достигает 0, а возврата каретки не было, происходит переход к командам вывода на терминал сообщения о переполнении. Здесь также имеется двухуровневое вложение циклов, причем внутренний цикл представляет собой ожидание единицы в бите состояния выходных данных.

```
            MOV     FLAG,0              ;СБРОСИТЬ ФЛАЖОК
INPUT:      IN      AL,STAT1            ;ПРОВЕРИТЬ STAT1 И ПРИ НЕГОТОВНОСТИ
            TEST    AL,20H              ;ПЕРЕЙТИ К DEV2, ИНАЧЕ ВВОДИТЬ
            JZ      DEV2                ;ИЗ УСТРОЙСТВА 1
            CALL    FAR PTR PROC1
            CMP     FLAG,1              ;ЕСЛИ ФЛАЖОК СБРОШЕН,
            JNZ     INPUT               ;ВВЕСТИ СЛЕДУЮЩЕЕ ДАННОЕ
DEV2:       IN      AL,STAT2            ;ПРОВЕРИТЬ STAT2 И ПРИ НЕГОТОВНОСТИ
            TEST    AL,20H              ;ПЕРЕЙТИ К DEV3, ИНАЧЕ ВВОДИТЬ
            JZ      DEV3                ;ИЗ УСТРОЙСТВА 2
            CALL    FAR PTR PROC2
            CMP     FLAG,1              ;ЕСЛИ ФЛАЖОК СБРОШЕН,
            JNZ     INPUT               ;ВВЕСТИ СЛЕДУЮЩЕЕ ДАННОЕ
DEV3:       IN      AL,STAT3            ;ПРОВЕРИТЬ STAT3 И ПРИ ГОТОВНОСТИ
            TEST    AL,20H              ;ВВОДИТЬ ИЗ УСТРОЙСТВА 3,
            JZ      NO_INPUT            ;ИНАЧЕ ПРОВЕРИТЬ ФЛАЖОК
            CALL    FAR PTR PROC3
NO_INPUT:   CMP     FLAG,1              ;ЕСЛИ ФЛАЖОК СБРОШЕН, ВВОДИТЬ
            JNZ     INPUT               ;СЛЕДУЮЩЕЕ ДАННОЕ
             .
             .
             .
```

Рис. 6.7. Приоритетный опрос

Если введенная строка содержит числа, которые до их использования необходимо преобразовать в формат целых чисел или другой формат, то при обработке строки следует предусмотреть требуемые преобразования. Они осуществляются в соответствии с положениями, изложенными в § 5.5, и реализуются вызовом процедуры преобразования, которая может быть системной процедурой. Когда строку необходимо разделять на поля, а содержащиеся в них числа требуется преобразовать в различные форматы, процесс преобразования оказывается весьма сложным.

При использовании программного ввода-вывода для нескольких устройств придется опрашивать биты готовности всех устройств. На рис. 6.7 показана программа ввода данных от трех устройств. Адреса их регистров состояния обозначены STAT1, STAT2 и STAT3, а для выполнения собственно ввода вызываются процедуры PROC1, PROC2 и PROC3. Во всех трех регистрах состояния битами готовности служат биты 5. Переменная FLAG предназначена для окончания процесса ввода и первоначально сбрасывается в 0. Предполагается, что первая процедура ввода проверяет условие окончания и устанавливает FLAG в состояние 1, что прекращает процесс ввода после обслуживания ожидающих входов. Программа задает приоритеты устройств. Устройство,

```
            MOV     FLAG,0              ;СБРОСИТЬ ФЛАЖОК
INPUT:      IN      AL,STAT1            ;ВВЕСТИ ИЗ УСТРОЙСТВА 1,
            TEST    AL,20H              ;ЕСЛИ ОНО ГОТОВО К ВВОДУ
            JZ      DEV2
            CALL    FAR PTR PROC1
DEV2:       IN      AL,STAT2            ;ВВЕСТИ ИЗ УСТРОЙСТВА 2,
            TEST    AL,20H              ;ЕСЛИ ОНО ГОТОВО К ВВОДУ
            JZ      DEV3
            CALL    FAR PTR PROC2
DEV3:       IN      AL,STAT3            ;ВВЕСТИ ИЗ УСТРОЙСТВА 3,
            TEST    AL,20H              ;ЕСЛИ ОНО ГОТОВО К ВВОДУ
            JZ      NO_INPUT
            CALL    FAR PTR PROC3
NO_INPUT:   CMP     FLAG,1              ;ПОВТОРИТЬ ЦИКЛ, ЕСЛИ ФЛАЖОК
            JNZ     INPUT               ;ВСЕ ЕЩЕ СБРОШЕН
             .
             .
             .
```

Рис. 6.8. Кольцевой опрос

соответствующее STAT1, имеет наибольший приоритет и остальные два устройства должны ожидать, пока оно не будет обслужено. Устройство, соответствующее STAT3, не обслуживается до тех пор, пока два других находятся в состоянии готовности.

На рис. 6.8 показано, как устройства обслуживаются по очереди с помощью так называемого *кольцевого механизма*, который придает всем трем устройствам одинаковые приоритеты. В данном случае переменная FLAG проверяется только в конце цикла; если она находится в состоянии 1, осуществляется выход из цикла без проверки дополнительных входов.

6.3. ВВОД-ВЫВОД ПО ПРЕРЫВАНИЯМ

Хотя программный ввод-вывод оказывается весьма простым, он связан со значительными потерями времени на ожидание активного состояния готовности. Если человек печатает на терминале со скоростью 10 символов в секунду, а для ввода каждого символа компьютеру требуется всего 10 мкс, то примерно

$$\frac{99000}{100000} \times 100\,\% = 99{,}99\,\%$$

времени расходуется впустую. Это допустимо, если в процессе ввода не требуется выполнять другую обработку, но если эту обработку задерживать нельзя, приходится искать другой подход.

В § 4.4 *прерывание* было определено как событие, заставляющее ЦП инициировать фиксированную последовательность событий, называемую *последовательностью прерывания.* До того как в микропроцессоре 8086 может начаться последовательность прерывания, он должен завершить текущую команду, если ею не являются команды HLT или WAIT. (Команда WAIT в основном применяется для ожидания завершения команды сопроцессора и рассматривается в гл. 11.)

Так как префикс считается частью команды, запрос прерывания не распознается между префиксом и командой. Для команды с префиксом REP запрос прерывания распознается после завершения примитивной операции, находящейся после REP, а адресом возврата служит ячейка с префиксом REP. Если в командах MOV и POP указан сегментный регистр, запрос прерывания не распознается до завершения следующей за ними команды. Так как сегментный регистр загружается первым, данное обстоятельство позволяет модифицировать содержимое сегментного регистра и указателя без прерываний.

После распознавания прерывания микропроцессор 8086 реализует такую последовательность прерывания:

1. Определить тип N.
2. Включить в стек текущее содержимое PSW, CS и IP (именно в таком порядке).
3. Сбросить флажки IF и TF.
4. Передать содержимое ячейки памяти 4*N в IP, а содержимое 4*N + 2 в CS.

Таким образом, прерывание заставляет микропроцессор приостановить нормальное выполнение программы и перейти к ячейке, адресуемой двойным словом, находящимся по адресу, равному учетверенному типу прерывания (т. е. указателю прерывания). Управление можно возвратить в точку возникновения прерывания командой IRET, находящейся в конце процедуры прерывания.

Ранее говорилось, что имеются два класса прерываний: внутренние и внешние. Внешние прерывания вызывает сигнал, подаваемый на входы INT и NMI микропроцессора. Прерывание, инициируемое сигналом на входе NMI, называется немаскируемым: оно вызывает прерывание типа 2 независимо от состояния флажка IF. Сигналы немаскируемых прерываний обычно формируют схемы, фиксирующие катастрофические события, например отказ питания, внешние события или импульсы таймера, которые должны обрабатываться немедленно. Прерывание на входе INT маскируется флажком IF: если IF = = 0, прерывание не распознается. Когда IF = 1 и возникает маскируемое прерывание, ЦП через выход $\overline{\text{INTA}}$ возвращает в интерфейс устройства сигнал подтверждения и инициирует последовательность прерывания. Сигнал подтверждения заставляет интерфейс выдать в ЦП (по шине адреса) байт, который определяет тип прерывания и, следовательно, адрес указателя прерывания. Указатель, в свою очередь, дает начальный адрес процедуры прерывания. Как было показано на рис. 4.26, тип маскируемого внешнего прерывания должен находиться в диапазоне 5 . . . 255. Полный процесс обслуживания прерывания представлен на рис. 6.9, где номерами обозначены следующие действия:

1 – интерфейс посылает сигнал прерывания;
2 – после завершения текущей команды возвращается подтверждение;
3 – в ЦП передается тип N;
4 – текущее содержимое PSW, CS и IP включается в стек;
5 – флажки IF и TF сбрасываются;
6 – в IP загружается содержимое 4*N, а в CS – содержимое 4*N + 2;
7 – начинается процедура прерывания;
8 – прерывания разрешены;
9 – команда IRET осуществляет извлечение из стека IP, CS и PSW;
10 – производится возврат в прерванную программу.

На немаскируемое внешнее прерывание (тип 2) сигнал $\overline{\text{INTA}}$ не выдается. В приводимых далее примерах предполагаются маскируемые прерывания, если не сделана специальная оговорка.

В предыдущем примере ввода с терминала строки символов при использовании ввода-вывода по прерыванию система не должна находиться в состоянии ожидания между ударами по клавишам. Если интерфейс выдает в ЦП сигнал прерывания при наличии данных в буферном регистре данных, то, ожидая прерывания, ЦП может выполнять другие задачи. Когда интерфейс имеет символ для ввода, ЦП извещается об этом сигналом прерывания, процедура прерывания вводит символ, а затем ЦП возвращается к прерванной обработке.

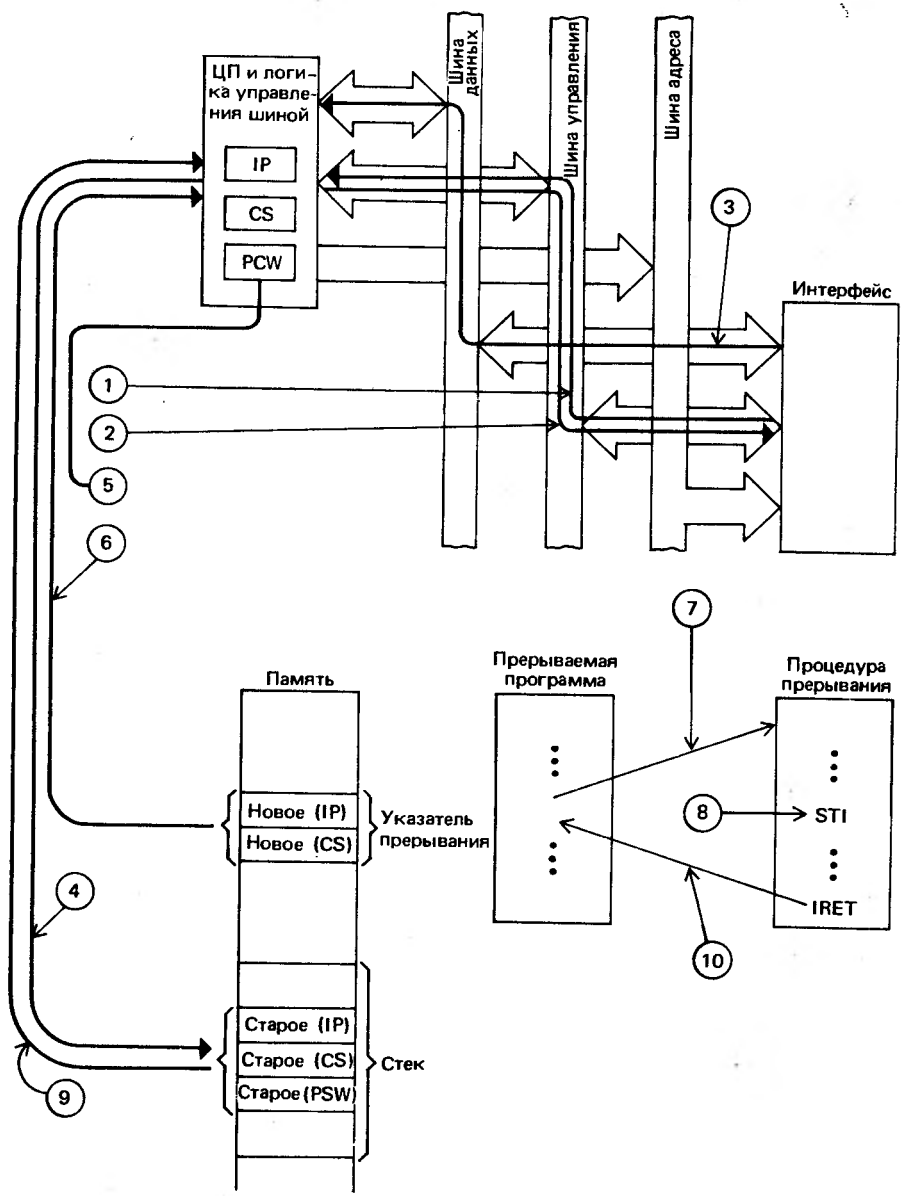


Рис. 6.9. Последовательность событий во время маскируемого прерывания и последующего возврата

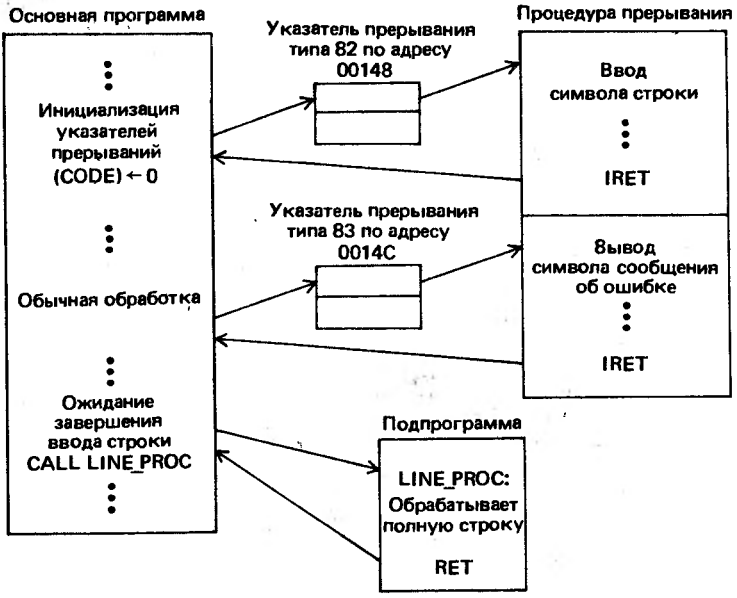


Рис. 6.10. Взаимодействия компонентов обработки при вводе строки по прерываниям

Для примера рассмотрим компоненты обработки и их взаимосвязи, приведенные на рис. 6.10. Основная программа должна инициализировать необходимые указатели прерываний, а затем начинает свою обычную обработку. При ее выполнении для ввода строки символов в буфер, адресуемый BUFF_POINT, применяется ввод-вывод по прерываниям. Предполагается, что все переменные определены в сегменте DATA_SEG, сегментный адрес которого находится в DS. Ячейка CODE, первоначально сброшенная в 0, показывает, когда введена целая строка (CODE = 2) и когда произошло переполнение буфера (CODE = 1). Переполнение возникает, когда принят 81 символ и не обнаружен возврат каретки. При переполнении прерывания по вводу запрещаются, разрешаются прерывания по выводу и для вывода сообщения об ошибке из MESSAGE применяется ввод-вывод по прерываниям. После ввода строки для ее обработки вызывается подпрограмма LINE_PROC. Обработка может заключаться в пересылке данных в больший буфер текста, преобразовании данных в другой формат и т. п.

Процедура прерывания для выполнения ввода и (в случае переполнения) вывода представлена на рис. 6.11. Первой дана процедура ввода, а затем процедура вывода. В обеих процедурах используемые ими регистры включаются в стек, а затем извлекаются из стека непосредственно перед возвратом. Как показывают директивы EQU, адреса буферных регистров данных для ввода и вывода равны 0052 и 0053, адрес регистра управления равен 0054, а биты 1 и 0 регистра управления разрешают и запрещают прерывания при вводе и

```
INT_SEG     SEGMENT
  ASSUME  CS:INT_SEG, DS:DATA_SEG          ;ДОСТУП К ПАРАМЕТРАМ ЧЕРЕЗ DS
    IN_BUF      EQU   32H
    OUT_BUF     EQU   33H
    CONTROL     EQU   34H
    ENABLE_OUT  EQU   00000001B
    INT_ROUT:   PUSH  AX                   ;ЗАПОМНИТЬ РЕГИСТРЫ
                PUSH  BX
                IN    AL,IN_BUFF           ;ВВЕСТИ СИМВОЛ
                MOV   BX,BUFF_POINT        ;И ЗАПОМНИТЬ ЕГО
                MOV   [BX],AL              ;В БУФЕРЕ
                INC   BX                   ;ИНКРЕМЕНТ УКАЗАТЕЛЯ БУФЕРА
                INC   COUNT                ;И СЧЕТЧИКА
                MOV   BUFF_POINT,BX        ;ЗАПОМНИТЬ УКАЗАТЕЛЬ БУФЕРА
                CMP   AL,0DH               ;ПРОВЕРИТЬ НА ВОЗВРАТ
                JNZ   NO_CR                ;КАРЕТКИ И ДОБАВИТЬ
                MOV   BYTE PTR [BX+1],0AH  ;ПЕРЕВОД СТРОКИ
                INC   COUNT
                MOV   CODE,2               ;УСТАНОВИТЬ CODE НА 2, ЧТОБЫ
                XOR   AL,AL                ;МОЖНО БЫЛО ВЫЗВАТЬ LINE_PROC,
                OUT   CONTROL,AL           ;ЗАПРЕТИТЬ ВВОД
                JMP   CONT
    NO_CR:      CMP   COUNT,81             ;ПРОВЕРИТЬ НА ПЕРЕПОЛНЕНИЕ;
                JB    CONT                 ;ЕСЛИ НЕТ, ВОЗВРАТ
                MOV   CODE,1               ;ИНАЧЕ УСТАНОВИТЬ CODE НА 1,
                MOV   MSGCOUNT,0           ;СБРОСИТЬ MSGCOUNT,
                MOV   AL,ENABLE_OUT        ;ЗАПРЕТИТЬ ВВОД
                OUT   CONTROL,AL           ;И РАЗРЕШИТЬ ВЫВОД
    CONT:       POP   BX                   ;ВОССТАНОВИТЬ РЕГИСТРЫ
                POP   AX
                IRET

    ;СЛЕДУЮЩАЯ ПРОЦЕДУРА ОБСЛУЖИВАНИЯ ПРЕРЫВАНИЯ ВЫВОДИТ ОДИН СИМВОЛ
    ;ИЗ MESSAGE, КОГДА ВОЗНИКАЕТ ПРЕРЫВАНИЕ ОТ УСТРОЙСТВА ВЫВОДА

    OVERFLOW:   PUSH  AX                   ;ЗАПОМНИТЬ РЕГИСТРЫ
                PUSH  BX
                MOV   BX,MSGCOUNT
                MOV   AL,MESSAGE[BX]       ;ВЫВЕСТИ СИМВОЛ
                OUT   OUT_BUF,AL
                INC   MSGCOUNT             ;ИНКРЕМЕНТ СЧЕТЧИКА
                CMP   AL,0AH               ;ПОСЛЕДНИЙ СИМВОЛ В MESSAGE ?
                JNE   RETURN               ;НЕТ, ВОЗВРАТ
                XOR   AL,AL                ;ИНАЧЕ ЗАПРЕТИТЬ ДАЛЬНЕЙШИЕ
                OUT   CONTROL,AL           ;ПРЕРЫВАНИЯ ОТ ВЫВОДА
    RETURN:     POP   BX                   ;ВОССТАНОВИТЬ РЕГИСТРЫ
                POP   AX
                IRET
INT_SEG     ENDS
```

Рис. 6.11. Процедура прерывания для ввода строки символов

выводе. Переменные-слова COUNT и MSGCOUNT первоначально сбрасываются в 0. Переменная COUNT подсчитывает символы, введенные в буфер строки. Отметим, что при возникновении переполнения команды

```
MOV  AL,ENABLE_OUT
OUT  CONTROL,AL
```

запрещают интерфейсу генерировать прерывания по вводу и разрешают формировать прерывания по выводу. Этим обеспечивается вывод на пользовательский терминал сообщения о переполнении.

На рис. 6.12 приведена программа инициализации указателей прерываний; в ней предполагается, что прерывание по вводу имеет тип 82, а по выводу — тип 83. Последние две команды разрешают прерывания по вводу. Код для загрузки указателей прерываний может быть в пользовательской программе, ню, если прерывания обрабатываются операционной системой, он является частью инициализации, выполняемой операционной системой. Если указатели

```
PUSH    DS                      ;ЗАПОМНИТЬ DS
XOR     AX,AX
MOV     DS,AX                   ;СБРОСИТЬ DS ДЛЯ АДРЕСАЦИИ
MOV     AX,OFFSET INT_ROUT      ;АБСОЛЮТНОЙ ЯЧЕЙКИ
MOV     BX,148H                 ;ПЕРЕДАТЬ СМЕЩЕНИЕ INT_ROUT
MOV     [BX],AX                 ;В ЯЧЕЙКУ 148H
MOV     AX,OFFSET OVERFLOW      ;ПЕРЕДАТЬ СМЕЩЕНИЕ OVERFLOW
MOV     [BX+4],AX               ;В ЯЧЕЙКУ 14CH
MOV     AX,INT_SEG
MOV     [BX+2],AX               ;ПЕРЕДАТЬ БАЗУ СЕГМЕНТА В 14AH
MOV     [BX+6],AX               ;ПЕРЕДАТЬ БАЗУ СЕГМЕНТА В 14EH
POP     DS                      ;ВОССТАНОВИТЬ DS
MOV     AL,00000010B
OUT     CONTROL,AL              ;РАЗРЕШИТЬ ВВОД
  .
  .
  .
```

Рис. 6.12. Фрагмент инициализации указателей прерываний

прерываний устанавливаются пользовательской программой, их можно задать при загрузке программы, поместив в ее начале следующие директивы:

```
INT_VEC     SEGMENT AT 0
            ORG     148H
            DD      INT_ROUT
            DD      OVERFLOW
INT_VEC     ENDS
```

В приведенном примере в памяти имеется только один буфер. Когда в буфер введена строка текста, для пересылки данных в другую область или обработки их на месте вызывается LINE_PROC. Предполагается, что ввод в буфер приостанавливается до завершения действий LINE_PROC, а после ее окончания буфер может воспринимать новую строку. При этом BUFF_POINT необходимо установить на начальный адрес буфера (лучше всего это сделать в LINE_PROC).

В программе, которая непрерывно получает новые данные и в которой на время обработки буфера ввод приостанавливать нельзя, потребуется минимум два буфера. Схема на рис. 6.13 показывает структурирование LINE_PROC при двойном буферировании. Здесь BUFFER1 и BUFFER2 представляют собой начальные адреса двух буферных областей. Предполагается, что обычная обработка первоначально сбрасывает FLAG в 0 и загружает BUFFER1 в BUFF_POINT. Когда первая строка готова для обработки, LINE_PROC устанавливает FLAG в 1 и изменяет BUFF_POINT на BUFFER2. Она помещает адрес BUFFER1 в BX и счетчик символов в DI и при обработке строки использует содержимое этих двух регистров. С появлением каждой последующей строки содержимое FLAG, которое инвертируется при вызове LINE_PROC, вызывает коммутацию адресов BUFFER1 и BUFFER2. Пока заполняется один буфер, происходит обработка другого буфера и наоборот. Отметим, что время обработки строки должно быть меньше времени ввода строки. Чтобы обеспечить завершение инициализации до ввода следующей строки, при коммутации буферов необходимо запрещать прерывания. Когда IF устанавливается в 1, посылаемые терминалом символы вызывают прерывания и помещаются в текущий заполняемый буфер.

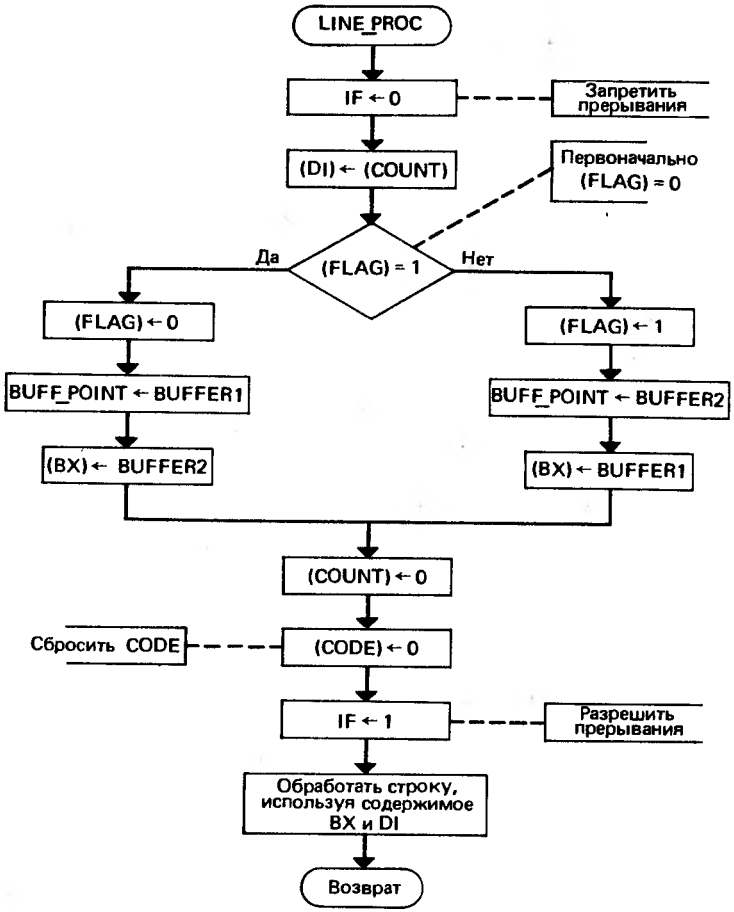


Рис. 6.13. Схема LINE_PROC при использовании двойного буферирования

Ранее мы специально не учитывали возможность одновременного (или, по крайней мере, в одном и том же цикле команды) появления нескольких прерываний. Ничего также не говорилось о введении в систему прерываний механизма приоритетов. Имеется несколько способов организации приоритетов для ввода-вывода по прерываниям. Одни из них реализуются аппаратно, другие — программно, а третьи — комбинацией первых двух. Рассмотрим следующие средства введения приоритетов в систему прерываний: опрос, приоритетная цепочка и схема управления приоритетными прерываниями.

Хотя для интерфейсов в системе на базе микропроцессора 8086 имеются многочисленные типы внешних прерываний и редко требуется назначать интерфейсам один и тот же тип прерывания, это можно сделать и для учета при-

оритетов применить опрос. Введя в начале процедуры прерывания программу, аналогичную показанной на рис. 6.2, можно установить приоритеты интерфейсам в том порядке, в котором они опрашиваются программой.

Простым аппаратным решением введения приоритетов является *приоритетная цепочка*. В каждом интерфейсе предусматривается логическая схема и прохождение сигнала подтверждения через зти схемы показано на рис. 6.14,*а*. Более подробно логическая схема приоритетной цепочки представлена на рис. 6.14, *б*. Если интерфейс выдал запрос, активный сигнал $\overline{\text{INTA}}$ заставляет выдать в интерфейс активный сигнал подтверждения прерывания, а передача $\overline{\text{INTA}}$ в следующий интерфейс блокируется. Когда же запрос интерфейса от-

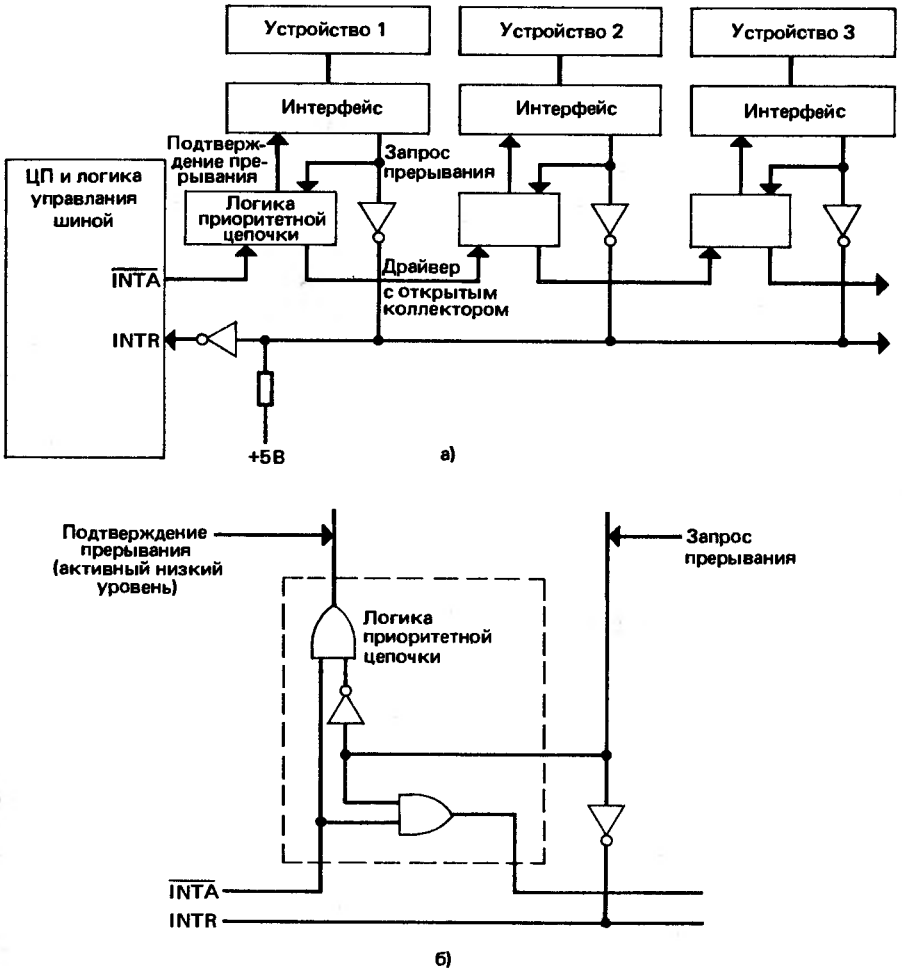


Рис. 6.14. Структурная (*а*) и логическая (*б*) схемы приоритетной цепочки

сутствует, сигналу $\overline{\text{INTA}}$ разрешается проходить через схему приоритетной цепочки. Следовательно, по мере прохождения сигнала $\overline{\text{INTA}}$ по цепочке ближайший к ЦП запрашивающий интерфейс "перехватывает" сигнал подтверждения, выдает тип и завершает последовательность прерывания; после обслуживания запрос снимается. Более удаленные по цепочке интерфейсы, сделавшие запросы, не получают сигнала подтверждения и сохраняют свои запросы. После того как команда STI разрешает прерывания или команда IRET восстанавливает PSW с IF = 1, ЦП распознает другие запросы и выдает новый сигнал $\overline{\text{INTA}}$. К этому времени интерфейс, запрос которого обслуживался, снимает свой запрос и не препятствует прохождению сигнала $\overline{\text{INTA}}$. Таким образом приоритет интерфейса определяется его положением в цепочке: чем ближе интерфейс к ЦП, тем выше его приоритет.

Более гибкий аппаратно-программный механизм приоритетов реализуется программируемой схемой управления приоритетными прерываниями, которая входит в логику управления шиной. Общий вид такой схемы и место ее в системе показаны на рис. 6.15. Линии INTR и $\overline{\text{INTA}}$ подключаются только к схеме управления, а не к интерфейсам. Линии запросов прерываний от

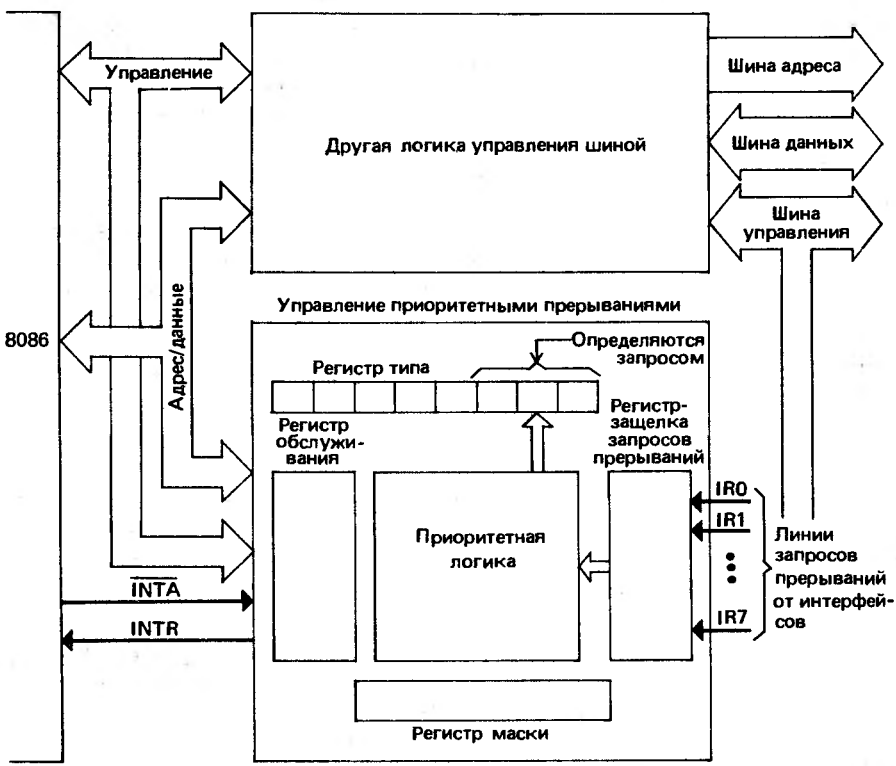


Рис. 6.15. Типичная схема управления приоритетными прерываниями

интерфейсов подаются в схему управления и не объединяются. Схема управления содержит логику, необходимую для назначения приоритетов поступающим запросам. Например, наибольший приоритет можно назначить IR0, следующий — IR1 и т. д. Когда запрос прерывания распознается приоритетной логикой как имеющий наибольший приоритет, три младших бита в регистре типа устанавливаются на номер линии с запросом, устанавливается бит в регистре обслуживания и выдается прерывание в ЦП. Если IF =1, ЦП возвращает сигнал подтверждения и схема управления выдает в ЦП тип прерывания. Все запросы с меньшими приоритетами блокируются до тех пор, пока не будет сброшен бит в регистре обслуживания (это обычно реализует процедура прерывания). Следовательно, когда команда STI повторно разрешает прерывания, запросы с более высоким приоритетом могут прервать текущую процедуру прерывания, тогда как запросы с меньшими приоритетами остаются заблокированными до сброса установленного бита в регистре обслуживания. Теперь процедура прерывания может управлять восприятием запросов с меньшими приоритетами. Чтобы программа могла сбрасывать биты в регистре обслуживания, он должен быть программируемым, т. е. должен иметь адрес порта ввода-вывода, делающий регистр доступным командам IN и OUT. Кроме встроенных приоритетов, предусмотрен однобайтный регистр маски, позволяющий маскировать отдельные запросы. Бит n в этом регистре маскирует вход IRn. Предполагается, что этот регистр также программируемый.

Младшие три бита регистра типа прерывания определяются запросом, выбранным приоритетной логикой. Если сделать регистр программируемым, появляется возможность инициализировать пять старших бит при включении системы.

Во многих микропроцессорных семействах имеются микросхемы управления приоритетными прерываниями. Для работы с микропроцессорами 8086/8088 предлагается программируемый контроллер прерываний 8259A фирмы Intel. Он аналогичен схеме, показанной на рис. 6.15, но имеет больше возможностей (подробнее см. гл. 8).

Ясно, что в системе с несколькими устройствами ввода-вывода, которые осуществляют ввод и (или) вывод в процессе обработки, аппаратное и программное обеспечение прерываний будет очень сложным. Особенно это относится к мультипрограммным системам. Независимо от того, является ли программная система узкоспециализированной или широкого назначения, в ней обычно имеется основная программная компонента, которая управляет всеми другими. В системе широкого назначения это резидентный монитор. В любом случае именно основная компонента инициализирует систему при ее включении. Среди других действий она размещает по младшим адресам памяти (00000 . . . 003FF) указатели прерываний. В специализированной системе установка указателей прерываний обычно остается фиксированной до выключения системы, но в мультипрограммной среде операционная система при необходимости может изменить некоторые указатели прерываний.

Усложнение программной системы связано с проблемой взаимодействия с процедурами прерываний. Хотя при вызове процедур передача адресов па-

раметров через стек или таблицу реализуется довольно просто, этот метод не применим для процедур прерываний, обрабатывающих внешние прерывания. Поскольку внешние прерывания возникают в произвольных точках программы, невозможно установить, где ввести код связи, требуемый для передачи параметров. Следовательно, параметры необходимо передавать через общие области и (или) абсолютные ячейки. В специализированной однопрограммной системе можно скомпилировать и связать все программные компоненты так, чтобы они взаимодействовали без передачи параметров. Но, если действиями управляет операционная система, именно она должна следить за программными компонентами, находящимися в памяти, а передачи данных между процедурами прерываний и другими программными компонентами осуществляются с помощью операционной системы. Обычно операционная система поддерживает несколько таблиц, называемых *системными*, в которых содержится важная информация о состоянии системы. Некоторые таблицы, например содержащие важные физические характеристики устройств ввода-вывода, оказываются статическими, а другие, к которым относится и таблица начальных адресов процедур, — динамическими. В мультипрограммных системах появляются очереди ожидающих обслуживания запросов ввода-вывода; конечно, ассоциируемые с этими очередями таблицы являются динамическими.

6.4. БЛОКОВЫЕ ПЕРЕДАЧИ И ПРЯМОЙ ДОСТУП К ПАМЯТИ

Если скорость передачи данных в устройстве ввода-вывода относительно невелика, взаимодействие с ним организуется с помощью программного ввода-вывода или прерываний. Однако, иногда выполнение команд и последовательностей прерываний занимает больше времени, чем это допустимо. Накопители на лентах и дисках и аналого-цифровые преобразователи работают со столь высокой скоростью, что быстродействия ЦП при передаче отдельных байт или слов недостаточно. Скорость передачи данных в этом случае определяется самими устройствами, а не ЦП, и компьютер должен осуществлять ввод-вывод в соответствии с максимальной скоростью устройства. В дисковом накопителе скорость передачи данных определяется скоростью их прохождения под головкой считывания/записи и часто превышает 200 000 байт/с. Следовательно, на передачу байта в (из) память (и) приходится менее 5 мкс. При таком быстродействии требуются блоковые передачи, в которых для непосредственного взаимодействия с памятью применяются контроллеры ПДП.

Действия, обеспечивающие передачу по системной шине байта или слова, называются *циклом шины*. Выполнение команды может потребовать более одного цикла шины; например, команда MOV AL,TOTAL кроме циклов выборки команды инициирует цикл шины для передачи содержимого TOTAL. Блоковая передача 100 байт реализуется за 100 циклов шины, если передаются байты, и за 50 циклов шины, если передаются слова, начинающиеся по четному адресу.

В течение любого цикла шины системной шиной управляет одна из подключенных к ней компонент. Эта компонента в течение данного цикла называется *ведущей*, а компонента, с которой она взаимодействует, – *ведомой*. Обычно ведущим является ЦП с его логикой управления шиной, но и другие компоненты могут получить управление шиной, выдавая в ЦП *запрос шины*. После завершения текущего цикла шины ЦП возвращает сигнал *разрешения* (*подтверждения*) *шины* и выдавшая запрос компонента становится ведущей. Получение управления шиной на один цикл шины называется *пропуском цикла* (буквально – "кражей" цикла). Как и логика управления шиной, ведущий шины во время цикла шины должен помещать адрес на шину адреса и управлять действиями на шине. Компонентами, которые могут стать ведущими шины, являются процессоры (с их логикой управления шиной) и контроллеры ПДП. Иногда контроллер ПДП связан с одним интерфейсом, но часто он рассчитан на подключение нескольких интерфейсов.

Микропроцессор 8086 воспринимает запросы шины по входу HOLD и выдает разрешения через выход подтверждения запроса HLDA. Запрос производится, когда потенциальная ведущая компонента шины формирует сигнал 1 на входе HOLD. Обычно после завершения текущего цикла шины микропроцессор реагирует сигналом 1 на выходе HLDA. Когда запрашивающее устройство получает этот сигнал разрешения, оно становится ведущим шины и остается им до снятия своего сигнала на входе HOLD. По окончанию сигнала HOLD микропроцессор сразу же снимает сигнал HLDA. Одно из исключений обычной последовательности возникает при обращении к слову по нечетному адресу; передача происходит за два цикла шины и разрешение не выдается до окончания второго цикла. Другие исключения рассматриваются в последующих главах.

Контроллер ПДП, становясь ведущим шины, помещает адрес памяти на шину адреса и посылает в интерфейс необходимые сигналы, чтобы заставить его выдать данные или принять данные с шины данных. Так как контроллер ПДП сам определяет, когда снять запрос шины, он может вернуть управление ЦП после передачи одного данного, а при готовности следующего данного вновь запросить управление, но он может управлять шиной и до завершения передачи целого блока. Обычно применяется первый способ, так как ЦП может продолжать свои действия до готовности следующего данного. Последовательность действий предпринимаемых при выводе одного данного в режиме ПДП, иллюстрируется рис. 6.16, где цифры обозначают следующие действия:

1 – интерфейс готов принимать данные и делает запрос ЦП;
2 – сформирован запрос шины;
3 – возвращается разрешение шины;
4 – контроллер ПДП помещает адрес на шину адреса;
5 – подтверждается запрос ПДП;
6 – память помещает данные на шину данных;

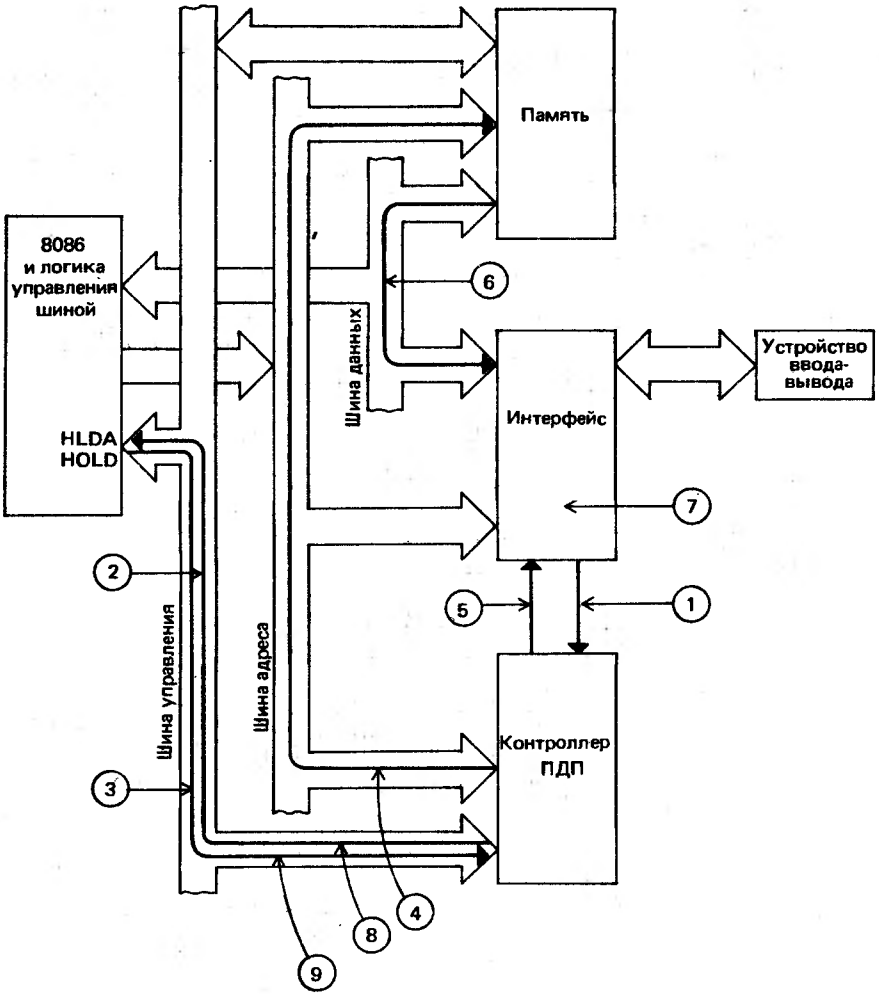


Рис. 6.16. Вывод одного данного в блоковой передаче

7 – интерфейс фиксирует данные;

8 – снимается запрос шины и управление возвращается микропроцессору 8086;

9 – микропроцессор 8086 снимает разрешение шины.

Блоковая передача представляет собой последовательность рассмотренных передач одного данного. В каждом следующем цикле ПДП используется смежная ячейка памяти. Хотя разработано множество конфигураций контроллеров ПДП, все они должны удовлетворять нескольким требованиям. Так как, являясь ведущим шины, контроллер ПДП должен выдавать адрес,

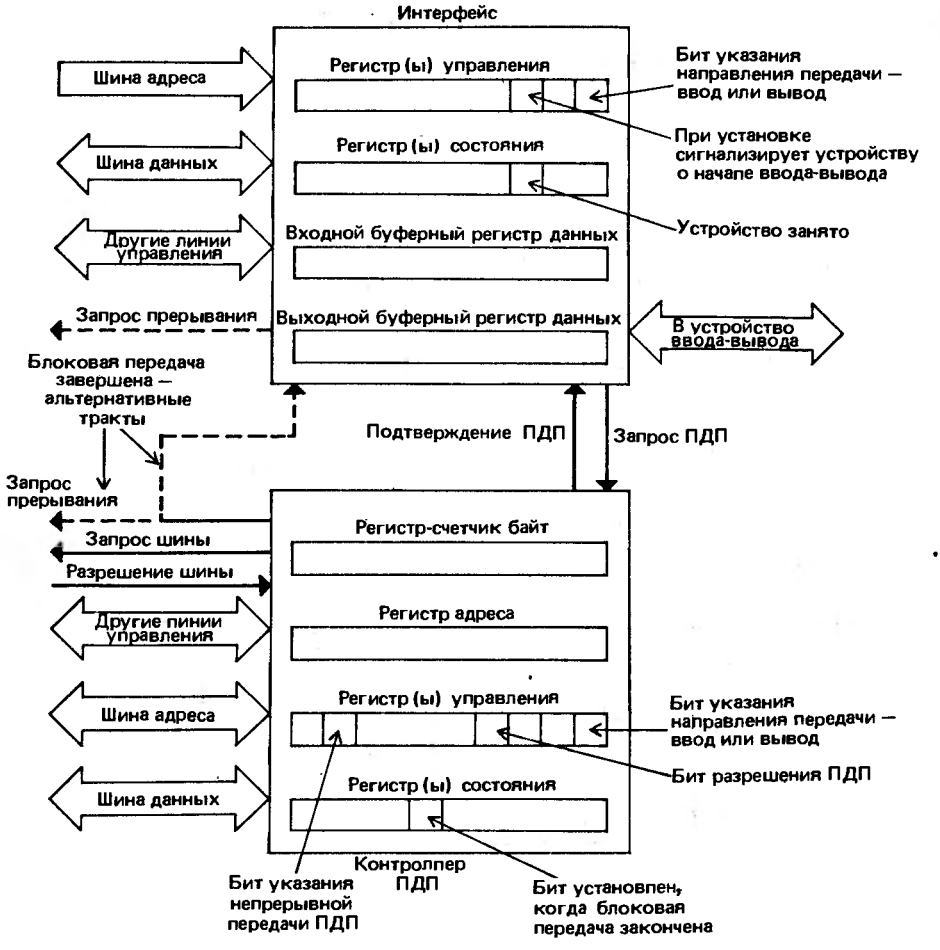


Рис. 6.17. Минимальная конфигурация контроллера ПДП и интерфейса

в нем необходимы средства хранения следующего адреса памяти. Он должен также узнавать, когда остановить блоковую передачу. На рис. 6.17 приведена минимальная конфигурация контроллера ПДП и интерфейса. Кроме регистров состояния и управления, в контроллере потребуются еще два регистра: для хранения адреса следующей ячейки памяти и числа байт, подлежащих передаче. После передачи каждого данного производится инкремент (или декремент) регистра адреса и декремент счетчика байт. Инкремент и декремент осуществляются на 1 при передаче байт и на 2 при передаче слов. Во время блоковой передачи одного байта при вводе (данное передается из интерфейса в память) имеет место такая последовательность событий:

1. Интерфейс выдает в контроллер запрос ПДП.
2. Контроллер получает управление шиной.
3. Содержимое регистра адреса помещается на шину адреса.
4. Контроллер посылает в интерфейс подтверждение, заставляющее интерфейс выдать данные на шину данных (при выводе по этому сигналу интерфейс фиксирует данные, находящиеся на шине).
5. Байт данных передается в ячейку памяти, определяемую шиной адреса.
6. Контроллер освобождает шину.
7. Инкремент регистра адреса на 1.
8. Декремент счетчика байт на 1.
9. Если счетчик байт не содержит 0, вернуться к шагу 1; в противном случае остановить передачу.

На схеме показаны двунаправленные линии адреса, подключенные к контроллеру, и однонаправленные линии адреса в интерфейс. Это объясняется тем, что контроллер может стать ведущим шины и выдавать адреса на шину, а интерфейс может только принимать адреса. К интерфейсу и контроллеру идут двунаправленные линии данных, хотя только интерфейс может передавать данные между шиной данных и устройством ввода-вывода. Как и в интерфейсе, ЦП должен взаимодействовать с регистрами контроллера, а это осуществляется по линиям данных.

Если интерфейс подключен к устройству, которое может производить ввод и вывод, в его регистрах состояния и управления потребуется бит, указывающий тип выполняемой передачи. Кроме того, в этих регистрах необходимы бит "начать" для инициирования действий ввода-вывода (устройство проверяет этот бит) и бит, показывающий, занято устройство или нет. В регистрах состояния и управления контроллера необходим бит разрешения, управляющий восприятием запросов ПДП от устройства, и бит, определяющий тип ПДП (освобождается шина между передачами или запрос шины сохраняется активным в течение всей передачи). Потребуется также бит направления данных, чтобы, став ведущим шины, контроллер мог определить операцию ввода или вывода. Для инициирования блоковой передачи необходимо соответственно установить управляющие биты интерфейса и контроллера и загрузить начальные значения в регистр адреса и счетчик байт. Последним должен устанавливаться тот бит в регистре управления интер-

```
IDLE:   IN      AL,INSTAT       ;ОЖИДАТЬ ГОТОВНОСТИ УСТРОЙСТВА
        TEST    AL,4
        JNZ     IDLE
        MOV     AX,BYTE_COUNT   ;ЗАГРУЗИТЬ РАЗМЕР БЛОКА
        OUT     BC_REG,AX       ;В СЧЕТЧИК БАЙТ
        LEA     AX,BUFFER       ;ЗАГРУЗИТЬ НАЧАЛЬНЫЙ АДРЕС
        OUT     ADDR_REG,AX     ;В РЕГИСТР АДРЕСА
        MOV     AL,DMAC         ;УСТАНОВИТЬ БИТЫ НАПРАВЛЕНИЯ
        OR      AL,0AH          ;И РАЗРЕШЕНИЯ В УПРАВЛЯЮЩЕМ БАЙТЕ
        OUT     DMACON,AL       ;И ВЫВЕСТИ В КОНТРОЛЛЕР
        MOV     AL,INTC         ;УСТАНОВИТЬ БИТЫ НАПРАВЛЕНИЯ
        OR      AL,5            ;И "НАЧАТЬ" В РЕГИСТРЕ ПРИКАЗОВ
        OUT     INTCON,AL       ;ИНТЕРФЕЙСА
                .
                .
                .
```

Рис. 6.18. Типичный фрагмент инициирования блоковой передачи

фейса, который сигнализирует устройству ввода-вывода о выполнении операции, – бит "начать".

Типичный фрагмент запуска блоковой передачи при вводе приведен на рис. 6.18. В нем предполагаются следующие определения бит:

Бит 2 в INSTAT. Бит занятости устройства ввода-вывода.

Бит 1 в DMACON. Информирует контроллер о направлении передачи (при вводе содержит 1).

Бит 3 в DMACON. Разрешает контроллер и он будет воспринимать запросы ПДП.

Бит 6 в DMACON. Содержит 0, если шина должна освобождаться между передачами.

Бит 0 в INTCON. Информирует интерфейс о направлении передачи (при вводе содержит 1).

Бит 2 в INTCON. Бит "начать", который запускает действия ввода-вывода.

После выполнения приведенного фрагмента устройство ввода-вывода начинает ввод данных, а контроллер ПДП запрашивает цикл шины и передает байт из интерфейса в память каждый раз, когда байт помещается в буферный регистр данных интерфейса.

Если к системе на базе микропроцессора 8086 подключен контроллер ПДП, который выдает только младшие 16 бит адреса, контроллер может управлять блоковыми передачами в один из 64К сегментов памяти. Чтобы иметь доступ ко всей памяти, контроллер должен генерировать 20-битные адреса, а микропроцессор должен передавать в контроллер как старшие 4 бита начального адреса, так и младшие 16 бит. Поэтому по сравнению с рис. 6.16 в контроллер потребуется выводить из ЦП минимум еще один дополнительный байт адреса. Этот байт следует выводить в отдельный порт ввода-вывода, инкремента которого во время блоковой передачи не производится. При этом в передаче участвует один сегмент 64К, хотя в различных блоковых передачах он может быть разным.

По завершению блоковой передачи в одном из регистров состояния контроллера устанавливается бит "закончено", а на один из контактов контроллера выдается сигнал. Текущая программа может обнаружить конец блоковой передачи, периодически проверяя бит "закончено"; выходной же сигнал контроллера можно использовать для инициирования прерывания. В последнем случае сигнал либо непосредственно подается в логику управления шиной, либо посылается в интерфейс, который и обрабатывает прерывание. Обычно контроллер ПДП не формирует тип прерывания; поэтому, если сигнал подается непосредственно в логику управления шиной, в ней должна быть схема приоритетного управления. Часто сигнал о завершении подается в интерфейс, так как генерировать прерывание требуется и по другим причинам, кроме завершения блоковой передачи, например из-за ошибок, которыми занимается только интерфейс и которых не касается контроллер.

В интерфейсе и контроллере имеются регистры состояния, которые во время передачи фиксируют важную информацию. Конкретная информация зависит от системы, но в любом случае необходимо отмечать ошибки переда-

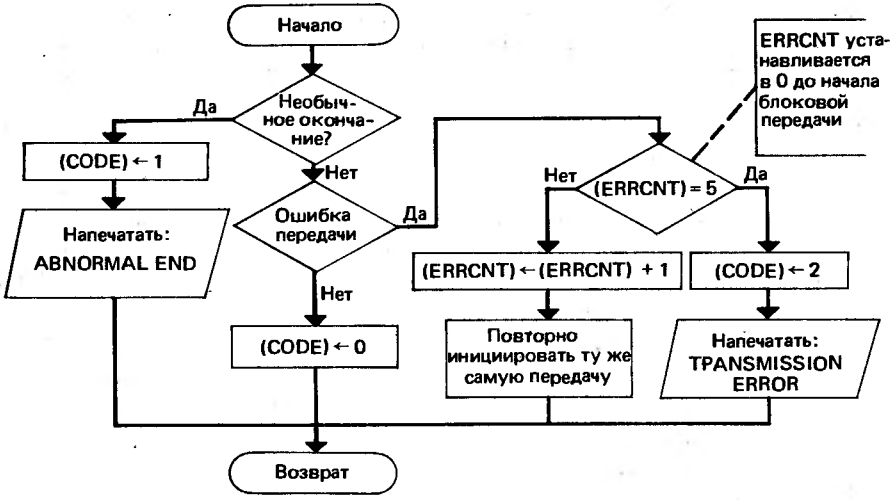


Рис. 6.19. Схема процедуры завершения

чи или конец данных до достижения счетчиком байт нуля (например, конец ленты). После завершения передачи или аварийного прекращения текущая программа или процедура прерывания должны проверить регистры состояния и предпринять соответствующие действия, в частности повторный ввод той же самой информации и (или) печать сообщения об ошибке. Процедура, выполняющая эти проверки и действия, называется *процедурой завершения.*

Схема процедуры завершения на рис. 6.19 обслуживает ввод с диска. Процедура контролирует аварийное прекращение и ошибки при передаче. В первом случае печатается сообщение об ошибке и в ячейку памяти CODE загружается 1. При отсутствии аварийного прекращения, но при наличии ошибки передачи переменная ERRCNT (которая до начала передачи сбрасывается в 0) сравнивается с 5. Если ERRCNT < 5, производится ее инкремент и инициируется повторная передача; в противном случае CODE устанавливается на 2 и печатается сообщение об ошибке. Таким образом, до прекращения передачи ошибка должна появиться в пяти последовательных блоковых передачах. Наконец, когда передача завершается без ошибки, единственное действие процедуры заключается в установке CODE на 0.

Когда к контроллеру ПДП подключается несколько интерфейсов, некоторые регистры приходится дублировать. Если интерфейсы могут выполнять блоковые передачи одновременно, в контроллере для каждого интерфейса должны быть свои счетчик байт и регистр адреса. Кроме того, контроллер должен учитывать приоритеты запросов. Регистры управления контроллера должны иметь биты индивидуального разрешения входных линий запросов и управления отдельными блоковыми передачами. В регистрах состояния необходимы биты, индивидуально показывающие, какие интерфейсы заверши-

ли свои передачи. Если контроллер имеет только один выход для сигнала о завершении, этот сигнал указывает только факт завершения какой-то передачи, а какой именно – должна определять процедура завершения, контролируя биты состояния контроллера.

Хотя конфигурация контроллера ПДП не зависит от обслуживаемых им устройств, интерфейс должен учитывать характеристики подключенного к нему устройства ввода-вывода. Если интерфейс связан не с устройством внешней памяти, достаточно минимальной конфигурации, приведенной на рис. 6.17, но при связи с устройством внешней памяти интерфейс должен осуществлять поиск и адресацию информации в нем. В интерфейсе подсистемы одноканального аналого-цифрового преобразования обычно не требуется более 2 – 3 байт информации управления и состояния, но оказываются необходимыми следующие биты:

1. Разрешения прерывания.
2. Указания ошибок.
3. Определения частоты дискретизации.
4. Разрешения ПДП.
5. Инициирования ввода (т. е. установки бита "начать").

Некоторые интерфейсы, в том числе и для подсистем аналого-цифрового преобразования, рассчитаны на передачи байт или слов, а также блоковые передачи. Даже если такие интерфейсы подключаются к контроллерам ПДП, они могут работать и без них. Если в подсистеме аналого-цифрового преобразования имеется несколько каналов, в интерфейсе потребуются регистры для определения номера канала и режима мультиплексирования.

В подсистеме магнитной ленты необходимы биты управления для определения таких параметров, как:

1. Выбираемый накопитель.
2. Плотность информации в битах на дюйм и скорость ленты.
3. Движение ленты (перемотка, вперед и т. п.).

Биты состояния должны отражать:

1. Ошибки хранения и передачи.
2. Обнаружение конца файла.
3. Обнаружение конца ленты.
4. Выбранное устройство не включено.
5. Выбранное устройство занято.

В ленточных накопителях информация хранится в виде файлов и записей, сопровождаемых идентификаторами. Она считывается посредством поиска идентификаторов, а не адресации их. Следовательно, в интерфейсах ленточных накопителей не нужны регистры адреса для указания физического положения информации.

Кроме рассмотренной управляющей информации, интерфейсы дисковых накопителей, памяти на цилиндрических магнитных доменах и других видов устройств адресуемой внешней памяти должны иметь регистры для указания физического положения информации. Например, для доступа к диску

необходимо определять поверхность, дорожку и сектор. Следовательно, на фазе инициализации блоковой передачи потребуется указывать эту информацию, а также другую управляющую информацию.

Выше приведено краткое введение во взаимодействие с быстродействующими внешними устройствами, а подробное обсуждение этих устройств и их интерфейсов содержится в гл. 9.

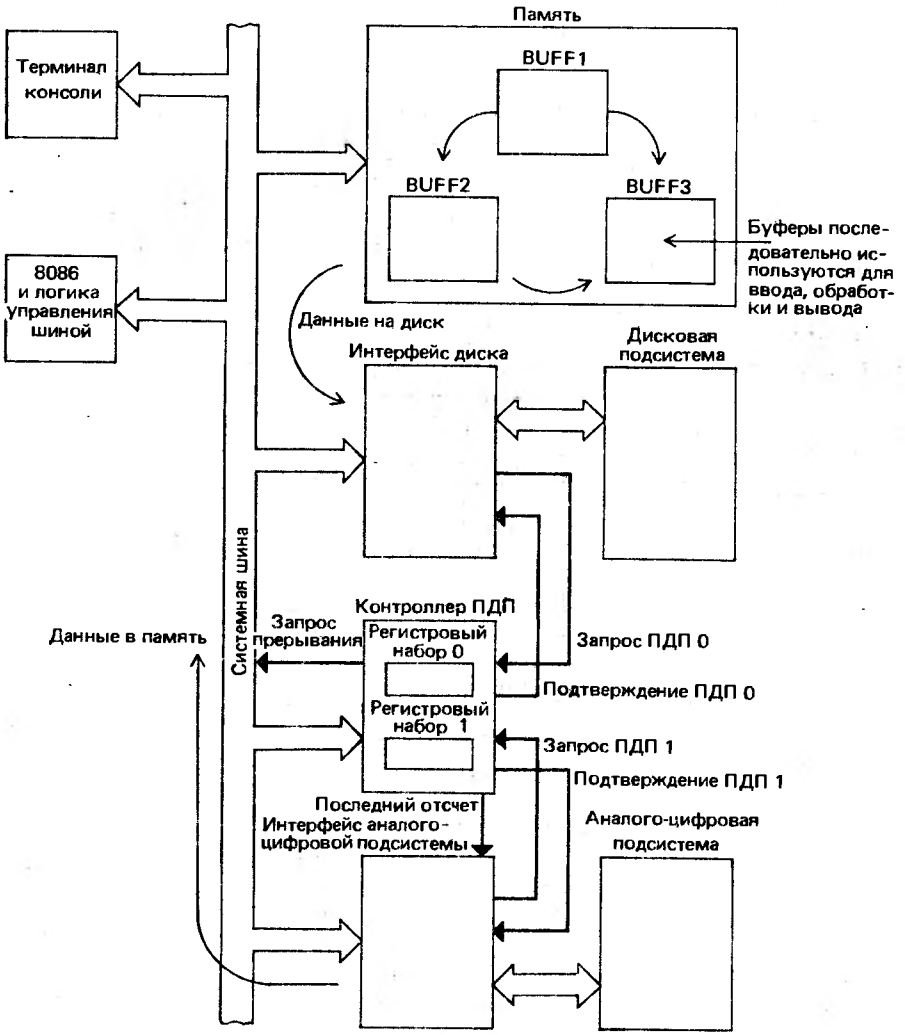


Рис. 6.20. Аппаратная конфигурация для примера с тройным буферированием

6.5. ПРИМЕР ПРОЕКТИРОВАНИЯ СИСТЕМЫ ВВОДА-ВЫВОДА

Чтобы объединить изложенные общие принципы организации ввода-вывода, рассмотрим достаточно сложный пример. Предположим, что данные непрерывно вводятся в память от аналого-цифрового преобразователя, обрабатываются, а затем выводятся на диск. Для реализации этих действий применяются блоковые передачи совместно с тройным буферированием. Интер-

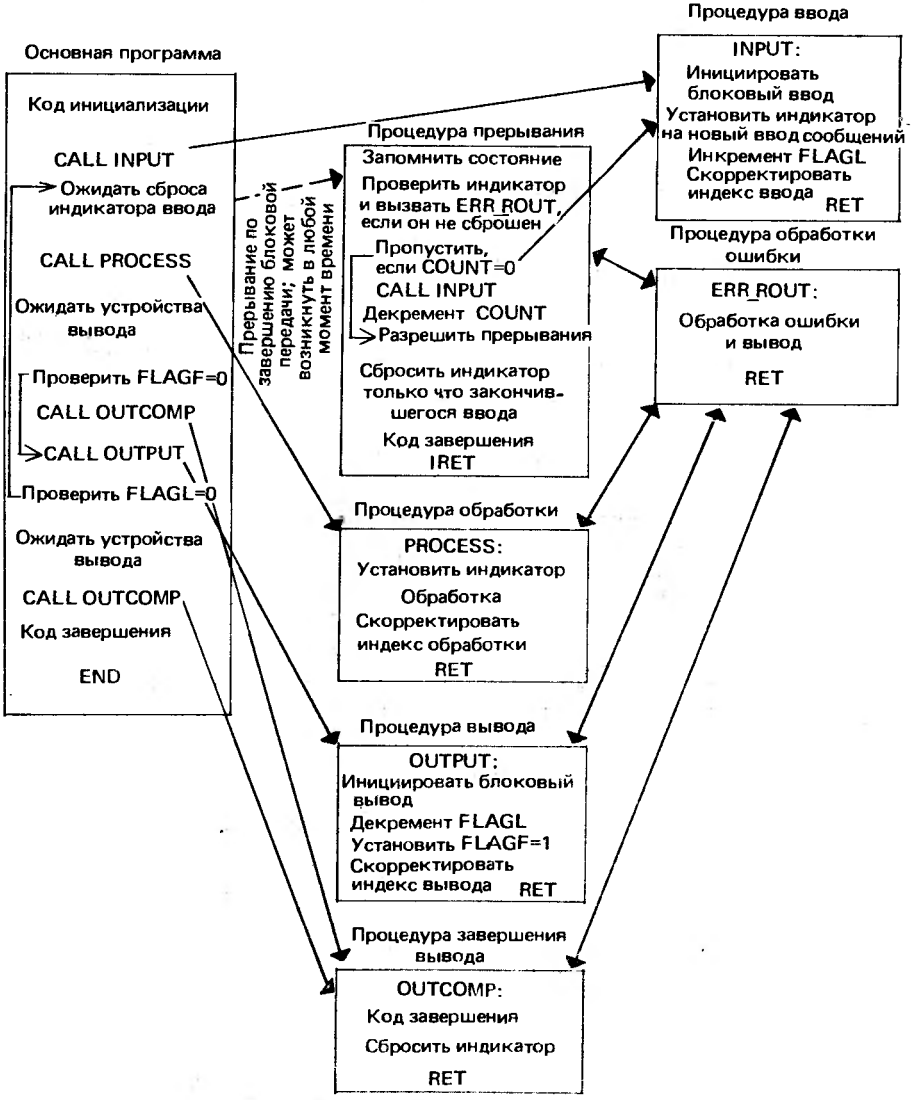
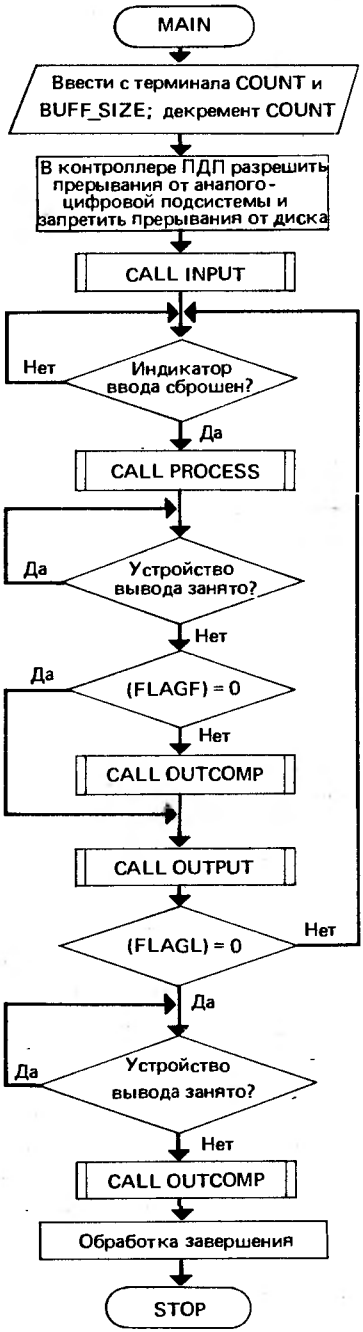


Рис. 6.21. Диаграмма взаимосвязей компонент



фейсы диска и преобразователя подключены к одному контроллеру ПДП. Конфигурация аппаратной системы показана на рис. 6.20. Ради простоты предполагается, что все три буфера находятся в младшем сегменте 64К памяти, поэтому контроллеру требуется только 16-битный адрес.

Проектируемая программа разделена на задачи, взаимосвязи между которыми отражены на рис. 6.21. Имеются основная программа, процедура прерывания и пять процедур. Четыре процедуры вызываются основной программой, а одна из них – процедура ввода – вызывается также процедурой прерывания. Пятая процедура ERR_ROUT вызывается при возникновении ошибки в других компонентах программы. Процедура INPUT инициирует блоковую передачу при вводе из подсистемы аналого-цифрового преобразования, процедура PROCESS реализует обработку, OUTPUT инициирует блоковую передачу при выводе на диск, а OUTCOMP выполняет задачи завершения после каждого блокового вывода. Прерывание возникает тогда, когда заканчивается передача при вводе, но не после передачи при выводе. Процедура прерывания вызывает процедуру INPUT для запуска новой передачи при вводе, если она необходима, а затем выполняет задачи завершения для только закончившейся передачи при вводе.

Схема основной программы приведена на рис. 6.22, а сводка важных переменных – на рис. 6.23. Сначала основная программа вводит с терминала число вводимых блоков данных и число байт в каждом блоке. Эти значения помещаются соответственно в COUNT и BUFF_SIZE. Затем программа производит декремент COUNT, корректирует необходимые биты управления в контроллере ПДП (включая биты разрешения прерываний в конце передач

Рис. 6.22. Схема основной программы

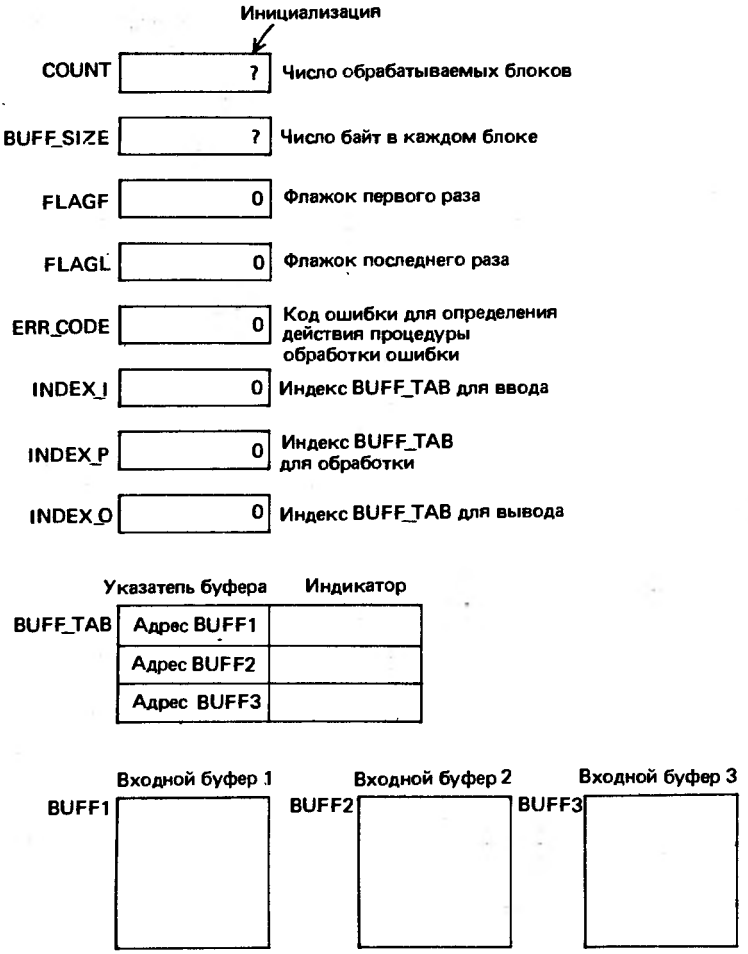


Рис. 6.23. Важнейшие переменные в примере с тройным буферированием

от преобразователя и запрещения их в конце передач на диск) и запускает первую передачу ввода, вызывая INPUT. Затем она входит в цикл, который вводит, обрабатывает и выводит последовательные блоки данных посредством ввода их в один буфер, обработки содержимого второго буфера и вывода из третьего буфера. Первым ячейкам трех буферов даны имена BUFF1, BUFF2 и BUFF3. Выход из цикла осуществляется при достижении счетчиком нуля и после того, как введенные блоки обработаны и выведены. После цикла процедура OUTCOMP осуществляет окончательный вывод и реализуется обработка завершения.

Как это часто бывает, необходимо предпринять специальные действия при запуске и завершении процесса. Чтобы пропустить процедуру завершения

вывода при первом проходе цикла (т. е. до инициирования первого вывода) применяется флажок FLAGF "первого раза". Он сначала сбрасывается в 0 и устанавливается в 1 процедурой OUTPUT после того, как она начинает первую передачу вывода. После запуска последнего ввода цикл необходимо повторять до обработки и вывода всех блоков. Для проверки этого предусмотрена переменная FLAGL, участвующая в управлении циклом. В начале каждого ввода производится ее инкремент, а в начале каждого вывода – декремент. Так как переменная FLAGL принимает значение 0 только после окончания всех вводов и обработок и при запуске последнего вывода, цикл повторяется до тех пор, пока переменная FLAGL не равна нулю, и заканчивается, когда FLAGL возвращается к нулю. При каждом проходе по циклу программа должна проверять, закончен ли ввод следующего обрабатываемого блока, а затем обрабатывать его и выводить следующий буфер. Однако перед запуском вывода программа должна убедиться, что диск не занят предыдущим выводом. Если диск занят, программа должна зафиксировать, когда он станет доступным. Новый ввод начинается всякий раз, когда возникает прерывание в конце предыдущего ввода; следовательно, процедура ввода явно из цикла не вызывается.

Чтобы обеспечить циклическое использование буферов для задач ввода, обработки и вывода с гарантией невозможности использовать один и тот же буфер одновременно, образована таблица с начальным адресом BUFF_TAB. Таблица содержит три пары слов; в первом слове каждой пары находится адрес буфера, а во втором – индикатор. Индикатор устанавливается в 1, когда начинается ввод в буфер, и сбрасывается в 0 по завершению ввода. Кроме того, индикатор устанавливается в начале обработки содержимого буфера и сбрасывается после вывода из буфера. Перед запуском ввода или обработки буфера проверяется индикатор и, если буфер уже используется, он считается недоступным. Этим исключается наложение ввода на обработку или вывод и наоборот. Отметим, что в отличие от задачи ввода, которая инициируется прерыванием в любой момент времени, задачи обработки и вывода выполняются в фиксированной последовательности. Этим гарантируется невозможность совмещения обработки и вывода одного и того же блока данных.

С таблицей BUFF_TAB связаны три индекса INDEX_I, INDEX_P и INDEX_O, которые адресуют пары в BUFF_TAB. Когда инициируется ввод, INDEX_I показывает относительную позицию пары адрес/индикатор того буфера, в который осуществляется ввод. Он корректируется на 4 сразу после запуска ввода и правильно устанавливается для следующего ввода. Аналогичным образом индексы INDEX_P и INDEX_O адресуют пары текущих буферов, участвующих в обработке и выводе. Данные указатели циркулируют по мере циклического использования буферов.

После передач ввода и вывода расположены фрагменты завершения, которые контролируют ошибки, возникшие в предыдущей передаче, и подготавливают систему для следующих ввода или вывода. Ошибки обнаруживаются в результате анализа бит состояния в соответствующем интерфейсе. Каждому

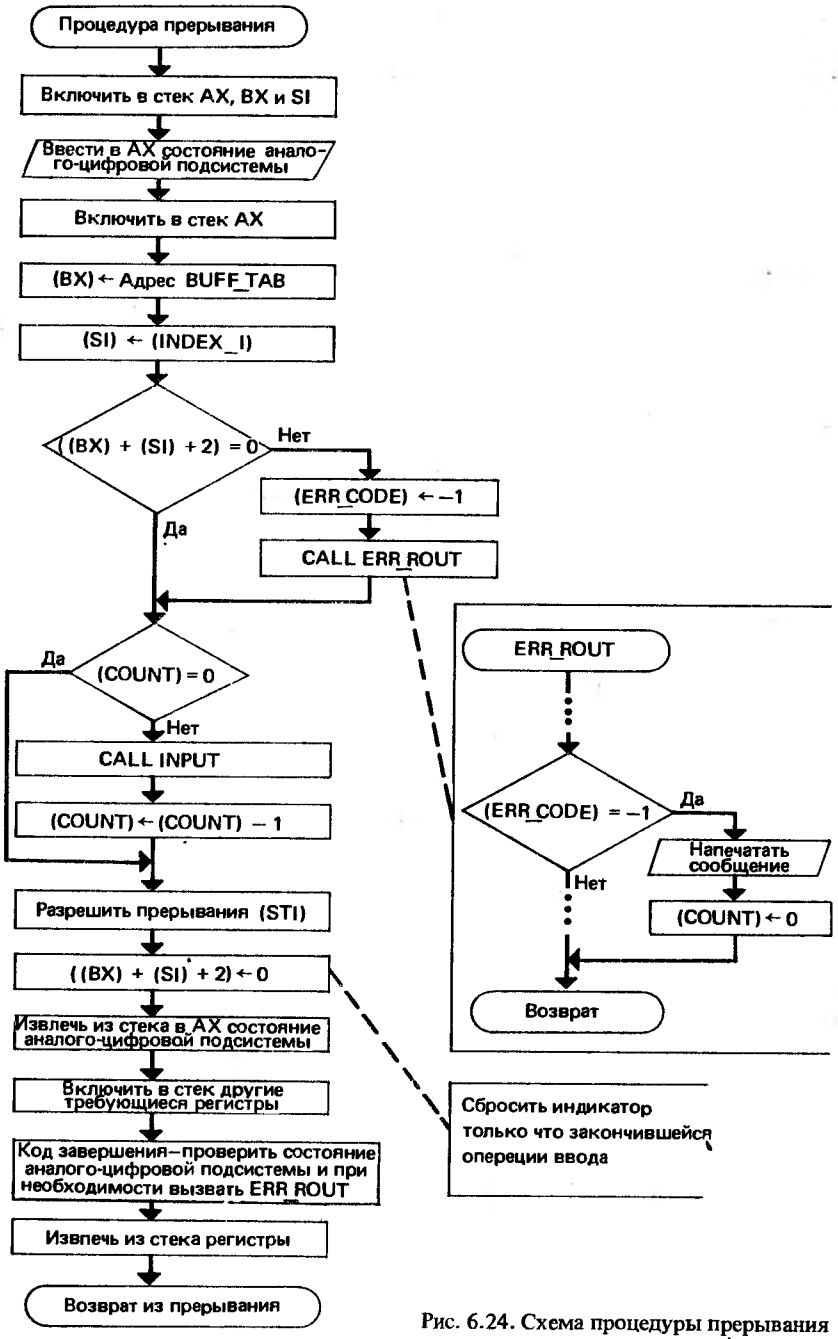


Рис. 6.24. Схема процедуры прерывания

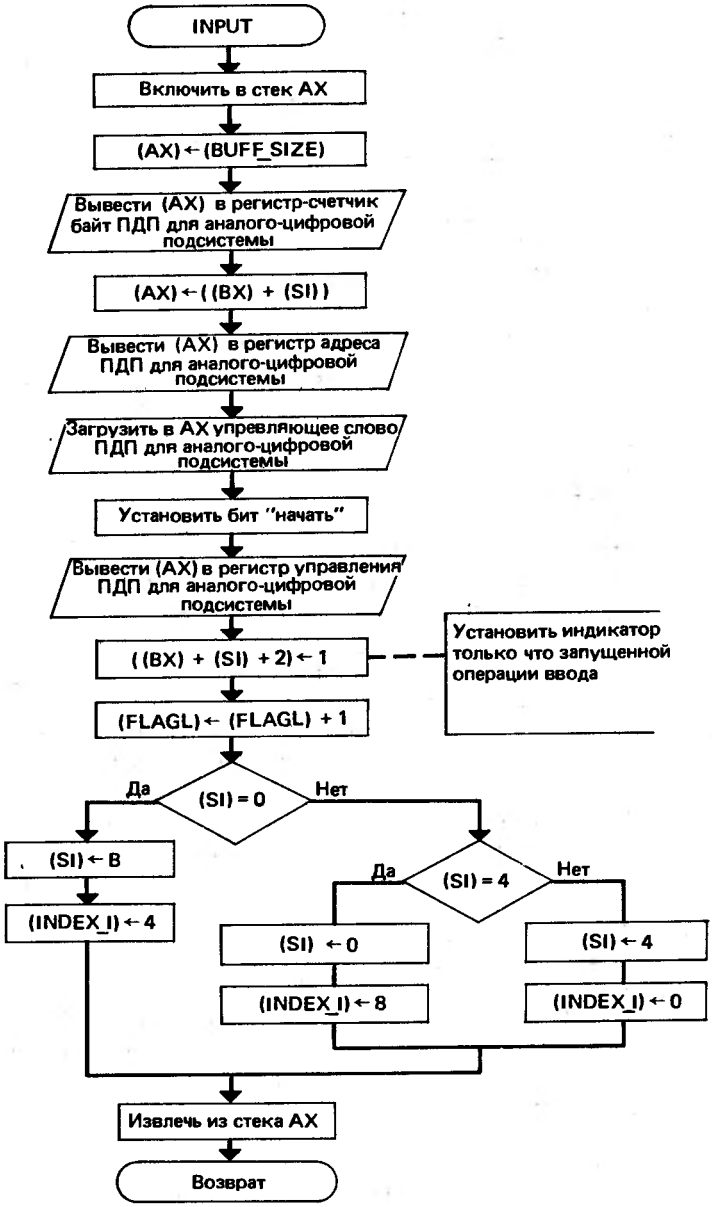


Рис. 6.25. Схема процедуры INPUT

типу ошибок соответствует уникальный код и при обнаружении ошибки этот код помещается в ERR_CODE и вызывается ERR_ROUT. Процедура

ERR_ROUT обрабатывает ошибку и осуществляет возврат, но в случае неисправимых ошибок, называемых *фатальными*, вызывает окончание программы. Например, подсистема аналого-цифрового преобразования должна вводить данные непрерывно. Так как на последовательность прерывания и обработку, которая должна быть произведена до запуска следующего ввода, расходуется время, возможна потеря данных при коммутации буферов. Обычно эту ситуацию можно обнаружить, анализируя после запуска нового ввода определенные биты состояния в интерфейсе аналого-цифрового преобразования. Если данные потеряны, а ERR_CODE загружается уникальный код, например 3. Процедура ERR_ROUT фиксирует наличие кода 3 в ERR_CODE и переходит к фрагменту, который печатает на терминале:

ДАННЫЕ ПРОПУЩЕНЫ В БЛОКЕ (*номер блока*).

На рис. 6.24 и 6.25 приведены подробные схемы процедур прерывания и INPUT. При возникновении прерывания из интерфейса вводится состояние подсистемы аналого-цифрового преобразования. Чтобы минимизировать вероятность потери данных, состояние проверяется не сразу, а включается в стек и контролируется после запуска следующей передачи ввода. При вызове процедур INPUT, PROCESS и OUTPUT предполагается, что в BX находится адрес BUFF_TAB, а в регистре SI – соответствующий индекс. Следовательно, процедура прерывания или основная программа должны загрузить эти регистры до вызова INPUT. Кроме того, перед вызовом INPUT проверяются индикатор следующего буфера для ввода и значение COUNT. Если индикатор установлен, в ERR_CODE помещается –1 и вызывается ERR_ROUT. Эта процедура печатает сообщение и сбрасывает COUNT в 0. Значение COUNT проверяется после индикатора, поэтому следующий ввод не запускается, если очередной буфер все еще обрабатывается или выводится или значение COUNT естественным образом достигло 0 (например, введено заданное число блоков). Если COUNT $\neq$ 0, вызывается INPUT и производится декремент COUNT. Затем команда STI разрешает прерывания и индикатор предыдущего ввода сбрасывается. В конце фрагмент завершения анализирует состояние предыдущего ввоца и выполняется команда IRET.

Процедура INPUT загружает содержимое BUFF_SIZE и адрес следующего буфера в счетчик и регистр адреса в регистровый набор 1 контроллера ПДП, а затем устанавливает бит "начать", инициируя передачу ввода. Затем она устанавливает индикатор буфера в 1, производит инкремент FLAGL, корректирует INDEX_I для адресации следующего буфера и загружает регистр SI, чтобы процедура прерывания могла использовать его для сброса индикатора только что заполненного буфера.

Чтобы избежать потери данных, объем обработки между моментом появления прерывания и установкой бита "начать" должен быть минимальным. За счет усложнения программы и ухудшения ее структуры управления можно сократить объем обработки, показанный на схемах. Но даже если фрагмент инициирования ввода поместить в самое начало процедуры прерывания, скорость дискретизации аналого-цифрового преобразователя будет все

же ограничиваться временем, расходуемым на последовательность прерывания и коммутацию буферов. Если вводить только в один буфер, исключив коммутацию буферов, скорость дискретизации будет в основном ограничиваться временем, требуемым на передачу данного по шине. Таким образом, временные соотношения играют очень важную роль и их необходимо тщательно проанализировать, когда в системе используются прерывания, блоковые передачи и многократное буферирование.

Рассмотренную программу можно приспособить для трех ситуаций:

1. Данные можно вводить в один буфер со скоростью, ограниченной продолжительностью цикла шины. Время на обработку и вывод может быть неограниченным.

2. Данные можно вводить в несколько (минимум три) буферов со скоростью, ограниченной временем коммутации буферов. Время на обработку и вывод может быть неограниченным.

3. Можно вводить любое число блоков со скоростью, ограниченной временем коммутации буферов. Время обработки блока и время вывода блока не должны превышать времени ввода блока.

Упражнения

1. Приведите блок-схему, показывающую, каким образом блок из N байт вводится в память с помощью программного ввода-вывода и блоковой передачи.

2. Приведите машинные коды следующих команд:

а) IN AL,52H б) OUT 0CH,AL

в) OUT DX,AX г) IN AX,DX

3. Напишите программу, которая попеременно проверяет регистры состояния двух устройств. Когда обнаруживается единичное состояние бита 0 регистра состояния, из соответствующего устройства вводится байт данных, а когда бит 3 любого регистра состояния находится в состоянии 1, процесс ввода прекращается. Регистры состояния должны иметь адреса портов 0024 и 0036, а соответствующие регистры входных данных – адреса 0026 и 0038. Входные данные помещаются в буферы, начинающиеся в BUFF1 и BUFF2.

4. Контроль паритета входных байт часто осуществляет интерфейс и при обнаружении ошибки паритета устанавливается бит состояния. Для примера на рис. 6.6 предположите, что интерфейс автоматически сбрасывает бит паритета, но, если входной байт имеет нечетный паритет, устанавливается бит 4 регистра состояния. Модифицируйте соответственно код на рис. 6.6.

5. Предположим, что строка, вводимая кодом на рис. 6.6, является 5-разрядным 16-ричным адресом и что компьютер должен реагировать выводом 16-ричного содержимого байта по этому адресу. Введите в пример код для производства необходимых преобразований и реализации вывода. Считайте, что входное и выходное преобразования, а также вывод реализуются отдельными процедурами типа NEAR.

6. Способ программных приоритетов в примере на рис. 6.7 оказывается не гибким, так как программа не может управлять приоритетами устройств. Предположим, что массив STAT_TAB из трех слов содержит адреса регистров состояния, а массив PROC_TAB из трех двойных слов – смещения и сегментные адреса процедур ввода. Порядок размещения адресов в этих массивах определяет приоритет. Перепишите соответствующим образом код на рис. 6.7.

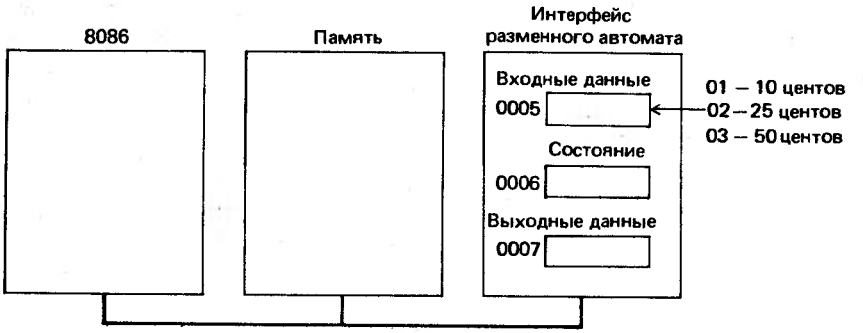


Рис. 6.26. Схема разменного автомата

7. Пользуясь приведенной на рис. 6.26 схемой разменного автомата, напишите программу его работы, пользуясь программным вводом-выводом. Программа ожидает в холостом цикле до обнаружения монеты, которая устанавливает бит 2 регистра состояния в порту 0006. Затем она вводит указанный код монеты, ожидает единичного состояния бита 3 (бит готовности) порта 0006 и выводит в порт 0007 требуемое число 5-центовых монет.

8. Повторите упр. 7, но теперь считайте, что в автомате имеется переключатель, который при включении устанавливает бит 6 регистра состояния. Если переключатель выключен, размен производится как и прежде. Если же он включен, 25-центовая монета устанавливает входной порт на 12_{16} и выходной порт на 21_{16} (две монеты по 10 центов и одна 5 центов); 50-центовая монета устанавливает входной порт на 13_{16} и выходной порт на 50_{16} (5 монет по 10 центов).

9. Повторите упр. 7, пользуясь вводом-выводом по прерываниям (т. е. опускание монеты генерирует прерывание). Основной программой может быть цикл "ничего-неделания". Тип прерывания равен 10.

10. Пусть 0132 является портом, получающим установки от устройства с двумя реле. Напишите процедуру прерывания, которая вводит содержимое порта в AL, а затем производит следующие действия:

а) печатает "НИЧЕГО НЕТ", если ни одно реле не замкнуто;
б) вызывает PROC_A, если реле А замкнуто, а реле В разомкнуто;
в) вызывает PROC_B, если реле А разомкнуто, а реле В замкнуто;
г) печатает "АВАРИЯ", если оба реле замкнуты.

Состояние реле А показывает бит 3 порта, а реле В – бит 4. Если бит находится в состоянии 0, соответствующее реле разомкнуто; в противном случае оно замкнуто. Бит готовности принтера является битом 4 порта 0096, а буферный регистр выходных даиных принтера имеет адрес порта 0098. Сначала для печати воспользуйтесь программным вводом-выводом, а затем модифицируйте код с введением второй процедуры прерывания.

11. Напишите процедуру прерывания, содержащую механизм программных приоритетов. Она обрабатывает до восьми устройств и имеет массив ADDR_TAB из восьми адресов. Элемент *i* массива дает смещение процедуры типа NEAR обслуживания *i*-го устройства. До проверки регистров состояния базовый адрес массива помещается в BX, а SI сбрасывается. Всякий раз, когда обнаруживается нулевое состояние бита готовности, производится инкремент SI на 2. Если же фиксируется бит готовности, содержа-

щий 1, осуществляется косвенный переход через регистры BX и SI к соответствующей процедуре обслуживания. Процедуры обслуживания программировать не нужно.

12. Предположим, что процедура прерывания типа 9 с начальным адресом, обозначенным INT_ROUT, включена в тот же самый исходный модуль, что и основная программа, т. е. программа, которая может прерываться. Напишите в основной программе фрагмент, который будет правильно инициализировать указатель прерываний.

13. Реализуйте код LINE_PROC, схема которой приведена на рис. 6.13, пользуясь ассемблерными командами микропроцессора 8086.

14. Пусть устройства 0, 1, 2, 3, 4 и 5 включены в приоритетную цепочку в указанном порядке и прерывания возникают так, как показано ниже. Определите порядок завершения процедур прерываний, которые должны начинаться командой STI, если

а) устройства 3 и 4 одновременно выдают запросы прерываний;

б) устройство 5 выдает запрос прерывания до окончания процедуры прерывания устройства 4;

в) устройство 2 посылает запрос прерывания после осуществления возврата в процедуру прерывания устройства 3, но до завершения этой процедуры;

г) пока выполняется процедура прерывания устройства 2, устройство 3 формирует новый запрос;

д) после возврата в основную программу устройства 1, 3 и 5 одновременно выдают запросы прерываний.

15. Повторите упр. 14, но теперь считайте, что команд STI в процедурах прерывания нет и что команды IRET разрешают прерывания при извлечении из стека PSW.

16. Повторите упр. 14 и 15, полагая, что приоритетная цепочка заменена системой управления приоритетными прерываниями, показанной на рис. 6.15. Сначала считайте, что регистр маски не блокирует никаких прерываний, а затем – что запросы от устройств 0, 1 и 4 блокируются до шага д) в упр. 14.

17. Пусть инициализируется блоковая передача 256 байт с ленточного накопителя в буфер с начальным адресом 12A0. Перечислите по шагам все действия, предпринимаемые при вводе первых трех байт. Зафиксируйте содержимое счетчика байт и регистров адреса после каждого шага.

18. Рассмотрим комбинацию интерфейс/контроллер ленточного накопителя со следующими определениями управления и состояния:

Регистр управления интерфейса (0032)
Бит 0. Бит "начать".
Бит 4. Установлен при вводе, сброшен при выводе.
Бит 6. Когда установлен, разрешает прерывания по завершению передачи.

Регистр состояния интерфейса (0033)
Бит 4. Установлен, когда накопитель занят.
Бит 5. Установлен, когда во время передачи обнаружена ошибка паритета.
Бит 6. Установлен, когда во время передачи обнаружен конец ленты.

Регистр управления ПДП (0034)
Бит 1. Установлен при вводе, сброшен при выводе.
Бит 2. Когда установлен, запросы ПДП разрешены.
Бит 3. Когда установлен, в интерфейс выдается прерывание при завершении.

Регистр состояния ПДП
Бит 5. Установлен, когда передача завершена.
Счетчик байт – 16 бит (0036).
Регистр адреса – 16 бит (0038).
Порт адреса – старшие 4 бита адреса (003A).

Напишите программу, которая инициирует передачу вывода 200 байт из буфера с начальным адресом MT_OUT в интерфейс накопителя таким образом, чтобы в конце передачи возникало прерывание. (Напомним, что передача не может начинаться, если накопитель занят.) Напишите также процедуру завершения, которая переходит к PAR_ERR при наличии ошибки паритета и к EOT_ROUT, если обнаружен конец ленты; в противном случае она просто помещает 0 в STAT_CODE и осуществляет возврат.

19. Реализуйте схему основной программы для примера тройного буферирования (рис. 6.22). Считайте, что COUNT и BUFF_SIZE вводятся процедурой TERM_IN. Бит 4 регистра управления ПДП с адресом 0064 управляет прерываниями по завершению аналого-цифрового преобразования, а бит 5 выполняет такую же функцию для прерываний диска. Эти биты устанавливаются, когда соответствующие прерывания разрешаются. Бит 2 регистра состояния интерфейса диска с адресом 0080 установлен, когда диск занят. Обработка завершения заключается в печати сообщения

ОПЕРАЦИЯ ЗАВЕРШЕНА

Все компоненты программы ассемблируются вместе, поэтому передача параметров не нужна, но при вызове INPUT, PROCESS или OUTPUT регистр BX должен содержать адрес BUFF_TAB, а регистр SI – индекс BUFF_TAB.

20. Предположим, что адреса счетчика байт и регистра адреса в регистровом наборе 1 контроллера ПДП равны 0048 и 004A, а бит 0 в регистре управления аналого-цифрового преобразователя с адресом 006C является битом "начать" для аналого-цифрового ввода. Напишите процедуру INPUT, схема которой приведена на рис. 6.25.

21. Предположите, что в примере тройного буферирования из § 6.5 обработка заключается в замене каждого байта данных разностью между ним и предыдущим байтом данных, и напишите процедуру PROCESS. Для первого байта данных в первом блоке предыдущий байт возьмите равным 0. Когда разность выходит за диапазон –128 . . . 127, значением переменной ERR_CODE становится 2 и вызывается ERR_ROUT.

7. ВВЕДЕНИЕ В МУЛЬТИПРОГРАММИРОВАНИЕ

Программная единица, которая выполняет независимую задачу, называется *процессом*. Если процессы выполняются последовательно, система называется однопрограммной. Обычно в однопрограммных системах в любой момент времени в памяти находится только один процесс, а следующий процесс не загружается для выполнения, пока не завершается текущий процесс. Такая ситуация не подходит для большого числа применений, особенно если в них требуется реакция на события в реальном времени или быстрое и эффективное выполнение многочисленных программ. Предположим, например, что микросистема должна воспринимать и обрабатывать данные от двух независимых устройств сбора данных. При работе в однопрограммной среде любой процесс может пропустить часть входных данных в то время, когда выполняется другой процесс. Это может случиться, даже когда скорость поступления данных в обоих процессах невелика. Проблема вызывается не недостатком вычислительной мощности или быстродействием интерфейсов, а последовательным характером всего метода обработки. Кроме того, в средних и больших системах последовательное выполнение множества процессов ведет

к временным потерям, особенно если в системе применяется высокопроизводительный микропроцессор.

В *мультипрограммной* среде в памяти одновременно находятся коды двух и более процессов, которые выполняются с мультиплексированием во времени. Если в приведенном примере процессы выполняются в мультипрограммной системе, они попеременно используют ЦП и один процесс может выполнять свои вычисления, пока другой осуществляет ввод-вывод, и наоборот. Если скорость передачи данных имеет приемлемое значение, процессы можно коммутировать с помощью прерываний и успешно выполнять оба процесса.

Хотя мультипрограммирование означает разделение ЦП более чем одним процессом, единственный ЦП в любой момент времени может выполнять только одну команду. Однако благодаря разделению ресурсов мультипрограммная однопроцессорная система выглядит так, как будто она выполняет несколько процессов одновременно. Даже в системах широкого назначения, не связанных с обработкой данных в реальном времени, целесообразно применять мультипрограммирование, так как оно позволяет значительно повысить производительность системы путем совмещения операций ввода-вывода и действий ЦП.

Производительность системы часто измеряется числом заданий, выполненных за временной интервал; эта величина называется *пропускной способностью системы*. Чтобы показать, каким образом совмещение обработки и ввода-вывода в режиме ПДП может улучшить пропускную способность системы, рассмотрим два процесса. Типичные действия однопрограммной системы показаны на рис. 7.1, *а*. Процесс 1 начинается и продолжается до точки A, где ему потребовался ввод-вывод. Здесь инициируется ввод-вывод, а обработка продолжается параллельно с ним до тех пор, пока для обработки не потребовались входные данные (точка B); ЦП должен ожидать завершения ввода-вывода. Обработка возобновляется, когда ввод-вывод закончен (точка C). Аналогичная ситуация возникает в точках D, E и F. По окончании процесса 1 начинается процесс 2; он выполняется так же, как и процесс 1.

Действия в мультипрограммной среде показаны на рис. 7.1, *б*. Из него видно, что вместо простоя ЦП в ожидании ввода-вывода процесса 1 в мультипрограммной системе начинается процесс 2, который использует ЦП до тех пор, пока ему не потребуется ожидать ввода-вывода. Если процесс 1 закончил свои операции ввода-вывода, он может в этот момент возобновить использование ЦП. Таким образом общее время обработки значительно сокращается.

Кроме "перемешивания" нескольких действий ввода-вывода, мультипрограммная система может одновременно обслуживать нескольких пользователей. Пользователи поочередно управляют ЦП, когда каждому из них выделяется *временной квант* – получается *система с разделением времени*. Обычно к таким системам подключаются многочисленные терминалы и каждый пользователь взаимодействует с системой так, как будто она находится в его полном распоряжении (хотя слишком много пользователей вызывают заметные временные задержки).

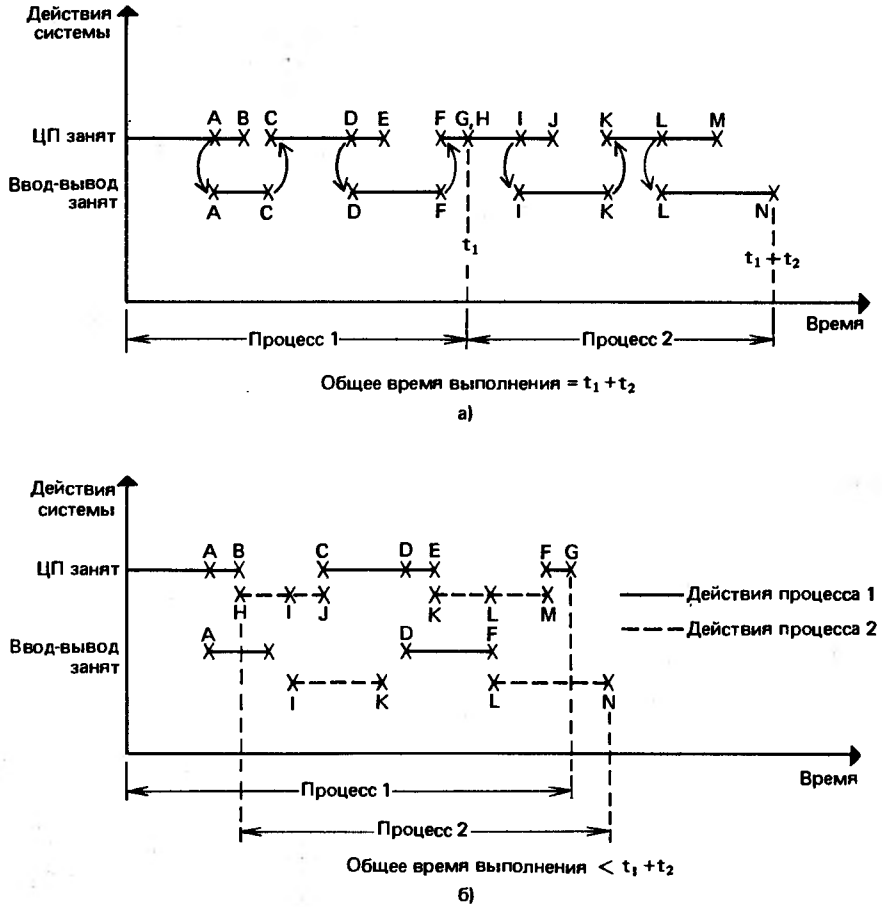


Рис. 7.1. Производительность однопроцессорной однопрограммной (*а*) и мультипрограммной (*б*) системы

Дальнейшее расширение мультипрограммирования заключается в переходе к системам, содержащим более одного процессора. В таких *мультипроцессорных системах* в каждый момент времени выполняется несколько команд (см. гл. 11).

В настоящей главе рассматриваются основные программные вопросы, характерные для мультипрограммной однопроцессорной системы. Хотя глава считается введением в мультипрограммирование, некоторые важные моменты и их реализация в системах на базе микропроцессоров 8086/8088 обсуждаются достаточно подробно. Управление процессами рассматривается в § 7.1, доступ к разделенным данным – в § 7.2 и разделение процедур – в § 7.3. Последние два параграфа касаются управления памятью и виртуальной памяти.

7.1. УПРАВЛЕНИЕ ПРОЦЕССАМИ

В памяти однопроцессорной мультипрограммной системы одновременно находятся несколько процессов, которые разделяют ЦП, но сам ЦП в любой момент времени может выполнять только один процесс. В простой мультипрограммной системе процессы могут находиться в трех состояниях, причем каждый процесс в любой момент времени находится только в одном из этих состояний. Указанными состояниями являются:

1. Выполнение (счет, прогон). Процесс выполняется центральным процессором.

2. Блокировка. Выполнение процесса продолжать невоэможно, так как он ожидает появления некоторого события, например завершения операции ввода-вывода.

3. Готовность. Выполнение процесса можно возобновить в любой момент времени. Например, ввод-вывод, завершения которого ожидал процесс, закончен и обработку можно продолжить.

Смена состояний каждого процесса во времени показана на рис. 7.2. При запуске ("старте") процесс помещается в очередь процессов, находящихся в состоянии готовности. Если текущий выполняемый процесс переходит из состояния выполнения в состояние блокировки из-за необходимости ожидания ввода-вывода, ЦП освобождается и планировщик процессов выбирает из очереди процесс и изменяет его состояние готовности на состояние выполнения. До тех пор, пока процесс находится в состоянии блокировки, его выполнение приостанавливается. После завершения ввода-вывода процесс переходит в состояние готовности и помещается в соответствующую очередь.

В наиболее простом механизме мультипрограммирования процесс выполняется до своего завершения или необходимости ожидать ввода-вывода. Однако для предотвращения монопольного использования процессора одним процессом систему можно спроектировать так, чтобы выполняемый процесс переходил в состояние готовности по истечении определенного временного

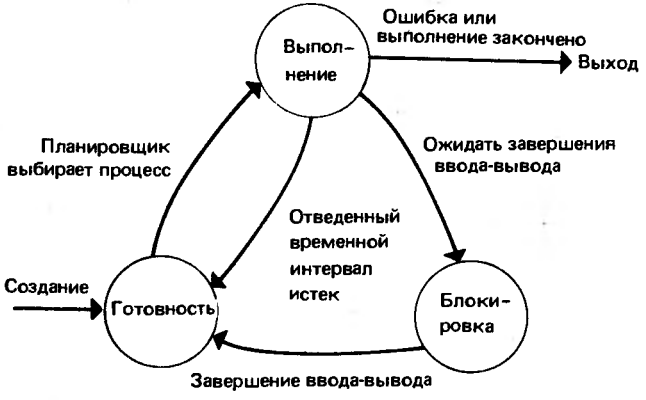


Рис. 7.2. Состояния процесса и их изменения

интервала, что обеспечивает другим процессам шансы на выполнение. Такими системами с разделением времени управляют внешние программируемые устройства синхронизации, называемые *часами* (*таймерами*) *реального времени* (см. § 9.3). В системе с разделением времени время измеряется от момента перехода процесса в состояние выполнения; часы реального времени формируют прерывание, когда отведенное процессу время (т. е. временной квант) исчерпано. После этого процедура прерывания, являющаяся частью резидентного монитора, инициирует изменения состояний процессов.

Реализация очереди процессов, находящихся в состоянии готовности, зависит от стратегии планирования. Простейшая стратегия "первый пришел/первый ушел" (FIFO) назначает всем процессам одинаковые приоритеты и всегда выбирает для выполнения процесс, который находится в очереди дольше всех. По существу процессы помещаются в низ ("хвост") очереди и берутся из ее вершины ("головы"). Структура списка готовности, опирающаяся на планирование FIFO и связанный список, представлена на рис. 7.3. Указатель первого элемента показывает, какой процесс находится в вершине очереди, а указатель последнего элемента – процесс в низу очереди. Указатель последнего элемента необходим для того, чтобы в очередь можно было помещать новые процессы без просмотра всего списка. В примере предполагается, что каждый процесс имеет ID (код идентификации или идентификатор), который совпадает с позицией в массиве, содержащем связанный список. Предполагается также, что очередь состоит из процессов 3, 6, 2, 8 и 5 (именно в таком порядке). Следовательно, указатель первого элемента содержит 3, указатель последнего элемента содержит 5, а процессы 2, 3, 5, 6 и 8 находятся в состоянии готовности. Остальные элементы в таблице процессов либо заняты выполняемыми или заблокированными процессами, либо в данное время не используются. Если число 0 применяется для указания конца списка, его нельзя употреблять в качестве ID процесса. Если ID процессов представлены одним байтом и не может быть процесса 0, то максимальное число процессов, которые могут находиться в системе одновременно, равно 255.

Каждый элемент в списке содержит прямой указатель очереди (*fqp*), который указывает на следующий элемент в списке, и указатель управляющего блока процесса для локализации блока памяти, который хранит информацию, относящуюся к процессу. Управляющий блок процесса служит областью запоминания процесса для хранения состояния машины, а также ID процесса, состояния процесса, кода, показывающего, как и почему процесс переведен в текущее состояние, и т. д.

Когда ЦП переключается с одного выполняющегося процесса на другой, монитор должен:

1. Запомнить машинное состояние выполняемого процесса в управляющем блоке процесса.

2. Модифицировать остальную часть управляющего блока процесса.

3. Взять ID следующего выполняемого процесса из указателя первого элемента.

4. Удалить процесс, только что переведенный в состояние выполнения, посредством установки указателя первого элемента равным *fqp* удаляемого

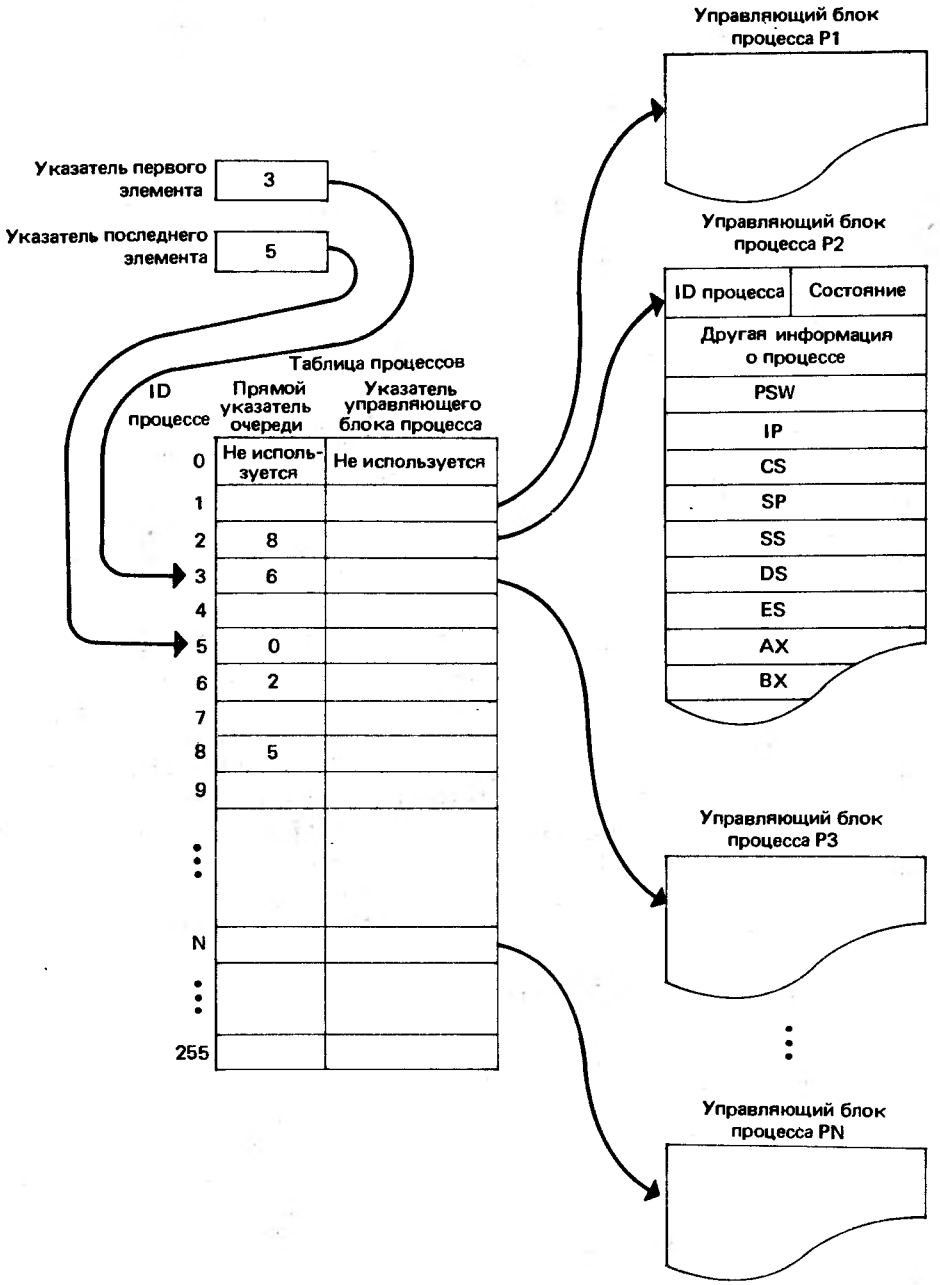


Рис. 7.3. Структура очереди процессов, находящихся в состоянии готовности

элемента. Если текущая выполняемая программа должна быть в очереди процессов, находящихся в состоянии готовности, ее необходимо поместить в низ очереди и модифицировать указатель последнего элемента.

5. Изменить состояние процесса, только что выбранного из списка готовности, на состояние выполнения и восстановить машинное состояние данного процесса.

В результате шага 5 новый выбранный процесс будет продолжаться с той точки, в которой он был приостановлен. Процесс, находящийся в системе, можно поместить в список готовности, реализуя следующие действия:

1. Сохранить ID этого процесса в *fqp* текущего последнего элемента (элемента, адресуемого указателем последнего элемента) и в указатель последнего элемента.

2. Сбросить *fqp* в элементе таблицы данного процесса.

3. Модифицировать управляющий блок этого процесса.

Очень часто, особенно при обработке данных в реальном времени, необходимо назначать различным процессам приоритеты, причем процессу, требующему самого быстрого обслуживания, назначается наивысший приоритет. Нескольким процессам можно назначить один и тот же приоритет и на каждом приоритетном уровне использовать стратегию FIFO. При этом в системе потребуется приоритетная таблица. Как показано на рис. 7.4, каждый элемент в этой таблице представляет собой приоритетный уровень и имеет два поля, одно из которых содержит указатель первого элемента приоритетного уровня, а другое – последнего. При изменении приоритета процесс необходимо удалить из его приоритетной цепочки и добавить в низ цепочки, имеющей его новый приоритет. Данное действие упрощается при наличии обратного указателя очереди (*bqp*), который ускоряет изменения приоритетов. В системе приоритетного планирования на рис. 7.4 предполагается, что в состоянии готовности находятся следующие процессы:

Приоритет 0: нет
Приоритет 1: 9, 4, 1, 3
Приоритет 2: 2, 7, 5
Приоритет 3: 10
Приоритет 4: нет
Приоритет 5: нет

Для выбора следующего выполняемого процесса планировщик должен просматривать приоритетную таблицу, начиная с наивысшего приоритета, до обнаружения ненулевого указателя первого элемента, а затем удалить этот процесс из очереди готовности.

Преимущество использования *bqp* заключается в возможности динамического изменения приоритета данного процесса. Предположим, что приоритет процесса i необходимо изменить на приоритет x. Сначала процесс i удаляется из его текущей приоритетной цепочки с помощью следующих операций:

$$fqp(bqp(i)) \leftarrow fqp(i),$$
$$bqp(fqp(i)) \leftarrow bqp(i).$$

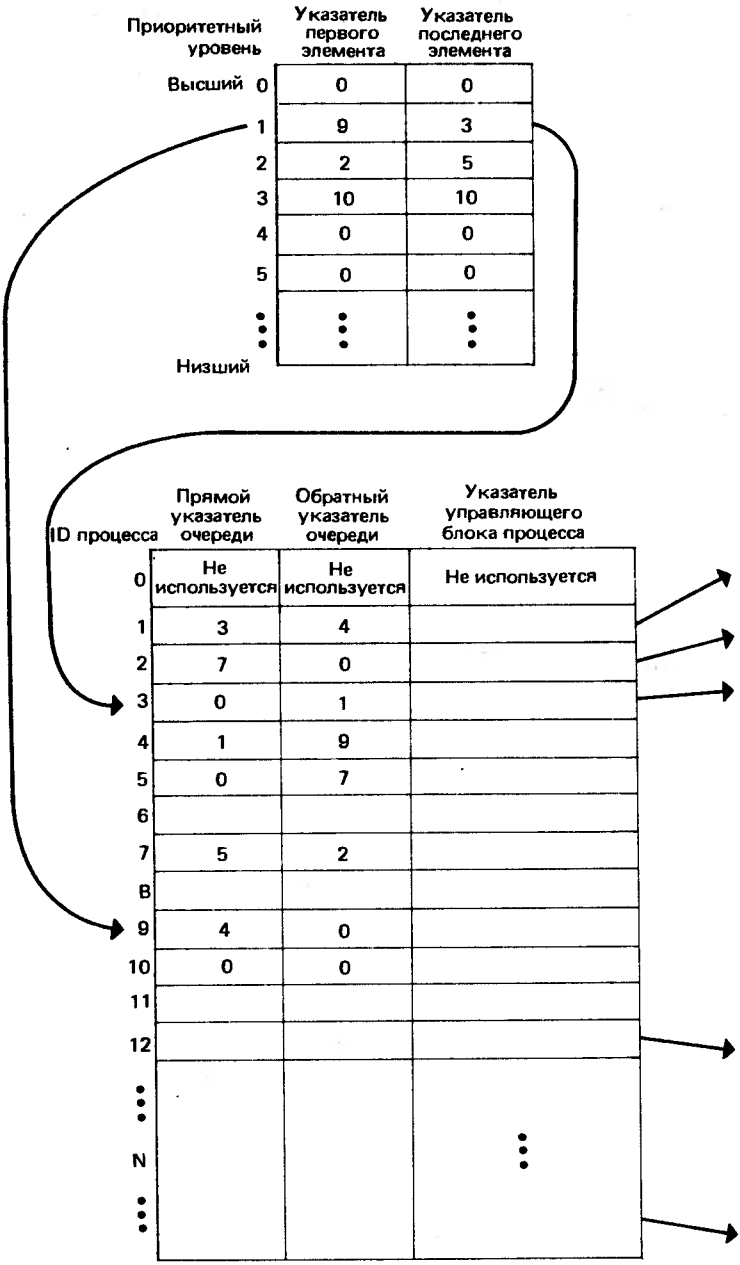


Рис. 7.4. Структура приоритетной очереди процессов, находящихся в состоянии готовности

Конечно, если процесс *i* является верхним или нижним элементом в его текущей приоритетной цепочке, потребуется дополнительная коррекция приоритетной таблицы. Затем процесс *i* добавляется в низ цепочки, соответствующей приоритетному уровню *x*. Это сопровождается модификацией указателя последнего элемента (а также указателя первого элемента, если цепочка была пустой), *fqp* текущего последнего элемента в цепочке, а также *fqp* и *bqp* добавляемого элемента.

Некоторые мультипрограммные операционные системы оказываются более сложными, чем показанная на рис. 7.2, и допускают более трех основных состояний процессов. Обычно в этом случае дополнительные состояния представляют собой другие формы заблокированного состояния. Рассмотрим в качестве примера операционную систему iRMX 86 фирмы Intel, состояния процессов в которой показаны на рис. 7.5. В ней имеются три заблокированных состояния: пассивное, приостановленное и пассивно-приостановленное. Система iRMX 86 относится к управляемым только событиями, а единственная временная зависимость заключается в том, что процесс может быть переведен в пассивное состояние на определенное время. Изменения состояний вызываются программными приказами, которые называются *системными вызовами* и реализуются с помощью программных прерываний или автоматически после возникновения определенных событий. Возможные изменения показаны стрелками и дугами, причем номера стрелок на рис. 7.5 соответствуют номерам следующих действий:

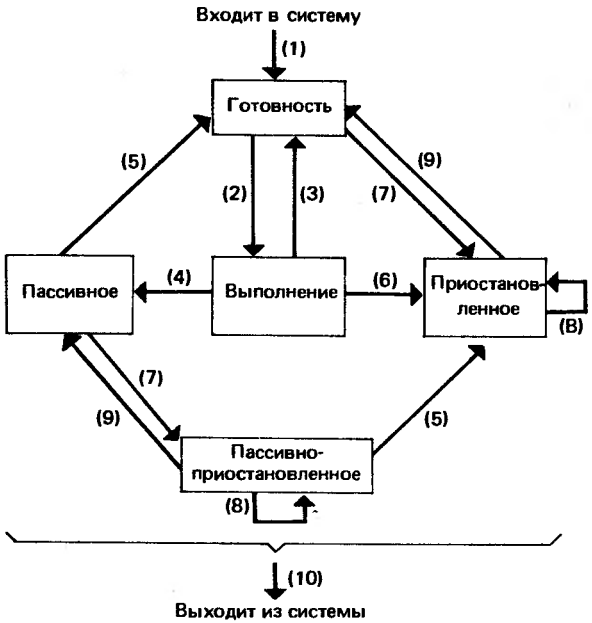


Рис. 7.5. Состояния процессов в операционной системе iRMX 86

1. Процесс входит в систему, когда он создается системным вызовом CREATE TASK.

2. Процесс переводится из состояния готовности в состояние выполнения, когда:

его приоритет выше приоритета выполняемого процесса;

выполняемый процесс переводится в пассивное или приостановленное состояние, если приоритет процесса в состоянии готовности столь же высок, как и приоритет любого другого процесса в состоянии готовности, но сам процесс находился в очереди готовности дольше любого процесса с равным приоритетом.

3. Процесс переводится из состояния выполнения в состояние готовности, когда выполняемый процесс должен уступить ЦП процессу с большим приоритетом, находящимся в состоянии готовности.

4. Процесс переводится из состояния выполнения в пассивное состояние, если выполняемый процесс встречает системный вызов SLEEP или должен ожидать информации, которую он запросил, но которая пока недоступна. В первом случае временно́е ограничение указывается в вызове SLEEP, а во втором процесс должен выразить готовность ожидать запрошенную информацию (через системный вызов).

5. Процесс переходит из пассивного состояния в состояние готовности или из пассивно-приостановленного состояния в приостановленное, когда истекает интервал ожидания, указанный в вызове SLEEP, или когда удовлетворяется запрос процесса.

6. Процесс переходит из состояния выполнения в приостановленное состояние, когда выполняемый процесс приостанавливает себя, выполняя системный вызов SUSPEND TASK. Кроме того, глубина приостановки для процесса устанавливается равной 1.

7. Процесс переводится из состояния готовности в приостановленное состояние или из пассивного состояния в пассивно-приостановленное, когда другой процесс определяет его в системном вызове SUSPEND TASK. Кроме того, его глубина приостановки устанавливается равной 1.

8. Состояние приостановленного или пассивно-приостановленного процесса не изменяется, но производится инкремент его глубины приостановки, когда выполняемый процесс реализует вызов SUSPEND TASK, определяющий процесс. Аналогично осуществляется декремент глубины приостановки приостановленного процесса, когда реализуется вызов RESUME TASK.

9. Приостановленный (пассивно-приостановленный) процесс переводится в состояние готовности (в пассивное состояние), если его глубина приостановки становится равной 0.

10. Процесс выходит из системы, если он указан в системном вызове DELETE TASK.

Отметим, что в iRMX 86 приоритеты имеют не только процессы, находящиеся в состоянии готовности. Процессам, находящимся в приостановленном или пассивно-приостановленном состоянии, также назначается разновидность приоритетов в соответствии с их глубинами приостановки.

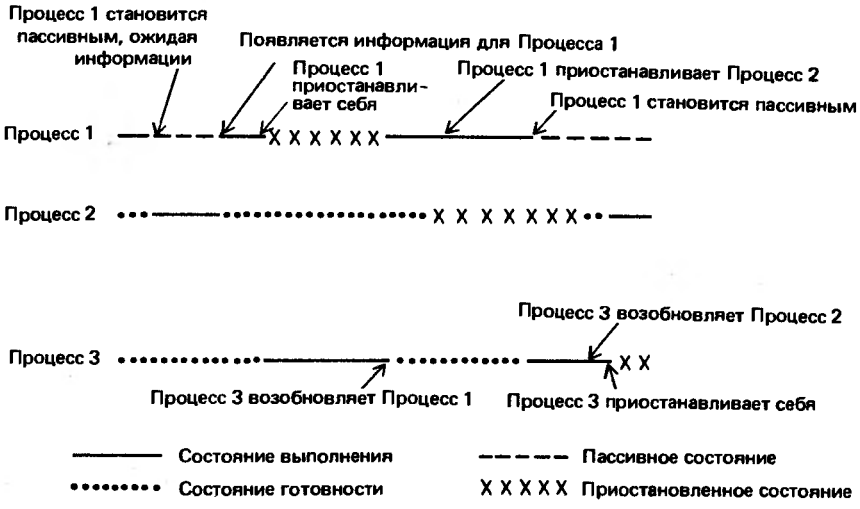


Рис. 7.6. Типичные действия под управлением операционной системы iRMX 86: —— состояние выполнения; . . . состояние готовности; – – – пассивное состояние; xxx – приостановленное состояние

На рис. 7.6 показано выполнение процессов 1, 2 и 3 под управлением операционной системы iRMX 86. Предполагается, что все три процесса находятся в системе, причем процесс 1 имеет наивысший приоритет, а процессы 2 и 3 имеют одинаковые, но меньшие приоритеты. Предполагается также, что первоначально процесс 2 находился в состоянии готовности дольше процесса 3. Сплошные линии показывают, какой процесс находится в состоянии выполнения.

Как уже упоминалось, системные вызовы iRMX 86 реализованы с помощью программных прерываний и параметры передаются в вызванную процедуру прерывания и из нее через стек. Фрагмент для удаления задачи имеет следующий вид.

```
PUSH   TASK_NO
PUSH   EXCPT_PTR
MOV    BP,SP
LEA    SI,SS:[BP + 2]
MOV    AX,201H
INT    184
  .
  .
  .
```

Здесь 201H и 184 представляют собой соответственно код входа и тип прерывания, соответствующие системному вызову DELETE TASK. Число 184 обозначает выполняемую процедуру прерывания, а код входа показывает этой процедуре, что необходимо произвести удаление. Предполагается, что

TASK_NO содержит номер удаляемого процесса, а EXCPT_PTR – указатель процедуры, которая выполняется, если при удалении возник особый случай. Процедура прерывания для доступа к стеку использует регистры BP и SI.

Хотя системные вызовы iRMX 86 можно реализовать на языке ассемблера, фирма Intel встроила системные вызовы в языки высокого уровня PL/M и Паскаль. Чтобы упростить использование iRMX 86, разработана микросхема 80130, которая содержит значительную часть необходимых аппаратных средств и постоянное запоминающее устройство емкостью 16К байт, хранящее код системных вызовов. Эта микросхема рассматривается подробнее в гл. 12.

7.2. СЕМАФОРНЫЕ ОПЕРАЦИИ

В мультипрограммных системах процессам разрешается разделять общие программные, аппаратные и информационные ресурсы. Во многих ситуациях одновременно обращаться к общему ресурсу и модифицировать его может только один процесс, а другие процессы должны ожидать завершения его операций. Такой ресурс, обычно называемый *последовательно используемым*, должен быть защищен от одновременного доступа и модификации двумя или более процессами. Ресурсом этого типа может быть аппаратный ресурс (принтер, карточный считыватель, ленточный накопитель), файл данных или разделенная область памяти.

Рассмотрим, например, файл персонала, который разделяется процессами 1 и 2. Предположим, что процесс 1 выполняет введения, удаления и изменения, а процесс 2 упорядочивает файл в алфавитном порядке по фамилиям. При последовательном доступе файл либо модифицируется процессом 1, а затем сортируется процессом 2, либо наоборот. Однако, если разрешить обоим процессам одновременный доступ к файлу, результаты окажутся непредсказуемыми и почти наверняка неправильными. Решение данной задачи заключается в том, чтобы разрешить одновременно только одному процессу выполнять его *критическую секцию кода*, т. е. секцию кода, которая осуществляет доступ к последовательно используемому ресурсу.

Предотвращение ситуации, когда два и более процессов одновременно выполняют их критические секции при доступе к разделенному ресурсу, называется *взаимным исключением*. Один из способов реализации взаимного исключения – использовать флажки, показывающие, когда разделенный ресурс уже используется. Чтобы пояснить этот способ и возникающие здесь вопросы, рассмотрим возможность использования только одного флажка. Если имеется один флажок и FLAG = 1 означает, что ресурс свободен, а FLAG = 0 показывает, что он занят, то доступ к ресурсу реализуется следующим образом:

```
                                ПРОЦЕСС 1

P1:             .
                .
                .
TRYAGAIN1:      TEST    FLAG,1
                JZ      TRYAGAIN1
                MOV     FLAG,0
```

```
                .
                .
                .}      КРИТИЧЕСКАЯ СЕКЦИЯ В ПРОЦЕССЕ 1
                MOV     FLAG,1
                .
                .
                .

                        ПРОЦЕСС 2

P2:             .
                .
                .
TRYAGAIN2:      TEST    FLAG,1
                JZ      TRYAGAIN2
                MOV     FLAG,0
                .
                .}      КРИТИЧЕСКАЯ СЕКЦИЯ В ПРОЦЕССЕ 2
                .
                MOV     FLAG,1
                .
                .
                .
```

Здесь первые команды MOV сбрасывают флажок в 0 при попытке предотвратить доступ к ресурсу другого процесса. Последние команды MOV возвращают флажок в 1 после завершения доступа, так что другой процесс может вновь обратиться к ресурсу.

Сложность данного подхода заключается в том, что имеется возможность выполнения критических секций обоими процессами. Для иллюстрации этой возможности предположим, что FLAG = 1 и происходит такая последовательность событий:

1. Выполняется команда TEST в процессе 1.
2. До выполнения команды JZ в процессе 1 происходит переключение на процесс 2.
3. Выполняется команда TEST в процессе 1.

Очевидно, в этой ситуации оба процесса могут выполнять их критические секции кода.

Во избежание этого можно использовать два флажка: один показывает нахождение в критической секции процесса 1, а второй – процесса 2. До того как каждый процесс сможет выполнить свою критическую секцию, он должен вначале сбросить свой флажок, показывая свое намерение, а затем повторно проверять флажок другого процесса до тех пор, пока он не станет единицей. Этот прием реализуется посредством кодирования обоих процессов следующим образом:

```
P1:             .
                .
                .
                MOV     FLAG1,0
CHECK1:         TEST    FLAG2,1
                JZ      CHECK1
                .
                .}      КРИТИЧЕСКАЯ СЕКЦИЯ
                MOV     FLAG1,1
                .
                .
                .
```

Хотя это решение гарантирует взаимное исключение, оно порождает еще одну трудность, называемую *взаимной блокировкой* (или *тупиком*). Если

переключение с процесса 1 на процесс 2 происходит между командами MOV и TEST, то процесс 2 может сбросить FLAG2 и оба флажка оказываются в состоянии 0. Так как теперь ни одна команда TEST не дает удовлетворительного результата, ни один процесс не может перейти к своей критической секции. Оба процесса "зависают" в своих циклах ожидания.

Для такого решения, которое обеспечивает взаимное исключение и предотвращает взаимную блокировку, потребуется третий флажок, который показывает, какой процесс имеет больший приоритет при возникновении взаимной блокировки. Впервые это решение было предложено С. Деккером и обсуждалось Е. Дейкстра. Последний обобщил решение на случай *n* процессов.

В микропроцессорах 8086/8088 простое решение проблемы обеспечивается командой обмена XCHG. Необходимое управление доступом к ресурсу получается при использовании одного флажка и структурировании процессов следующим образом:

```
P1:             .
                .
                .
                MOV     AL,0
TRYAGAIN1:      XCHG    AL,FLAG
                TEST    AL,AL
                JZ      TRYAGAIN1
                .
                .       КРИТИЧЕСКАЯ СЕКЦИЯ
                .
                MOV     FLAG,1
                .
                .
                .
```

Если FLAG = 1 при выполнении команды XCHG, он сбрасывается, а в AL оказывается 1. Даже если система переключает процессы после команды XCHG, новый процесс не сможет войти в свою критическую секцию, так как FLAG уже сброшен. Такое решение стало возможным благодаря загрузке и установке операнда одной командой (именно командой XCHG) и может быть применено к любому числу процессов.

Флажок, используемый для резервирования разделенного ресурса, называется *семафором*, а операции запроса и освобождения ресурса обычно называются *семафорными операторами* P и V. Как было показано, оператор P легко реализуется командами MOV, XCHG, TEST и JZ, а оператор V — командой MOV. Однако в этой реализации, когда один процесс находится в критической секции, другие процессы, претендующие на тот же самый ресурс, будут, по существу, простаивать в циклах ожидания операторов P. Эта ситуация приводит к значительным потерям времени ЦП. Для лучшего использования времени ЦП оператор P следует модифицировать следующим образом:

```
AGAIN:          MOV     AL,0
                XCHG    AL,SEMAPHORE
                TEST    AL,AL
                JNZ     NEXT
                .
                .       ВЫЗВАТЬ МОНИТОР, ТАК ЧТОБЫ
                .       ОН ПЕРЕВЕЛ ТЕКУЩИЙ ПРОЦЕСС
                .       В ЗАБЛОКИРОВАННОЕ СОСТОЯНИЕ
                JMP     SHORT AGAIN
NEXT:           .
                .       КРИТИЧЕСКАЯ СЕКЦИЯ
                .
```

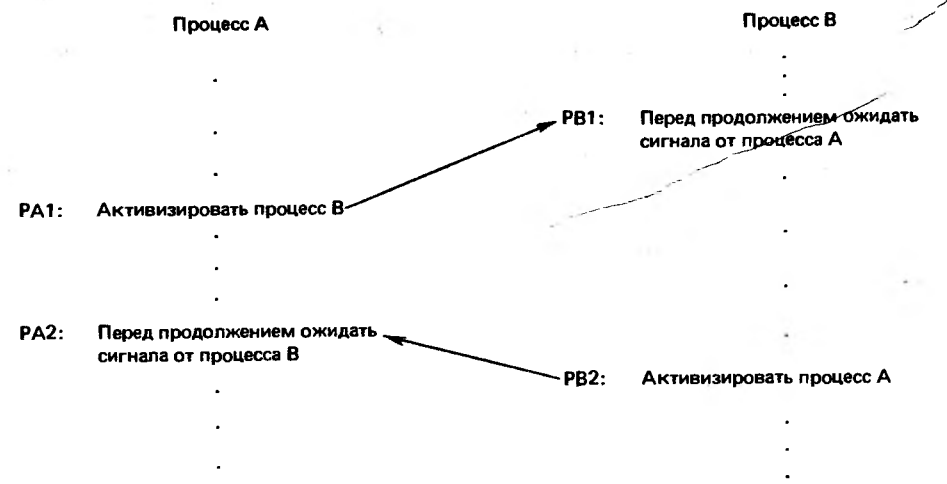


Рис. 7.7. Синхронизация двух процессов

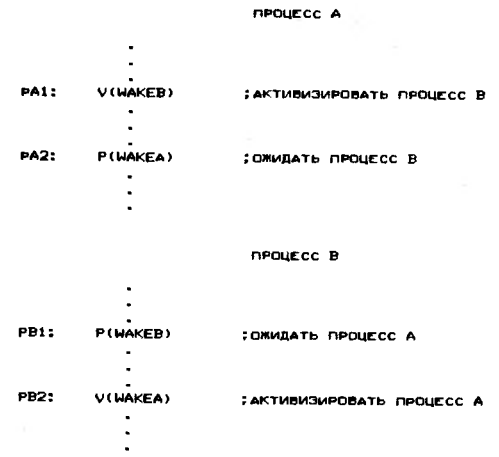
Если семафор сброшен, вместо повторения цикла ожидания монитор переводит текущий процесс в заблокированное состояние, а ЦП начинает выполнять выбранный процесс, находящийся в состоянии готовности. Аналогично следует модифицировать и оператор V:

```
MOV  SEMAPHORE,1
.        Вызвать монитор так, чтобы он
.        разблокировал те процессы, которые
.        остановлены из-зе семафора
```

Данная конструкция устанавливает семафор в 1 и позволяет монитору переслать те процессы, которые были заблокированы из-за недоступности разделенного ресурса, из списка заблокированных процессов в список процессов, находящихся в состоянии готовности. Первый из этих процессов, который возобновляет выполнение, сможет войти в свою критическую секцию.

Рассмотренный метод не только экономит значительное время процессора, но и может применяться для синхронизации двух процессов или для передачи сообщения от одного процесса другому. На рис. 7.7 процессы А и В должны взаимодействовать в определенных точках. Они могут выполняться в режиме с разделением времени до тех пор, пока процесс В не достигает точки PB1, где он должен ожидать сообщения или обращаться к результатам, полученным процессом А. После точки PA1 оба процесса вновь выполняются в режиме разделения времени до достижения точки PA2. В этой точке процесс А должен получать сообщение от процесса В или синхронизироваться с ним.

Оператор P можно использовать для блокирования процесса, а оператор V – для активизации процесса. В приведенном примере два семафора синхронизации WAKEA и WAKEB должны иметь начальные значения 0. После этого синхронизация процессов реализуется следующим образом:



7.3. РАЗДЕЛЕНИЕ ОБЩИХ ПРОЦЕДУР

В мультипрограммной системе желательно, чтобы несколько пользователей могли разделять процедуры. Такими процедурами, называемыми *общими*, обычно являются библиотечные или системные программы. например драйверы ввода-вывода, процедуры преобразований кодов и т. п. Например, несколько пользователей могут одновременно пожелать редактировать свои программы с помощью редактора текста. Если не разделять код, в память придется загрузить несколько копий редактора текста. При разделении кода потребуется иметь в памяти только одну копию, что дает значительную экономию объема памяти.

Общую процедуру можно разделять последовательно, как и любой другой разделенный ресурс. Обычным образом перед вызовом общей процедуры вызывающая процедура должна выполнить оператор P, а после выхода из нее — оператор V, чтобы сделать разделенную процедуру доступной другим пользователям. При последовательном разделении главное дополнительное требование заключается в том, что все локальные переменные в процедуре должны реинициализироваться, чтобы не использовались их значения от предыдущего вызова. Следовательно, последовательно разделяемая процедура должна реинициализировать сама себя в начале каждого вызова.

Применение последовательно разделяемых процедур довольно ограничено, так как требуется более универсальная форма разделения, т. е. форма, в которой процедура разделяется с мультиплексированием во времени (становится параллельно используемой). Эта форма означает: процедура должна быть такой, чтобы ее смог вызвать другой процесс до завершения выполнения процедуры от предыдущего вызова. Такая процедура называется *реентрантной* (*реентерабельной, повторно вызываемой*). Реентрантная процедура должна состоять из кода, называемого *чистым*, который не модифицирует сам себя. Для удовлетворения этого требования реентрантная процедура мо-

жет хранить модифицируемые данные только в ячейках памяти, которые ассоциируются с вызывающим процессом; она не может даже временно хранить такие данные в ячейках, являющихся для нее локальными. Для иллюстрации предположим, что TEMP является локальной переменной в реентрантной процедуре и что она используется для хранения промежуточного результата, и рассмотрим такую последовательность событий:

1. Процесс 1 выполняется и вызывает реентрантную процедуру.
2. Реентрантная процедура помещает промежуточный результат в TEMP.
3. Процесс 2 получает управление процессором.
4. Процесс 2 вызывает реентрантную процедуру.
5. Реентрантная процедура вновь помещает промежуточный результат в TEMP.

Очевидно, в этой точке первоначальный промежуточный результат разрушается, следовательно, при возврате в процесс 1 и возобновлении реентрантной процедуры процесс 1 сформирует неправильный результат.

Чтобы решить данную задачу, все результаты, включая и содержимое регистров, должны храниться в ячейках, ассоциированных с вызывающим процессом. В приведенном примере два промежуточных результата необходимо поместить в отдельные ячейки, одна из которых ассоциирована с про-

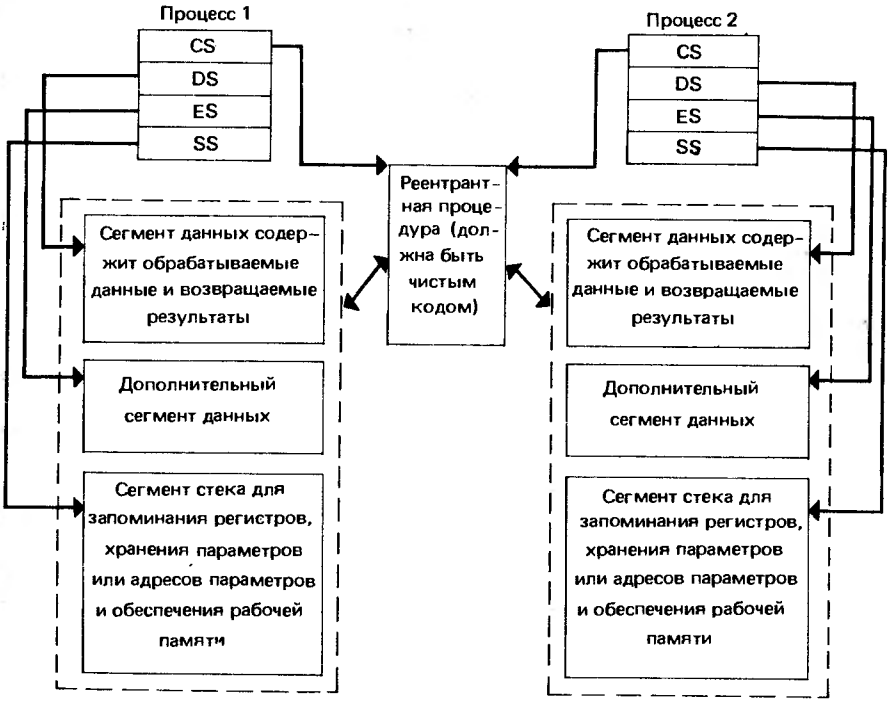


Рис. 7.8. Реентрантный код, разделяемый несколькими процессами

цессом 1, а другая – с процессом 2. Рассматриваемая задача аналогична обсуждавшейся в разделе 4.3.4, но в рекурсивной процедуре новые ячейки для хранения промежуточных результатов распределяются для каждого последовательного вызова. Для хранения многих копий промежуточных результатов по принципу LIFO применялся один и тот же стек, так как рекурсивные вызовы сопровождаются возвратами в обратном порядке. Здесь же информация запоминается в вызывающем процессе.

В процессорах, имеющих стек, обычный способ хранения промежуточных результатов заключается в том, чтобы ассоциировать стек с каждым процессом. На рис. 7.8 показано, как два процесса обращаются к реентрантной процедуре. Когда процесс 1 вызывает реентрантную процедуру, содержимое регистров SS и SP не изменяется и реентрантная процедура запоминает любые свои результаты в стеке процесса 1. Когда происходит переключение процессов, содержимое SS и SP изменяется с тем, чтобы адресовать вершину стека, ассоциированного с процессом 2. Когда вновь вызывается реентрантная процедура, она будет запоминать свои результаты в стеке, ассоциированном с процессом 2, а предыдущие результаты не искажаются.

Так как переключение между процедурами модифицирует и содержимое регистров DS и ES, эти регистры также можно использовать для хранения информации вызывающего процесса. По существу, возможно реализовать любую приемлемую связь подпрограмм с единственным требованием, что никакие данные нельзя хранить локально в реентрантной процедуре.

```
FRAME       STRUC
    TEMPZ       DW      ?,?
    TEMPY       DW      ?,?
    TEMPX       DW      ?,?
    SAVE_BP     DW      ?
    SAVE_CS_IP  DW      ?,?
    Z_ADDRESS   DW      ?
    Y_ADDRESS   DW      ?
    X_ADDRESS   DW      ?
FRAME       ENDS

MULTIPLY    PROC        FAR
            PUSH    BP                  ;ЗАПОМНИТЬ BP
            SUB     SP,12               ;ЗАРЕЗЕРВИРОВАТЬ 6 СЛОВ ДЛЯ
                                        ;РАБОЧЕЙ ОБЛАСТИ
            MOV     BP,SP               ;BP АДРЕСУЕТ КАДР
            PUSH    AX
            PUSH    BX
            PUSH    CX
            PUSH    DX
            PUSH    SI
            PUSH    DI
            MOV     SI,0                ;SI СЛУЖИТ ИНДИКАТОРОМ ЗНАКА
            MOV     BX,[BP].X_ADDRESS   ;ПЕРЕДАТЬ ОПЕРАНД X В TEMPX
            MOV     AX,[BX]
            MOV     [BP].TEMPX,AX
            MOV     AX,[BX+2]
            MOV     [BP].TEMPX+2,AX
            MOV     BX,[BP].Y_ADDRESS   ;ПЕРЕДАТЬ ОПЕРАНД Y В TEMPY
            MOV     AX,[BX]
            MOV     [BP].TEMPY,AX
            MOV     AX,[BX+2]
            MOV     [BP].TEMPY+2,AX
            CMP     [BP].TEMPX+2,0      ;X ПОЛОЖИТЕЛЬНЫЙ ?
            JGE     CHECK_Y             ;ДА, ПРОВЕРИТЬ Y
            NOT     [BP].TEMPX          ;ИНАЧЕ ОБРАЗОВАТЬ
            NOT     [BP].TEMPX+2        ;ДОПОЛНИТЕЛЬНЫЙ КОД X
            ADD     [BP].TEMPX,1
            ADC     [BP].TEMPX+2,0
            MOV     SI,1                ;ОТРИЦАТЕЛЬНЫЙ X
```

Рис. 7.9. Реентрантная процедура умножения знаковых 32-битных операндов

```
CHECK_Y:    CMP     [BP].TEMPY+2,0          ;Y ПОЛОЖИТЕЛЬНЫЙ ?
            JGE     START                   ;ДА, НАЧАТЬ УМНОЖЕНИЕ
            NOT     [BP].TEMPY              ;ИНАЧЕ ОБРАЗОВАТЬ
            NOT     [BP].TEMPY+2            ;ДОПОЛНИТЕЛЬНЫЙ КОД Y
            ADD     [BP].TEMPY,1
            ADC     [BP].TEMPY+2,0
            XOR     SI,1                    ;В SI ЗНАК ПРОИЗВЕДЕНИЯ
START:      MOV     AX,[BP].TEMPX           ;ЭТИ КОМАНДЫ ВЫПОЛНЯЮТ
            MUL     [BP].TEMPY              ;БЕЗЗНАКОВОЕ УМНОЖЕНИЕ
            MOV     [BP].TEMPZ,AX           ;С ДВОЙНОЙ ТОЧНОСТЬЮ И
            MOV     CX,DX                   ;ЗАПОМИНАЮТ МЛАДШИЕ ДВА
            MOV     AX,[BP].TEMPX+2         ;СЛОВА ПРОИЗВЕДЕНИЯ В TEMPZ
            MUL     [BP].TEMPY              ;И СТАРШИЕ ДВА СЛОВА
            MOV     BX,DX                   ;ПРОИЗВЕДЕНИЯ В DX:AX
            ADD     CX,AX
            ADC     BX,0
            MOV     AX,[BP].TEMPX
            MUL     [BP].TEMPY+2
            ADD     CX,AX
            ADC     BX,DX
            MOV     [BP].TEMPZ+2,CX
            MOV     CX,0
            ADC     CX,0
            MOV     AX,[BP].TEMPX+2
            MUL     [BP].TEMPY+2
            ADD     AX,BX
            ADC     DX,CX
            MOV     CX,[BP].TEMPZ+2
            CMP     SI,0                    ;ПРОВЕРИТЬ ЗНАК И ИЗМЕНИТЬ
            JE      STORE_Z                 ;ЗНАК ПРОИЗВЕДЕНИЯ, ЕСЛИ
            NOT     [BP].TEMPZ              ;ЗНАК ОТРИЦАТЕЛЬНЫЙ
            NOT     CX
            NOT     AX
            NOT     DX
            ADD     [BP].TEMPZ,1
            ADC     CX,0
            ADC     AX,0
            ADC     DX,0
STORE_Z:    MOV     BX,[BP].ADDRESS_Z       ;ЗАПОМНИТЬ ПРОИЗВЕДЕНИЕ
            MOV     [BX+6],DX               ;В ЯЧЕЙКЕ, АДРЕСУЕМОЙ
            MOV     [BX+4],AX               ;ADDRESS_Z
            MOV     [BX+2],CX
            MOV     AX,[BP].TEMPZ
            MOV     [BX],AX
            POP     DI                      ;ВОССТАНОВИТЬ РЕГИСТРЫ
            POP     SI
            POP     DX
            POP     CX
            POP     BX
            POP     AX
            ADD     SP,12
            POP     BP
            RET     6                       ;ВОЗВРАТ
MULTIPLY    ENDP
```

Для иллюстрации приведенных положений рассмотрим реентрантную процедуру, которая умножает два знаковых целых 32-битных числа и возвращает 64-битный результат. Обычно знаковое умножение заключается в умножении абсолютных значений операндов и преобразовании произведения в дополнительный код. Чтобы сохранить сомножители неизменными, процедура должна временно запомнить их абсолютные значения. Необходимо также временно запомнить часть беззнакового произведения. Реентрантная процедура умножения с двойной точностью приведена на рис. 7.9. В стеке для временного хранения распределены шесть слов: два для части произведения и по два для каждого операнда. Для распределения и доступа к этим словам из SP вычитается 12 и результат загружается в BP; последующие обращения осуществляются относительно BP.

Так как стек не является частью процедуры (см. рис. 7.8), процедура ничего не записывает в саму себя. Следовательно, главное требование реентрантного кода удовлетворяется и процедуру можно разделять с мультиплек-

сированием во времени. Если эту процедуру вызывает другой процесс до завершения текущего вызова, в операции переключения система должна запомнить SS и загрузить в SS новый сегментный адрес. Поэтому при вызове новым процессом реентрантной процедуры для хранения TEMPX, TEMPY и TEMPZ используется другая область и промежуточные данные приостановленного процесса не модифицируются.

После выполнения умножения осуществляется возврат в вызывающий процесс. Для этого восстанавливается содержимое всех регистров кроме BP, к SP прибавляется 12, восстанавливается BP и выполняется команда RET 6. Операнд 6 в команде возврата предназначен для компенсации включенных в стек вызывающим процессом смещений сомножителей X и Y и произведения Z.

Чистый код в реентрантной процедуре должен быть еще и позиционно-независимым, т. е. он может правильно выполняться независимо от его размещения в физической памяти или относительно вызывающих его процессов. Во всех локальных обращениях должна применяться какая-либо разновидность косвенной адресации или код должен корректировать свои локальные адреса в соответствии с текущим размещением. Лучшим методом образования позиционно-независимого кода оказывается относительная адресация, которая в микропроцессорах 8086/8088 реализуется, главным образом, с помощью сегментных регистров. Требование позиционной независимости реентрантных процедур объясняется тем, что они разделяются несколькими процессами и обычно во время выполнения динамически связываются с различными процедурами. Такая связь упрощается, если адреса в реентрантном коде модифицировать не нужно.

Рис. 7.10. Распределение памяти разделами

7.4. УПРАВЛЕНИЕ ПАМЯТЬЮ

Совместный набор программных единиц, которые объединены друг с другом, но не обращаются к единицам вне набора, кроме обращений через внешнюю память, называется *заданием* (или *программой*). Чтобы эффективно использовать системные ресурсы и достичь максимальной параллельности работы ЦП и ввода-вывода, желательно хранить в памяти максимальное число заданий. Следовательно, при проектировании мультипрограммной системы важное значение приобретает управление памятью так, чтобы свести к минимуму неиспользуемое пространство памяти.

Простейший способ хранения в памяти нескольких заданий заключается в *распределении разделов*. Как показано на рис. 7.10, в этом способе каждому заданию назначается смежная область памяти. Когда необходимо выполнить задание, загрузчик запрашивает необходимый объем памяти у *процедуры управления памятью*, которая является частью

операционной системы, отвечающей за распределение памяти. Если эта процедура находит свободную смежную область, размер которой больше размера задания, она возвращает загрузчику начальный адрес этой свободной области. После коррекции необходимых относительных адресов, такой, что они превращаются в физические адреса, загрузчик помещает задание в распределенную область. Конечно, начальным состоянием задания является "готовность" и допускается, что задание начинает выполняться не сразу. Если же достаточной смежной области нет, задание должно ожидать до тех пор, когда его будет можно загрузить из внешней памяти.

Способ распределения разделов требует организации в процедуре управления памятью так называемой таблицы карты памяти. Возможная реализация этой таблицы заключается в том, чтобы каждому разделу соответствовал элемент таблицы. Каждый элемент таблицы должен содержать состояние, размер и начальный адрес раздела; раздел может быть в таких состояниях:

Распределен. Раздел в данное время распределен заданию.

Свободен. Раздел доступен для использования.

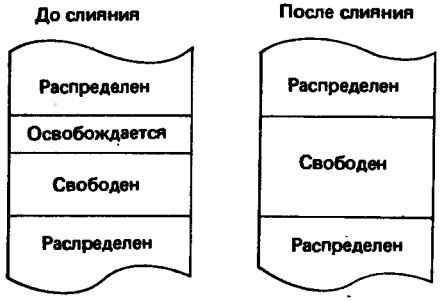
Не используется. Элемент не ассоциирован с разделом.

Третье состояние может возникнуть в том случае, когда два соседних свободных раздела объединены в один.

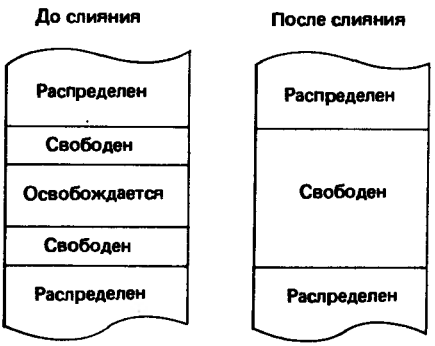
Получив запрос на распределение памяти, процедура управления памятью проверяет размер каждого свободного раздела, начиная с верха таблицы. Если обнаруживается свободный раздел, размер которого равен или больше запрошенного раздела, его состояние изменяется на "распределен" и возвращается его начальный адрес. В том случае, если запрошенный объем памяти меньше объема раздела, для учета лишнего пространства в таблицу добавляется новый свободный раздел. Данный алгоритм выбора свободного раздела называется *алгоритмом первого соответствия*. В другом алгоритме – *алгоритме наилучшего соответствия* – просматривается вся таблица для нахождения наименьшего свободного раздела, удовлетворяющего запросу.

Когда задание завершено, занятая им область возвращается в систему. Процедура управления памятью модифицирует таблицу карты памяти, чтобы отразить возможное слияние соседних свободных разделов, появляющихся при освобождении памяти. Две возможные ситуации, вызывающие слияние, представлены на рис. 7.11. В случае 1 освобожденный раздел оказывается соседним с имеющейся свободной областью. После слияния два свободных раздела объединены в один большой раздел, начальный адрес которого равен меньшему из двух первоначальных свободных адресов. В случае 2 две имеющиеся свободные области разделены освобожденным разделом. В результате слияния три раздела объединяются в один. Слияние необходимо для того, чтобы иметь максимальные размеры свободных разделов.

Основные достоинства способа распределения разделами – простота и отсутствие специальных аппаратных средств. Однако по мере завершения заданий и загрузки новых заданий начинают появляться все меньшие и меньшие области памяти – возникает *проблема фрагментации* (рис. 7.12, *а*). При большом числе таких небольших свободных областей система не сможет загру-



а) Случай 1



б) Случай 2

Рис. 7.11. Слияние соседних свободных разделов

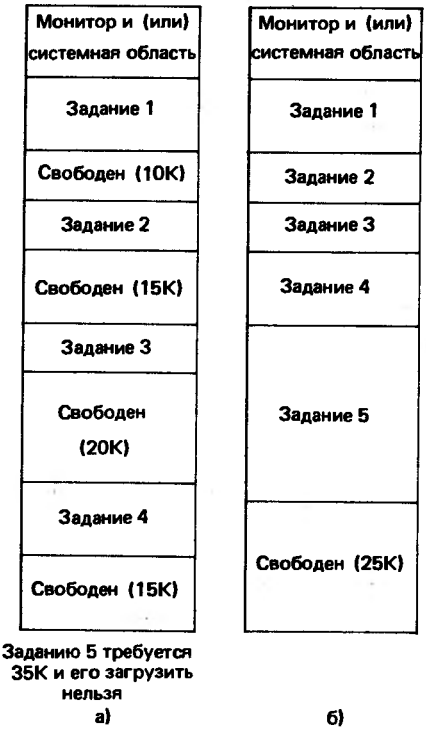


Рис. 7.12. Фрагментация памяти (*а*) и процесс уплотнения и загрузки задания 5 (*б*)

зить задание, даже если общий объем доступного пространства достаточно велик; иначе говоря, значительная часть памяти не используется.

Один из способов решения проблемы фрагментации заключается в том, чтобы объединить все свободные области в одну, "сжимая" или "уплотняя" находящиеся в памяти задания, как показано на рис. 7.12, *б*. Так реализуется способ *распределения разделов с перемещением.* Но для его применения компьютер должен иметь возможность корректировать каждое задание так, чтобы после его перемещения из одной области в другую можно было правильно возобновить выполнение задания.

В микропроцессорах 8086/8088 все физические адреса формируются сложением эффективных и сегментных адресов, умноженных на 16, причем сегментные адреса берутся из сегментных регистров. Такой способ позволяет легко преобразовать программу в позиционно-независимый код и пересылать ее из одной области памяти в другую без модификации машинного кода. При выполнении перемещения монитор должен скорректировать сегментные адреса (т. е. содержимое сегментных регистров).

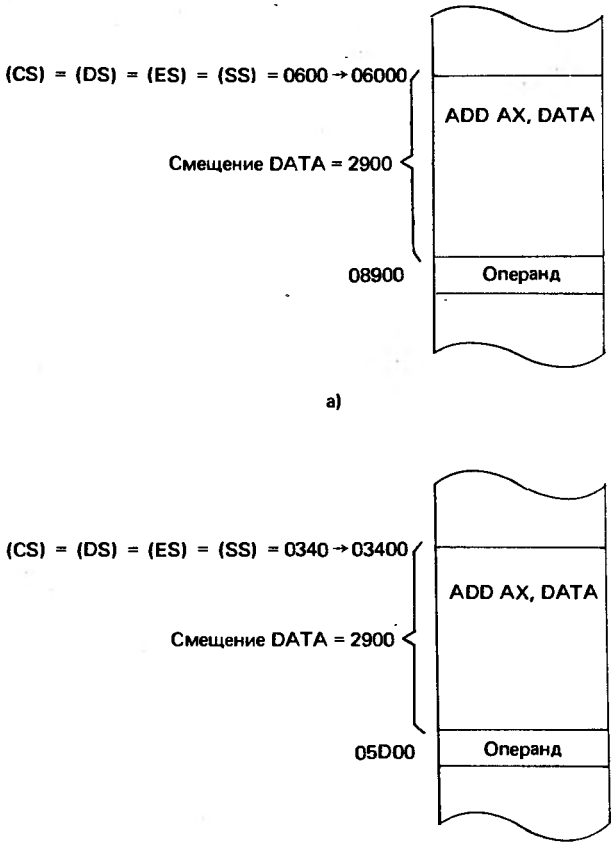


Рис. 7.13. Содержимое сегментного регистра до (*а*) и после (*б*) перемещения

На рис. 7.13 показана реализация перемещения с помощью сегментных регистров. В этом примере задание состоит из одного сегмента и, следовательно, все адреса берутся относительно начала этого сегмента. Когда программа перемещается с адреса 06000 в адрес 03400, монитор должен изменить только сегментные регистры CS, DS, ES и SS, загрузив в них 0340. Отметим, что до перемещения команда ADD обращается к DATA по адресу 06000 + 2900 = = 08900, а после перемещения она правильно обращается к DATA по адресу 03400 + 2900 = 05D00.

Чтобы обеспечить позиционную независимость, сегментными регистрами должна управлять только процедура управления памятью, а программа не должна их модифицировать (как это было в приведенных ранее примерах). Кроме того, все вызовы процедур и переходы должны иметь тип NEAR. В противном случае, если CS был запомнен вызовом до перемещения,

а возврат осуществляется после перемещения, CS будет содержать старый сегментный адрес и программа возвратится в неправильное место. Так как переместимой программе не разрешается модифицировать сегментные регистры, задание может иметь максимум четыре сегмента (кода, данных, дополнительных данных и стека) и максимальный размер задания ограничен 4 x 64К = 256К.

Так как при уплотнении операционная система должна пересылать задания из одной области памяти в другую, на уплотнение расходуется много времени и без необходимости его выполнять не следует. В способе распределения разделов с перемещением обычным образом процедура управления памятью распределяет пространство и организует таблицу карты памяти, как и в способе распределения разделов. Однако, если нельзя найти одну свободную область больше запрошенного размера, операционная система должна определить, достаточно ли всего свободного объема памяти. Если это так, инициируется процесс уплотнения, в котором корректируются базовые адреса всех заданий (они определяются запомненным содержимым сегментных регистров) и соответственно модифицируется таблица карты памяти.

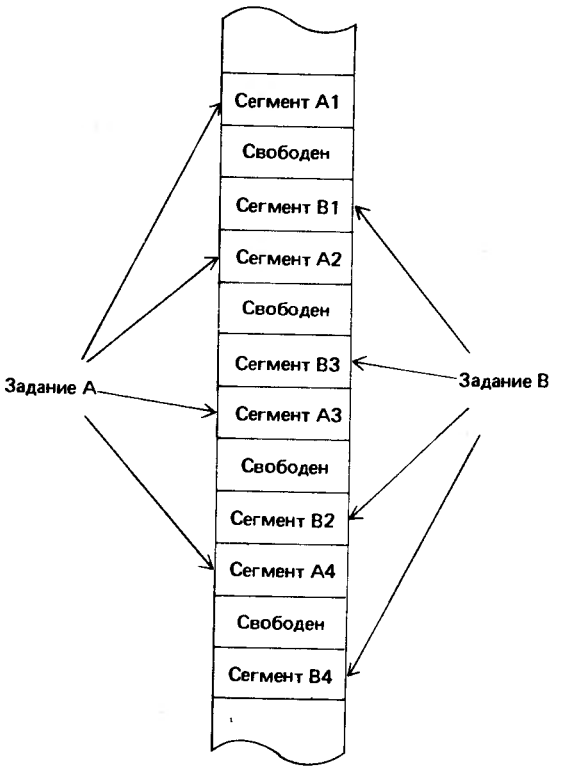


Рис. 7.14. Устранение фрагментации при использовании для каждого задания нескольких сегментов

Еще один подход к решению проблемы фрагментации заключается в том, чтобы разделить программу на несколько частей. Как показано на рис. 7.14, части каждого задания хранятся в отдельных областях, которые не обязательно будут физически смежными, но которые логически оказываются соседними. В микропроцессорах 8086/8088 задание можно разделить на несколько сегментов, длина каждого из которых варьируется от 16 байт до 64К байт. Куда загружается каждый сегмент, зависит от обстановки, которая сложилась во время выполнения. Для упрощения коррекции адресов, которая потребуется в процессе загрузки задания, лучше всего загружать сегментные регистры из таблицы, содержащей местоположения всех сегментов задания. При этом все межсегментные передачи управления необходимо выполнять командами косвенных переходов и вызовов, чтобы получать адреса переходов из таблицы.

Возможная реализация связана с построением единой таблицы указателей, содержащей набор указателей для каждого задания. Таблица указателей должна хранить местоположения сегментов кода, данных и стека соответствующего им задания. Регистр ES резервируется для обращения к таблице указателей. Межсегментная передача управления при этом реализуется косвенными переходом или вызовом через таблицу указателей с использованием сегментного адреса из ES. Для обращения к новому сегменту данных при загрузке регистра DS из таблицы применяется команда LDS. Данный способ позволяет операционной системе загружать каждый сегмент в отдельную область памяти, а процедура управления памятью должна будет модифицировать только таблицу указателей.

7.5. ВИРТУАЛЬНАЯ ПАМЯТЬ

Более сложное управление памятью обеспечивается аппаратным динамическим преобразованием адресов, показанным на рис. 7.15. В каждом обращении к памяти выдаваемый ЦП *логический адрес* преобразуется в *физи-*

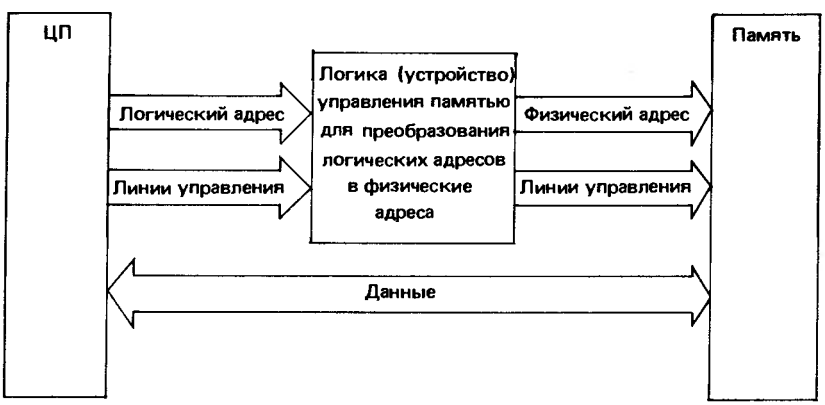


Рис. 7.15. Динамическое преобразование адреса

ческий, который схема управления памятью посылает в память. Логическими называются адреса, которые генерируют команды в соответствии с режимом адресации.

Так как логические адреса могут отличаться от физических, пользователь разрабатывает программу в логическом пространстве, называемом также *виртуальным*, не учитывая ограничений реальной памяти. При этом обеспечиваются два основных преимущества. Во-первых, при разделении программы на несколько частей, каждая из которых отображается на область реальной памяти, программа не обязательно должна занимать смежный блок памяти, что уменьшает фрагментацию памяти и пространство, расходуемое впустую. Во-вторых, при выполнении не обязательно хранить в памяти всю программу. При обращении к той части программы, которой в данный момент в памяти нет, операционная система может приостановить программу, загрузить требуемую секцию кода и возобновить выполнение программы. Следовательно, становятся допустимыми программы, размер которых больше объема имеющейся памяти. Другими словами, пользователю кажется, что он работает с большей памятью, чем на самом деле.

Программа разделяется на сегменты в соответствии со своей логической структурой. Такой способ управления памятью со схемой преобразования адресов называется *сегментацией*. В этом способе каждый логический адрес состоит из двух полей: номера сегмента и смещения в сегменте. Число бит в номере сегмента определяет максимальное число допустимых в программе сегментов, а число бит в смещении – максимальный размер сегмента. Если, например, номер сегмента и смещение имеют соответственно m и n бит, программа может состоять из 2^m сегментов, а максимальный размер сегмента равен 2^n байт, что обеспечивает пользователю виртуальную память 2^{m+n} байт.

Рис. 7.16 показывает отображение логических адресов в физические. Номер сегмента S в логическом адресе служит индексом сегментной таблицы, которая возвращает начальный физический адрес требуемого сегмента. Этот адрес суммируется со смещением L, образуя физический адрес ячейки памяти. Так как каждому заданию назначается отдельная область в сегментной таблице, базовый адрес той секции сегментной таблицы, которая ассоциируется с текущим выполняемым заданием, находится в регистре, называемом *регистром сегментной таблицы*. Индекс S берется относительно регистра сегментной таблицы. Поскольку преобразование адреса должно выполняться при каждом обращении к памяти, либо вся сегментная таблица, либо ее часть, содержащая начальные адреса сегментов выполняемого задания, должна храниться в регистрах, входящих в состав схемы управления памятью.

Каждый элемент в сегментной таблице называется *дескриптором сегмента*. В зависимости от конкретной реализации дескриптор сегмента кроме начального адреса сегмента содержит некоторые атрибуты сегмента. Наиболее часто применяются следующие атрибуты.

Поле состояния. Показывает, находится в памяти или нет сегмент, к которому производится обращение. Для различения двух возможных ситуаций достаточно одного бита. Если требуемый сегмент в памяти отсутствует, воз-

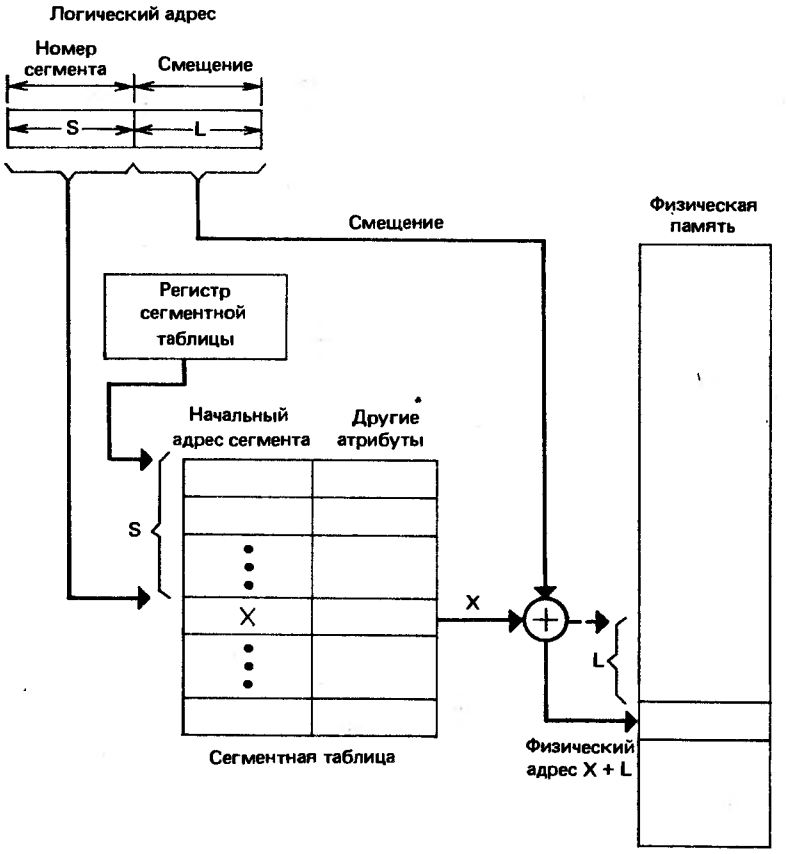


Рис. 7.16. Способ сегментации

никает сегментное нарушение, инициирующее вызов монитора. Монитор загружает нужный сегмент в память, модифицирует сегментную таблицу и возобновляет выполнение программы. Когда достаточного пространства в памяти нет, один или несколько сегментов передаются во внешнюю память.

Поле длины сегмента. Показывает размер сегмента. Эта информация предотвращает обращения к другим сегментам из-за недействительного адреса. Когда смещение больше длины сегмента, генерируется прерывание с вызовом монитора.

Поле защиты. Обеспечивает защиту от несанкционированных считывания, записи или выполнения. Обычно это поле содержит 3 бита, которые показывают следующие возможности по обращению к сегменту:

Считываемый: содержимое сегмента можно считывать как операнды.

Записываемый: содержимое сегмента допускается модифицировать.

Выполнимый: содержимое сегмента можно выбирать как команды.

Каждому сегменту, как логической единице программы, можно придавать несколько ограничений по доступу посредством установки комбинаций этих бит.

Поле доступа. Обеспечивает полезную информацию о том, как определить сегмент, подлежащий замене. Когда из внешней памяти вызывается новый сегмент, для увеличения доступного пространства может потребоваться удалить один из находящихся в памяти сегментов. Были предложены и исследованы такие алгоритмы замены, как FIFO, наименее часто используемый (LFU) и наиболее давно используемый (LRU). Однако, за исключением алгоритма FIFO, их реализации требуют сложных аппаратных средств. Альтернативой служат биты доступа, ассоциируемые с каждым сегментом. Когда к сегменту производится обращение, его бит доступа автоматически устанавливается в 1. После того, как все биты доступа установлены в 1, монитор может сбросить их в 0. Анализируя биты доступа, монитор выбирает сегмент, к которому давно не было обращений.

Поле изменения. Показывает, модифицировался сегмент или нет после передачи его в память. При замене сегмента, если он был изменен, необходимо передать его во внешнюю память перед загрузкой нового сегмента. Если же сегмент не модифицировался, монитор должен только загрузить новый сегмент из внешней памяти, что экономит операцию записи.

При обращении к памяти типичная последовательность операций, выполняемых схемой управления памятью, выглядит таким образом:

1. Определить номер сегмента и обратиться к соответствующему элементу сегментной таблицы.

2. Проверить состояние. Если в памяти нужного сегмента нет, генерировать прерывание в монитор, который инициирует процедуру обслуживания сегментного нарушения.

3. Сравнить смещение и длину сегмента. Если смещение слишком велико, генерировать прерывание в монитор.

4. Проверить тип доступа. При попытке несанкционированного обращения, генерировать прерывание в монитор.

5. Прибавить начальный адрес сегмента к смещению для образования физического адреса и выдать результат на шину адреса.

6. Модифицировать поля доступа и изменения, а затем ожидать следующего обращения к памяти.

Процедура обслуживания сегментного нарушения обычно реализует следующие действия:

1. Проверить таблицу карты памяти. Если требуемый сегмент помещается в свободную область памяти, перейти к шагу 4; иначе перейти к шагу 2.

2. Выбрать удаляемый сегмент, анализируя биты доступа, и модифицировать состояние выбранного сегмента.

3. Проверить бит изменения заменяемого сегмента и, если этот сегмент был модифицирован, записать его во внешнюю память.

4. Загрузить требуемый сегмент в память и модифицировать карту памяти и сегментную таблицу.

5. Возвратиться к команде, которая вызвала сегментное нарушение.

Прерывания, которые возникают во время преобразования адреса, но не связаны с сегментным нарушением, обычно вызывают прекращение текущего задания.

Фирма Intel в микросхеме 80286 объединила на одном кристалле усовершенствованный вариант микропроцессора 8086 и логику сегментного управления памятью, аналогичного описанному. Кроме средств управления памятью, микропроцессор 80286 имеет надмножество системы команд микропроцессора 8086 и работает в шесть раз быстрее. Микропроцессор 80286 может выполнять программы микропроцессоров 8086/8088 после повторного ассемблирования без изменений или с минимальными модификациями.

Микропроцессор 80286 имеет те же четыре сегментных регистра CS, DS, ES и SS и тот же набор общих регистров, что и микропроцессор 8086, но содержит несколько других регистров, предназначенных для мультипрограммирования.

Как сегментные регистры используются для формирования физического адреса, зависит от режима работы процессора. Имеется два режима работы: реального адреса и защищенного виртуального адреса (или просто защищенный режим). В первом режиме сегментные регистры выполняют те же функции, что и в микропроцессоре 8086.

Во втором режиме содержимое сегментного регистра используется как селектор сегмента, показывающий, где в дескрипторной таблице хранится дескриптор сегмента. Аппаратные средства оперируют с одной дескрипторной таблицей прерываний, одной глобальной дескрипторной таблицей и неограниченным числом локальных дескрипторных таблиц. Каждая задача имеет свою локальную дескрипторную таблицу, которая, следовательно, защищена от других задач, и глобальную дескрипторную таблицу для разделенных сегментов. Эти таблицы могут находиться по любым адресам памяти и адресуются

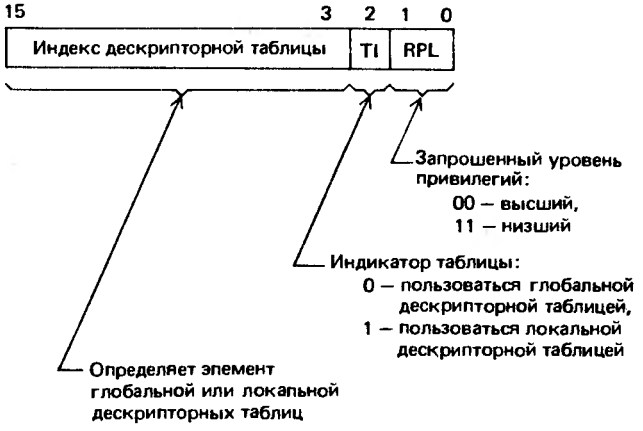


Рис. 7.17. Формат сегментного регистра микропроцессора 80286 в защищенном режиме

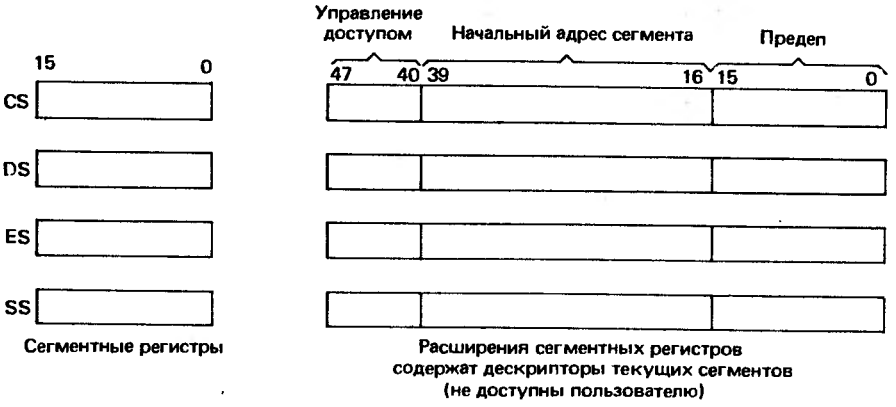


Рис. 7.18. Сегментные регистры и их расширения

регистром дескрипторной таблицы прерываний, регистром глобальной дескрипторной таблицы и регистром локальной дескрипторной таблицы. Регистры находятся в микропроцессоре 80286 и должны загружаться операционной системой с помощью специальных команд. Кроме того, с каждым сегментным регистром связаны 48-битные динамические модифицируемые регистры расширения, предназначенные для хранения дескрипторов текущих сегментов кода, данных, дополнительных данных и стека.

На рис. 7.17 показаны определения полей сегментных регистров, а на рис. 7.18 – формат их расширений. Бит 2 сегментного регистра определяет, где находится дескриптор сегмента: в глобальной или локальной дескрипторных таблицах. Биты 1 – 0 задают уровень привилегий сегмента; их можно использовать для защиты сегмента от обращений задач с меньшими уровнями привилегий. Остальные биты адресуют элемент дескрипторной таблицы, который участвует в преобразовании адреса. Каждое расширение состоит из 8-битного поля управления доступом, 24-битного поля начального адреса сегмента и 16-битного поля предела.

Если производится обращение к памяти, а используемый сегментный регистр не изменился, формируется 24-битный физический адрес посредством сложения 24-битного сегментного адреса из регистра расширения и эффективного адреса команды (при условии, что предел и права доступа не нарушаются). Если же сегментный регистр изменился явно (например, командой передачи данных) или неявно (например, командой перехода с типом FAR), процессор анализирует биты в сегментном регистре и автоматически загружает новое расширение из соответствующей дескрипторной таблицы, пользуясь смещением из сегментного регистра. Затем микропроцессор 80286 проверяет, находится ли нужный сегмент в памяти. При положительном ответе обращение к памяти завершается сложением сегментного адреса (из нового расширения) и эффективного адреса. При отрицательном ответе генерируется прерывание и операционная система передает сегмент из внешней памяти. После этого обращение к памяти завершается как и прежде.

Отметим, что в защищенном режиме микропроцессор 80286 выдает 24-битный адрес и может адресовать до 16М байт. Отметим также, что логический адрес состоит из 14-битного номера сегмента (13-битный индекс таблицы и однобитный индикатор таблицы) и 16-битного эффективного адреса. Это позволяет каждому заданию использовать виртуальную память до 1Г (2^{30}) байт.

Как показано на рис. 7.18, расширение сегментного регистра и все соответствующие дескрипторы содержат также поле управления доступом и поле предела. Поле предела хранит смещение последнего байта в сегменте. Для сегментов кода и данных это поле должно быть наибольшим смещением в сегменте, т. е. размером сегмента. Однако сегмент стека "растет" от своего наибольшего адреса к наименьшему. Следовательно, для сегмента стека поле предела должно хранить наименьшее смещение в сегменте и обращение подавляется, если эффективный адрес меньше этого предела.

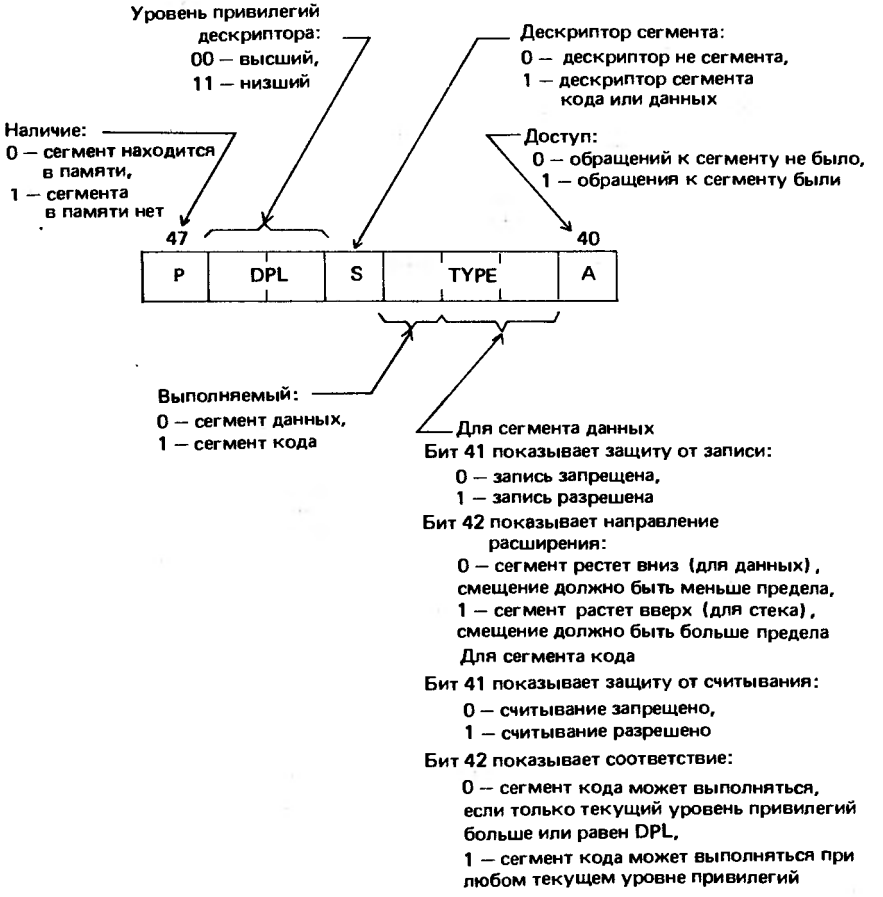


Рис. 7.19. Определение поля управления доступом

Поле управления доступом определено на рис. 7.19. Бит P показывает, находится сегмент в памяти или нет, а биты DPL определяют уровень привилегий сегмента. Вместе с полем TYPE эти биты могут защитить сегмент кода от выполнения той задачей, текущий уровень привилегий которой меньше уровня привилегий дескриптора. Бит S показывает, каким является дескриптор: сегментным или несегментным. Кроме сегментных дескрипторов для сегментов кода, данных и стека, в микропроцессоре 80286 имеются системные управляющие дескрипторы для специальных системных данных и операций передач управления. Старший бит поля TYPE (бит 43) различает сегмент данных от сегмента кода и его можно использовать для определения того, можно ли выполнять сегмент. Для сегмента данных бит 42 показывает направление расширения, а бит 41 обеспечивает защиту от операций записи. Для сегмента кода эти же два бита применяются для защиты от выполнения, если уровень привилегий текущей задачи меньше DPL, и от операций считывания соответственно.

Важно отметить, что расширения сегментных регистров не являются программируемыми. Как уже упоминалось, когда содержимое сегментного регистра изменяется командой, выбранный сегментный дескриптор автоматически загружается в расширение из соответствующей дескрипторной табли-

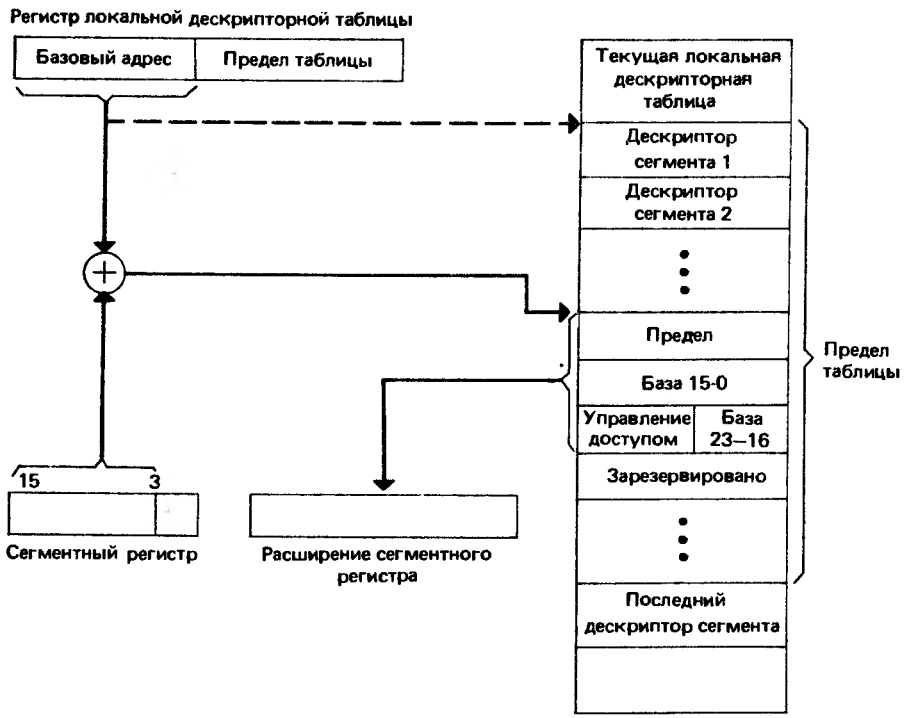


Рис. 7.20. Загрузка расширения сегментного регистра

цы, находящейся в памяти. Рис. 7.20 иллюстрирует этот процесс для случая, когда новый сегментный дескриптор находится в локальной дескрипторной таблице для текущей задачи. Базовый адрес в регистре локальной дескрипторной таблицы адресует базу текущей локальной дескрипторной таблицы, а старшие 13 бит в сегментном регистре служат индексом. Первые три слова выбранного дескриптора загружаются в расширение сегментного регистра. (В дескрипторе имеется и четвертое слово, но оно пока не используется.) Переключение с одной задачи на другую изменяет содержимое регистра локальной дескрипторной таблицы, и он адресует другую локальную дескрипторную таблицу. Микропроцессор 80286 имеет 16 команд загрузки и запоминания регистров дескрипторных таблиц, а также модификации и проверки различных полей дескрипторов.

Чтобы упростить переключение процессов (или задач), каждая задача имеет сегмент состояния задачи, который служит областью хранения всего состояния выполнения. Текущий сегмент состояния задачи определяется системным сегментным дескриптором, который адресуется так называемым регистром задачи. Сегмент состояния задачи в основном предназначен для запоминания регистров. Во время переключения задач микропроцессор 80286 запоминает все состояние выполнения в текущем сегменте состояния задачи, загружает новое состояние выполнения из выбранного сегмента состояния задачи и возобновляет новую задачу. Чтобы подробнее познакомиться с переключением задач и другими особенностями мультипрограммирования микропроцессора 80286, следует обратиться к фирменным руководствам.

Упражнения

1. Предположим, что выполняются задания A, B, C и D, ни в одном из которых нет ввода-вывода, и времена их выполнения равны 60, 20, 40 и 10 мин соответственно. Пусть все задания введены в систему одновременно и в указанном выше порядке.

а) Определите среднее оборотное время, если задания выполняются последовательно в указанном порядке.

б) Определите среднее оборотное время в системе с разделением времени без приоритетов.

Оборотное время задания определяется как время между моментом ввода задания в систему и моментом его окончания.

2. Рассмотрите мультипрограммную систему, которая осуществляет резервирование билетов с двух терминалов. Считайте, что каждый терминал инициирует процесс для выполнения следующих операций:

1. Ввести с терминала число запрошенных билетов.

2. Сравнить число имеющихся билетов, которое хранится в общей переменной TICKET_AVA, с запрошенным числом. Если запрошенное число больше, перейти к шагу 4; в противном случае перейти к шагу 3.

3. Вычесть запрошеиное число из TICKET_AVA, напечатать билеты и возвратиться к шагу 1.

4. Напечатать сообщение "НЕТ ДОСТАТОЧНОГО ЧИСЛА БИЛЕТОВ" и возвратиться к шагу 1.

Предположим, что в TICKET_AVA осталось 5 билетов, а с терминалов введены числа 4 и 3.

а) Если проблема критической секции не возникает, определите возможные конечные значения в TICKET_AVA и числа выданных билетов.

б) Определите все возможные результаты, если возникает проблема критической секции.

3. В некоторых компьютерах нет специальных команд проверки и установки. Предположите, что в микропроцессоре 8086 нет команды XCHG, и дайте возможную реализацию семафорных операторов (*Указание*. Воспользуйтесь командой циклического сдвига через перенос.)

4. Предположим, что процессы A и B структурированы следующим образом:

Процесс A

Шаг 1. Декремент COUNT на 1. (COUNT является разделенной переменной и может модифицироваться и другими процессами; следовательно, ее необходимо защищать от одновременного использования.)

Шаг 2. Активизировать процесс B.

Шаг 3. Ожидать активизации процессом B.

Шаг 4. Выполнить некритическую секцию кода.

Шаг 5. Перейти к шагу 1.

Процесс B

Шаг 1. Ожидать активизации процессом A.

Шаг 2. Выполнить критическую секцию кода.

Шаг 3. Активизировать процесс A.

Шаг 4. Перейти к шагу 1.

а) Определите требуемые семафоры и, при необходимости, их начальные значения.

б) Введите в процессы A и B семафорные операторы, которые требуются для синхронизации процессов (см. пример в конце § 7.2).

5. Что такое чистый код? Перечислите некоторые его применения.

6. Что такое фрагментация памяти? Как ее можно уменьшить?

7. Предположим, что ARRAY1, ARRAY2 и PROC1 находятся соответственно в сегментах SEGA, SEGB и SEGC. В следующем программном фрагменте укажите позиционно-независимые команды, если SEGA, SEGB и SEGC хранятся как единое целое вместе с сегментом кода, содержащим фрагмент:

```
        .
        .
        .
MOV     AX,SEGA
MOV     DS,AX
MOV     BX,OFFSET ARRAY1
ADD     DX,[BX][SI]
MOV     AX,SEGB
MOV     DS,AX
MOV     BX,OFFSET ARRAY2
MOV     [BX][SI],DX
CALL    FAR PTR PROC1
        .
        .
        .
```

8. Превратите фрагмент из упр. 7 в позиционно-независимый код, сделав следующие предположения:

а) имеется такая таблица указателей:

```
SEGPTR          SEGMENT
        ARRAY1PTR   DD    ARRAY1
        ARRAY2PTR   DD    ARRAY2
        PROC1PTR    DD    PROC1
          .
          .
          .
SEGPTR          ENDS
```

б) после ввода программы операционная система заполняет фактические сегментные адреса в таблице указателей;

в) сегментный регистр ES зарезервирован для обращений к таблице указателей и всегда содержит сегментный адрес SEGPTR.

9. Предположим, что имеется очередь процессов, находящихся в состоянии готовности (см. рис. 7.3), и что процесс из верха очереди переключается в состояние выполнения. Напишите программу, которая модифицирует указатель первого элемента и таблицу процессов, а также загружает в BX начальный адрес соответствующего управляющего блока процесса.

10. Повторите упр. 9 для приоритетной очереди (см. рис. 7.4).

11. Предположим, что в низ таблицы, показанной на рис. 7.3, добавляется процесс 11 и что BX содержит адрес его управляющего блока процесса. Напишите программу, которая модифицирует указатель последнего элемента и таблицу процессов.

12. Повторите упр. 11 для приоритетной очереди, показанной на рис. 7.4. Считайте, что приоритетный уровень процесса 11 равен 2.

13. Пусть максимальный размер сегмента составляет 1000 байт и задание может иметь в памяти только три сегмента. Для приведенной последовательности обращений

500, 1200, 1400, 2300, 900, 5300, 3500, 2800, 3800,
4300, 2500, 3600, 3700, 4200, 5400, 5500, 3650, 2700

определите число возникающих сегментных нарушений, если в системе применяется:

а) алгоритм замещения FIFO,

б) алгоритм замещения LFU,

в) алгоритм замещения LRU.

14. Необходимо ли вводить дополнительные схемы в логику управления памятью для реализации алгоритма замещения FIFO? Почему такие дополнительные схемы потребуются для алгоритмов LFU и LRU?

15. Какая информация потребуется от ЦП вместе с логическим адресом, чтобы схема управления памятью могла осуществлять защиту по считыванию,записи и выполнению?

8. СТРУКТУРА СИСТЕМНОЙ ШИНЫ

Множество проводников, предназначенных для передачи информации между компонентами вычислительной системы, называется *шиной*. Шина, соединяющая две вспомогательные компоненты в составе основной (например, устройство управления с рабочими регистрами в составе ЦП), называется *внутренней*. Шина, связывающая две основные компоненты, например ЦП и интерфейс, называется *внешней*. Так как внутренняя шина обычно на-

ходится в микросхеме и ее организация зависит от устройства, точная конфигурация ее не представляет интереса для пользователей. Внешние же шины имеют такие характеристики, которые необходимо учитывать при проектировании общей архитектуры вычислительной системы. В некоторых системах имеется несколько внешних шин, такие системы рассматриваются в гл. 12. В других системах имеется только одна внешняя шина, называемая *системной*. В данной главе обсудим базовую структуру системных шин, особенно предназначенных для работы с микропроцессорами 8086/8088.

Основополагающая структура системной шины и ее взаимосвязи с различными компонентами вычислительной системы представлены на рис. 8.1. Дан-

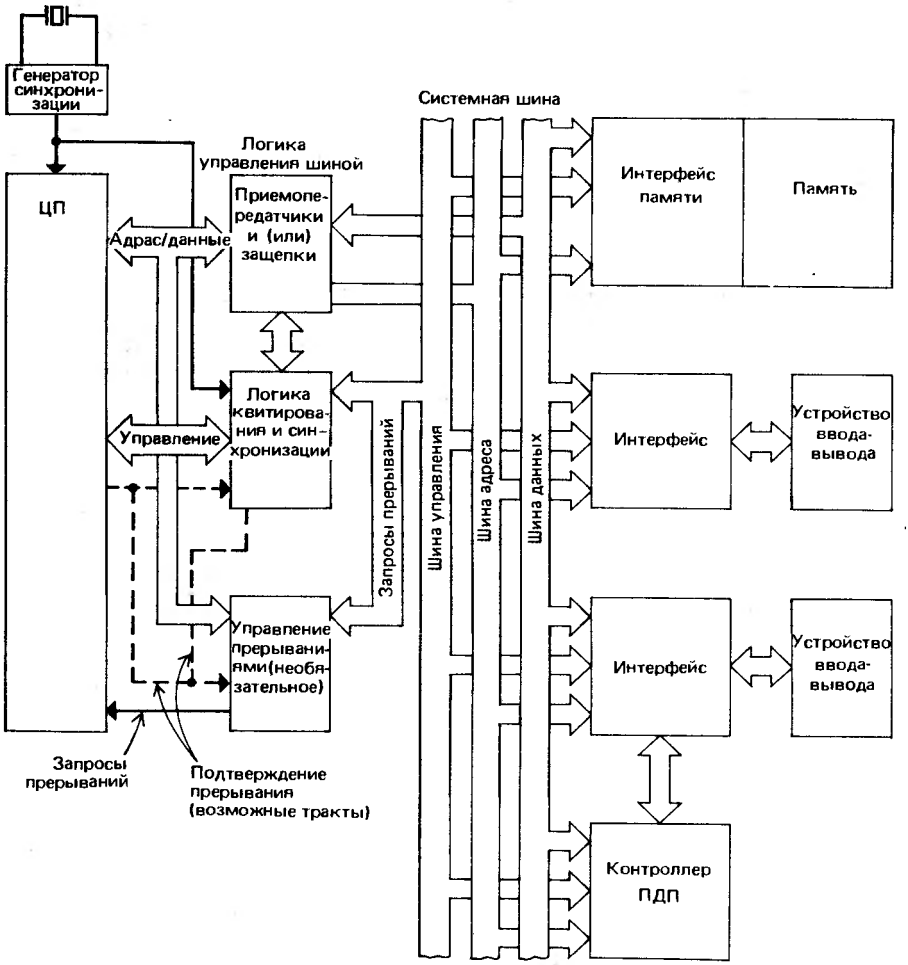


Рис. 8.1. Типичная архитектура системной шины

ная глава в основном касается интерфейса между ЦП и системной шиной — логики управления шиной. Вопросы о том, как подключенные к шине интерфейсы реагируют и интерпретируют сигналы на шине, обсуждаются в гл. 9 и 10. Сложность логики управления шиной зависит от объема преобразований между системной шиной и контактами ЦП, временных соотношений, наличия или отсутствия прерываний и размера системы. Если отличающаяся от ЦП компонента системы может быть ведущим шины, необходима возможность отключения от ЦП или логики управления шиной всех линий адреса и данных, а также большинства линий управления. Другими словами, большинство подключенных к шине линий должны обеспечивать перевод в высокоимпедансное состояние.

В общем выходные схемы однокристального ЦП имеют довольно ограниченную нагрузочную способность и их можно подключить лишь к небольшому числу интерфейсов. В небольшой системе большинство или все управляющие выходы ЦП можно использовать непосредственно, а логику квинтирования удается сократить или даже не применять. Для линий адреса и данных могут не потребоваться драйверы и приемники. Однако в системах с несколькими интерфейсами для сохранения параметров сигналов придется подключать к шине драйверы и приемники.

Кроме драйверов и приемников, временная диаграмма может потребовать введения регистров-защелок для хранения адреса, который выводится ЦП во время одной части цикла шины, но должен сохраняться весь цикл. В некоторых процессорах, включая 8086/8088, для адресов и данных применяются одни и те же контакты. Следовательно, в операции передачи данных выводимый первым адрес необходимо зафиксировать до того, как шина используется для передачи данных.

Временной диаграммой сигналов внутри ЦП и логики управления шиной управляет генератор синхронизации, а циклами шины и действиями ЦП — группы импульсов синхронизации. Точное число импульсов синхронизации (или тактов) в цикле шины оказывается переменным. Типичная операция ввода в ЦП развивается в такой последовательности: вывод адреса данных в первом такте синхронизации, сигнализация о производстве операции считывания во втором такте, ожидание в течение неопределенного числа тактов синхронизации, пока адресованное устройство не поместит данные на линии данных, ввод данных и сигнализация устройству о завершении передачи в последнем такте синхронизации. В операции вывода адрес по-прежнему выдается в первом такте, а выводимые данные вместе с сигналом записи — во втором. После того как адресованное устройство восприняло данные, оно возвращает подтверждение о завершении передачи, которое заставляет ЦП сформировать последний такт.

Как и ранее, основной материал относится к микропроцессору 8086, а для 8088 указываются только его отличия. Напомним, что микропроцессор 8088 полностью программно совместим с 8086, а будучи 8-битным процессором аппаратно совместим и периферийными схемами семейства 8080/8085. При использовании микропроцессора 8088 систему на базе 8080/8085 можно модифицировать с минимальными изменениями аппаратных средств.

В § 8.1 рассматриваются режимы работы и назначения контактов микропроцессоров 8086/8088 и одношинные конфигурации систем на их основе. Далее обсуждаются временные характеристики шины (§ 8.2) и контроллер приоритетных прерываний (§ 8.3). Наконец, в § 8.4 речь идет о стандартах шины с более подробным рассмотрением шины MULTIBUS фирмы Intel.

8.1. БАЗОВЫЕ КОНФИГУРАЦИИ МИКРОПРОЦЕССОРОВ 8086/8088

Для адаптации к возможно большему числу применений в микропроцессорах 8086/8088 предусмотрены два режима работы: минимальный и максимальный. Минимальный режим рассчитан на небольшие однопроцессорные системы, в которых все необходимые сигналы управления шиной генерирует непосредственно ЦП (что минимизирует необходимую логику управления шиной). Максимальный режим рассчитан на средние и большие системы, часто являющиеся мультипроцессорными. В этом режиме микропроцессоры 8086/8088 кодируют основные сигналы управления шиной в 3 бита состояния и используют освободившиеся контакты для дополнительной информации, которая требуется в мультипроцессорной конфигурации.

Разводка и определение контактов, общих для обоих режимов, приведены на рис. 8.2. Контакт 33 (MN/$\overline{MX}$) определяет вариант конфигурации. При подключении контакта на землю процессор работает в максимальном режиме, а при подключении к источнику питания +5 В – в минимальном. Оба процессора мультиплексируют сигналы адреса и данных и оба имеют 20 контак-

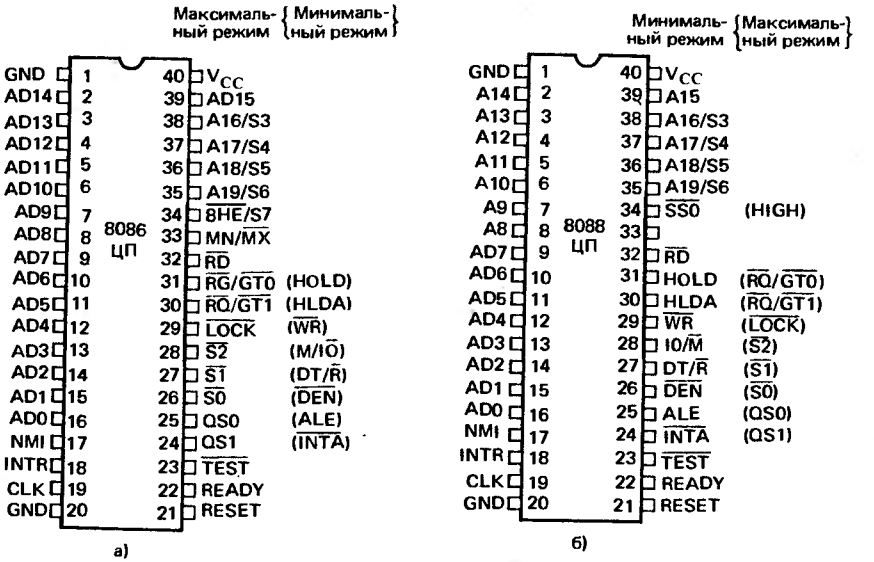


Рис. 8.2. Назначения контактов корпуса микропроцессоров 8086 (*а*) и 8088 (*б*) и определения их (*в*)

КОНТАКТ(Ы)	ОБОЗНАЧЕНИЕ	ВХОД/ВЫХОД 3-СТ.	ОПИСАНИЕ
1	GND		ЗЕМЛЯ
2-16[1]	AD14-AD0	ВХ/ВЫХ-З	В ТЕЧЕНИЕ ПЕРВОЙ ЧАСТИ ЦИКЛА ШИНЫ СОДЕРЖАТ АДРЕС, А В ОСТАЛЬНОЙ ЧАСТИ ВВОДЯТ ИЛИ ВЫВОДЯТ ДАННЫЕ
17	NMI	ВХОД	ЗАПРОС НЕМАСКИРУЕМОГО ПРЕРЫВАНИЯ (НАРАСТАЮЩИЙ ФРОНТ)
18	INTR	ВХОД	ЗАПРОС МАСКИРУЕМОГО ПРЕРЫВАНИЯ (УРОВЕНЬ)
19[2]	CLK	ВХОД	СИНХРОНИЗАЦИЯ С КОЭФФИЦИЕНТОМ ЗАПОЛНЕНИЯ 33% И ЧАСТОТОЙ, ЗАВИСЯЩЕЙ ОТ МОДЕЛИ ЦП: 8086 5МГЦ 8086-2 8МГЦ 8086-1 10МГЦ
20	GND		ЗЕМЛЯ
21	RESET	ВХОД	ПРЕКРАЩАЕТ ДЕЙСТВИЯ, СБРАСЫВАЕТ PSW, IP, DS, SS, ES И ОЧЕРЕДЬ КОМАНД, ЗАГРУЖАЕТ В CS КОД FFFF. ПОСЛЕ СНЯТИЯ СИГНАЛА СБРОСА РАБОТА НАЧИНАЕТСЯ С АДРЕСА FFFF0. МИНИМАЛЬНАЯ ПРОДОЛЖИТЕЛЬНОСТЬ СОСТАВЛЯЕТ 4 ТАКТА СИНХРОНИЗАЦИИ
22	READY	ВХОД	ПОДТВЕРЖДЕНИЕ ОТ ИНТЕРФЕЙСА ПАМЯТИ ИЛИ ВВОДА-ВЫВОДА О ТОМ, ЧТО ЦП МОЖЕТ ЗАКОНЧИТЬ ТЕКУЩИЙ ЦИКЛ ШИНЫ
23	$\overline{\text{TEST}}$	ВХОД	ПРИМЕНЯЕТСЯ С КОМАНДОЙ WAIT В МУЛЬТИПРОЦЕССОРНЫХ СИСТЕМАХ. КОМАНДА WAIT ЗАСТАВЛЯЕТ ЦП ОЖИДАТЬ (ЗА ИСКЛЮЧЕНИЕМ ОБРАБОТКИ ПРЕРЫВАНИЯ) ПОЯВЛЕНИЯ НА ЭТОМ ВХОДЕ СИГНАЛА 0 (СМ. ГЛ. 11)
24-31	--	--	ОПРЕДЕЛЕНИЕ ЗАВИСИТ ОТ РЕЖИМА (СМ. РИС. 8.3 и 8.8)
32	$\overline{\text{RD}}$	ВЫХОД-З	ПОКАЗЫВАЕТ ВЫПОЛНЕНИЕ СЧИТЫВАНИЯ ИЗ ПАМЯТИ ИЛИ ВВОДА-ВЫВОДА
33	MN/$\overline{\text{MX}}$	ВХОД	ПРИ ЗАЗЕМЛЕНИИ ЭТОГО ВХОДА ЦП РАБОТАЕТ В МАКСИМАЛЬНОМ РЕЖИМЕ, А ПРИ ПОДКЛЮЧЕНИИ К +5 В - В МИНИМАЛЬНОМ
34[3]	$\overline{\text{BHE}}$/S7	ВЫХОД-З	СИГНАЛ 0 В ПЕРВОЙ ЧАСТИ ЦИКЛА ШИНЫ ПОКАЗЫВАЕТ, ЧТО ПЕРЕДАЧА БАЙТА В ТЕКУЩЕМ ЦИКЛЕ ОСУЩЕСТВЛЯЕТСЯ ПО ЛИНИЯМ AD15-AD8; СИГНАЛ 1 ОЗНАЧАЕТ ПЕРЕДАЧУ ТОЛЬКО ПО ЛИНИЯМ AD7-AD0. СОСТОЯНИЕ S7 ВЫВОДИТСЯ В ОСТАЛЬНОЙ ЧАСТИ ЦИКЛА ШИНЫ, НО ПОКА ОНО НИКАКОЙ ФУНКЦИИ НЕ ВЫПОЛНЯЕТ
35-38	A19/S6-A16/S3	ВЫХОД-З	В ПЕРВОЙ ЧАСТИ ЦИКЛА ШИНЫ ВЫВОДЯТСЯ 4 СТАРШИХ БИТА АДРЕСА, А В ОСТАЛЬНОЙ ЧАСТИ ВЫВОДИТСЯ СОСТОЯНИЕ. S4 И S3 ПОКАЗЫВАЮТ ИСПОЛЬЗУЕМЫЙ СЕГМЕНТНЫЙ РЕГИСТР: S4 S3 РЕГИСТР 0 0 ES 0 1 SS 1 0 CS ИЛИ НИКАКОЙ 1 1 DS S5 СОВПАДАЕТ С СОСТОЯНИЕМ ФЛАЖКА IF, А S6 ВСЕГДА СОДЕРЖИТ 0
39[1]	AD15	ВХ/ВЫХ-З	СМ. AD14-AD0
40	VCC	--	НАПРЯЖЕНИЕ ПИТАНИЯ +5 В ± 10%

[1] В МИКРОПРОЦЕССОРЕ 8088 ЛИНИИ AD15-AD8 СТАНОВЯТСЯ ЛИНИЯМИ A15-A8 И СЛУЖАТ ТОЛЬКО ДЛЯ ВЫВОДА АДРЕСА.

[2] ДЛЯ МИКРОПРОЦЕССОРА 8088 МАКСИМАЛЬНАЯ ЧАСТОТА СИНХРОНИЗАЦИИ СОСТАВЛЯЕТ 5МГЦ.

[3] В МИКРОПРОЦЕССОРЕ 8088 ЭТОТ КОНТАКТ ОБОЗНАЧЕН $\overline{\text{SSO}}$ И ПРИМЕНЯЕТСЯ В МИНИМАЛЬНОМ РЕЖИМЕ ДЛЯ УКАЗАНИЯ СОСТОЯНИЯ. ЛОГИЧЕСКИ ЭКВИВАЛЕНТЕН ЛИНИИ $\overline{\text{SO}}$ (СМ. РИС. 8.8). В МАКСИМАЛЬНОМ РЕЖИМЕ ВСЕГДА СОДЕРЖИТ СИГНАЛ 1.

тов адреса, причем сигналы адреса и состояния мультиплексируются на контакты четырех старших бит адреса. Так как микропроцессор 8088 может одновременно передавать только 8 бит данных, для передач данных у него используются восемь контактов, а в микропроцессоре 8086 — шестнадцать. Контакт 28 различается только в минимальном режиме. В микропроцессоре 8088 он инвертирован по сравнению с 8086, поэтому микропроцессор 8088 совместим с микросхемой 8085.

В микропроцессоре 8086 контакт 34 ($\overline{BHE}$) определяет, передается ли байт по линиям AD15-AD8. Сигнал $\overline{BHE}$ = 0 показывает, что старшие линии данных используются, в противном случае байт передается только по линиям AD7-AD0. Два сигнала $\overline{BHE}$ и A0 указывают подключенным к шине интерфейсам, каким образом данные должны появиться на шине. Четыре возможных их комбинации определены в табл. 8.1.

Таблица 8.1.

Использование линий данных

Операция	$\overline{BHE}$	A0	Линия данных
Запись/считывание слова по четному адресу	0	0	AD15-AD0
Запись/считывание байта по четному адресу	1	0	AD7-AD0
Запись/считывание байта по нечетному адресу	0	1	AD15-AD8 [1]
Запись/считывание слова по нечетному адресу	1	0	AD7-AD0 [2]

[1] Первый цикл шины: младший байт данных находится на линиях AD15-AD8.
[2] Следующий цикл шины: старший байт данных находится на линиях AD7-AD0.

Так как в микропроцессоре 8088 данные передаются только по линиям AD7-AD0, сигнал $\overline{BHE}$ не нужен и контакт 34 несет сигнал состояния.

Контакты 1 и 20 заземляются. Контакты 2-16 и 39 (AD15-AD0) в первой части цикла шины несут сигналы адреса, необходимого для передачи, а в остальной части цикла используются для передачи данных.

Контакты 17 и 18 (NMI и INTR) предназначены для запросов прерываний (см. гл. 6). На контакт 19 (CLK) подается сигнал синхронизации, который координирует действия в ЦП.

Контакт 21 (RESET) воспринимает сигнал системного сброса. В большинстве систем имеется линия, которая подключена ко всем компонентам и на которой при включении системы автоматически формируется импульс сброса. Кроме того, он вырабатывается и вручную, что позволяет оператору реинициализировать систему. Сигнал 1 на линии сброса заставляет все компоненты перейти в выключенное состояние. В процессоре регистры PSW, IP, DS, SS, ES и очередь команд сбрасываются, а в CS загружается код FFFF. Когда (IP) = 0000 и (CS) = FFFF, процессор начинает выполнение с адреса FFFF0. Обычно эта ячейка находится в постоянном запоминающем устройстве и содержит команду перехода JMP к программе инициализации системы и загрузки прикладной программы или операционной системы. Данная программа называется *настраивающим загрузчиком*.

Контакт 22 (READY) предназначен для ввода подтверждения от интерфейсов памяти или ввода-вывода о том, что в следующем такте входные данные помещаются на шину данных или выходные данные воспринимаются с шины данных. В любом случае ЦП и его логика управления шиной могут завершить текущий цикл шины после следующего такта синхронизации. Контакт 23 ($\overline{\text{TEST}}$) применяется вместе с командой WAIT и предназначен в основном для мультипроцессорных систем (см. гл. 11). Сигналы на контактах 24-31 зависят от режима работы и рассматриваются далее. Контакт 32 ($\overline{\text{RD}}$) показывает, что выполняется операция ввода, и в минимальном режиме испольуется вместе с контактами 28 (различает передачу памяти от передачи ввода-вывода) и 29 (указывает операцию вывода) для определения типа передачи.

В течение первой части цикла шины на контакты 35-38 (A19/S6-A16/S3) выводятся старшие 4 бита адреса, а в остальной части цикла – информация о состоянии. Биты S3 и S4 показывают, какой сегментный регистр участвовал в формировании адреса, а бит S5 отражает состояние флажка IF. Бит S6 всегда содержит 0 и показывает, что микропроцессор управляет системной шиной.

На контакт 40 подается напряжение питания, равное +5 В ± 10 %. Системы на базе микропроцессоров 8086/8088 обычно требуют только ТТЛ-совместимого питания +5 В, что упрощает разработку источника питания.

8.1.1. МИНИМАЛЬНЫЙ РЕЖИМ

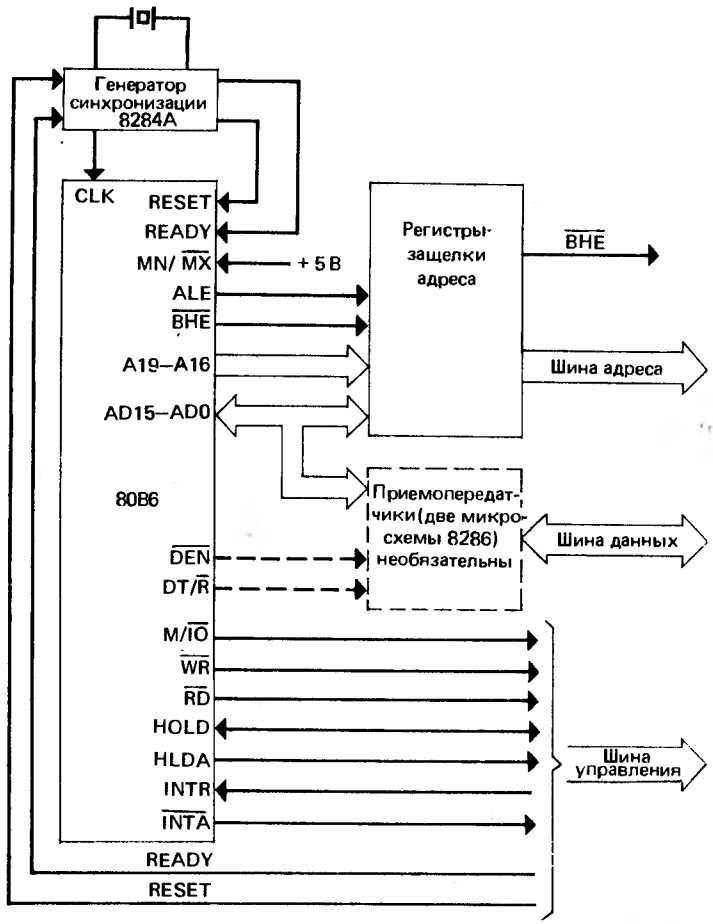
Процессор работает в минимальном режиме, когда контакт MN/$\overline{\text{MX}}$ подключен к питанию +5 В. Определения контактов 24-31 для минимального режима даны на рис. 8.3, а типичная конфигурация минимального режима показана на рис. 8.4. Адрес необходимо фиксировать в регистрах-защелках, так как он доступен только в течение первой части цикла шины. Сигнал ALE (разрешения регистра-защелки адреса) на контакте 25 уровнем 1 извещает о готовности адреса к загрузке в регистр. Обычно в качестве регистра-защелки применяется микросхема 8282 (см. рис. 8.5), рассчитанная на 8 бит, поэтому для 16-битного адреса необходимы две микросхемы, а для полного 20-битного адреса – три. В системе на базе микропроцессора 8086 необходимо также фиксировать сигнал $\overline{\text{BHE}}$. В небольшой системе на базе микропроцессора 8088, имеющей память только 64К байт, достаточно двух микросхем 8282.

КОНТАКТ	ОБОЗНАЧЕНИЕ	ВХОД/ВЫХОД 3-СТ.	ОПИСАНИЕ
24	$\overline{\text{INTA}}$	ВЫХОД-3	ПОКАЗЫВАЕТ РАСПОЗНАВАНИЕ ЗАПРОСА ПРЕРЫВАНИЯ. СОСТОИТ ИЗ ДВУХ ОТРИЦАТЕЛЬНЫХ ИМПУЛЬСОВ, ВЫДАВАЕМЫХ В ДВУХ СМЕЖНЫХ ЦИКЛАХ ШИНЫ
25	ALE	ВЫХОД	ИМПУЛЬС В НАЧАЛЕ ЦИКЛА ШИНЫ, ПОКАЗЫВАЮЩИЙ НАЛИЧИЕ АДРЕСА НА ЛИНИЯХ АДРЕСА
26	$\overline{\text{DEN}}$	ВЫХОД-3	ВЫДАЕТСЯ В ПОЗДНЕЙ ЧАСТИ ЦИКЛА ШИНЫ И ИНФОРМИРУЕТ ПРИЕМОПЕРЕДАТЧИКИ О ТОМ, ЧТО ЦП ГОТОВ ВЫДАВАТЬ ИЛИ ПРИНИМАТЬ ДАННЫЕ
27	DT/$\overline{\text{R}}$	ВЫХОД-3	ПОКАЗЫВАЕТ ПРИЕМОПЕРЕДАТЧИКАМ, ЧТО ОНИ ДОЛЖНЫ: ВЫДАВАТЬ (1) ИЛИ ПРИНИМАТЬ (0) ДАННЫЕ

Рис. 8.3. Определения сигналов минимального режима

28[1]	M/$\overline{IO}$	ВЫХОД-3	РАЗЛИЧАЕТ ПЕРЕДАЧИ ПАМЯТИ И ВВОДА-ВЫВОДА
29	$\overline{WR}$	ВЫХОД-3	СИГНАЛ О ПОКАЗЫВАЕТ ВЫПОЛНЕНИЕ ОПЕРАЦИИ ЗАПИСИ. ПРИМЕНЯЕТСЯ ВМЕСТЕ С СИГНАЛАМИ M/$\overline{IO}$ И $\overline{RD}$ ДЛЯ ОПРЕДЕЛЕНИЯ ТИПА ПЕРЕДАЧИ
30	HLDA	ВЫХОД	ВЫХОД РАЗРЕШЕНИЯ ШИНЫ ЗАПРАШИВАЮЩЕМУ ВЕДУЩЕМУ. КОГДА HLDA=1, ВСЕ ТРИСТАБИЛЬНЫЕ ВЫХОДЫ ПЕРЕВОДЯТСЯ В ВЫСОКОИМПЕДАНСНОЕ СОСТОЯНИЕ
31	HOLD	ВХОД	ВХОД ЗАПРОСОВ ШИНЫ ОТ ВЕДУЩИХ ШИНЫ. МИКРОПРОЦЕССОРЫ 8086/8088 НЕ ПОЛУЧАЮТ УПРАВЛЕНИЯ ШИНОЙ ДО СНЯТИЯ ЭТОГО СИГНАЛА

[1] В МИКРОПРОЦЕССОРЕ 8088 ЭТОТ СИГНАЛ ОБОЗНАЧЕН IO/$\overline{M}$



Примечание. В системе на базе микропроцессора 8088 сигнал $\overline{BHE}$ становится $\overline{SS0}$, сигнал M/$\overline{IO}$ превращается в IO/$\overline{M}$ и требуется одна микросхема 8286.

Рис. 8.4. Система минимального режима

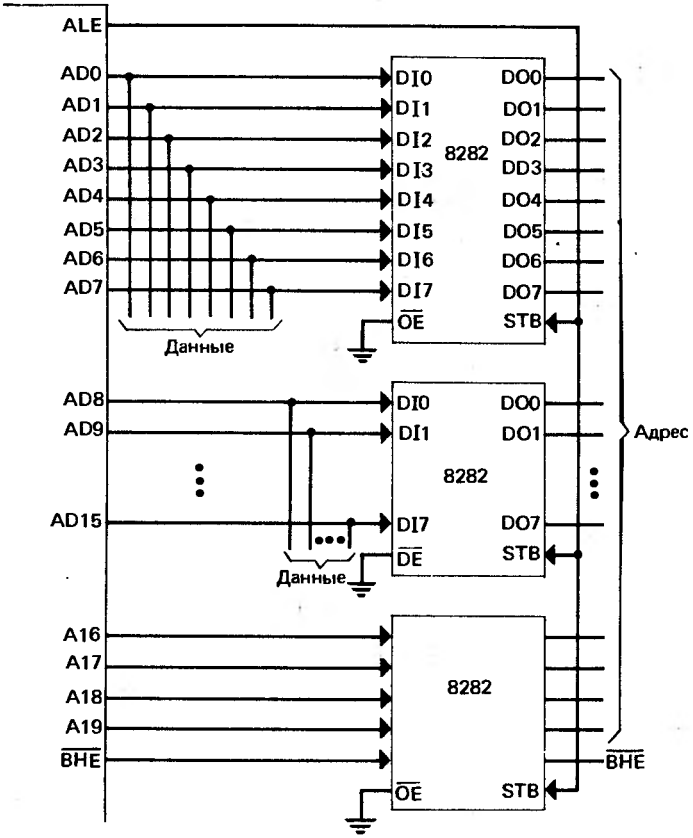


Рис. 8.5. Применение регистров-защелок 8282

Сигнал строба на контакте STB фиксирует биты, поданные на линии входных данных DI7-DI0. Поэтому вход STB соединяется с выходом ALE, а входы DI7-DI0 подключаются к восьми линиям адреса. Сигнал $\overline{OE}$ разрешает выходы DO7-DO0 регистра-защелки, причем $\overline{OE}$ = 1 переводит выходы в высокоимпедансное состояние. В однопроцессорных системах без контроллера ПДП (см. гл. 9) этот контакт заземляется.

В системе с несколькими интерфейсами в отличие от небольших одноплатных систем потребуются драйверы и приемники на линиях данных. Для реализации блока приемопередатчиков, показанного на рис. 8.4, предназначена микросхема 8286. Она содержит 16 тристабильных элементов – 8 приемников и 8 драйверов (передатчиков). Следовательно, для обслуживания всех 8 линий данных в микропроцессоре 8088 необходима одна микросхема, а для 16 линий данных в микропроцессоре 8086 – две таких микросхемы. На рис. 8.6 показаны подключение 8286 в системе и логическая схема одного

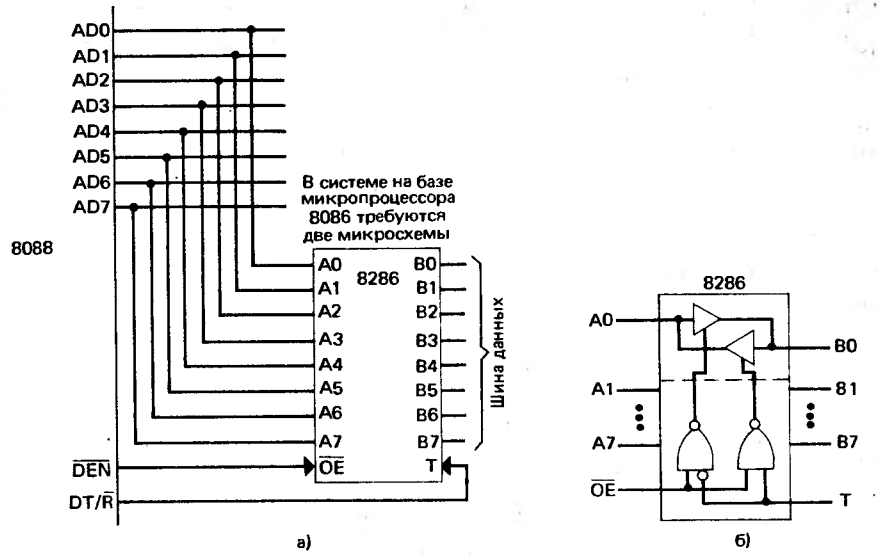


Рис. 8.6. Применение и внутреннее устройство приемопередатчиков 8286:
а – схема подключения к микропроцессору 8088; *б* – внутренняя схема

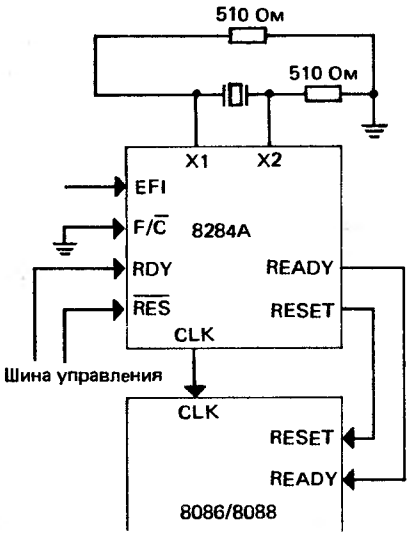


Рис. 8.7. Типичное подключение синхрогенератора 8284A

прибора. Микросхема 8286 симметрична относительно двух наборов данных: контакты A7-A0 могут быть входами, а B7-B0 – выходами или наоборот. Сигнал разрешения выхода $\overline{OE}$ определяет, разрешается ли данным проходить через микросхему 8286 или нет, а сигнал передачи T управляет направлением передачи. Когда $\overline{OE} = 1$, данные не проходят через 8286 ни в одном направлении. Если же $\overline{OE} = 0$, сигнал $T = 1$ заставляет линии A7-A0 быть входными, а $T = 1$ переводит во входные линии B7-B0. Вход $\overline{OE}$ следует подключить к выходу $\overline{DEN}$ микропроцессора, который имеет низкий уровень всегда, когда микропроцессор выполняет операцию ввода-вывода[1]. Линии A7-A0 подключаются к соответствующим линиям адреса/данных, а вход T – к выходу $DT/\overline{R}$ микропроцессора. Следовательно, когда процес-

[1] Точнее, операцию считывания/записи. – *Прим. перев.*

сор выводит, данные передаются с A7-A0 на B7-B0, а при вводе передача осуществляется в обратном направлении. Процессор переводит линии $\overline{DEN}$ и DT/$\overline{R}$ в высокоимпедансное состояние в ответ на запрос шины по линии HOLD.

Иногда системная шина проектируется так, что сигналы адреса и (или) данных инвертируются. Поэтому микросхемы 8282 и 8286 имеют свои полные аналоги 8283 и 8287, осуществляющие инверсию сигналов при передаче со входов на выходы.

Третьей компонентой на рис. 8.4, отличной от процессора, является генератор синхронизации 8284A. Этот прибор подробнее показан на рис. 8.7. Кроме формирования последовательности импульсов с постоянной частотой, он осуществляет привязку сигналов готовности (RDY) и сброса ($\overline{RES}$) к импульсам синхронизации. Хотя эти сигналы могут поступить в любой момент времени, микросхема не превращает их в выходные сигналы READY и RESET до среза импульса синхронизации того такта, в котором они восприняты.

Источником частоты для генератора 8284A может быть генератор импульсов, который подключается на вход EFI, или осциллятор (кварц), включаемый между X1 и X2. Если на входе F/$\overline{C}$ = 1, частота определяется входом EFI, а при F/$\overline{C}$ = 0 – входом осциллятора. В любом случае частота на выходе синхронизации CLK равна 1/3 входной частоты.

Все микросхемы 8282, 8286 и 8284A требуют напряжения питания +5 В. Их входы и выходы ТТЛ-совместимы, поэтому они совместимы друг с другом и с микропроцессорами 8086/8088. Более подробная информация об этих микросхемах, особенно 8284A, имеется в фирменных руководствах.

В минимальной системе линии управления допускается использовать непосредственно, без приемопередатчиков. Сигналы M/$\overline{IO}$, $\overline{RD}$ и $\overline{WR}$ определяют тип передачи в соответствии с табл. 8.2.

Таблица 8.2

Типы передач минимального режима

M/$\overline{IO}$	$\overline{RD}$	$\overline{WR}$	Тип передачи
0	0	1	Считывание ввода-вывода
0	1	0	Запись ввода-вывода
1	0	1	Считывание из памяти
1	1	0	Запись в память

Назначение линий запроса шины (HOLD), разрешения шины (HLDA), запроса прерывания (INTR) и подтверждения прерывания ($\overline{INTA}$) были рассмотрены в гл. 6. Сигнал $\overline{INTA}$ состоит из двух отрицательных импульсов, выдаваемых в двух смежных циклах шины. Первый импульс сообщает интерфейсу, что его запрос воспринят, а при получении второго импульса интерфейс посылает в процессор по шине данных тип прерывания.

8.1.2. МАКСИМАЛЬНЫЙ РЕЖИМ

Процессор работает в максимальном режиме, когда контакт MN/$\overline{MX}$ заземляется. Определения контактов 24-31 для максимального режима даны на рис. 8.8, а типичная конфигурация максимального режима показана на рис. 8.9. Из рис. 8.9 видно, что основное различие между конфигурациями минимального и максимального режимов заключается в необходимости дополнительных схем для преобразования сигналов управления. Эти схемы предназначены для преобразования бит состояния $\overline{S0}$, $\overline{S1}$ и $\overline{S2}$ в сигналы передачи памяти и ввода-вывода, необходимые для определения направления передач данных и для управления микросхемами 8282 и 8286. Обычно дополнительные схемы сосредоточены в контроллере шины 8288. В системе показано устройство управления приоритетными прерываниями, но его наличие не обязательно.

КОНТАКТ	ОБОЗНАЧЕНИЕ	ВХОД/ВЫХОД 3-СТ.	ОПИСАНИЕ
24,25	QS1,QS0	ВЫХОД	ПОКАЗЫВАЮТ СОСТОЯНИЕ ОЧЕРЕДИ КОМАНД. СОСТОЯНИЕ ОТРАЖАЕТ ДЕЙСТВИЕ В ОЧЕРЕДИ В ПРЕДЫДУЩЕМ ТАКТЕ СИНХРОНИЗАЦИИ (СМ. ГЛ. 11)
26,27,28	$\overline{S0}$, $\overline{S1}$, $\overline{S2}$	ВЫХОД-3	ИДЕНТИФИЦИРУЮТ ТИП ПЕРЕДАЧИ В ТЕКУЩЕМ ЦИКЛЕ ШИНЫ: $\overline{S2}$ $\overline{S1}$ $\overline{S0}$ 0 0 0 ПОДТВЕРЖДЕНИЕ ПРЕРЫВАНИЯ 0 0 1 СЧИТЫВАНИЕ ИЗ ПОРТА ВВОДА-ВЫВОДА 0 1 0 ЗАПИСЬ В ПОРТ ВВОДА-ВЫВОДА 0 1 1 ОСТАНОВ 1 0 0 ВЫБОРКА КОМАНДЫ 1 0 1 СЧИТЫВАНИЕ ИЗ ПАМЯТИ 1 1 0 ЗАПИСЬ В ПАМЯТЬ 1 1 1 ПАССИВНЫЙ СОСТОЯНИЕ СТАНОВИТСЯ АКТИВНЫМ ДО НАЧАЛА ЦИКЛА ШИНЫ И ВОЗВРАЩАЕТСЯ В ПАССИВНОЕ В ПОСЛЕДНЕЙ ЧАСТИ ЦИКЛА
29	$\overline{LOCK}$	ВЫХОД-3	ПОКАЗЫВАЕТ, ЧТО ШИНА НЕ ОСВОБОЖДАЕТСЯ ДРУГИМ ПОТЕНЦИАЛЬНЫМ ВЕДУЩИМ. ИНИЦИИРУЕТСЯ ПРЕФИКСОМ LOCK И СОХРАНЯЕТСЯ ДО КОНЦА СЛЕДУЮЩЕЙ ЗА НИМ КОМАНДЫ (СМ. ГЛ. 11). АКТИВЕН ВО ВРЕМЯ И МЕЖДУ ИМПУЛЬСАМИ $\overline{INTA}$
30	$\overline{RQ/GT1}$	ВХ/ВЫХ	ДЛЯ ВЫВОДА ЗАПРОСОВ ШИНЫ И ВЫДАЧИ РАЗРЕШЕНИЙ ШИНЫ
31	$\overline{RQ/GT0}$	ВХ/ВЫХ	АНАЛОГИЧЕН $\overline{RQ/GT1}$, НО ЗАПРОС НА ЛИНИИ $\overline{RQ/GT0}$ ИМЕЕТ БОЛЬШИЙ ПРИОРИТЕТ

ПРИМЕЧАНИЕ. В МАКСИМАЛЬНОМ РЕЖИМЕ КОНТАКТЫ МИКРОПРОЦЕССОРОВ 8086 И 8088 ОПРЕДЕЛЯЮТСЯ ОДИНАКОВО, НО НА ВЫХОДЕ 34 МИКРОПРОЦЕССОРА 8088 ВСЕГДА ДЕЙСТВУЕТ СИГНАЛ 1

Рис. 8.8. Определения сигналов максимального режима

Биты состояния $\overline{S0}$, $\overline{S1}$ и $\overline{S2}$ определяют тип выполняемой передачи и при использовании контроллера шины делают ненужными сигналы M/$\overline{IO}$, $\overline{WR}$, $\overline{INTA}$, ALE, DT/$\overline{R}$ и $\overline{DEN}$ минимального режима. За исключением случая $\overline{S1} = \overline{S0} = 1$ сигнал $\overline{S2} = 0$ указывает передачу между интерфейсом ввода-вывода и ЦП, а $\overline{S2} = 1$ означает обращение к памяти. Бит $\overline{S1}$ определяет, что выполняется: ввод или вывод. По сигналам состояния контроллер шины 8288

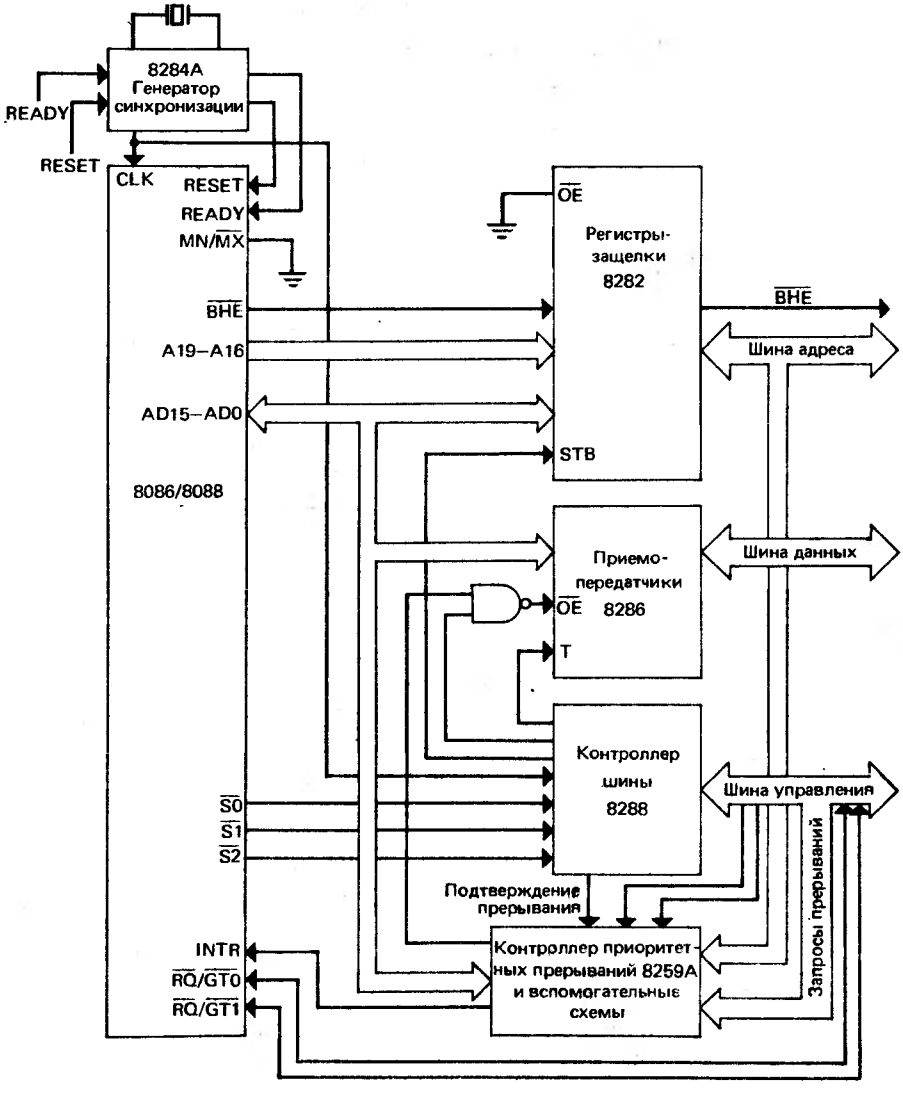


Рис. 8.9. Типичная конфигурация максимального режима:
Примечание. Сигнал $\overline{BHE}$ в системе на базе микропроцессора 8088 отсутствует

формирует сигнал ALE для микросхем 8282, сигналы DEN и DT/$\overline{R}$ – для приемопередатчиков 8286 и сигнал $\overline{INTA}$ – для контроллера прерываний.

Сигналы QS0 и QS1 позволяют внешней по отношению к ЦП системе опросить состояние очереди команд и определить текущую выполняемую команду. Сигнал $\overline{LOCK}$ показывает, что выполняется команда с префиксом LOCK

и шину не может использовать другой потенциальный ведущий шины. Данные сигналы требуются только в мультипроцессорных системах и подробно обсуждаются в гл. 11.

Сигналы HOLD и HLDA превращаются в сигналы $\overline{RQ}/GT0$ и $\overline{RQ}/GT1$. По линиям этих сигналов передаются и запросы шины, и разрешения шины. Обе линии совершенно одинаковы, но при наличии одновременных запросов шины запросу на линии $\overline{RQ}/GT0$ придается больший приоритет. Запрос представляет собой отрицательный импульс перед началом текущего цикла шины. Разрешение имеет вид отрицательного импульса в начале текущего цикла шины при условии, что:

1. Предыдущая передача по шине не была передачей младшего байта слова в (из) нечетный (ого) адрес (а) (в микропроцессоре 8086). В микропроцессоре 8088 независимо от выравнивания адреса сигнал разрешения не выдается до обращения ко второму байту слова.

2. В предыдущем цикле шины не было первого импульса подтверждения прерывания.

3. Не выполняется команда с префиксом LOCK.

Если не удовлетворяются условия 1 или 2, разрешение не выдается до следующего цикла шины, а если не удовлетворяется условие 3, разрешение должно ожидать завершения заблокированной команды. В ответ на разрешение тристабильные линии переводятся в высокоимпедансное состояние и следующий цикл шины отдается запрашивающему ведущему шины. Процессор эффективно отключается от системной шины до тех пор, пока ведущий не пошлет в процессор второй импульс по линии $\overline{RQ}/GT$.

Более подробно подключения к контроллеру шины показаны на рис. 8.10. Входы контроллера $\overline{S0}$, $\overline{S1}$ и $\overline{S2}$ предназначены для восприятия соответствующих бит состояния от процессора. Сигналы ALE, DT/$\overline{R}$ и DEN такие же, что и выдаваемые процессором в минимальном режиме (сигнал DEN инвертирован). Вход CLK позволяет синхронизировать действия контроллера шины и процессора. Сигналы $\overline{AEN}$, IOB и CEN предназначены для мультипроцессорных систем и обсуждаются в гл. 11. В однопроцессорной системе линии $\overline{AEN}$ и IOB обычно заземляются, а на вход CEN подается 1. Смысл выходного сигнала MCE/$\overline{PDEN}$ зависит от режима, определяемого сигналом на входе IOB. Когда IOB = 0, сигнал разрешения ведущего MCE можно использовать для управления несколькими приборами 8259A (см. § 8.3). Если IOB = 1, сигнал разрешения периферийных данных $\overline{PDEN}$ применяется в многошинных конфигурациях.

Остальные контакты на рис. 8.10 имеют следующие определения:

$\overline{INTA}$. Выдает два импульса подтверждения прерывания в контроллер прерываний или прерывающее устройство, когда $\overline{S0} = \overline{S1} = \overline{S2} = 0$.

$\overline{IORC}$ (приказ считывания ввода-вывода). Заставляет интерфейс ввода-вывода поместить данные из адресованного порта на шину данных.

$\overline{IOWC}$ (приказ записи ввода-вывода). Заставляет интерфейс ввода-вывода воспринять данные с шины данных и загрузить их в адресуемый порт.

$\overline{MRDC}$ (приказ считывания из памяти). Заставляет память поместить на шину данных содержимое адресованной ячейки памяти.

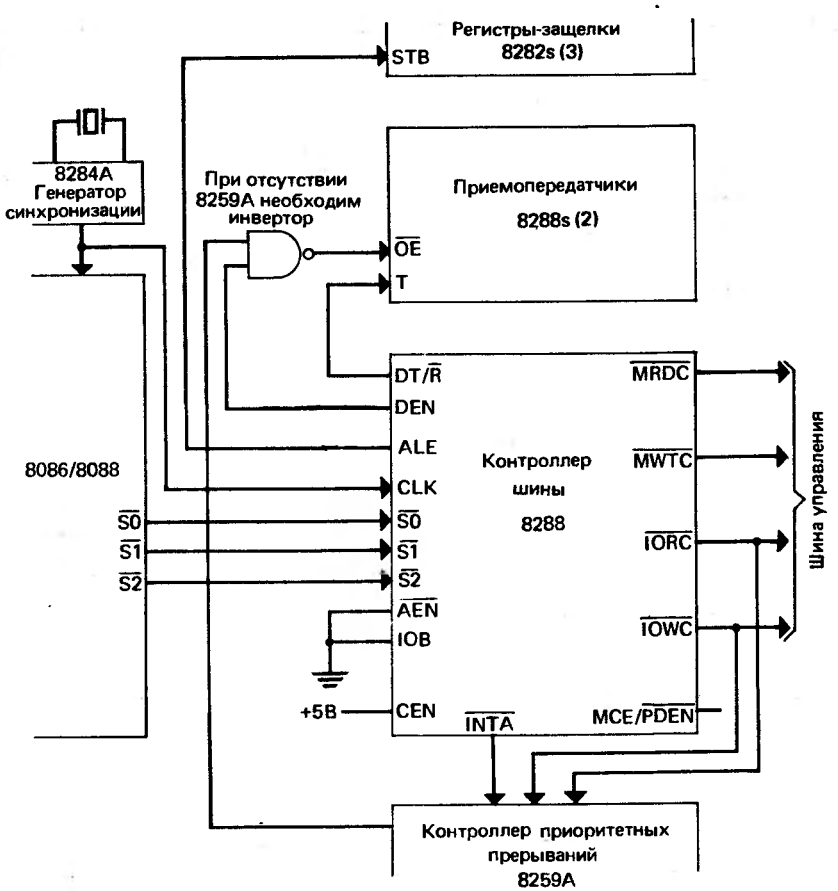


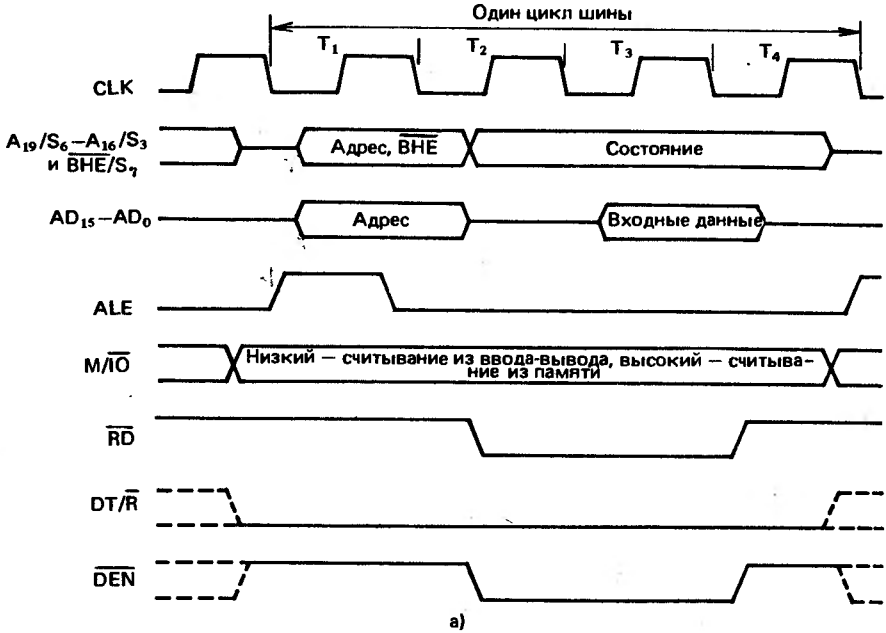
Рис. 8.10. Подключения к контроллеру шины

$\overline{\text{MWTC}}$ (приказ записи в память). Заставляет память воспринять данные с шины данных и поместить их в адресованную ячейку памяти.

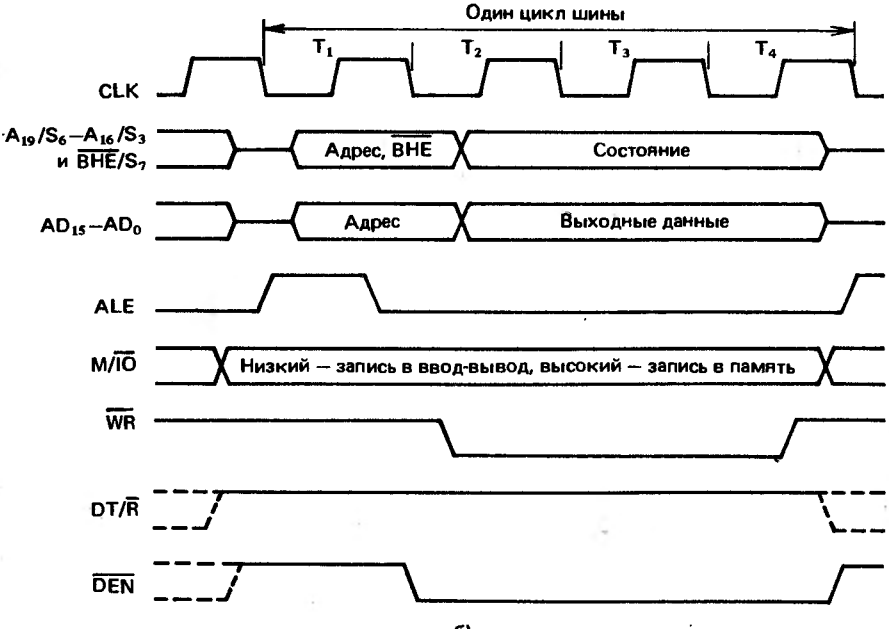
Все эти сигналы выдаются в средней части цикла шины. Очевидно, в любом цикле шины будет выдан только один из них.

На рис. 8.10 не показаны сигналы $\overline{\text{AIOWC}}$ (приказ опережающей записи ввода-вывода) и $\overline{\text{AMWC}}$ (приказ опережающей записи в память). Они выполняют те же функции, что и сигналы $\overline{\text{IOWC}}$ и $\overline{\text{MWTC}}$, но выдаются на один импульс синхронизации раньше. С их помощью медленным интерфейсам дается дополнительный такт синхронизации для подготовки к приему данных.

Контроллер шины имеет питание +5 В и ТТЛ-совместимые входы и выходы. Более подробную информацию о нем можно найти в фирменных руководствах.


Один цикл шины
T1
T2
T3
T4
CLK
A19/S6–A16/S3 и BHE/S7
Адрес, BHE
Состояние
AD15–AD0
Адрес
Входные данные
ALE
M/IO
Низкий — считывание из ввода-вывода, высокий — считывание из памяти
RD
DT/R
DEN

а)


Один цикл шины
T1
T2
T3
T4
CLK
A19/S6–A16/S3 и BHE/S7
Адрес, BHE
Состояние
AD15–AD0
Адрес
Выходные данные
ALE
M/IO
Низкий — запись в ввод-вывод, высокий — запись в память
WR
DT/R
DEN

б)

8.2. ВРЕМЕННЫЕ ДИАГРАММЫ СИСТЕМНОЙ ШИНЫ

До сих пор мы только упоминали первую, среднюю и последнюю части цикла шины. В данном разделе определим точнее временные соотношения системной шины. Продолжительность цикла шины состоит из четырех тактов T_1-T_4 плюс неопределенное число тактов ожидания T_W. Если шина по завершению своего цикла остается пассивной, промежуток между последовательными циклами заполняется холостыми тактами T_I. Состояния T_W вводятся между T_3 и T_4, когда при передаче интерфейсы памяти или ввода-вывода не успевают реагировать достаточно быстро. Типичная последовательность циклов шины представлена на рис. 8.11.

На рис. 8.12 показаны временные диаграммы передач ввода и вывода, не требующих состояния ожидания (микропроцессор 8086 работает в минимальном режиме). Когда процессор готов инициировать цикл шины, в такте T_1 он выдает сигнал ALE. До его среза сигналы адреса $\overline{BHE}$, M/$\overline{IO}$, $\overline{DEN}$ и DT/$\overline{R}$ должны быть стабильными, причем $\overline{DEN} = 1$ и DT/$\overline{R} = 0$ при вводе и DT/$\overline{R} = 1$ при выводе. По срезу сигнала ALE регистры-защелки 8282 фиксируют адрес. В такте T_2 адрес снимается, на линии A16/S3-A19/S6 и $\overline{BHE}$/S7 выдаются сигналы состояния S3-S7 и понижается уровень сигнала $\overline{DEN}$ для разрешения приемопередатчиков 8286. Если выполняется ввод, в такте T_2 активизируется сигнал $\overline{RD}$, а линии AD15-AD0 переводятся в высокоимпедансное состояние, подготавливаясь к вводу. Если интерфейсы памяти или ввода-вывода могут выполнить передачу сразу же, состояния ожидания не вводятся и в такте T_3 данные помещаются на шину. После того как процессор принял данные, в начале такта T_4 сигнал $\overline{RD}$ становится равным 1 и по его переходу интерфейс снимает сигналы данных. В операции вывода процессор выдает в T_2 сигнал $\overline{WR} = 0$ и данные, а в такте T_4 сигнал $\overline{WR}$ и дан-

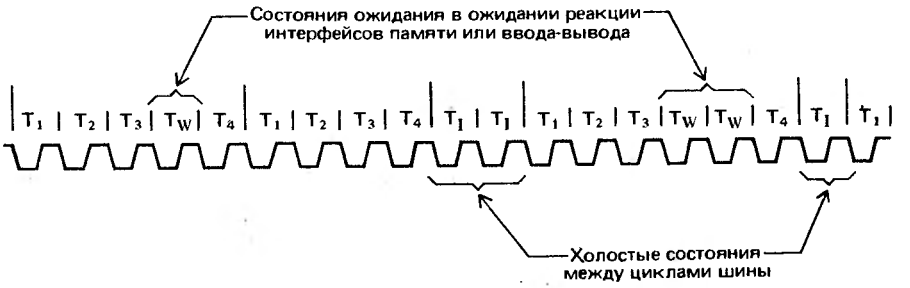


Рис. 8.11. Типичная последовательность циклов шины

←

Рис. 8.12. Временные диаграммы шины микропроцессора 8086 в минимальном режиме: *а* – ввод; *б* – вывод

Примечание. В микропроцессоре 8088 сигнал M/$\overline{IO}$ становится IO/$\overline{M}$, а сигнал $\overline{BHE}$/S7 превращается в $\overline{SS0}$, который действует во всем цикле шины (т. е. он изменяется в те же моменты времени, что и сигнал IO/$\overline{M}$). Кроме того, данные передаются только по линиям AD7-AD0.

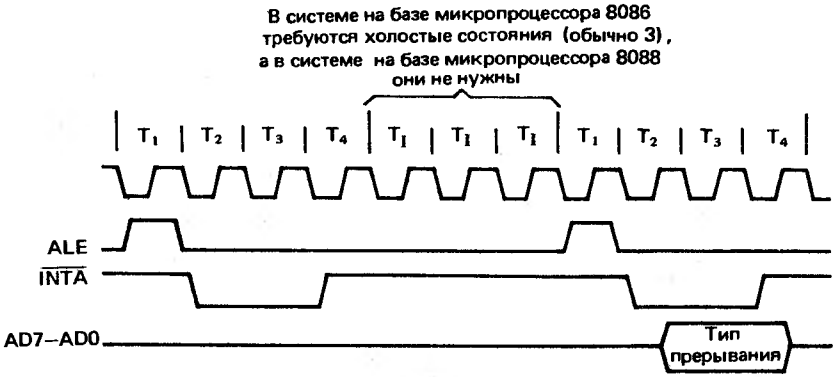


Рис. 8.13. Подтверждение прерывания

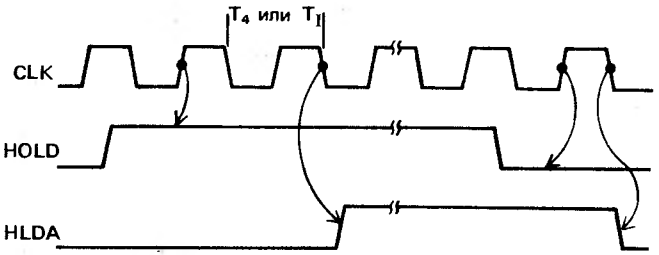


Рис. 8.14. Временная диаграмма запроса и разрешения шины в системе минимального режима

ные снимаются. В любой операции сигнал $\overline{DEN}$ снимается в такте T_4 для запрещения приемопередатчиков, а сигнал $M/\overline{IO}$ в этом такте или в последнем такте T_I устанавливается в соответствии со следующей передачей.

Диаграмма шины разработана так, чтобы участвующие в передаче интерфейсы памяти или ввода-вывода могли управлять выдачей данных или восприятием их с шины. Для этого интерфейс посылает в процессор (возможно, через генератор 8284A) сигнал READY, когда он выдал или воспринял данные. Если процессор не получил к началу T_3 сигнал READY, он вводит между T_3 и T_4 такты ожидания T_W до восприятия сигнала READY. Действия на шине в течение T_W такие же, как и в такте T_3. Подаваемый на вход RDY генератора 8284A сигнал готовности вызывает появление сигнала READY в процессор в момент среза текущего такта. Следовательно, чтобы избежать состояния ожидания, входной сигнал RDY должен быть получен до начала такта T_3.

Временная диаграмма подтверждения прерывания показана на рис. 8.13. Если в предыдущем цикле шины распознан запрос прерывания и команда завершена, в текущем и следующем тактах шины на выходе $\overline{INTA}$ формируются отрицательные импульсы. Они длятся от такта T_2 до такта T_4. Реагируя на второй импульс, воспринимающий подтверждение интерфейс помещает

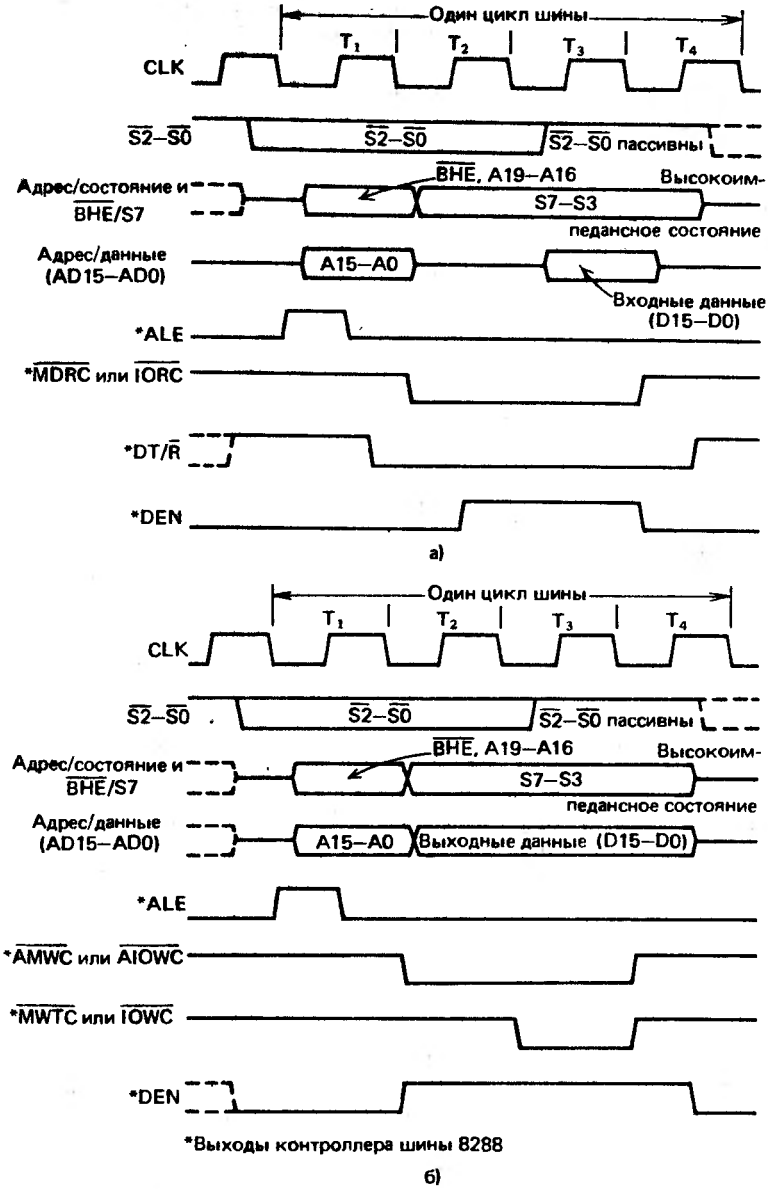


Рис. 8.15. Временные диаграммы шины микропроцессора 8086 в максимальном режиме: *а* – ввод; *б* – вывод

Примечание. В микропроцессоре 8088 сигнал $\overline{BHE}$ отсутствует, а данные передаются только по линиям AD7-AD0

тип прерывания на линии AD7-AD0, которые остальное время в течение двух циклов находятся в высокоимпедансном состоянии. Тип прерывания находится на шине от такта T_2 до такта T_4.

На рис. 8.14 показана диаграмма запроса и разрешения шины в системе с минимальным режимом работы. Сигнал на входе HOLD проверяется по фронту каждого импульса синхронизации. Если процессор воспринял сигнал HOLD до состояния T_4 или в состоянии T_I, он выдает сигнал HLDA и последующие циклы шины передаются запрашивающему ведущему шины до тех пор, пока он не снимет запрос. Снятие запроса обнаруживается по фронту следующего такта синхронизации, и сигнал HLDA снимается по срезу сигнала синхронизации этого такта. Когда сигнал HLDA = 1, все тристабильные выходы процессора переводятся в высокоимпедансное состояние. Команды, находящиеся во внутренней очереди команд, продолжают выполняться до тех пор, пока какая-нибудь из них не потребует цикла шины. Команда MOV AX,BX выполнится полностью, а команда MOV AX,NUMBER выполняется до момента, когда необходимо передавать данные из ячейки NUMBER.

Временные диаграммы передач ввода и вывода системы, работающей в максимальном режиме, представлены на рис. 8.15. Состояния бит $\overline{S0}$, $\overline{S1}$ и $\overline{S2}$ устанавливаются перед началом цикла шины. При обнаружении перехода из пассивного состояния $\overline{S0}$ = S1 = S2 = 1 контроллер 8288 выдает в такте T_1 импульс ALE и необходимый уровень сигнала DT/$\overline{R}$. В такте T_2 контроллер устанавливает DEN = 1, разрешая приемопередатчики, и при вводе активизирует сигналы $\overline{MRDC}$ или $\overline{IORC}$, которые сохраняются до такта T_4. В операции вывода в тактах T_2-T_4 действуют сигналы $\overline{AMWC}$ или $\overline{AIOWC}$, а в тактах T_3-T_4 – сигналы $\overline{MWTC}$ или $\overline{IOWC}$. Сигналы состояния $\overline{S0}$, S1 и S2 сохраняются активными до такта T_3 и становятся пассивными (единичными) в тактах T_3 и T_4. Как и в минимальном режиме, если до начала T_3 не подается активный сигнал READY, между тактами T_3 и T_4 вводятся состояния ожидания.

Сигналы подтверждения прерывания аналогичны сигналам минимального режима, но от такта T_2 первого цикла шины до такта T_2 второго цикла действует сигнал LOCK = 0. Однако сигналы запросов и разрешений шины действуют по-другому (см. диаграмму $\overline{RQ}/\overline{GT}$ на рис. 8.16). Последовательность запроса/разрешения освобождения шины реализуется тремя импульсами. Сигналы $\overline{RQ}/\overline{GT}$ проверяются по фронту каждого импульса синхронизации,

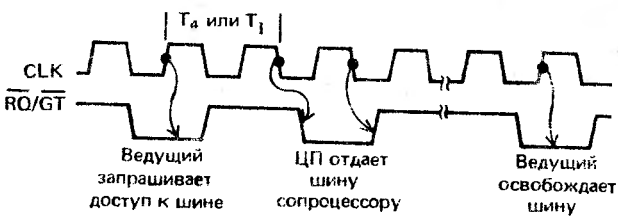


Рис. 8.16. Временная диаграмма запроса и разрешения шины в системе максимального режима

и при обнаружении запроса (при удовлетворении указанных условий) процессор выдает импульс разрешения на $\overline{RQ/GT}$ сразу после следующего такта T_4 или T_1. Когда запрашивающий ведущий принимает этот импульс, он начинает распоряжаться шиной и может управлять ею в течение одного или нескольких циклов шины. При готовности освободить шину ведущий посылает в процессор импульс по той же линии, по которой осуществлялся запрос. Как уже говорилось, линии $\overline{RQ/GT0}$ и $\overline{RQ/GT1}$ аналогичны, но линия $\overline{RQ/GT0}$ имеет больший приоритет.

8.3. УПРАВЛЕНИЕ ПРИОРИТЕТНЫМИ ПРЕРЫВАНИЯМИ

Показанную на рис. 8.1 логику управления приоритетными прерываниями можно реализовать несколькими способами. Она не требуется в системах с программным управлением приоритетами или с простой приоритетной цепочкой, но в более сложных системах для эффективной работы необходимы аппаратные средства управления прерываниями ввода-вывода. Многие фирмы, в том числе и Intel, выпускают микросхемы управления приоритетами. Проблемы совместимости не возникают, если микросхемы ЦП и устройства управления приоритетами выпускаются одной и той же фирмой. Поэтому мы рассмотрим программируемый контроллер прерываний (PIC) 8259A фирмы Intel, который специально предназначен для работы с микропроцессорами 8086/8088.

Микросхема 8259A спроектирована так, что она может работать одна или совместно с несколькими микросхемами. Сначала рассмотрим системы с одним контроллером прерываний, а затем перейдем к системам, имеющим до девяти микросхем.

8.3.1. СИСТЕМА ПРЕРЫВАНИЙ С ОДНИМ КОНТРОЛЛЕРОМ

Микросхема 8259A выпускается в 28-контактном корпусе типа DIP и требует один источник питания напряжением +5 В. Ее организация и подключения в системе с максимальным режимом показаны на рис. 8.17. Ниже даются определения контактов микросхемы:

D7-D0. Для взаимодействия с ЦП по шине данных. В больших системах могут потребоваться шинные драйверы, а в малых системах достаточно прямых соединений.

INT. Для выдачи в ЦП сигнала запроса прерывания.

$\overline{INTA}$. Для приема сигналов подтверждения прерывания от ЦП. Предполагается, что подтверждение состоит из двух импульсов, что делает контроллер совместимым с микропроцессорами 8086/8088.

$\overline{RD}$. Сигнализирует контроллеру поместить на шину данных содержимое регистров IMR, ISR или IMR или приоритетный уровень. Что выдается на шину, зависит от состояния контроллера (см. далее).

$\overline{WR}$. Сигнализирует контроллеру, что он должен принять данные с шины данных и использовать их для установки бит в словах приказов. Распределение принятых данных рассматривается далее.

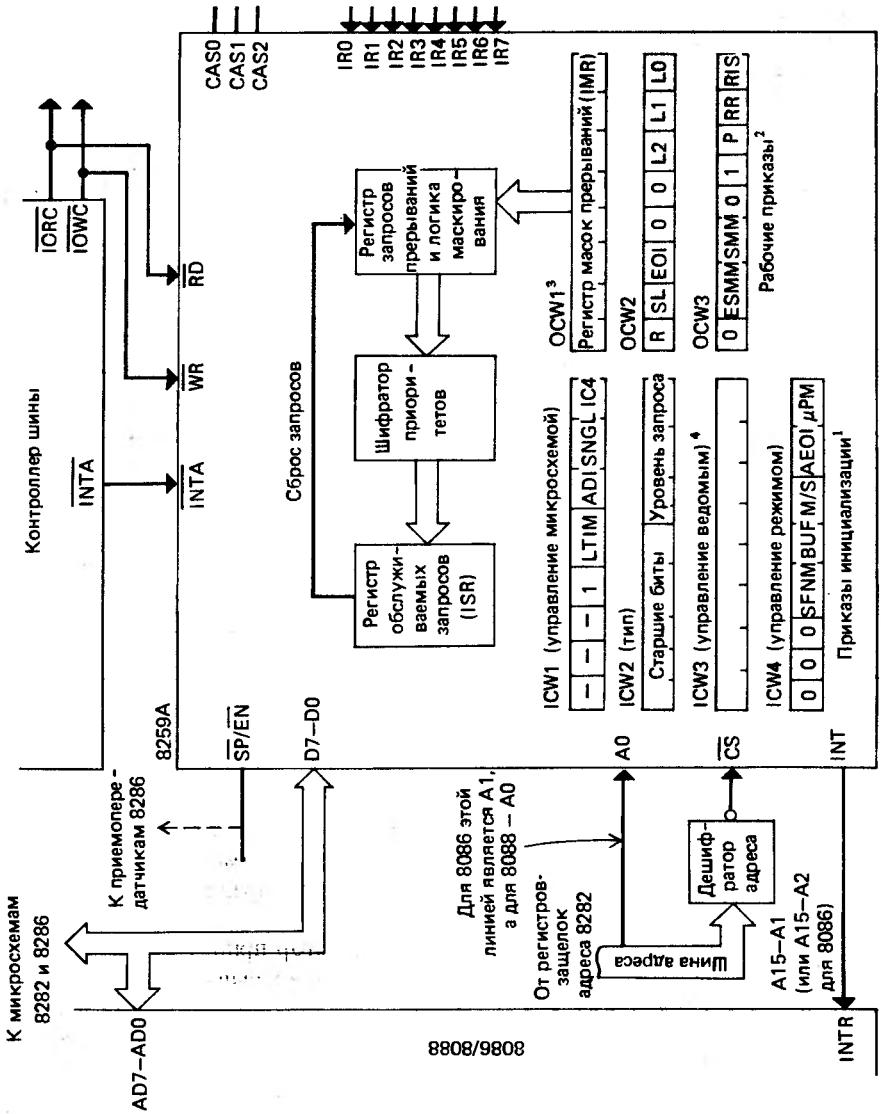


Рис. 8.17. Организация программируемого контроллера прерываний:

[1] A0 = 0 для адресации первого слова (ICW1) и A0 = 1 для адресации последующих слов.

[2] A0 = 1 для адресации первого слова и A0 = 0 для адресации последующих слов.

[3] Биты соответствуют входам IR: бит = 1 означает, что IR замаскирован.

[4] Если 8259A является ведущим, бит = 1 показывает, что соответствующий вход IR подключен к ведомому контроллеру. Для ведомого контроллера биты 3-7 равны 0, а биты 0-2 идентифицируют ведомого.

$\overline{CS}$. Идентифицирует обращение к контроллеру. Подключается к шине адреса через дешифратор, который сравнивает старшие биты адреса 8259A с адресом, находящимся на шине адреса.

A0. Указывает порт 8259A, к которому производится обращение. Для каждого контроллера в адресном пространстве ввода-вывода системы необходимо зарезервировать два адреса.

IR7-IR0. Для восприятия запросов от интерфейсов ввода-вывода или других (ведомых) контроллеров.

CAS2-CAS0. Для идентификации конкретного ведомого контроллера (назначение см. в разделе 8.3.2).

$\overline{SP}$/$\overline{EN}$. Выполняет две функции. Как вход, определяет, каким является контроллер: ведущим ($\overline{SP}/\overline{EN} = 1$) или ведомым ($\overline{SP}/\overline{EN} = 0$). Как выход, осуществляет запрещение приемопередатчиков шины данных, когда данные передаются из контроллера в ЦП. Использование в качестве входа или выхода рассматривается далее.

В системе на базе микропроцессора 8088 два адреса 8259A обычно соседние и линия адреса A0 подключается на вход A0. Так как контроллер имеет всего восемь линий данных, а микропроцессор 8086 всегда вводит указатель прерывания по младшим 8 битам своей 16-битной шины данных, все передачи данных в (из) 8259A должны производиться по младшей половине шины. Наиболее просто гарантировать удовлетворение этого требования: подключить линию A1 на вход A0 и использовать два соседних четных адреса, первый из которых кратен 4. Однако ради упрощения последующего обсуждения в обоих случаях будем считать второй адрес нечетным.

Секция управления 8259A имеет несколько программируемых бит, которые можно считать находящимися в семи 8-битных регистрах. Эти регистры разделены на две группы, первая из которых содержит слова приказов инициализации (ICW), а вторая — слова рабочих приказов (OCW). *Слова приказов инициализации* обычно устанавливаются процедурой инициализации при включении вычислительной системы и при работе не изменяются. *Слова рабочих приказов* применяются для динамического управления обработкой прерываний.

Регистр IRR (и логика маскирования), шифратор приоритетов и регистр ISR предназначены для восприятия и управления прерываниями на входах IR7-IR0. Регистр IRR фиксирует входные запросы и вместе с шифратором приоритетов разрешает незамаскированным запросам с достаточным приоритетом сформировать сигнал 1 на выходе INT. Шифратор приоритетов определяет приоритеты запросов в IRR, а регистр ISR предназначен для хранения текущих обрабатываемых запросов.

После того как бит в IRR установлен в состояние 1, он сравнивается с соответствующим битом маски из IMR. Если бит маски содержит 1, запрос передается в шифратор приоритетов, в противном же случае запрос блокируется. Когда запрос прерывания подается в шифратор приоритетов, проверяется его приоритет, и если в соответствии с текущим состоянием шифратора приоритетов необходимо выдать прерывание в процессор, формируется сигнал INT.

Если флажок IF = 1, процессор реализует последовательность прерывания по завершению текущей команды и возвращает два отрицательных импульса по линии $\overline{\text{INTA}}$. При действии первого импульса триггеры-защелки в IRR запрещаются, поэтому IRR игнорирует последующие сигналы на входах IR7-IR0. Такое состояние сохраняется до окончания второго импульса $\overline{\text{INTA}}$. Кроме того, первый импульс $\overline{\text{INTA}}$ устанавливает соответствующий бит ISR и сбрасывает соответствующий бит в IRR. Второй импульс $\overline{\text{INTA}}$ передает на линии D7-D0 текущее содержимое ICW2 и этот байт процессор использует как тип прерывания. Если бит автоматического окончания прерывания (AEOI) в ICW4 содержит 1, по окончанию второго импульса $\overline{\text{INTA}}$ сбрасывается бит ISR, который был установлен первым импульсом $\overline{\text{INTA}}$; в противном случае бит ISR не сбрасывается до выдачи в OCW2 соответствующего приказа окончания прерывания (EOI).

Как уже говорилось, слова приказов инициализации обычно загружает процедура инициализации при включении системы и они содержат биты управления, не изменяющиеся при работе системы. Контроллер 8259A имеет четный адрес (A0 = 0) и нечетный адрес (A0 = 1) и слова приказов инициализации должны загружаться последовательно с использованием четного адреса для ICW1 и нечетного для остальных ICW.

Биты ICW1 определяются следующим образом:

Биты 7-5. Предназначены только для системы на базе микропроцессоров 8080/8085.

Бит 4. Всегда установлен в 1. Он направляет принятый байт в ICW1 в отличие от OCW2 или OCW3, которые также используют четный адрес (A0 = 0).

Бит 3 (LTIM). Определяет режим запуска фронтом (LTIM = 0) или уровнем (LTIM = 1). Режим запуска фронтом вызывает сброс бита IRR, когда устанавливается соответствующий бит ISR.

Бит 2 (ADI). Предназначен только для систем на базе микропроцессоров 8080/8085.

Бит 1 (SNGL). Показывает, каскалируется ли 8259A с другими контроллерами. Бит SNGL = 1, когда в системе прерываний имеется один контроллер.

Бит 0 (IC4). Устанавливается в 1, если в последовательности инициализации выводится ICW4. В системе с микропроцессорами 8086/8088 этот бит должен всегда содержать 1, так как бит 0 в ICW4 должен быть установлен в 1.

Биты 7-3 ICW2 загружаются из соответствующих бит второго байта, выводимого ЦП при инициализации 8259A, а биты 2-0 устанавливаются в соответствие с уровнем приоритета запроса прерывания, например запрос на линии IR6 загружает в эти биты код 110. Слово ICW3 предназначено для систем с несколькими контроллерами прерываний и выводится, если только SNGL = = 0 (см. раздел 8.3.2). Слово ICW4 выводится, если только бит 0 (IC4) установлен в 1; в противном случае содержимое ICW4 сбрасывается. Биты ICW4 имеют следующие определения:

Биты 7-5. Всегда содержат 0.

Бит 4 (SFNM). Если установлен в 1, применяется специальный вложенный режим, предназначенный для систем с несколькими 8259A и рассматриваемый далее.

Бит 3 (BUF). Состояние BUF = 1 означает, что $\overline{SP/EN}$ служит выходом для запрещения системных приемопередатчиков 8286, пока ЦП вводит данные из 8259A. Если приемопередатчиков нет, BUF должен быть сброшен в 0 и в системах с одним контроллером на вход $\overline{SP/EN}$ следует подать 1.

Бит 2 (M/S). Этот бит игнорируется, если BUF = 0. В системах с одним контроллером этот бит должен содержать 1; в противном случае он должен быть в состоянии 1 для ведущего контроллера и в состоянии 0 для ведомых контроллеров.

Бит 1. (AEOI). Если AEOI = 1, в конце второго импульса $\overline{INTA}$ сбрасывается бит ISR, который вызвал прерывание.

Бит 0 (μPM). Состояние μPM = 1 показывает, что контроллер находится в системе на базе микропроцессоров 8086/8088. Нулевое состояние подразумевает систему на базе микропроцессоров 8080/8085.

Типичный фрагмент установки содержимого ICW имеет следующий вид (четный адрес 8259A равен 0080):

```
MOV  AL,13H
OUT  80H,AL
MOV  AL,18H
OUT  81H,AL
MOV  AL,0DH
OUT  81H,AL
```

Первые две команды определяют запуск запросов фронтом, наличие в системе одного контроллера и необходимость вывода ICW4. Следующие две команды задают 5 старших бит типа прерывания равными 00011. Слово ICW3 не выводится, так как SNGL = 1; следовательно, последние две команды определяют ICW4 = 0D, которое сообщает следующую информацию: специальный вложенный режим не применяется, сигнал $\overline{SP/EN}$ используется для запрещения приемопередатчиков, контроллер является ведущим, для сброса бита ISR необходим приказ EOI, контроллер 8259A работает в системе на базе микропроцессоров 8086/8088.

Имеется три слова OCW рабочих приказов. Слово OCW1 применяется для маскирования запросов прерываний; если бит маски, соответствующий запросу прерывания, содержит 1, запрос блокируется. Слова OCW2 и OCW3 предназначены для управления режимом контроллера и приема приказов EOI. В OCW1 байт выводится с указанием нечетного адреса 8259A, а в OCW2 и OCW3 – с указанием четных адресов. Слово OCW2 отличается от OCW3 содержимым бита 3 байта данных. Если бит 3 содержит 0, байт помещается в OCW2, а если он содержит 1 – в OCW3. Оба слова OCW2 и OCW3 отличаются от ICW1, которое также использует четный адрес, значением бита 4 данных. Если бит 4 равен 0, то байт помещается в OCW2 или OCW3 в зависимости от состояния бита 3. Неоднозначности интерпретации слов ICW2, ICW3, ICW4 и OCW1, использующих нечетный адрес, не возникает, так как слова инициализации должны всегда следовать за ICW1, как этого требует последовательность инициализации, и в середине этой последовательности вывод в OCW1 производить нельзя.

Обратимся к рис. 8.17. Биты L2-L0 в OCW2 обозначают уровень IR, бит 5 предназначен для задания приказов EOI, а биты 6 и 7 управляют уровнями IR. Напомним, что, когда бит AEOI в ICW4 содержит 1, установленный запросом бит ISR автоматически сбрасывается в конце второго импульса $\overline{\text{INTA}}$. Если же AEOI = 0, бит ISR необходимо явно сбрасывать приказом EOI, который заключается в выдаче OCW2 с установленным в 1 битом 5. Когда выдается приказ EOI, четыре возможные комбинации бита 7 (бит R – ротации) и бита 6 (бит SL – установки уровня) приведены в табл. 8.3.

Таблица 8.3

Интерпретация бит R и SL

R	SL	Действие
0	0	Режим обычных приоритетов
0	1	Сбрасывает бит ISR, определяемый полем L2-L0
1	0	Циклически изменяет приоритеты влево на одну позицию
1	1	Назначает позиции, определяемой полем L2-L0, низший приоритет с циклическим изменением остальных приоритетов

Биты OCW2 сохраняются в 8259A только на время выполнения определяемых ими действий. Это положение особенно важно для бита EOI. Рассмотрим перечисленные возможности несколько подробнее, начиная с режима обычных приоритетов.

Обычно запрос на линии IR0 имеет наивысший приоритет, на линии IR1 – следующий меньший и т. д. Когда появляется первый импульс $\overline{\text{INTA}}$, шифратор приоритетов разрешает только незамаскированному запросу с наибольшим приоритетом установить свой бит ISR. Так как три младших бита ICW2 (слово ICW2 указывает тип прерывания и определяет адрес указателя прерывания) определяются тем, какой бит ISR установлен, то и адрес процедуры прерывания зависит от установленного бита ISR. Следовательно, первой начинается процедура прерывания устройства, подключенного к входу IR с наибольшим приоритетом, а остальные запросы должны ожидать разрешения дальнейших прерываний.

В режиме обычных приоритетов при установленном бите ISRn шифратор приоритетов не распознает запросов на линиях IR7-IR(n + 1), но распознает незамаскированные запросы на входах IR(n – 1)-IR0. Следовательно, если в процессоре флажок IF = 1, запросы с приоритетами выше обрабатываемого вызывают прерывание текущей процедуры прерывания, а запросы с меньшими приоритетами ожидают и обрабатываются в соответствии со своими приоритетами по мере сброса бит ISR с большими приоритетами. Если AEOI = 1, соответствующий прерыванию бит ISR автоматически сбрасывается в конце второго импульса $\overline{\text{INTA}}$. Когда же AEOI = 0, бит ISR должна сбрасывать процедура прерывания посредством установки бита 5 в OCW2.

Рассмотрим пример режима обычных приоритетов, приняв первоначально, что AEOI = 0 и все биты ISR и IMR сброшены. Предположим также, что, как

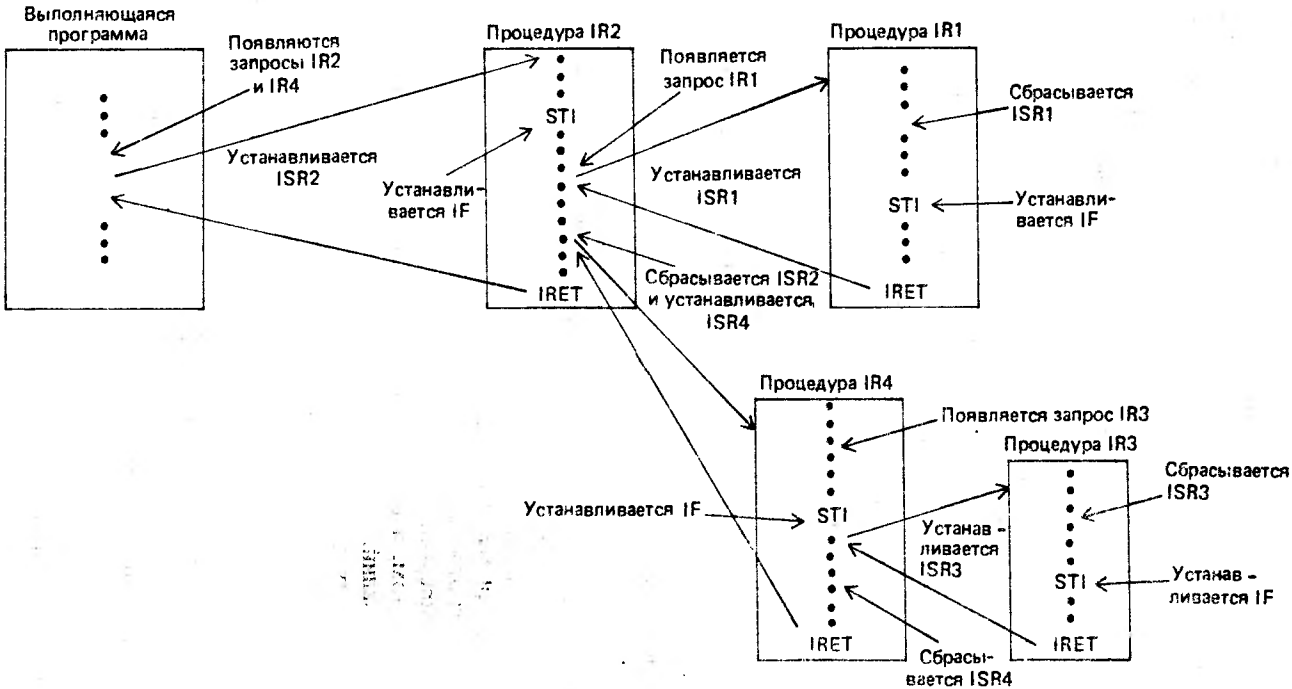


Рис. 8.18. Действия в режиме обычных приоритетов при возникновении типичной последовательности прерывания

показано на рис. 8.18, одновременно появляются запросы IR2 и IR4, затем появляется запрос на IR1 и последним появляется запрос на IR3. Сначала устанавливается бит ISR2 и начинает выполняться процедура прерывания, ассоциируемая с запросом IR2. После того как эта процедура устанавливает IF = 1 и появляется запрос IR1, устанавливается бит ISR1 и полностью выполняется процедура обслуживания запроса IR1. При своем выполнении она должна установить IF = 1 и выдать необходимый приказ для сброса ISR1. При возврате в процедуру IR2 сбрасывается бит ISR2. Затем устанавливается ISR4 и начинается его процедура, в течение которой возникает запрос IR3. Он подтверждается сразу же после установки IF = 1, устанавливается бит ISR3 и инициируется процедура обслуживания IR3. До своего завершения эта процедура должна сбросить ISR3 и установить IF = 1. Осуществляется возврат в процедуру IR4, которая перед возвратом в процедуру IR2 должна сбросить ISR4. Процедура IR2, которая уже сбросила бит ISR2, осуществляет обычный возврат в прерванную программу. (Отметим, что если флажок IF не устанавливается в самой процедуре, дальнейшие прерывания не обрабатываются до завершения процедуры, т. е. до выполнения команды IRET.)

Единичное состояние бита 5 в OCW2 обычно осуществляет сброс бита ISR с максимальным приоритетом (т. е. последнего установленного бита ISR). Но бит ISR можно сбросить явно, выдавая OCW2, в котором биты R, SL и EOI содержат комбинацию 011, а поле L2-L0 идентифицирует номер сбрасываемого бита. Если, например, в OCW2 посылается байт 01100011, сбрасывается бит ISR3.

Кроме рассмотренного режима обычных приоритетов, приказ OCW2 может циклически изменять приоритеты, назначая низший приоритет любому из уровней IR. В этом случае остальные приоритеты изменяются так, как будто обычное упорядочивание " поворачивается вкруговую". Если, например, низший приоритет назначен IR4, получается следующий порядок приоритетов.

IR5, IR6, IR7, IR0, IR1, IR2, IR3, IR4.

Здесь IR5 "поворачивается" в позицию высшего приоритета. Циклическое изменение определяет комбинация 10 бит R и SL. Если эти биты содержат 11, низший приоритет назначается IR, определяемому полем L2-L0. Если, например, высший приоритет имеет IR5 и в OCW2 загружается байт 10100000, получается следующий порядок приоритетов:

IR6, IR7, IR0, IR1, IR2, IR3, IR4, IR5.

Когда же в OCW2 посылается байт 11100010, приоритеты упорядочиваются таким образом:

IR3, IR4, IR5, IR6, IR7, IR0, IR1, IR2.

Биты R и SL влияют на работу и когда EOI = 0. В этом случае комбинация R = 1 и SL = 0 вызывает автоматическое циклическое изменение приоритетов, когда AEOI = 1, а комбинация R = SL = 0 выключает это действие. Комбинация R = SL = 1 и EOI = 0 назначает низший приоритет запросу, определяемому полем L2-L0, без выдачи приказа EOI = 0. Оставшаяся комбинация R = 0 и SL = 1 не производит никаких действий.

В слове OCW3 биты ESMM (разрешение режима специальной маски) и SMM (режим специальной маски) можно использовать для отмены рассмотренных выше режимов. Если в OCW3 посылается байт, в котором ESMM = = SMM = 1, незамаскированные запросы прерываний обрабатываются по мере их появления (если флажок IF в процессоре содержит 1) и порядок их приоритетов игнорируется. Посылка в OCW3 байта, в котором ESMM = 1 и SMM = = 0, восстанавливает приоритетное упорядочивание запросов. Если, наконец, в OCW3 посылается байт с битом ESMM = 0, бит SMM не действует и режим специальной маски не изменяется.

Бит P (полинга, опроса) переводит контроллер в режим опроса. В этом режиме предполагается, что процессор не воспринимает прерываний (IF = 0) и необходимо опрашивать запросы прерываний в IRR. Когда P = 1, следующий сигнал $\overline{RD}$ вызывает установку соответствующего бита в ISR, как будто получен импульс $\overline{INTA}$, и передает в регистр AL процессора байт со следующим форматом:

I	—	—	—	—	W2	W1	W0

Здесь I = 1 показывает наличие запроса прерывания, а поле W2 = W0 содержит уровень IR прерывания с наибольшим приоритетом. Пусть, например, P = 1, существует такое упорядочивание приоритетов

IR3, IR4, IR5, IR6, IR7, IR0, IR1, IR2

и имеются незамаскированные прерывания на IR4 и IR1. Тогда команда IN AL,80H (где 0080 – четный адрес контроллера) загружает в регистр AL следующий байт:

1	—	—	—	—	1	0	0

Когда P = 0, содержимое IRR или ISR можно считать в регистре AL установкой RR = 1 и выполнением команды IN AL,80H. Если во время выполнения команды IN бит RIS = 0, вводится содержимое IRR, а в противном случае – содержимое ISR. Содержимое IMR можно считать в любой момент времени, пользуясь нечетным адресом контроллера 8259A, например, при прежнем назначении адресов команда IN AL,81H вводит в AL содержимое IMR.

Так как биты OCW3 (кроме бита ESMM) определяют, находится ли контроллер в режиме специальной маски и какая информация выдается на шину данных при считывании, они сохраняются до изменения следующим выводом в OCW3. Например, когда P = 0, RR = 1 и RIS = 0, любое считывание по четному адресу до посылки в OCW3 нового байта осуществляет ввод в процессор содержимого IRR.

В заключение рассмотрим, что произойдет, когда на запросы влияют помехи. На входе IR должен сохраняться высокий уровень до фронта первого импульса $\overline{INTA}$. Если это условие не удовлетворяется, контроллер "моделирует" 1 на входе IR7. Таким образом, если устройство к входу IR7 не подключено, запросы по этой линии показывают наличие помех на других лини-

ях запросов и процедура прерывания IR7 служит процедурой "очистки" помех. Если на вход IR7 подключено устройство, этот способ обнаружения помех все же можно использовать, так как запрос от устройства установит ISR7, а "запрос" помехи на линии IR7 не воздействует на ISR7. Следовательно, процедура IR7 может различить эти два события, считывая ISR и проверяя бит 7.

8.3.2. СИСТЕМА ПРЕРЫВАНИЙ С НЕСКОЛЬКИМИ КОНТРОЛЛЕРАМИ

Схема системы прерываний с несколькими контроллерами 8259A представлена на рис. 8.19. На ней не показаны шинные драйверы, но их можно подключить в соответствии с рис. 8.9. Выход $\overline{SP}/\overline{EN}$ ведущего контроллера подключен к приемопередатчикам шины данных, а на входы $\overline{SP}/\overline{EN}$ ведомых приборов подан уровень 0. Показан только один ведомый 8259A, но аналогично подключаются еще до 7 контроллеров, что обеспечивает максимум 64 линии запросов прерываний. Пра разработке дешифратора адреса каждому контроллеру необходимо назначить свою пару адресов в пространстве ввода-вывода. Драйверы на линиях CAS2-CAS0 могут и не потребоваться в зависимости от расстояний между ведущим и ведомыми контроллерами.

В рассматриваемой системе необходимо инициализировать ведущий и ведомые контроллеры. Ведущий инициализируется, как описано выше, но бит SNGL должен быть 0 и потребуется загружать слово ICW3. В каждый бит ICW3, для которого соответствующий вход IR подключен к ведомому прибору, необходимо записать 1, а в остальные биты записываются нули. Бит SFNM устанавливается в 1 для организации специального вложенного режима. При инициализации ведомых контроллеров бит SNGL должен содержать 0. Таким образом, для каждого ведомого прибора потребуется ICW3, но у каждого прибора это слово несет различное содержание. Слово ICW3 имеет следующий формат:

0	0	0	0	0	ID2	ID1	ID0

Три младших бита определяют код идентификации ведомого контроллера. Он должен совпадать с номером той линии запроса ведущего контроллера, к которой подключается выход INT.

Сигнал INT = 1 ведомого контроллера подается на соответствующий вход IR ведущего 8259A. Если IMR и шифратор приоритетов не блокируют этот сигнал, он посылается в процессор через выход INT ведущего контроллера. Когда процессор возвращает сигнал $\overline{INTA}$, ведущий контроллер не только устанавливает бит ISR и сбрасывает бит IRR, но и проверяет соответствующий бит в ICW3 — возникло прерывание от ведомого прибора или нет. В случае прерывания от ведомого контроллера ведущий выдает на линии CAS2-CAS0 номер уровня IR; в противном случае он помещает содержимое ICW2 на шину данных и не выдает сигналов на линии CAS2-CAS0. Сигнал $\overline{INTA}$ поступает во все ведомые 8259A, но его воспринимает только тот прибор, код идентификации которого соответствует номеру, выданному ведущим прибо-

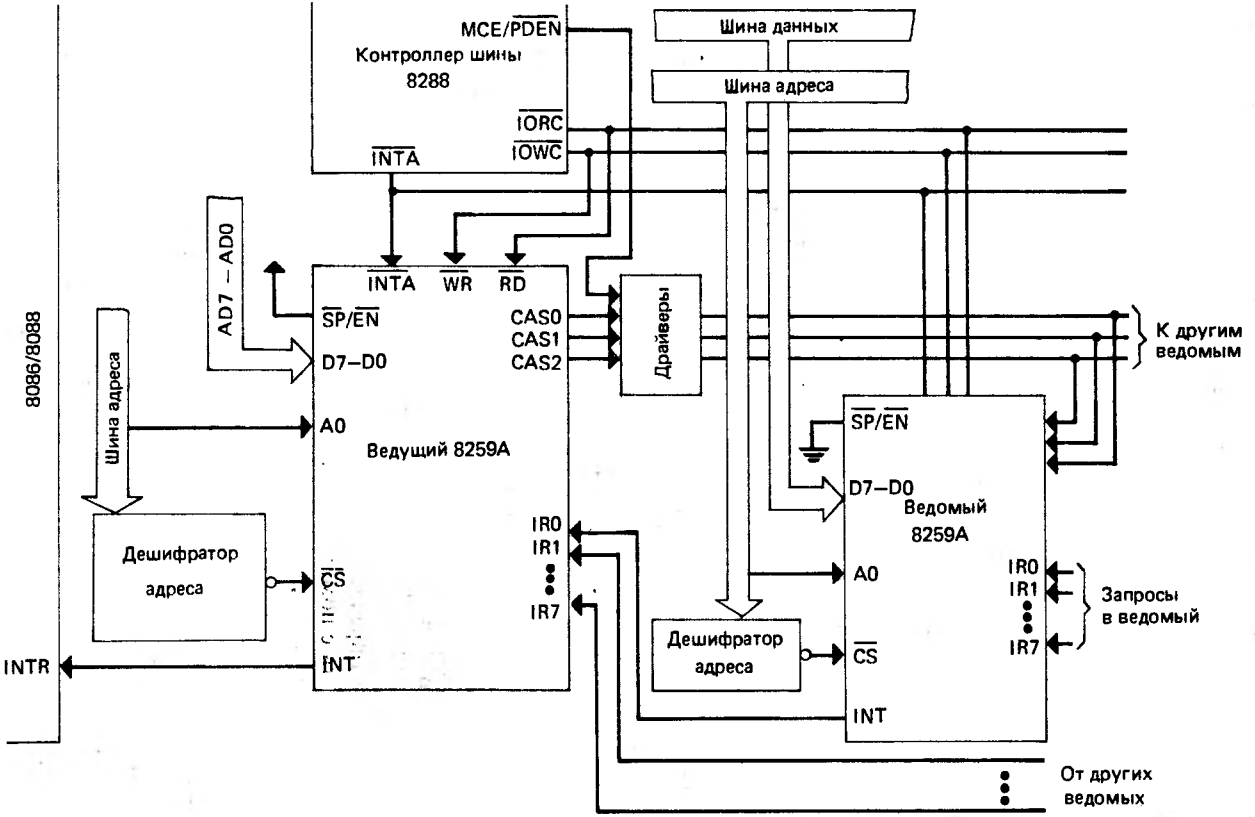
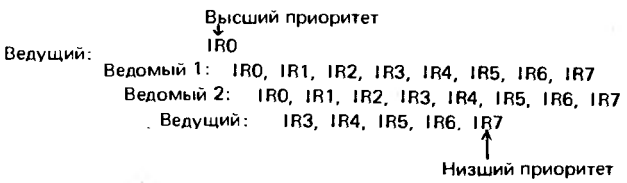


Рис. 8.19. Система прерываний с несколькими контроллерами прерываний

ром на линии CAS2-CAS0. В выбранном ведомом приборе соответствующий бит ISR устанавливается, соответствующий бит IRR сбрасывается, а содержимое его ICW2 выдается на шину данных. Так как ICW2 содержит тип прерывания, в процессе инициализации важно загрузить однозначные комбинации в ICW2 ведущего и ведомых приборов. Для этих приборов требуются приказы EOI, если их биты AEOI содержат 0.

За исключением реакции на сигнал $\overline{\text{INTA}}$ действия всех приборов в системе одинаковы. Аналогично же производится управление их режимами и считывание регистров. Однако имеется одно исключение. Если бит SFNM в ICW4 ведущего контроллера инициализирован на 1, он вводит специальный вложенный режим, который применяется с режимом обычных прерываний и AEOI = 0. В этом случае ведущий контроллер разрешает незамаскированным запросам с достаточным приоритетом проходить на выход INT, даже если соответствующий бит ISR уже установлен в 1. Это означает, что, если в ведомом контроллере появляется запрос с большим приоритетом в то время, когда обрабатываются запросы одного или нескольких ведомых контроллеров, новый запрос сформирует сигнал INT через ведущий контроллер. При использовании специального вложенного режима процедура прерывания может выдавать два приказа EOI. Сначала приказ неконкретного (неадресуемого) EOI выдается в ведомый прибор, который вызвал прерывание, а затем проверяется его ISR. Если и только если ISR содержит нули, в ведущий прибор выдается приказ неконкретного EOI. Пусть, например, ведомый 1 подключен на вход IR1 ведущего, ведомый 2 подключен на вход IR2 ведущего, имеются только эти два ведомых прибора и наибольший приоритет во всех контроллерах назначен IR0. Тогда получается следующий вложенный порядок приоритетов:



Конечно, для блокирования некоторых запросов в ведущем и ведомых контроллерах можно применять маски.

8.4. СТАНДАРТЫ ШИНЫ

Большинство микросистем, включая и мультипроцессорные конфигурации, реализуются с главной системной шиной, к которой подключаются все основные компоненты системы. Системы с такой организацией оказываются достаточно простыми, гибкими и дешевыми. При проектировании изделий учитывается та шина, которая объединяет все устройства. Основные характеристики шины формализуются и образуют так называемый *стандарт шины*. Если семейство изделий пользуются успехом, его стандарт шины получает широкое распространение и другие изготовители могут разрабатывать уст-

ройства, удовлетворяющие требованиям стандарта, особенно если он принят Институтом инженеров по электротехнике и электронике (IEEE). Благодаря наличию множества совместимых устройств пользователь может значительно сократить время разработки системы и уменьшить расходы, выбрав тщательно спроектированную шину. Кратко рассмотрим шину MULTIBUS фирмы Intel, которая получила широкое распространение и многие другие фирмы выпускают совместимые с ней модули. Шина рассчитана на 8- и 16-битные устройства и мультипроцессорные системы с несколькими ведущими шинами. В любой момент времени через шину могут взаимодействовать только два устройства – ведущий и ведомый. Отношение ведущий/ведомый является динамическим и распределением шины управляют специальные сигналы запроса/разрешения. Физически шина MULTIBUS реализуется на печатной плате; к шине через два разъема P1 и P2 подключаются все модули (см. рис. 8.20). Разъем P1 имеет 86 контактов, несущих основные сигналы шины, а необязательный разъем P2 имеет 60 вспомогательных линий, предназначенных, главным образом, для обнаружения и реакции на отказ питания. (Подробная информация о назначении и разводке разъемов содержится в фирменном руководстве.)

Линии P1 разделяются на следующие функциональные группы: адреса; данных; приказов и квитирования; управления доступом к шине; служебные.

Шина MULTIBUS имеет 20 линий адреса $\overline{\text{ADR0}}$ – $\overline{\text{ADR13}}$ (нумерация дается в 16-ричной системе), с помощью которой ведущий шины определяет ячейку памяти или порт ввода-вывода. Стандарт разрешает 16-битному ведущему или микропроцессору 8088 использовать все 20 линий при обращении к

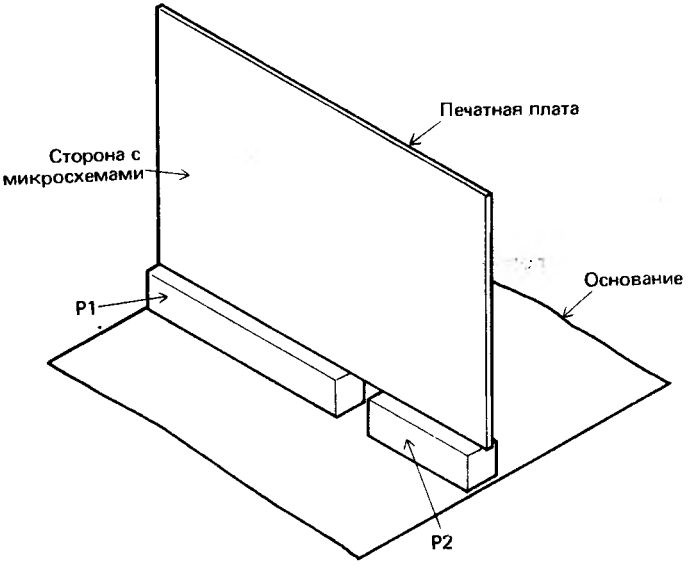


Рис. 8.20. Подключение модуля к шине MULTIBUS

памяти, но только младшие 12 линий адреса ($\overline{ADRB}$ – $\overline{ADR0}$) для выбора порта ввода-вывода, хотя микропроцессоры 8086/8088 допускают 16-битные адреса ввода-вывода. Для 8-битных ведущих допускаются 16 линий адреса памяти ($\overline{ADRF}$ – $\overline{ADR0}$) и 8 линий адреса портов ввода-вывода ($\overline{ADR7}$ – $\overline{ADR0}$).

Шина имеет также 3 линии управления адресом: разрешение старшего байта ($\overline{BHEN}$), запрещение ЗУПВ ($\overline{INH1}$) и запрещение ПЗУ ($\overline{INH2}$). Сигнал $\overline{BHEN}$ используется 16-битными устройствами для указания передачи данных по старшей половине шины данных; это необходимо при передаче слов. Стандарт требует, чтобы отдельные байты всегда передавались по 8 младшим линиям шины; следовательно, любой 16-битный интерфейс должен содержать буфер для обмена байт, чтобы все байтные передачи осуществлялись только по младшим линиям данных. (Так как микропроцессор 8086 ожидает байт на старшей половине шины при активном сигнале $\overline{BHE}$, можно реализовать нестандартные передачи между процессором и памятью.)

Два сигнала запрещения предназначены для перекрытия (оверлея) ЗУПВ, ПЗУ и вспомогательного ПЗУ в общем адресном пространстве. Например, настраивающий загрузчик можно хранить во вспомогательном ПЗУ, а монитор – в ПЗУ. Так как загрузчик нужен только после сброса, при его выполнении можно выдать оба активных сигнала $\overline{INH1}$ и $\overline{INH2}$. Когда же системой управляет монитор, достаточно установить $\overline{INH2} = 1$ и $\overline{INH1} = 0$. Если управление передается пользователю, можно сделать оба сигнала пассивными, что позволяет ЗУПВ при нормальной работе занимать все адресное пространство.

Имеется 16 двунаправленных линий данных ($\overline{DATF}$-$\overline{DAT0}$), только 8 из которых применяются в 8-битных системах. Передачи данных по шине MULTIBUS сопровождаются сигналами квитирования, аналогичными рассмотренным ранее. Линии считывания из памяти ($\overline{MRDC}$), записи в память ($\overline{MWTC}$), считывания ввода-вывода ($\overline{IORC}$) и записи ввода-вывода ($\overline{IOWC}$) определяются так же, как для контроллера шины 8288. Сигнал подтверждения $\overline{XACK}$ выполняет ту же функцию, что и сигнал READY в логике управления шиной, т. е. контролирует окончания передачи. В общем, его может принимать любой ведущий шины. Так как ведущего шины необходимо оповещать о завершении передачи, продолжительность цикла шины варьируется в зависимости от быстродействия ведущего и ведомого устройств. Этот асинхронный принцип позволяет системе взаимодействовать с медленными устройствами, "не принося в жертву" быстрые устройства.

Когда микропроцессоры 8086/8086 работают в максимальном режиме, контроллер шины 8288 генерирует приказы, которые полностью совместимы с электрическими требованиями шины MULTIBUS. Процессоры в минимальном режиме также могут взаимодействовать с шиной MULTIBUS при наличии внешней логики для генерирования необходимых сигналов. В мультипроцессорной системе процессоры 8086/8088 должны работать в максимальном режиме. Чтобы получить инверсные сигналы шины MULTIBUS, следует применять регистры-защелки 8283 и приемопередатчики 8287 (вместо микросхем 8282 и 8286, соответственно). Предусмотрены также 8 линий запросов прерываний ($\overline{INT7}$ – $\overline{INT0}$) и одна линия ($\overline{INTA}$) подтверждения прерывания.

Шесть линий предназначены для использования логикой доступа к шине в мультипроцессорных конфигурациях. Они обычно подключаются к арбитру шины, который обеспечивает в любой момент времени доступ к шине только одного ведущего шины (см. гл. 11). Группа служебных линий состоит из линий земли, питания +5 В, –5 В, +12 В, –12 В и синхронизации.

Вспомогательная шина, которая подключается к печатным платам через разъем P2, является необязательной. Она в основном предназначена для обнаружения и реагирования на отказы питания (но есть и редко используемые линии управления). Когда переменное напряжение уменьшается ниже определенного уровня, блок питания реагирует сигналом отказа питания. Этот сигнал заставляет специальную схему сформировать прерывание в ведущий шины, который запоминает важную информацию текущей программы. После восстановления переменного напряжения та же схема сигнализирует ведущему шины о необходимости инициировать последовательность включения. В ходе этой последовательности процессор восстанавливает информацию и возобновляет выполнение приостановленной программы. При наличии резервного источника питания (обычно аккумулятора) память и другие модули могут оставаться активными во время отказов питания довольно продолжительное время.

Упражнения

1. Пользуясь логической схемой приемопередатчика 8286 на рис. 8.6 и таблицами истинности, проверьте утверждения относительно направления данных, разрешения и запрещения передач данных.

2. Считая, что команда XCHG BL,DATA уже находится в очереди и готова к выполнению, постройте временные диаграммы действий шины при ее выполнении (аналогичные диаграммам на рис. 8.12, *а* и 8.15) для каждого из следующих случаев:

а) микропроцессор 8086 в минимальном режиме без состояний ожидания;

б) микропроцессор 8088 в минимальном режиме без состояний ожидания;

в) микропроцессор 8086 в максимальном режиме без состояний ожидания;

г) микропроцессор 8086 в максимальном режиме, если в середине T_3 принимается сигнал готовности продолжительностью в два такта синхронизации (приведите также временную диаграмму сигнала готовности);

д) микропроцессор 8086 в максимальном режиме без состояний ожидания, но при передаче ввода обнаруживается запрос шины и снимается только через один цикл шины.

3. Рассмотрите систему на базе микропроцессора 8086 в максимальном режиме, которая выполняет команды:

```
MOV  AX,DATA  (DATA имеет четный адрес)
CMP  AX,1
```

Предположите, что при выполнении первой команды появляется запрос прерывания, и дайте временную диаграмму действий шины от начала первой команды до приема процессором типа прерывания.

4. Считая, что команда уже находится в очереди, определите число циклов шины, необходимых для выполнения каждой из следующих команд:

а) PUSH AX

б) CALL NEAR PTR PROC_A

в) CALL FAR PTR PROC_B

г) MOV DATA,AX (четный адрес)

д) MOV DATA,AX (нечетный адрес)

е) OUT 60H,AX

5. Чему равно минимальное число циклов шины между моментом распознавания запроса прерывания и выборкой первой команды процедуры прерывания?

6. Приведите временную диаграмму микропроцессора 8086, которая показывает сигналы на линии синхронизации и всех линиях адреса/данных при выполнении команды ADD AX,TAX. Сначала считайте, что TAX имеет четный адрес, а затем – нечетный.

7. Пусть команда

```
JMP  NEAR  PTR BEGIN
```

уже находится в очереди и переход осуществляется к команде

```
BEGIN:  CMP  AL,0
```

Опишите действия в каждом цикле шины от начала выполнения команды JMP до окончания выполнения команды CMP. Дайте ответы для каждого из следующих случаев:

а) система на базе микропроцессора 8088;

б) система на базе микропроцессора 8086 и BEGIN является четным адресом;

в) система на базе микропроцессора 8086 и BEGIN является нечетным адресом.

8. Напишите последовательность инициализации для единственного контроллера 8259A в системе на базе микропроцессора 8086. Четный адрес контроллера равен 2130. Эта последовательность должна определять:

запросы запускаются уровнем,

тип прерывания для запроса IR0 равен 28,

$\overline{SP}/\overline{EN}$ выводит сигнал запрещения в приемопередатчики шины данных,

биты ISR автоматически сбрасывается в конце второго импульса $\overline{INTA}$,

IMR сбрасывается.

9. Напишите фрагмент, устанавливающий следующие приоритеты в 8259A (четный адрес его равен 08A0):

IR4, IR5, IR6, IR7, IR0, IR1, IR2, IR3.

Сделайте упражнение два раза, первый раз считая, что текущий наивысший приоритет имеет IR0, а второй – IR3.

10. Напишите команды, которые передают содержимое IRR, ISR и IMR контроллера 8259A (четный адрес 0500) и смежные ячейки памяти, начинающиеся в REG_ARR.

11. Напишите команды, которые блокируют запросы на IR3, IR4 и IR6 в контроллере 8259A (четный адрес равен 1208).

12. Предположим, что IF = 0. Приведите команды опроса 8259A (четный адрес 1000) и перехода к *n*-му смещению в массиве из 8 смещений, где *n* – номер линии запроса прерывания с наибольшим приоритетом, на которой действует запрос. При отсутствии запроса выполнение продолжается в естественном порядке.

13. Заданы приведенная ниже последовательность событий, режим обычных приоритетов, необходимость вывода приказов EOI:

а) Запрос на IR3

б) Запрос на IR2

в) Запрос на IR6

г) Флажок IF устанавливается в 1

д) Флажок IF устанавливается в 1

е) EOI для сброса ISR2

ж) EOI для сброса ISR3

з) Флажок IF устанавливается в 1

и) EOI для сброса ISR6

Постройте диаграмму, аналогичную рис. 8.18.

14. Рассмотрим систему на базе микропроцессора 8086, которая имеет ведущий 8259A и один ведомый 8259A, который подключен на вход IR1 ведущего контроллера. Четные адреса контроллеров равны 0580 (ведущий) и 0582 (ведомый). Тип прерывания для IR0 ведущего прибора равен 30, а для IR0 ведомого – 38; все запросы запускаются фронтом. Для сброса бит ISR применяются приказы EOI. В обоих контроллерах регистры IMR сброшены и $\overline{SP}/\overline{EN}$ используется как вход. Приведите фрагмент инициализации системы прерываний.

15. Для системы из упр. 14 напишите фрагмент для сброса соответствующих бит ISR после того, как ведомый контроллер сформировал запрос прерывания. Напомним, что необходимо проверить нулевое содержимое ISR ведомого контроллера до сброса ISR1 в ведущем контроллере.

16. Рассмотрите систему прерываний, содержащую ведущий контроллер и три ведомых прибора, причем ведомые подключены на входы IR2, IR3 и IR6 ведущего контроллера. Предположим, что IMR в ведущем приборе содержит 01010000, а во всех ведомых контроллерах регистры IMR сброшеиы. Во всех контроллерах используются обычные приоритеты, кроме ведомого 8259A на входе IR3, в котором линией запроса с наивысшим приоритетом является IR5. Перечислите линни незамаскированных запросов в порядке убывания их приоритетов. Повторите упражнение для случая, когда в ведущем контроллере иаивысший приоритет имеет линия IR3.

9. ИНТЕРФЕЙСЫ ВВОДА-ВЫВОДА

Интерфейсы памяти и ввода-вывода связаны с логикой управления шиной. Между ней и интерфейсами находятся только электрические проводники шины; следовательно, интерфейсы должны быть спроектированы для передачи и приема сигналов, совместимых с логикой управления шиной и ее временной диаграммой. При наличии сходства интерфейсов памяти и ввода-вывода между ними имеются и существенные различия. В данной главе рассматриваются интерфейсы ввода-вывода, а памяти и ее интерфейсам посвящена гл. 10.

Интерфейс ввода-вывода должен выполнять следующие функции:

1. Интерпретировать сигналы адреса и выбора между памятью и вводом-выводом, чтобы определить обращение к нему, и в случае такого обращения определить, к каким регистрам происходит обращение.

2. Определять, выполияется ввод или вывод; при выводе воспринять с шины выходные данные или управляющую информацию, а при вводе поместить на шину входные данные или информацию о состоянии.

3. Вводить или выводить данные в подключенное устройство ввода-вывода и преобразовывать параллельные данные в формат, воспринимаемый устройством, или наоборот.

4. Посылать сигнал готовности, когда данные восприняты или помещены на шину данных, информируя процессор о завершении передачи.

5. Формировать запросы прерываний и (при отсутствии в логике управления шиной управления приоритетными прерываниями) принимать подтверждения прерываний и выдавать тип прерывания.

6. Принимать сигнал сброса и реинициализировать себя и, возможно, подключенное устройство.

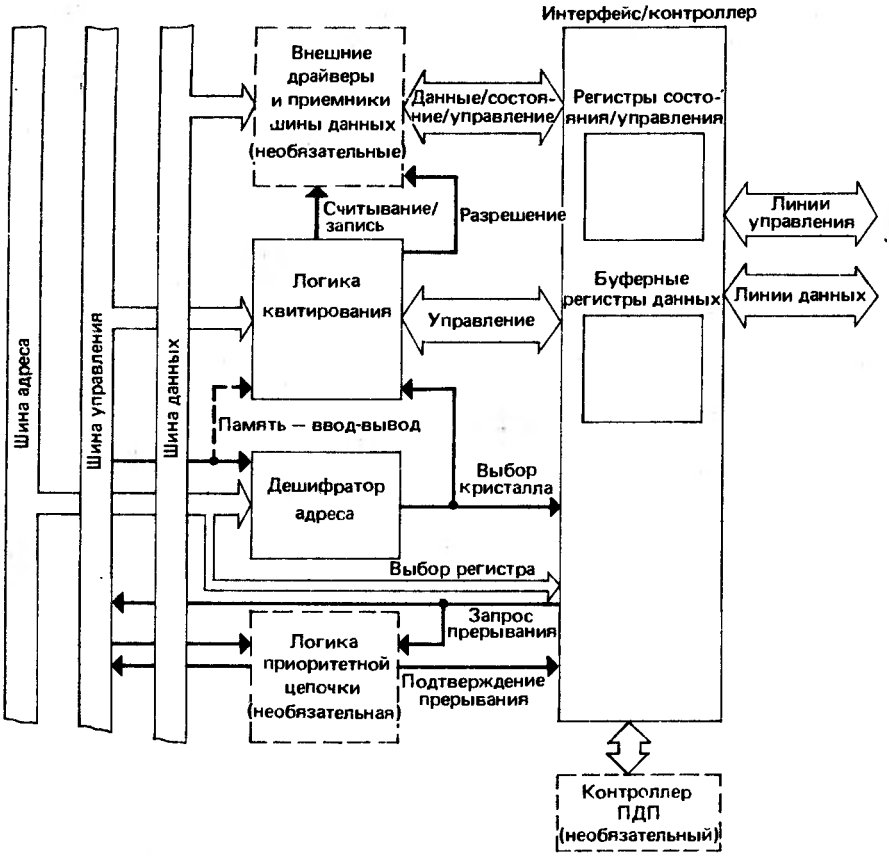


Рис. 9.1. Схема типичного интерфейса ввода-вывода

Схема типичного интерфейса ввода-вывода показаиа на рис. 9.1. Главные функции интерфейса сводятся к преобразованию сигналов между системной шиной и устройством ввода-вывода и реализации буферов, необходимых для удовлетвореиия двух наборов временных ограничений. Значительная часть функций интерфейса выполняется блоком, находящимся иа рисунке справа. Часто он реализуется в виде микросхемы, но иногда функции этого блока могут быть разбросаны по нескольким приборам. Очевидно, его фуикции полиостью определяются устройством ввода-вывода, с которым должен взаимодействовать интерфейс.

Интерфейс можно разделить на две части, взаимодействующие с устройством и с системной шиной. Первая из них определяется устройством, а вторые части всех интерфейсов в данной системе довольно похожи, так как они связаны с одной и той же шиной. В иих должны быть шинные драйверы и приемники, схемы преобразования интерфейсных сигналов управления в соответствующие квитирующие сигналы и схемы для дешифрирования появляющихся на шине адресов. В системах с микропроцессорами 8086/8088 подключение к шине данных можно осуществить приемопередатчиками

8286, которые применяются и в логике управления шиной. Однако основные интерфейсные устройства имеют встроенные драйверы и приемники, которых достаточно в небольших, одноплатных системах.

Логику квитирования нельзя спроектировать, не зная управляющих сигналов, необходимых основному интерфейсному устройству, а эти сигналы в различных интерфейсах варьируются. Обычно эта логика должна воспринимать сигналы считывания/записи, определяющие направление передачи, и выдавать для микросхем 8286 сигналы $\overline{OE}$ и T. В максимальном режиме в нее подаются сигналы $\overline{IOWC}$ (или $\overline{AIOWC}$) и $\overline{IORC}$ от контроллера шины, а в минимальном режиме – сигналы $\overline{RD}$, $\overline{WR}$ и M/$\overline{IO}$ (или IO/$\overline{M}$). Через эту логику должны также проходить линии запроса прерывания, готовности и сброса. Иногда управляющие линии шины проходят через логику квитирования неизменными (т. е. подключаются прямо к основному интерфейсному устройству).

Дешифратор адреса должен принимать адрес и, возможно, бит, показывающий нахождение адреса в адресном пространстве ввода-вывода или в адресном пространстве памяти. В системе с минимальным режимом этот бит можно взять с линии M/$\overline{IO}$ (или IO/$\overline{M}$), а в системе с максимальным режимом выбор памяти или ввода-вывода определяется линиями $\overline{IOWC}$ и $\overline{IORC}$. Когда дешифратор определяет обращение к интерфейсу, дешифратор должен выдать в основное интерфейсное устройство сигналы о том, что оно выбрано и к какому регистру производится обращение. Определяющие регистр биты могут быть младшими битами адреса, но часто они генерируются в интерфейсном устройстве из сигналов считывания/записи и адреса. Пусть, например, имеется два регистра A и B, из которых можно считывать, и два регистра C и D, в которые можно записывать. Иногда для задания регистра можно использовать сигналы считывания и записи и бит 0 шины адреса следующим образом:

Запись	Считывание	Бит 0 адреса	Выбираемый регистр
0	1	0	A
0	1	1	B
1	0	0	C
1	0	1	D

Если в системе вместо контроллера прерываний применяется приоритетная цепочка, в каждом интерфейсе потребуются логика цепочки (см. рис. 6.14, *б*) и логика формирования типа прерывания. Кроме того, интерфейс может быть связан с контроллером ПДП.

Многие интерфейсы рассчитаны на обнаружение минимум двух видов ошибок. Так как связывающие интерфейс с устройством линии почти всегда подвержены помехам, при передаче информационных байт к ним обычно добавляются биты паритета. При использовании четного паритета бит паритета устанавливается так, чтобы общее число единиц, включая и бит паритета, было четным. В случае нечетного паритета число единиц должно быть нечетным. По мере приема байт осуществляется контроль паритета и при наличии ошибки устанавливается определенный бит в регистре состояния. Некоторые интерфейсы контролируют избыточные байты, которые помещаются после блоков данных. Второй вид ошибок, которые могут обнаруживать большинство интерфейсов, – это *ошибка перегрузки*. Когда компьютер вводит данные, он считывает их из буферного регистра входных данных. Если по какой-то причине содержимое этого регистра заменяется новыми данными до того, как они введены компьютером, возникает ошибка перегрузки. Такая же ошибка возникает, когда данные помещаются в буферный регистр выходных данных до того, как в устройство передано текущее содержимое регистра. Ошибка перегрузки, как и ошибка паритета, вызывает установку определенного бита состояния.

Интерфейсы можно классифицировать в соответствии со способом взаимодействия их со своими устройствами ввода-вывода. В § 9.1 рассматриваются интерфейсы таких устройств, которые передают и принимают информацию последовательно; затронут также вопрос о связи на большие расстояния; § 9.2 посвящен интерфейсам, которые взаимодействуют с устройствами в параллельном формате. В § 9.3 обсуждаются интерфейсы, обеспечивающие синхронизацию внешних или внутренних событий или подсчитывают внешние события. В § 9.4 показано проектирование интерфейса устройства с клавиатурой и индикатором. Наконец, в § 9.5, 9.6 речь идет о быстродействующей связи с обсуждением контроллеров ПДП и накопителя на гибком диске.

В главу включены несколько примеров, в которых описываются интерфейсные микросхемы фирмы Intel, имеющие только восемь линий данных. Поэтому проще считать, что в примерах используется микропроцессор 8088. Микросхемы можно подключать и к 16-битной шине микропроцессора 8086, но так как в нем байты с нечетными адресами передаются по 8 старшим битам шины, в интерфейсе потребуются иекоторые модификации и(или) нововведения. Поэтому в § 9.4 – 9.6 предполагается 8-битная шина микропроцессора 8088, а в § 9.7 указаны те изменения, которые необходимо осуществить при подключении устройств к 16-битной шине микропроцессора 8086. Далее, везде, кроме § 9.7, рассматривается система с минимальным режимом работы, а в § 9.7 показаны модификации, необходимые для системы с максимальным режимом работы.

9.1. ИНТЕРФЕЙСЫ ПОСЛЕДОВАТЕЛЬНОЙ СВЯЗИ

Многие устройства ввода-вывода передают информацию в или из компьютера последовательно, т. е. по одному биту, по паре проводников, причем каждый бит занимает определенный временной интервал. Типичная конфигурация последовательного интерфейса показана на рис. 9.2. Регистр состояния содержит информацию о состоянии текущей передачи (например, об ошибках), а регистр управления хранит информацию о режиме работы интерфейса. Буфер входных данных подключен к регистру сдвига с последовательным входом и параллельным выходом. В операции ввода биты по одному подаются в регистр сдвига, а после приема символа информация передается в буферный регистр входных данных и ожидает ввода в ЦП. (Обычно одним данным является буквенно-цифровой символ, который далее называется просто символом, но это не обязательно.) Буферный регистр выходных данных аналогично подключен к регистру сдвига с параллельным входом и последовательным выходом. Вывод осуществляется выдачей данных в буфер выходных данных, передачей их в регистр сдвига и последующим сдвигом данных на последовательную выходную линию.

Хотя имеется несколько способов адресации четырех регистров порта, будем полагать, что из регистра состояния можно только считывать, а в регистр управления – только записывать. Следовательно, активный сигнал на линии считывания идентифицирует либо регистр состояния, либо буферный регистр входных данных, а линию A0 можно использовать для различения этих регистров. Аналогично сигналы записи и A0 допускают выбор одного из остальных двух регистров.

Интерфейс на рис. 9.2 имеет отдельные линии для передачи и приема информации. Когда для двух направлений сигналов применяются различные линии, связь называется *дуплексной*. Такая система может передавать и принимать одновременно. Терминалы и другие последовательные устройства, подключенные к компьютеру дуплексной системой, обычно посылают каждый символ в компьютер без его индикации. Компьютер сразу же посылает символ обратно (как эхо) и терминал индицирует его. Потерь времени при этом не возникает, так как эхо-символы можно передавать одновременно с вводом в компьютер новых символов. Пользователь осуществляет визуальную проверку не только того, что он ударил по правильной клавише, но и того, что компьютер получил

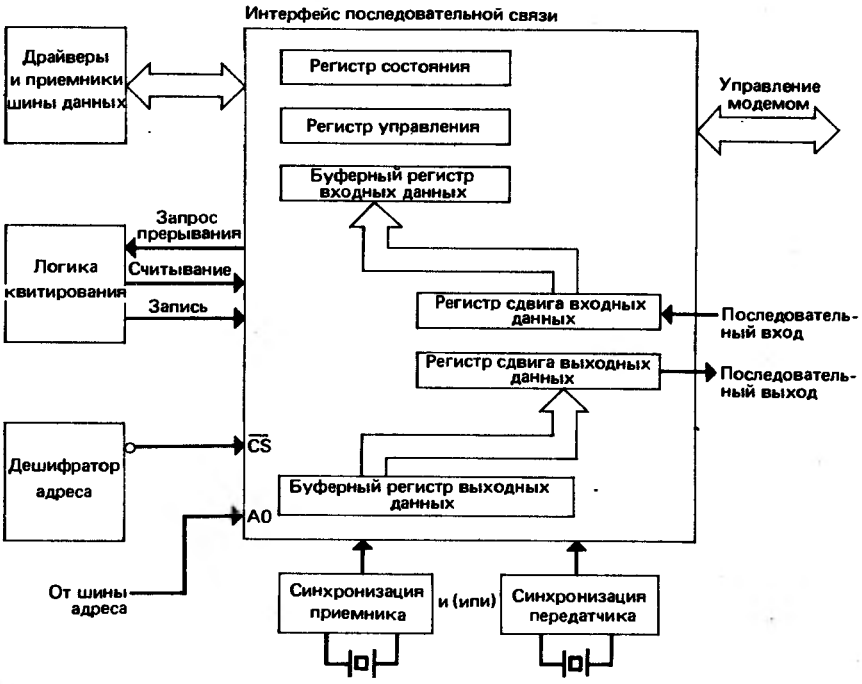


Рис. 9.2. Последовательный интерфейс

правильный символ. В *полудуплексной* связи для ввода и вывода применяется одна и та же линия. Здесь напечатанный на терминале символ индицируется одновременно с передачей в компьютер и принцип эхо-печати не применяется. Если компьютер принимал символы, а затем "хочет" передавать их (или наоборот), линию связи необходимо скоммутировать, на что уходит некоторое время. Хотя дуплексная связь и не требует комму-

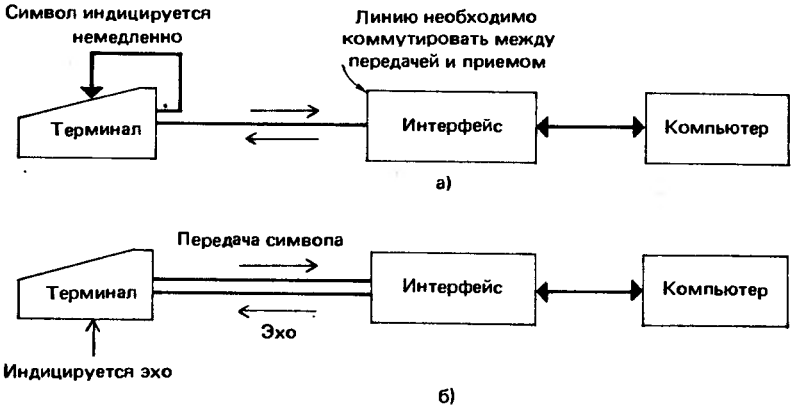


Рис. 9.3. Основные режимы передачи: полудуплексный (*а*) и дуплексный (*б*)

тации и обеспечивает эхо-передачу, для нее нужна дополнительная линия. Два рассмотренных режима связи иллюстрируются рис. 9.3. Многие терминалы и интерфейсы допускают коммутацию для полудуплексной связи и имеют отдельные контакты для дуплексной связи. Если устройство, например принтер, требует только односторонней связи, достаточно полудуплексной связи, а коммутация не нужна.

Имеются два основных вида последовательной связи. При *асинхронной последовательной связи* символы разделяются специальными двоичными наборами, а при *синхронной* должны быть специальные символы "синхронизации" в начале каждого сообщения и специальные "холостые" символы для "заполнения времени", когда информация не передается. Асинхронная передача допускает любые промежутки между символами, а в синхронной передаче символы должны точно размещаться, даже если некоторые символы не содержат информации. Максимальная скорость передачи информации синхронной линии выше, чем асинхронной линии с той же двоичной скоростью, так как при асинхронной передаче каждый символ содержит дополнительные биты. С другой стороны, частоты синхронизации передатчика и приемника могут быть не точно одинаковыми (пока они находятся в допустимых пределах), так как специальные наборы допускают ресинхронизацию в начале каждого символа. При синхронной передаче действия должны быть синхронными, так как именно это определяет положение каждого бита. Следовательно, помимо данных требуется передавать сигналы синхронизации.

9.1.1. АСИНХРОННАЯ СВЯЗЬ

Формат асинхронного символа на рис. 9.4 показывает, что символ содержит закодированную информацию и несколько дополнительных бит. До начала передачи символа линия должна находиться в состоянии 1, которое часто называется состоянием *маркера*. Переход из этого состояния в состояние 0, или *пробел*, отмечает начало символа. Первый бит всегда содержит 0 и называется *стартовым битом*. Затем следуют 5-8 информационных битов, первым из которых является младший бит символа. После информационных бит находится необязательный бит четного или нечетного паритета. Число последних *стоповых бит* может быть 1, $1^1/_2$ или 2.

Хотя число информационных бит, тип паритета (если он есть) и число стоповых бит могут изменяться от одной передачи (т. е. последовательности символов) к другой, эти параметры в одной передаче являются константами. В некоторых интерфейсах параметры программируются с помощью регистров управления, а в других определяются положениями переключателей. Если параметры программируются, в интерфейсе появляется регистр управления, аналогичный показанному на рис. 9.5. В нем биты 0 и 1 определяют

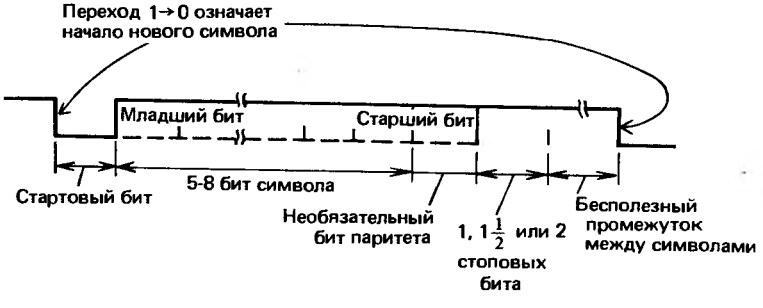


Рис. 9.4. Формат стандартной асинхронной передачи

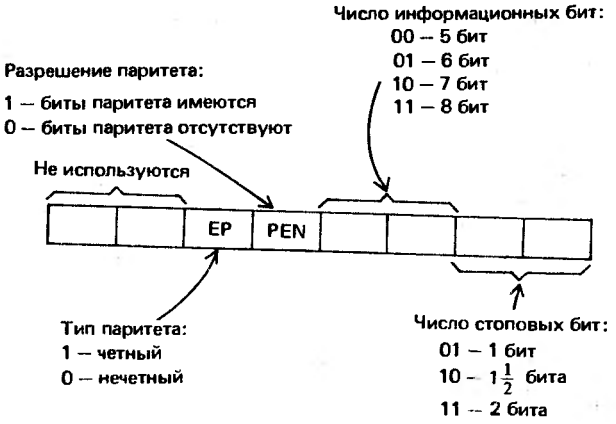


Рис. 9.5. Типичный регистр управления для определения формата символа

число стоповых бит, биты 2 и 3 – число информационных бит, а бит 4 показывает наличие или отсутствие бита паритета. Если бит 4 содержит 1, бит 5 определяет тип паритета.

В передатчике необходим генератор синхронизации, определяющий интервал каждого бита посредством регулирования временных отрезков между сдвигами в регистрах сдвига. После выдвигания всех бит передатчик обычно выводит маркер, пока ие будет готов к передаче следующий символ. В приемнике также необходим генератор синхронизации для измерения интервала между сдвигами, так чтобы входной сигнал опрашивался в правильные моменты. Обычно частоты генераторов в 16, 32 или 64 раза больше двоичной частоты. Если множитель равен 16, после обнаружения перехода $1 \rightarrow 0$ в начале символа приемник должен отсчитать 8 импульсов синхронизации и опросить вход. При обнаружении 0 он считает, что переход вызван стартовым битом, а не помехой. Затем приемник опрашивает вход через интервалы в 16 периодов импульсов синхронизации до ввода всех бит символа, включая и стоповые биты, после чего он прекращает опрос и ожидает следующего перехода $1 \rightarrow 0$. Важно отметить, что ЦП ие выдает и не принимает стартовый и стоповый биты, а также бит паритета. При выводе передатчик вводит эти биты в каждый символ, а при вводе приемник удаляет их из принятых данных.

Такой формат позволяет передатчику вводить между символами промежутки любой длины, а приемнику ресинхронизировать себя в иачале каждого символа. Без этого механизма ресинхронизации два генератора выйдут из режима синхронизации и приемник будет опрашивать вход неправильно. При иаличии ресинхронизации приемник должен учитывать скорость передачи бит только для одного символа. Неправильный опрос возникает только тогда, когда две частоты синхронизации столь различны, что сдвиг в приемнике выполияется в неверный момент времени уже через несколько бит после стартового бита. Если же это происходит, появляется большая вероятность того, что приемник обнаружит нулевой бит на месте стопового. Когда вместо стопового бита обнаруживается 0, возникает так иазываемая *ошибка кадра*. Таким образом, большинство асинхронных последовательных интерфейсов должны обнаруживать три вида ошибок: паритета, перегрузки и кадра.

Сигналы синхронизации передачи и приема, которые определяют временные соотношения в интерфейсе, не обязательно должны быть одними и теми же и не обязательно должны иметь одинаковую частоту. Но имеются очевидные достоинства в единой синхронизации для формирования всех необходимых в интерфейсе импульсов сиихрониза-

ции. Если двоичные скорости передачи и приема различны, электронные схемы на другом конце линии связи должны быть рассчитаны на работу с двумя скоростями. Скорость приема в этих схемах должна соответствовать скорости передачи интерфейса и наоборот.

Устройства, которые выполняют прием и передачу данных в формате рис. 9.4, параллельно-последовательное и последовательно-параллельное преобразования и обнаруживают ошибки паритета, перегрузки и кадра, называются *универсальными асинхронными приемопередатчиками* (УАПП или UART). Многие фирмы выпускают микросхемы УАПП и их разработки стандартизованы. Усилия фирм, выпускающих микропроцессоры, направлены на разработки законченных и универсальных интерфейсов. Иногда УАПП является только небольшой частью интерфейса, который может реализовать большинство видов связи.

9.1.2. СИНХРОННАЯ СВЯЗЬ

Передаваемый синхронно символ также состоит из 5-8 бит с необязательным битом паритета, но не имеет стартового и стоповых бит. Все символы содержат одинаковое число бит n и время передачи разделяется на интервалы из n бит каждый. Опросом в приемнике управляет та же самая синхронизация, которая применяется для генерирования бит, что гарантирует синхронность двух процессов. Передатчик должен передавать символ в течение каждого n-битного интервала. Если символ к началу интервала отсутствует, возникает недогрузка и передатчик вводит холостой символ. В зависимости от системы приемник может интерпретировать холостые символы как *ошибки недогрузки.*

Проблемы запуска или коммутации в асинхронной системе не существует, так как передатчик просто выдает в линию маркер до готовности начала передачи. Если помеха вызывает случайный переход $1 \rightarrow 0$, фиксируется ложный сигнал, когда первый опрос не обнаруживает стартового бита, и система ожидает следующего перехода $1 \rightarrow 0$. В синхронной передаче проблема запуска после включения или другого нарушения оказывается более сложной. Все передачи должны начинаться с серии символов синхронизации, которые нельзя спутать с другими символами. Обычно они совпадают с холостыми символами; в коде ASCII символ синхронизации кодируется как 0010110.

Приемник, который должен знать код символа синхронизации, проверяет каждый бит по мере его появления и, когда последовательность бит точно соответствует битам в символе синхронизации, полагает, что началась передача. Затем он считает следующие символы передаваемой информацией или пытается соотнести один или несколько из них с символом синхронизации. Так как помеха может вызвать ложную идентификацию символа синхронизации, в большинстве систем требуется, чтобы передача начиналась серией символов синхронизации. В этом случае приемник не фиксирует начала передачи до поступления нужного числа символов синхронизации. Ненужные холостые символы и символы синхронизации удаляет приемник или программа ввода.

Допускается программирование числа информационных бит, наличия и вида паритета, числа и кодов символов синхронизации. В интерфейсе потребуются специальные регистры для хранения символа синхронизации и другой информации о формате.

9.1.3. СТАНДАРТЫ ФИЗИЧЕСКОЙ СВЯЗИ

Все более широкое применение компьютеров требует оборудования, обеспечивающего связь человек – компьютер и компьютер – компьютер на длинные и короткие расстояния. Так как в этом вопросе смыкаются интересы фирм, производящих телефонное и связное оборудование и компьютеры, были установлены некоторые определения и стандарты. Они касаются, в основном трех аспектов: скорости передачи, электрических характеристик, определения и обозначения линий.

Скорости передачи измеряются в *битах в секунду* (*bps*) и в *бодах*; бод – это число дискретных условий, передаваемых в одну секунду. Если в любой момент времени может быть только одно из двух возможных условий, обе скорости совпадают. Такая ситуация характерна для машинных интерфейсов, так как они обычно вводят и выводят только единицы и нули; поэтому в данной книге оба термина считаются синонимами. Но следует отметить, что многие линии связи допускают в каждый момент одно из четырех или более условий. Например, сигнал с фазовой модуляцией может иметь одну из 4 фаз; тогда каждая фаза представляет собой два бита и скорость передачи в битах в секунду оказывается в два раза выше скорости в бодах.

Наиболее широко применяются следующие стандартные скорости передачи в бодах: 110, 300, 600, 1200, 1800, 2400, 4800, 9600 и 19 200. Большинство терминалов с дисплеем могут работать с любой из этих скоростей вплоть до 9600 бод, а терминалы с принтерами ограничены быстродействием печатающего механизма. Пишущие машинки обеспечивают скорость 110 бод, а некоторые точечно-матричные принтеры могут принимать до 2400 бод. В большинстве интерфейсов скорости передачи и приема устанавливаются раздельно и часто эти скорости программируются.

В качестве примера рассмотрим асинхронную передачу, в которой каждый символ состоит из стартового бита, 7 информационных бит, бита паритета и одного стопового бита. Если скорость передачи линии равна 1200 бод, то максимальное число символов, которое она может передать в секунду, равно 1200/10 = 120. Максимальная скорость достигается только при отсутствии промежутков между символами. Синхронная же линия со скоростью 1200 бод и без паритета может передать 4 символа синхронизации и сообщение из 100 символов за время

$$7\,(100 + 4)/1200 = 0{,}6067 \text{ с}$$

Следовательно, за секунду можно передать 100/0,6067 = 165 информационных символов.

Стандарты на электрические характеристики и определения линий устанавливают в основном две организации: внутренние – Ассоциация Электрической Промышленности (EIA), и международные – Международный Консультативный Комитет по Телефонии и Телеграфии (CCITT), являющийся подразделением ООН. Наибольший интерес для нас представляет стандарт RS-232-C, аналогичный стандарту CCITT V.24. Он регламентирует передачу последовательных двоичных потоков между интерфейсами или терминалами и связным оборудованием.

Расстояние, на которое можно передать выходной двоичный поток интерфейса или терминала до появления серьезных искажений, зависит от скорости передачи и электрических характеристик линии связи. Типичная зависимость, показывающая максимальное расстояние как функцию скорости передачи для несбалансированной экранированной пары с погонной емкостью 55 пкФ/м, приведена на рис. 9.6. Видно, что максимальное расстояние быстро уменьшается при скорости выше 1000 бод; при скорости 9600 бод оно составляет около 125 м. Если рабочая точка находится ниже кривой на рис. 9.6, связного оборудования не требуется и дуплексное соединение может состоять из трех проводников: передающая линия, приемная линия и общая сигнальная земля.

Если рабочая точка находится выше кривой, необходимо применять связное оборудование, как показано на рис. 9.7. С обеих сторон линии связи располагаются аналогичные устройства для модуляции бит, подаваемых в линию, и демодуляции бит, приходящих из линии. Такие устройства модулятор/демодулятор называют *модемами*. Обычно линией связи служит телефонная линия, которая может быть прямой (или выделенной) или коммутируемой.

Эквивалентная схема одной из линий между интерфейсами или терминалом и модемом показана на рис. 9.8. Электрические параметры RS-232-C определены в терминах

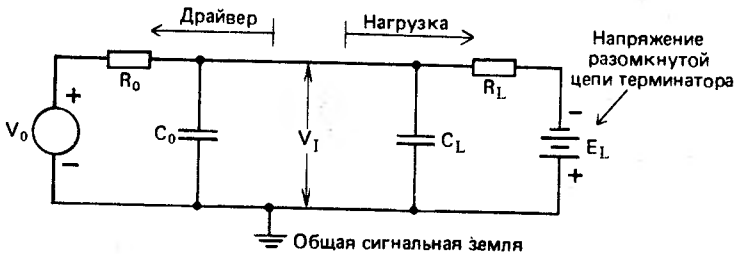


Рис. 9.8. Эквивалентная схема линии между интерфейсом или терминалом и модемом

Рис. 9.6. Зависимость максимального расстояния от скорости передачи для уровней напряжения интерфейса EIA RS-232-C

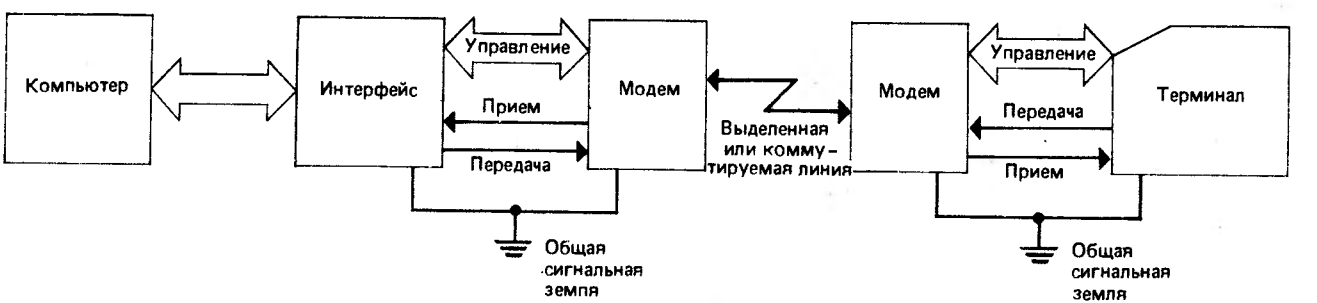


Рис. 9.7. Последовательная связь на большие расстояния

V_0 < 25 В

МАКСИМАЛЬНЫЙ ТОК КОРОТКОГО ЗАМЫКАНИЯ В ЛЮБОМ ПРОВОДНИКЕ КАБЕЛЯ 0,5 А

СИГНАЛ MARK ПРИ НАГРУЗКЕ < −3 В

СИГНАЛ SPACE ПРИ НАГРУЗКЕ > +3 В

СИГНАЛ MARK НА ВЫХОДЕ ДРАЙВЕРА < −5 В И > −15 В

СИГНАЛ SPACE НА ВЫХОДЕ ДРАЙВЕРА > +5 В И < +15 В

R_L < 7000 ОМ ПРИ ИЗМЕРЕНИИ С НАПРЯЖЕНИЕМ ОТ 3 ДО 25 В, НО >3000 ОМ

C_L, ВКЛЮЧАЯ ЕМКОСТЬ ЛИНИИ, < 2500 ПКФ

ПРИ E_L = 0, 3 В < V_I < 15 В

R_0 > 300 ОМ ПРИ ВЫКЛЮЧЕННОМ ПИТАНИИ

C_0 ТАКОВА, ЧТО СКОРОСТЬ ИЗМЕНЕНИЯ ВЫХОДНОГО НАПРЯЖЕНИЯ ДРАЙВЕРА < 30 В/МКС, НО ПЕРЕХОД МЕЖДУ −3 В И +3 В НЕ ДОЛЖЕН ПРЕВЫШАТЬ МЕНЬШЕГО ИЗ 1 МС ИЛИ 4% ИНТЕРВАЛА БИТА

Рис. 9.9. Основные электрические параметры интерфейса RS-232-C

этой схемы и представлены на рис. 9.9. Если интерфейс или терминал передают информацию в модем, они считаются драйвером, а модем – нагрузкой; при приеме информации их роли меняются. В любом случае различные компоненты эквивалентной схемы должны удовлетворять спецификациям, сформулированным на рис. 9.9. Отметим, что сигналы данных инвертированы в том смысле, что 1 представляется меньшим напряжением, чем 0. Что представляет собой драйвер для вывода данных: часть основного интерфейсного устройства или отдельную схему, зависит от интерфейса и длины линии. На рис. 9.10 показана схема преобразования сигналов ТТЛ-уровней и стандарта RS-232-C.

Частью стандарта RS-232-C являются определения и символические обозначения линий управления, содержащих интерфейс или терминал с модемом (см. рис. 9.11). Соединения обычно реализуются 25-контактными разъемами, номера контактов которых приведены во втором столбце. Номера в скобках, находящиеся в первом столбце, относятся к стандарту CCITT.

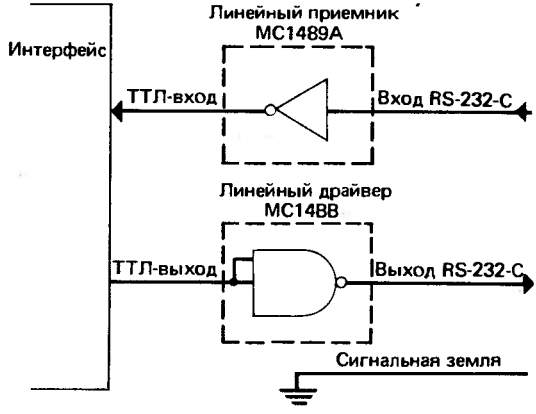


Рис. 9.10. Схема согласования передающей и приемной линий интерфейса RS-232-C

ОБОЗНАЧЕНИЕ EIA (CCITT)	КОНТАКТ	НАЗВАНИЕ	ОПИСАНИЕ
AA (101)	1	ЗАЩИТНАЯ ЗЕМЛЯ	ИСПОЛЬЗУЕТСЯ КАК ПРОВОДНИК ЗАЗЕМЛЕНИЯ ОБОРУДОВАНИЯ
AB (102)	7	СИГНАЛЬНАЯ ЗЕМЛЯ	ИСПОЛЬЗУЕТСЯ КАК ОБЩАЯ ЗЕМЛЯ ДЛЯ ВСЕХ СИГНАЛОВ
BA (103)	2	ПЕРЕДАВАЕМЫЕ ДАННЫЕ	ДЛЯ ВЫВОДА ДАННЫХ В МОДЕМ
BB (104)	3	ПРИНИМАЕМЫЕ ДАННЫЕ	ДЛЯ ВВОДА ДАННЫХ ИЗ МОДЕМА
CA (105)	4	ЗАПРОС ПЕРЕДАЧИ	В МОДЕМ, ДЛЯ ВКЛЮЧЕНИЯ И ВЫКЛЮЧЕНИЯ ПЕРЕДАТЧИКА МОДЕМА ПРИ РАБОТЕ В ПОЛУДУПЛЕКСНОМ РЕЖИМЕ
CB (106)	5	СБРОС ПЕРЕДАЧИ	ИЗ МОДЕМА, УКАЗЫВАЕТ, ЧТО МОДЕМ ГОТОВ ПЕРЕДАВАТЬ
CC (107)	6	ГОТОВНОСТЬ МОДЕМА	ИЗ МОДЕМА, УКАЗЫВАЕТ, ЧТО МОДЕМ ВКЛЮЧЕН И НЕ НАХОДИТСЯ В АВТОНОМНОМ РЕЖИМЕ
CF (109)	8	ДЕТЕКТОР СИГНАЛА ПРИЕМНОЙ ЛИНИИ	ИЗ МОДЕМА, УКАЗЫВАЕТ, ЧТО МОДЕМ ПРИНИМАЕТ СИГНАЛ ОТ МОДЕМА НА ДРУГОМ КОНЦЕ ЛИНИИ СВЯЗИ
CD (108/2)	20	ГОТОВНОСТЬ ТЕРМИНАЛА	В МОДЕМ, ГОТОВИТ ЕГО К ПОДКЛЮЧЕНИЮ К ЛИНИИ СВЯЗИ И НАЧАЛУ ПЕРЕДАЧИ
CE (125)	22	ИНДИКАТОР ЗВОНКА	ИЗ МОДЕМА, УКАЗЫВАЕТ, ЧТО В ЛИНИИ ОБНАРУЖЕН СИГНАЛ ЗВОНКА
CG (110)	21	ДЕТЕКТОР КАЧЕСТВА СИГНАЛА	ИЗ МОДЕМА, АКТИВЕН ПРИ МАЛОЙ ВЕРОЯТНОСТИ ОШИБКИ В ПРИНЯТЫХ ДАННЫХ
CH (111)	23	СЕЛЕКТОР СКОРОСТИ СИГНАЛОВ ДАННЫХ (ИСТОЧНИК DTE)	В МОДЕМ, УКАЗЫВАЕТ МОДЕМУ ОДНУ ИЗ ДВУХ СИНХРОННЫХ СКОРОСТЕЙ ДАННЫХ ИЛИ ОДИН ИЗ ДИАПАЗОНОВ СКОРОСТЕЙ
CI (112)	23	СЕЛЕКТОР СКОРОСТИ СИГНАЛОВ ДАННЫХ (ИСТОЧНИК DCE)	ИЗ МОДЕМА, УКАЗЫВАЕТ ИНТЕРФЕЙСУ ИЛИ ТЕРМИНАЛУ ОДНУ ИЗ ДВУХ СИНХРОННЫХ СКОРОСТЕЙ ДАННЫХ ИЛИ ОДИН ИЗ ДИАПАЗОНОВ СКОРОСТЕЙ
DA (113)	24	СИНХРОНИЗАЦИЯ ПЕРЕДАТЧИКА (ИСТОЧНИК DTE)	В МОДЕМ, СИГНАЛ СИНХРОНИЗАЦИИ ПЕРЕДАТЧИКА МОДЕМА
DB (114)	15	СИНХРОНИЗАЦИЯ ПЕРЕДАТЧИКА (ИСТОЧНИК DCE)	ИЗ МОДЕМА, ОБЕСПЕЧИВАЕТ ИНТЕРФЕЙС ИЛИ ТЕРМИНАЛ СИГНАЛОМ СИНХРОНИЗАЦИИ ПЕРЕДАТЧИКА
DD (115)	17	СИНХРОНИЗАЦИЯ ПРИЕМНИКА	ИЗ МОДЕМА, ОБЕСПЕЧИВАЕТ ИНТЕРФЕЙС ИЛИ ТЕРМИНАЛ СИГНАЛОМ СИНХРОНИЗАЦИИ ПРИЕМНИКА
SBA (118)	14	ВТОРИЧНЫЕ ПЕРЕДАВАЕМЫЕ ДАННЫЕ	В МОДЕМ, ДЛЯ ВЫВОДА ДАННЫХ С НИЗКОЙ СКОРОСТЬЮ
SBB (119)	16	ВТОРИЧНЫЕ ПРИНИМАЕМЫЕ ДАННЫЕ	В МОДЕМ, ДЛЯ ВВОДА ДАННЫХ С НИЗКОЙ СКОРОСТЬЮ
SCA (120)	19	ВТОРИЧНЫЙ ЗАПРОС ПЕРЕДАЧИ	В МОДЕМ, ДЛЯ ВКЛЮЧЕНИЯ ВТОРИЧНОГО ПЕРЕДАТЧИКА МОДЕМА
SCB (121)	13	ВТОРИЧНЫЙ СБРОС ПЕРЕДАЧИ	ИЗ МОДЕМА, ПОКАЗЫВАЕТ ГОТОВНОСТЬ ВТОРИЧНОГО ПЕРЕДАТЧИКА
SCF (122)	12	ДЕТЕКТОР ПРИНЯТОГО СИГНАЛА НА ВТОРИЧНОЙ ЛИНИИ	ИЗ МОДЕМА, ПОКАЗЫВАЕТ, ЧТО НА ВТОРИЧНОЙ ЛИНИИ ОБНАРУЖЕН СИГНАЛ

Рис. 9.11. Определения линий управления стандарта RS-232-C

Для связи по прямой телефонной линии требуются только восемь первых из показанных на рис. 9.11 линий. При передаче в модем посылается сигнал "запрос передачи" (CA), который подтверждается сигналом "сброса передачи" (CB). Затем по линии "передаваемые даниые" (BA) начинается собственно передача. При приеме модем возбуждает сигнал на линии "детектор сигнала приемной линии" (CF), показывая прием сигнала от модема с другого конца линии связи. Принятые данные передаются в интерфейс или терминал по линии "принятые данные" (BB). Сигнал "готовность модема" означает,

что модем включен и находится в рабочем режиме. Эта линия должна быть активной как при передаче, так и при приеме.

Если линия связи является частью коммутируемой телефонной системы, потребуются еще минимум две линии. Сигнал в модем по линии "готовность терминала" (CD) управляет коммутацией модема, что позволяет образовать линию связи. Модем посылает в интерфейс или терминал сигнал "индикатор звонка" (CE), показывая, что он принимает сигнал звонка. Для ответа на этот сигнал интерфейс или терминал должны реагировать активным сигналом "готовность терминала". Он заставляет модем отвечать "как звенящий телефон".

В синхронной связи совместно с данными необходимо передавать синхронизирующую информацию. Для этого предусмотрены две линии "синхросигнал передатчика". Линия "источник DTE" (DA) служит для передачи синхронизирующей информации из интерфейса или терминала в модем, а линия "источник DCE" – для передачи в другом направлении. Так как модемы синхронной связи часто допускают две скорости передачи данных или два диапазона скоростей, линии "селектор скорости данных" (CH и CI) несут информацию о скорости передачи. По линии CH эта информация передается из интерфейса или терминала в модем, а по линии CI – в другом направлении, показывая текущую скорость передачи модема.

В быстродействующих синхронной или асинхронной линиях связи оказалось эффективнее передавать информацию пакетами, содержащими наряду с данными идентификацию и информацию для обнаружения ошибок. Пакеты обычно применяют при взаимодействии компьютеров друг с другом. Вместе с передачей пакетов необходимы определенные квитирование и контроль, что лучше всего осуществить по вторичной (обратной) низкоскоростной линии связи. Последние пять линий, определенные на рис. 9.11, предназначены именно для вторичных передач. Мы не будем касаться этих линий, так как подробное рассмотрение пакетов и протоколов связи выходит за рамки книги.

Следует отметить наличие еще одного стандарта на передачу данных между интерфейсом или терминалом и модемом. Он называется *стандартом 20-мА петли* и реализуется

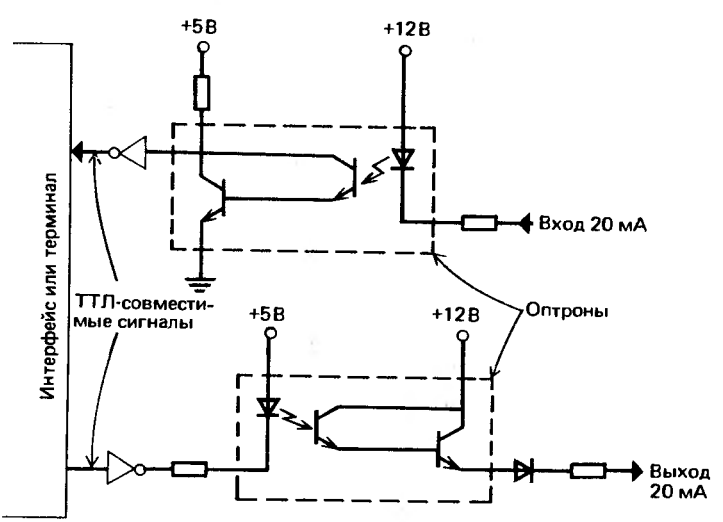
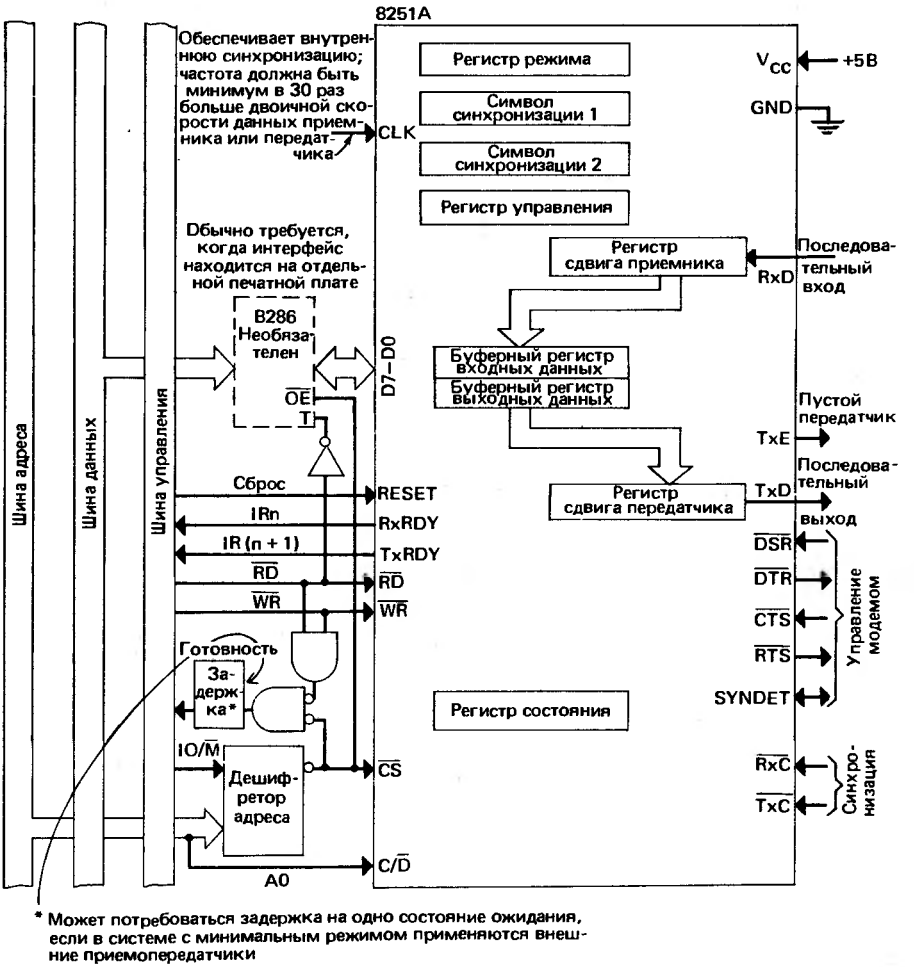


Рис. 9.12. Схема передачи и приема сигналов 20-мА петли

петлевой линией, в которой имеется ток 20 мА в состоянии маркера (1), а в состоянии пробела (0) ток отсутствует. В данной конфигурации одно из взаимодействующих устройств должно быть активным и служить источником тока, а другое устройство должно быть пассивным. Исторической основой этого стандарта являются механические телетайпы, в которых ток управлял соленоидами, но он сохранился даже в большинстве современных терминалов с дисплеем наряду со стандартом RS-232-C. Схемы для передачи и приема сигналов 20-мА токовой петли приведены на рис. 9.12.

Мы не касались многих деталей стандартов. Имеется множество книг по теории и практике передачи данных, некоторые из которых упомянуты в списке литуратуры к данной главе. Мы стремились показать, как соединяются интерфейс компьютера и система связи, и дать информацию, необходимую при проектировании интерфейса.



* Может потребоваться задержка на одно состояние ожидания, если в системе с минимальным режимом применяются внешние приемопередатчики

Рис. 9.13. Последовательный связной интерфейс 8251A

9.1.4. ПРОГРАММИРУЕМЫЙ СВЯЗНОЙ ИНТЕРФЕЙС

Как пример устройства последовательного интерфейса, рассмотрим микросхему 8251A программируемого связного интерфейса, схема которой представлена на рис. 9.13. Эту микросхему можно запрограммировать для асинхронной или синхронной передачи. Буферные регистры входных и выходных данных разделяют порт с одним и тем же адресом. Последовательный двоичный поток со входа RxD вводится в регистр сдвига приемника, а затем биты данных передаются в буферный регистр входных данных, а оттуда в процессор. Выводимые биты данных процессор помещает в буферный регистр выходных данных, затем они передаются в регистр сдвига передатчика и вместе с битами синхронизации – на выход TxD. Содержимое регистра режима, который инициализируется выполняемой программой, определяет синхронный или асинхронный режим и формат передаваемых и принимаемых символов. Регистр управления, который также устанавливается программой, координирует работу интерфейса, а регистр состояния обеспечивает программе определенную информацию. Для синхронной связи необходимы регистры для хранения символов синхронизации.

Несмотря на то что процессор может адресовать семь регистров микросхемы, она ассоциируется с адресами всего двух портов. Вход $C/\overline{D}$ подключается к линии адреса A0 и она дифференцирует адреса двух портов. Микросхема интерпретирует сигналы $C/\overline{D}$, $\overline{RD}$ и $\overline{WR}$ в соответствии с табл. 9.1.

Таблица 9.1

Интерпретация сигналов управления

$C/\overline{D}$(= A0)	$\overline{RD}$	$\overline{WR}$	Действие
0	0	1	Ввод данных из буфера входных данных
0	1	0	Вывод данных в буфер выходных данных
1	0	1	Передача регистра состояния на шину данных
1	1	0	Загрузка с шины данных в регистры режима, управления или символа синхронизации

Остальные комбинации сигналов переводят тристабильные выходы D7-D0 в высокоимпедансное состояние.

Выбор регистров режима, управления или символа синхронизации зависит от последовательности обращения, схема которой представлена на рис. 9.14. После аппаратного сброса или приказа с установленным в 1 битом сброса следующий вывод с A0 = 1 (т. е. с $C/\overline{D}$ = 1, $\overline{RD}$ = 1 и $\overline{WR}$ = 0) предназначен для регистра режима. Форматы регистра режима для асинхронного и синхронного случаев показаны на рис. 9.15. Если два младших бита режима содержат нули, интерфейс переводится в синхронный режим и старший бит определяет число символов синхронизации. При этом следующие один или два байта, выводимые с A0 = 1, становятся символами синхронизации. Если два младших бита режима не равны 0, интерфейс работает в асинхронном режиме. В любом случае все последующие байты до другого сброса направляются в регистр управления (если A0 = 1) или в буферный регистр выходных данных (если A0 = 0).

В синхронном режиме скорости в бодах передатчика и приемника, равные частотам сдвига регистров сдвига, совпадают с частотами сигналов на входах $\overline{TxC}$ и $\overline{RxC}$ соответственно. При работе в асинхронном режиме три оставшиеся комбинации младших бит в регистре режимов определяют множитель скорости. Взаимосвязь между частотами

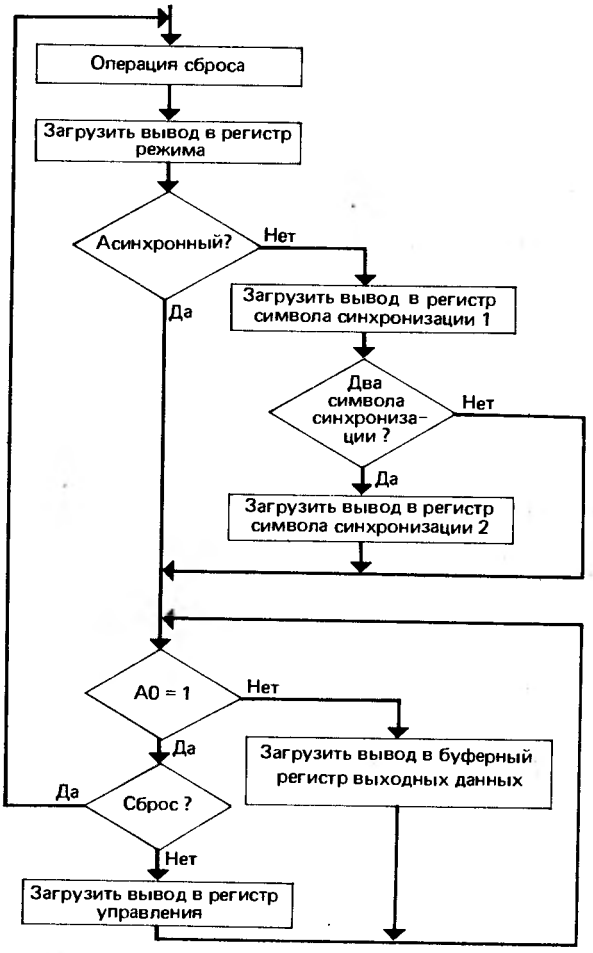


Рис. 9.14. Схема размещения выводимых байт

$\overline{TxC}$ и $\overline{RxC}$ входов синхронизации и скоростями в бодах передатчика и приемника имеет следующий вид:

частота синхронизации = множитель × скорость в бодах.

Когда младшие биты регистра режима содержат 10 и скорости в бодах передатчика и приемника равны 300 и 1200, частота на входе $\overline{TxC}$ должна быть 4800 Гц, а на входе $\overline{RxC}$ – 19,2 кГц.

В обоих режимах биты 2 и 3 показывают число информационных бит в каждом символе, бит 4 определяет наличие или отсутствие бита паритета, а бит 5 – тип паритета (нечетный или четный). В асинхронном режиме два старших бита показывают число стоповых бит. Для синхронного режима бит 6 показывает, каким является сигнал SYNDET: входным или выходным, а бит 7 задает число символов синхронизации.

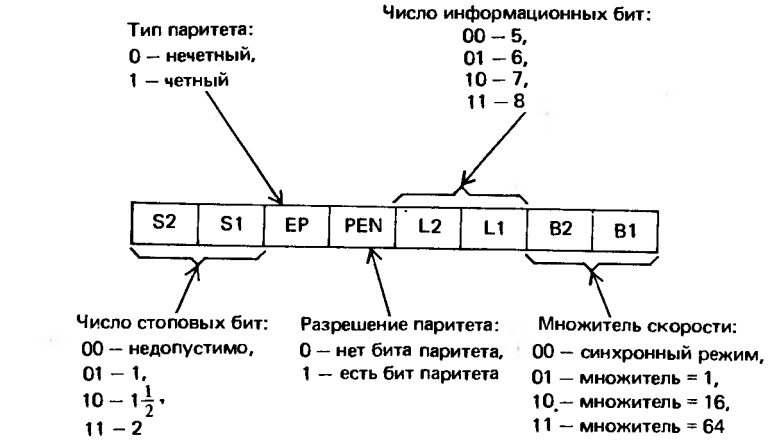


Рис. 9.15. Форматы регистра режима: асинхронный (*а*) и синхронный (*б*)

Если контакт SYNDET служит выходом, его сигнал становится активным, когда обнаружено соответствие входного двоичного потока и символа (ов) синхронизации. Когда же поиск символов синхронизации осуществляется внешним устройством, на вход SYNDET подается сигнал о фиксации соответствия. Этот же контакт, но только в качестве выхода применяется и при асинхронной работе. Сигнал на этом выходе иазывается сигналом обиаружения разрыва и он высоким уровнем отмечает прием символа, состоящего из одних нулей.

Формат регистра управления приведен иа рис. 9.16. Бит 0 этого регистра должен содержать 1 до вывода данных, а бит 2 должен быть в состоянии 1 до приема данных. Программиый ответ модему осуществляется установкой в 1 бита 1, так как при этом на выходе $\overline{\text{DTR}}$ формируется 0, а инверсия $\overline{\text{DTR}}$ обычно подключается к линии CD модема. Состояние 1 бита 3 формирует TxD = 0, вызывая передачу символов разрыва. Установ-

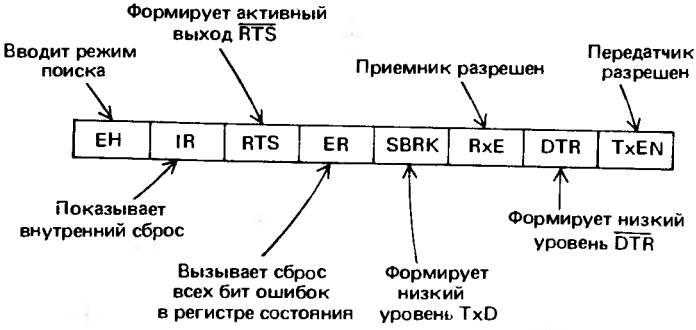


Рис. 9.16. Формат регистра управления

Примечание. Во всех случаях действие предпринимается, когда бит установлен в 1

ка в 1 бита 4 вызывает сброс всех бит ошибок в регистре состояния. Бит 5 применяется для выдачи сигнала "запрос передачи" ($\overline{RTS}$) в модем. Если инверсия сигнала на контакте $\overline{RTS}$ подключена к линии CA модема, то загрузка 1 в бит 5 вызывает высокий уровень на линии CA. Установка бита 6 в состояние 1 вызывает реинициализацию 8251A и переход к последовательности сброса (т. е. осуществляется возврат в начало схемы на рис. 9.14 и следующий вывод производится в регистр режима). Бит 7 применяется только в синхронном режиме; установка его в 1 заставляет 8251A начать поиск символа(ов) синхронизации.

На рис. 9.17 показаны типичные схемы подключения к модемам для асинхронной и синхронной передач. При синхронной передаче предполагается, что синхронизацией управляет модем и его связное оборудование. Кроме того, если это же оборудование применяется для обнаружения символа(ов) синхронизации, оно информирует 8251A об их обнаружении по линии SYNDET. С другой стороны, если символ(ы) синхронизации отыскивает сам 8251A, он может использовать линию SYNDET, чтобы сообщить модему об успешном поиске. Для удовлетворения требований стандарта RS-232-C необходимы драйверы и приемники, преобразующие ТТЛ-совместимые сигналы TxD и RxD в требуемые уровни напряжения (см. рис. 9.10).

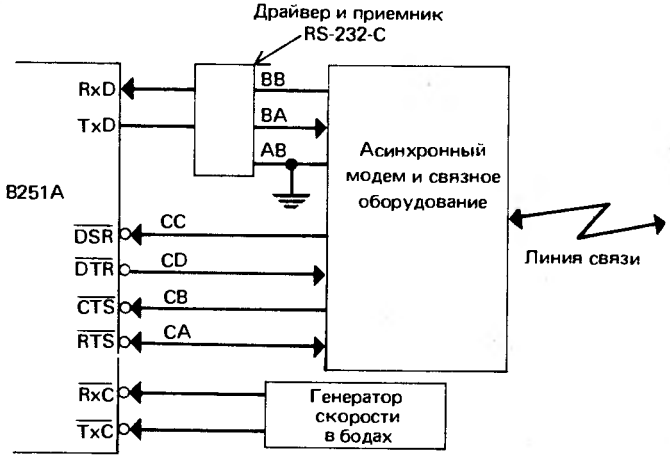
Ниже приведен программный фрагмент, который инициализирует регистр режима и содержит приказ разрешения передатчика и запуска асинхронной передачи со следующим форматом: 7 бит символа, бит четного паритета и 2 стоповых бита:

```
MOV  AL,11111010B
OUT  51H,AL
MOV  AL,00110011B
OUT  51H,AL
```

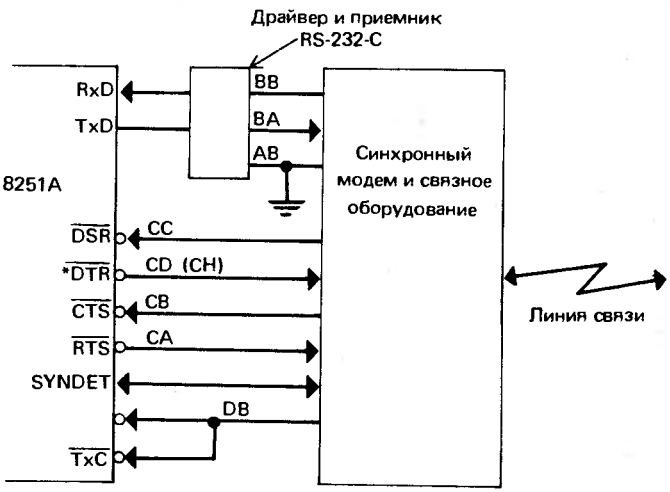
Здесь предполагается, что регистры режима и управления имеют адрес 51H, а частоты синхронизации в 16 раз больше соответствующих скоростей в бодах.

Следующий фрагмент переводит 8251A в синхронный режим и заставляет его начать поиск двух смежных символов синхронизации:

```
MOV  AL,00111000B
OUT  51H,AL
MOV  AL,16H
OUT  51H,AL
MOV  AL,10010100B
OUT  51H,AL
```



а)



Можно также использовать для выбора скорости — линия модема CH

б)

Рис. 9.17. Подключения микросхемы 8251A к асинхронному (*а*) и синхронному (*б*) модему

Примечание. Подключения линий управления зависят от спецификаций модема

Как и в первом примере, символы содержат 7 информационных бит и бит четного паритета, но, разумеется, стоповые биты отсутствуют.

Формат регистра состояиия приведен иа рис. 9.18. Биты 1, 2 и 6 отражают состояния сигналов RxRDY, TxE и SYNDET. Бит TxRDY показывает, что буфер выходных данных

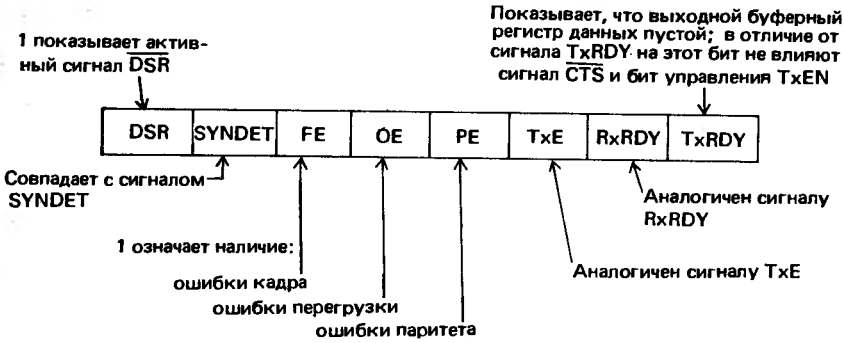


Рис. 9.18. Формат регистра состояния

пустой. В отличие от контакта TxRDY на этот бит не влияет входной сигнал $\overline{\text{CTS}}$ или бит управления TxEN. Бит RxRDY показывает, что символ принят и готов для ввода в процессор. Биты TxRDY и RxRDY можно использовать для программного ввода-вывода, а сигналы с соответствующих контактов можно подключить к линиям запросов прерываний для организации ввода-вывода по прерываниям. Бит TxRDY автоматически сбрасывается, когда имеется символ для передачи, а бит RxRDY – когда установивший его символ введен процессором. Бит 2 показывает, что регистр сдвига передатчика ожидает загрузки символа из буферного регистра выходных данных. В синхронной передаче, пока этот бит установлен, передатчик будет брать данные из регистров символов синхроизации до загрузки данных в буферный регистр выходных данных. Биты 3, 4 и 5 показывают ошибки паритета, перегрузки и кадра. Когда обнаруживается ошибка, соответствующий ей бит устанавливается в 1. Если инверсию выхода $\overline{\text{DSR}}$ подключить к линии CC, то бит 7 отражает состояние модема – он устанавливается в 1, когда модем включен и находится в рабочем режиме.

На рис. 9.19 приведена типичная программа, в которой используется программный ввод-вывод для ввода 80 символов из 8251A (адрес буферного регистра данных равен 0050) и размещения их в буфере памяти, начинающегося с LINE. Внутренний цикл непрерывно проверяет бит RxRDY до тех пор, пока он не устанавливается символом, принятым в буферный регистр входных данных. Затем новый принятый символ пересыла-

```
        MOV     AL,00110101B        ;РАЗРЕШИТЬ ПЕРЕДАЧУ И ПРИЕМ
        OUT     51H,AL              ; И СБРОСИТЬ БИТЫ ОШИБОК
        MOV     DI,0                ;ИНИЦИАЛИЗИРОВАТЬ ИНДЕКС
        MOV     CX,80               ;ИНИЦИАЛИЗИРОВАТЬ СЧЕТЧИК В CX
BEGIN:  IN      AL,51H              ;ОЖИДАТЬ ВВОД
        TEST    AL,02H
        JZ      BEGIN
        IN      AL,50H              ;ВВЕСТИ СИМВОЛ И ПОМЕСТИТЬ
        MOV     LINE[DI],AL         ;В БУФЕР СТРОКИ
        INC     DI
        IN      AL,51H              ;ПРОВЕРИТЬ БИТЫ ОШИБОК И,
        TEST    AL,00111000B        ;ЕСЛИ ОШИБОК НЕ ОБНАРУЖЕНО,
        JNZ     ERROR               ;ПРОДОЛЖАТЬ ВВОД
        LOOP    BEGIN
        JMP     SHORT EXIT
ERROR:  CALL    NEAR PTR ERR_ROUT   ;ИНАЧЕ ВЫЗВАТЬ ERR_ROUT
EXIT:    .
         .
         .
```

Рис. 9.19. Ввод строки символов через интерфейс 8251A

ется в память и контролируются биты ошибок. Если текущий символ появился до ввода предыдущего символа или во время передачи возникли ошибки паритета или кадра, ввод прекращается и вызывается процедура обработки ошибки, которая обычно проверяет отдельные биты ошибок, печатает соответствующее сообщение и сбрасывает биты ошибок.

Ввод символа автоматически сбрасывает бит RxRDY, поэтому если другой символ не принят до начала внутреннего цикла, внутренний цикл должен повторяться до тех пор, пока следующий входной символ не установит RxRDY = 1. Если входные символы имеют меньше 8 бит, неиспользуемые старшие биты в буферном регистре данных всегда возвращаются в состояние 0. Бит паритета не передается в процессор, поэтому контроль ошибок паритета можно осуществить только по состоянию бита ошибки паритета в регистре состояния. Когда при выводе длина символов меньше 8 бит, ненужные старшие биты в буферном регистре выходных данных игнорируются.

9.2. ПАРАЛЛЕЛЬНАЯ СВЯЗЬ

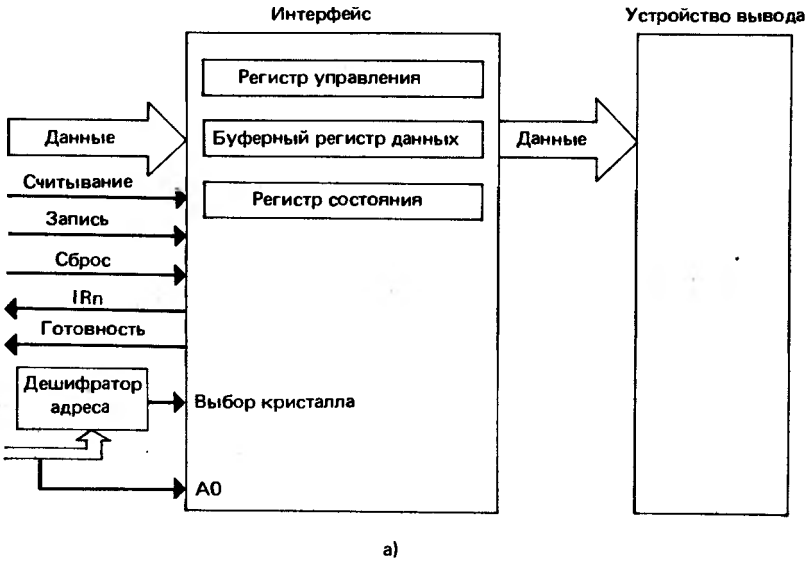
Параллельная связь осуществляется одновременной передачей нескольких бит по отдельным линиям. Ее преимущество по сравнению с последовательной связью заключается в том, что для линий с заданной максимальной двоичной скоростью обеспечивается более высокая скорость передачи информации. Недостатком же является, конечно, стоимость дополнительных линий, и так как расходы увеличиваются с расстоянием, параллельная связь применяется на большие расстояния, если только требуется высокая скорость передачи данных.

В отличие от последовательной связи четкие стандарты на параллельную связь отсутствуют. Параллельные передачи выполняются либо простым помещением данных в буферный регистр выходных данных интерфейса или восприятием данных из буферного регистра входных данных интерфейса, либо ими управляют сигналы квитирования и (или) синхронизации. Обычно одновременно передается один символ (или другая единица информации) и необходимости определения конца символа или начала передачи не возникает. Четко определенный формат синхронной или асинхронной передачи отсутствует, но, если действия устройства и интерфейса координирует сигнал синхронизации, передача считается синхронной. Если же применяются только квитирующие сигналы, передача считается асинхронной.

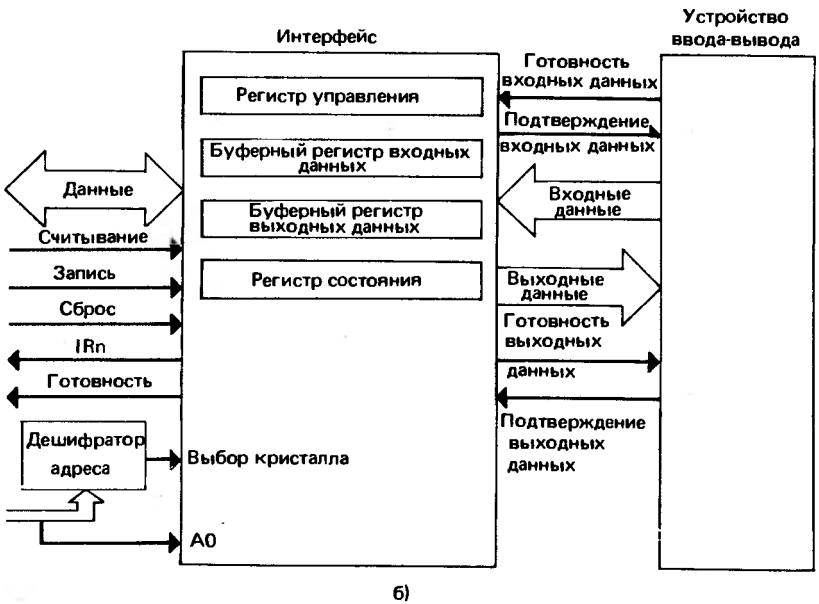
Интерфейс можно спроектировать только на вывод, только на ввод, на ввод и вывод по отдельным наборам линий, или выполнять ввод-вывод по одному набору двунаправленных линий. Если интерфейс подключен к строчному принтеру, он должен только выводить данные, а если он обслуживает карточный считыватель, интерфейс должен только вводить данные. В интерфейсе, обслуживающем перфоленточные считыватель и перфоратор, потребуются один набор входных линий и один набор выходных линий, но в интерфейсе для устройства, которое не производит одновременные ввод и вывод данных, можно применить один набор двунаправленных линий.

На рис. 9.20, *а* показан типичный выходной параллельный интерфейс, в котором отсутствуют линии управления. В этом случае содержимое буферного регистра данных постоянно выдается на линии данных, связывающие интерфейс и устройство вывода. Если устройство вывода состоит из регистра-защелки и управляемого им набора реле, компьютер может управлять этими реле, просто помещая данные в буферный регистр данных.

На рис. 9.20, *б* изображен типичный параллельный интерфейс с линиями квитирования и отдельными подключениями к периферийному устройству для ввода и вывода. В этом случае при вводе данные сначала помещаются на шину данных, а затем на линии готовности входных данных устанавливается 1. Интерфейс должен отреагировать загрузкой данных в буфер входных данных и формированием 1 на линии подтверждения вход-



а)



б)

Рис. 9.20. Типичные интерфейсы параллельной связи:
а – только ввод; *б* – ввод-вывод

ных данных. При получении подтверждения устройство снимает сигналы данных и готовности. Когда интерфейс принял данные, он устанавливает бит состояния "готовность" и, возможно, посылает сигнал запроса прерывания. После того как процессор вводит

данные, интерфейс сбрасывает бит состояния "готовность" и переводит линии данных в высокоимпедансное состояние. Если был сделан запрос прерывания, он обрабатывается обычным образом.

При выводе интерфейс устанавливает бит состояния "готовность" и, возможно, формирует запрос прерывания, когда буфер выходных данных освободился. После того как процессор выводит данные, интерфейс сбрасывает бит состояния "готовность", помещает данные на шину выходных данных и сигнализирует устройству по линии готовности выходных данных. Когда устройство готово воспринять данные, оно загружает их в регистр-защелку, а затем возвращает подтверждение. После этого интерфейс снимает сигнал готовности выходных данных и вновь устанавливает бит состояния "готовность".

9.2.1. ПРОГРАММИРУЕМЫЙ ПЕРИФЕРИЙНЫЙ ИНТЕРФЕЙС

Примером параллельного интерфейса служит микросхема 8255A программируемого периферийного интерфейса. Как показано на рис. 9.21, она имеет регистр управления и три отдельно адресуемых порта A, B и C. Обращение к 8255A определяет сигнал $\overline{CS}$, а направление обращения – сигналы $\overline{RD}$ и $\overline{WR}$. Адресуемый регистр определяют сигналы на входах A1 и A0. Следовательно, младший адрес порта, назначаемый 8255A, должен быть кратным 4. Полная адресация 8255A содержится в табл. 9.2.

Таблица 9.2

Адресация микросхемы 8255A

A1	A0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	Описание передачи
0	0	0	1	0	Порт A на шину данных
0	1	0	1	0	Порт B на шину данных
1	0	0	1	0	Порт C на шину данных
0	0	1	0	0	Шина данных в порт A
0	1	1	0	0	Шина данных в порт B
1	0	1	0	0	Шина данных в порт C
1	1	1	0	0	Шина данных в регистр управления, если D7 = = 1; если D7 = 0, вход с шины данных считается командой установки/сброса
X	X	X	X	1	D7-D0 в высокоимпедансном состоянии
1	1	0	1	0	Запрещенная комбинация
X	X	1	1	0	D7-D0 в высокоимпедансном состоянии

Так как биты порта C иногда используются как биты управления, 8255A спроектирован так, что в них можно выводить по отдельности, пользуясь командой установки/сброса. Когда 8255A принимает байт, направляемый в его регистр управления, он анализирует бит 7 данных. Если этот бит содержит 1, данные передаются в регистр управления; если же бит 7 = 0, данные считаются командой установки/сброса и применяются для установки или сброса определяемого командой бита порта C. Биты 3-1 дают номер изменяемого бита, а бит 0 показывает сброс или установку. Остальные биты не используются.

Биты трех портов выведены на контакты, которые подключаются к устройству ввода-вывода. Они разделены на группы A и B, причем группу A образуют биты порта A и 4 старших бита порта C, а группу B – порт B и 4 младших бита порта C. Группа A может работать в трех режимах (0, 1, 2), а группа B – в двух режимах (0 и 1). Режимы определяются содержимым регистра управления, формат которого приведен на рис. 9.22.

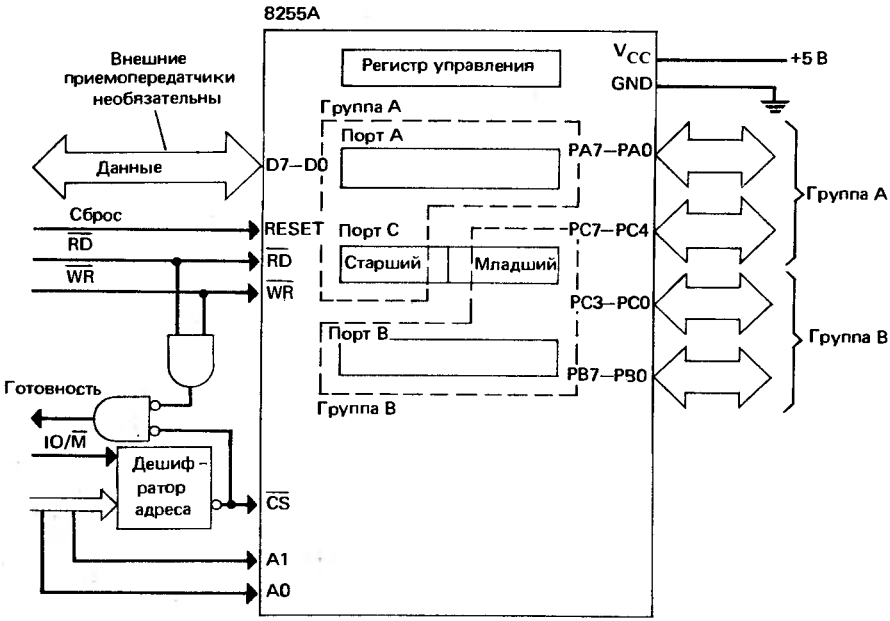


Рис. 9.21. Схема интерфейса 8255A

Режим 0. В этом режиме группа разделяется на два набора. В группе А этими наборами являются порт А и старшие 4 бита порта С, а в группе В – порт В и младшие биты порта С. Каждый набор можно использовать для ввода или вывода, но не для двунаправленных передач. Биты D4, D3, D1 и D0 в регистре управления определяют, какие наборы предназначены для ввода, а какие – для вывода. Эти биты ассоциируются с наборами следующим образом:

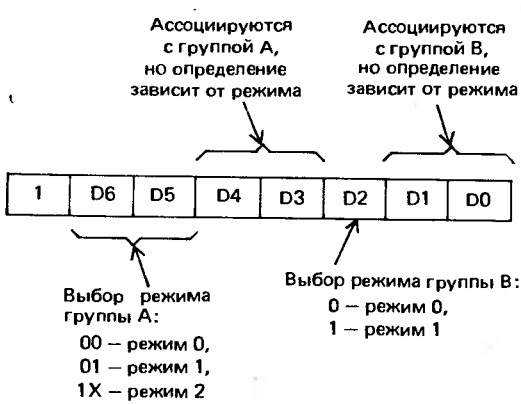


Рис. 9.22. Формат регистра управления микросхемы 8255A

D4 – порт A,
D3 – старшая половина порта C,
D1 – порт B,
D0 – младшая половина порта C.

Если бит содержит 0, соответствующий набор применяется для вывода, а в противном случае – для ввода.

Режим 1. Когда в этом режиме работает группа A, порт A используется для ввода или вывода в соответствии с битом D4 (D4 = 1 указывает ввод), а старшая половина порта C применяется для сигналов квитирования и управления.

При *вводе* старшим битам порта C назначены следующие названия и определения:

PC4 $\overline{STB}_A$ – сигнал 0 на этом контакте заставляет PA7-PA0 "стробироваться" в порт A.

PC5 IBF_A – показывает, что входной буфер заполнен. Устанавливается в 1, когда порт A содержит данные, которые еще не введены в процессор. Когда действует сигнал 0, устройство может вводить в интерфейс новый байт.

PC6,PC7 – применяются для вывода сигналов управления в устройство или ввода состояния из устройства. Если бит D3 в регистре управления содержит 0, эти линии выходные; в противном случае – входные.

При *выводе*:

PC4,PC5 – выполняют те же функции, что и PC6, PC7 при вводе.

PC7 $\overline{OBF}_A$ – показывает, что выходной буфер заполнен. Формирует сигнал 0 в устройство, когда порт A выводит в устройство новые данные.

PC6 $\overline{ACK}_A$ – устройство подает 0 на этот вход, когда оно восприняло данные из порта A.

В режиме 1 с группой A ассоциируется PC3, обозначаемый $INTR_A$. Этот сигнал применяется как линия запроса прерывания и подключается к одной из линий IR системной шины. При вводе в порт A на этом выходе формируется сигнал 1, когда новые данные помещаются в порт A (т. е. им управляет PC4), и сигнал 0, когда процессор считывает данные. При выводе на выходе $INTR_A$ формируется сигнал 1, когда содержимое порта A воспринято устройством, и сигнал 0, когда процессор загружает новые данные. Если группа B находится в режиме 1, порт B является входным или выходным в зависимости от состояния бита D1 регистра управления (D1 = 1 определяет порт B входным). При вводе PC2 и PC1 обозначаются $\overline{STB}_B$ и IBF_B; они выполняют для группы B те же функции, что и сигналы $\overline{STB}_A$ и IBF_A для группы A. Аналогично при выводе PC1 и PC2 обозначаются $\overline{OBF}_B$ и $\overline{ACK}_B$. Выход PC0 превращается в $INTR_B$ и используется аналогично $INTR_A$. Разрешением прерываний для группы A управляют PC4 в режиме ввода и PC6 в режиме вывода. Например, в режиме ввода прерывание для группы A разрешается установкой PC4 и запрещается сбросом PC4 с помощью команды установки/сброса. Аналогично разрешением прерываний для группы B управляет установка/сброс PC2.

Режим 2. В данном режиме может работать только группа A, хотя в ней для генерирования запросов прерываний используется PC3. В режиме 2 порт A становится двунаправленным, а 4 старших бита порта C определяются следующим образом:

PC4 $\overline{STB}_A$ – сигнал 0 на этой линии заставляет данные с PA7-PA0 стробироваться в порт A.

PC5 IBF_A – формируется сигнал 1, когда в порт A загружаются новые данные с линий PA7-PA0, и сигнал 0, когда процессор считывает данные.

PC6 $\overline{ACK}_A$ – показывает, что устройство готово воспринимать данные с линий PA7-PA0.

PC7 $\overline{OBF}_A$ – формирует сигнал 0, когда процессор загружает в порт А новые данные, и сигнал 0, когда данные приняты устройством.

Когда группа А находится в режиме 2, группа В может работать в режиме 0 или 1. Если группа В находится в режиме 0, только PC2-PC0 можно использовать для ввода или вывода, так как PC3 служит запросом прерываний для группы А. Обычно, если группа А работает в режиме 2, PC2-PC0 подключается к контактам управления и состояния устройства, которое подсоединено к линиям порта А. Для этих же целей может применяться порт В.

Во всех трех режимах порт С отражает сигналы на линиях PC7-PC0 и его можно считать командой IN.

9.2.2. ПРИМЕР ИСПОЛЬЗОВАНИЯ

На рис. 9.23 показано возможное подключение микросхемы 8255А к аналого-цифровой и цифро-аналоговой подсистемам. Так во время аналого-цифрового преобразования аналоговое напряжение должно оставаться неизменным, необходима схема выборки и сохранения. Группа А работает на ввод в режиме 1. Преобразование инициируется сигналом на выходе PC7, что стимулирует преобразователь выдать сигнал занятости. Линия занятости подключена на вход управления выборкой и сохранением (S/H) и на вход одновибратора, запускаемого спадающим фронтом. Пока сигнал занятости имеет высокий уровень, схема выборки и сохранения поддерживает постоянный выход, а когда в конце преобразования сигнал занятости снимается, запускается одновибратор. Его выход инвертируется и подается на вход $\overline{STB}_A$ (PC4) микросхемы 8255А. Он заставляет цифровой отсчет стробироваться в порт А. В цифро-аналоговой части подсистемы порт В работает как выходной в режиме 0; он подключен непосредственно на двоичный вход цифро-аналогового преобразователя. Квитирование здесь не применяется.

Предположим, что порты А, В и С и регистр управления имеют адреса FFF8, FFF9, FFFA и FFFB. Тогда команды

```
MOV  AL,10110000B
OUT  DX,AL
```

заставляют порт А работать в режиме 1, порт В – в режиме 0, а PC7 быть выходом. Следующий фрагмент формирует импульс на входе запуска аналого-цифрового преобразователя:

```
MOV  DX,0FFFBH
MOV  AL,00001111B
OUT  DX,AL
MOV  AL,00001110B
OUT  DX,AL
```

Здесь первая команда загружает в регистр DX адрес, ассоциируемый с командой установки/сброса, который совпадает с адресом регистра управления. Следующие две команды формируют сигнал PC7 = 1, а последние две – сигнал PC7 = 0. Фрагмент программного ввода преобразованных данных имеет вид

```
        MOV  DX,0FFFAH
AGAIN:  IN   AL,DX
        TEST AL,00100000B
        JZ   AGAIN
        MOV  DX,0FFF8H
        IN   AL,DX
```

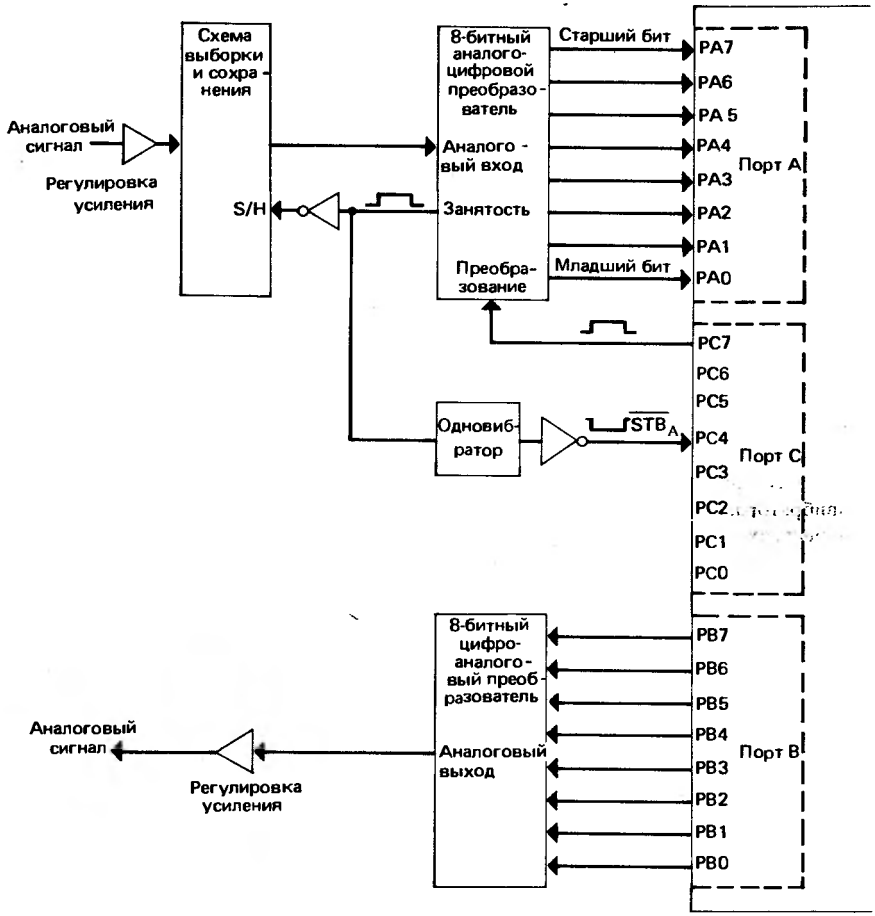


Рис. 9.23. Интерфейс аналого-цифровой и цифро-аналоговой подсистем с применением микросхемы 8255A

Для вывода байта из регистра AL в цифро-аналоговый преобразователь требуются только две команды:

```
MOV  DX,0FFF9H
OUT  DX,AL
```

Как только байт появляется в порту B, его биты сразу подаются на входы цифро-аналогового преобразователя, который в свою очередь сразу же преобразует их в аналоговый сигнал.

В данном примере предполагается, что синхронизация преобразований осуществляется программой и что коэффициенты усиления входного и выходного аналоговых усилителей регулируются. Чтобы получить от программы равномерное распределение входных и выходных отсчетов, необходимо учитывать времена выполнения команд. Между

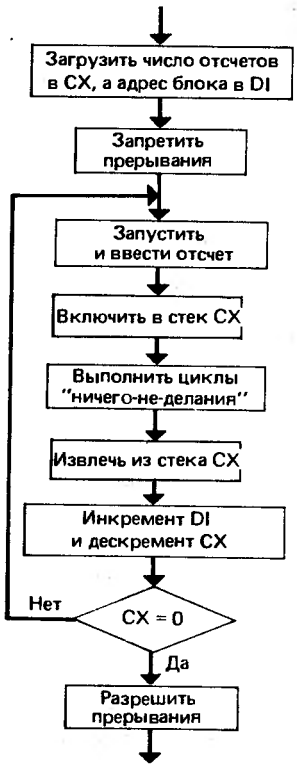


Рис. 9.24. Программная синхронизация ввода отсчетов

отсчетами выполняются одни и те же команды и общее время их выполнения точно известно. Прерывания необходимо запретить, так как они случайным образом вводят выполнение разного числа команд. Интервал между отсчетами можно скорректировать с помощью цикла "ничего-не-делания", например:

```
      MOV  CX,N
IDLE: NOP
      LOOP IDLE
```

между вводами или выводами. Схема ввода с программной синхронизацией отсчетов аналого-цифрового преобразования представлена иа рис. 9.24.

Часто в аналого-цифровой и цифро-аналоговой подсистеме применяются программируемые генераторы синхронизации и усилители, что позволяет более точно управлять расположением отсчетов и цифро-аналоговых выходов, а также дииамически изменять усиление. Кроме того, в такую подсистему часто вводят коитроллер ПДП для увеличения скорости ввода и вывода.

На рис. 9.23 показаны только 8-битные преобразователи, имеющие разрешающую способность 1 из 256. Если диапазон входного или выходного напряжеиий равен −10 . . . +10 В, разрешающая способность составляет

$$20/256 = 0{,}078 \text{ В}.$$

Для улучшения разрешающей способности применяются 10-, 12- или 14-битные преобразователи. В этом случае приходится пользоваться комбинацией портов A и C или B и C (см. упр. 14) или параллельно включать две микросхемы 8255A (см. § 9.7).

9.3. ПРОГРАММИРУЕМЫЕ ТАЙМЕРЫ И СЧЕТЧИКИ СОБЫТИЙ

Довольно часто требуется устройство формирования временных интервалов для процессора и внешних устройств, подсчета внешних событий и ввода показаний в процессор, а также генерирования внешней сиихронизации, которую может программировать процессор. Такое устройство называется *программируемым интервальным таймером/счетчиком событий*. Некоторыми областями применения такого устройства являются:

прерывание операционной системы с разделением времени через равномерные интервалы, чтобы она осуществляла переключение программ;

вывод точных временных сигналов с программируемыми периодами в устройство ввода-вывода (иапример, в аналого-цифровой преобразователь);

программируемая генерация скорости передачи в бодах;

измерение времеиной задержки между внешними событиями;

подсчет числа появлений событий во внешнем эксперименте и ввод показания в компьютер;

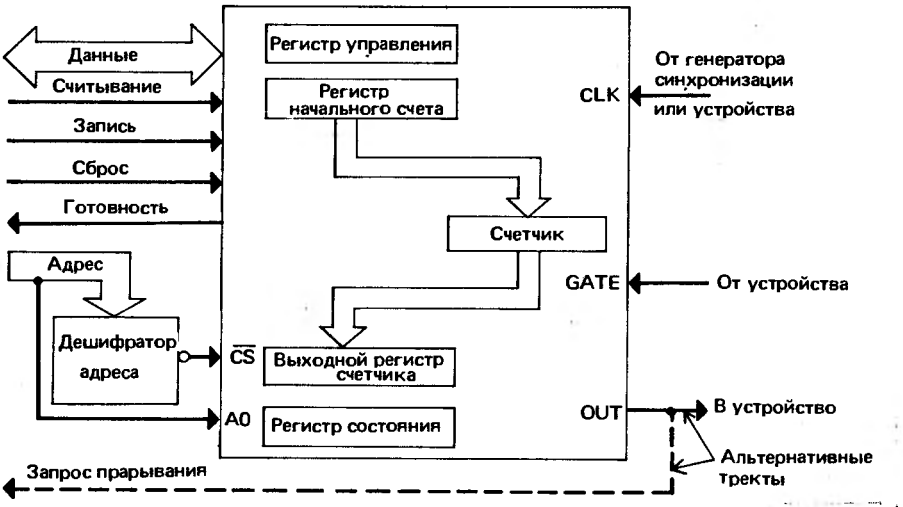


Рис. 9.25. Типичный интервальный таймер/счетчик событий

прерывание процессора после появления запрограммированного числа внешних событий.

Типичная организация интервального таймера/счетчика событий показана на рис. 9.25. Слева находятся четыре доступных компьютеру регистра: два верхних являются выходными портами, а два нижних – входными. Сам счетчик прямо процессору не доступен, но может инициализироваться из регистра начального счета и считывается посредством передачи его содержимого в выходной регистр счетчика. Счетчик запускается с начального значения и отсчитывает до 0. Вход CLK определяет скорость счета, сигнал GATE разрешает и запрещает вход CLK и, возможно, выполняет другие функции, а выход OUT становится активным при достижении счетчиком 0 или, возможно, при подаче сигнала GATE. Выход OUT подключается к линии запроса прерывания в системной шине, поэтому прерывание возникает при достижении счетчиком 0; его же можно подключить к устройству ввода-вывода для инициирования необходимых действий.

Устройство вводит значение в регистр начального счета, передает его в счетчик и выполняет счет "назад" (т. е. вычитание) импульсами со входа CLK. Текущее содержимое счетчика в любой момент можно ввести в процессор, не нарушая работы счетчика, посредством передачи его в выходной регистр счетчика с последующим считыванием из этого регистра. При буферировании содержимого счетчика не требуется вводить его в процессор немедленно. Индикация нуля в счетчике обычно фиксируется на выходе OUT и в одном бите регистра состояния. Поэтому для обнаружения нуля допускается применять программный ввод-вывод и ввод-вывод по прерываниям.

Регистр управления определяет режим работы и выполняет другие функции. Режим точно определяет, что происходит при достижении счетчиком 0 и (или) при подаче сигнала на вход GATE. Возможными действиями являются:

вход GATE применяется для разрешения и запрещения входа CLK;

вход GATE вызывает реинициализацию счетчика;

вход GATE прекращает счет и формирует высокий уровень на выходе OUT;

при достижении 0 счетчик выдает сигнал OUT и останавливается;

при достижении 0 счетчик выдает сигнал OUT и автоматически реинициализируется из регистра начального счета.

Режимы могут также определяться комбинациями перечисленных возможностей.

Рассмотрим, например, применение интервального таймера в операционной системе с разделением времени. В этом случае на вход CLK подаются сигналы синхронизации, а выход OUT подключается к линии запроса прерывания, возможно, немаскируемого прерывания. Вход GATE здесь не требуется. При включении системы в регистр начального счета загружается значение

$$\text{начальный счет} = \text{частота синхронизации} \times T,$$

где T – продолжительность каждого временного кванта в секундах. Задается такой режим, что при достижении счетчиком 0 содержимое регистра начального счета вновь загружается в счетчик, а выход OUT становится активным. Поскольку сигнал OUT используется как запрос прерывания, процедура прерывания для переключения программ будет выполняться с интервалом T секунд.

9.3.1. ПРОГРАММИРУЕМЫЙ ИНТЕРВАЛЬНЫЙ ТАЙМЕР

На рис. 9.26 представлена схема интервального таймера/счетчика событий 8254 фирмы Intel. В нем имеются три одинаковые счетные схемы со своими входами CLK и GATE и выходом OUT. Каждая схема имеет регистр управления и состояния, регистр счетчика (CR) для приема начального счета, счетного элемента (CE), который выполняет счет, но непосредственно процессору недоступен, и выходного регистра-защелки (OL) для фиксации содержимого CE, так что его может считать процессор. Полагается, что CR, CE и OL представляют собой пары 8-битных регистров. (Реальные схемы несколько отличаются от приведенных, но для программиста рисунок абсолютно точен.)

Обращения к регистрам производятся в соответствии с табл. 9.3.

Таблица 9.3

Обращения к регистрам таймера/счетчика 8254

$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	A1	A0	Передача
0	1	0	0	0	В счетчик 0 CR
0	1	0	0	1	В счетчик 1 CR
0	1	0	1	0	В счетчик 2 CR
0	1	0	1	1	В регистр управления или показывает приказ
0	0	1	0	0	Из счетчика 0 OL или регистра состояния
0	0	1	0	1	Из счетчика 1 OL или регистра состояния
0	0	1	1	0	Из счетчика 2 OL или регистра состояния

Все остальные комбинации приводят к тому, что линии данных переводятся в высокоимпедансное состояние. Когда A1 = A0 = 1, выполнение записи в регистр управления или выдачи приказа зависит от старшего бита выводимого байта. В последних трех комбинациях считывание OL или регистра состояния определяет предыдущий приказ.

Имеются два вида приказов. Приказ фиксации счетчика заставляет соответствующий OL зафиксировать содержимое CE счетчика, определяемого двумя старшими битами приказа. Приказ обратного считывания фиксирует комбинацию CE или "готовит" комбинацию регистров состояния для считывания. Подготовка регистра состояния означает

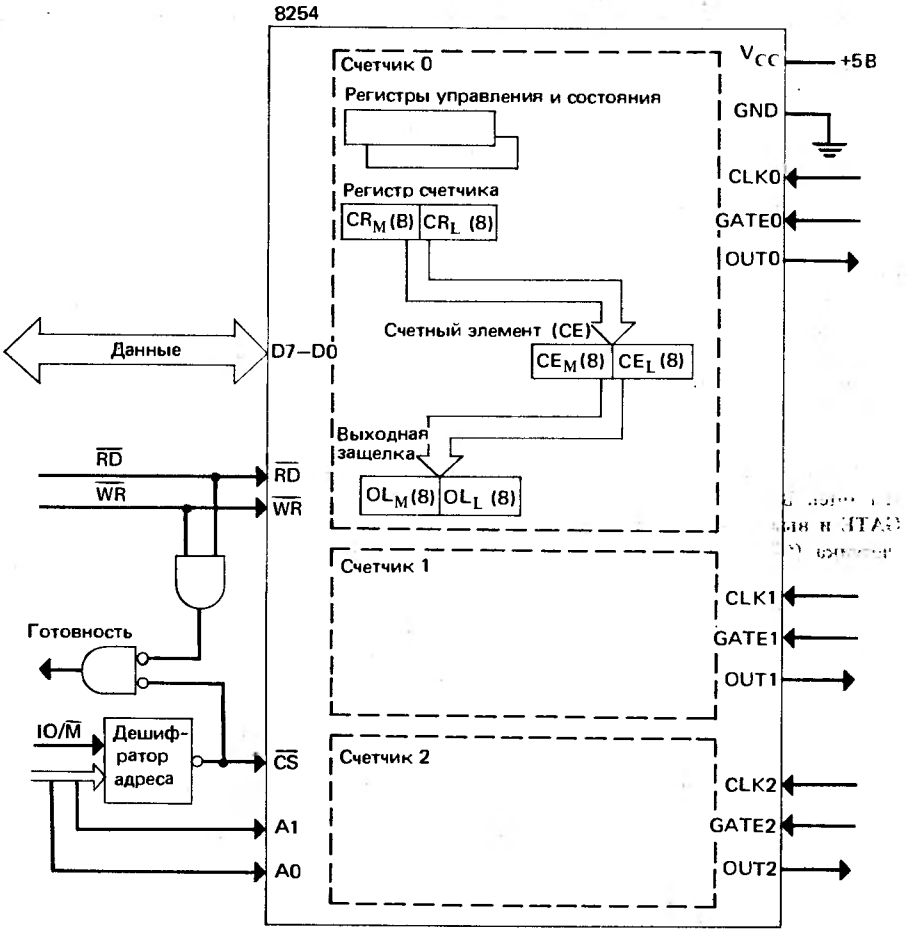


Рис. 9.26. Схема таймера/счетчика 8254

его считывание, когда в следующей операции считывания осуществляется ввод из счетчика. Состояния 00, 01 и 10 двух старших бит показывают приказ фиксации счетчика, а состояние 11 – приказ обратного считывания. В приказе фиксации биты 5 и 4 должны быть нулевыми, а остальные биты не используются. Приказ обратного считывания имеет следующий формат:

1	1	$\overline{\text{COUNT}}$	$\overline{\text{STAT}}$	CNT2	CNT1	CNT0	0

Если бит $\overline{\text{COUNT}} = 0$, фиксируются CE всех счетчиков, биты CNT которых содержат 1. Если, например, CNT0 = CNT2 = 1 и CNT1 = 0, то CE в счетчиках 0 и 2 фиксируются, а в счетчике 1 не фиксируется. Аналогично $\overline{\text{STAT}} = 0$ вызывает подготовку регистров

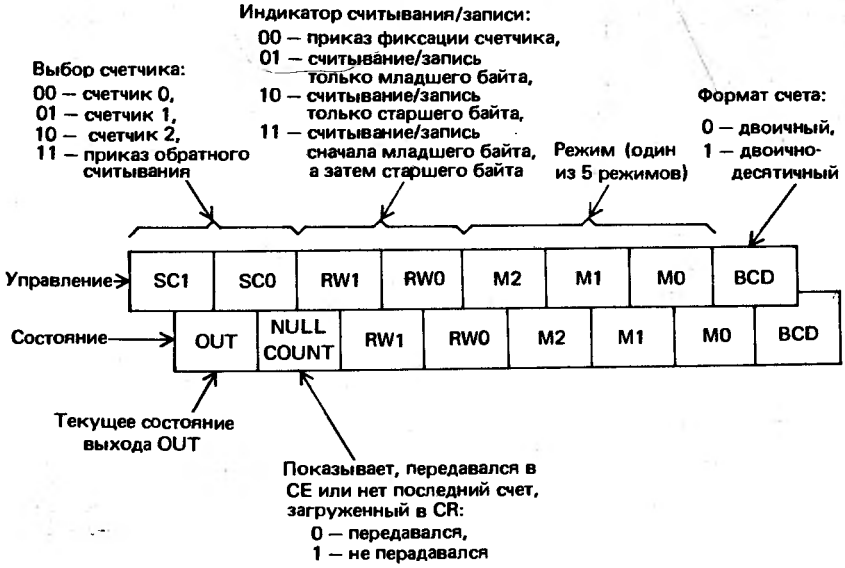


Рис. 9.27. Регистры управления и состояния счетчиков

состояний счетчиков для ввода. Действия по фиксации и подготовке допускается указывать в одном приказе.

Форматы регистров управления и состояния показаны на рис. 9.27. Если два старших бита равны 1, они задают приказ обратного считывания; в противном случае они определяют счетчик. Если указан счетчик и биты 5-4 содержат нули, то имеет место приказ фиксации, который направляется в регистр управления выбираемого старшими битами счетчика. Когда же комбинация в битах 5-4 отличается от 00, она указывает тип ввода из OL или вывода в CR. Комбинация 01 идентифицирует операции считывания/записи из (в) OL_L/CR_L, комбинация 10 – из (в) OL_M/CR_M и комбинация 11 – выполнение операций парами (первый байт из (в) OL_L/CR_L и второй из (в) OL_M/CR_M). Запись одного байта в CR вызывает сброс другого байта. Биты 1, 2 и 3 определяют режим, а бит 0 определяет формат счета.

Таймер/счетчик имеет следующие режимы работы (через N обозначен начальный счет):

Режим 0 (прерывания по концу счета). Сигнал GATE = 1 разрешает счет, а GATE = 0 запрещает счет, причем GATE не влияет на выход OUT. Содержимое CR передается в CE по первому импульсу CLK после того, как процессор осуществил запись в CR, независимо от сигнала на входе GATE. Импульс, который загружает CE, не учитывается при счете. На выходе OUT формируется низкий уровень при записи в регистр управления, который сохраняется до достижения счетчиком 0. Режим 0 предназначен в основном для счета событий.

Режим 1 (аппаратно перезапускаемый одновибратор). После загрузки значения N в CR переход 0 → 1 на входе GATE вызывает загрузку CE, переход 1 → 0 на выходе OUT и запускает счет. Когда счетчик достигает 0, на выходе OUT формируется высокий уровень; таким образом, результатом является отрицательный импульс на выходе OUT с продолжительностью N периодов синхронизации.

Режим 2 (периодический интервальный таймер). После загрузки значения N в CR следующий импульс синхронизации осуществляет передачу из CR в CE. На выходе OUT возникает переход 1 → 0, когда счетчик достигает 0; низкий уровень сохраняется в течение одного импульса CLK. Затем на выходе OUT появляется высокий уровень, производится повторная загрузка CE из CR; в результате на выходе OUT появляется отрицательный импульс через N тактов синхронизации. Сигнал GATE = 1 разрешает счет, а GATE = 0 – запрещает. Переход 0 → 1 на входе GATE вызывает реинициализацию счета следующим импульсом синхронизации. Данный режим применяется для реализации периодического интервального таймера.

Режим 3 (генератор прямоугольного сигнала). Аналогичен режиму 2, но на выходе OUT формируется низкий уровень при достижении половины начального счета; этот уровень сохраняется до достижения счетчиком 0. Как и прежде, сигнал GATE разрешает и запрещает счет, а его переход 0 → 1 реинициализирует счет. Этот режим применяется в генераторах, определяющих скорость передачи в бодах.

Режим 4 (программно-запускаемый строб). Аналогичен режиму 0, но на выходе OUT в процессе счета действует высокий уровень, а при достижении счетчиком 0 появляется отрицательный импульс с продолжительностью в один такт синхронизации.

Режим 5 (аппаратно-запускаемый строб с перезапуском). После загрузки CR переход 0 → 1 на входе GATE вызывает передачу из CR в CE следующим импульсом CLK. В процессе счета на выходе OUT действует высокий уровень, а при достижении счетчиком 0 формируется отрицательный импульс с продолжительностью в один период CLK. Сигнал GATE может в любой момент времени реинициализировать счет.

Начальное значение счета 0 во всех режимах интерпретируется как 2^{16} или 10^4 в зависимости от формата счета. Мы привели общие принципы работы микросхемы 8254, а подробное описание содержится в фирменных материалах.

9.3.2. ПРИМЕНЕНИЕ ТАЙМЕРА В АНАЛОГО-ЦИФРОВОЙ ПОДСИСТЕМЕ

Применение микросхемы 8254 в качестве программируемого генератора частоты отсчетов в аналого-цифровой подсистеме представлено на рис. 9.28. Благодаря наличию в ней трех счетчиков можно не только программировать частоту опросов, но и определять временные интервалы, в течение которых берутся отсчеты. Предпопожим, что счетчик 0 работает в режиме 2, счетчик 1 – в режиме 1, счетчик 2 – в режиме 3 и что начальные значения для них равны L, M и N. Если F – частота синхронизации, то на входе CLK1 действует частота F/N и на выходе OUT1 формируется импульс с продолжительностью MN/F. Следовательно, в интервале MN/F секунд на выходе OUT0 появляются импульсы с частотой F/L. При подключении OUT0 на вход "преобразование" аналого-цифрового преобразователя за MN/F секунд будут формироваться отсчеты с частотой F/L после того, как все три счетчика инициализированы и реле или тумблер замыкаются. После передачи в порт A преобразованного отсчета формируется запрос прерывания $INTR_A$ и процедура прерывания вводит отсчет в процессор. На рис. 9.29 приведена последовательность инициализации системы. Предполагается, что таймер/счетчик 8254 имеет адреса 0070-0073; переменные LCNT, MCNT и NCNT содержат L, M и N; L и N меньше 256. Последовательность инициализации задает режимы счетчиков и загружает начальные значения в CR. Начальные значения L и N представлены в двоичном формате, а значение M – в BCD-формате.

9.4. КЛАВИАТУРА И ИНДИКАТОР

Пульты управления (или консоли) небольших дешевых систем часто реализуются как устройства ввода и вывода простых клавиатуры и индикатора. С помощью клавиатуры данные, адреса памяти и машинные коды вводятся в удобном 16-ричном формате.

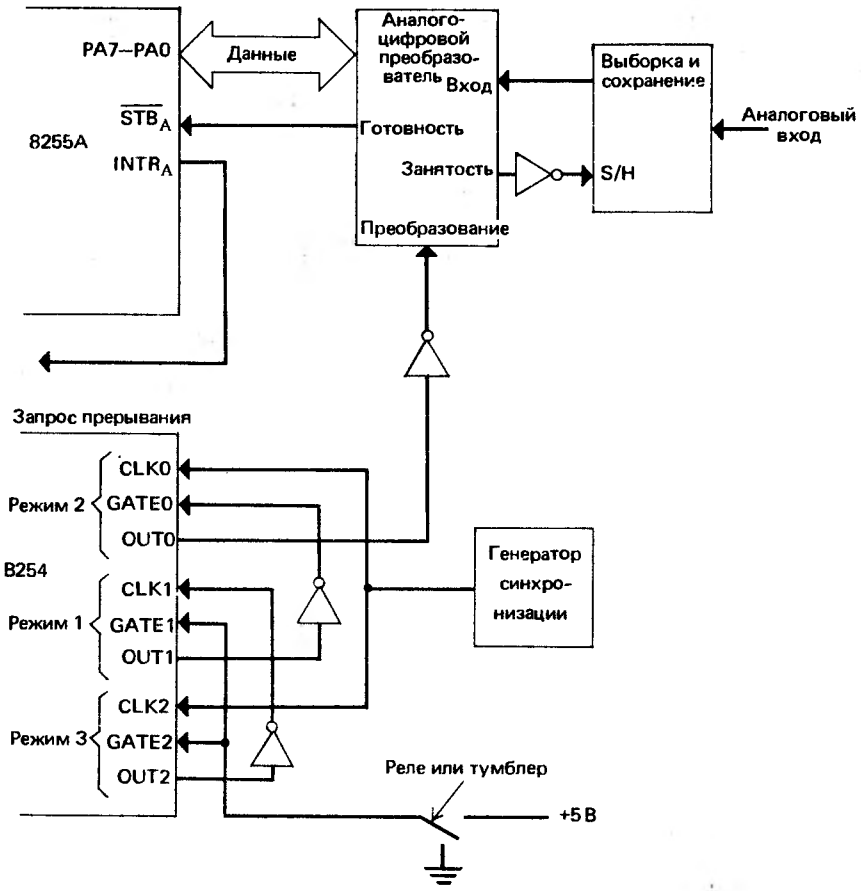


Рис. 9.28. Применение интервального таймера

```
MOV     AL,00010100B        ;ВЫВЕСТИ В СЧЕТЧИК 0
OUT     73H,AL              ;УПРАВЛЕНИЕ - РЕЖИМ 2
MOV     AL,LCNT             ;ВЫВЕСТИ В СЧЕТЧИК 0
OUT     70H,AL              ;НАЧАЛЬНЫЙ ОТСЧЕТ - ДВОИЧНЫЙ
MOV     AL,01110011B        ;ВЫВЕСТИ В СЧЕТЧИК 1
OUT     73H,AL              ;УПРАВЛЕНИЕ - РЕЖИМ 1
MOV     AX,MCNT             ;ВЫВЕСТИ В СЧЕТЧИК 1
OUT     71H,AL              ;НАЧАЛЬНЫЙ ОТСЧЕТ - BCD
MOV     AL,AH
OUT     71H,AL
MOV     AL,10010110B        ;ВЫВЕСТИ В СЧЕТЧИК 2
OUT     73H,AL              ;УПРАВЛЕНИЕ - РЕЖИМ 3
MOV     AL,NCNT             ;ВЫВЕСТИ В СЧЕТЧИК 2
OUT     72H,AL              ;НАЧАЛЬНЫЙ ОТСЧЕТ - ДВОИЧНЫЙ
.
.
.
```

Рис. 9.29. Инициализация счетчиков в системе, показанной на рис. 9.28

Кроме цифровых клавиш клавиатура имеет функциональные клавиши для ввода приказов управления. При выводе адреса памяти и данные отображаются на светодиодных индикаторах.

9.4.1. СХЕМА КЛАВИАТУРЫ

В отличие от терминала клавиатура с механическими контактами, клавишные переключатели которой представлены матрицей, не имеет никаких электронных схем. На рис. 9.30 показано подключение 64-клавишной клавиатуры к микрокомпьютеру через два параллельных порта ввода-вывода, например, микросхемы 8255А. При нажатии кла-

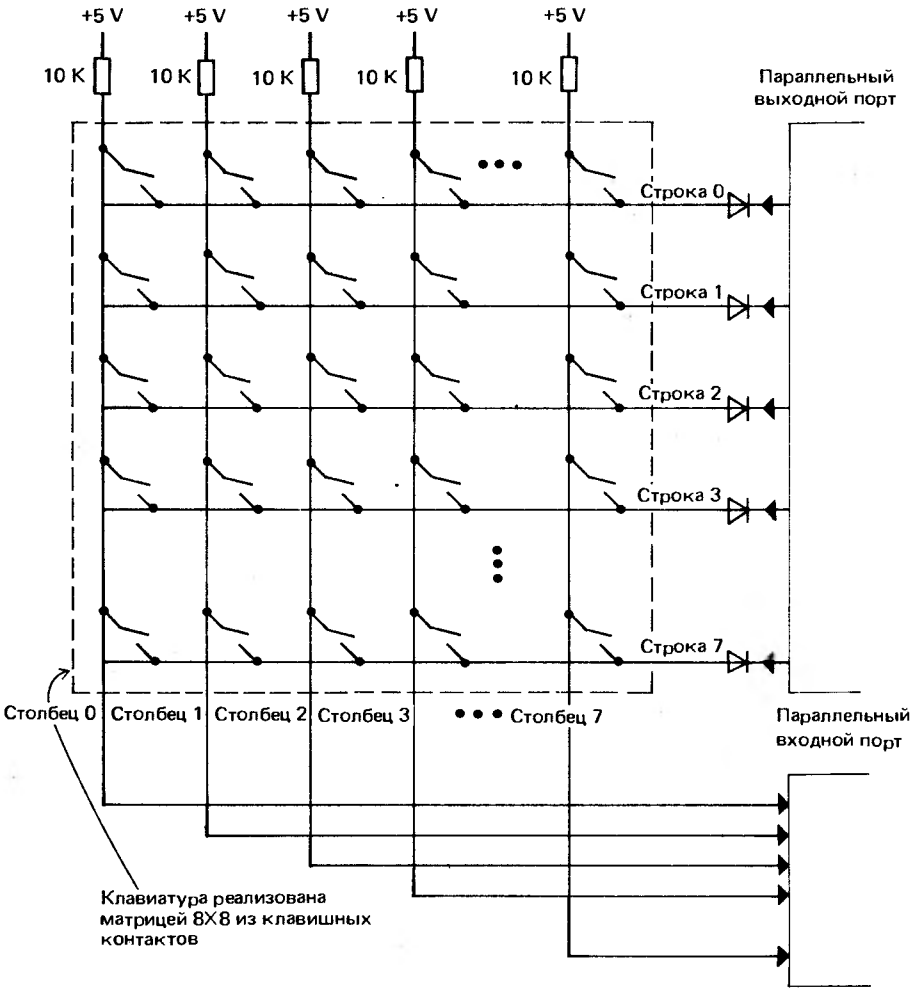


Рис. 9.30. Организация механической клавиатуры

виши соответствующие строка и столбец замыкаются, образуя соединение. Обнаруживая позиции строки и столбца, в которых произошло замыкание контактов, определяют кодовое слово, соответствующее нажатой клавише. Данный процесс называется сканированием клавиатуры и реализуется следующим образом. Выходной порт посылает сигнал 0 в строку 0 и сигналы 1 во все остальные строки. Затем считываются и проверяются линии столбцов. Если в строке 0 нет сигнала 0, процесс повторяется для строки 1, затем для строки 2 и т. д. Когда же фиксируется 0, обнаруживается иажатая клавиша, позиция строки которой известна по выводимой комбинации, а позиция столбца – по результату ввода. Объединяя позиции строки и столбца обнаруженного 0, можно образовать однозначное слово, показывающее положение нажатой клавиши.

С клавиатурой связаны две сложности: дребезг контактов и одновременное нажатие нескольких клавиш. Когда клавиша нажимается или отпускается, ее контакт несколько раз перескакивает ("дрожит") из одного состояния в другое до перехода в устойчивое замкнутое или разомкнутое положение. Продолжительность дребезга не постоянна, но обычно меньше 10 мс. Чтобы предотвратить обнаружение ложных срабатываний клавиш, замыкания контактов из-за дребезга необходимо игнорировать – эта операция называется *подавлением дребезга*. Подавление дребезга осуществляется аппаратно или программно, но программный подход ведет к чрезмерным потерям времени процессора. Наиболее простое решение проблемы одновременного срабатывания нескольких клавиш заключается в том, чтобы сканировать весь массив и вводить замыкания в порядке их обнаружения.

9.4.2. СХЕМА ИНДИКАТОРА

Имеется много разновидностей цифровых и буквенно-цифровых индикаторов. Для индикации 16-ричных цифр обычно применяются семисегментные светодиодные индикаторы (рис. 9.31). Цифра в семисегментном коде подается на входы *a-g* и DP (десятичная точка). В зависимости от типа индикатора (с общим анодом или с общим катодом) активные сигналы имеют высокий или низкий уровень. Сегмент начинает светиться, когда соответствующий светодиод смещен в прямом направлении.

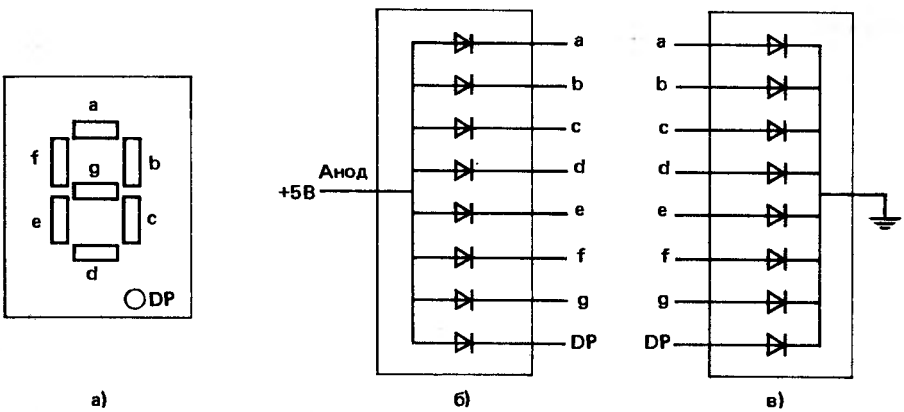


Рис. 9.31. Типичный семисегментный светодиодный индикатор (*а*), индикатор с общим анодом (*б*) и общим катодом (*в*)

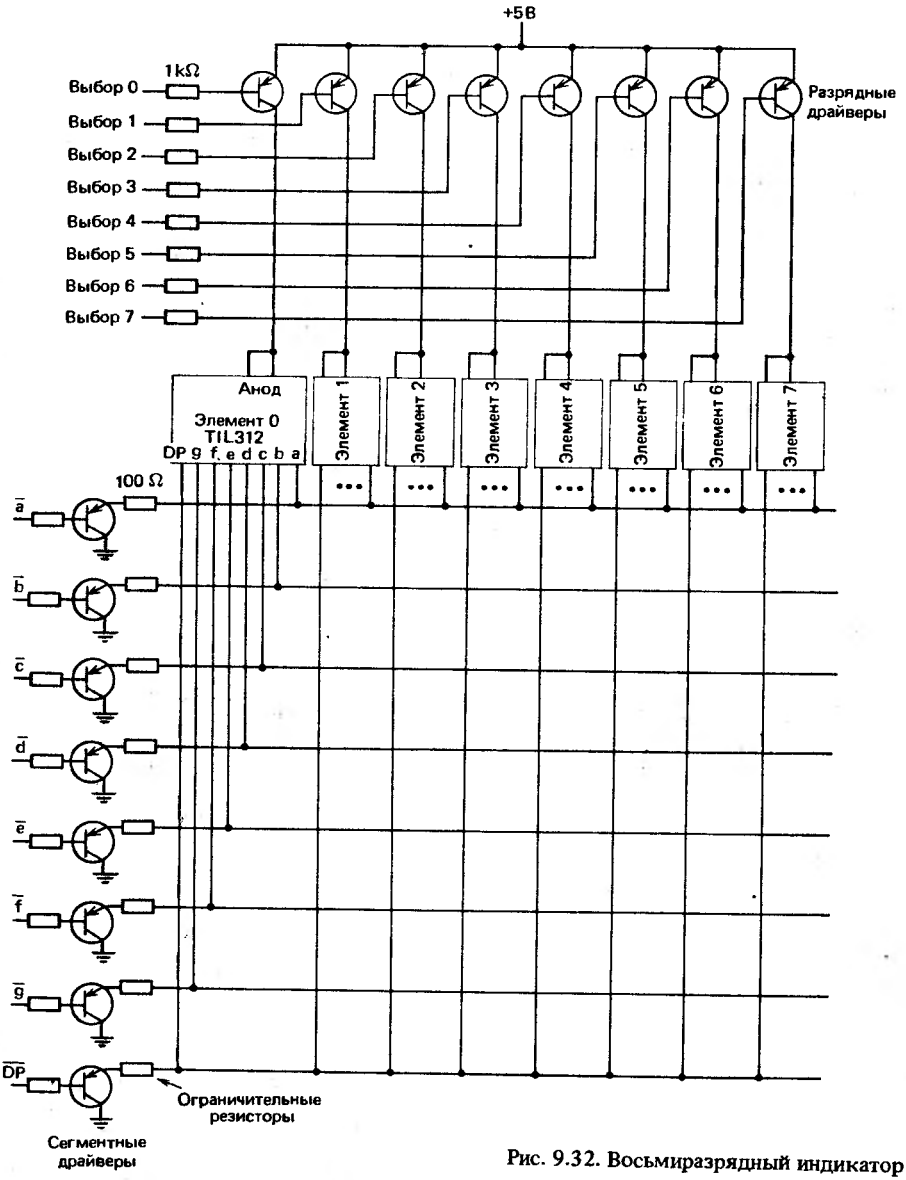


Рис. 9.32. Восьмиразрядный индикатор

На рис. 9.32 показан многоразрядный индикатор, образованный из 8 элементов. Чтобы не применять в каждом элементе регистр-защелку данных, все элементы подключены к двум 8-битным параллельным портам и работают в мультиплексном режиме. Все элементы разделяют общие линии сегментов, а с помощью линий выбора разряда в любой момент времени выбирается только один элемент. Каждый индикатор включается

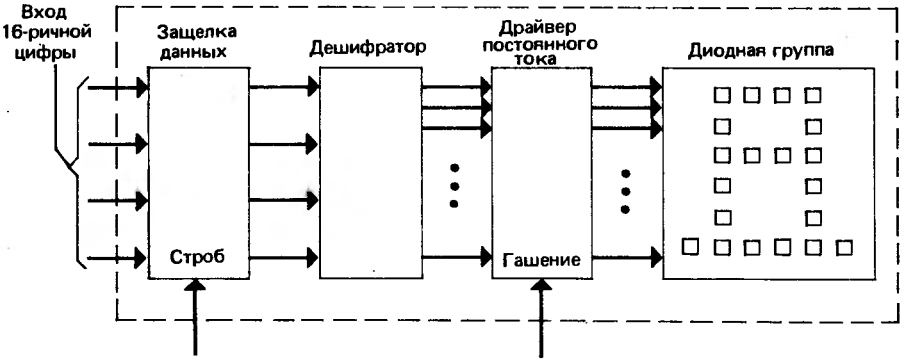


Рис. 9.33. Точечно-матричный 16-ричный индикатор

на 1 мс, а после прохождения всех разрядов последовательность операций повторяется, начиная с первого разряда. Таким образом, индикаторы периодически регенерируют, создавая иллюзию непрерывно работающего многоразрядного индикатора.

Так как типичный средний ток включенного сегмента достигает 20 мА и в каждом разряде может светиться несколько сегментов, суммарный ток превышает нагрузочную способность обычного ТТЛ-вентиля. Следовательно, в индикаторе потребуются драйверы разрядов и драйверы сегментов. Если напряжения на базе транзистора драйвера разряда мало, этот транзистор включается и подсоединяет индикатор к источнику +5 В. Если напряжение на базе транзистора драйвера сегмента мало, этот транзистор включается и через сегмент начинает протекать ток. Кроме того, так как падение напряжения на смещенном в прямом направлении диоде сегмента постоянно, для ограничения тока последовательно с каждым сегментом включается резистор, защищающий индикатор и драйвер. Расчетное сопротивление резистора определяется требуемой яркостью свечения.

Еще один тип индикатора 16-ричных цифр показан на рис. 9.33. В этом индикаторе TIL311 применяется точечная матрица из 20 светодиодов. На его вход подается 4-битное двоичное число, которое сам индикатор преобразует в видимое изображение эквивалентной 16-ричной цифры. Благодаря встроенным регистру-защелке и драйверам постоянного тока входы этого индикатора ТТЛ-совместимы. Входной регистр-защелка запоминает данные, и регенерации не требуется.

9.4.3. КОНТРОЛЛЕР КЛАВИАТУРЫ/ИНДИКАТОРА

Хотя подключения клавиатуры и многоразрядного индикатора к параллельным портам ввода-вывода в схемном отношении просты, при вводе и выводе процессор должен выполнять процедуры сканирования клавиатуры и регенерации индикатора. Микросхема 8279 контроллера клавиатуры/индикатора освобождает процессор от выполнения длительных операций сканирования и регенерации.

Общая структура контроллера 8279 и его интерфейс с шиной представлены на рис. 9.34. Регистры управления и состояния разделяют нечетный адрес, а буферный регистр данных имеет четный адрес. Адресация регистров осуществляется в соответствии с табл. 9.4.

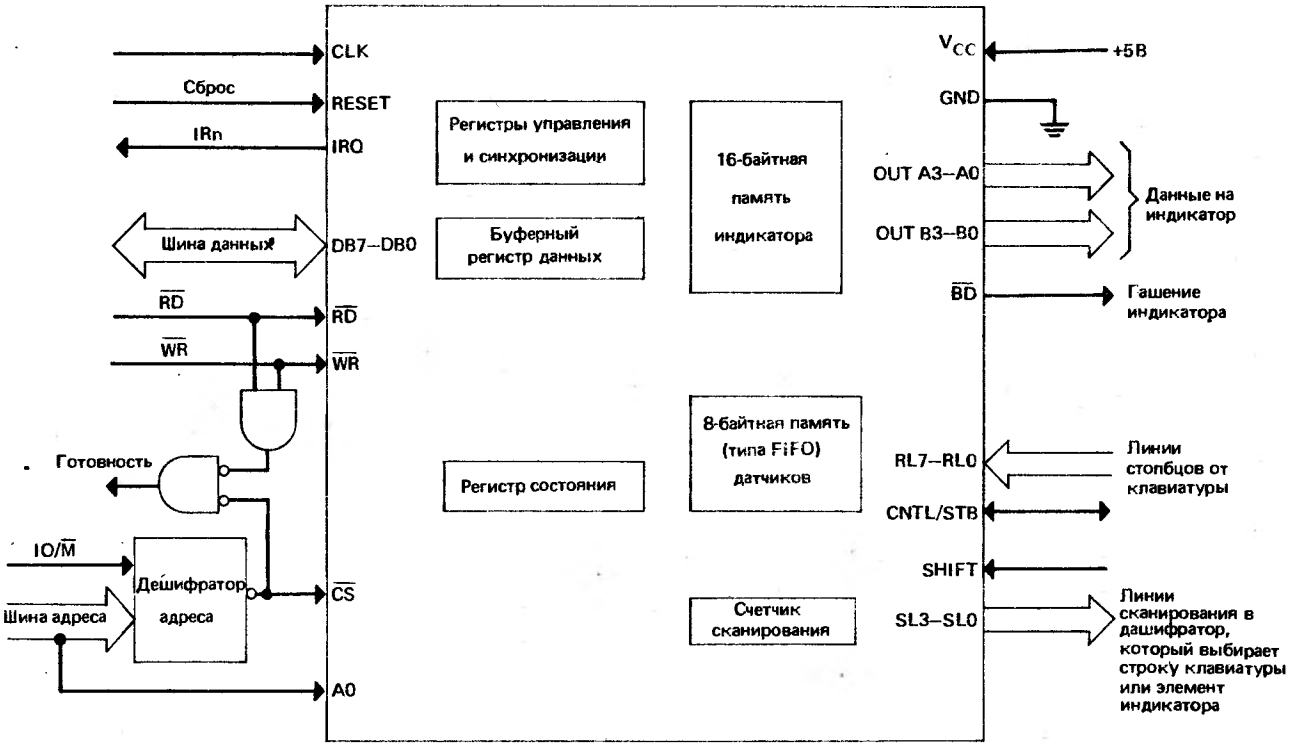


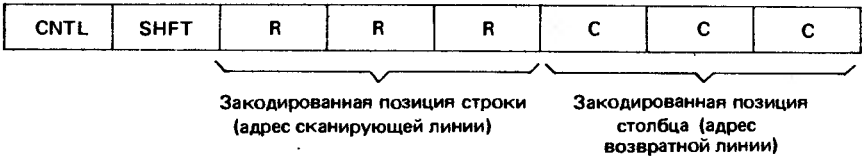
Рис. 9.34. Структура контроллера клавиатуры/индикатора

Таблица 9.4

Адресация регистров контроллера 8279

CS	$\overline{RD}$	$\overline{WR}$	A0	Описание передачи
0	1	0	0	Шина данных в буферный регистр данных
0	1	0	1	Шина данных в регистр управления
0	0	1	0	Буферный регистр данных на шину данных
0	0	1	1	Регистр состояния на шину данных

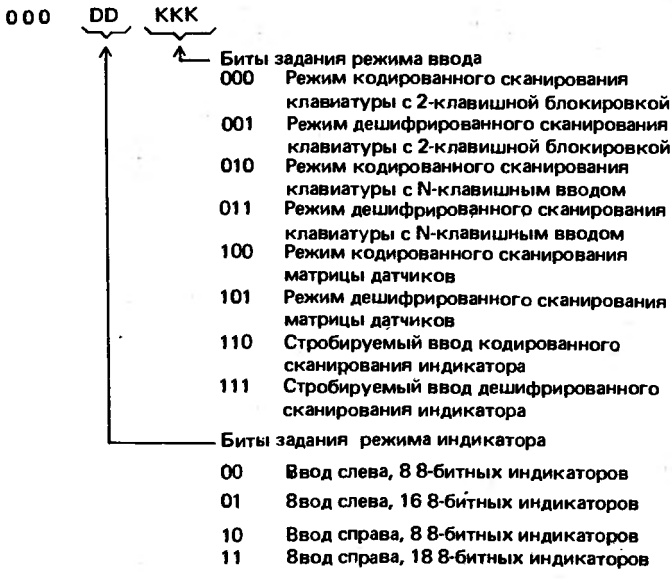
Управляя клавиатурой, контроллер непрерывно сканнрует каждую строку клавиатуры посредством выдачи адресов строк на линии SL2-SL0 и ввода сигналов с возвратных линий RL7-RL0, представляющих собой адреса столбцов. Отметнм, что линии SL3-SL0 применяются и для сканирования клавиатуры, и для регенерации индикатора; контроллер может обслуживать до 16 индикаторов. Когда обнаруживается нажатая клавиша, осуществляется подавление дребезга – вводится цикл ожиданйя и через 10 мс проверяется, осталась ли клавиша нажатой. Если клавиша нажата, формируется 8-битное кодовое слово, соответствующее позиции клавиши; в этом слове объединяются закодированные позиции строки и столбца, состояние клавиши переключения регистров и клавиши управления. Кодовое слово имеет следующий формат:



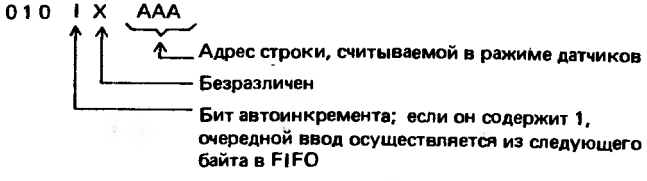
Входы SHFT и CNTL предназначены для клавиатуры типа клавиатуры пишущей машинки, в которой имеются клавиши управления и переключения регистров. Затем позиция клавиши вводится в память датчиков 8 X 8 с организацией FIFO и выдается сигнал IRQ (запрос прерывания), если до этого память датчиков была пустой.

Регистры управления и синхронизации физически представляют собой набор флажков и регистров, обращения к которым осуществляют приказы, посылаемые по нечетному адресу. Три старших бита приказа определяют его тип, а смысл остальных пяти бит зависит от типа. Мы рассмотрим три приказа, хотя всего их восемь (описание остальных типов и подробная информация о контроллере имеется в фирменных материалах). Интересующие нас три приказа имеют следующие форматы.

Установка режимов клавиатуры и индикатора. Определяет режимы ввода и индикации; применяется для инициализации контроллера. Формат приказа:

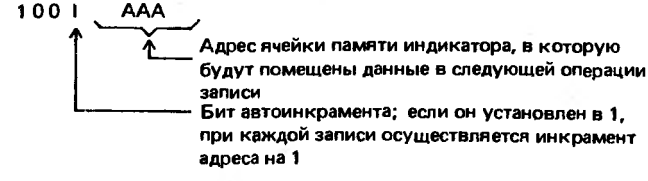


Считывание памяти датчиков (FIFO). Определяет, что считывание из буферного регистра данных будет вводить байт из памяти FIFO, и, если контроллер находится в режиме датчиков, показывает, какая строка считывается. Этот приказ необходимо подать до ввода данных из памяти FIFO. Формат приказа:



Отметим, что, если режим ввода является режимом сканирования клавиатуры, считывание всегда производится из байта, который был помещен в FIFO первым; следовательно, биты I и AAA игнорируются.

Запись в память индикатора. Показывает, что запись в буферный регистр данных будет помещать данные в память индикатора. Этот приказ необходимо выдать до того, как процессор будет выводить в контроллер индицируемые символы. Формат приказа:



Контроллер 8279 предлагает два варианта обработки ситуации, когда примерно в одно и то же время нажимаются несколько клавиш. Вариант 2-клавишной блокировки:

если вторая клавиша нажимается в то время, когда подавляется дребезг первой клавиши, в FIFO будет помещено кодовое слово той клавиши, которая освобождается последней. Если вторая клавиша нажимается в пределах двух циклов сканирования после того, как подавлен дребезг первой клавиши, и первая клавиша остается нажатой после освобождения второй, распознается первая нажатая клавиша. В противном случае обе клавиши игнорируются. В варианте N-клавишного ввода нажатие каждой клавиши анализируется независимо. Если нажимаются несколько клавиш, после нажатия они все вводятся в том порядке, в каком воспринимаются.

Контроллер 8279 имеет также режим матрицы датчиков, в котором сигналы с возвратных линий запоминаются в строке FIFO, соответствующей адресу сканирования. В отличие от режима сканирования клавиатура состояния SHFT и CNTL и адрес сканирования не вводятся. Здесь FIFO хранит образ состояния матрицы датчиков; этот режим удобен, когда информация от нескольких устройств вводится посредством опроса каждого устройства по линиям сканирования.

Состояние FIFO отображается в регистре состояния. Биты 0, 1 и 2 этого регистра показывают число находящихся в FIFO байт данных, а равный 1 бит 3 сигнализирует о заполнении FIFO. Биты 4 и 5 фиксируют антипереполнение (оно возникает при попытке считать из пустой памяти FIFO) и переполнение (возникающее при попытке ввести данные в заполненную память FIFO). В обоих случаях состояние 1 означает наличие ошибки. Состояние 1 бита 6 показывает замыкание, когда контроллер находится в режиме матрицы датчиков, и многократное замыкание, когда он работает в режиме специальной ошибки. Бит 7 показывает доступность индикатора.

Для управления индикатором в контроллере предусмотрены 16-байтная память индикатора и логика регенерации. Каждый адрес в этой памяти соответствует одному разряду индикатора, причем адрес 0 относится к левому разряду. После того как процессор загрузил в память подлежащие индикации символы, контроллеру не нужны дополнительные инструкции и процессор освобождается от регенерации индикатора. Вывод заключается в том, что контроллер 8279 периодически выдает символы на линии OUT A3-A0 и OUT B3-B0 и адреса выбора разрядов на линии SL3-SL0. Хотя память индикатора допускает прямую адресацию, процессор может последовательно вводить данные в нее слева или справа. При выводе слева с автоинкрементом после каждой записи в память производится инкремент адреса на 1, поэтому следующий символ появляется в следующем разряде справа. Автоинкрементный вывод справа позволяет индицировать символы в той форме, которая принята в большинстве калькуляторов. Индикация сдвигается влево на один символ, а следующий символ появляется справа.

Один из вариантов подключения к контроллеру 64-клавишной клавиатуры и 8-разрядного семисегментного индикатора показан на рис. 9.35. Клавиатура и индикатор сканируются и регенерируются под управлением сигналов выбора SL2-SL0. Выход SL3 не подключен, так как индикатор имеет только восемь разрядов. Выходы обоих дешифраторов имеют активные сигналы низкого уровня. Один дешифратор выбирает строку клавиатуры, а другой разрешает один из 8-разрядных драйверов.

Чтобы показать программирование контроллера, предположим, что он подключен к клавиатуре и многоразрядному индикатору в соответствии с рис. 9.35, имеет адреса FFE8 и FFE9 и выход запроса прерывания IRQ не используется. Вначале необходимо инициализировать прибор, посылая в регистр управления приказ установки режима. Следующие команды устанавливают контроллер в режим кодированного сканирования клавиатуры с 2-клавишной блокировкой и режим ввода слева 8-разрядного индикатора:

```
MOV  DX,0FFE9H
MOV  AL,0
OUT  DX,AL
```

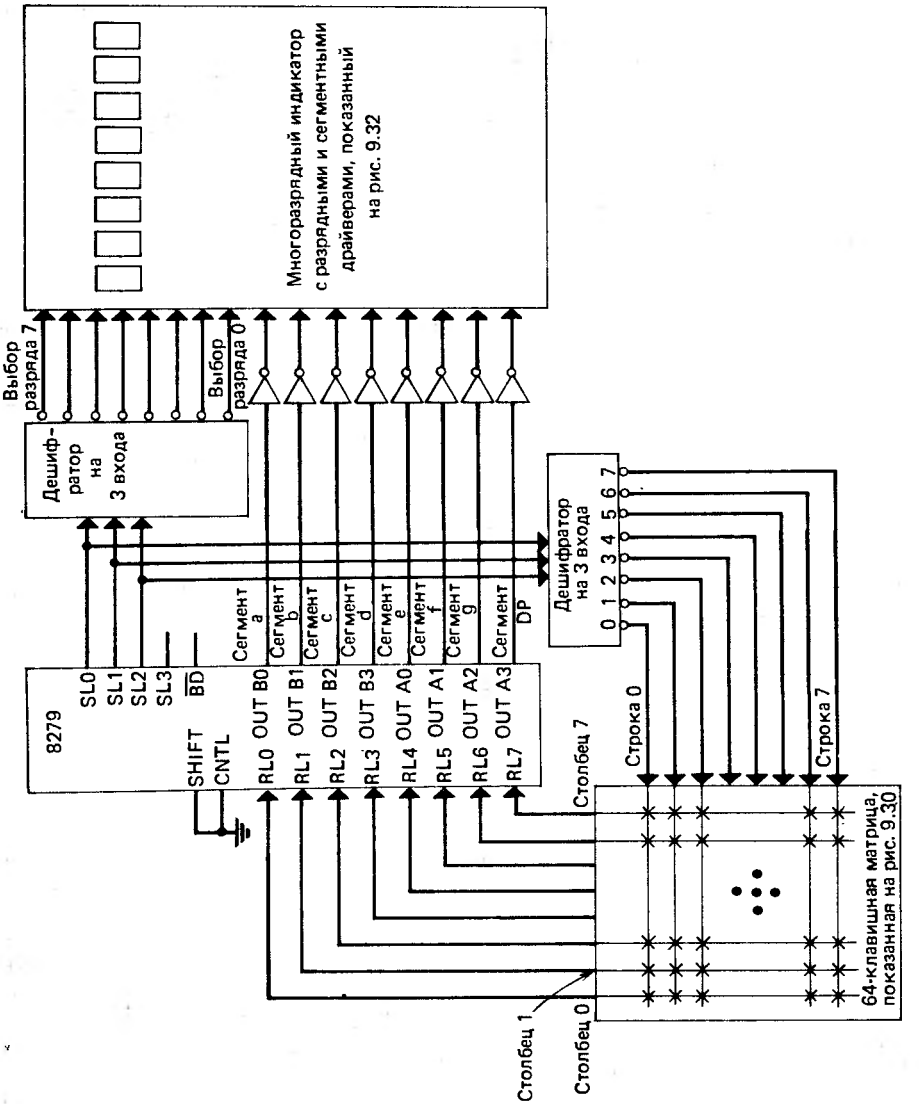


Рис. 9.35. Применение контроллера 8279 в интерфейсе клавиатуры и многоразрядного индикатора

Таким образом, символы, сформированные нажатыми клавишами, можно считать через память FIFO. Далее приведен программный фрагмент, в котором применяется программный ввод-вывод для ввода восьми кодовых слов и запоминания их в 8-байтном массиве KEYS (первый байт находится по старшему адресу):

```
        MOV   SI,8
        MOV   DX,0FFE9H
        MOV   AL,01000000B
        OUT   DX,AL
NEXT:   MOV   DX,0FFE9H
IDLE:   IN    AL,DX
        TEST  AL,0FH
        JZ    IDLE
        MOV   DX,0FFE8H
        IN    AL,DX
        MOV   KEYS[SI-1],AL
        DEC   SI
        JNZ   NEXT
```

Первая команда инициализирует счетчик в SI, следующие три задают ввод по четному адресу как ввод из FIFO. Очередные три команды, начиная с метки IDLE, заставляют процессор ожидать до готовности ввода, а находящиеся за ними еще три команды передают введенные данные в KEYS. Последние две команды вызывают повторение последовательности до тех пор, пока не будут введены восемь символов.

Для индикации символов процессор должен сначала выдать приказ записи в память индикатора, а затем выводить в нее данные. Следующий фрагмент индицирует восемь цифр, которые хранятся, начиная с DIGITS (младшая цифра хранится по меньшему адресу):

```
        MOV   SI,8
        MOV   DX,0FFE9H
        MOV   AL,10010000B
        OUT   DX,AL
        MOV   DX,0FFE8H
AGAIN:  MOV   AL,DIGITS[SI-1]
        OUT   DX,AL
        DEC   SI
        JNZ   AGAIN
```

Первая команда образует в SI счетчик цифр, следующие три выводят приказ записи в память индикатора, а находящаяся за ними команда загружает в DX адрес буферного регистра данных. Последние команды образуют цикл вывода цифр в память индикатора.

9.5. КОНТРОЛЛЕРЫ ПРЯМОГО ДОСТУПА К ПАМЯТИ

В гл. 6 было показано, что контроллер ПДП может стать ведущим шины и управлять передачей данных между интерфейсом ввода-вывода и памятью или интерфейсом внешней памяти и памятью. В процессе передачи он должен помещать адреса на шину, а также выдавать и принимать сигналы квитирования так, как это делает логика управления шиной. Назначение контроллера ПДП – осуществлять последовательность передач (т. е. передачу блока) посредством пропусков ("кражи") циклов шины.

Обычно контроллер ПДП рассчитан на обслуживание одного или нескольких интерфейсов ввода-вывода или внешней памяти и каждый интерфейс подключается к контроллеру набором проводников. Часть контроллера ПДП, предназначенная для обслуживания одного интерфейса, называется *каналом*.

Общая организация одноканального контроллера ПДП и его основные связи показаны на рис. 9.36. Кроме обычных регистров управления и состояния, в каждом канале должны быть регистр адреса и счетчик байт (или слов). Инициализация контроллера со-

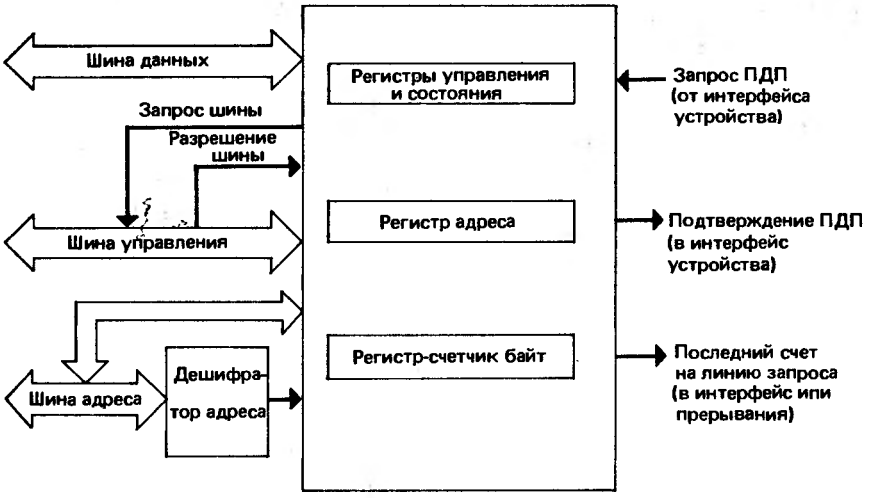


Рис. 9.36. Общая организация контроллера ПДП

стоит в загрузке в эти регистры начального (конечного) адреса массива в памяти, который служит буфером, и числа передаваемых байт (слов). При вводе в память при наличии данных интерфейс делает запрос ПДП. Затем контроллер формирует запрос шины и при получении разрешения шины выводит содержимое регистра адреса на шину адреса, посылает подтверждение в интерфейс и выдает сигналы считывания ввода-вывода и записи в память. После этого интерфейс помещает данные на шину данных и снимает свой запрос. Когда память воспринимает данные, она возвращает в контроллер сигнал готовности; контроллер осуществляет инкремент (декремент) регистра адреса, декремент счетчика байт (слов) и снимает запрос шины. При достижении счетчиком нуля процесс прекращается и выдается сигнал в процессор как запрос прерывания или в интерфейс для извещения о том, что передачи завершены. Вывод реализуется аналогично, но контроллер выдает сигналы записи ввода-вывода и считывания из памяти, а данные передаются в другом направлении.

В качестве примера рассмотрим контроллер ПДП 8237 фирмы Intel. Его общая организация вместе с дополнительной логикой, необходимой в системе на базе микропроцессора 8088, приведена на рис. 9.37, а конфигурация в минимальном режиме с контроллером ПДП – на рис. 9.38.

Когда данные помещаются в регистры контроллера или считываются из них, контроллер ведет себя как любой другой интерфейс ввода-вывода. В качестве ведомого он воспринимает 16-битные адреса, причем 12 их старших бит определяют выбор микросхемы, а 4 младших применяются для внутренней адресации. Когда сигналы HRQ = $\overline{CS}$ = = 0, контроллер превращается в ведомого и линии $\overline{IOR}$ и $\overline{IOW}$ являются входными. Процессор может обратиться к внутренним регистрам, формируя активный сигнал $\overline{IOR}$ или $\overline{IOW}$. Сигнал AEN, которым ведущий контроллер идентифицирует выдачу адреса, находится в состоянии 0 до тех пор, пока система взаимодействует с внутренними регистрами.

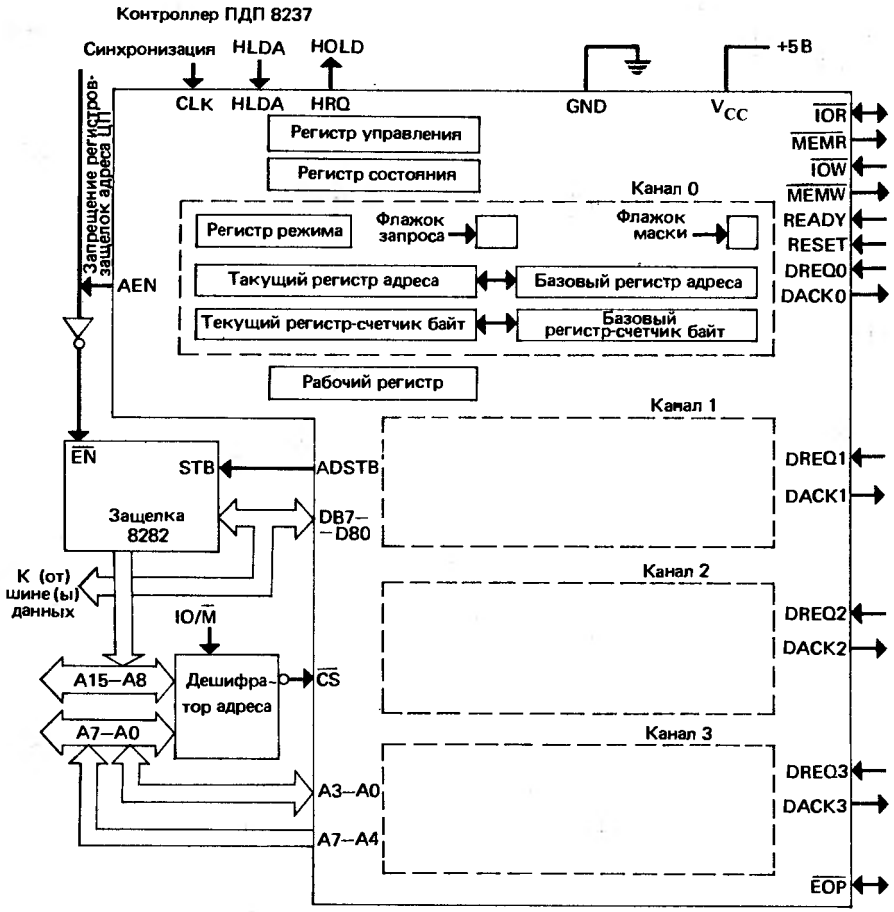


Рис. 9.37. Организация контроллера ПДП 8237

В качестве ведущего контроллер должен выдавать адреса шины. Младший байт адреса выводится на линии A7-A0, а старший – на линии DB7-DB0 с одновременной установкой AEN = 1. При таком уровне сигнала AEN разрешается внешний регистр-защелка адреса, обеспечивая вывод старшего байта адреса на линии A15-A8. Сигнал AEN применяется также для запрещения регистров-защелок адреса 8282, подключенных к линиям A19-A8 и AD7-AD0 процессора. Через небольшой временной интервал после выдачи сигнала AEN на выходе строба адреса (ADSTB) формируется импульс. Он применяется для загрузки бит 8-15 адреса в регистры-защелки 8282, подключенные к линиям A15-A8. Старшие 4 бита адреса A19-A16 контроллер 8237 не выдает, и их необходимо программировать отдельно до начала блоковой передачи. Для хранения этих бит адреса потребуется 4-битный порт ввода-вывода, в который можно вывести данные, как в любой другой порт. Во время блоковой передачи содержимое этого порта изменять нельзя; следовательно, за одну передачу можно передать максимум 2^{16} байт.

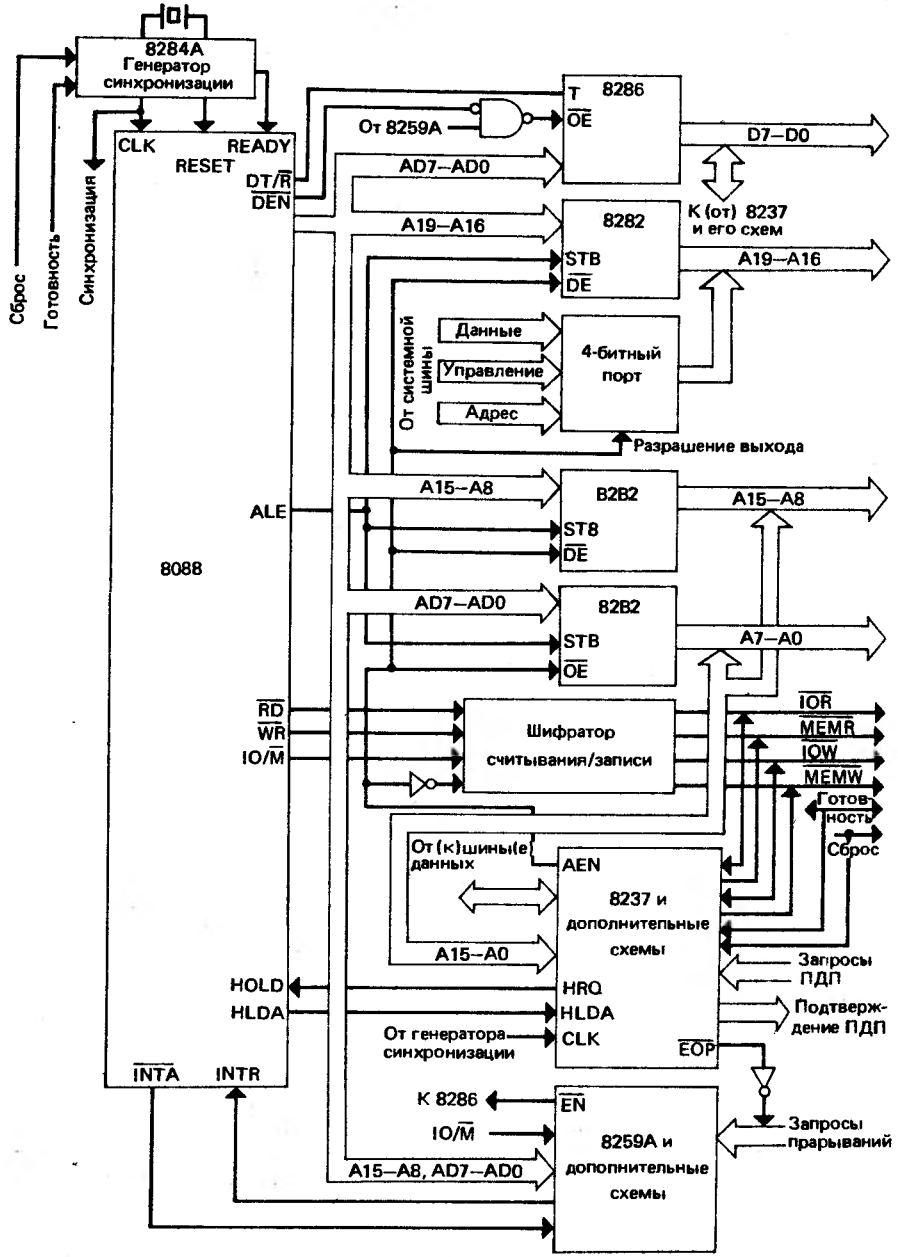


Рис. 9.38. Система на базе микропроцессора 8088 в минимальном режиме с контроллером 8237

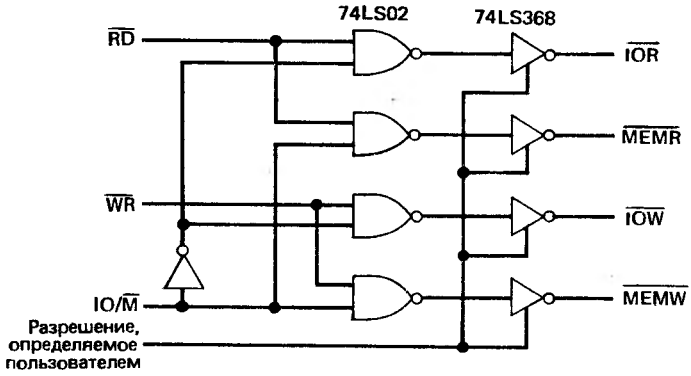


Рис. 9.39. Схема кодирования сигналов $\overline{RD}$, $\overline{WR}$ и IO/$\overline{M}$

Примечание. Если не требуются тристабильные линии приказов, можно использовать дешифратор (8205 или 74S138). Микросхема 74LS257 не рекомендуется из-за возможных выбросов напряжения, когда она переходит в высокоимпедансное состояние или выходит из него

Являясь ведущим, контроллер должен выводить также необходимые сигналы считывания/записи. Это уже знакомые нам сигналы $\overline{IOR}$ (считывание ввода-вывода), $\overline{MEMR}$ (считывание из памяти), $\overline{IOW}$ (запись ввода-вывода) и $\overline{MEMW}$ (запись в память). Так как эти сигналы не соответствуют выходным сигналам $\overline{RD}$, $\overline{WR}$ и IO/$\overline{M}$ микропроцессора 8088 в минимальном режиме, для преобразования сигналов контроллера потребуется схема кодирования сигналов считывания/записи (рис. 9.39). Во время передачи ПДП контроллер запрещает выходы схемы кодирования считывания/записи сигналом AEN. Конечно, интерфейсы памяти и ввода-вывода в системе необходимо спроектировать так, чтобы они реагировали на сигналы $\overline{MEMR}$, $\overline{MEMW}$, $\overline{IOR}$, $\overline{IOW}$ вместо сигналов $\overline{RD}$, $\overline{WR}$ и IO/$\overline{M}$. Как и в процессоре, сигнал READY применяется для расширения циклов шины при обслуживании медленных устройств посредством введения состояний ожидания. Сигнал RESET сбрасывает регистры управления, состояния и временного хранения и флажки запросов, а также устанавливает флажки масок. Линия $\overline{EOP}$ – двунаправленная. При достижении счетчиком нуля на ней формируется отрицательный импульс, который либо служит запросом прерывания, либо подается в интерфейс устройства. Но и интерфейс может выдать на эту линию сигнал с активным низким уровнем. В любом случае низкий уровень на линии $\overline{EOP}$ приостанавливает блоковую передачу. После приостановки ее можно возобновить в соответствии с режимом работы контроллера 8237.

В составе контроллера имеются регистры управления, состояния и временного хранения, а также четыре канала. Каждый канал имеет свои регистр режима, текущий регистр адреса, базовый регистр адреса, текущий регистр счетчика байт, базовый регистр счетчика байт, флажок запроса и флажок маски. Каждый из каналов допускает работу в одном из четырех режимов; текущий режим определяется битами 7 и 6 канального регистра режима. Кратко охарактеризуем режимы работы контроллера.

Режим одиночной передачи (01). После каждой передачи контроллер освобождает шину процессору минимум на один цикл шины, но сразу же начинает проверку входов DREQ и, как только обнаруживает активный сигнал, инициирует следующую передачу.

Режим блоковой передачи (10). Активный сигнал DREQ должен сохраняться только до формирования активного сигнала DACK, после чего шина не освобождается до завершения передачи всего блока.

Режим передачи по требованию (00). Аналогичен предыдущему, но после каждой передачи проверяется сигнал DREQ. Если он пассивный, передачи приостанавливаются до тех пор, пока сигнал DREQ не будет активным. После этого передача продолжается с той точки, в которой она была приостановлена. Данный режим позволяет интерфейсу остановить передачу, если устройство не может продолжать ее.

Каскадный режим (11). Обеспечивает каскадирование контроллеров 8237 для случая, когда в подсистеме ПДП требуется более четырех каналов. При каскадировании контроллеры второго уровня подключаются к контроллерам первого уровня линиями HRQ к DREQ и HLDA к DACK. Объем данной книги не позволяет рассмотреть этот режим подробнее.

В любом режиме при достижении счетчиком нуля формируется сигнал $\overline{EOP} = 0$ и передача прекращается.

Бит 5 в регистре режима определяет инкремент (0) или декремент (1) регистра адреса после каждой передачи, т. е. идентифицирует порядок запоминания данных в памяти. Когда бит 4 содержит 1, разрешается автоинициализация. При начальной загрузке текущих регистров адреса и счетчика байт их содержимое помещается также в базовые регистры адреса и счетчика байт. Если автоинициализация разрешена, при достижении счетчиком нуля содержимое базовых регистров автоматически передается в текущие регистры. Биты 2 и 3 определяют тип производимой передачи: контроль (00), запись (01) и считывание (10). Контроль предназначен для проверки информации, относящейся к предыдущей операции ввода или вывода и фактически не связан с текущей передачей. Мы не будем подробнее останавливаться на контроле. Два младших бита при выводе в регистр режима идентифицируют выбираемый канал ПДП.

Кроме блоковых передач между памятью и устройствами ввода-вывода или внешней памятью, контроллер 8237 может управлять передачами памяти – память. При этом байты из области-источника помещаются во внутренний 8-битный регистр временного хранения, а затем передаются в область-получатель. Следовательно, для каждой передачи память – память требуются два цикла шины. Для адресации источника применяется текущий регистр адреса канала 0. Текущие регистры адреса и счетчика байт канала 1 обеспечивают адресацию получателя и счет. Инкремент или декремент адреса получателя необходимо производить как обычно, но при установке соответствующего бита в регистре управления адрес источника можно сохранять неизменным. При этом один и тот же байт данных передается во всю область-получатель.

В регистре управления передача память – память разрешается установкой бита 0 в состояние 1, при этом единичное состояние бита 1 показывает, что адрес источника необходимо сохранять неизменным. Бит 2 применяется для разрешения (0) или запрещения (1) контроллера, а бит 3 определяет тип синхронизации. Если позволяет быстродействие системы, бит 3 можно установить в состояние 1, определяя сжатую синхронизацию. В этом случае для выполнения большинства передач требуются всего два такта синхронизации (см. далее). Бит 4 определяет фиксированные или циклические (круговые) приоритеты. Обычно канал 0 имеет высший приоритет, а канал 3 – низший; но если бит 4 содержит 1, приоритеты после каждой передачи циклически изменяются. Например, если до передачи были приоритеты 2-3-0-1, то после передачи они станут 3-0-1-2. Циклическое изменение приоритетов позволяет контроллеру предотвратить монополизацию шины одним каналом. При использовании обычной синхронизации контроллер допускает программное определение продолжительности сигналов $\overline{IOW}$ и $\overline{MEMW}$. Состояние 1 бита 5 показывает, что эти сигналы расширяются на два такта синхронизации. Программа может также определить уровень активности сигналов DREQ и DACK установкой или сбросом бита 6 (DREQ) и бита 7 (DACK). Единичное состояние бита 6 показывает,

что DREQ имеет активный низкий уровень, а бита 7 – что DACK имеет активный высокий уровень. Установка состояний этих бит зависит от характеристик шины.

В регистре состояния младшие 4 бита показывают состояния счетчиков четырех каналов, а старшие 4 бита фиксируют наличие или отсутствие запросов ПДП. Для младших 4 бит состояние 1 бита *n* показывает, что в канале *n* счетчик достиг 0; для старших 4 бит состоянне 1 бита *n* + 4 сигнализирует о наличии запроса в канале *n*.

В каждом канале имеются также флажок запроса и флажок маски. Запрос ПДП можно запрограммировать так, как будто подается сигнал DREQ. Установка в канале флажка запроса имеет те же последствия, что и активный сигнал DREQ; флажок сбрасывается активным сигналом $\overline{EOP}$. При установке в состояние 1 флажок маски запрещает канал, поэтому запросы ПДП (аппаратные или программные) не распознаются. Если канал не запрограммирован на автоинициализацию, флажок маски автоматически устанавливается активным сигналом $\overline{EOP}$. Флажки запроса и маски программируются с помощью приказов, в которых бит 2 определяет состояние флажка, а биты 1 и 0 задают номер канала. Остальные биты приказа не используются. Предусмотрен также приказ воздействия сразу на все 4 флажка масок. В этом приказе состояние бита *n* переводит в аналогичное состояние флажок маски канала *n*.

Кроме приказов, определяющих состояния флажков, имеются приказы главного сброса и сброса триггера первый/последний. Приказ главного сброса имеет те же последствия, что и сигнал RESET. Триггер первый/последний предназначен для загрузки 16-битных регистров адреса и счетчиков. Так как контроллер 8237 может одновременно воспринимать только по одному байту, загрузка каждого регистра производится двумя операциями вывода. Если первоначально триггер первый/последний находится в состоянии 0, первый выводимый в эти регистры байт загружается в младший байт, а триггер устанавливается в состояние 1. При выводе второго байта единичное состояние триггера направляет его в старший байт. После этого триггер сбрасывается в состояние 0. Приказ сброса триггера первый/последний предназначен для инициализации триггера до производства операций вывода. Ни один из двух рассмотренных приказов не связан с передачей данных по шине данных. Они автоматически выполняются контроллером, когда осуществляется запись по соответствующему адресу.

Адресация разнообразных регистров и приказов осуществляется с помощью линий $\overline{CS}$, $\overline{IOR}$, $\overline{IOW}$ и A3-A0. Сигнал $\overline{CS} = 0$ определяет обращение к контроллеру. Сигнал A3 = 0 при адресации регистров адреса или счетчика и A3 = 1 при обращении к регистрам управления или состояния, а также при выдаче приказа. При адресации регистров адреса или счетчика линии A2 и A1 определяют номер канала; A0 = 0 задает текущий регистр адреса и A0 = 1 – текущий счетчик. В операции считывания $\overline{IOR} = 0$ и $\overline{IOW} = 1$, а в операции записи $\overline{IOR} = 1$ и $\overline{IOW} = 0$. Запись в текущий регистр адреса или счетчик одновременно загружает в соответствующий базовый регистр. Суммарные сведения по адресации сведены в табл. 9.5.

Таблица 9.5

Адресация в контроллере ПДП

A3	A2	A1	A0	$\overline{IOR}$	$\overline{IOW}$	Передача или приказ
1	0	0	0	0	1	Считывание из регистра состояния
1	0	0	0	1	0	Запись в регистр управления
1	0	0	1	1	0	Запись во флажок запроса
1	0	1	0	1	0	Запись во флажок маски

Окончание табл. 9.5

A3	A2	A1	A0	$\overline{IOR}$	$\overline{IOW}$	Передача или приказ
1	0	1	1	1	0	Запись в регистр режима
1	1	0	0	1	0	Сброс триггера первый/последний
1	1	0	1	0	1	Считывание временного регистра
1	1	0	1	1	0	Главный сброс
1	1	1	0	1	0	Сброс флажков масок
1	1	1	1	1	0	Запись во все флажки масок

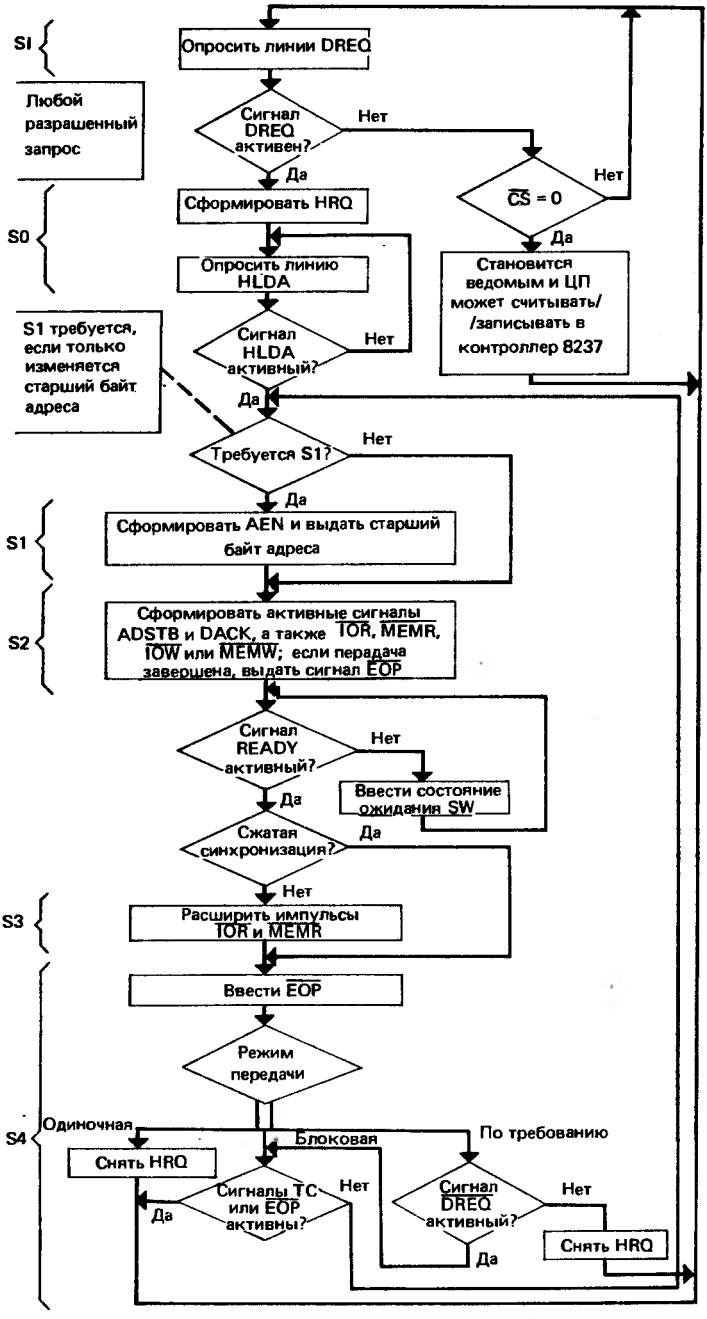
Отметим, что наименьший адрес порта, назначаемый контроллеру 8237, должен быть кратным 16. Регистр временного хранения можно считать только по завершению всей блоковой передачи память – память. Все не показанные в таблице комбинации являются недопустимыми.

Действия контроллера во времени разделяются на состояния SI, S0, S1, S2, S3, S4 и SW (см. схему на рис. 9.40). Между передачами контроллер реализует последовательность холостых состояний SI. В каждом состоянии SI производится проверка линий DREQ – не запрашивается ли передача ПДП. Если $\overline{CS} = 0$ и все сигналы DREQ пассивны, контроллер превращается в ведомого и с ним может взаимодействовать процессор. При наличии активного сигнала DREQ формируется сигнал HRQ и контроллер переходит в состояние S0. Состояния S0 повторяются до получения ответного сигнала HLDA, после чего контроллер повторяет состояния S1-S4. Состояние S1 можно пропустить, если старший байт адреса передачи изменять не нужно. Когда в передаче участвует устройство, которое не может реагировать достаточно быстро, вводятся состояния ожидания SW. В обычной синхронизации необходимы состояния S3, но, если шина и интерфейсы допускают сжатую синхронизацию, состояния S3 удаляются. В состоянии S4 проверяется режим передачи и, за исключением незаконченной передачи блока в режиме блоковой передачи и в режиме по требованию, выполняется возврат в состояние SI. Временная диаграмма передачи из интерфейса в память представлена на рис. 9.41.

9.6. КОНТРОЛЛЕРЫ НАКОПИТЕЛЕЙ НА ГИБКИХ ДИСКАХ

Для хранения больших объемов данных в основном применяют дисковые и ленточные накопители, хотя определенное распространение получает и ЦМД-память (память на цилиндрических магнитных доменах). Пока ленты и диски имеют такие преимущества по сравнению с ЦМД-памятью, как транспортабельность и емкость. Ленты имеют меньшую удельную стоимость и более долговечны, но зато диски обеспечивают намного меньшее время обращения. Выпускаются ленты и диски разнообразных размеров и форм, поэтому аппаратура считывания/записи также отличается разнообразием. Подробное рассмотрение магнитных внешних накопителей выходит за рамки настоящей книги; мы остановимся на самых популярных сейчас носителях для микросистем – *гибких дисках* (*дискетах* или *флоппи-дисках*).

Физическая конструкция гибкого диска и общий формат хранения данных представлены на рис. 9.42. Гибкий диск имеет майларовую основу с нанесенным на нее магнитным покрытием. В центре находится шпиндельное отверстие, а на некотором смещении от центра имеется минимум одно индексное отверстие. Назначение индексного отверстия – обеспечить накопителю отсчетную точку при считывании или записи данных. Гибкий диск помещен в квадратный чехол, в котором также имеются шпиндельное и ин-


SI
Любой разрашенный запрос
S0
S1 требуется, если только изменяется старший байт адреса
S1
S2
S3
S4
Опросить линии DREQ
Сигнал DREQ активен?
Нет
Да
Сформировать HRQ
Опросить линию HLDA
Сигнал HLDA активный?
Нет
Да
Требуется S1?
Нет
Да
Сформировать AEN и выдать старший байт адреса
Сформировать активные сигналы ADSTB и DACK, а также TOR, MEMR, IOW или MEMW; если передача завершена, выдать сигнал EOP
Сигнал READY активный?
Нет
Да
Ввести состояние ожидания SW
Сжатая синхронизация?
Да
Нет
Расширить импульсы TOR и MEMR
Ввести EOP
Режим передачи
Одиночная
Блоковая
По требованию
Снять HRQ
Сигналы TC или EOP активны?
Нет
Да
Сигнал DREQ активный?
Нет
Да
Снять HRQ
CS = 0
Нет
Да
Становится ведомым и ЦП может считывать/ /записывать в контроллер 8237

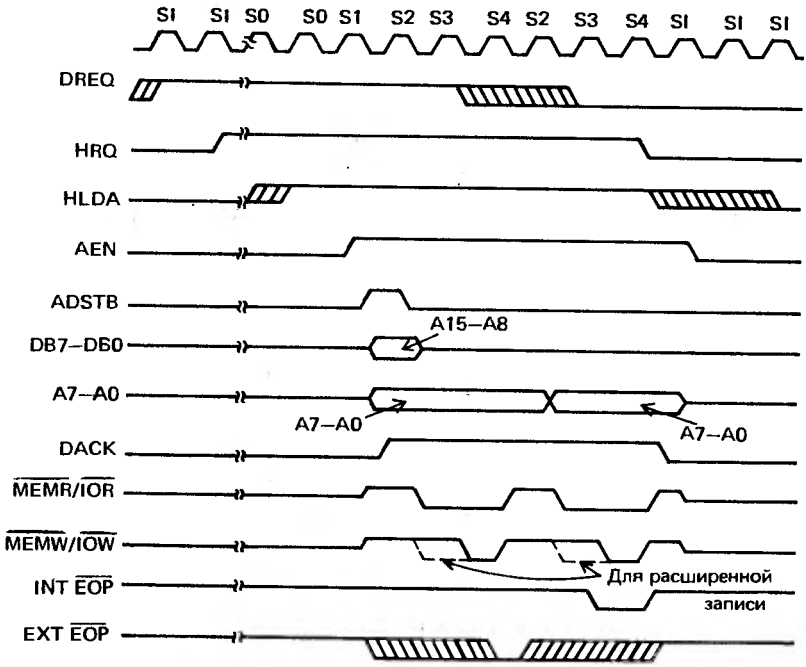


Рис. 9.41. Типичная временная диаграмма работы контроллера 8237

дексное отверстия. Кроме того, в чехле есть прорезь для контактирования головки считывания/записи с поверхностью диска и вырез защиты от записи.

Данные хранятся в последовательной форме на концентрических окружностях, называемых *дорожками*, и группируются по дугам, называемым *секторами*. В некоторых накопителях имеется одна головка считывания/записи и данные хранятся только на одной поверхности диска, а в других есть две головки и используются обе поверхности диска. Во втором случае пары дорожек, находящихся на одном и том же расстоянии от центра диска, называются *цилиндрами*.

Стандартные гибкие диски имеют размер стороны чехла 203 и 133 мм (*минидиски*). Различают диски с *программным разделением на секторы*, когда диск имеет одно индексное отверстие, и с *аппаратным*, когда число отверстий равно числу секторов. Каждый сектор должеи начинаться со служебной информации, которая отмечает начало сектора, и топько первый сектор на дорожке отмечается индексным отверстием. Во втором случае все секторы отмечаются индексными отверстиями.

Дорожки (и цилиндры) пронумерованы, причем внешняя дорожка имеет номер 0. Секторы также нумеруются, и в диске с программным разделением на секторы первый сектор после индексного отверстия имеет номер 1. Когда диск помещается в накопитель, шпиндель проходит через отверстие, диск сцепляется со шпинделем и вращается в чехле со скоростью 360 об/мин. Подвижную головку считывания/записи можно пози-

← Рис. 9.40. Схема действий контроллера 8237 по состояниям

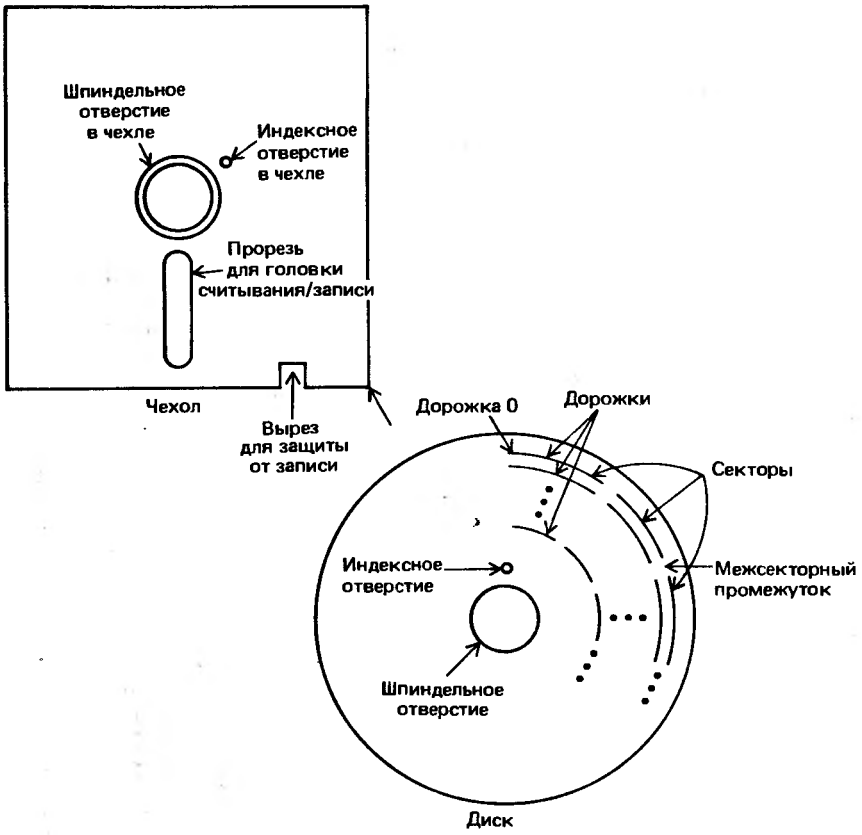


Рис. 9.42. Конструкция гибкого диска

ционировать на любую дорожку, после чего вращательное движение заставляет все секторы выбранной дорожки проходить под головкой, обеспечивая тем самым доступ к отдельным секторам. Для выполнения записи или считывания головка опускается (*загружается*) на поверхность диска через прорезь в чехле и позиционируется на нужную дорожку. В диске с программным разделением на секторы накопитель ожидает появления сигнала от индексного отверстия, а затем начинает считывать служебную информацию о формате сектора. После нахождения нужного сектора начинается собственно операция считывания или записи. В случае диска с аппаратным разделением на секторы требуемый сектор находится с помощью сигналов от индексных отверстий.

Необходимое для доступа к сектору время складывается из следующих составляющих:

Время загрузки. Для приведения головки в контакт с поверхностью диска.

Время позиционирования. Для позиционирования головки на дорожку.

Вращательное запаздывание. Для поворота диска до достижения требуемого сектора.

Средние значения этих времен равны 16, 225 и 80 мс соответственно. Когда сектор найден, средняя скорость передачи данных составляет (байт/с):

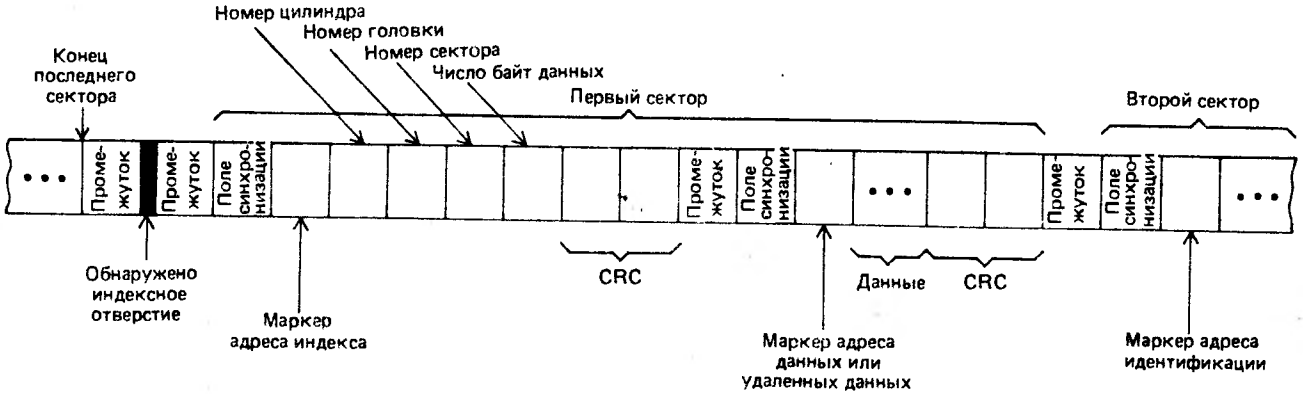


Рис. 9.43. Форматы дорожки и сектора гибкого диска с программным разделением на секторы

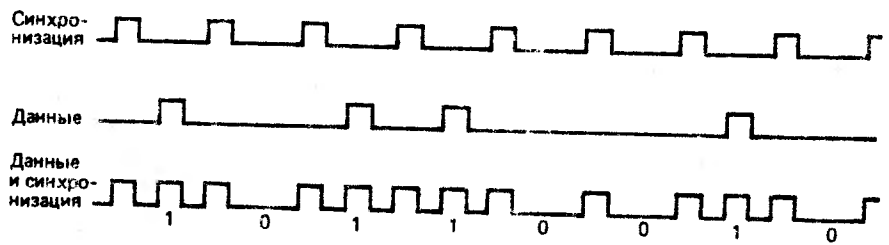


Рис. 9.44. Двоичное представление хранимых данных

Число байт в секторе X число секторов на дорожке X скорость в об/с (здесь учитывается время прохождения промежутков между данными).

Рассмотрим более подробно конкретный тип гибкого диска – 200-мм диск с программным разделением на секторы, совместимый с накопителем 3740 фирмы IBM. Диск имеет 77 дорожек с 15 (одинарная плотность) или 26 (двойная плотность) секторами, хотя в наших примерах допустимое число секторов варьируется от 8 до 26. Обычно используются 75 дорожек, т. е. допускаются две дефектные дорожки. Рабочие дорожки нумеруются от 0 до 74.

Форматы дорожки и сектора рассматриваемых дисков показаны на рис. 9.43. Между смежными секторами находятся промежутки, и сигнал от индексного отверстия должен появляться в промежутке между последним и первым сектором дорожки. Каждый сектор содержит поле идентификации и поле данных, оба из которых начинаются с поля синхронизации. Между двумя полями имеется промежуток, необходимый для коммутации головки на выполнение заданной операции. Поле идентификации начинается с маркера адреса, после которого следуют номер цилиндра (или дорожки), номер головки, номер сектора, число байт данных в секторе и два байта контроля ошибок.

Маркер адреса в поле идентификации первого сектора обычно отличается от маркеров в других секторах, а маркеры адреса в полях идентификации отличаются от маркеров в полях данных. Кроме того, маркеры адреса в полях данных различаются в зависимости от того, содержит поле полезную информацию или занято байтами-заполнителями.

Имеются два главных метода генерирования байт контроля ошибок. Один из них заключается в получении контрольной суммы посредством суммирования всех байт в поле, находящемся перед контрольной суммой (с игнорированием переносов). Во втором методе биты поля считаются коэффициентами одного двоичного полинома, а байты контроля ошибок являются остатком от деления этого полинома на фиксированный поли-

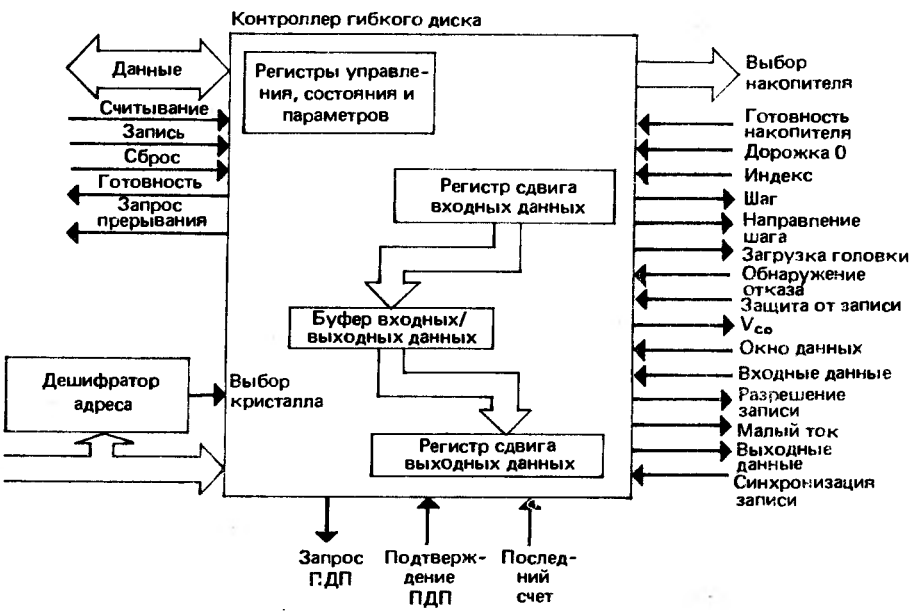


Рис. 9.45. Организация контроллера накопителя на гибком диске

ном степени 16 (метод называется *циклическим избыточным контролем* – CRC). Далее предполагается применение метода CRC.

На рис. 9.44 показано двоичное представление последовательно хранимых данных. Информация группируется в элементы, каждый из которых разделен на четыре интервала; в первом интервале находится импульс синхронизации. Третий интервал предназначен для бита данных и будет содержать импульс, если бит данных равен 1. Типичное значение временного отрезка элемента составляет 4 мкс (при скорости 360 об/мин).

Рис. 9.45 иллюстрирует организацию типичного контроллера накопителя на гибком диске, а рис. 9.46 показывает место контроллера в подсистеме внешней памяти на гибком диске. Из-за последовательного хранения информации входы и выходы накопителя представляют собой двоичные потоки. Поэтому входные байты принимаются в регистр сдвига и затем передаются в буферный регистр данных, а выходные данные загружаются в регистр сдвига, из которого они выдвигаются в последовательной форме. Выходные схемы должны объединять импульсы со входа синхронизации записи и биты данных так, чтобы получался формат, представленный на рис. 9.44.

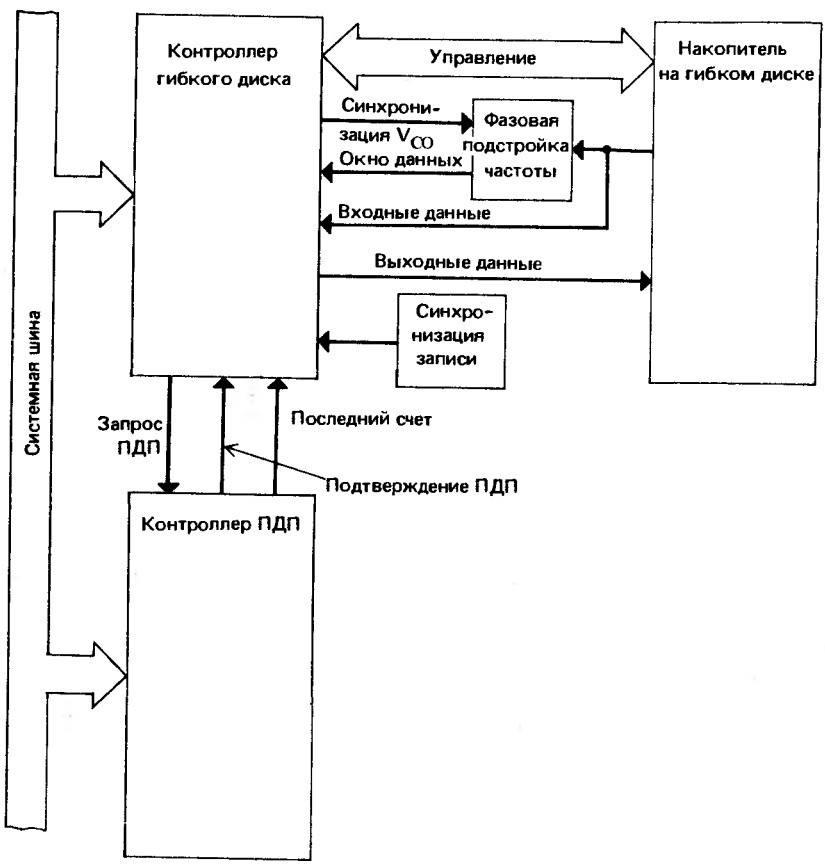


Рис. 9.46. Подсистема гибкого диска

Так как большинство контроллеров рассчитано на управление несколькими накопителями, некоторые контакты контроллера необходимо зарезервировать для линий выбора накопителя. Кроме того, должны быть такие выходы в накопитель, как линия импульса шага для прохождения по дорожкам, линия сигнала направления шага и линия сигнала загрузки головки. Необходимы также входные сигналы, показывающие нахождение головки на дорожке 0, извещающие о возникновении отказа или ошибки, фиксирующие индексное отверстие; нужен сигнал и о том, что гибкий диск защищен от записи.

При считывании с диска данные необходимо отделить от импульсов синхронизации. Обычно это осуществляется схемой фазовой подстройки частоты, которая формирует сигнал окна. В эту схему контроллер должен выдавать сигнал синхронизации V_{CO}. Для записи данных кроме линии выходных данных требуются линии разрешения записи и малого тока. Линия малого тока требуется в связи с тем, что при записи на внутренние дорожки (их номера больше 43) применяется меньший ток.

Если гибкий диск имеет 26 секторов, каждый сектор содержит 128 байт и скорость вращения составляет 360 об/мин, средняя скорость передачи данных

$$26 \times 128 \times 360/60 \approx 19968 \text{ байт/с.}$$

Средний период оказывается равным примерно 50 мкс. С учетом промежутков реальный период передачи байта составляет 32 мкс (или 4 мкс на бит). Хотя и можно осуществлять передачи с такой скоростью без контроллера ПДП, применение контроллера оказывается более эффективным. Поэтому контроллеры дисков обычно имеют сигналы запроса ПДП, подтверждения ПДП и окончания счета. Контроллер ПДП оказывается лучшим решением еще и потому, что передачи с диска всегда связаны с передачами блоков, а не отдельных байт.

Рассмотрим подробнее типичный контроллер диска 8272 фирмы Intel, организация которого показана на рис. 9.47. В нем имеются два регистра, которые может адресовать процессор: регистр состояния, который считывается при A0 = 0, и регистр входных/выходных данных, обращение к которому производится с A0 = 1. Формат регистра состояния представлен на рис. 9.48. Через регистр входных/выходных данных можно адресовать несколько других регистров управления, состояния и параметров, а также несколько флажков. Обращение к этим регистрам и флажкам или выполнение ввода-вывода данных определяется последовательностью доступа.

Выполняемые контроллером операции разделяются на фазы приказа, выполнения и результата. Во время фазы приказа байты передаются в регистры управления и флажки через регистр входных/выходных данных. Затем на фазе выполнения реализуется запрошенная операция, по завершении которой наступает фаза результата, когда процессор считывает информацию состояния. Выводы во время фазы приказа и вводы во время фазы результата выполняются однобайтными передачами, хотя любыми передачами данных во время фазы выполнения обычно управляет контроллер ПДП.

Контроллер гибкого диска имеет следующие приказы:

Считать данные. Считывает данные из поля данных на гибком диске.

Записать данные. Записывает данные в поле данных на гибком диске.

Считать удаленные данные. Считывает данные из поля, отмеченного как удаленное.

Записать удаленные данные. Записывает маркер адреса удаленных данных и помещает символы-заполнители в поле данных.

Считать дорожку. Считывает поля данных всей дорожки.

Считать ID. Считывает поле идентификации.

Форматировать дорожку. Записывает на дорожку форматирующую информацию, используя выданные программой параметры форматирования.

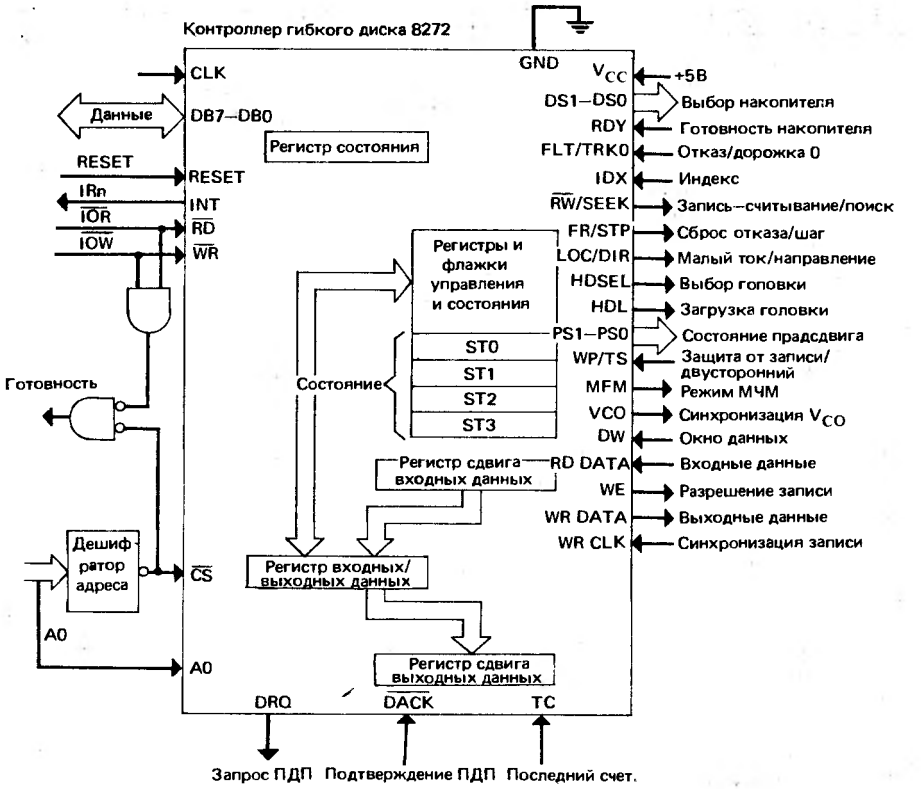


Рис. 9.47. Схема контроллера гибкого диска 8272

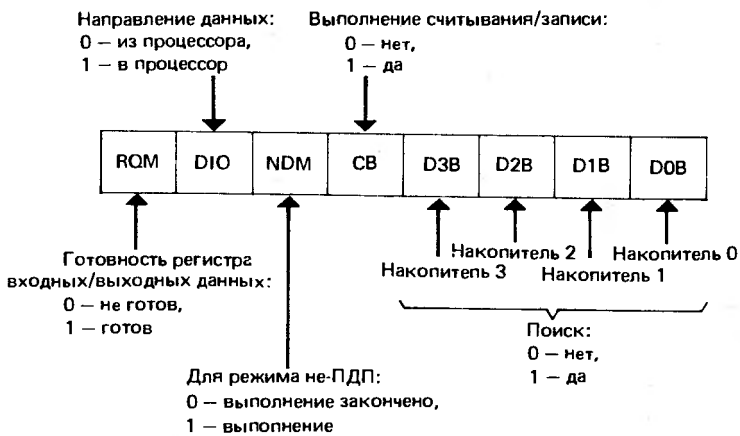


Рис. 9.48. Формат регистра состояния контроллера 8272

Сканировать до равно, сканировать до меньше или равно, сканировать до больше или равно. Сканирует данные на указанное условие и формирует запрос прерывания, когда условие удовлетворяется.

Инициализировать. Вызывает перемещение головки на дорожку 0.

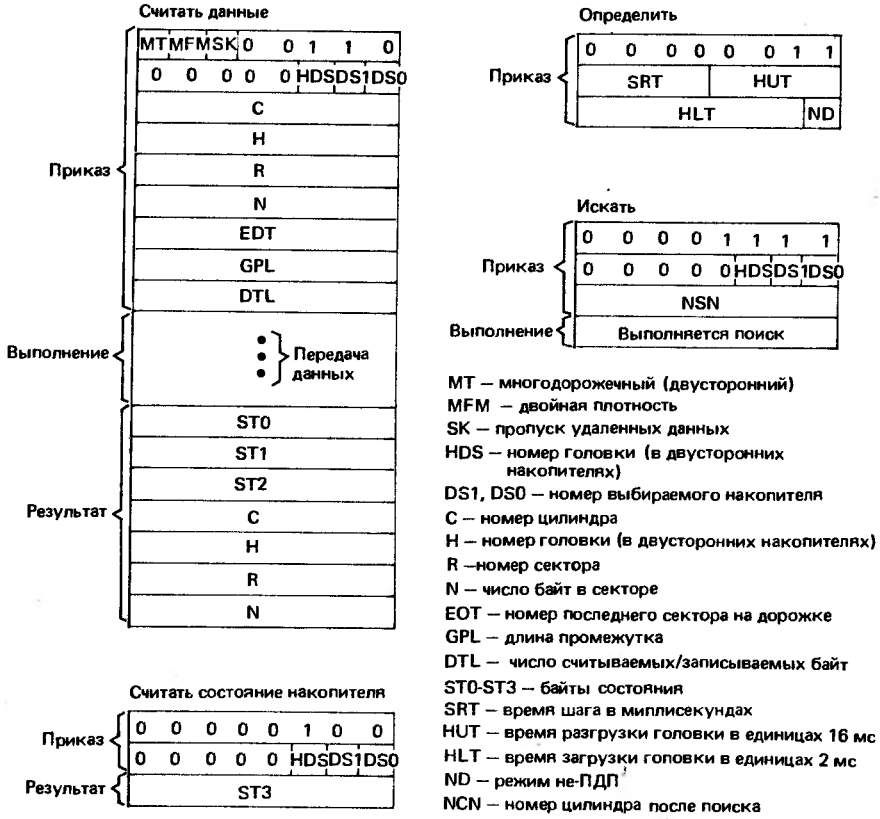
Считать состояние прерывания. Считывает информацию состояния из ST0 после прерываний, вызванных изменением линии готовности и операцией поиска.

Определить. Устанавливает скорость шагов головки, время разгрузки (подъема) головки, время загрузки головки и режим ПДП.

Считать состояние накопителя. Вводит состояние накопителя из ST3.

Искать. Позиционирует головку на указанную дорожку.

В качестве примера на рис. 9.49 приведена последовательность, необходимая для выполнения четырех приказов. Приказ считывания данных включает в себя все три фазы, приказ поиска – только фазы приказа и выполнения, приказ считывания состояния накопителя – фазы приказа и результата, а приказ определения – только фазу приказа. Во всех случаях приказы должны выполняться целиком (включая и фазу результата, если она применима), иначе они считаются незаконченными. При получении недействительного приказа контроллер возвращает байт состояния ST0 в ответ на следующий ввод из регистра входных/выходных данных.



MT – многодорожечный (двусторонний)
MFM – двойная плотность
SK – пропуск удаленных данных
HDS – номер головки (в двусторонних накопителях)
DS1, DS0 – номер выбираемого накопителя
C – номер цилиндра
H – номер головки (в двусторонних накопителях)
R –номер сектора
N – число байт в секторе
EOT – номер последнего сектора на дорожке
GPL – длина промежутка
DTL – число считываемых/записываемых байт
ST0-ST3 – байты состояния
SRT – время шага в миллисекундах
HUT – время разгрузки головки в единицах 16 мс
HLT – время загрузки головки в единицах 2 мс
ND – режим не-ПДП
NCN – номер цилиндра после поиска

Рис. 9.49. Типичные приказы контроллера 8272

Несколько приказов требуют, чтобы на их фазах результатов считывались некоторые из байт состояния ST0, ST1, ST2 и ST3. Биты в этих регистрах показывают ошибки CRC, ошибки перегрузкн, выбранный накопитель, конец поиска, является ли гибкий диск двусторонним или нет и т. д.

В качестве примера рассмотрим контроллер 8272, четный адрес которого равен 002A. Его можно инициализировать на скорость шагов 6 мс на дорожку, время разгрузки головки 48 мс, время загрузки головки 16 мс и режим ПДП следующей последовательностью приказов:

```
%CHECK  (2AH,80H)
            MOV    AL,03H
            OUT    2BH,AL
%CHECK  (2AH,80H)
            MOV    AL,63H
            OUT    2BH,AL
%CHECK  (2AH,80H)
            MOV    AL,10H
            OUT    2BH,AL
```

Макрокоманда %CHECK имеет такое определение:

```
%*DEFINE  (CHECK  (PORT, STAT)) LOCAL  AGAIN
(%AGAIN:   IN       AL,%PORT
           AND      AL,0C0H
           XOR      AL,%STAT
)          JNE      %AGAIN
```

Причина необходимости макрокоманды %CHECK перед каждым выводом заключается в том, что перед записью в контроллер 8272 байта каждого приказа старшие два бита регистра состояния должны содержать 10. На фазе результата перед считыванием из регистра данных каждого байта эти два бита должны содержать 11.

Последовательность приказа поиска

```
%CHECK  (2AH,80H)
            MOV    AL,0FH
            OUT    2BH,AL
%CHECK  (2AH,80H)
            MOV    AL,02H
            OUT    2BH,AL
%CHECK  (2AH,80H)
            MOV    AL,30
            OUT    2BH,AL
```

производит в накопителе 1 перемещение головки на цилиндр 30 и выбор головки 0.

На рис. 9.50 определены две макрокоманды для выполнения приказа считывания данных. Предполагается, что перед вызовом макрокоманды READ_COM находятся команды

```
IDLE: IN     AL,2AH
      TEST   AL,12H
      JNZ    IDLE
```

Они проверяют доступность требуемого накопителя (которым в этом примере служит накопитель 1). Перед вызовом макрокоманды READ_COM необходимо инициализировать контроллер ПДП, чтобы передачи ПДП начинались сразу по окончании операции

```
%*DEFINE (READ_COM (ADDR,HDSDS,C,H,R,N.DTL))
(                PUSH    AX              ;ЗАПОМНИТЬ РЕГИСТРЫ
    %CHECK (2AH,80H)
                 MOV     AL,46H          ;ВЫВЕСТИ КОД ПРИКАЗА
                 OUT     %ADDR,AL
    %CHECK (2AH,80H)
                 MOV     AL,%HDSDS       ;ВЫВЕСТИ НОМЕРА ГОЛОВКИ
                 OUT     %ADDR,AL        ;И НАКОПИТЕЛЯ
    %CHECK (2AH,80H)
                 MOV     AL,%C           ;ВЫВЕСТИ НОМЕР КАНАЛА,
                 OUT     %ADDR,AL
    %CHECK (2AH,80H)
                 MOV     AL,%H           ;НОМЕР ГОЛОВКИ,
                 OUT     %ADDR,AL
    %CHECK (2AH,80H)
                 MOV     AL,%R           ;НОМЕР СЕКТОРА,
                 OUT     %ADDR,AL
    %CHECK (2AH,80H)
                 MOV     AL,%N           ;ЧИСЛО БАЙТ ДАННЫХ,
                 OUT     %ADDR,AL
    %CHECK (2AH,80H)
                 MOV     AL,26           ;ЧИСЛО СЕКТОРОВ,
                 OUT     %ADDR,AL
    %CHECK (2AH,80H)
                 MOV     AL,17           ;ДЛИНУ ПРОМЕЖУТКА
                 OUT     %ADDR,AL
    %CHECK (2AH,80H)
                 MOV     AL,%DTL         ;И ДЛИНУ ДАННЫХ
                 OUT     %ADDR,AL
                 POP     AX
)
%*DEFINE (READ_STAT (ADDR,RDSTAT)) LOCAL LP
(                PUSH    AX              ;ЗАПОМНИТЬ РЕГИСТРЫ
                 PUSH    CX
                 PUSH    SI
                 MOV     SI,0            ;ВВЕСТИ СОСТОЯНИЕ
                 MOV     CX,7            ;В МАССИВ RDSTAT
    %LP:         NOP
    %CHECK (2AH,0C0H)
                 IN      AL,%ADDR
                 MOV     %RDSTAT[SI],AL
                 INC     SI
                 LOOP    %LP
                 POP     SI              ;ВОССТАНОВИТЬ РЕГИСТРЫ
                 POP     CX
    )            POP     AX
```

Рис. 9.50. Макрокоманды для выполнения приказов считывания данных

считывания. Предполагается, что вызов макрокоманды READ_STAT делается только после фазы выполнения и при условии завершения считывания (например, после возникновения прерывания по завершению считывания).

9.7. ИНТЕРФЕЙС МАКСИМАЛЬНОГО РЕЖИМА И 16-БИТНОЙ ШИНЫ

Как говорилось во введении к данной главе, все примеры в § 9.1 – 9.6 относились к процессору 8088 в минимальном режиме. Для перехода к системе с максимальным режимом потребуются два изменения. Первое связано с тем, что в системе необходим контроллер шины, подключаемый в соответствии с рис. 8.9 и 8.10. В системе, имеющей контроллер ПДП 8237, контроллер шины заменяет схему кодирования сигналов $\overline{RD}$, $\overline{WR}$ и IO/$\overline{M}$, показаниую на рис. 9.38 и более подробно на рис. 9.39. В любом случае с появлением контроллера шины входы интерфейсов $\overline{RD}$ и $\overline{WR}$ подключаются к выходам $\overline{IORC}$ и $\overline{IOWC}$ контроллера, а линии IO/$\overline{M}$ на входах дешифратора адреса больше не нужны.

Второе важное изменение касается сигналов HRQ и HLDA, реализующих запросы и разрешения шины. Контроллер ПДП выдает потенциальный сигнал запроса HRQ до тех пор, пока он не готов освободить шину; после этого сигнал HRQ снимается. Процессор также выдает потенциальный сигнал HLDA. Такие действия совместимы с работой микропроцессоров 8086/8088 в минимальном режиме. Однако в максимальном режиме

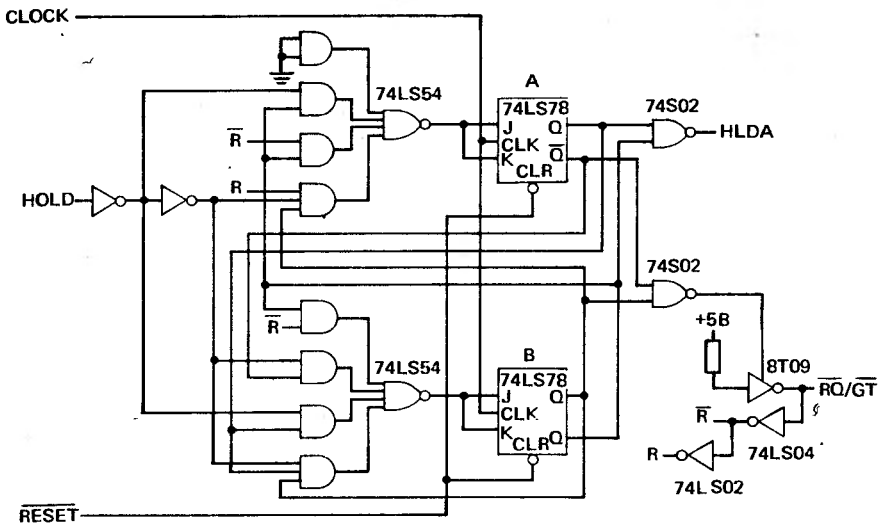


Рис. 9.51. Преобразования запроса и разрешения шины при подключении контроллера ПДП 8237 к микропроцессорам 8086/8088 в максимальном режиме

процессоры воспринимают запросы и выдают разрешения по одной линии $\overline{RQ/GT}$ и ожидают сигнала запроса шины в виде одиночного импульса. Запрос подтверждается выходным импульсом, а по завершении операций ПДП контроллер должен послать в процессор второй импульс. Схема преобразования двух потенциальных сигналов контроллера ПДП в импульсы на одной линии, предназначенная для процессора в максимальном режиме, представлена на рис. 9.51.

Трудности подключения интерфейсов 8-битных устройств к 16-битной шине процессора 8086 связаны с необходимостью передач байт с четными адресами по младшей половине шины данных, а байт с нечетными адресами – по старшей половине. Если интерфейс взаимодействует только с процессором (т. е. не использует ПДП), проблема решается довольно просто. Вместо подключения к входам An – A0 интерфейса линий адреса для выбора внутренних регистров интерфейса, например An – A0, следует подключить к этим входам линии A(n + 1) – A1 в соответствии с рис. 9.52. Это означает, что интерфейсу будут назначены только четные адреса в пространстве ввода-вывода, начиная с адреса, кратного 2^{n+1}, а нечетные адреса из этого диапазона не используются. Если, например, входы A1 и A0 микросхемы 8255A подключены к линиям адреса A2 и A1 и начальный адрес портов 8255A равен 08F8, все передачи в (из) 8255A осуществляются по младшему байту шины. У портов A, B и C будут адреса 08F8, 08FA и 08FC соответственно, а у регистра управления будет адрес 08FE. Аналогично интерфейсу можно назначить последовательные нечетные адреса, если он подключен к старшему байту шины.

Для использования последовательных адресов и обеих половин 16-битной шины данных следует воспользоваться подключениями к шине, показанными на рис. 9.53. Для доступа к регистру микропроцессор 8086 использует сигналы $\overline{BHE}$ и A0 в соответствии с табл. 9.6.

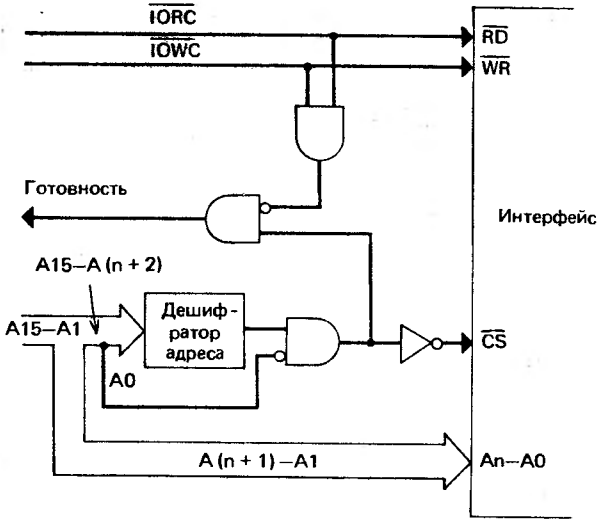


Рис. 9.52. Обращения к портам ввода-вывода, имеющим только четные адреса

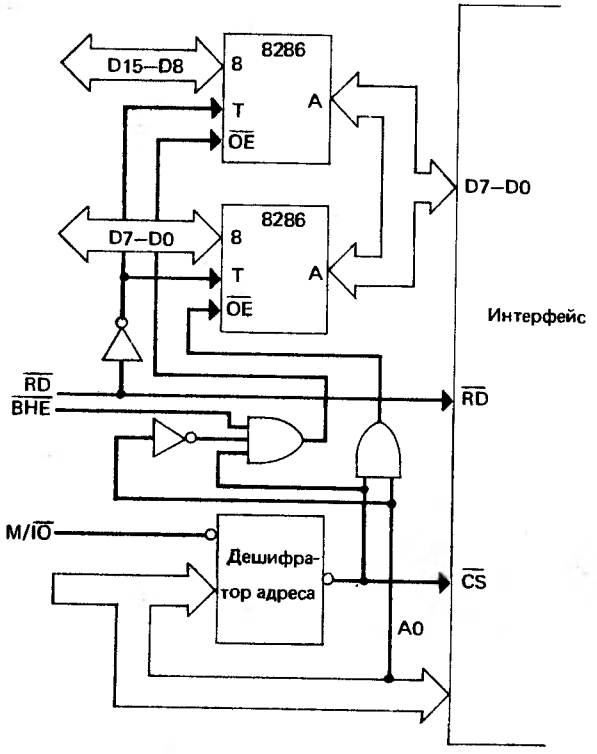


Таблица 9.6

Доступ к регистру с помощью сигналов $\overline{BHE}$ и A0

$\overline{BHE}$	A0	Передача
0	0	Не применяется
0	1	Байт с нечетным адресом на старшую половину шины
1	0	Байт с четным адресом на младшую половину шины
1	1	Невозможна

Пользуясь этими сигналами и двумя приемопередатчиками 8286, можно передать данные между интерфейсом и линиями шины данных D7-D0, когда $\overline{BHE}$ = 1 и A0 = 0, и линиями шины D15-D8, когда $\overline{BHE}$ = 0 и A0 = 1. Сигнал $\overline{RD}$ применяется для опреде-

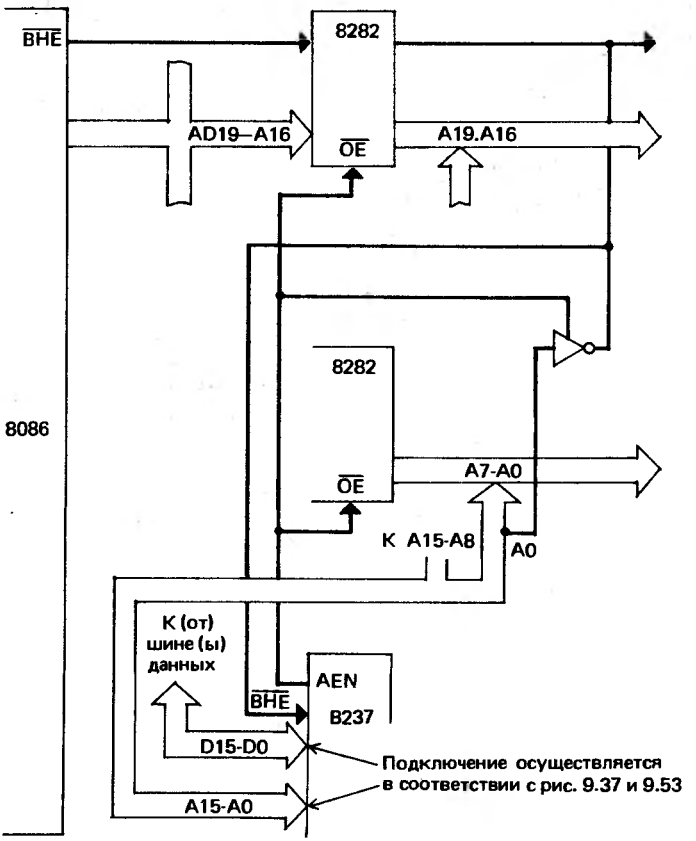


Рис. 9.54. Применение контроллера ПДП 8237 в системе с 16-битной шиной

← Рис. 9.53. Подключение 8-битного интерфейса к 16-битной шине

ления направления передачи данных. Схемы, относящиеся к сигналу готовности, не показаны.

В системе с контроллером ПДП 8237 логику управления шиной, приведенную на рис. 9.38, необходимо изменить в соответствии с рис. 9.54. Выходной сигнал $\overline{BHE}$ процессора 8086 подается в тот же регистр-защелку 8282, который применяется для линий A19-A16. Линия A0 подключается к линии $\overline{BHE}$ через тристабильный инвертор, управляемый сигналом AEN контроллера ПДП. Когда сигнал AEN активен и A0 = 0, сигнал $\overline{BHE}$ имеет высокий уровень, показывая передачу по младшему байту шины. Если AEN активен и A0 = 1, сигнал $\overline{BHE}$ имеет низкий уровень и передача осуществляется по старшему байту. Кроме того, интерфейс и контроллер ПДП следует подключить к шине через дополнительные схемы, аналогичные приведенным на рис. 9.53.

Ранее мы рассмотрели взаимодействие между 8-битным интерфейсом и 16-битной шиной данных; сделаем несколько замечаний относительно 16-битного интерфейса. Такой интерфейс может передавать на (с) шину (ы) целые слова и удваивает эффективность циклов шины. Схема 16-битного интерфейса на основе двух микросхем 8255A показана на рис. 9.55. Линии A2 и A1 шины адреса подключены на входы A1 и A0 обеих

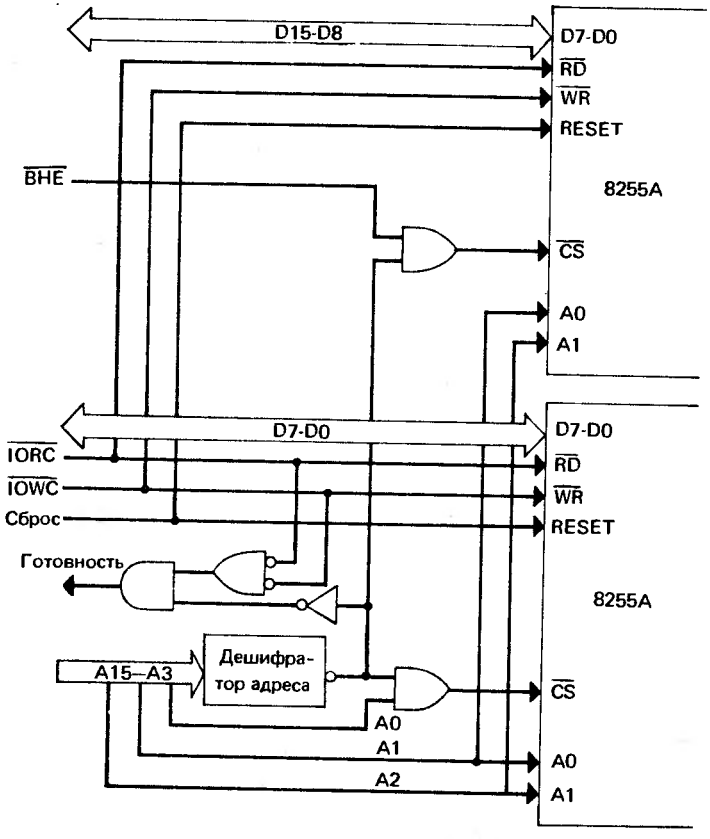


Рис. 9.55. Пример параллельного 16-битного интерфейса ввода-вывода

микросхем; таким образом из пар портов A, B, C и регистров управления/состояния получаются 16-битные порты. Нижняя микросхема 8255A занимает четыре смежных четных адреса, а верхняя – четыре смежных нечетных адреса. Если биты A15-A13 соответствуют адресу, встроенному в дешифратор адреса, он выдает сигнал выбора кристалла с нулевым значением. Если для нижней микросхемы оба сигнала выбора кристалла и A0 равны 0, то на вход $\overline{CS}$ подается 0. В верхней микросхеме для образования $\overline{CS} = 0$ необходимо, чтобы нулевые значения имели сигналы выбора кристалла и $\overline{BHE}$ (этим допускается раздельная адресация микросхем). Линии управления считывания, записи и сброса подаются на входы $\overline{RD}$, $\overline{WR}$ и RESET обеих микросхем, а сигнал готовности возвращается, если активен любой из сигналов $\overline{CS}$.

Интерфейс можно спроектировать и так, что регистры устройства ввода-вывода будут считаться ячейками памяти. Такой метод называется *вводом-выводом, отображенным на память*, и требует,чтобы интерфейс реагировал на сигналы считывания из памяти и записи в память. Главное достоинство такого ввода-вывода заключается в том, что к регистрам можно обращаться любой командой и любым режимом адресации, которые обращаются к операндам в памяти, а это обеспечивает гибкость программирования. Но если регистры управления и состояния разделяют один и тот же адрес (как это имеет место в некоторых микросхемах фирмы Intel), не допускается пользоваться такими командами, как

```
OR  CONTROL,00000001B
```

Эта команда осуществляет ввод из регистра состояния, а выводит в регистр управления. Так как один и тот же адрес невозможно назначить ячейке памяти и регистру, ввод-вывод, отображенный на память, сокращает число доступных ячеек памяти.

Упражнения

1. Приведите временную диаграмму передачи по асинхронной последовательной линии связи буквы C, если применяется 7-битный код ASCII, четный паритет и два стоповых бита.

2. Предположим, что промежутков между символами нет, символы содержат 7 информационных бит, имеются бит паритета и два стоповых бита, а скорость передачи составляет 600 бит/с. Сколько символов передается в секунду? Сравните ее со скоростью синхронной линии с той же двоичной скоростью 600 бит/с, при отсутствии холостых символов, если сообщение состоит из двух символов синхронизации и последующих 200 7-битных информационных символов.

3. Пользуясь материалом раздела 9.1.3, определите максимальное расстояние, на которое можно передать сигнал со скоростью 4800 бод без связного оборудования.

4. Пусть в эквивалентной схеме на рис. 9.8 $R_0 = 500$ Ом, $C_0 = 2$ мкФ, C_L определяется для линии длиной 100 м с погонной емкостью 20 пФ/м, $R_L = 4000$ Ом и $E_L = 0$ В. Чему будет равно V_I, когда $V_0 = 7$ В? Определите максимум производной V_I, когда V_0 мгновенно изменяется от -7 до $+7$ В. Сколько времени длится переход V_I от -3 до $+3$ В, если V_0 переключается от -7 к $+7$ В? Будут ли удовлетворяться электрические спецификации интерфейса RS-232-C, если линия работает со скоростью 1200 бод?

5. Каков будет формат передаваемого символа, если регистр режима микросхемы 8251A содержит 01111011? Какие частоты должны действовать на входах RxC и TxC, чтобы приемник имел скорость 300 бод, а передатчик – 1200 бод?

6. Определите форматы символа и сообщения последовательной передачи, если в регистре режима микросхемы 8251A содержится код 00010100?

7. Пусть после сброса выводится следующая последовательность с A0 = 1:

00010100
00010110
00010110
10110011
.
.
. } Вывод с A0 = 0
01000000

Какие действия предпринимаются в микросхеме 8251A на каждый выводимый байт? (См. рис. 9.14).

8. Напишите фрагмент инициализации микросхемы 8251A на асинхронную передачу со следующим форматом: 7 информационных бит, 1 стоповый бит, бит паритета отсутствует. Множитель скорости передачи равен 64. Четный адрес 8251A равен 008A. Включите во фрагмент приказ, который выдает сигналы RTS и DTR и разрешает передатчик.

9. Расширьте фрагмент из упр. 8, чтобы вывести с помощью программного ввода-вывода сообщение

ВОЗНИКЛА ОШИБКА ПЕРЕГРУЗКИ

а затем сбросить бит ошибки перегрузки.

10. Напишите процедуру прерывания, которая выводит 20 байт, начиная с MESSAGE, в микросхему 8251A, четный адрес которой равен 25C0. При каждом вызове процедура выводит один байт, а после вывода 20 байт она помещает 1 в FLAG. Для хранения счетчика байт предназначена ячейка CNT.

11. Считая, что начальный адрес микросхемы 8255A равен 0500, напишите фрагмент, который производит следующие действия:

а) переводит обе группы A и B в режим 0, причем порты A и C являются входными, а порт B – выходным;

б) переводит группу A в режим 2, а группу B в режим 1, причем порт B является выходным;

в) переводит группу A в режим 1, причем порт A служит для ввода, а PC6 и PC7 – для вывода, и группу B в режим 1 (порт B является входным).

12. Предположим, что микросхема 8255A с начальным адресом 0030 инициализирована так, что группа A работает в режиме 2, группа B – в режиме 0, причем PB7-PB0 являются входами, а PC2-PC0 – выходами. Напишите программу, которая выводит 100 байт из массива с начальным адресом OUT_ARRAY в устройство, подключенное к порту A. Перед выводом каждого байта программа должна ожидать активных уровней на входах PB1, PB2 и PB3, а после вывода – установить PC1 в 1.

13. Пользуясь микросхемами 8255A, 8282, 8286 и другими, спроектируйте устройство для установки и считывания группы из восьми реле. Считайте, что драйверов 8286 достаточно для управления выходными схемами, которые подают ток в обмотки реле. Определите режимы микросхемы 8255A и все необходимые сигналы управления. Сами реле и их вспомогательные схемы считайте заданными.

14. Используя порт B в режиме 1, модифицируйте схему на рис. 9.23 так, чтобы она работала с 12-битным аналого-цифровым преобразователем. Напишите программу для ввода отсчета и запоминания его в слове с адресом SAMPLE.

15. Опишите, как микросхему 8254 можно использовать для подсчета входных импульсов в течение временного интервала, управляемого вторым входом. По окончании всего счета выдается запрос прерывания по линии IR2, а после каждых 20 подсчитанных импульсов формируется запрос прерывания по линии IR3. Напишите программу инициализации 8254.

16. Предположим, что микросхема 8254 подключена к генератору синхронизации с частотой 1 кГц и должна поддерживать время дня в BCD-формате с точностью до секунд. Она запускается сразу после загрузки в байты HOURS (часы), MINUTES (минуты), SECONDS (секунды) и AMPM (после полудня) текущего времени. Напишите программу, которая инициализирует 8254, и процедуру прерывания, которая модифицирует время по истечении каждой секунды.

17. Считайте, что на рис. 9.30 порты микросхемы 8255A подключены к клавиатуре. Составьте схему программы сканирования клавиатуры. (Срабатывания нескольких клавиш не учитывайте.)

18. Предположим, что микросхема 8255A на рис. 9.32 управляет многоразрядным индикатором в мультиплексном режиме.

а) Покажите соединения между 8255A и индикатором.

б) Постройте схему программы индикации.

19. В соответствии с условиями упр. 18 приведите схему, показывающую, как можно обойтись без операций регенерации, пользуясь внешними регистрами-защелками данных.

20. Предположим, что необходимо контролировать восемь устройств, каждое из которых имеет 8-битный регистр состояния. Составьте логическую схему, показывающую применение для этой задачи микросхемы 8279.

21. Составьте программу передачи память – память 1000 байт из области SOURCE в область DST с помощью контроллера ПДП 8237. Адреса портов контроллера равны 0600-060F.

22. Поясните, как происходит управление мнкросхемами 8286 на рис. 9.53 во время:

а) вывода в регистр управления;

б) ввода из регистра состояния;

в) передачи из интерфейса ввода-вывода в память;

г) передачи память – память.

Считайте, что интерфейсом является контроллер ПДП 8237.

23. Напишите программу, которая заставляет контроллер ПДП 8237 выполнить в режиме по требованию блоковую передачу 500 байт в массив с начальным адресом BUFFER. Байты запоминаются с наибольшего адреса (т. е. BUFFER + 499). Адреса портов контроллера равны 1020-102F.

24. Приведите пары команд, которые выполняют следующие действия:

а) маскируют каналы 0 и 3 контроллера ПДП 8237;

б) вызывают программио-инициируемую передачу ПДП в канале 2 контроллера 8237 в соответствии с текущим содержимым его регистров управления, адреса и счетчика байт.

25. Постройте временную диаграмму работы контроллера ПДП 8237 при блоковой передаче 4 байт из устройства ввода-вывода в память.

26. Пусть время загрузки головки 32 мс, время поиска 8 мс на дорожку, имеется 40 дорожек, скорость вращения 360 об/мин. Определите среднее время обращения для гибкого диска с программным разделением на секторы. Затем определите среднее время, необходимое для перевода головки на дорожку 0, а затем время обращения к произвольному сектору.

27. Напишите программу, которая использует приказ считывания состояния накопителя и помещает 1 в ячейку FLAG2, если накопитель 2 защищен от записи.

28. Напишите программу, которая ожидает в цикле освобождения одного из накопителей на гибком диске, а затем переходит к ROUTn, где n – номер первого освободившегося (т. е. находящегося в холостом состоянии) накопителя. В подсистеме имеется контроллер 8272.

29. Постройте схему законченной последовательности вывода блока данных, если в подсистеме внешней памяти имеются контроллеры 8272 и 8237. В схеме необходимо учесть этапы инициализации обоих контроллеров и этапы, необходимые для реализации фаз приказа записи данных.

30. Подробно опишите действия шины при передаче двух смежных байт из памяти в устройство ввода-вывода по 16-битной шине микропроцессора 8086 с применением контроллера ПДП 8237.

31. Рассмотрите 16-битное параллельное устройство ввода-вывода, показанное на рис. 9.55. Приведите программу, которая переводит обе группы в обеих микросхемах 8255A в режим 0, причем порты A и C являются входными, а порт B – выходным. Затем напишите две команды, которые выводят 1720 через порт B.

32. Модифицируйте рис. 9.47 так, чтобы буферному регистру данных и регистрам управления/состояния были назначены смежные четные адреса.

10. ПОЛУПРОВОДНИКОВАЯ ПАМЯТЬ

В наиболее общем смысле слово *память* (*запоминающее устройство*) означает любое устройство, которое хранит информацию для дальнейшего использования. При таком определении память компьютера можно разделить на два класса. Один класс относится к той части компьютера, которая хранит команды и данные, обрабатываемые в текущее время, т. е. к той части, к которой процессор может обращаться непосредственно. Другой класс состоит из средств, которые могут хранить информацию, но эту информацию необходимо передавать в память первого класса, прежде чем к ней может обращаться процессор. В данной книге компоненты из первого класса называются *основной памятью* (или просто *памятью*), а компоненты второго класса – *массовой* (*внешней*) *памятью*. Глава 10 посвящена основной памяти и ее интерфейсу с системной шиной.

Основная память состоит из групп бит, называемых байтами и словами, которые адресуются как целое. До сих пор термин "слово" относился к процессору и обозначал число бит, которое одновременно передается по шине данных и обрабатывается процессором. Однако разработчики памяти понимают под термином "слово" наименьшую группу бит, которая ассоциируется с адресом. В системах на базе микропроцессоров 8086/8088 такими группами являются 8-битные байты. Следовательно, при рассмотрении памяти мы часто будем пользоваться термином "слово" в том контексте, где ранее фигурировал термин "байт". Если же может возникнуть путаница, мы будем говорить "слово памяти" и "слово процессора". Число бит в слове памяти называется его *длиной* (например, длина слова памяти микропроцессоров 8086/8088 равна 8).

Одной из важных характеристик памяти является ее способность сохранять или не сохранять содержимое при выключении питания. Память, сохраняющая свое содержимое, называется *энергонезависимой*, а не сохраняющая – *энергозависимой*. Данная характеристика особенно важна для памяти, в которой хранятся команды. Если эта память является энергозависимой,

то даже при кратковременных выключениях питания команды исчезают и память необходимо загружать при каждом включении компьютера.

Другая важная характеристика зависит от способов обращения к памяти. Если из памяти можно только считывать, она называется *постоянным запоминающим устройством* (*ПЗУ или ROM*), но если память допускает считывание и запись, она называется память со считыванием и записью. *Произвольный доступ* (или *произвольная выборка*) относится к памяти, в которой к любой ячейке можно обратиться за один и тот же временной интервал; при *последовательном доступе* к ячейкам можно обращаться по порядку (например, на ленте). Основная память всех современных компьютеров допускает произвольный доступ; однако применительно к основной памяти термин "произвольная выборка" исторически используется как синоним считывания/записи и память со считыванием/записью называется *запоминающим устройством с произвольной выборкой* (*ЗУПВ или RAM*). Таким образом, имеются два типа основной памяти – ПЗУ и ЗУПВ.

Наиболее распространенным видом энергонезависимой памяти со считыванием/записью является *память на ферритовых сердечниках* (или, короче, *ферритовая память*), состоящая из небольших кольцевых магнитов. Энергозависимая память со считыванием/записью обычно реализуется на полупроводниковых микросхемах. Хотя ферритовая память и является энергонезависимой, она требует громоздкой электроники обрамления и оказывается сравнительно дорогой. Поэтому в большинстве микросистем применяется полупроводниковая память, а для хранения информации при выключении питания предусматривается внешняя память или память с резервным питанием. Энергонезависимые ПЗУ также реализуются на полупроводниковых микросхемах. Из-за отсутствия схем записи ПЗУ наиболее просты для проектирования. В них часто хранятся настраивающие загрузчики, так что систему можно включить, не выполняя инициализацию вручную. Но, разумеется, ПЗУ можно использовать только для хранения данных и программ, которые никогда не изменяются. Так как ферритовая память редко применяется в микропроцессорных системах, она в данной книге не рассматривается.

Глава начинается с обсуждения общей организации памяти. В § 10.2 и 10.3 рассматриваются два вида полупроводниковых ЗУПВ – статические и динамические; § 10.4 посвящен проектированию резервного питания для энергозависимой памяти, наличие которого превращает ее в энергонезависимую; § 10.5 касается разнообразных видов ПЗУ.

10.1. ОБЩАЯ ОРГАНИЗАЦИЯ ПАМЯТИ

Память вычислительной системы обычно состоит из одной или нескольких печатных плат, которые подключены к системной шине. На каждой плате находится модуль, адресуемый старшими битами шины адреса. Как показано на рис. 10.1, в большинстве систем имеются модули ПЗУ и ЗУПВ. Однако следует указать, что в малых системах типа контроллеров могут быть только ПЗУ, а сама память находится на той же печатной плате (и даже в одной и той же микросхеме), что и процессор.

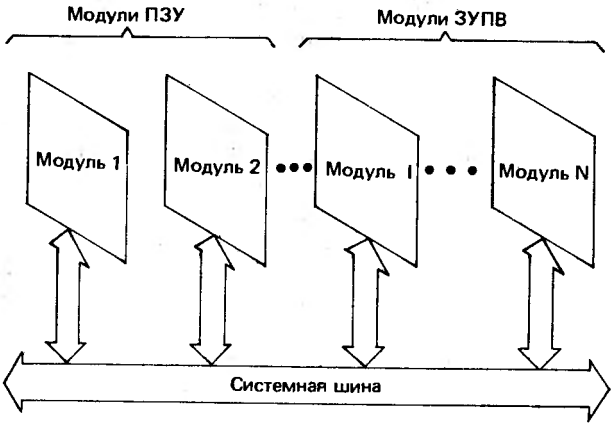


Рис. 10.1. Общая организация памяти

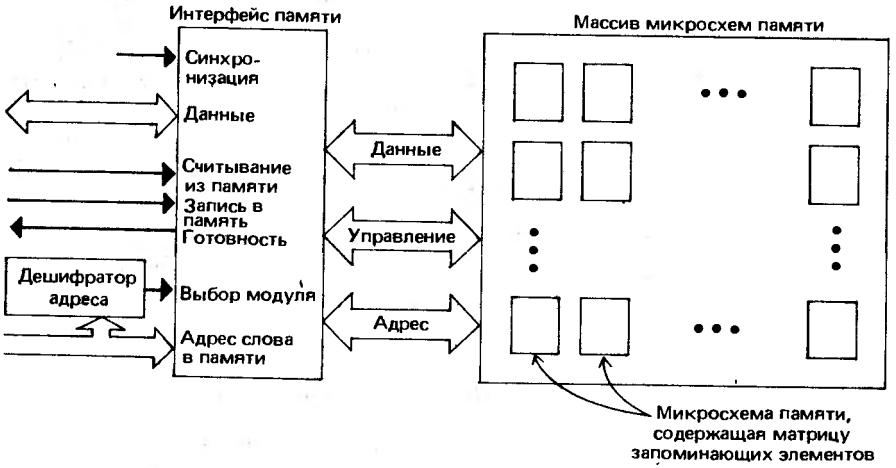


Рис. 10.2. Типичный модуль памяти

Общий вид модуля памяти представлен на рис. 10.2. В его состав входят интерфейс и набор микросхем памяти, каждая из которых содержит массив запоминающих элементов; запоминающий элемент может хранить 1 бит. К элементам в микросхеме можно обращаться отдельно или группами, но в любом случае соблюдаются следующие отношения:

$$\text{Число микросхем в строке} = \frac{\text{Число бит в слове}}{\text{Число элементов в группе}};$$

$$\text{Число микросхем в столбце} = \frac{\text{Число слов в модуле}}{\text{Число групп в микросхеме}}.$$

Говорят, что микросхема памяти имеет организацию $M \times N$, если она содержит M групп из N элементов, а модуль имеет организацию $K \times L$, если он содержит K слов длиной L бит каждое. Для иллюстрации введенных определений на рис. 10.3 приведены несколько модулей, реализованных на основе типичных микросхем памяти.

Важнейшими критериями при проектировании памяти являются: стоимость, емкость, быстродействие, потребление энергии, надежность, энергонезависимость и возможности доступа.

Стоимость модуля обычно складывается из двух компонент, одна из которых не зависит от размера модуля и называется *накладными расходами*, а вторая пропорциональна размеру и называется *инкрементной стоимостью*. Накладные расходы в основном связаны с электроникой обрамления, а инкрементная стоимость соотносится со стоимостью микросхем. Обе компоненты зависят от числа контактных соединений и сложности печатной платы. Следовательно, микросхемы с большей емкостью требуют меньше служебных приборов и обеспечивают выигрыш в стоимости. Так как накладные расходы почти не зависят от емкости модуля, но должны учитываться в каждом модуле, предпочтительнее реализовать память заданной емкости, используя минимум модулей. Еще одним фактором, учитываемым в накладных расходах, является стоимость блока питания. Чем меньше число напряжений питания, тем менее сложной становится разработка блока питания и платы.

Быстродействие памяти характеризуется *временем обращения* (или *доступа*), которое определяется как временной интервал от момента поступления

Размер (емкость) памяти	Тип микросхем ЗУПВ	Число микросхем в столбце	Число микросхем в строке	Число микросхем в модуле
4K X 8	1K X 1	4	8	32
	4K X 1	1	8	8
	256 X 4	16	2	32
	1K X 4	4	2	8
4K X 16	1K X 1	4	16	64
	4K X 1	1	16	16
	256 X 4	16	4	64
	1K X 4	4	4	16
16K X 8	1K X 4	16	2	32
	4K X 1	4	8	32
	8K X 1	2	8	16
	16K X 1	1	8	8
64K X 8	16K X 1	4	8	32
	64K X 1	1	8	8

Рис. 10.3. Типичные массивы микросхем памяти

стабильных сигналов адреса до получения выходных данных. Время обращения зависит от многих факторов и даже связано с емкостью микросхемы. Для быстродействующих транзисторов приходится отводить бо́льшую площадь кристалла, что уменьшает число запоминающих элементов. Кроме того, быстродействующие микросхемы, которые обычно производятся по биполярной технологии, оказываются более дорогими и энергоемкими.

Потребляемая энергия очень важна для систем, которые иногда должны работать от аккумуляторов или солнечных элементов (например, в космических объектах). Определяющим фактором для потребляемой каждым запоминающим элементом энергии является применяемая технология. Наиболее часто память с минимальным потреблением энергии производится по КМОП-технологии. Основной ее недостаток связан с увеличением площади кристалла для каждого запоминающего элемента, что уменьшает емкость микросхемы. К сожалению, потребляемая энергия и быстродействие связаны пропорциональной зависимостью, поэтому оптимизировать оба эти показателя сложно и дорого. Сейчас наиболее хороший компромисс между быстродействием, потреблением энергии и емкостью обеспечивает высококачественная МОП-технология.

Поскольку надежность микросхем после их тщательного контроля довольно высока, надежность модуля сильно зависит от числа паяных соединений и сложности платы. Следовательно, при уменьшении общего числа контактов надежность модуля увеличивается, что дополнительно стимулирует минимизацию числа микросхем в модуле.

Энергонезависимость и возможности доступа во многом определяются условиями применения. Если применение не требует энергонезависимости, нет никаких причин обеспечивать ее. Когда же требуется энергонезависимое ЗУПВ, приходится использовать ферритовую память, а для полупроводниковой памяти вводить резервное питание. По возможности следует как можно шире применять ПЗУ как наименее дорогие, энергонезависимые, надежные и помехоустойчивые устройства, обладающие высокой плотностью упаковки.

10.2. СТАТИЧЕСКИЕ ЗУПВ

Основной запоминающий элемент на шести МОП-транзисторах, применяемый в статической памяти, показан на рис. 10.4. Хранимая информация определяется состояниями транзисторов Q_1 и Q_2. В этой транзисторной паре с перекрестными связями один из транзисторов включен, а другой выключен. Состояние, когда Q_2 включен, а Q_1 выключен, представляет собой 1, а противоположное состояние — 0. Транзисторы Q_3 и Q_4 выполняют функции резисторов, а транзисторы Q_5 и Q_6 действуют как разрешающие вентили. В операции записи сначала производится выбор элемента посредством установки высокого уровня на линии выбора. При этом транзисторы Q_5 и Q_6 действуют как короткозамкнутые цепи, поэтому линия считывания/записи 1 подключается к затвору Q_2, а линия считывания/записи 0 — к затвору Q_1. Для записи в элемент 1 на линии считывания/записи 1 устанавливается 1, а на линии считывания/записи 0 — 0; это приводит к включению Q_2 и выключению

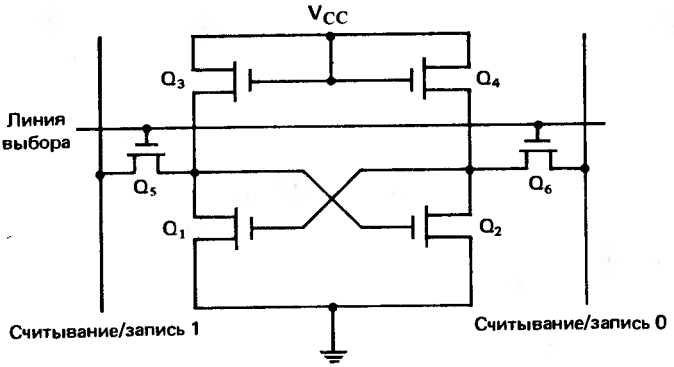


Рис. 10.4. Схема 6-транзисторного запоминающего элемента статического ЗУПВ

Q_1. Если же в элемент необходимо записать 0, на линиях считывания/записи действуют противоположные сигналы. В любом случае установленные состояния Q_1 и Q_2 не изменяются до следующей операции записи. Считывание из элемента производится просто подачей напряжения на линию выбора. При этом состояние Q_1 передается на линию считывания/записи 0, а состояние Q_2 – на линию считывания/записи 1.

Число запоминающих элементов и их организация в статической памяти варьируются в широких пределах. Диапазон размеров составляет от 256 x 4 до 16К x 1. ЗУПВ 254 x 4 состоит из 256 ячеек, каждая из которых имеет 4 бита, а ЗУПВ 16К x 1 обеспечивает 16К ячеек, каждая из которых содержит всего 1 бит. Общая организация статического ЗУПВ 1К x 1 представлена на рис. 10.5. Запоминающие элементы организованы в матрицу из 32 строк и 32 столбцов. Биты адреса A9-A0 разделены на адреса строк и столбцов, определяя одну из 1024 ячеек. Входы адреса строки A4-A0 дешифрируются и выбирают одну из 32 строк запоминающих элементов. Входы адреса столб-

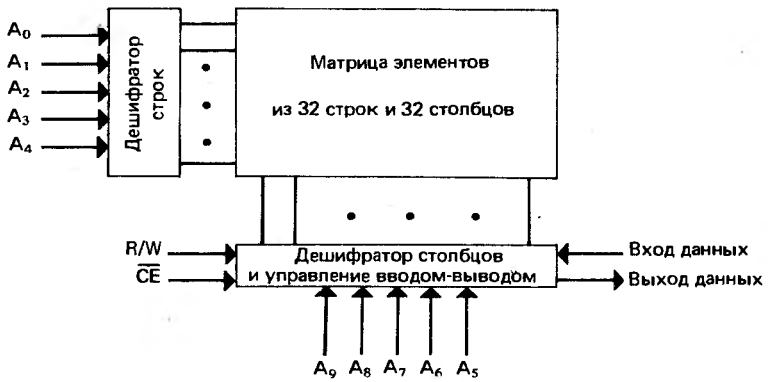


Рис. 10.5. Структура микросхемы памяти с организацией 1К x 1

ца A9-A5 не только выбирают один из столбцов, но еще и разрешают схемы ввода-вывода, состоящие из драйверов и усилителей считывания. Эти схемы позволяют вывести хранимый бит в операции считывания и изменить его в операции записи. Вход R/$\overline{W}$ (считывание/запись) определяет тип операции (высокий уровень при считывании и низкий при записи). Вход разрешения кристалла $\overline{CE}$ предназначен для выбора соответствующей строки микросхем в модуле памяти.

На рис. 10.6 показан модуль памяти 4К × 8, построенный из микросхем 1К × 1. Если кристалл разрешен, выполняется операция считывания или записи в соответствии с уровнем сигнала R/$\overline{W}$. В противном случае сигнал R/$\overline{W}$ не распознается и выход переводится в высокоимпедансное состояние. Это позволяет непосредственно соединять выходы нескольких микросхем, поэтому, применяя в столбцах микросхемы 1К × 1, можно реализовать модули 2К × 8, 4К × 8 и т. д., причем каждый столбец "вносит" один бит в байт данных. Выводимый бит зависит не только от сигналов на линиях адреса, но и от того, на какие микросхемы подается сигнал разрешения кристалла. Каждая строка в массиве подключена к линии разрешения строки, а всеми линиями разрешения кристалла управляют старшие биты адреса (в данном примере – линии A11 и A10). Когда строка выбрана, каждая микросхема в строке будет вводить или выводить бит в соответствии с сигналами на линиях A9-A0. Если адрес содержит 16 бит, линии A15-A12 выбирают модуль, A11 и A10 – строку, а A9-A0 – биты в микросхемах, образующие адресованный байт.

Из-за сложности запоминающего элемента плотность упаковки статической памяти меньше, чем динамической памяти. Кроме того, статическая память потребляет больше энергии, так как в запоминающем элементе один из транзисторов всегда включен. Основное достоинство статической памяти заключается в том, что ее не нужно регенерировать.

Микросхемы полупроводниковой памяти легко объединять друг с другом, так как они имеют встроенную электронику обрамления. Однако временные ограничения входных сигналов довольно критичны, а временные характеристики микросхем варьируются. Для обеспечения правильной работы логика управления на плате памяти должна формировать входы адреса и управляющие сигналы, соответствующие спецификациям применяемых микросхем. Временная диаграмма входов в операции считывания отличается от диаграммы в операции записи.

Наиболее важным временным параметром при выборе микросхем является *время обращения*. Максимальная временная задержка от входа адреса до выхода данных больше задержки между разрешением кристалла и выходом данных, поэтому временем обращения обычно считается первый параметр. Время обращения наиболее распространенных МОП-ЗУПВ изменяется от 50 до 500 нс.

В операции считывания после стабилизации выходных данных вход адреса нельзя сразу же снимать, чтобы запустить следующую операцию считывания. Это объясняется тем, что перед следующей операцией прибору требуется некоторое время (называемое *временем восстановления при считывании*), что-

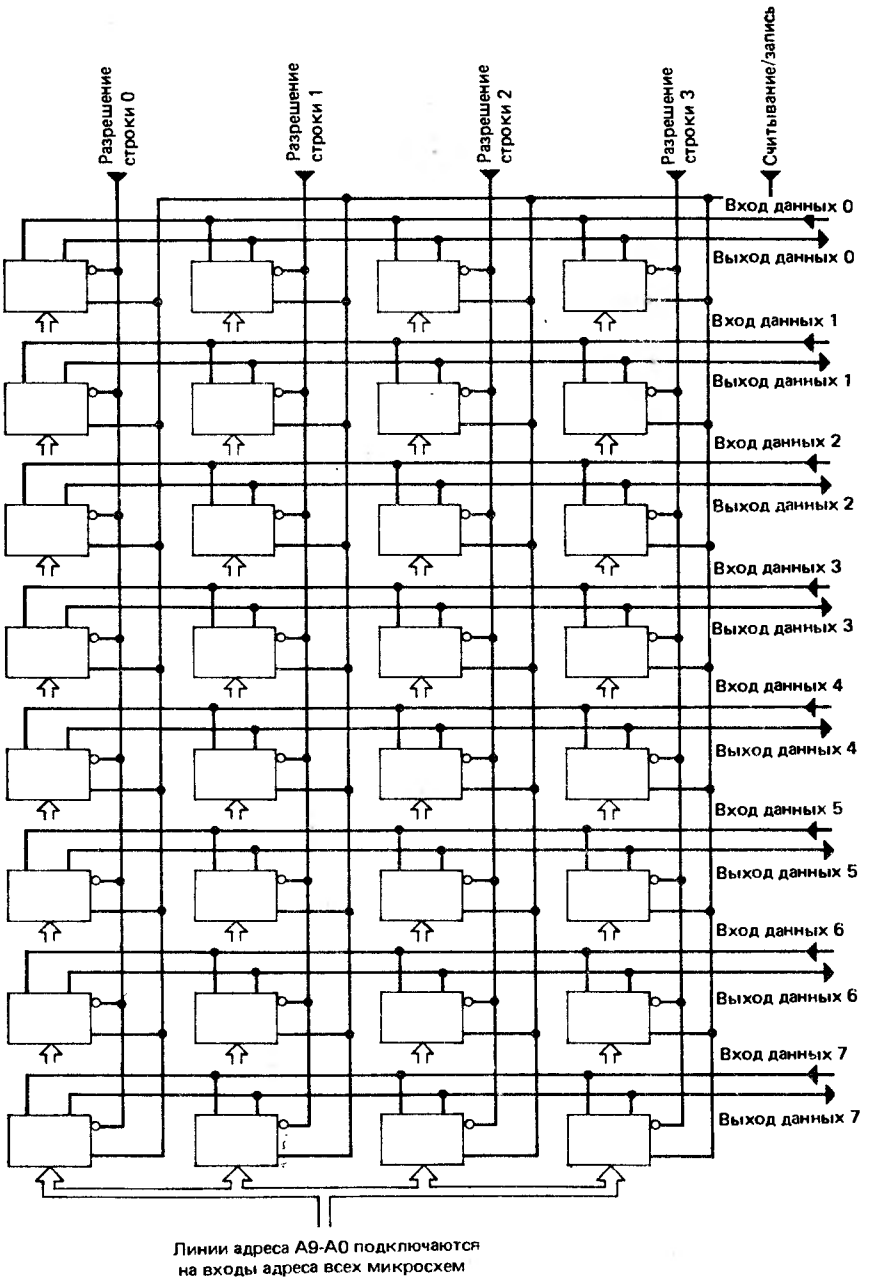


Рис. 10.6. Модуль 4К × 8 на микросхемах памяти с организацией 1К × 1

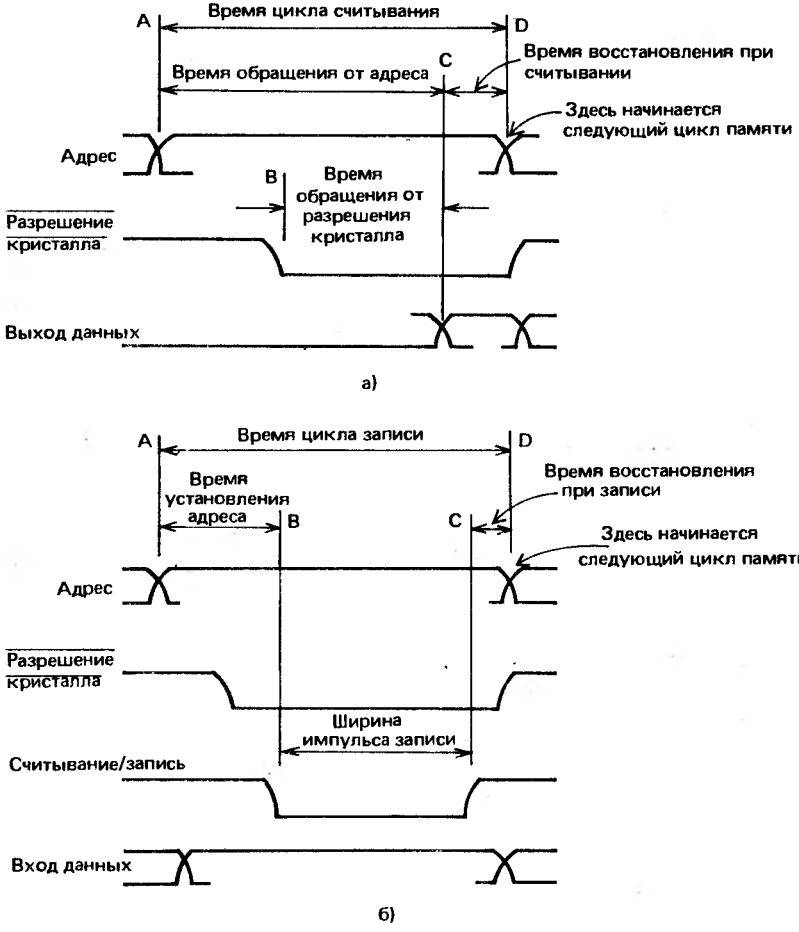


Рис. 10.7. Временны́е диаграммы циклов обращения к памяти:
а – цикл считывания; *б* – цикл записи

бы закончить внутренние действия. Сумма времени обращения и времени восстановления при считывании образует *время цикла считывания*. Именно это время необходимо между запуском операции считывания и запуском следующего цикла памяти. *Время цикла записи* можно определить аналогично и оно может отличаться от времени цикла считывания. На рис. 10.7, *а* приведена временная диаграмма цикла считывания из памяти. Адрес подается в точке A, которая является началом цикла считывания, и должен сохраняться стабильным весь цикл. Чтобы уменьшить время обращения, вход разрешения кристалла следует подавать до точки B. Выходные данные становятся действительными после точки C и сохраняются такими, пока действуют входы

адреса и разрешения кристалла. На временной диаграмме считывания вход $R/\overline{W}$ не показан, но он должен иметь высокий уровень в течение всего цикла.

В типичном цикле записи, показанном на рис. 10.7, *б*, кроме входов адреса и разрешения кристалла необходимо еще подать отрицательный импульс на линию $R/\overline{W}$ и запоминаемые данные. В течение всего этого цикла данные просто сохраняются стабильными. Однако подача импульса записи имеет два критичных временных параметра: время установления адреса и ширина импульса записи. *Время установления адреса* — это время, необходимое для стабилизации адреса, т. е. временной интервал, который должен пройти до подачи импульса записи. На рис. 10.7, *б* время установления адреса равно временному интервалу между точками A и B. *Ширина импульса записи* определяет продолжительность активного низкого уровня на входе записи. Время цикла записи равно временному интервалу между точками A и D; оно представляет собой сумму времени установления адреса, ширины импульса записи и времени восстановления при записи. В некоторых микросхемах допускаются нулевые времена восстановления в обеих операциях.

Важно отметить, что время обращения и время цикла являются минимальными временными требованиями для самих микросхем. Время обращения и время цикла во всей системе памяти значительно больше из-за задержек, вносимых логикой управления вводом-выводом, логикой системной шины и логикой интерфейса памяти.

На рис. 10.8 представлен модуль статической памяти 16К × 8 для микропроцессора 8088 в максимальном режиме. Предполагается, что входы $\overline{CE}$ и $\overline{WE}$ и линии D7-D0 статического ЗУПВ 4К × 8 имеют взаимосвязи, указанные в табл. 10.1.

Таблица 10.1

Сигналы в статическом ЗУПВ

$\overline{CE}$	$\overline{WE}$	Функция	D7-D0
1	X	Не выбран	Высокоимпедансное состояние
0	1	Считывание	На линии выводятся данные
0	0	Запись	С линий воспринимаются данные

Если микросхема не выбрана (т. е. $\overline{CE} = 1$), она переходит в пассивное состояние, позволяющее работать с пониженным потреблением энергии.

Шина адреса разделяется на две части: линии A19-A14 применяются для выбора модуля, а линии A13-A12 подаются в логику разрешения кристалла, которая более подробно изображена на рис. 10.9, *а*. Она имеет четыре выхода $\overline{CE0}$-$\overline{CE3}$, из которых в любой момент времени может быть активным только один. Выход $\overline{CE0}$ подключен на вход микросхемы, имеющей младшие 4К адресов, и активен, когда A13 = A12 = 0. Аналогично сигнал $\overline{CE1}$ активен, когда A13 = 0 и A12 = 1; $\overline{CE2}$ активен при A13 = 1 и A12 = 0 и $\overline{CE3}$ активен, когда A13 = A12 = 1. Во всех случаях на линии выбора модуля должен действовать сигнал 1, прежде чем формируется активный сигнал $\overline{CE}$. Сигнал на линии

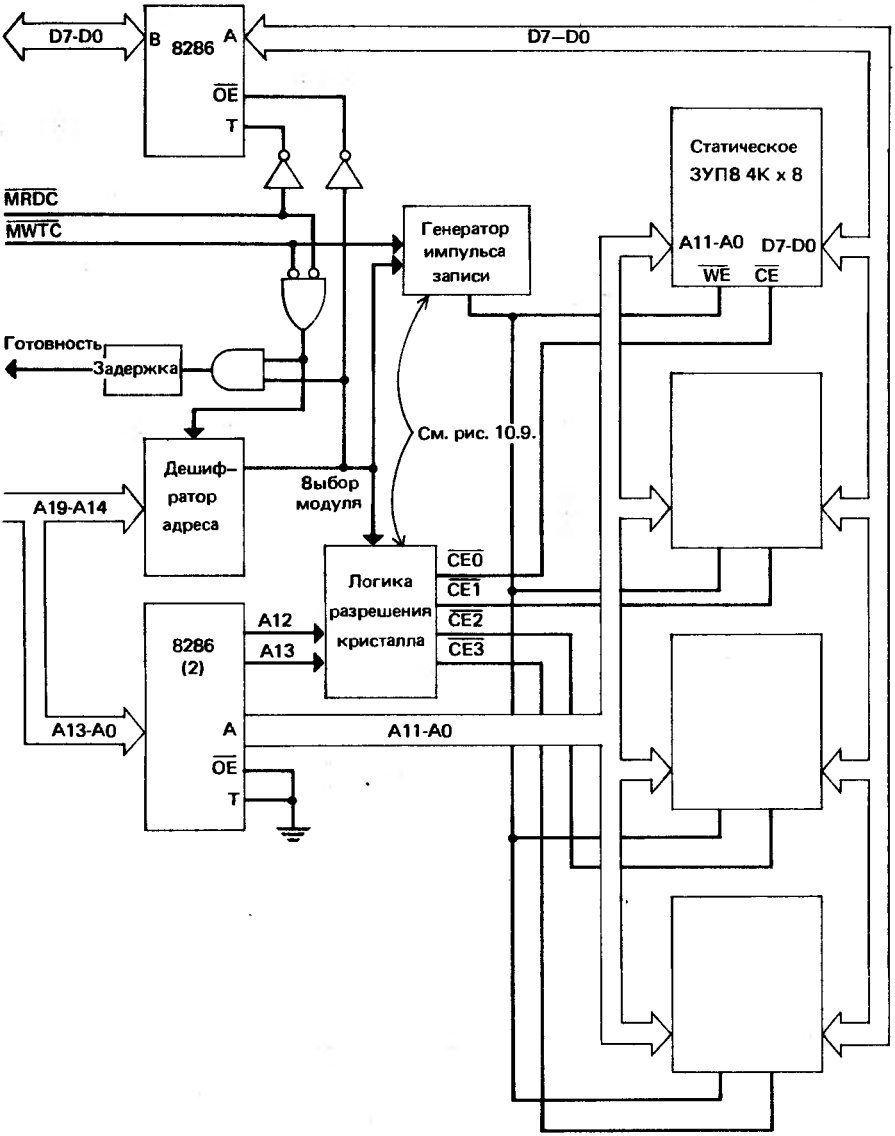


Рис. 10.8. Модуль памяти 16К × 8 для микропроцессора 8088 в максимальном режиме

выбора модуля активен только тогда, когда модуль выбран и распознается операция считывания или записи. Линии адреса A11-A0 подаются на входы A11-A0 всех микросхем памяти.

В генератор импульса записи, построенный на двух одновибраторах, подаются сигнал $\overline{\text{MWTC}}$ и линия выбора модуля (см. рис. 10.9, *б*). Выход генера-

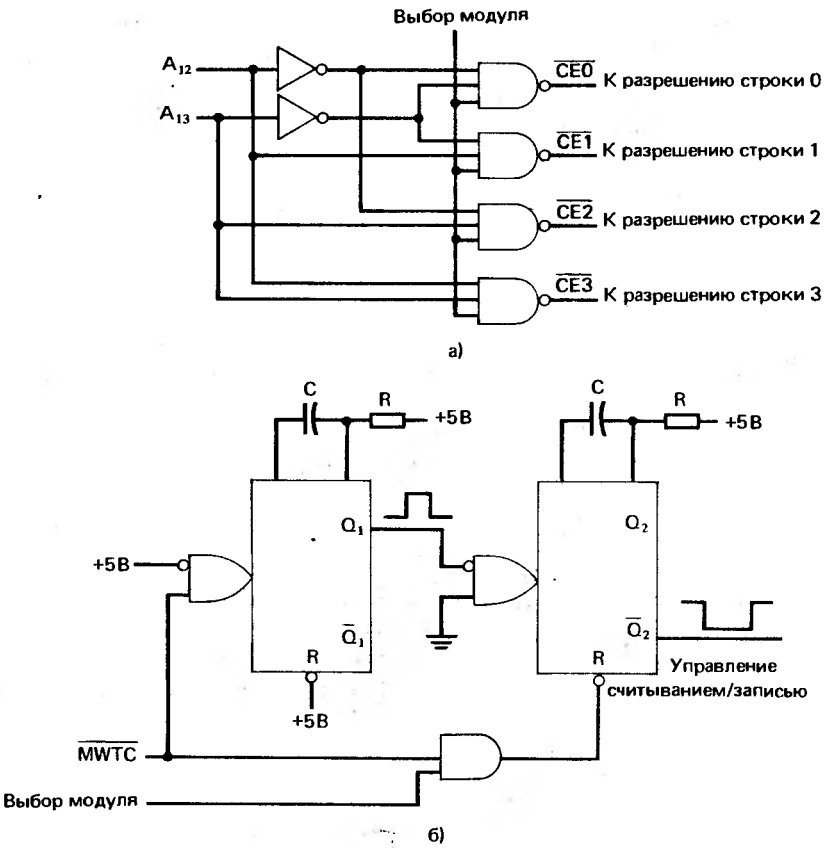


Рис. 10.9. Вспомогательная логика для модуля памяти, показанного на рис. 10.8: *а* – логика разрешения кристалла; *б* – генератор импульса записи

тора подключен на входы разрешения записи $\overline{WE}$ всех микросхем памяти и вызывает загрузку данных с линий D7-D0 в адресованный байт.

Отметим, что в микросхемах на рис. 10.6 имеются отдельные линии входных и выходных данных, а в микросхемах на рис. 10.8 линии данных являются двунаправленными. Внутри микросхемы необходимы отдельные линии для считывания и записи в запоминающие элементы, но можно разделить двунаправленные сигналы на сигналы считывания и записи в самом устройстве.

10.3. ДИНАМИЧЕСКИЕ ЗУПВ

Как и в статических ЗУПВ, память в кристаллах динамической памяти организована в матрицу запоминающих элементов. Простейший элемент динамического ЗУПВ состоит всего из одного транзистора и одного конденсатора (см. рис. 10.10). Хранение в элементе 1 или 0 определяется наличием или отсутствием заряда на конденсаторе. В операции считывания на одной из ли-

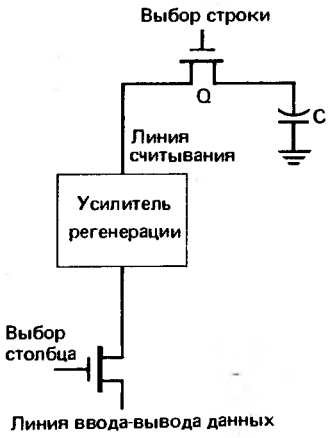


Рис. 10.10. Типичный однотранзисторный запоминающий элемент динамического ЗУПВ

ний выбора строки устанавливается высокий уровень посредством дешифрирования адреса строки (младшие биты адреса). Сигнал на этой линии включает ключевые транзисторы Q во всех элементах выбранной строки. При этом подключенный к каждому столбцу усилитель регенерации воспринимает уровень напряжения на соответствующем конденсаторе и интерпретирует его как 0 или 1. Адрес столбца (старшие биты адреса) разрешает один элемент в выбранной строке на выход. Во время этих действий конденсаторы во всей строке разряжаются. Чтобы сохранить информацию, усилители регенерации производят повторную запись в элементы этой же строки. Операция записи осуществляется аналогично, но в выбранном элементе запоминаются входные данные, а остальные элементы в выбранной строке просто регенерируются.

Из-за разряда конденсатора током утечки p–n-перехода элементы динамической памяти необходимо периодически считывать и восстанавливать – этот процесс называется *регенерацией памяти*. Скорость разряда увеличивается с повышением температуры и период регенерации составляет 1 . . . . . 100 мс. При рабочей температуре +70°С типичное значение его 2 мс. Хотя строка элементов регенерируется в операции считывания или записи, случайность обращений к памяти не может гарантировать, что каждое слово в модуле памяти регенерируется с требуемым периодом 2 мс. Необходима периодическая регенерация памяти с помощью специальных циклов.

В *цикле регенерации* в микросхемы подается адрес строки и выполняется операция считывания, чтобы восстановить выбранную строку запоминающих элементов. Однако этот цикл отличается от обычного цикла считывания следующими моментами:

1. Входной адрес подается в микросхемы не с шины адреса, а от специального двоичного счетчика, называемого *счетчиком адреса регенерации*. В каждом цикле регенерации производится инкремент этого счетчика и он проходит по всем адресам строк. Адрес столбца в регенерации не участвует, так как все элементы строки восстанавливаются одновременно.

2. В цикле регенерации разрешаются все микросхемы, поэтому она производится одновременно во всех микросхемах модуля памяти. Такой прием сокращает число циклов регенерации. В обычном же цикле считывания разрешена максимум одна строка микросхем.

3. Кроме входа разрешения кристалла динамическое ЗУПВ обычно имеет входной сигнал разрешения выхода данных. Эти два сигнала управления объединяются внутри микросхемы так, что выход данных переводится в высокоимпедансное состояние, если оба эти входа не активны. В цикле регенерации

сигнал разрешения выхода данных имеет пассивный уровень. Это необходимо потому, что все микросхемы в одном и том же столбце оказываются выбранными, а их выходы данных соединены друг с другом. Во время обычного цикла считывания выбрана только одна строка микросхем и сигналы разрешения выхода данных всех строк имеют активный уровень.

Рассмотрим модуль памяти емкостью 16К байт, реализованный на динамических ЗУПВ 4К × 1. Массив микросхем имеет 4 строки и 8 столбцов. Каждая микросхема содержит 64 строки и 64 столбца запоминающих элементов и имеет отдельные входы адреса строки (6 бит) и адреса столбца (6 бит). Предполагается, что входы разрешения кристалла и разрешения выхода обозначены $\overline{CE}$ и $\overline{CS}$. Схема на рис. 10.11 показывает логику, необходимую для формирования сигналов разрешения кристалла и адреса регенерации. Сигнал цикла регенерации формируется синхрогенератором в модуле памяти. Если текущий цикл является циклом регенерации, мультиплексор выбирает адрес строки от счетчика адреса регенерации; в противном случае адрес строки берется с шины адреса. Считая, что каждый элемент в микросхемах должен восстанавливаться в течение 2 мс, найдем, что цикл регенерации необходимо инициировать через $2 \times 10^{-3}/64 = 31{,}25$ мкс. В конце каждого цикла производится инкремент двоичного счетчика на 1 и он показывает следующую строку, подлежащую регенерации. Во время этого цикла все микросхемы разрешены для выполнения операции считывания активными сигналами $\overline{CE}$. Выходы данных переводятся в высокоимпедансное состояние пассивным уровнем сигнала разрешения выхода.

Кроме необходимости введения логики регенерации, главным недостатком динамических ЗУПВ является то, что во время цикла регенерации в модуле нельзя инициировать обычные операции считывания и записи до окончания этого цикла. В результате для удовлетворения запроса считывания или записи может потребоваться вдвое больше времени, если начат цикл регенерации. Если время цикла равно 400 нс (для всех циклов памяти – регенерации, считывания и записи), на регенерацию затрачивается

$$\frac{64 \times 400 \times 10^{-9}}{2 \times 10^{-3}} \times 100\,\% = 1{,}28\,\%$$

времени памяти.

Однако динамические ЗУПВ привлекают разработчиков памяти (особенно большой емкости) по нескольким причинам, основными из которых являются:

высокая плотность упаковки. В статических ЗУПВ запоминающий элемент состоит из шести МОП-транзисторов, а в динамических – из трех, двух и даже одного транзистора. В результате можно увеличить число элементов на кристалле и сократить число микросхем, необходимых для построения модуля. Емкость микросхем динамических ЗУПВ составляет 16К × 1, 64К × 1 и более;

малое потребление энергии. Удельное потребление энергии в динамических ЗУПВ гораздо меньше, чем в статических ЗУПВ: менее 0,05 и 0,2 мВт/бит соответственно. Это позволяет уменьшить мощность, потребляемую систе-

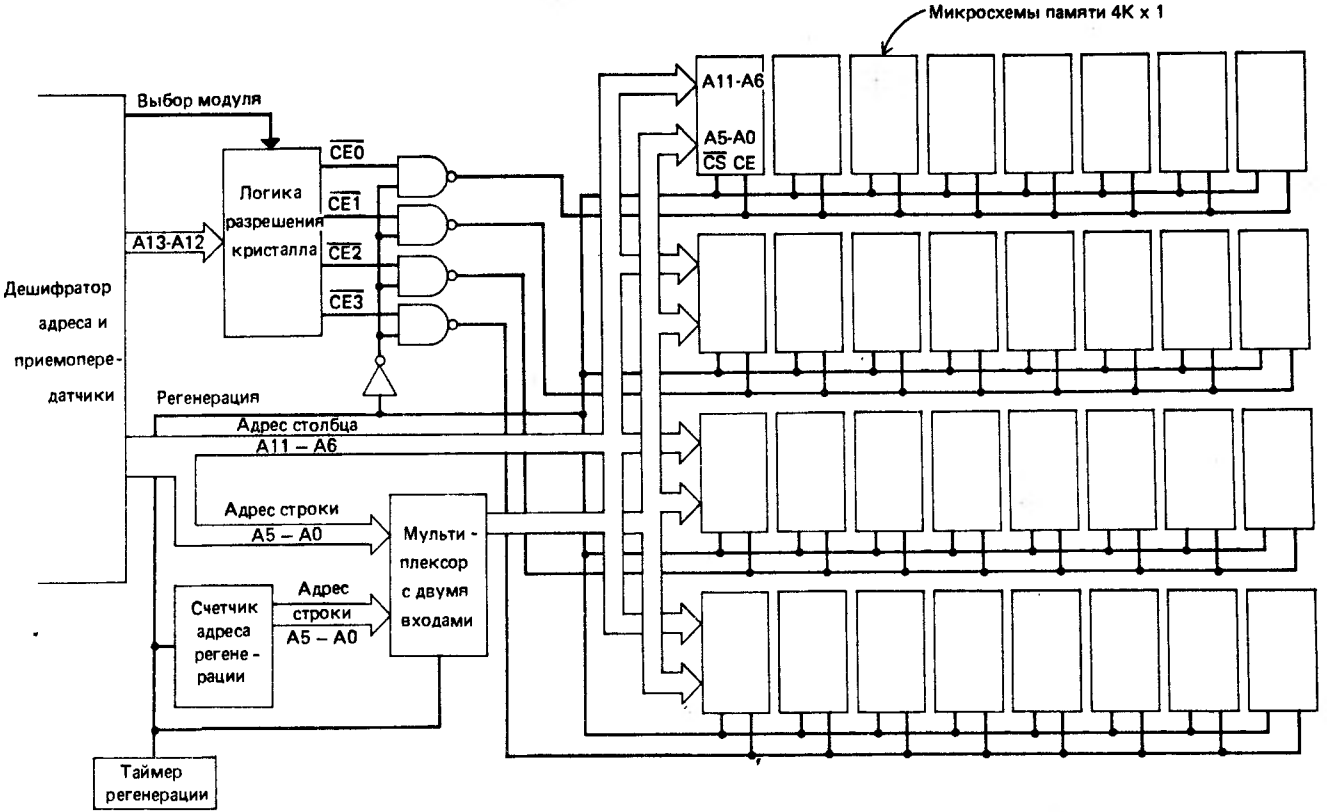
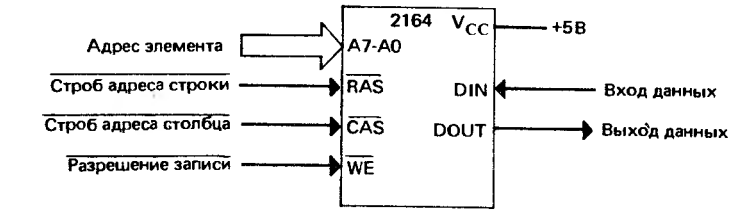


Рис. 10.11. Логика адреса регенерации для ЗУПВ 16К × 8, реализованного на микросхемах динамической памяти 4К × 1

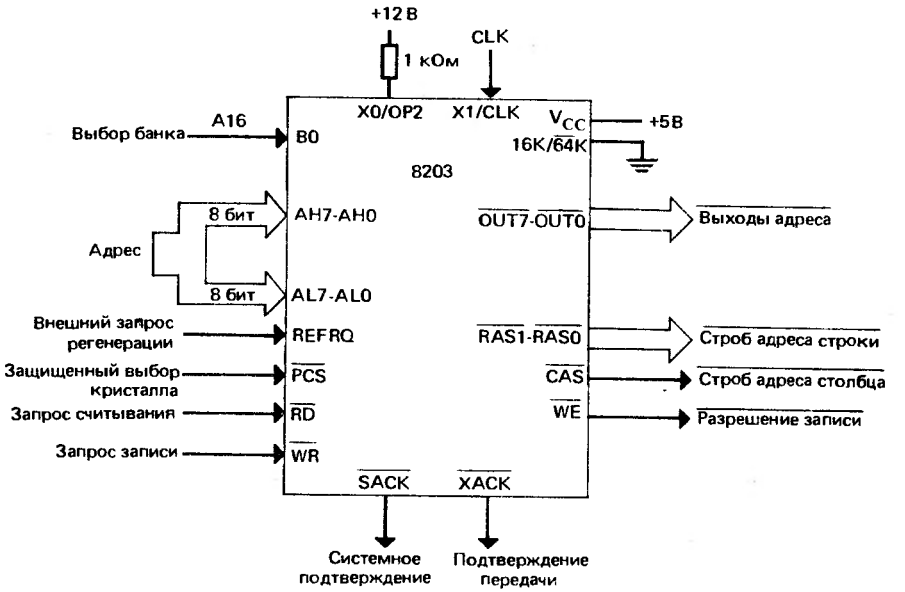
мой, и снизить стоимость. Кроме того, для динамических ЗУПВ требуется исключительно мало энергии в пассивном режиме, что делает их привлекательными для реализации энергонезависимой памяти с резервным питанием;

экономичность. Удельная стоимость динамических ЗУПВ ниже стоимости статических ЗУПВ. Однако динамические ЗУПВ требуют больше вспомогательных схем, поэтому при реализации памяти небольшой емкости они дают незначительный экономический эффект или совсем не дают его.

В ЗУПВ большой емкости адрес строки и адрес столбца обычно разделяют одни и те же контакты, что сокращает число контактов корпуса микросхемы. Некоторые микросхемы памяти содержат логику управления регенерации, а также логику управления контактами адреса строки/столбца. Фирма Intel выпустила контроллер динамических ЗУПВ 8203, который рассчитан на ми-



а)



б)

Рис. 10.12. Входные и выходные линии микросхем 2164 (*а*) и 8203 (*б*)

кросхемы памяти 2117, 2118 и 2164. Далее рассматривается применение контроллера 8203 с микросхемами 2164, имеющими организацию 64К × 1. Сигналы приборов 8203 и 2164 приведены на рис. 10.12. (Контроллер 8203 может работать в двух режимах в зависимости от уровня на входе 16К/$\overline{64К}$; показаны назначения контактов только для режима 64К.) Микросхема 2164 имеет четыре матрицы 128 × 128 запоминающих элементов и всего восемь входов адреса A7-A0. Это означает, что адреса строки и столбца должны разделять одни и те же контакты и приниматься друг за другом. Адрес строки стробируется отрицательным импульсом на входе $\overline{RAS}$, а адрес столбца — отрицательным импульсом на входе $\overline{CAS}$ (в это время $\overline{RAS}$ сохраняется нулевым). Старшие биты адреса строки и адреса столбца определяют одну из четырех матриц запоминающих элементов. В цикле регенерации вход адреса A7 не используется и все четыре матрицы работают одновременно. Таким образом, за 128 циклов осуществляется регенерация всей микросхемы.

Временные диаграммы циклов считывания, записи и (только) регенерации изображены на рис. 10.13. В цикле считывания сигнал $\overline{WE}$ должен быть пассивным до подачи импульса $\overline{CAS}$ и оставаться пассивным до окончания импульса $\overline{CAS}$. После стробирования адреса столбца формируется высокий уровень сигнала $\overline{RAS}$ и при сигналах $\overline{RAS}$ = 1 и $\overline{CAS}$ = 0 на линии DOUT появляется бит данных. В цикле записи сигнал DIN необходимо подать к моменту, когда сигнал $\overline{CAS}$ переводится на низкий уровень, но после того, как на входе $\overline{WE}$ устанавливается низкий уровень. Запись выполняется по входу DIN, когда $\overline{RAS}$ = $\overline{CAS}$ = $\overline{WE}$ = 0. В цикле записи линия DOUT находится в высокоимпедансном состоянии. В цикле регенерации стробируется только адрес строки, а сигнал $\overline{CAS}$ остается пассивным. Линия DOUT находится в высокоимпедансном состоянии.

Контроллер 8203 формирует сигналы, временная диаграмма которых удовлетворяет требованиям микросхемы 2164. На линиях OUT7-OUT0 действуют адреса строк и столбцов в правильной последовательности, линии $\overline{RAS1}$-$\overline{RAS0}$ несут стробы адреса строки для двух банков микросхем 2164, а по линиям $\overline{CAS}$ и $\overline{WE}$ во все микросхемы памяти в модуле подаются сигналы строба адреса столбца и расширения записи. (Отметим, что контроллер 8203 выдает адреса инвертированными; это обстоятельство не вызывает никаких проблем.)

Вход выбора банка B0 определяет один из двух активных сигналов $\overline{RAS}$. Линии AL7-AL0 используются для генерирования адреса строки, а линии AH7-AH0 — для генерирования адреса столбца. Обычно циклы регенерации формирует сам контроллер 8203, но вход REFRQ позволяет инициировать циклы регенерации от внешнего источника. Выбор модуля осуществляется по входу $\overline{PCS}$. Он называется входом защищенного выбора кристалла, поскольку, как только он станет активным, цикл памяти аннулировать нельзя, даже если сигнал $\overline{PCS}$ сразу же переходит в пассивное состояние. Входы $\overline{RD}$ и $\overline{WR}$ определяют в памяти операцию считывания или записи.

Выход $\overline{XACK}$ представляет собой строб, показывающий, что данные доступны в цикле считывания, или что данные записаны в цикле записи. Его

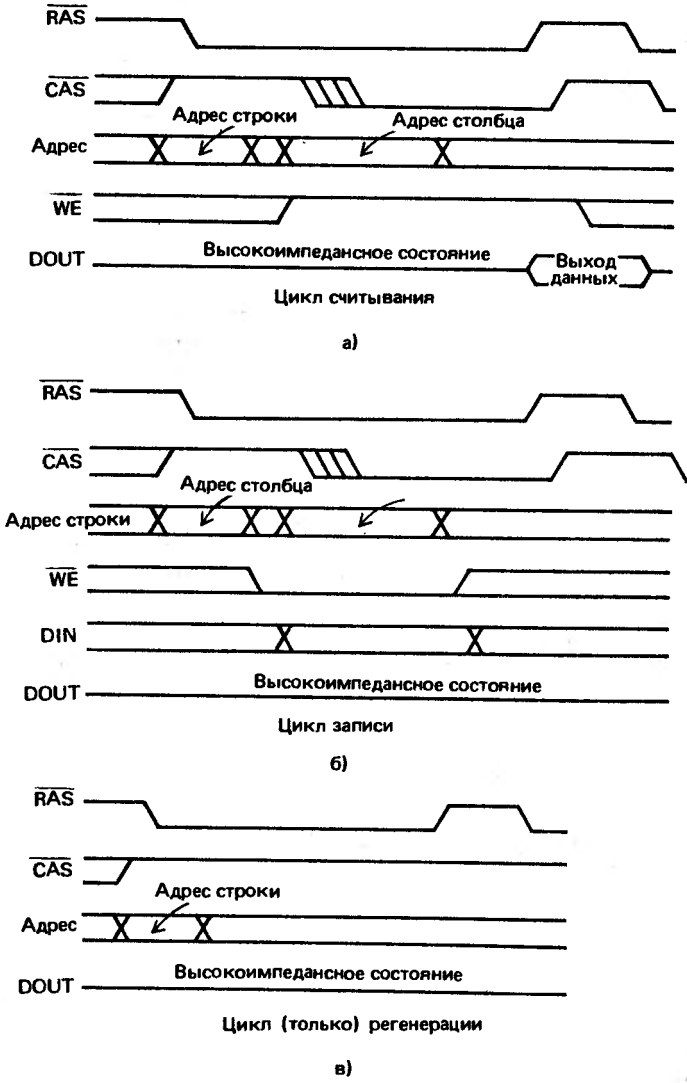


Рис. 10.13. Временны́е диаграммы работы микросхемы 2164

можно использовать для стробирования данных в регистры-защелки и для выдачи в процессор сигнала готовности. Выход $\overline{SACK}$ сигнализирует о начале цикла обращения к памяти и, если при запросе памяти происходит цикл регенерации, сигнал $\overline{SACK}$ задерживается до начала цикла считывания или записи. Если известно, что быстродействие микросхем памяти недостаточно велико, чтобы гарантировать окончание цикла считывания к концу такта T_3 или

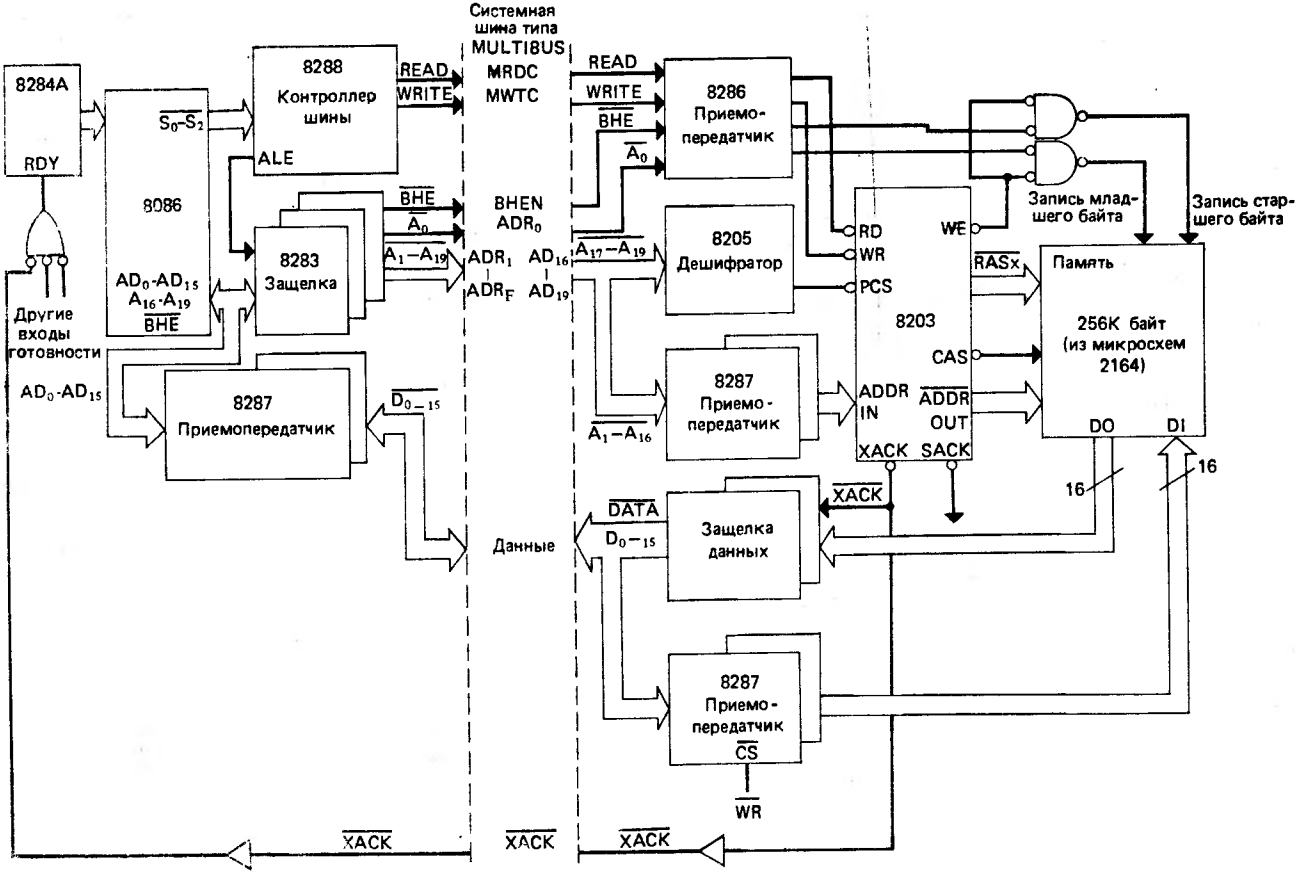


Рис. 10.14. Система с микропроцессором 8086 в максимальном режиме, шиной MULTIBUS и модулем памяти 256К байт

окончание цикла записи к концу такта T_4, выход $\overline{SACK}$ используется как сигнал готовности вместо выхода $\overline{XACK}$. При этом экономятся такты ожидания при использовании сигнала $\overline{XACK}$.

На входы X0 и X1 подключается осциллятор либо на вход CLK подается внешний сигнал синхронизации (OP2 подключается при этом к напряжению +12 В). Этот сигнал можно взять с линии синхронизации шины либо от генератора в модуле памяти. Основным питанием служит напряжение +5 В, но для OP2 необходимо еще напряжение +12 В. (Мы не рассматриваем использование входа REFRQ для опережающего считывания и цикл считывания-модификации-записи.)

На рис. 10.14 показана организация модуля памяти 256К байт, содержащего контроллер 8203 и 32 микросхемы 2164. Модуль рассчитан на микропроцессор 8086 в максимальном режиме и шину MULTIBUS. Предполагается, что в этой разработке шины адреса и данных инвертированы, поэтому для интерфейса с ними применяются микросхемы 8283 и 8287 вместо "обычных" микросхем 8282 и 8286. Массив микросхем имеет 16 столбцов, что допускает обращения к словам. При этом необходимо, чтобы в определении записи только младшего байта, только старшего байта или всего слова участвовали сигналы $\overline{BHE}$ и A0 с линий шины и сигнал $\overline{WE}$.

Еще один способ уменьшения числа обслуживающих микросхем в динамических ЗУПВ заключается в том, чтобы разместить логику регенерации в каждой микросхеме, чтобы она регенерировала сама себя. Такая микросхема называется интегрированным ЗУПВ и за исключением того, что обращения к памяти иногда задерживаются из-за циклов регенерации, она представляется для пользователя статическим ЗУПВ. Примером такого подхода служит интегрированное ЗУПВ 2186/7 фирмы Intel, имеющее организацию 8К x 8. Разводка контактов этой микросхемы характерна для статических ЗУПВ; в частности, она имеет входы $\overline{OE}$, $\overline{WE}$ и $\overline{CE}$ с тем же назначением.

10.4. РЕЗЕРВНОЕ ПИТАНИЕ ДЛЯ ПОЛУПРОВОДНИКОВОЙ ПАМЯТИ

Один из главных недостатков применения МОП-ЗУПВ в основной памяти связан с тем, что хранимая информация исчезает даже при кратковременных сбоях питания. Этот недостаток устраняется обеспечением резервного источника питания, который работает при отказе основного источника. По соображениям стоимости от неисправности питания защищается только часть системы памяти. При отказе источника питания состояние и важные данные выполняемой программы запоминаются в модулях энергонезависимой памяти; после восстановления его наличие этой информации позволяет продолжить выполнение программы.

Некоторые МОП-ЗУПВ в режиме пассивного хранения потребляют намного меньше энергии по сравнению с обычным режимом работы, когда память выполняет операции записи и считывания. Для уменьшения потребляемой энергии от резервного источника питания модуль памяти переводится в пассивный режим, в котором все микросхемы запрещены и просто сохраняют

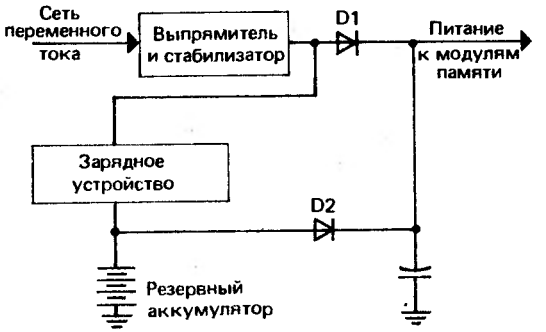


Рис. 10.15. Система питания с резервным аккумулятором

информацию. Благодаря такой возможности в качестве резервного источника питания МОП-памяти применяются аккумуляторы.

На рис. 10.15 показан блок питания с аккумуляторным резервом. При нормальной работе питание подается от блока питания, который преобразует сетевое переменное напряжение в стабилизированное постоянное, поддерживаемое на уровне V_{CC}. Выходное напряжение аккумулятора меньше обычного V_{CC}, поэтому диод D1 смещен в прямом направлении, а диод D2 – в обратном. При отказе основного источника питания конденсатор разряжается до тех пор, пока напряжение на нем не достигнет напряжения аккумулятора. В этот момент диод D2 смещается в прямом направлении и питание в память подается от аккумулятора. После восстановления основного источника диод D2 запирается, а аккумулятор подзаряжается. Часто для коммутации питания между основным источником и аккумулятором применяется схема с реле. Она работает так, что при нормальной работе питание поступает от основного источника, а при его отказе – от аккумулятора. Реле управляет схема обнаружения отказа питания.

Тип и число аккумуляторов для резервного питания определяют следующие факторы:

ток, потребляемый модулями памяти;

характеристики разряда аккумулятора;

размер, масса и стоимость аккумуляторов;

максимальный временной интервал питания памяти от резервного источника.

Так как модуль памяти состоит из массива микросхем памяти и электроники обрамления, общий разрядный ток

$$\text{Разрядный ток} = \frac{N_m \times P_m + P_s}{V}.$$

где N_m – число микросхем памяти; P_m – энергия, потребляемая каждой микросхемой; P_s – энергия, потребляемая электроникой обрамления; V – напряжение питания.

Требуемый ток оказывается значительно меньше, если при отказе источника питания память переводится в пассивный режим. Потребляемую энергию можно уменьшить также, применяя в интерфейсе и схеме управления КМОП-схемы.

Емкость аккумулятора измеряется в ампер-часах при конкретном разрядном токе. Однако значение в ампер-часах уменьшается при увеличении разрядного тока. Время защиты, обеспечиваемое резервным источником питания, равно отношению ампер-часов к разрядному току при условии, что ток не превышает допустимого значения. Пусть аккумулятор имеет емкость 3,2 А · ч при разрядном токе 1 А, а модуль памяти потребляет 0,8 А. Тогда аккумулятор может питать модуль памяти минимум в течение 4 ч, а при параллельном включении трех аккумуляторов это время возрастает до 12 ч.

Желательно, чтобы при разряде выходное напряжение не изменялось. Этому критерию удовлетворяют заряжаемые и незаряжаемые аккумуляторы. К незаряжаемым относятся ртутные и серебряно-окисные аккумуляторы, имеющие достаточную емкость при малых размерах. К широко распространенным заряжаемым аккумуляторам относятся никель-кадмиевые и свинцово-кальциевые. Хотя они значительно больше и тяжелее большинства аккумуляторов, в некоторых применениях важную роль играет возможность их подзарядки от основного источника питания.

10.5. ПОСТОЯННЫЕ ЗАПОМИНАЮЩИЕ УСТРОЙСТВА

Содержимое ПЗУ после его записи изменить невозможно, по крайней мере без специального оборудования. Плотность упаковки ПЗУ выше, чем в ЗУПВ, благодаря более простой схеме запоминающего элемента (не требуется тракта записи). ПЗУ являются энергонезависимыми и имеют высокую надежность. Но, разумеется, их можно использовать только там, где не требуется запись. Как отмечалось во введении к данной главе, в большинство систем памяти входят ПЗУ и ЗУПВ, причем модули ПЗУ применяются для хранения, например, настраивающего загрузчика и таблиц фиксированных данных. Иногда в ПЗУ хранят монитор и другие компоненты операционной системы (при этом устраняется необходимость иметь настраивающий загрузчик). Наконец, в ПЗУ можно хранить даже интерпретаторы языков.

Имеются четыре основных типа ПЗУ, различающиеся способом записи их содержимого. Задание содержимого ПЗУ иногда называется программированием, но, конечно, его нельзя путать с обычным пониманием программирования. В одном из типов ПЗУ содержимое определяется операцией маскирования в процессе изготовления кристалла. Их содержимое пользователь изменить не может и именно они и называются ПЗУ. Содержимое ПЗУ второго типа пользователь может задавать сам, имея для этого специальное оборудование, — *программируемые постоянные запоминающие устройства* (*ППЗУ*). Как и в маскированных ПЗУ, их содержимое после программирования изменить нельзя. ПЗУ третьего и четвертого типов пользователь может не только программировать, но и при наличии специального оборудования стирать и

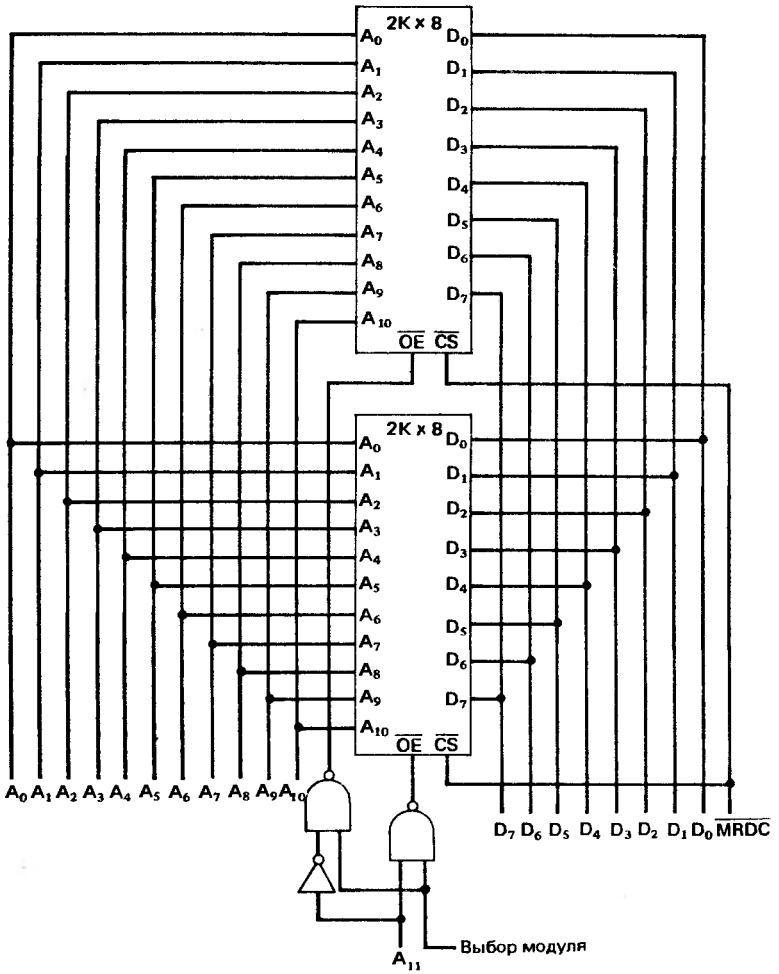


Рис. 10.16. Постоянное запоминающее устройство 4К × 8

репрограммировать много раз. Они называются *стираемыми ППЗУ* (*СППЗУ*) и *электрически изменяемыми ПЗУ* (*ЭИПЗУ*).

Так как маскированные ПЗУ требуют изготовления маски (фотошаблона) для заданного содержимого или программы, первое ПЗУ оказывается дорогим, но дополнительные копии гораздо дешевле. Микросхема ПЗУ имеет входы адреса, выходы данных и вход выбора кристалла. На рис. 10.16 показан модуль памяти 4К × 8, построенный из двух ПЗУ с организацией 2К × 8. Конечно, кроме показанных схем, в нем потребуются дешифратор выбора модуля, приемопередатчики и регистры-защелки.

Обычно ППЗУ имеют в своей основе диодную матрицу. Они программируются посредством выбора внешними входами диодных связей, которые "пережигаются" или "расплавляются", что приводит к фиксированному программированию диодной матрицы. Единицы представлены сохранившимися диодными связями, а нули – разрушенными. Так как микросхемы ПЗУ для всех заказчиков одинаковы, их стоимость не зависит от их числа, но они гораздо сложнее и дороже именно из-за программируемости. Следовательно, при небольшом числе микросхем ППЗУ дешевле маскируемых ПЗУ, но если требуется большое число ПЗУ, дешевле использовать маскируемые. Поэтому на этапе макетирования применяются ППЗУ (или СППЗУ), а в массовой продукции – маскируемые.

ППЗУ программируются бит за битом посредством задания адреса содержащей бит ячейки на входах адреса и подачи тока в выходную линию данных; при этом подаются импульсы по линии питающего напряжения и соответствующих управляющих входов. Значение тока, амплитуда и продолжительность импульсов, а также способ подачи их зависят от типа микросхемы. Обычно циклы программирования чередуются с циклами контроля и продолжительность чередования в два раза больше времени, необходимого для программирования бита.

В отличие от ППЗУ, в которых содержимое программируется раз и навсегда путем "пережигания" диодных связей, содержимое СППЗУ определяется распределением заряда. СППЗУ программируются путем инжекции заряда и после программирования распределение заряда сохраняется до тех пор, пока не будет нарушено каким-либо внешним источником энергии, например ультрафиолетовым светом. Вместо заключения кристалла в светонепроницаемый корпус, как это делается в остальных микросхемах, СППЗУ имеют над кристаллом кварцевое окно, через которое проходит внешняя энергия. При экспозиции памяти от внешнего источника энергии в течение нескольких минут (10 . . . 50 мин в зависимости от типа микросхемы) заряды перераспределяются в их естественное состояние, стирая старое содержимое памяти. После этого СППЗУ можно программировать вновь.

Как и ППЗУ, СППЗУ обычно применяют на этапе разработки изделия, а в серийной продукции заменяют маскированными ПЗУ. Благодаря возможности стирания ими можно пользоваться несколько раз. Однако в отличие от ППЗУ их содержимое может исчезать, поэтому СППЗУ не рекомендуется встраивать в изделия, рассчитанные на длительную эксплуатацию. Продолжительность сохранения содержимого зависит от условий окружающей среды и варьируется от нескольких месяцев до нескольких лет.

Программируется СППЗУ посредством подачи адреса на адресные входы, а также высоких или низких уровней напряжения на все выходы данных; после этого на входы питания и управления подаются необходимые импульсы. Например, при программировании СППЗУ 2764 с организацией 8К x 8 (фирмы Intel) на вход V_{pp} подается напряжение 21 В, а на вход $\overline{CE}$ низкий уровень напряжения. Одновременно адрес программируемого байта подается на входы A12-A0, а байт данных на контакты O7-O0. Затем байт данных записывается в адресуемый байт подачей импульса напряжения +5 В на вход

$\overline{\text{PGM}}$. После записи необходимо содержимое каждого байта проконтролировать.

Микросхемы ППЗУ и СППЗУ, выпускаемые даже одной фирмой, имеют различные спецификации программирования. Эти спецификации, особенно для ППЗУ, оказываются довольно сложными и их требуется неукоснительно соблюдать. Поэтому многие фирмы предлагают специальные приборы, называемые *программаторами ППЗУ* и предназначенные для программирования ППЗУ и СППЗУ. Чтобы удовлетворить спецификации нескольких типов микросхем, в некоторых программаторах предусмотрены схемные модули, называемые *платами персонификации*, которые формируют все электрические сигналы для программирования микросхем определенного типа. Кроме плат персонификации, в управляющий модуль программатора обычно встраивается микропроцессор. Управляющий модуль позволяет одним приказом записать блок данных из системы проектирования в ППЗУ или СППЗУ. В программаторе имеется также индикатор для контроля содержимого ППЗУ. Для универсального программатора ППЗУ фирмы Intel в операционной системе ISIS-II имеются приказы для выполнения следующих операций:

1. Загрузка программируемых данных из выбранного устройства ввода (дисковый файл, перфолента или системная консоль) в память системы проектирования.
2. Индикация или изменение данных в памяти системы проектирования.
3. Запись в сегмент ППЗУ данных, которые хранятся в памяти системы проектирования по заданному начальному адресу (т. е. собственно программирование).
4. Передача блока данных из ППЗУ в память, что позволяет просмотреть содержимое ППЗУ с системной консоли или использовать для программирования дублирующей микросхемы.
5. Передача блока данных из ППЗУ в дисковый файл.
6. Сравнение блока данных из ППЗУ с содержимым области памяти (т. е. контроль программирования).

Электрически изменяемые ППЗУ также допускают репрограммирование и обладают удобным преимуществом: возможностью отдельно стирать и репрограммировать каждый байт. К недостатку таких микросхем относится сравнительно высокая стоимость.

Упражнения

1. Заполните следующую таблицу:

Размер памяти	Тип микросхемы	Массив микросхемы	
		Число строк	Число столбцов
4К × 8	2К × 4		
64К × 8	32К × 1		
8К × 16	4К × 4		
1М × 16	64К × 1		

2. Имеется модуль памяти емкостью 32К байт с контрольными битами паритета, выполненный на микросхемах с организацией 8К × 1. Определите конфигурацию массива микросхем – число микросхем в каждой строке, в каждом столбце и во всем модуле. Повторите упражнение для микросхем с организацией 16К × 1.

3. Постройте таблицу, которая отражает основные вопросы проектирования, рассмотренные в § 10.1.

4. Постройте логическую схему, показывающую подробности реализации схемы на рис. 10.5. Условное изображение запоминающего элемента приведено на рис. 10.17.

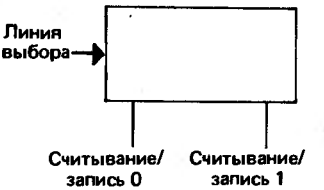


Рис. 10.17. Условное изображение запоминающего элемента

5. Покажите, как можно упростить схему на рис. 10.8, если общая емкость памяти составляет 8К × 8. Дайте возможности дальнейшего упрощения в случае одноплатной системы на базе микропроцессора 8088 в минимальном режиме.

6. Рассмотрите модуль памяти 256К × 8, построенный на динамических ЗУПВ с организацией 32К × 1, внутри которых имеется 128 строк запоминающих элементов. Если вся память регенерируется один раз в миллисекунду, каков период циклов регенерации строк? Постройте график потери времени на регенерацию в зависимости от времени цикла памяти, если оно изменяется от 100 до 800 нс.

7. Перестройте схему на рис. 10.11, считая, что модуль имеет емкость 64К байт и построен на микросхемах 32К × 1 (матрица запоминающих элементов имеет 128 строк).

8. Приведите достоинства и недостатки динамических ЗУПВ по сравнению со статическими.

9. Пусть в модуле памяти с емкостью 256К байт применяются динамические ЗУПВ 64К × 1, имеющие питание +5 В. Какой ток потребляет модуль, если каждая микросхема рассеивает 1 Вт, а электроника обрамления 2 Вт? Сколько аккумуляторов необходимо для резервного питания в течение 8 ч, если аккумулятор имеет емкость 2 А • ч при разрядном токе 2 А?

10. Повторите упр. 9, считая, что резервное питание требуется всего на 15 мин, но допустимый разрядный ток равен 0,7 А.

11. Постройте интерфейсную логику для модуля ПЗУ 4К байт, приведенного на рис. 10.16.

12. Одно из применений ПЗУ связано с заменой схем, реализующих сложные логические функции. Рассмотрите следующую булеву функцию:

$$f = X_0 X_1 \bar{X}_2 X_3 X_4 X_5 X_6 X_7 \bar{X}_8 + \bar{X}_0 \bar{X}_1 X_2 \bar{X}_3 X_4 X_5 X_6 X_7 + X_2 \bar{X}_3 \bar{X}_4 \bar{X}_5 X_6 X_7 X_8 + \\ + X_0 \bar{X}_2 X_3 X_4 X_5 X_6 \bar{X}_7 \bar{X}_8$$

Сколько основных логических элементов (инверторов, двухвходовых схем И, двухвходовых схем ИЛИ) потребуется для реализации этой функции? Определите содержимое ПЗУ 512 × 8, которое применяется для реализации этой функции, причем младший бит выходных данных служит выходом f. (*Указание.* Важен только младший бит каждого слова.) Проанализируйте задержки распространения, возникающие в каждом варианте.

13. Сколько булевых функций можно реализовать на ПЗУ 2К × 8 и каковы ограничения на входные переменные этих функций?

11. МУЛЬТИПРОЦЕССОРНЫЕ КОНФИГУРАЦИИ

В гл. 9 было показано, что контроллер ПДП может улучшить пропускную способность системы благодаря параллельному выполнению ввода-вывода и действий ЦП. Такая возможность объясняется тем, что процессор не использует все циклы шины. В зависимости от условий применения микропроцессор 8086 расходует только 50 – 80 % доступного времени шины. Контроллер ПДП может получать циклы шины для передачи данных, оказывая минимальное воздействие на операции ЦП. Кроме того, он освобождает ЦП от относительно медленных операций ввода-вывода. Такая конфигурация служит простым примером системы с более чем одним процессором, причем оба процессора работают параллельно, улучшая производительность системы.

Хотя возможности контроллера ПДП довольно ограничены, принцип параллельных операций, являющийся эффективным способом улучшения работы системы, можно распространить и на более сложные компоненты. Если в системе имеются две или более компонент, которые могут одновременно выполнять команды, она называется *мультипроцессорной системой.* Подключаемыми процессорами могут быть специализированные процессоры, рассчитанные на эффективное выполнение определенных задач, или процессоры широкого назначения. Например, из-за ограниченного формата данных и отсутствия команд, оперирующих числами с плавающей точкой, в микропроцессоре 8086 на выполнение одной операции с плавающей точкой требуется много команд. В системе, рассчитанной на сложные расчеты, целесообразно реализовать их на вспомогательном процессоре числовых данных, который специально разработан для быстрой обработки чисел с плавающей точкой и чисел с более длинными, чем обычные данные, форматами. Иногда имеет смысл применить в системе процессор ввода-вывода с бо́льшими возможностями, чем контроллер ПДП. Помимо обычных операций ПДП такой процессор может выполнять обработку цепочек, преобразование кода, поиск символов и проверку бит. В этом случае ЦП может сосредоточиться на функциях более высокого уровня.

По мере уменьшения отношения стоимость/производительность однокристальных микропроцессоров становится более экономичным применять несколько процессоров вместо одного сложного многокристального, что характерно для так называемого централизованного подхода. Кроме улучшения экономических показателей системы, мультипроцессорная конфигурация обеспечивает несколько положительных качеств, которых нет в однопроцессорной системе. Во-первых, несколько процессоров лучше приспосабливаются под требования конкретного применения, исключая расходы на ненужные возможности централизованной системы. Более того, модульность мультипроцессорной системы позволяет по мере необходимости вводить дополнительные процессоры. Во-вторых, в мультипроцессорной системе задачи разделяются между модулями. При возникновении отказа проще и дешевле найти и заменить неисправный процессор, чем отыскивать и заменять отказавший элемент в сложном процессоре.

При проектировании мультипроцессорной системы приходится решать две задачи: состязания за доступ к шине и межпроцессорные взаимодействия. Поскольку память и устройства ввода-вывода по общей системной шине разделяют несколько процессоров, потребуется дополнительная логика для обеспечения того, чтобы в любой момент времени доступ к шине имел только один процессор. Чтобы один процессор диспетчировал задачу или возвращал результат другому процессору, необходим строго определенный способ взаимодействия процессоров. Принятые решения указанных задач определяют соединения между процессорами.

Максимальный режим микропроцессоров 8086/8088 специально предназначен для реализации мультипроцессорных систем. Имеющиеся мультипроцессорные средства максимального режима рассчитаны на три базовые конфигурации: сопроцессор, сильно связанная конфигурация и слабо связанная конфигурация. Первые две конфигурации очень похожи друг на друга в том отношении, что ЦП и вспомогательный (или внешний) процессор разделяют не только всю подсистему памяти и ввода-вывода, но и логику управления шиной и генератор синхронизации (см. рис. 11.1). В обеих конфигурациях микропроцессор 8086/8088 является ведущим (главным), а вспомогательный процессор – ведомым. Управление доступом к шине осуществляет ЦП, поэтому сигнал запроса шины вспомогательного процессора подается в ЦП. В сильно связанной конфигурации вспомогательный процессор действует независимо, но, являясь сопроцессором, он должен взаимодействовать непосредственно с ЦП. При организации сопроцессора между процессорными элементами имеется больше линий. Так как микропроцессоры 8086/8088 всегда выступают ведущими, в рассматриваемых конфигурациях не может быть двух микропроцессоров 8086/8088.

Слабо связанные конфигурации применяют в средних и больших системах. Любой модуль в такой системе может быть ведущим системной шины и содержать 8086, 8088, другой процессор, который может быть ведущим ши-

Рис. 11.1. Сильно связанная конфигурация

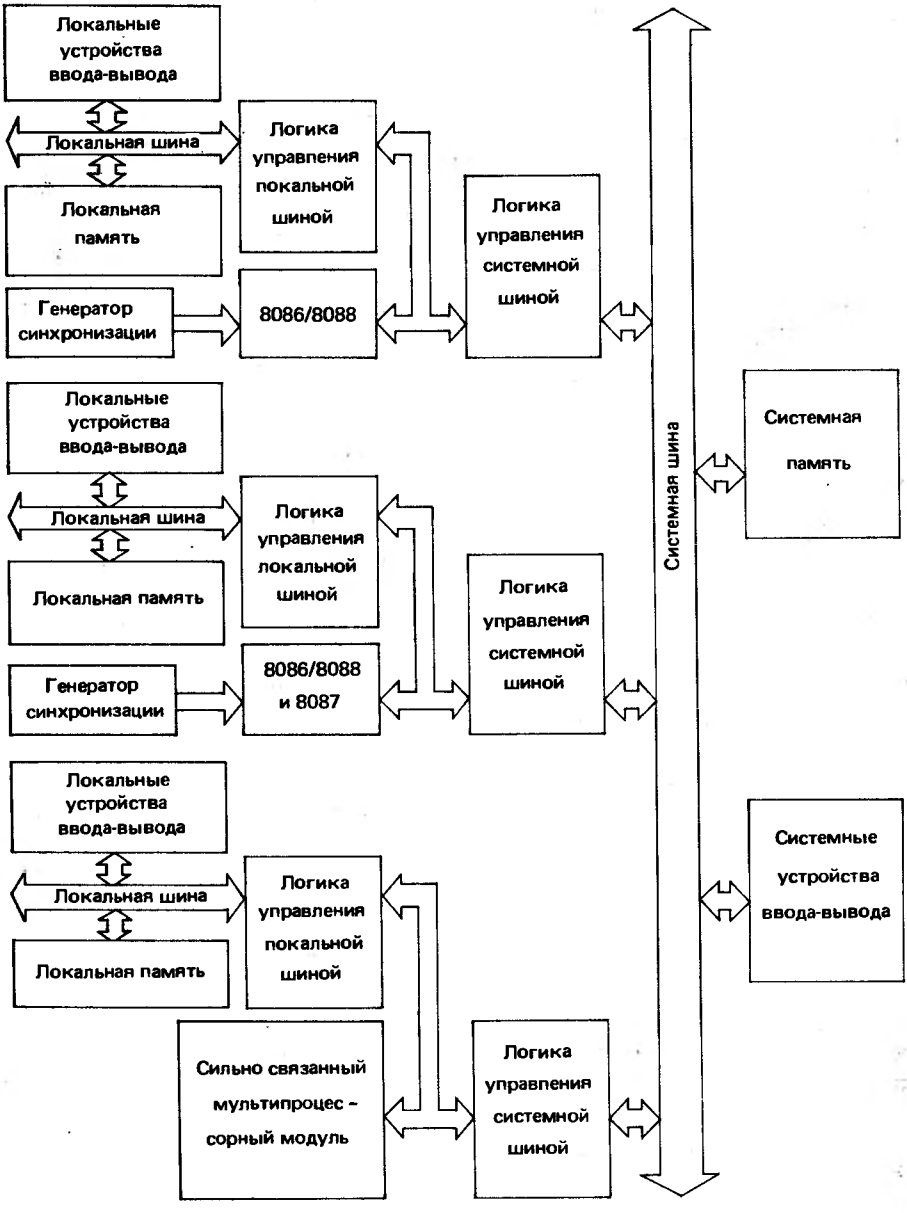


Рис. 11.2. Слабо связанная конфигурация

ны, сопроцессор или сильно связанную конфигурацию. Системные ресурсы разделяют несколько модулей, а проблему состязаний при доступе к шине должна решать логика управления системной шиной. Как показано на рис. 11.2, каждый потенциальный ведущий шины работает независимо и прямые связи между ними отсутствуют. Межпроцессорное взаимодействие осуществляется через разделенные ресурсы. Кроме них у каждого модуля могут быть свои память и устройства ввода-вывода. Процессоры в отдельных модулях могут одновременно обращаться к своим локальным подсистемам по локальным шинам и выполнять независимо друг от друга выборки команд и обращения к локальным данным, что повышает степень параллельности обработки.

В настоящей главе сначала рассматриваются мультипроцессорные возможности микропроцессоров 8086/8088, включая специальные команды и управляющие сигналы; § 11.2 посвящен реализации различных мультипроцессорных конфигураций. В § 11.3, 11.4 описаны процессор числовых данных 8087 и процессор ввода-вывода 8089 фирмы Intel.

11.1. СОСТОЯНИЕ ОЧЕРЕДИ И БЛОКИРОВКА ШИНЫ

Мы пока не рассматривали мультипроцессорные возможности максимального режима и контроллера шины 8288. Обратимся к мультипроцессорным ситуациям, хотя их детальное обсуждение будет проведено далее.

Наличие очередей команд в микропроцессорах 8086/8088 приводит к тому, что выбираемая команда может выполняться не сразу же. Чтобы внешняя логика могла контролировать последовательность выполнения команд, микропроцессоры 8086/8088 в максимальном режиме выдают на выходы QS1 и QS0 состояние очереди. Оно анализируется в каждом такте синхронизации и биты QS0 и QS1 кодируются следующим образом:

00 – из очереди команды не выбиралась,
01 – из очереди выбран первый байт текущей команды,
10 – содержимое очереди использовать нельзя из-за передачи управления,
11 – из очереди выбран байт, отличающийся от первого байта команды.

Контролируя шину и состояние очереди, внешняя логика может моделировать последовательность выполнения команд процессором и определять, какая команда выполняется в данный момент времени.

Как указывалось в гл. 7, в мультипрограммной среде необходимо управлять обращениями к разделенному ресурсу. Обычно для обеспечения того, что в любой момент времени только один процесс может войти в критическую секцию кода с обращением к разделенному ресурсу, применяются семафоры. Напомним реализацию семафора:

```
            MOV    AL,0
TRYAGAIN:   XCHG   SEMAPHORE, AL
            TEST   AL,AL
            JZ     TRYAGAIN
              .  }
              .  }  Критическая секция, в которой процесс
              .  }  обращается к разделенному ресурсу
              .  }
            MOV    SEMAPHORE,1
```

Такая реализация прекрасно работает в системе, где все процессы выполняет один и тот же процессор, так как он не может переключиться с одного процесса на другой в середине команды. Однако, если конкурирующие процессы выполняются разными процессорами, ситуация усложняется.

Предположим, что процессор A готов модифицировать запись в памяти, а процессор B в это же время готов сортировать эту же запись. Так как процессоры работают независимо, они проверяют семафор одновременно. Отметим, что команда XCHG требует двух циклов шины, в одном из которых в ЦП вводится старый семафор, а во втором выводится новый. Возможно, что после того, как процессор A считал семафор, процессор B получит управление следующим циклом шины и считает тот же самый семафор.

Предположим, что ячейка SEMAPHORE содержит 1 и оба процессора A и B выполняют команду

```
TRYAGAIN: XCHG SEMAPHORE,AL
```

Предположим также, что далее имеют место следующие действия:

1. Процессор A использует первый доступный цикл шины для считывания содержимого SEMAPHORE.

2. Процессор B использует следующий цикл шины для получения содержимого SEMAPHORE.

3. Процессор A в следующем цикле шины сбрасывает SEMAPHORE, завершая этим выполнение команды XCHG.

4. Процессор B в следующем цикле шины сбрасывает SEMAPHORE и также заканчивает команду XCHG.

После этих действий в регистрах AL обоих процессоров будет 1 и команда TEST AL,AL установит ZF = 0, условие в команде JZ не удовлетворяется и оба процессора входят в критические секции кода.

Чтобы решить эту задачу, процессор, первым начавший выполнение команды XCHG (в приведенном примере — это процессор A), должен монопольно использовать шину до завершения команды XCHG. В микропроцессорах 8086/8088 такое исключительное использование шины гарантируется префиксом блокировки LOCK:

1 1 1 1 0 0 0 0

Этот префикс вызывает появление активного выходного сигнала $\overline{\text{LOCK}}$ при выполнении следующей за префиксом команды. Сигнал $\overline{\text{LOCK}}$ уведомляет логику управления шиной о том, что другие процессоры не должны по-

лучать управления шиной до окончания заблокированной команды. Следовательно, в приведенном примере команде XCHG следует предпослать префикс LOCK:

```
TRYAGAIN: LOCK XCHG SEMAPHORE,AL
```

Наличие префикса гарантирует, что обмен будет осуществлен в двух последовательных циклах шины.

Физически в слабо связанной системе каждый процессорный модуль имеет схему арбитра шины и все арбитры соединены специальными линиями управления системной шины. Одной из них является линия занятости, несущая активный сигнал, когда шина используется каким-либо модулем. Когда арбитр процессорного модуля получает управление шиной, он формирует активный сигнал на линии занятости, что предотвращает передачу шины другим арбитрам до следующего цикла шины. Если в управляющий шиной арбитр подан сигнал $\overline{\text{LOCK}}$, этот арбитр сохраняет управление системной шиной, поддерживая активный сигнал на линии занятости до снятия сигнала $\overline{\text{LOCK}}$. Следовательно, если процессор выдает сигнал $\overline{\text{LOCK}}$ во время выполнения команды, его арбитр не освобождает системную шину до завершения команды. Подробнее об арбитрах шины речь идет в разделе 11.2.3, посвященном слабо связанным конфигурациям.

Если в модуле, содержащем сопроцессор или сильно связанную конфигурацию, при активном сигнале $\overline{\text{LOCK}}$ сделан запрос шины по линиям $\overline{\text{RQ}}/\text{GT}$ микропроцессоров 8086/8088, запрос не удовлетворяется до снятия сигнала $\overline{\text{LOCK}}$. После этого микропроцессор реагирует на запрос и возвращает разрешение на использование шины. Кроме инициирования префиксом LOCK сигнал $\overline{\text{LOCK}}$ выдается в ответ на запрос прерывания $\overline{\text{INTR}}$ от начала первого импульса $\overline{\text{INTA}}$ до окончания второго импульса $\overline{\text{INTA}}$. Этим гарантируется управление шиной до завершения цикла прерывания.

Блокировка шины применяется также для быстрого выполнения команд, которые требуют несколько циклов шины. Например, в мультипроцессорной системе можно передать блок данных с большей скоростью, пользуясь префиксом LOCK следующим образом:

```
LOCK REP MOVS DEST,SRC
```

При выполнении этой команды системная шина резервируется для монопольного использования процессором, выполняющим команду.

Префикс LOCK считается расширением следующей за ним команды; следовательно, прерывания, возникающие при выполнении команды с префиксом LOCK, не подтверждаются до окончания всей команды. Однако, если префикс LOCK объединен с префиксом REP, как в приведенном примере, прерывания подтверждаются по окончании каждой операции пересылки. Более того, при возврате из прерывания восстанавливается только префикс, непосредственно предшествующий команде MOVS. Поэтому рекомендуется запрещать прерывания до выполнения комбинации LOCK REP.

Контроллер шины 8288 также обеспечивает управляющие функции для поддержки слабо связанных систем, в которых он применяется вместе

с арбитром шины. Эти функции реализуются сигналами $\overline{\text{AEN}}$, $\overline{\text{IOB}}$, CEN и MCE/$\overline{\text{PDEN}}$, которые подробно рассматриваются в разделе 11.2.3. В предыдущих разделах указанные сигналы были подключены для использования контроллера шины в системах с одним ЦП.

11.2. МУЛЬТИПРОЦЕССОРНЫЕ СИСТЕМЫ НА БАЗЕ МИКРОПРОЦЕССОРОВ 8086/8088

Рассмотрим три основные мультипроцессорные конфигурации, которые поддерживают микропроцессоры 8086/8088. В настоящем параграфе показано их применение в качестве ведущих процессоров, а в последующих разделах приведены конкретные примеры, касающиеся других процессоров фирмы Intel.

11.2.1. СОПРОЦЕССОРНЫЕ КОНФИГУРАЦИИ

Хотя микропроцессоры 8086/8088 являются мощными однокристальными процессорами, их системы команд недостаточно для эффективной реализации некоторых сложных применений. В этих случаях микропроцессоры 8086/8088 необходимо дополнить сопроцессорами, которые расширяют систему команд в направлениях, обеспечивающих более эффективную реализацию требуемых специальных вычислений. Например, микропроцессоры 8086/8088 не имеют команд арифметики с плавающей точкой, а при использовании в качестве сопроцессора процессора числовых данных 8087 легко реализуются требования вычислений с плавающей точкой.

Система с сопроцессором не требует никакой дополнительной логики, отличающейся от той, которая требуется в системе с максимальным режимом. ЦП и сопроцессор выполняют свои команды из одной и той же программы,

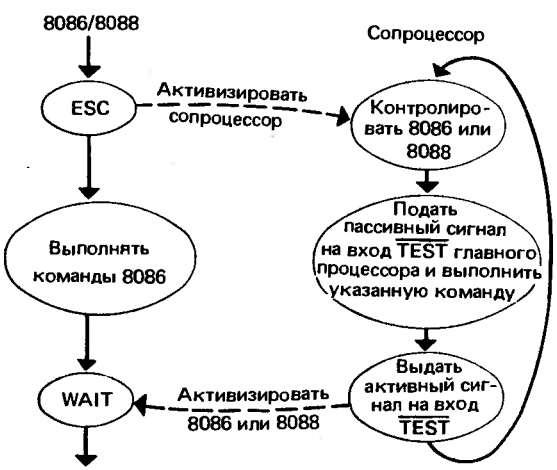


Рис. 11.3. Синхронизация микропроцессора 8086 с его сопроцессором

которая написана в надмножестве системы команд микропроцессоров 8086/8088. Если команда выполняется сопроцессором, ЦП кроме возможного считывания операнда для сопроцессора не предпринимает более никаких действий.

Взаимодействие между ЦП и сопроцессором, когда команда выполняется сопроцессором, показано на рис. 11.3. Предназначенная для сопроцессора команда определяется появлением в программе команды ESC. Выбирать команды может только главный ЦП, но сопроцессор также получает все команды и контролирует выполнение команд главным ЦП. Команда ESC содержит код внешней операции, показывающий операцию сопроцессора, и одновременно дешифрируется сопроцессором и главным ЦП. В этой точке ЦП может просто перейти к следующей команде или считать первое слово из памяти как операнд для сопроцессора, а затем перейти к следующей команде. Если ЦП считывает первое слово операнда, сопроцессор перехватывает слово данных и его 20-битный физический адрес. Когда операнд-источник длиннее одного слова, сопроцессор получает остальные слова посредством запросов циклов шины. Если же определенный в команде ESC операнд является получателем, сопроцессор игнорирует считанное ЦП слово данных, а позднее запоминает результат по перехваченному адресу. В любом случае сопроцессор выдает сигнал занятости (высокий уровень) на вход $\overline{\text{TEST}}$ ЦП и пока ЦП продолжает выполнение программы сопроцессор выполняет указанную кодом в команде ESC операцию. Такая параллельная работа продолжается до тех пор, пока ЦП не понадобится сопроцессор для выполнения другой операции или ЦП не потребуется получить результат текущей операции. При этом ЦП должен выполнять команду WAIT и ожидать, пока сопроцессор не выдаст активный сигнал на вход $\overline{\text{TEST}}$. Команда WAIT периодически проверяет сигнал $\overline{\text{TEST}}$ и, когда он становится активным, осуществляет передачу управления находящейся за ней команде.

Ассемблерная команда ESC имеет два операнда. Первый из них показывает код внешней операции, определяющий действия сопроцессора. Если второй операнд определяет ячейку памяти, ЦП считывает слово из этой ячейки для сопроцессора и может передать сопроцессору адрес для запоминания результата. Если вторым операндом является регистр, адрес регистра считается частью кода внешней операции и ЦП не делает ничего.

Мнемоника команды ESC имеет два формата, показанные на рис. 11.4. В обоих случаях первый байт содержит 11011, а младшие три бита образуют код внешней операции. В формате, представленном на рис. 11.4, *а*, второй байт определяет режим адресации памяти и еще 3 бита кода внешней операции, что обеспечивает до 64 кодов внешних операций. Если режим адресации требует смещения, оно размещается в дополнительных одном или двух байтах. Второй байт формата, представленного на рис. 11.4, *б*, содержит в старших битах комбинацию 11 с последующими 6 битами кода внешней операции. Всего в этом формате код внешней операции занимает 9 бит, что обеспечивает 512 кодов внешних операций.

Интерфейс сопроцессора с главным ЦП показан на рис. 11.5. Как видно, оба процессора имеют общие генератор синхронизации и логику управления ши-

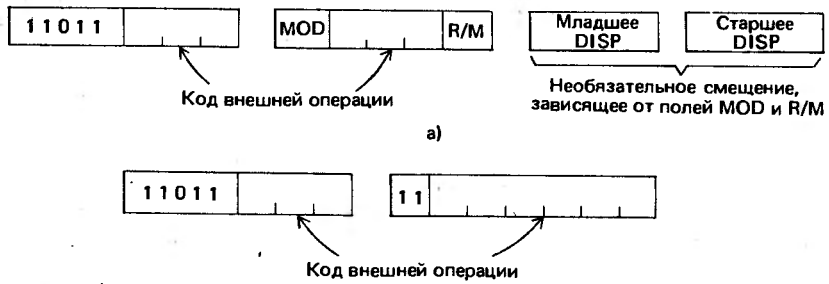


Рис. 11.4. Форматы машинного кода команды ESC, когда операнд находится в памяти (*а*) и не находится (*б*)

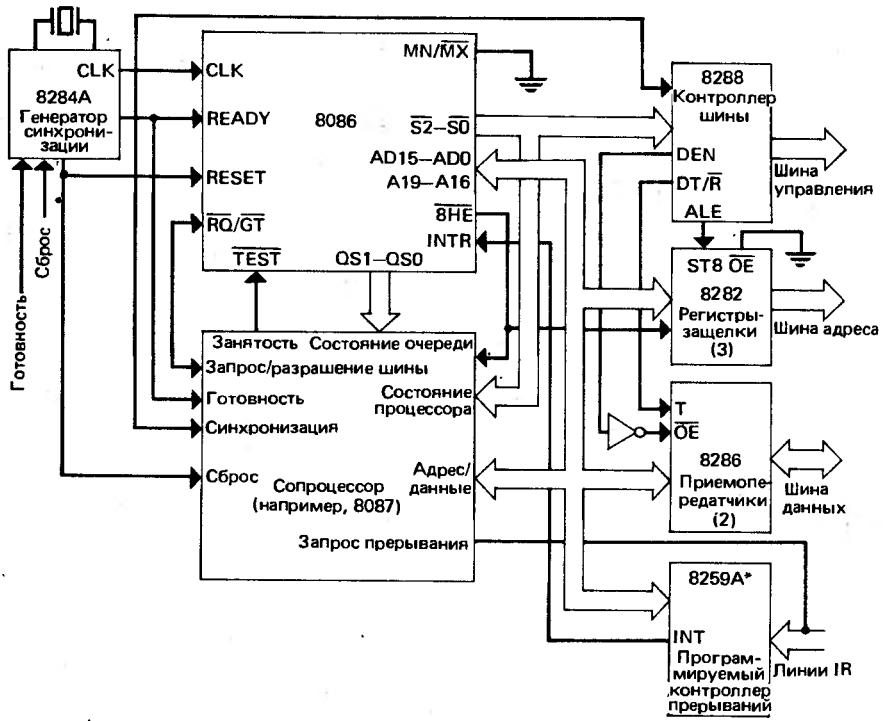


Рис. 11.5. Сопроцессорная конфигурация
* Другие подключения 8259A в соответствии с гл. 8

ной. К одному ЦП допускается подключать два сопроцессора; при этом сопроцессорам необходимо назначать подмножества множества кодов внешних операций и каждый сопроцессор должен распознавать и выполнять коды операций своего подмножества. Подключения сопроцессоров к ЦП большей частью

осуществляются по параллельным линиям, но один из них подсоединяется к линии $\overline{RQ/GT0}$ микропроцессоров 8086/8088, а другой – к линии $\overline{RQ/GT1}$. Кроме того, сопроцессоры подключаются к различным входам запросов прерываний контроллера 8259A.

Чтобы сопроцессор мог определить, когда главный ЦП выполняет команду ESC, он должен контролировать состояние ЦП по линиям $\overline{S2}$-$\overline{S0}$ и выборку команды по линиям AD15-AD0. Так как ЦП выбирает команды с опережением, выборка команды ESC не означает ее немедленного выполнения, а при наличии перед ней команды перехода она может не выполняться совсем. Сопроцессор должен следить за командным потоком, контролируя биты состояния очереди QS1 и QS0 и управляя своей очередью команд, идентичной очереди ЦП. Если состояние очереди равно 00, сопроцессор не делает ничего, но если оно равно 01, он должен сравнивать пять старших бит первого байта в очереди с кодом 11011. При наличии соответствия к выполнению готова команда ESC и сопроцессор (считая, что он распознал код внешней операции) выполняет указанную операцию; в противном случае байт игнорируется и удаляется из очереди. Состояние очереди 10 означает, что очередь в ЦП нарушается и сопроцессор также должен очистить свою очередь. Состояние 11 означает, что первый байт в очереди не является первым байтом команды и этот байт сопроцессор анализирует, если только он является частью команды ESC. Когда же он не является частью команды ESC, этот байт игнорируется.

При возникновении ошибки в процессе дешифрирования и выполнения команды ESC сопроцессор должен формировать запрос прерывания (обычно он подается в контроллер 8259A). Кроме того, сопроцессор должен запрашивать шину по одной из линий $\overline{RQ/GT}$ главного процессора, когда ему требуется считывать из памяти или записывать в память дополнительные данные. Наконец, находясь в состоянии занятости сопроцессор должен выдавать высокий уровень на вход $\overline{TEST}$ главного процессора.

Сопроцессор должен знать, с чем он работает: с микропроцессором 8086 или с микропроцессором 8088, так как они имеют различные шины данных и очереди разной длины. Тип главного процессора можно определить после каждого включения системы, если заставить сопроцессор проверять уровень сигнала на контакте 34 сразу после сигнала RESET. На этот контакт микропроцессор 8088 в максимальном режиме всегда выдает высокий уровень, а в микропроцессоре 8086 на этот контакт выведен сигнал $\overline{BHE}$/S7. Напомним, что после сброса первая команда выбирается по адресу FFFF0. Следовательно, первый адрес на шине всегда четный и первоначально сигнал $\overline{BHE} = 0$. Чтобы сопроцессор знал, когда ему проверять сигнал на контакте 34, сопроцессор и главный процессор должны иметь общую линию сброса.

Когда к ЦП подключены сопроцессор и независимый процессор, который выбирает свои команды, сопроцессор должен определять, выбирается команда независимым процессором или ЦП; в противном случае сопроцессор может ошибочно модифицировать свою очередь команд. Для этого сопроцессор контролирует бит состояния S6 – микропроцессоры 8086/8088 всегда выводят низкий уровень, а процессор ввода-вывода 8089 – высокий уровень.

Пока в качестве сопроцессора применяют только процессор числовых данных 8087, но для каждого применения можно спроектировать специализированный сопроцессор. Разумеется, любой такой сопроцессор должен быть совместим с микропроцессорами 8086/8088, работающими в максимальном режиме. Он должен воспринимать состояния ЦП и очереди команд, формировать запросы шины и прерываний, иметь сигналы сброса и готовности, получать разрешения шины, управлять очередью команд и дешифрировать коды внешних операций в командах ESC.

11.2.2. СИЛЬНО СВЯЗАННЫЕ КОНФИГУРАЦИИ

Микропроцессоры 8086/8088 поддерживают еще один тип внешнего процессора, называемого независимым, который в отличие от сопроцессора выполняет свой командный поток. Для уменьшения стоимости разработки независимый процессор можно подключить к ЦП с образованием сильно свя-

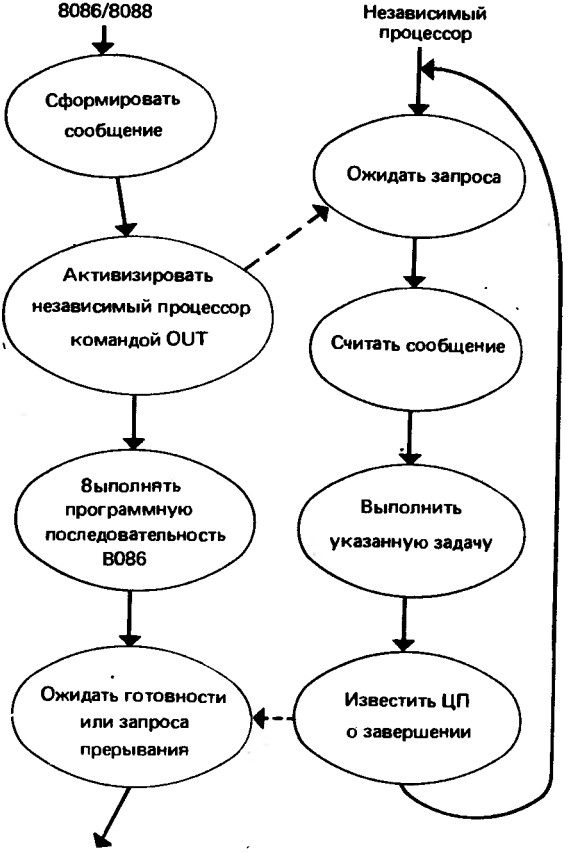


Рис. 11.6. Межпроцессорное взаимодействие через разделенную память

занной мультипроцессорной системы, в которой оба процессора разделяют генератор синхронизации и логику управления шиной. Для выборки команд и обращений к данным независимый процессор запрашивает шину по линиям $\overline{RQ}/\overline{GT}$ главного процессора.

Вместо специальных команд ESC и WAIT взаимодействие между ЦП и независимым процессором осуществляется через разделенное пространство памяти. Как показано на рис. 11.6, главный процессор формирует сообщение в памяти и активизирует независимый процессор, посылая приказ в один из его портов. Затем независимый процессор обращается к разделенной памяти, получает оттуда предназначенную ему задачу и выполняет ее параллельно с ЦП. После завершения задачи независимый процессор извещает об этом ЦП с помощью бита состояния или запроса прерывания. Формат сообщения обусловлен видом независимого процессора и особенностями применения. Обычно сообщение должно определять выполняемую задачу, входные параметры и адреса ячеек для запоминания результатов. Примером независимого процессора служит процессор ввода-вывода 8089; формат его сообщения рассмотрен в § 11.4.

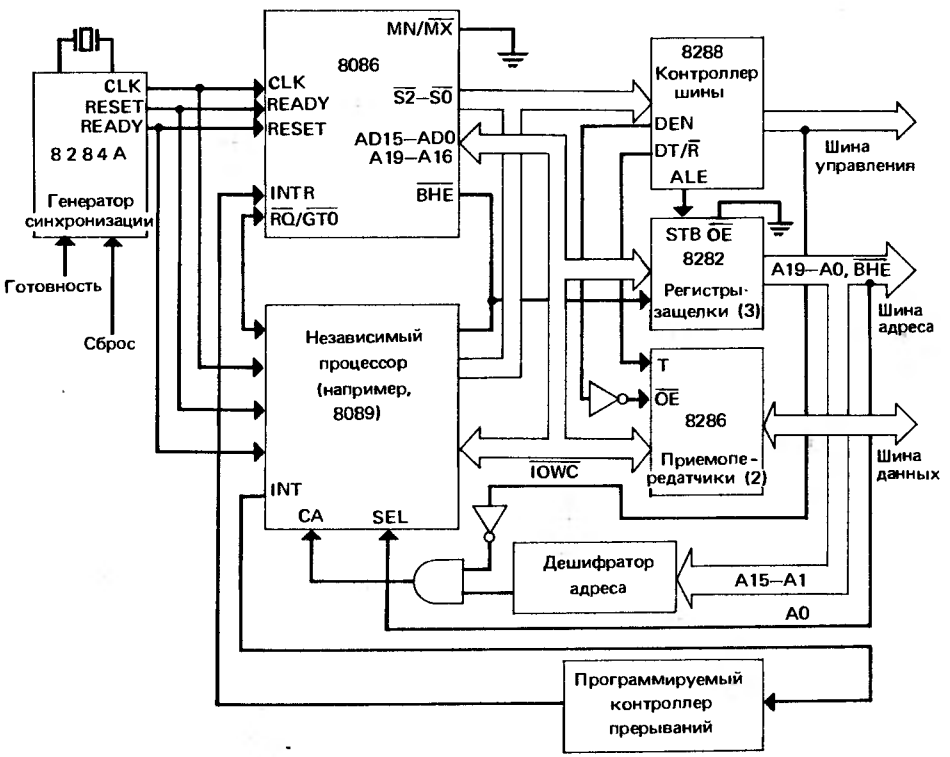


Рис. 11.7. Подключения микропроцессора 8086 в сильно связанной конфигурации

Типичное подключение микропроцессора 8086 и независимого процессора с образованием сильно связанной конфигурации показано на рис. 11.7. Так как независимый процессор выполняет свою программу, он не контролирует биты состояния очереди команд главного процессора, но запрашивает шину на линии $\overline{RQ}/\overline{GT}$. Когда один процессор использует шину, другой переводит свои выходы состояния и адреса/данных в высокоимпедансное состояние, логически отключаясь от шины. Для активизации независимого процессора главный выполняет команду вывода OUT в порт, назначенный независимому процессору. По завершении своей задачи независимый процессор устанавливает бит состояния в разделенном пространстве памяти, но может также сформировать сигнал прерывания. Так как микропроцессоры 8086/8088 имеют две линии $\overline{RQ}/\overline{GT}$, к главному процессору можно подключить более

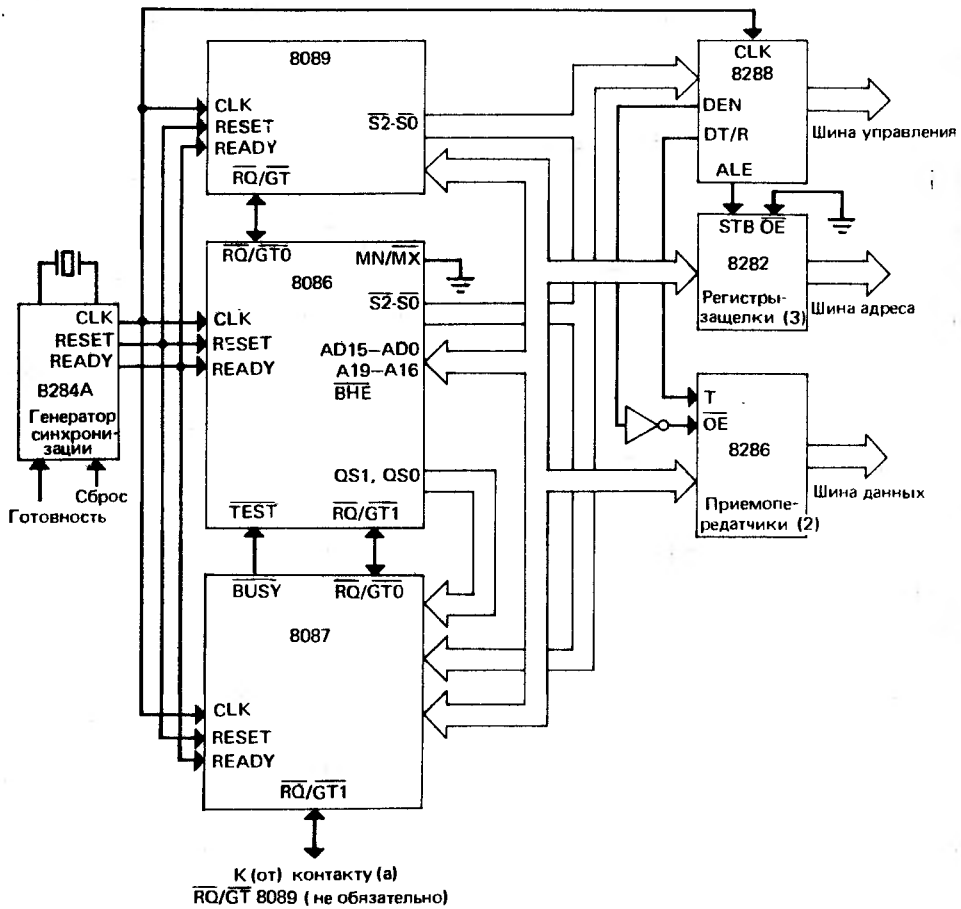


Рис. 11.8. Конфигурация с независимым процессором и сопроцессором

одного независимого процессора и (или) сопроцессора, образуя дешевую мультипроцессорную систему. Обычно сильно связанные конфигурации образуют малые и средние мультипроцессорные системы. На рис. 11.8 показано, как можно объединить сопроцессор и сильно связанную конфигурацию. В данном примере процессоры 8087, 8089 и главный процессор 8086, который выполняет функцию арбитража шины, вместе используют логику управления шиной. Так как линия $\overline{RQ}/GT0$ подключена к процессору ввода-вывода 8089, при одновременных запросах шины приоритет отдается именно ему. Но если шину использует сопроцессор 8087, запрос 8089 не подтверждается до освобождения шины. Для уменьшения максимального времени ожидания линию $\overline{RQ}/GT$ процессора 8089 можно подключить к линии $\overline{RQ}/GT1$ сопроцессора 8087. В этом случае запрос шины от 8089 заставит 8087 освободить шину после текущего цикла шины, даже если ему для завершения текущей команды требуются еще несколько циклов шины.

11.2.3. СЛАБО СВЯЗАННЫЕ КОНФИГУРАЦИИ

В мультипроцессорной системе невозможно непосредственно подключить два микропроцессора 8086 или 8088. В слабо связанной мультипроцессорной системе каждый ЦП имеет свою логику управления шиной, а арбитраж шины достигается путем расширения этой логики и введения общей для всех ведущих модулей внешней логики. Следовательно, несколько ЦП могут образовать очень большую систему, а к каждому из них можно подключить независимые процессоры и (или) сопроцессор. Слабо связанная конфигурация обладает следующими достоинствами:

при наличии нескольких ЦП повышается пропускная способность системы;

система допускает модульное расширение. Каждый ведущий модуль является независимым устройством и обычно размещается на отдельной печатной плате. Следовательно, такие модули можно добавлять или удалять, не влияя на остальные модули в системе;

отказ в одном модуле обычно не вызывает простоя всей системы, а отказавший модуль можно легко найти и заменить;

каждый ведущий шины может иметь локальную шину для доступа к собственным памяти или устройствам ввода-вывода, чем достигается более высокая степень параллельной обработки.

В слабо связанной мультипроцессорной системе доступ к разделенной системной шине осуществляют несколько ведущих модулей. Поскольку каждый из них работает независимо, для разрешения задачи арбитража шины потребуется дополнительная логика управления шиной. Она называется *логикой доступа к шине* и должна обеспечивать, чтобы в любой момент времени шиной управлял только один ведущий шины. Одновременные запросы шины учитываются на приоритетной основе. Имеется три способа задания приоритетов: приоритетная цепочка, опрос (полинг) и независимое запрашивание. Общие принципы, лежащие в основе этих способов, представлены на рис. 11.9. Реализация логики управления может варьироваться в зависимости от сложности логики доступа к шине, имеющейся в каждом модуле.

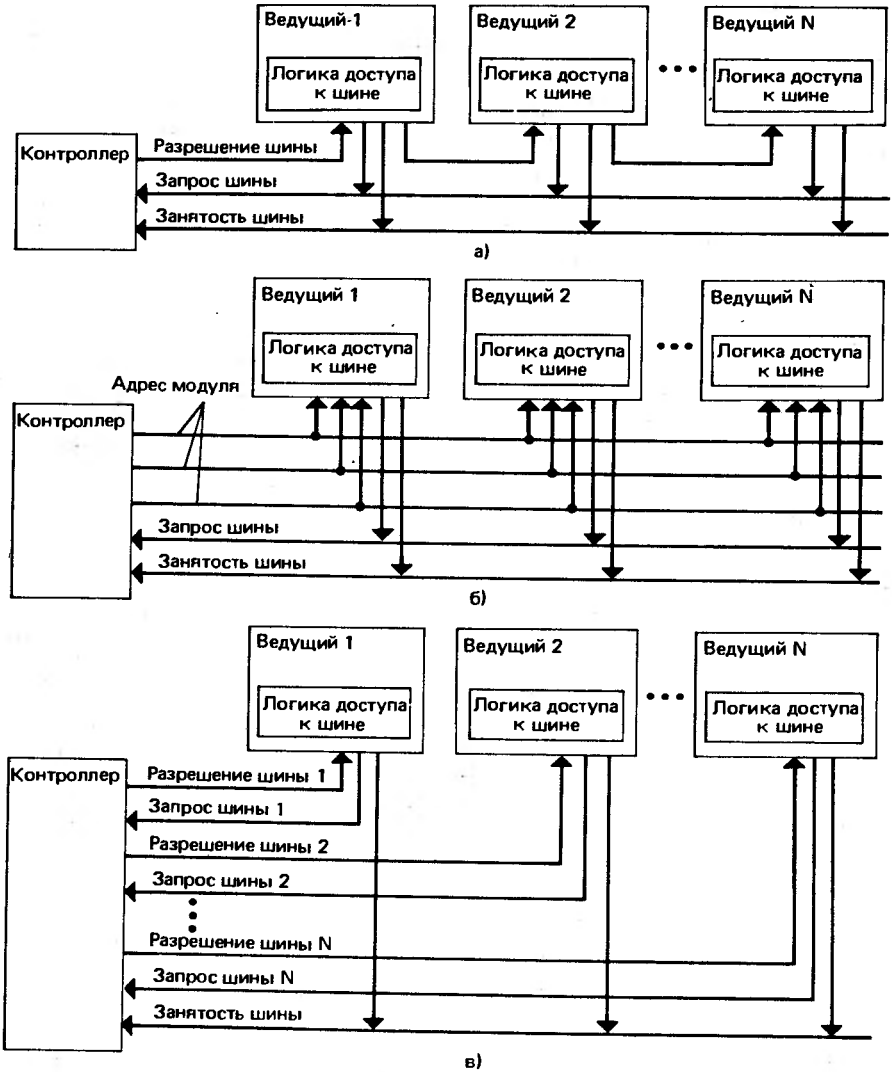


Рис. 11.9. Способы распределения шины: приоритетной цепочки (*а*), опроса (*б*) и независимых запросов (*в*)

Для способа приоритетной цепочки характерны простота и дешевизна. Запросы шины всех ведущих модулей осуществляются по одной и той же линии. Реагируя на сигнал запроса шины, контроллер выдает сигнал разрешения шины, если сигнал занятости шины пассивен. Сигнал разрешения последовательно проходит через ведущие модули до тех пор, пока не встречается

первый модуль, запрашивающий доступ к шине. Этот модуль блокирует распространение сигнала разрешения шины, формирует активный сигнал занятости шины и получает управление шиной. Следовательно, ни один другой из запрашивающих модулей не получит сигнала разрешения и приоритет определяется физическим размещением модулей. Ближайший к контроллеру модуль имеет наивысший приоритет.

По сравнению с двумя остальными способами приоритетная цепочка требует минимального числа линий управления, которое не зависит от числа модулей в системе. Однако время арбитража оказывается значительным из-за задержки распространения сигнала разрешения шины. Эта задержка прямо пропорциональна числу модулей, поэтому в системе с приоритетной цепочкой может быть небольшое число модулей. Более того, приоритет каждого модуля фиксирован его физическим размещением и отказ модуля приводит к выходу из строя всей системы.

В способе опроса имеется набор линий, достаточный для адресации каждого модуля. В ответ на запрос шины контроллер генерирует последовательность адресов модулей. Когда запрашивающий модуль распознает свой адрес, он формирует активный сигнал занятости шины и начинает использовать шину. Основное достоинство опроса заключается в том, что приоритеты можно динамически изменять, модифицируя хранящуюся в контроллере последовательность опроса.

В способе независимых запросов приоритеты учитывают параллельно. Каждый модуль имеет отдельную пару линий запроса шины и разрешения шины и каждой паре назначен свой приоритет. Находящийся в контроллере дешифратор приоритетов выбирает запрос с максимальным приоритетом и возвращает соответствующий сигнал разрешения шины. Арбитраж реализуется очень быстро и не зависит от числа модулей в системе. Способ независимых запросов имеет максимальное быстродействие из всех рассмотренных способов; однако в нем требуется больше линий запроса шины и разрешения шины ($2n$ линий для n модулей).

Главный в модуле микропроцессор 8086 или 8088 не имеет средств запроса доступа к шине и распознавания разрешения шины. Следовательно, как уже отмечалось, в каждом модуле, содержащем ведущего шины, необходима дополнительная логика выдачи и приема сигналов доступа к шине. Арбитр шины 8289 фирмы Intel специально разработан для реализации необходимого квитирования доступа к шине. Работая совместно с контроллером шины, арбитр шины управляет доступом к шине своего ведущего, пользуясь способами приоритетной цепочки или независимых приоритетов.

На рис. 11.10 показаны соединения между ведущим шины и арбитром шины в предположении, что все устройства ввода-вывода и память являются глобальными и разделяются ведущими шины через системную шину. Другие ситуации, когда ведущий шины имеет доступ к своим локальным устройствам ввода-вывода и памяти по локальной шине, рассматриваются позже.

Как показано на рисунке, арбитр шины вводит и контролирует биты состояния $\overline{S2}$-$\overline{S0}$ ЦП, чтобы определить, когда запрашивать и освобождать ши-

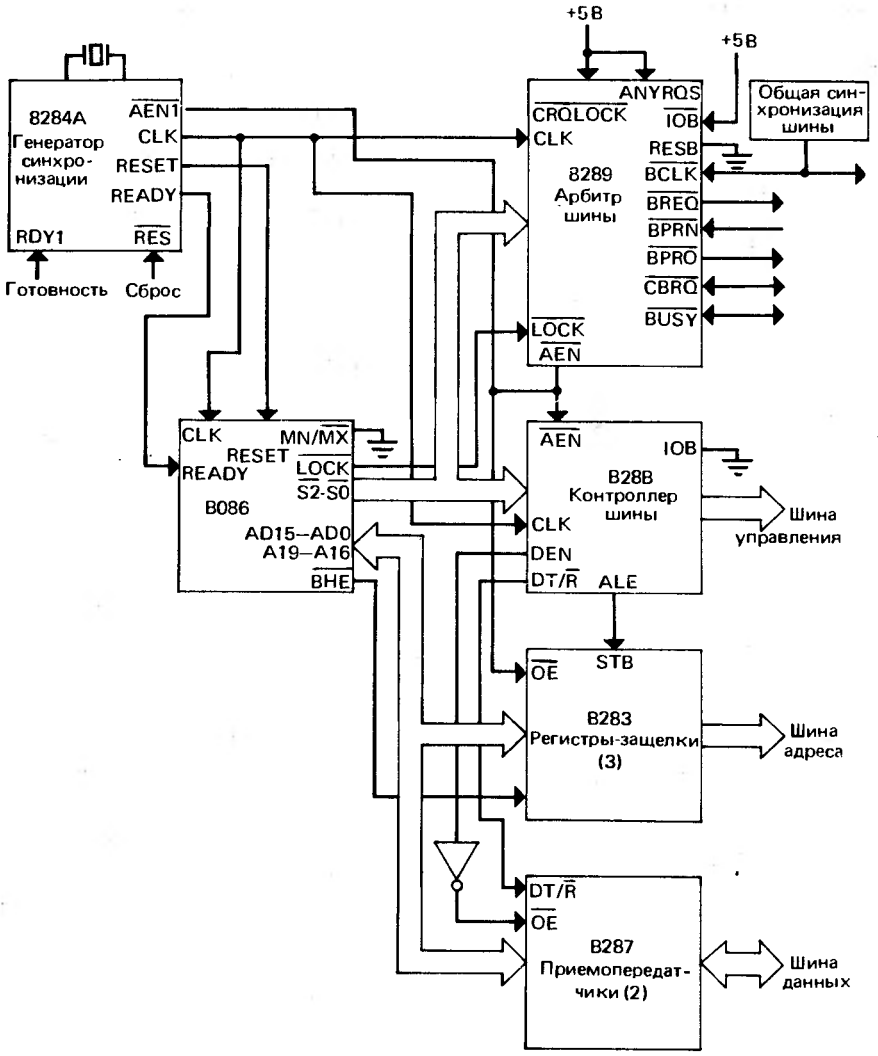


Рис. 11.10. Однопроцессорный модуль

ну. Контроллер шины 8288 также контролирует состояние ЦП, чтобы обнаружить начало цикла шины. После того как ЦП инициирует цикл шины, сигнал на линии ALE фиксирует адрес в регистрах-защелках 8283. Если арбитр шины в данное время управляет шиной, он разрешает выходы контроллера шины 8288 и регистров-защелок адреса 8283, а выход DEN контроллера шины разрешает приемопередатчики данных. После этого цикл шины развивается обычным образом. Если же арбитр шины в данный момент не управляет ши-

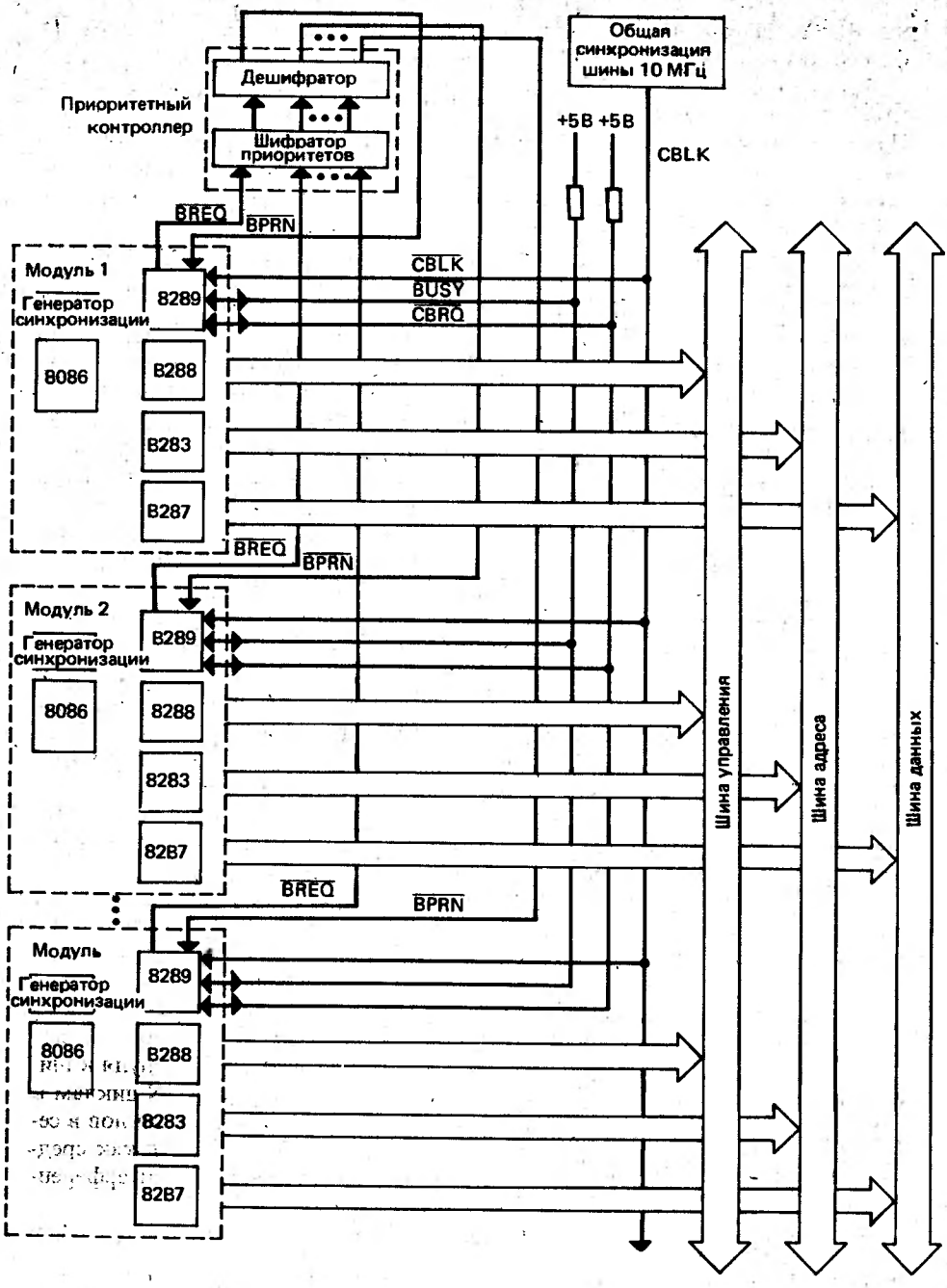
ной, он формирует выходной сигнал $\overline{AEN} = 1$, который переводит выходы приказов контроллера шины 8288 и выходы адреса регистров 8283 в высокоимпедансное состояние. Так как сигнал $\overline{AEN}$ управляет и выходом DEN контроллера шины, приемопередатчики данных запрещаются. Кроме того, сигнал $\overline{AEN} = 1$ запрещает генератору синхронизации 8284A выдавать сигнал готовности в ЦП, поэтому ЦП вводит состояние ожидания сразу же после такта T_3 текущего цикла.

Когда арбитру шины разрешен доступ к шине, он формирует активные сигналы $\overline{BUSY}$ и $\overline{AEN}$. Сигнал $\overline{AEN}$ помещает на системную шину адрес и (после задержки на время установления) разрешает сигналы приказов; кроме того, посредством линии DEN он разрешает приемопередатчики данных. После завершения передачи данных адресуемая ячейка по линии готовности возвращает сигнал подтверждения, который заставляет генератор синхронизации 8284A выдать сигнал READY. При восприятии этого сигнала ЦП выходит из состояния ожидания и завершает текущий цикл шины.

Выход блокировки $\overline{LOCK}$ ЦП непосредственно подключен к арбитру шины. Низкий уровень этого сигнала не разрешает арбитру освобождать шину. Контакты RESB (режим резидентной шины) и $\overline{IOB}$ (режим шины ввода-вывода) подключаются к постоянным уровням, образуя конфигурации, в которых процессор может обращаться к локальным устройствам ввода-вывода и памяти. Так как на рис. 11.10 предполагается наличие только разделенных ресурсов, оба эти режима запрещены.

Четыре остальные линии арбитра шины предназначены для сигналов запроса, разрешения и освобождения шины. Они рассчитаны на реализацию способов приоритетной цепочки (последовательный) и независимых запросов (параллельный). На рис. 11.11 показано, как подключаются арбитры шины для реализации способа независимых запросов. В использующей данный способ системе линия $\overline{BREQ}$ (запрос шины) от каждого арбитра подается в приоритетный контроллер. Шифратор приоритетов образует двоичное число, соответствующее входу запроса шины с наибольшим приоритетом. Этот номер дешифрируется и в выбранный арбитр шины на его вход $\overline{BPRN}$ (вход приоритета шины) подается сигнал разрешения шины. Так как синхронизация в каждом модуле осуществляется от независимого генератора, необходимо иметь отдельный общий генератор для синхронизации запросов шины.

Линии $\overline{CBRQ}$ (общий запрос шины) и $\overline{BUSY}$ (занятость) предназначены для того, чтобы текущий ведущий шины мог освободить шину. Обе они являются двунаправленными и имеют выход типа открытый коллектор. Их подключения также показаны на рис. 11.11. Когда модулю необходимо часто обращаться к системной шине, его арбитру лучше сохранять управление шиной (за исключением ситуации, когда его процессор пассивен) до тех пор, пока шину не потребует другой модуль. В противном случае каждый цикл шины непроизводительно растягивается действиями запроса и разрешения. Ведущий шины с бо́льшим приоритетом может получить шину с помощью своей линии $\overline{BREQ}$ и логики учета приоритетов. После того, как текущий ведущий шины завершает свой цикл шины, он освобождает шину и снимает сиг-


Общая синхронизация шины 10 МГц
Приоритетный контроллер
Дешифратор
Шифратор приоритетов
+5В +5В
CBLK
BREQ
BPRN
Модуль 1
Генератор синхронизации
8289
8086
8288
8283
8287
CBLK
BUSY
CBRQ
BREQ
BPRN
Модуль 2
Генератор синхронизации
8289
8086
8288
8283
8287
BREQ
BPRN
Модуль
Генератор синхронизации
8289
8086
8288
8283
8287
Шина управления
Шина адреса
Шина данных

нал $\overline{\text{BUSY}}$. Запрашивающий ведущий шины, который получает сигнал $\overline{\text{BPRN}}$, формирует активный сигнал $\overline{\text{BUSY}}$ и начинает использовать шину.

Арбитр шины с меньшим приоритетом может запросить шину по линии $\overline{\text{CBRQ}}$ (но для этого линии $\overline{\text{CBRQ}}$ включаются в соответствии с рис. 11.11). При низком уровне сигнала $\overline{\text{CBRQ}}$ текущий ведущий шины освобождает шину, если он находится в холостом состоянии T_I. Логика учета приоритетов распределяет шину запрашивающему арбитру со следующим по старшинству приоритетом, посылая сигнал по соответствующей линии $\overline{\text{BPRN}}$.

В конфигурации, разрешающей устройствам с меньшим приоритетом получать управление шиной, арбитр шины 8289 имеет два входа – ANYRQST (любой запрос) и $\overline{\text{CRQLCK}}$ (блокировка общего запроса шины). Вход ANYRQST рассчитан на аппаратное программирование. Когда он подключен к источнику +5 В, сигнал $\overline{\text{CBRQ}}$ заставляет арбитр шины освободить шину по окончании текущего цикла независимо от приоритета запрашивающего ведущего шины. Вход $\overline{\text{CRQLCK}}$ позволяет арбитру шины игнорировать сигнал $\overline{\text{CBRQ}}$. Следовательно, активный сигнал $\overline{\text{CRQLCK}}$ запрещает арбитру освобождать шину любому другому ведущему шины, имеющему более низкий приоритет.

При реализации способа приоритетной цепочки внешний приоритетный контроллер не нужен (см. рис. 11.12). Линия $\overline{\text{BPRO}}$ (выход приоритета шины) каждого модуля подключается на вход $\overline{\text{BPRN}}$ следующего ведущего шины с меньшим приоритетом. Для любого процессорного модуля в цепочке низкий уровень на входе $\overline{\text{BPRN}}$ означает, что он имеет наибольший приоритет и может распоряжаться шиной. Если же на входе $\overline{\text{BPRN}}$ действует высокий уровень, модуль не получил сигнала разрешения шины и не передает сигнал разрешения на выход $\overline{\text{BPRO}}$. Сигнал $\overline{\text{BPRN}}$ арбитра с наибольшим приоритетом подключается на землю. Таким образом, на входе $\overline{\text{BPRN}}$ запрашивающего арбитра с наибольшим приоритетом будет низкий уровень, а у всех арбитров далее по цепочке – высокий. Так как распределение шины должно производиться за один период сигнала $\overline{\text{BCLK}}$ (синхронизация шины), в цепочке обычно не допускается более трех процессорных модулей.

По мере подключения к системной шине большего числа процессорных модулей использование шины быстро достигает насыщения. Перегрузка шины серьезно ухудшает производительность системы и ликвидирует такое достоинство мультипроцессорной системы, как быстродействие. Влияние числа процессоров на производительность системы можно грубо оценить, рассматривая систему, содержащую k идентичных модулей. Допустим, что скорость выполнения команд пропорциональна частоте обращения модуля к шине. Предположим, что полоса пропускания шины соответствует N циклам в секунду и что без интерференции каждый модуль использует M циклов в секунду. Коэффициент использования шины модулем определяется как среднее используемых модулем циклов шины в предположении, что интерферен-

---

← Рис. 11.11. Подключения арбитров шины 8289 к приоритетному контроллеру

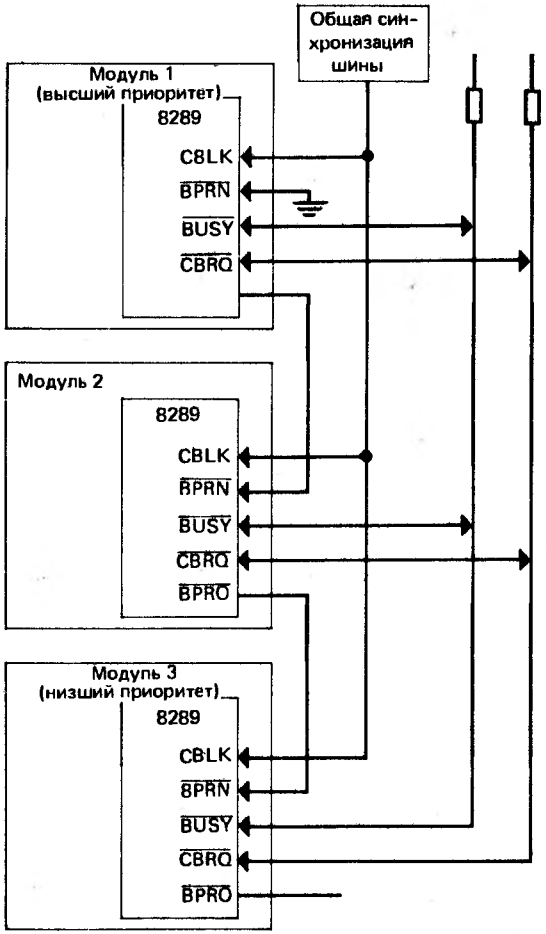


Рис. 11.12. Распределение шины с помощью приоритетной цепочки арбитров шины

ция отсутствует; этот коэффициент $B_p = M/N$. Конечно, коэффициент B_p должен быть меньше 1 и будет зависеть от выполняемых команд. Для микропроцессора 8086 типичный диапазон значений B_p составляет 0,5 . . . 0,8. Пусть I_s – системная скорость выполнения команд (т. е. число команд, выполняемых системой в секунду), а I_p – скорость выполнения команд каждым модулем при отсутствии интерференции. Тогда имеют место следующие соотношения:

$$I_s \approx \begin{cases} kI_p, & \text{если } k \leqslant 1/B_p, \\ I_p/B_p, & \text{если } k > 1/B_p. \end{cases}$$

Типичный график зависимости I_s от числа процессорных модулей для $B_p = 1/2$ показан на рис. 11.13, *а*. Хотя I_s примерно равна kI_p в ненасыщенной

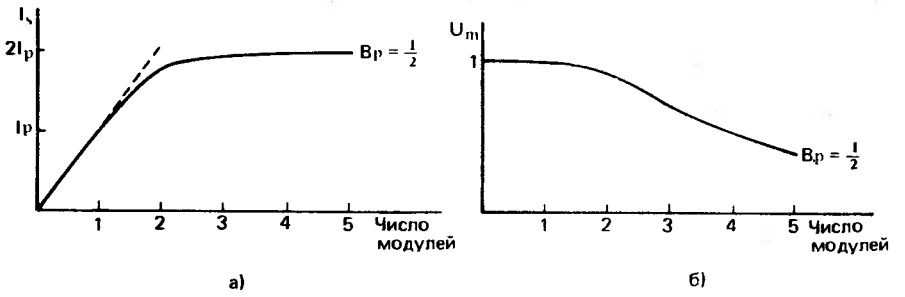


Рис. 11.13. Критерии производительности мультипроцессорной системы (*а*) и отдельного модуля (*б*)

системе, фактически I_s меньше kI_p, так как модуль вводит состояния ожидания, если ему необходим доступ к системной шине, а она занята другим процессором. Важно отметить, что при достижении насыщения шины добавление большего числа модулей не увеличивает производительность системы. Шина становится узким местом системы, а сама система называется *системой, ограниченной шиной.*

Коэффициент использования каждого модуля, который определяется как отношение скорости выполнения команд (при наличии других модулей) к I_p, ухудшается по мере увеличения перегрузки шины, т. е. когда kM приближается к N. Этот коэффициент равен

$$U_m \leqslant \frac{N/k}{M} = \frac{N}{kM} = 1/kB_p.$$

На рис. 11.13, *б* показано влияние числа модулей на производительность каждого модуля для $B_p = 1/2$.

Наиболее эффективное решение проблемы перегрузки шины заключается в том, чтобы ввести в каждый процессорный модуль локальную память, предназначенную только для его ЦП. Каждый модуль может использовать локальную память для хранения своей программы и для рабочей области, а разделенная память резервируется, главным образом, для межмодульных взаимодействий. Это позволяет нескольким ЦП выбирать команды и обращаться к операндам одновременно, сокращая передачи по системной шине и повышая степень параллельной обработки.

Если процессорный модуль имеет локальную шину, в нем требуются две логики управления шиной: для доступа к системной шине и для доступа к локальной шине. Каждая схема должна содержать контроллер шины, регистры-защелки адреса и приемопередатчики данных. Так как локальная шина предназначена только для одного ЦП, здесь не требуется арбитр шины, а регистры-защелки адреса всегда разрешены (в предположении, что локальный контроллер ПДП отсутствует). В течение цикла шины адрес дешифрируется для определения того, в каком пространстве он находится: в локальном или разделенном. Если адрес находится в локальном пространстве, разрешается

контроллер локальной шины и цикл шины продолжается обычным образом. В это же время выход дешифратора запрещает контроллер системной шины (и заставляет арбитр шины освободить системную шину, если он управляет ею). Если же адрес находится в адресном пространстве системной шины, дешифратор адреса выдает активный сигнал на вход SYSB/$\overline{\text{RESB}}$ арбитра шины и делается попытка доступа к системной шине. В данной ситуации запрещен контроллер локальной шины и он не выдает управляющих сигналов шины, а интерфейс системной шины разрешен. Затем арбитр шины запрашивает использование системной шины и инициирует цикл шины, когда он становится арбитром с наибольшим приоритетом.

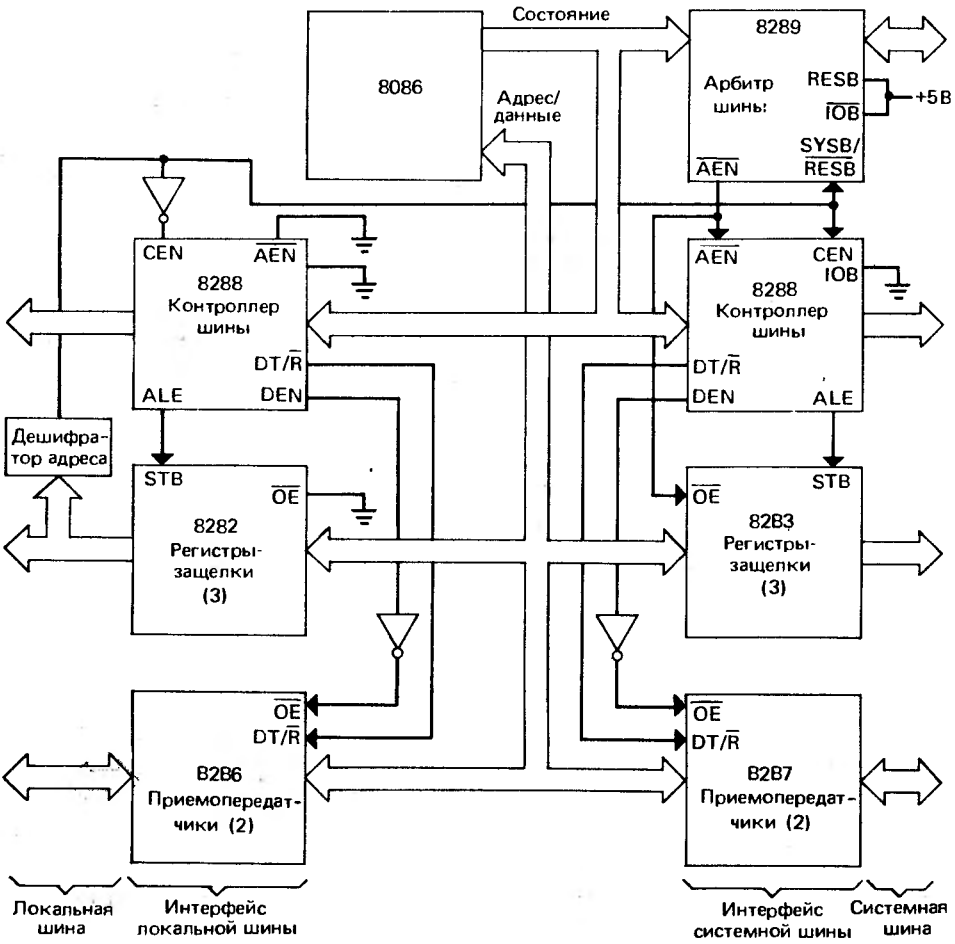


Рис. 11.14. Конфигурация с локальной и системной шинами

Как видно на рис. 11.14, линии RESB и $\overline{IOB}$ подключены к линии питания +5 В. При введении локальных ресурсов каждый процессорный модуль должен лишь изредка обращаться к системной шине. Следовательно, целесообразно разрешить ANYRQST, чтобы арбитр 8289 освобождал шину процессорным модулям с меньшими приоритетами, как только завершается текущий цикл шины.

На рис. 11.15 показана еще одна конфигурация, которая сокращает передачи по системной шине, а схемно оказывается проще конфигурации на рис. 11.14. Здесь весь ввод-вывод реализован на локальной шине, а вся память – на системной. Разделение адресных пространств ввода-вывода и памяти позволяет арбитру шины управлять передачами адресов и управляющих сигналов, что делает ненужными дешифратор адреса и контроллер локаль-

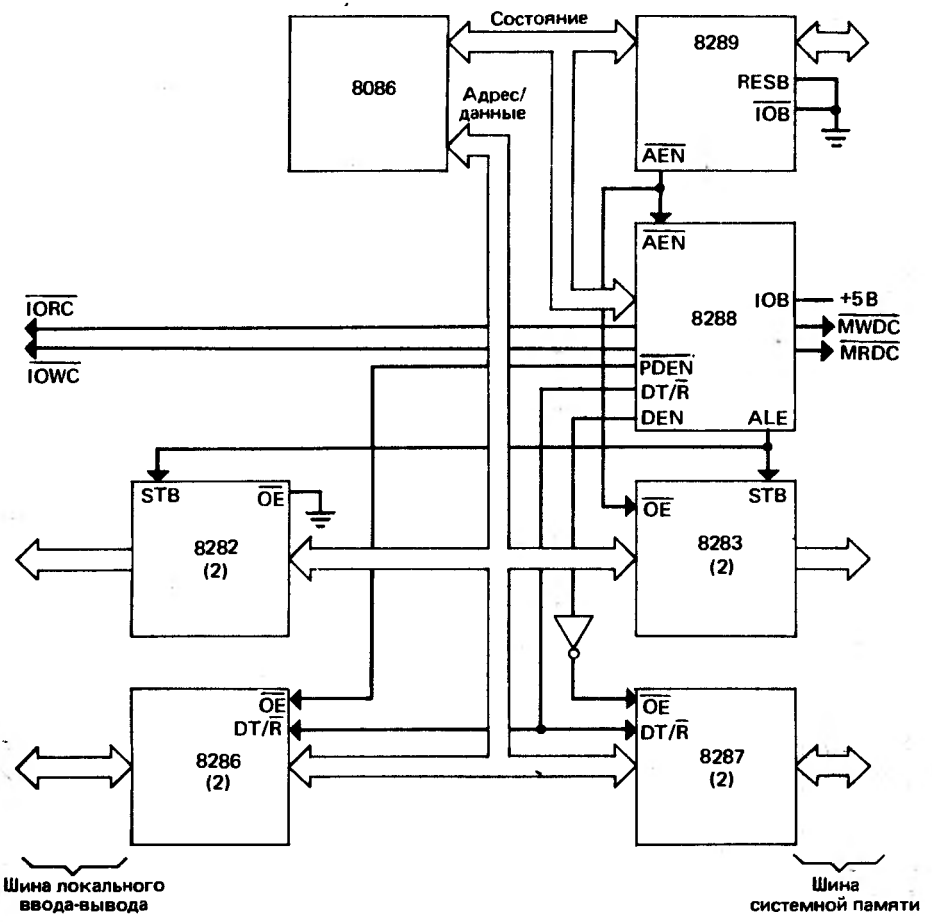


Рис. 11.15. Конфигурация с отдельной локальной шиной ввода-вывода

ной шины. При подключении входа RESB у арбитра шины на землю сигнал SYSB/$\overline{\text{RESB}}$ игнорируется, а при подключении входа $\overline{\text{IOB}}$ у арбитра шины на землю, а входа IOB у контроллера шины к источнику +5 В оба эти прибора переводятся в режим шины ввода-вывода. В этом режиме системная шина запрашивается и освобождается в соответствии с уровнем на линии $\overline{\text{S2}}$, который показывает передачу ввода-вывода (низкий уровень) или памяти (высокий уровень). Выход $\overline{\text{AEN}}$ арбитра шины 8289 все еще используется для управления линиями адреса/данных и приказов памяти, но он не влияет на линии управления вводом-выводом $\overline{\text{IORC}}$, $\overline{\text{IOWC}}$, $\overline{\text{AIOWC}}$ и $\overline{\text{INTA}}$ (выходы которых постоянно разрешены). Когда линия IOB контроллера шины 8288 подключена к источнику +5 В, линия MCE/$\overline{\text{PDEN}}$ становится выходом разрешения периферийных данных ($\overline{\text{PDEN}}$), который применяется для разрешения приемопередатчиков данных ввода-вывода. При выполнении передачи ввода-вывода сигнал $\overline{\text{PDEN}}$ активен, а сигнал DEN пассивен; при обращении же к памяти сигнал DEN активен, а сигнал $\overline{\text{PDEN}}$ пассивен. Данная конфигурация особенно удобна, когда она реализуется совместно с процессором ввода-вывода 8089. Допускается также подключать к системной шине интерфейсы ввода-вывода, но их порты должны адресоваться как память (реализуются устройства ввода-вывода, отображенные на память).

Режимы локальной шины и шины ввода-вывода можно объединить, если подключить у арбитра шины линию RESB к источнику +5 В, а линию $\overline{\text{IOB}}$ на землю. Получающаяся конфигурация позволяет разместить часть памяти на локальной шине вместе с устройствами ввода-вывода.

Ранее говорилось, что локальная память в мультипроцессорной системе уменьшает использование системной шины. Однако процессорный модуль первоначально должен загрузить управляющую программу из разделенной системной памяти в свою локальную память и пересылать результаты из локальной памяти в системную, если они потребуются другим модулям. Чтобы еще более сократить передачи по системной шине, вызванные межпроцессорным взаимодействием, можно применить двухпортовую память. Такая па-

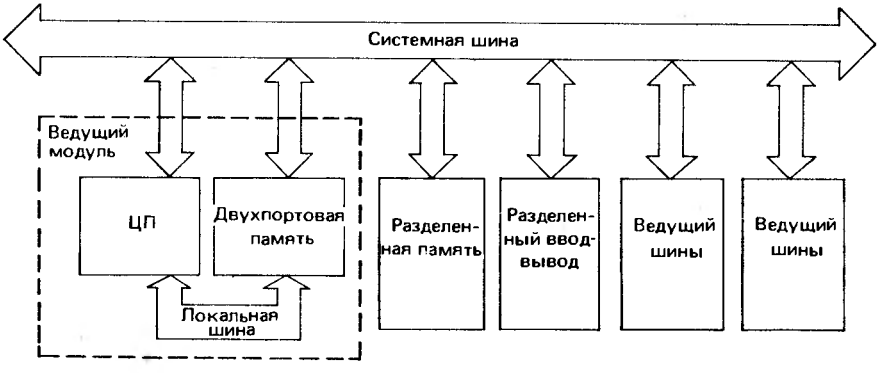


Рис. 11.16. Двухпортовая память

мять аппаратно реализует принцип, аналогичный программному принципу использования общей области для реализации взаимодействия между вызывающей программой и подпрограммой. Память имеет два порта, обеспечивающих обращения с локальной и системной шинами. Как видно из рис. 11.16, процессор ведущего модуля, в котором находится двухпортовая память, обращается к памяти по локальной шине. Другие процессорные модули так-

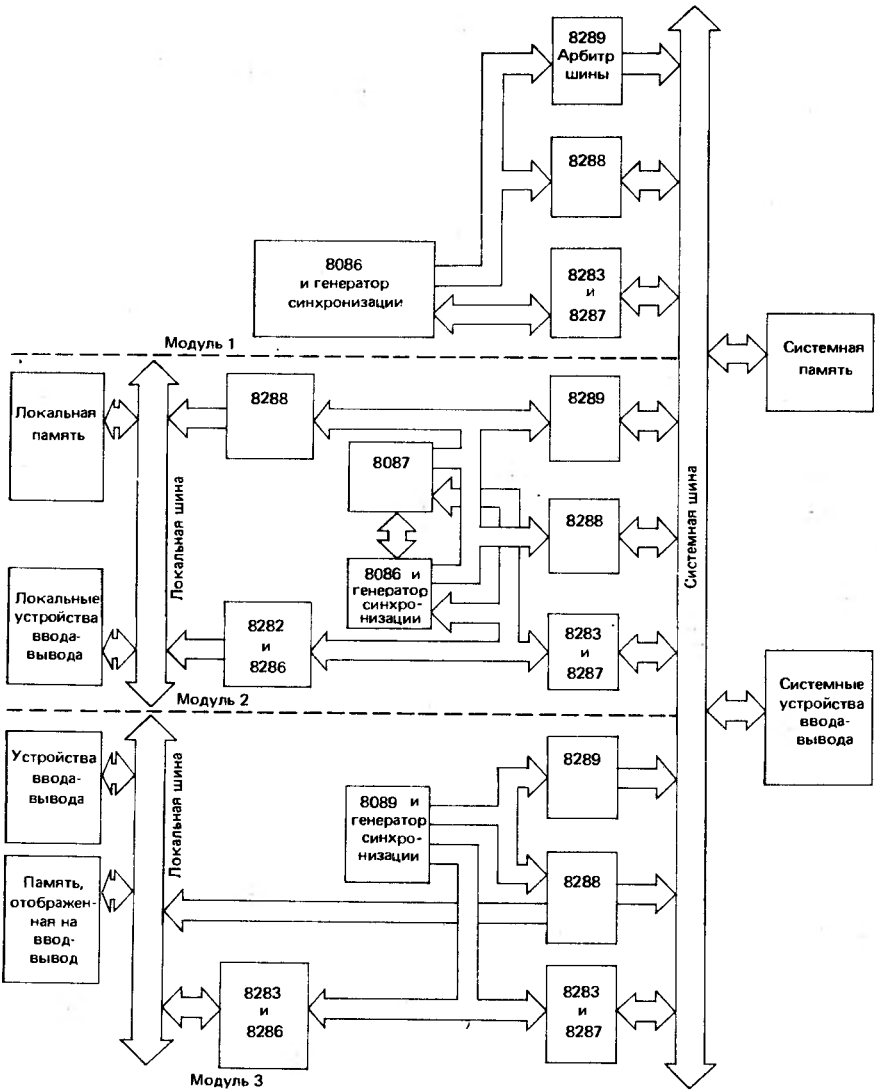


Рис. 11.17. Сложная мультипроцессорная система

же обращаются к этой же памяти, но только по системной шине. Данная конфигурация особенно удобна при вводе или выводе больших блоков данных через процессорный модуль, который специализирован на операциях ввода-вывода. Здесь блокировка системной шины не препятствует доступу ведущего модуля к двухпортовой памяти и наоборот. Чтобы в данном способе избежать проблем критических секций (см. § 11.1), в разделенной системной памяти должен быть организован семафор; в противном случае реализация значительно усложняется.

Таким образом, различные конфигурации процессорных модулей можно объединять, образуя сложную слабо связанную мультипроцессорную систему. Каждым модулем в такой системе может быть:

один микропроцессор 8086/8086 или независимый процессор, например процессор ввода-вывода 8089;

группа процессоров, состоящая из микропроцессоров 8086/8088 и сопроцессора и (или) независимого процессора;

группа независимых процессоров, например процессоров ввода-вывода 8089.

Кроме того, каждый модуль может иметь локальную шину или специализированную шину ввода-вывода. На рис. 11.17 показана одна из многих мультипроцессорных систем, содержащая несколько сложных процессорных модулей.

11.2.4. МИКРОКОМПЬЮТЕРНЫЕ СЕТИ

Рассмотренные мультипроцессорные конфигурации имеют общее свойство – все процессоры разделяют одну и ту же системную шину. Следовательно, межпроцессорные взаимодействия осуществляются через разделенную память, а процессоры физически должны быть расположены близко друг к другу.

При использовании последовательных линий связи множество микрокомпьютеров может взаимодействовать друг с другом и разделять некоторые аппаратные и программные ресурсы. Большие системы подобного типа называются *компьютерными сетями*. Обычно для передач сообщений между системами применяется синхронная последовательная связь. Приказы и данные передаются в виде пакетов сообщений. Каждый пакет состоит из нескольких полей, содержащих символы синхронизации, адрес отправителя, адрес получателя, текст, контрольные символы и символы-терминаторы. Управляющая информация и ее размещение называется системным *протоколом связи*. Каждый микрокомпьютер (или мультипроцессорный элемент) относительно независим от других, а расстояния между ними могут составлять тысячи километров. Отказ в одном элементе сети можно легко изолировать, не влияя на остальную часть системы.

11.3. ПРОЦЕССОР ЧИСЛОВЫХ ДАННЫХ

Процессор числовых данных 8087 специально разработан для эффективного выполнения арифметических операций. Он может работать с целыми, десятичными и вещественными числами, длина которых составляет 2-10 байт.

Система команд включает в себя не только разнообразные формы сложения, вычитания, умножения и деления, но и такие функции, как извлечение квадратного корня, возведение в степень, нахождение тангенса и т. д. Например, процессор 8087 умножает два 64-битных вещественных числа за 27 мкс и извлекает квадратный корень за 36 мкс. При эмуляции на микропроцессоре 8086 эти операции длятся соответственно около 2 и 20 мс. Процессор 8087 обеспечивает простой и эффективный способ повышения производительности систем на базе микропроцессоров 8086/8088, особенно для вычислительных применений.

Процессор 8087 не может выбирать свои команды, поэтому он должен работать с микропроцессорами 8086/8088, которые действуют как главный процессор в сопроцессорной конфигурации. Интерфейс между ними сводится к простым соединениям соответствующих контактов и никаких дополнительных компонент не требуется (см. рис. 11.5).

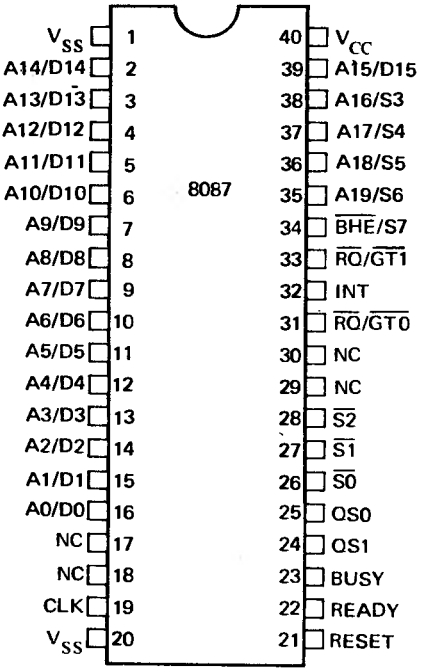


Рис. 11.18. Разводка контактов корпуса процессора 8087

Разводка контактов процессора 8087 показана на рис. 11.18. Как видно, линии адреса/данных, состояния, готовности, сброса, синхронизации, земли и питания выведены на те же контакты, что и у микропроцессоров 8086/8087. Из оставшихся восьми контактов четыре не используются. Подключения остальных контактов: контакт $\overline{BUSY}$ к контакту $\overline{TEST}$ главного процессора; контакт $\overline{RQ}/\overline{GT0}$ к контактам $\overline{RQ}/\overline{GT0}$ или $\overline{RQ}$/GT1 главного процессора; контакт INT в логику управления прерываниями (предполагая, что прерывания 8087 разрешены); контакт $\overline{RQ}/GT1$ к контакту $\overline{RQ}/\overline{GT}$ независимого процессора. Такой простой интерфейс позволяет легко усовершенствовать систему, заменив исходный ЦП на сдвоенный модуль, состоящий из микропроцессора 8086/8088 и процессора 8087.

11.3.1. ТИПЫ ДАННЫХ

Процессор 8087 может оперировать данными семи различных типов: целое (слово), короткое целое, длинное целое, упакованное BCD, короткое вещественное и временное вещественное. Число байт, формат и примерный диа-

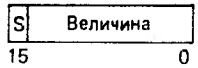
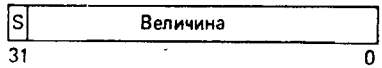
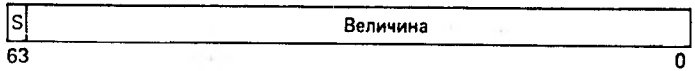
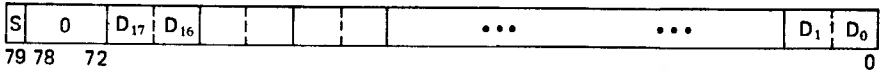
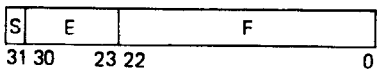
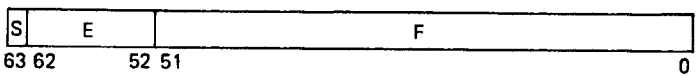
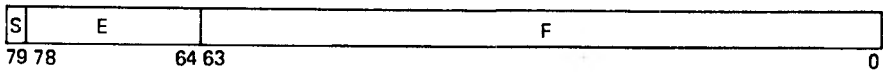
Тип данных	Число байт	Примерный диапазон (десятичный)	Формат
Целое слово	2	–32,768...32,767	
Короткое целое	4	$\approx -2 \times 10^{9} \ldots 2 \times 10^{9}$	
Длинное целое	8	$\approx -9 \times 10^{18} \ldots 9 \times 10^{18}$	
Упакованное BCD	10	$-10^{18}+1 \ldots 10^{18}-1$	
Короткое вещественное	4	$\approx \pm 1 \times 10^{-38} \ldots \pm 3 \times 10^{38}$	
Длинное вещественное	8	$\approx \pm 10^{-308} \ldots \pm 10^{308}$	
Временное вещественное	10	$\approx \pm 10^{-4932} \ldots \pm 10^{4932}$	

Рис. 11.19. Форматы данных процессора 8087

пазон этих типов данных приведены на рис. 11.19. В памяти младший байт числа всегда хранится по меньшему адресу.

Микропроцессоры 8086/8088 оперируют целыми числами длиной 16 бит, имеющими диапазон $-2^{15} \ldots 2^{15} - 1$. Кроме формата целого слова, процессор 8087 допускает целочисленные операнды длиной 32 бита (короткое целое) и 64 бита (длинное целое). Отрицательные целые числа представляются в дополнительном коде. Увеличение длины чисел соответственно расширяет диапазон до $-2^{31} \ldots 2^{31} - 1$ (короткое целое) и $-2^{63} \ldots 2^{63} - 1$ (длинное целое), что примерно составляет $\pm 2 \times 10^9$ и $\pm 9 \times 10^{18}$.

В упакованном BCD-формате десятичное число занимает 10 байт. Весь старший байт отведен для знака. Его старший бит показывает, каким является десятичное число: положительным (0) или отрицательным (1). Остальные 9 байт – это значащие разряды, причем каждый байт содержит две цифры. Допустимый диапазон значений в этом формате составляет $-10^{18} + 1 \ldots \ldots 10^{18} - 1$.

Вещественные данные, называемые также числами с плавающей точкой, допускают представление очень малых и очень больших значений, причем в вычислениях сохраняется постоянное число значащих цифр. Формат действительных данных состоит из трех полей: знак, порядок (exp) и мантисса. Другими словами, число X равно:

$$X = \pm 2^{exp} \times \text{мантисса}.$$

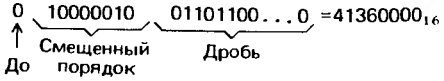
Порядок определяет положение двоичной точки в мантиссе. Уменьшение его на 1 передвигает двоичную точку вправо на один разряд. Следовательно, с помощью отрицательного порядка можно представлять очень малые числа без потери точности. Однако, за исключением чисел, мантисса которых точно попадает в диапазон формата, число может оказаться представимым не точно, что вызывает ошибку округления. Если разрешить наличие старших нулей, любое число имеет несколько представлений в формате вещественного числа. Но так как число бит мантиссы фиксировано, старшие нули увеличивают ошибку округления. Поэтому для минимизации ошибок округления после каждой операции процессор 8087 удаляет старшие нули, соответственно корректируя порядок. Ненулевое вещественное число называется нормализованным, если его мантисса имеет вид 1.F, где F – дробь.

Обычно к истинному порядку прибавляется смещение, так что истинный порядок равен числу в поле порядка минус значение смещения. Применение смещенных порядков позволяет сравнивать два нормализованных вещественных числа с одинаковыми знаками обычным образом слева направо, как будто они являются целыми числами.

Процессор 8087 распознает три типа вещественных данных: короткое вещественное, длинное вещественное и временное вещественное. В формате коротких вещественных данных смещенный порядок E и дробь F имеют соответственно 8 и 23 бита, а число представляется в виде

$$(-1)^S \times 2^{E-127} \times 1.F,$$

где S — бит знака. Так как 1 перед дробью присутствует всегда, она подразумевается, но физически не хранится. Например, предположим, что короткое вещественное число хранится в следующем виде:



Тогда истинный порядок равен 130 − 127 = 3 и число с плавающей точкой равно

$$+1.011011 \times 2^3 = +1011.011 = 11.375_{10}.$$

Покажем преобразование числа 20.59375_{10} в короткий вещественный формат. Сначала целую и дробную части необходимо преобразовать в двоичную систему:

$$10100 + 0.10011 = 10100.10011.$$

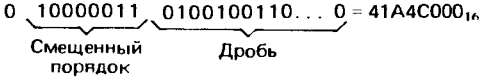
После нормализации посредством перемещения двоичной точки в положение между первым и вторым битами получим

$$1.010010011 \times 2^4.$$

Отсюда видно, что

$$S \doteq 0, \quad E = 127 + 4 = 131 = 10000011, \quad F = 010010011$$

и число в коротком вещественном формате имеет вид



В данном формате диапазон смещенного порядка составляет $0 < E < 255$. Следовательно, можно представлять числа от $\pm 2^{-126}$ до $\pm 2^{128}$ или от $\pm 1 \times 10^{-38}$ до $\pm 3 \times 10^{38}$.

Смещенный порядок из всех единиц зарезервирован для бесконечности или "не-числа" (NAN). Смещенный порядок из всех нулей используется для представления +0 (все нули со знаком +), −0 (все нули со знаком −) или денормализованного числа. Денормализованное число представляет собой результат, который вызывает антипереполнение и имеет старшие нули в мантиссе даже после того, как порядок скорректирован до наименьшего возможного значения. "Не-числа" и денормализованные числа обычно применяются для указания переполнений и антипереполнений соответственно, хотя они могут использоваться и для других целей.

Формат длинного вещественного имеет 11 бит порядка и 52 бита мантиссы. Как и в предыдущем формате, первый ненулевой бит мантиссы подразумевается, но не хранится. Диапазон представимых ненулевых величин расширяется от $\pm 10^{-308}$ до $\pm 10^{308}$. Для сравнения укажем, что диапазон ненулевых вещественных чисел для IBM 370 составляет "всего" $\pm 10^{-78} \ldots \pm 10^{76}$.

Внутри процессора 8087 все числа хранятся во временном вещественном формате, имеющем 15 бит порядка и 64 бита мантиссы. В отличие от корот-

кого и длинного вещественных форматов старший бит мантиссы представлен явно. Благодаря увеличенной точности (19-20 десятичных цифр) целые и упакованные BCD-операнды допускают внутреннюю обработку операциями с плавающей точкой, давая при этом точные результаты. Временный вещественный формат используется для внутреннего хранения данных, при этом уменьшается вероятность переполнений и антипереполнений в последовательности вычислений, окончательный результат которых находится в требуемом диапазоне. Рассмотрим, например, вычисление выражения

$$D \leftarrow (A*B)/C,$$

где A, B, C и D представлены в коротком вещественном формате. Результат операции умножения может быть слишком велик для короткого вещественного формата, но после деления на C окончательный результат может находиться в допустимом диапазоне. Благодаря 15-битному порядку диапазон допустимых чисел во временном вещественном формате составляет примерно $\pm 10^{\pm 4932}$; следовательно, в большинстве применений пользователь может не беспокоиться о переполнениях и антипереполнениях в промежуточных вычислениях.

Чтобы поддерживать форматы данных процессора 8087, в ассемблере ASM-86 кроме директив DB, DW и DD предусмотрены директивы объявления данных DQ (определить счетверенное слово) и DT (определить 10 байт) для распределения памяти и определения данных. Директива DQ определяет 8-байтный блок памяти для длинного целого и длинного вещественного чисел. С помощью директивы DT распределяются 10 байт и память можно инициализировать на числа в упакованном десятичном или временном вещественном форматах. Примеры использования директив DQ и DT приведены на рис. 11.20. С типами DQ и DT можно использовать оператор PTR так же, как с типами DB, DW и DD.

```
SHORT_REAL      DD      1.25            ;ЗАПОМНИТЬ КАК КОРОТКОЕ ВЕЩЕСТВЕННОЕ
SHORT_INT       DD      1234568H        ;ЗАПОМНИТЬ КАК КОРОТКОЕ ЦЕЛОЕ
LONG_REAL       DQ      -1.9            ;ЗАПОМНИТЬ КАК ДЛИННОЕ ВЕЩЕСТВЕННОЕ
LONG_INT        DQ      123456ABCDH     ;ЗАПОМНИТЬ КАК ДЛИННОЕ ЦЕЛОЕ
BCD             DT      -91567          ;ЗАПОМНИТЬ КАК УПАКОВАННОЕ ДЕСЯТИЧНОЕ
REAL_CONST      DT      -1.3            ;ЗАПОМНИТЬ КАК ВРЕМЕННОЕ ВЕЩЕСТВЕННОЕ
```

Рис. 11.20. Примеры директив DD, DQ и DT

11.3.2. АРХИТЕКТУРА ПРОЦЕССОРА

Общая схема процессора 8087 приведена на рис. 11.21. Логика контроля и управления обеспечивает работу 6-байтной очереди команд (при работе вместе с микропроцессором 8088 используются только 4 байта) и следит за последовательностью выполнения команд главного процессора. Если текущая команда главного процессора оказывается командой ESC, процессор 8087 дешифрирует код внешней операции для выполнения указанной опера-

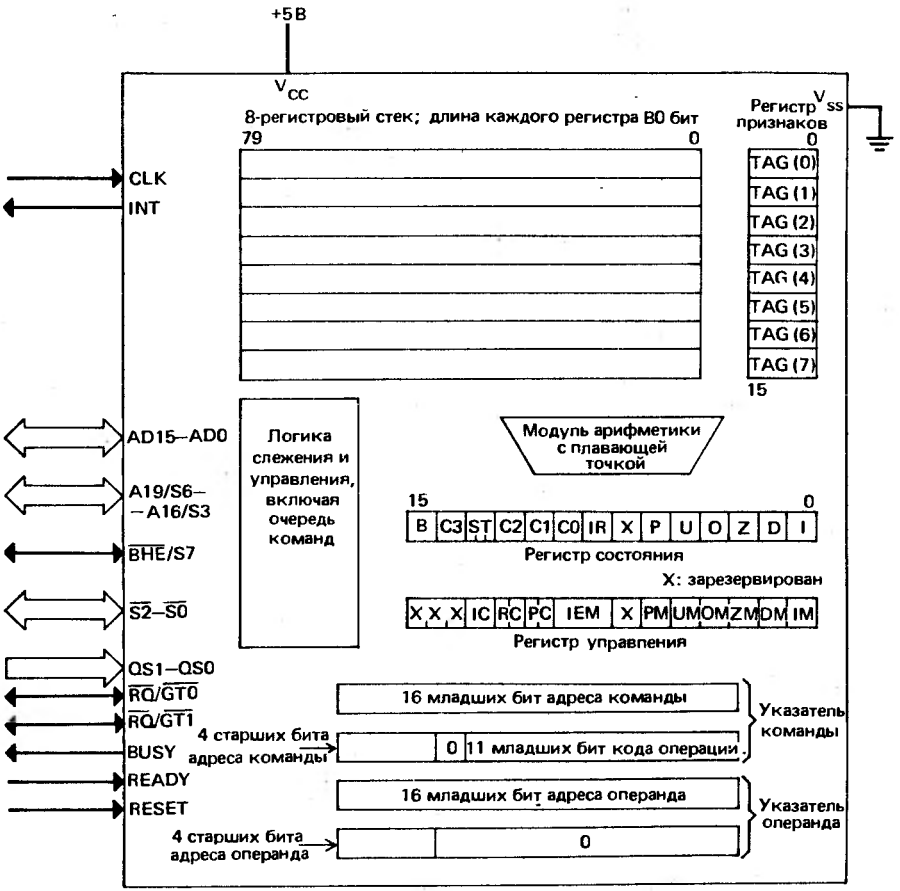


Рис. 11.21. Схема процессора 8087

ции, а также фиксирует операнд и адрес операнда. Процессор 8087 игнорирует все команды, не являющиеся командами ESC.

К восьми регистрам данных можно обращаться как к стеку или произвольно относительно вершины стека. Операнд можно извлечь из стека или включить в стек. На верхний элемент стека указывают биты ST, которые являются битами 13, 12 и 11 регистра состояния. Операция включения сначала производит декремент ST на 1, а затем загружает операнд в новый верхний элемент стека; операция извлечения считывает операнд из вершины стека, а затем производит инкремент ST на 1.

Как в обычном регистровом файле, к каждому регистру можно обратиться, пользуясь индексированием относительно указателя стека. При такой адресации, называемой относительной стековой, регистры располагаются как

бы по окружности, причем регистр 7 следует за регистром 0. Если, например, ST содержит 3, то ST(2) и ST(6) представляют собой регистры 5 и 1 соответственно. Так как все числа внутри процессора 8087 хранятся во временном вещественном формате, каждый регистр содержит 80 бит.

Длина регистра состояния равна 16 битам. Он фиксирует различные ошибки, хранит код условия для некоторых команд, определяет регистр – вершину стека и показывает состояние занятости. Единица в битах 0-5, 7 и 15 означает, что данное условие существует.

Бит 0 (I) – недействительная операция, например переполнение стека, антипереполнение (опустошение) стека, недействительный операнд, извлечение квадратного корня из отрицательного числа и т. д.

Бит 1 (D) – операнд не нормализован.

Бит 2 (Z) – ошибка деления на нуль.

Бит 3 (O) – ошибка переполнения порядка, т. е. смещенный порядок слишком велик.

Бит 4 (U) – антипереполнение порядка, т. е. смещенный порядок слишком мал.

Бит 5 (P) – ошибка точности, т. е. результат точно не представим в формате получателя и округляется. Этот бит обычно используется только тогда, когда требуются точные результаты.

Бит 6 – зарезервирован.

Бит 7 (IR) – сформирован запрос прерывания.

Биты 8, 9, 10, 14 – показывают код условия, C0, C1, C2 и C3. Код условия устанавливается командами сравнения и анализа, которые рассматриваются далее.

Биты 13, 12 и 11 (ST) – показывают элемент (регистр), являющийся вершиной стека.

Бит 15 (B) – текущая операция не закончена.

После сброса или инициализации процессора 8087 все биты состояния, за исключением кода условия, сбрасываются.

Процессор 8087 распознает шесть типов ошибок, но каждый тип можно индивидуально замаскировать от генерирования прерывания установкой в 1 соответствующих битов маски в регистре управления. Биты маски обозначены IM (недействительная операция), DM (денормализованный операнд), ZM (деление на нуль), OM (переполнение), UM (антипереполнение) и PM (ошибка точности). Если ошибка замаскирована, она не вызывает прерывания, а процессор 8087 реализует стандартную реакцию и переходит к следующей команде. (Описания стандартных реакций приведены в фирменных материалах.) Например, ошибку точности, для которой стандартной реакцией является "возвратить округленный результат", необходимо замаскировать в арифметике с плавающей точкой, так как в большинстве применений ошибки точности возникают очень часто. Ошибки точности имеют последствия только в специальных ситуациях.

Генерирование прерываний, включая и запросы из-за ошибок, зависит также от бита маски разрешения прерываний (IEM) в регистре управления. Когда он установлен в 1, все прерывания запрещены, за исключением случая,

когда ЦП выполняет команду WAIT. Если же IEM = 0, незамаскированная ошибка вызывает прерывание ЦП с переходом на процедуру обработки ошибки. В этой процедуре хранимые в процессоре 8087 указатели текущей команды и операнда можно передать в память соответствующими командами процессора 8087 (см. раздел 11.5.3), а затем проанализировать. Содержимое указателей идентифицирует адреса команды и операнда, когда возникла ошибка. Отметим, что в указателе команды хранятся только 11 младших бит кода операции, так как старшие 5 бит всегда содержат 11011 (код операции команды ESC).

Остальные биты в регистре управления обеспечивают гибкое управление точностью (PC), округлением (RC) и представлением бесконечности (IC). Эти биты определяются следующим образом:

Биты PC — 00 означает точность 24 бита,
01 зарезервирована,
10 означает точность 53 бита,
11 означает точность 64 бита.

Биты RC — 00 означает округление к ближайшему,
01 означает округление к $-\infty$,
10 означает округление к $+\infty$,
11 означает усечение (отбрасывание).

Бит IC — 0 показывает, что $+\infty$ и $-\infty$ считаются одной беззнаковой бесконечностью,
1 показывает, что $+\infty$ и $-\infty$ считаются двумя знаковыми бесконечностями.

При сбросе или инициализации процессора 8087 производится следующая установка регистра управления: (PC) = 11, (RC) = 00, (IC) = 0, (IEM) = 0, все биты масок ошибок устанавливаются в 1.

Регистры признаков (тэгов) хранят состояние содержимого регистров данных. Каждый регистр данных ассоциируется с регистром признаков, который показывает, является его содержимое действительным (00), нулевым (01), специальным значением (NAN, бесконечностью или денормализованным числом) (10) или пустым (11).

11.3.3. СИСТЕМА КОМАНД

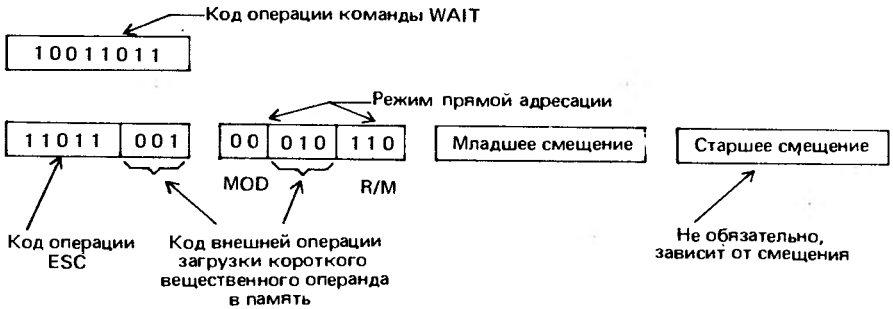
Процессор 8087 имеет 68 команд, которые по выполняемым им функциям можно разделить на шесть групп: передач данных, арифметические, сравнения, трансцендентные, оперирующие константами и управления процессором.

Так как процессор 8087 и главный ЦП разделяют один и тот же командный поток, ассемблер ASM-86 дает возможность пользователю писать программы в системе команд, объединяющей команды обоих процессоров (8086 и 8087). Для каждой команды процессора 8087 ассемблер генерирует две команды — команду ожидания WAIT и команду ESC, код внешней операции

которой показывает команду процессора 8087. Например, если SHORT_REAL определено директивой DD, команда

```
FST  SHORT_REAL
```

преобразования вершины стека в короткий вещественный формат и запоминания результата в SHORT_REAL будет ассемблирована следующим образом:



Команда WAIT перед командой ESC заставляет микропроцессоры 8086/8088 ввести состояние ожидания до появления активного сигнала на входе $\overline{\text{TEST}}$. Она необходима для того, чтобы команда ESC не дешифрировалась до завершения процессором 8087 его текущей команды. Если сигнал $\overline{\text{TEST}}$ уже активен или становится активным, процессор 8087 дешифрирует команду ESC, а затем оба процессора 8086 и 8087 работают параллельно.

Однако команду WAIT необходимо указывать явно, когда ЦП требуется обратиться в память к операнду, участвовавшему в предыдущей команде ESC. Например, в следующем фрагменте ЦП должен ожидать до тех пор, пока процессор 8087 не возвратит свой результат в SHORT_REAL, прежде чем перейти к команде CMP:

```
FST   SHORT_REAL
  .
  .   Команды 8086, в которых не фигурирует
  .   SHORT_REAL
  .
WAIT
CMP   BYTE PTR SHORT_REAL + 3, 0
JG    POSITIVE_REAL
  .
  .
  .
```

Отметим, что фирма Intel разработала программный модуль, который эмулирует все команды процессора 8087. В программе, выполняемой посредством эмуляции, для синхронизации необходимо пользоваться командой FWAIT (это другая мнемоника команды WAIT). Так как при эмуляции процессора 8087, формирующего сигнал на входе $\overline{\text{TEST}}$, нет, модуль заменяет любую команду FWAIT или введенную команду WAIT на команду NOP,

чтобы избежать бесконечного ожидания. Однако явные команды WAIT из пользовательского объектного кода не удаляются.

Рассмотрим определения ассемблерных команд процессора 8087. Многие ассемблерные команды позволяют определять операнды несколькими способами, явно или неявно. В таких командах для разделения альтернативных форм операндов применяется наклонная черта. Например, поле операнда в виде

```
//SRC/DST,SRC
```

означает, что команду допустимо кодировать в одной из следующих форм:

1. Операнд-источник и операнд-получатель указаны неявно (между первыми двумя наклонными чертами ничего нет).
2. Операнд-источник определен, а операнд-получатель неявный.
3. Определены операнд-источник и операнд-получатель.

Операндом может быть регистр в регистровом стеке или операнд в памяти. Для операнда-регистра ST представляет собой верхний элемент стека, а ST(i) — i-й регистр ниже вершины стека (т. е. регистр, соответствующий ST + i). Операнд в памяти адресуется в любом режиме адресации данных микропроцессора 8086. Операнд в памяти может иметь любой тип данных.

Группа команд передач данных содержит 9 команд, приведенных на рис. 11.22. Операнд с любым типом данных в памяти можно преобразовать во временный вещественный формат, а затем включить в регистровый стек.

НАЗВАНИЕ	МНЕМОНИКА И ФОРМЫ ОПЕРАНДОВ	ОПИСАНИЕ И ВОЗМОЖНЫЕ ОШИБКИ
ЗАГРУЗИТЬ (ВЕЩЕСТВЕННЫЙ ФОРМАТ)	FLD SRC (SRC: ST(*i*) ИЛИ ОПЕРАНД В ПАМЯТИ КОРОТКОГО ВЕЩЕСТВЕННОГО, ДЛИННОГО ВЕЩЕСТВЕННОГО ИЛИ ВРЕМЕННОГО ВЕЩЕСТВЕННОГО ТИПА)	ПРОИЗВЕСТИ ДЕКРЕМЕНТ ST, ПРЕОБРАЗОВАТЬ СОДЕРЖИМОЕ SRC ВО ВРЕМЕННОЕ ВЕЩЕСТВЕННОЕ И ПОМЕСТИТЬ РЕЗУЛЬТАТ В (ST). ОШИБКИ: I, D
ЗАГРУЗИТЬ (ЦЕЛЫЙ ФОРМАТ)	FILD SRC (SRC: ОПЕРАНД В ПАМЯТИ КОРОТКОГО ЦЕЛОГО ИЛИ ДЛИННОГО ЦЕЛОГО ТИПА ИЛИ ЦЕЛОЕ СЛОВО)	ПРОИЗВЕСТИ ДЕКРЕМЕНТ ST, ПРЕОБРАЗОВАТЬ СОДЕРЖИМОЕ SRC ВО ВРЕМЕННОЕ ВЕЩЕСТВЕННОЕ И ПОМЕСТИТЬ РЕЗУЛЬТАТ В (ST). ОШИБКИ: I
ЗАГРУЗИТЬ (BCD-ФОРМАТ)	FBLD SRC (SRC: ОПЕРАНД В ПАМЯТИ УПАКОВАННОГО ДЕСЯТИЧНОГО ТИПА)	ПРОИЗВЕСТИ ДЕКРЕМЕНТ ST, ПРЕОБРАЗОВАТЬ СОДЕРЖИМОЕ SRC ВО ВРЕМЕННОЕ ВЕЩЕСТВЕННОЕ И ПОМЕСТИТЬ РЕЗУЛЬТАТ В (ST). ОШИБКИ: I
ЗАПОМНИТЬ (ВЕЩЕСТВЕННЫЙ ФОРМАТ)	FST DST (DST: ST(*i*) ИЛИ ОПЕРАНД В ПАМЯТИ КОРОТКОГО ВЕЩЕСТВЕННОГО ИЛИ ДЛИННОГО ВЕЩЕСТВЕННОГО ТИПА)	ПРЕОБРАЗОВАТЬ (ST) В ВЕЩЕСТВЕННЫЙ ФОРМАТ ПОЛУЧАТЕЛЯ И ЗАПОМНИТЬ РЕЗУЛЬТАТ В DST. ОШИБКИ: I, O, U, P
ЗАПОМНИТЬ (ЦЕЛЫЙ ФОРМАТ)	FIST DST (DST: ОПЕРАНД В ПАМЯТИ КОРОТКОГО ЦЕЛОГО ТИПА ИЛИ ЦЕЛОЕ СЛОВО)	ПРЕОБРАЗОВАТЬ (ST) В ЦЕЛОЕ И ЗАПОМНИТЬ РЕЗУЛЬТАТ В DST. ОШИБКИ: I,P
ЗАПОМНИТЬ И ИЗВЛЕЧЬ (BCD-ФОРМАТ)	FBSTP DST (DST: ОПЕРАНД В ПАМЯТИ УПАКОВАННОГО ДЕСЯТИЧНОГО ТИПА)	ПРЕОБРАЗОВАТЬ (ST) В УПАКОВАННОЕ ДЕСЯТИЧНОЕ, ЗАПОМНИТЬ РЕЗУЛЬТАТ В DST И ПРОИЗВЕСТИ ИНКРЕМЕНТ ST. ОШИБКИ: I
ЗАПОМНИТЬ И ИЗВЛЕЧЬ (ВЕЩЕСТВЕННЫЙ ФОРМАТ)	FSTP DST (DST: ST(*i*) ИЛИ ОПЕРАНД В ПАМЯТИ КОРОТКОГО ВЕЩЕСТВЕННОГО, ДЛИННОГО ВЕЩЕСТВЕННОГО ИЛИ ВРЕМЕННОГО ВЕЩЕСТВЕННОГО ТИПА)	ПРЕОБРАЗОВАТЬ (ST) В ВЕЩЕСТВЕННЫЙ ФОРМАТ ПОЛУЧАТЕЛЯ, ЗАПОМНИТЬ РЕЗУЛЬТАТ В DST И ПРОИЗВЕСТИ ИНКРЕМЕНТ ST. ОШИБКИ: I, O, U, P

Рис. 11.22. Команды передач данных

ЗАПОМНИТЬ И ИЗВЛЕЧЬ (ЦЕЛЫЙ ФОРМАТ)	FISTP DST (DST: ОПЕРАНД В ПАМЯТИ КОРОТКОГО ЦЕЛОГО ИЛИ ДЛИННОГО ЦЕЛОГО ТИПА ИЛИ ЦЕЛОЕ СЛОВО)	ПРЕОБРАЗОВАТЬ (ST) В ЦЕЛОЕ, ЗАПОМНИТЬ РЕЗУЛЬТАТ В DST И ПРОИЗВЕСТИ ИНКРЕМЕНТ ST. ОШИБКИ: I, P
ОБМЕНЯТЬ РЕГИСТРЫ	FXCH //DST (DST: ST(*i*), А ЕСЛИ НЕ УКАЗАН, ПОДРАЗУМЕВАЕТСЯ ST(1).	ОБМЕНЯТЬ СОДЕРЖИМОЕ РЕГИСТРА-ПОЛУЧАТЕЛЯ И (ST). ОШИБКИ: I

В стек можно включить также любой регистр. Содержимое верхнего элемента стека можно преобразовать из временного вещественного формата в формат данных получателя, а затем запомнить в памяти. Кроме того, есть команды для извлечения содержимого регистра из стека и обмена содержимого регистра с вершиной стека. После каждой команды передачи данных модифицируется регистр признаков.

Процессор 8087 имеет много команд арифметических операций сложения, вычитания, умножения и деления. Результат всегда запоминается в вершине стека или в указанном регистре, а один из операндов-источников должен быть в вершине стека. В командах вещественной арифметики второй операнд может находиться в указанном регистре или в памяти. Предусмотрены специальные формы команды для извлечения данных из стека после завершения арифметической операции. Так как данные внутри процессора 8087 представляются во временном вещественном формате, арифметику с операндами других типов можно всегда выполнить первоначальной загрузкой операндов в соответствующие регистры командами передач данных, а затем выполнением вещественной арифметической операции. Однако имеются команды, которые допускают операнды в памяти в коротком вещественном, длинном вещественном, временном вещественном, целом (слово) или коротком целом форматах и автоматически преобразуют такие операнды до выполнения операции во временный вещественный формат.

Кроме основных арифметических операций, предусмотрены команды извлечения квадратного корня, масштабирования с применением степеней 2, деления по модулю, округления вещественных значений до целых, выделения порядка и мантиссы, образования абсолютного значения и изменения знака. Эти команды оперируют одним или двумя верхними элементами стека и помещают результат в вершину стека. Арифметические команды показаны на рис. 11.23.

НАЗВАНИЕ	МНЕМОНИКА И ФОРМЫ ОПЕРАНДОВ	ОПИСАНИЕ И ВОЗМОЖНЫЕ ОШИБКИ
СЛОЖИТЬ (ВЕЩЕСТВЕННЫЙ ФОРМАТ)	FADD //SRC/DST,SRC (СМ. ПРИМ. О ФОРМАХ ОПЕРАНДОВ)	(DST)← (DST)+(SRC) ОШИБКИ: I, D, O, U, P
СЛОЖИТЬ И ИЗВЛЕЧЬ (ВЕЩЕСТВЕННЫЙ ФОРМАТ)	FADDP DST,SRC (DST:ST(*i*), SRC:ST)	(ST(*i*)) ← (ST( ))+(ST) ИНКРЕМЕНТ ST. ОШИБКИ: I, D, O, U, P
СЛОЖИТЬ (ЦЕЛЫЙ ФОРМАТ)	FIADD SRC (SRC: ОПЕРАНД В ПАМЯТИ КОРОТКОГО ЦЕЛОГО ТИПА ИЛИ ЦЕЛОЕ СЛОВО)	(ST) ← (ST)+(SRC) ОШИБКИ: I, D, O, P
ВЫЧЕСТЬ (ВЕЩЕСТВЕННЫЙ ФОРМАТ)	FSUB //SRC/DST,SRC (СМ. ПРИМ. О ФОРМАХ ОПЕРАНДОВ)	(DST)← (DST)-(SRC) ОШИБКИ: I, D, O, U, P
ВЫЧЕСТЬ И ИЗВЛЕЧЬ (ВЕЩЕСТВЕННЫЙ ФОРМАТ)	FSUBP DST,SRC (DST:ST(*i*), SRC:ST)	(ST(*i*)) ← (ST(*i*))-(ST) ИНКРЕМЕНТ ST. ОШИБКИ: I, D, O, U, P

Рис. 11.23. Арифметические команды

ВЫЧЕСТЬ НАОБОРОТ (ВЕЩЕСТВЕННЫЙ ФОРМАТ)	FSUBR //SRC/DST,SRC (СМ. ПРИМ. О ФОРМАХ ОПЕРАНДОВ)	(DST)← (SRC)-(DST) ОШИБКИ: I, D, O, U, P
ВЫЧЕСТЬ НАОБОРОТ И ИЗВЛЕЧЬ (ВЕЩЕСТВЕННЫЙ ФОРМАТ)	FSUBRP DST,SRC (DST:ST(i), SRC:ST)	(ST(i))← (ST)-(ST(i)) ИНКРЕМЕНТ ST. ОШИБКИ: I, D, O, U, P
ВЫЧЕСТЬ (ЦЕЛЫЙ ФОРМАТ)	FISUB SRC (SRC: ОПЕРАНД В ПАМЯТИ КОРОТКОГО ЦЕЛОГО ТИПА ИЛИ ЦЕЛОЕ СЛОВО)	(ST)← (ST)-(SRC) ОШИБКИ: I, D, O, P
ВЫЧЕСТЬ НАОБОРОТ (ЦЕЛЫЙ ФОРМАТ)	FISUBR SRC (SRC: ОПЕРАНД В ПАМЯТИ КОРОТКОГО ЦЕЛОГО ТИПА ИЛИ ЦЕЛОЕ СЛОВО)	(ST)← (SRC)-(ST) ОШИБКИ: I, D, O, P
УМНОЖИТЬ (ВЕЩЕСТВЕННЫЙ ФОРМАТ)	FMUL //SRC/DST,SRC (СМ. ПРИМ. О ФОРМАХ ОПЕРАНДОВ)	(DST)← (DST)*(SRC) ОШИБКИ: I, D, O, U, P
УМНОЖИТЬ И ИЗВЛЕЧЬ (ВЕЩЕСТВЕННЫЙ ФОРМАТ)	FMULP DST,SRC (DST:ST(i), SPC:ST)	(ST(i))← (ST(i))*(ST) ИНКРЕМЕНТ ST. ОШИБКИ: I, D, O, U, P
УМНОЖИТЬ (ЦЕЛЫЙ ФОРМАТ)	FIMUL SRC (SRC: ОПЕРАНД В ПАМЯТИ КОРОТКОГО ЦЕЛОГО ТИПА ИЛИ ЦЕЛОЕ СЛОВО)	(ST)← (ST)*(SRC) ОШИБКИ: I, D, O, P
РАЗДЕЛИТЬ (ВЕЩЕСТВЕННЫЙ ФОРМАТ)	FDIV //SRC/DST,SRC (СМ. ПРИМ. О ФОРМАХ ОПЕРАНДОВ)	(DST)← (DST)/(SRC) ОШИБКИ: I, D, Z, D, U, P
РАЗДЕЛИТЬ И ИЗВЛЕЧЬ (ВЕЩЕСТВЕННЫЙ ФОРМАТ)	FDIVP DST,SRC (DST:ST(i), SRC:ST)	(ST(i))← (ST(i))/(ST) ИНКРЕМЕНТ ST. ОШИБКИ: I, D, Z, O, U, P
РАЗДЕЛИТЬ НАОБОРОТ (ВЕЩЕСТВЕННЫЙ ФОРМАТ)	FDIVR //SRC/DST,SRC (СМ. ПРИМ. О ФОРМАХ ОПЕРАНДОВ)	(DST)← (SRC)/(DST) ОШИБКИ: I, D, Z, D, U, P
РАЗДЕЛИТЬ НАОБОРОТ И ИЗВЛЕЧЬ (ВЕЩЕСТВЕННЫЙ ФОРМАТ)	FDIVRP DST,SRC (DST:ST(i), SRC:ST)	(ST(i))← (ST)/(ST(i)) ИНКРЕМЕНТ ST. ОШИБКИ: I, D, Z, O, U, P
РАЗДЕЛИТЬ (ЦЕЛЫЙ ФОРМАТ)	FIDIV SRC (SRC: ОПЕРАНД В ПАМЯТИ КОРОТКОГО ЦЕЛОГО ТИПА ИЛИ ЦЕЛОЕ СЛОВО)	(ST)← (ST)/(SRC) ОШИБКИ: I, D, Z, O, U, P
РАЗДЕЛИТЬ НАОБОРОТ (ЦЕЛЫЙ ФОРМАТ)	FIDIVR SRC (SRC: ОПЕРАНД В ПАМЯТИ КОРОТКОГО ЦЕЛОГО ТИПА ИЛИ ЦЕЛОЕ СЛОВО)	(ST)← (SRC)/(ST) ОШИБКИ: I, D, Z, O, U, P
НАЙТИ АБСОЛЮТНОЕ ЗНАЧЕНИЕ	FABS (БЕЗ ОПЕРАНДОВ)	(ST) ← |(ST)| ОШИБКИ: I
ИЗМЕНИТЬ ЗНАК	FCHS (БЕЗ ОПЕРАНДОВ)	(ST) ← -(ST) ОШИБКИ: I
НАЙТИ ЧАСТИЧНЫЙ ОСТАТОК	FPREM (БЕЗ ОПЕРАНДОВ)	(ST) ← (ST) MOD (ST(1)) ОШИБКИ: I, D, U
ОКРУГЛИТЬ ДО ЦЕЛОГО	FRNDINT (БЕЗ ОПЕРАНДОВ)	(ST) ← ЦЕЛАЯ ЧАСТЬ (ST) (РЕЖИМ ОКРУГЛЕНИЯ ОПРЕДЕЛЯЕТСЯ ПОЛЕМ RC УПРАВЛЯЮЩЕГО СЛОВА) ОШИБКИ: I, P
МАСШТАБИРОВАТЬ	FSCALE (БЕЗ ОПЕРАНДОВ)	(ST) ← (ST)*2^{n} , ГДЕ n ЦЕЛАЯ ЧАСТЬ (ST(1)) ОШИБКИ: I, O, U
ИЗВЛЕЧЬ КВАДРАТНЫЙ КОРЕНЬ	FSQRT (БЕЗ ОПЕРАНДОВ)	(ST) ← $\sqrt{(ST)}$ ОШИБКИ: I, D, P
ВЫДЕЛИТЬ ПОРЯДОК И МАНТИССУ	FXTRACT (БЕЗ ОПЕРАНДОВ)	(РАБОЧИЙ РЕГИСТР 1)← ПОРЯДОК (ST) (РАБОЧИЙ РЕГИСТР 2)← МАНТИССА (ST) (ST)← (РАБОЧИЙ РЕГИСТР 1) ДЕКРЕМЕНТ ST (ST)← (РАБОЧИЙ РЕГИСТР 2) ОШИБКИ: I

ПРИМЕЧАНИЕ. ЕСЛИ DST НЕ ОПРЕДЕЛЕН, ПОДРАЗУМЕВАЕТСЯ ST И SRC ДОЛЖЕН БЫТЬ ОПЕРАНДОМ В ПАМЯТИ КОРОТКОГО ВЕЩЕСТВЕННОГО ИЛИ ДЛИННОГО ВЕЩЕСТВЕННОГО ТИПОВ. ЕСЛИ ОПРЕДЕЛЕНЫ ОБА ОПЕРАНДА, ИМИ ДОЛЖНЫ БЫТЬ ST,ST(i) ИЛИ ST(i),ST. КОГДА НЕ ОПРЕДЕЛЕНЫ ОБА ОПЕРАНДА, ПОДРАЗУМЕВАЮТСЯ ST(1),ST. А ПОСЛЕ ВЫПОЛНЕНИЯ АРИФМЕТИЧЕСКОЙ ОПЕРАЦИИ ПРОИЗВОДИТСЯ ИЗВЛЕЧЕНИЕ ИЗ СТЕКА И РЕЗУЛЬТАТ ОСТАЕТСЯ В НОВОЙ ВЕРШИНЕ СТЕКА.

Команды сравнения сравнивают содержимое вершины регистрового стека с операндом-источником (регистр или память) и соответственно устанавливают коды условия. Верхний элемент стека можно сравнить с нулем или проанализировать для определения его признака, знака или нормализации. Значения кодов условий можно проверить в памяти, передав туда регистр состояния одной из команд управления процессором 8087. Семь команд из группы сравнения определены на рис 11.24.

НАЗВАНИЕ	МНЕМОНИКА И ФОРМЫ ОПЕРАНДОВ	ОПИСАНИЕ И ВОЗМОЖНЫЕ ОШИБКИ
СРАВНИТЬ (ВЕЩЕСТВЕННЫЙ ФОРМАТ)	FCOM //SRC (SRC:ST(i) ИЛИ ОПЕРАНД В ПАМЯТИ КОРОТКОГО ВЕЩЕСТВЕННОГО ИЛИ ДЛИННОГО ВЕЩЕСТВЕННОГО ТИПА; ЕСЛИ ОПЕРАНД НЕ ОПРЕДЕЛЕН, ПОДРАЗУМЕВАЕТСЯ ST(1).	(ST)-(SRC) И УСТАНОВИТЬ КОДЫ УСЛОВИЯ ПО ПРИМЕЧАНИЮ 1. ОШИБКИ: I, D
СРАВНИТЬ И ИЗВЛЕЧЬ (ВЕЩЕСТВЕННЫЙ ФОРМАТ)	FCOMP //SRC (SRC:ST(i) ИЛИ ОПЕРАНД В ПАМЯТИ КОРОТКОГО ВЕЩЕСТВЕННОГО ИЛИ ДЛИННОГО ВЕЩЕСТВЕННОГО ТИПА; ЕСЛИ ОПЕРАНД НЕ ОПРЕДЕЛЕН, ПОДРАЗУМЕВАЕТСЯ ST(1).	(ST)-(SRC) И УСТАНОВИТЬ КОДЫ УСЛОВИЯ ПО ПРИМЕЧАНИЮ 1. ИНКРЕМЕНТ ST. ОШИБКИ: I, D
СЛОЖИТЬ И ИЗВЛЕЧЬ ДВАЖДЫ (ВЕЩЕСТВЕННЫЙ ФОРМАТ)	FCOMPP (БЕЗ ОПЕРАНДОВ)	(ST)-(ST(1)) И УСТАНОВИТЬ КОДЫ УСЛОВИЯ ПО ПРИМЕЧАНИЮ 1. ИНКРЕМЕНТ ST НА 2. ОШИБКИ: I, D
СРАВНИТЬ (ЦЕЛЫЙ ФОРМАТ)	FICOM SRC (SRC: ОПЕРАНД В ПАМЯТИ КОРОТКОГО ЦЕЛОГО ТИПА ИЛИ ЦЕЛОЕ СЛОВО)	(ST)-(SRC) И УСТАНОВИТЬ КОДЫ УСЛОВИЯ ПО ПРИМЕЧАНИЮ 1. ОШИБКИ: I, D
СРАВНИТЬ И ИЗВЛЕЧЬ (ЦЕЛЫЙ ФОРМАТ)	FICOMP SRC (SRC: ОПЕРАНД В ПАМЯТИ КОРОТКОГО ЦЕЛОГО ТИПА ИЛИ ЦЕЛОЕ СЛОВО)	(ST)-(SRC) И УСТАНОВИТЬ КОДЫ УСЛОВИЯ ПО ПРИМЕЧАНИЮ 1. ИНКРЕМЕНТ ST. ОШИБКИ: I, D
ПРОВЕРИТЬ ВЕРШИНУ СТЕКА	FTST (БЕЗ ОПЕРАНДОВ)	(ST)-0.0 И УСТАНОВИТЬ КОДЫ УСЛОВИЯ ПО ПРИМЕЧАНИЮ 2. ОШИБКИ: I, D
ПРОАНАЛИЗИРОВАТЬ ВЕРШИНУ СТЕКА	FXAM (БЕЗ ОПЕРАНДОВ)	УСТАНОВИТЬ КОДЫ УСЛОВИЙ В СООТВЕТСТВИИ С (ST) ПО ПРИМЕЧАНИЮ 3.

ПРИМЕЧАНИЕ 1:

ОТНОШЕНИЕ	C3	C0
(ST) > (SRC)	0	0
(ST) < (SRC)	0	1
(ST) = (SRC)	1	0
НЕ СРАВНИМЫ	1	1

ПРИМЕЧАНИЕ 2:

ОТНОШЕНИЕ	C3	C0
(ST) > 0.0	0	0
(ST) < 0.0	0	1
(ST) = 0.0	1	0
НЕ СРАВНИМЫ	1	1

ПРИМЕЧАНИЕ 3:

(ST)	C3	C2	C1	C0
+НЕНОРМАЛИЗОВАННОЕ	0	0	0	0
+НЕ ЧИСЛО	0	0	0	1
-НЕНОРМАЛИЗОВАННОЕ	0	0	1	0
-НЕ ЧИСЛО	0	0	1	1
+НОРМАЛИЗОВАННОЕ	0	1	0	0
+БЕСКОНЕЧНОСТЬ	0	1	0	1
-НОРМАЛИЗОВАННОЕ	0	1	1	0
-БЕСКОНЕЧНОСТЬ	0	1	1	1
+0	1	0	0	0
ПУСТОЕ	1	0	0	1
-0	1	0	1	0
ПУСТОЕ	1	0	1	1
+ДЕНОРМАЛИЗОВАННОЕ	1	1	0	0
ПУСТОЕ	1	1	0	1
-ДЕНОРМАЛИЗОВАННОЕ	1	1	1	0
ПУСТОЕ	1	1	1	1

Рис. 11.24. Команды сравнения

Пять трансцедентных команд (рис. 11.25) вычисляют $\mathrm{tg}\,\theta$ $(0 < \theta < \pi/4)$, $\mathrm{arctg}\,(Y/X)$ $(0 < Y < X < \infty)$, $2^X - 1$ $(0 \leqslant X \leqslant 0{,}5)$, $Y*\log_2 X$ и $Y*\log_2 (X+1)$. В качестве операнда(ов) используются ST или ST(1), а результаты запоминаются в стеке. Другие тригонометрические, обратные тригонометрические,

гиперболические, обратные гиперболические, логарифмические и показательные функции можно реализовать из пяти приведенных функций, пользуясь математическими тождествами. Например, вычисление натурального логарифма X эквивалентно нахождению значения $(1/\log_2 e)\log_2 X$.

НАЗВАНИЕ	МНЕМОНИКА	ОПИСАНИЕ И ВОЗМОЖНЫЕ ОШИБКИ
ВЫЧИСЛИТЬ 2^X-1	F2XM1 (БЕЗ ОПЕРАНДОВ)	В ПРЕДПОЛОЖЕНИИ, ЧТО $0 \leqslant (ST) \leqslant 0.5$, $(ST) \leftarrow 2^{(ST)}-1$ ОШИБКИ: U, P
ВЫЧИСЛИТЬ ЧАСТИЧНЫЙ АРКТАНГЕНС	FPATAN (БЕЗ ОПЕРАНДОВ)	В ПРЕДПОЛОЖЕНИЙ, ЧТО $0 < (ST(1)) < (ST) < \infty$ (РАБОЧИЙ РЕГИСТР 1) $\leftarrow$ ARCTG [(ST(1))/(ST)] ИНКРЕМЕНТ ST (ST) $\leftarrow$ (РАБОЧИЙ РЕГИСТР 1) ОШИБКИ: U, P
ВЫЧИСЛИТЬ ЧАСТИЧНЫЙ ТАНГЕНС	FPTAN (БЕЗ ОПЕРАНДОВ)	В ПРЕДПОЛОЖЕНИИ, ЧТО $0 < (ST) < \pi/4$, (ST) $\leftarrow$ ЧИСЛИТЕЛЬ TG(ST) ДЕКРЕМЕНТ ST (ST) $\leftarrow$ ЗНАМЕНАТЕЛЬ TG(ST) ОШИБКИ: I, P
ВЫЧИСЛИТЬ $Y*\log_2 X$	FYL2X (БЕЗ ОПЕРАНДОВ)	В ПРЕДПОЛОЖЕНИИ, ЧТО $0 < (ST) < \infty$ И $-\infty < (ST(1)) < \infty$, (РАБОЧИЙ РЕГИСТР 1) $\leftarrow$ $(ST(1))*\log_2[(ST)]$ ИНКРЕМЕНТ ST (ST) $\leftarrow$ (РАБОЧИЙ РЕГИСТР 1) ОШИБКИ: P
ВЫЧИСЛИТЬ $Y*\log_2[X+1]$	FYL2XP1 (БЕЗ ОПЕРАНДОВ)	В ПРЕДПОЛОЖЕНИИ, ЧТО $0 < [(ST)] < 1-\sqrt{2}/2$ И $-\infty < [(ST(1))] < \infty$, (РАБОЧИЙ РЕГИСТР) $\leftarrow$ $(ST(1))*\log_2[(ST)+1]$ ИНКРЕМЕНТ ST (ST) $\leftarrow$ (РАБОЧИЙ РЕГИСТР) ОШИБКИ: P

Рис. 11.25. Трансцендентные команды

Как пример на тригонометрические функции рассмотрим команду частичного тангенса FPTAN, которая вычисляет $\operatorname{tg}\theta$, где θ измеряется в радианах и находится в ST. Результатом является отношение Y/X, причем Y заменяет θ, а X включается в стек. Функция синуса связана с функцией тангенса следующим образом:

$$\sin\theta = \frac{\operatorname{tg}\theta}{\sqrt{1+\operatorname{tg}^2\theta}} .$$

Когда $0 < \theta < \pi/4$, значение $\sin\theta$ вычисляется по формуле

$$\frac{Y}{\sqrt{X^2+Y^2}} .$$

Если θ находится вне допустимого диапазона, необходимо с помощью какой-либо процедуры привести θ в диапазон $0 \ldots \pi/4$. Например, для приведения угла из диапазона $\pi/4 \ldots \pi/2$ к допустимому диапазону следует воспользоваться командой FPREM и тождеством (см. упр. 10):

$$\operatorname{tg}\theta = 1/\operatorname{tg}(\pi/2-\theta).$$

В процессоре 8087 имеется семь команд для формирования констант. Они применяются для включения в стек значений +0,0, +1,0, π, $\log_2 10$, $\log_2 e$, $\log_{10} e$ и $\log_{10} 2$. Константы хранятся во временном вещественном формате и

имеют точность примерно в 19 десятичных цифр. Команды формирования констант приведены на рис. 11.26.

НАЗВАНИЕ	МНЕМОНИКА	ОПИСАНИЕ И ВОЗМОЖНЫЕ ОШИБКИ
ЗАГРУЗИТЬ НУЛЬ	FLDZ (БЕЗ ОПЕРАНДОВ)	ДЕКРЕМЕНТ ST (ST) ← 0.0 ОШИБКИ: I
ЗАГРУЗИТЬ ЕДИНИЦУ	FLD1 (БЕЗ ОПЕРАНДОВ)	ДЕКРЕМЕНТ ST (ST) ← 1.0 ОШИБКИ: I
ЗАГРУЗИТЬ π	FLDPI (БЕЗ ОПЕРАНДОВ)	ДЕКРЕМЕНТ ST (ST) ← π ОШИБКИ: I
ЗАГРУЗИТЬ $\log_2 e$	FLDL2E (БЕЗ ОПЕРАНДОВ)	ДЕКРЕМЕНТ ST (ST) ← $\log_2 e$ ОШИБКИ: I
ЗАГРУЗИТЬ $\log_2 10$	FLDL2T (БЕЗ ОПЕРАНДОВ)	ДЕКРЕМЕНТ ST (ST) ← $\log_2 10$ ОШИБКИ: I
ЗАГРУЗИТЬ $\log_{10} 2$	FLDLG2 (БЕЗ ОПЕРАНДОВ)	ДЕКРЕМЕНТ ST (ST) ← $\log_{10} 2$ ОШИБКИ: I
ЗАГРУЗИТЬ $\log_e 2$	FLDLN2 (БЕЗ ОПЕРАНДОВ)	ДЕКРЕМЕНТ ST (ST) ← $\log_e 2$ ОШИБКИ: I

Рис. 11.26. Команды загрузки констант

Последняя группа команд представлена на рис. 11.27. Они предназначены для манипуляций регистрами управления и состояния, запоминания и восстановления различных компонент состояния процессора. Обычно эти команды требуются в подпрограммах и процедурах прерываний, которым необходим доступ к процессору 8087, а также при обработке ошибок и переключении процессов. Команда инициализация переводит процессор 8087 в начальное состояние и обычно требуется до использования процессора. Несколько команд загружают в регистр управления новое содержимое, допуская динамическое изменение управляющих бит в связи с изменяющимися обстоятельствами.

НАЗВАНИЕ	МНЕМОНИКА И ФОРМЫ ОПЕРАНДОВ [1]	ОПИСАНИЕ [2]
ИНИЦИАЛИЗИРОВАТЬ ПРОЦЕССОР	FINIT/FNINIT (БЕЗ ОПЕРАНДОВ)	ПРОИЗВОДИТСЯ СБРОС ПРОЦЕССОРА 8087. В УПРАВЛЯЮЩЕЕ СЛОВО ЗАГРУЖАЕТСЯ 03FF, А РЕГИСТР ПРИЗНАКОВ ОТМЕНАЕТСЯ КАК ПУСТОЙ. СБРАСЫВАЮТСЯ ФЛАЖКИ ВСЕХ ОШИБОК, ЗАНЯТОСТИ И ПРЕРЫВАНИЯ. УКАЗАТЕЛЬ СТЕКА ST СБРАСЫВАЕТСЯ НА 0
ЗАПРЕТИТЬ ПРЕРЫВАНИЯ	FDISI/FNDISI (БЕЗ ОПЕРАНДОВ)	БИТ МАСКИ ПРЕРЫВАНИЯ УСТАНАВЛИВАЕТСЯ В 1
РАЗРЕШИТЬ ПРЕРЫВАНИЯ	FENI/FNENI (БЕЗ ОПЕРАНДОВ)	БИТ МАСКИ ПРЕРЫВАНИЯ СБРАСЫВАЕТСЯ

[1] АЛЬТЕРНАТИВНЫЕ МНЕМОНИКИ СО ВТОРЫМ СИМВОЛОМ N ПРЕДНАЗНАЧЕНЫ ДЛЯ ИСПОЛЬЗОВАНИЯ В СИТУАЦИЯХ, КОГДА КОМАНДА WAIT НЕ НУЖНА. ПОЭТОМУ АССЕМБЛЕР НЕ ВВОДИТ КОМАНДУ WAIT ПЕРЕД КОМАНДАМИ С ТАКИМИ МНЕМОНИКАМИ.

[2] ОШИБКИ НЕВОЗМОЖНЫ.

Рис. 11.27. Команды управления процессором

СБРОСИТЬ ФЛАЖКИ ОШИБОК	FCLEX/FNCLEX (БЕЗ ОПЕРАНДОВ)	СБРАСЫВАЮТСЯ БИТЫ B, IR, P, U, O, Z, D И I В РЕГИСТРЕ СОСТОЯНИЯ
ИНКРЕМЕНТ УКАЗАТЕЛЯ СТЕКА	FINCSTP (БЕЗ ОПЕРАНДОВ)	ST⟵ ST+1 (НЕ УСТАНАВЛИВАЕТ ПРИЗНАК РЕГИСТРА ПРЕДЫДУЩЕЙ ВЕРШИНЫ СТЕКА КАК ПУСТОЙ)
ДЕКРЕМЕНТ УКАЗАТЕЛЯ СТЕКА	FDECSTP (БЕЗ ОПЕРАНДОВ)	ST⟵ ST-1
ЗАПОМНИТЬ РЕГИСТР СОСТОЯНИЯ	FSTSW/FNSTSW DST (DST: ДВУХБАЙТНЫЙ ОПЕРАНД В ПАМЯТИ)	(DST)⟵ (РЕГИСТР СОСТОЯНИЯ)
ЗАПОМНИТЬ РЕГИСТР УПРАВЛЕНИЯ	FSTCW/FNSTCW DST (DST: ДВУХБАЙТНЫЙ ОПЕРАНД В ПАМЯТИ)	(DST)⟵ (РЕГИСТР УПРАВЛЕНИЯ)
ЗАГРУЗИТЬ РЕГИСТР УПРАВЛЕНИЯ	FLDCW SRC (DST: ДВУХБАЙТНЫЙ ОПЕРАНД В ПАМЯТИ)	(РЕГИСТР УПРАВЛЕНИЯ)⟵ (SRC)
ЗАПОМНИТЬ СРЕДУ	FSTENV/FNSTENV DST (DST: 14-БАЙТНЫЙ ОПЕРАНД В ПАМЯТИ)	(DST)⟵ (РЕГИСТР УПРАВЛЕНИЯ) (DST+2)⟵ (РЕГИСТР СОСТОЯНИЯ) (DST+4)⟵ (РЕГИСТР ПРИЗНАКОВ) (DST+6)⟵ (УКАЗАТЕЛЬ КОМАНДЫ) (DST+10)⟵ (УКАЗАТЕЛЬ ОПЕРАНДА) УСТАНАВЛИВАЮТСЯ МАСКИ ВСЕХ ОШИБОК
ЗАГРУЗИТЬ СРЕДУ	FLDENV SRC (SRC: 14-БАЙТНЫЙ ОПЕРАНД В ПАМЯТИ)	(РЕГИСТР УПРАВЛЕНИЯ)⟵ (SRC) (РЕГИСТР СОСТОЯНИЯ)⟵ (SRC+2) (РЕГИСТР ПРИЗНАКОВ)⟵ (SRC+4) (УКАЗАТЕЛЬ КОМАНДЫ)⟵ (SRC+6) (УКАЗАТЕЛЬ ОПЕРАНДА)⟵ (SRC+10)
ЗАПОМНИТЬ СОСТОЯНИЕ	FSAVE/FNSAVE DST (DST: 94-БАЙТНЫЙ ОПЕРАНД В ПАМЯТИ)	В DST ЗАПОМИНАЮТСЯ РЕГИСТР УПРАВЛЕНИЯ, РЕГИСТР СОСТОЯНИЯ, РЕГИСТР ПРИЗНАКОВ, УКАЗАТЕЛЬ КОМАНДЫ, УКАЗАТЕЛЬ ОПЕРАНДА, ST, ST(1), ST(2), ST(3), ST(4), ST(5), ST(6) И ST(7). ПРОЦЕССОР ИНИЦИАЛИЗИРУЕТСЯ
ЗАГРУЗИТЬ СОСТОЯНИЕ	FRSTOR SRC (SRC: 94-БАЙТНЫЙ ОПЕРАНД В ПАМЯТИ)	В SRC ЗАГРУЖАЮТСЯ РЕГИСТР УПРАВЛЕНИЯ, РЕГИСТР СОСТОЯНИЯ, РЕГИСТР ПРИЗНАКОВ, УКАЗАТЕЛЬ КОМАНДЫ, УКАЗАТЕЛЬ ОПЕРАНДА, ST, ST(1), ST(2), ST(3), ST(4), ST(5), ST(6) И ST(7)
ОСВОБОДИТЬ РЕГИСТР	FFREE SRC (DST: ST(*i*))	ПРИЗНАК ST(*i*) ОТМЕЧАЕТСЯ КАК ПУСТОЙ
ХОЛОСТАЯ КОМАНДА	FNOP (БЕЗ ОПЕРАНДОВ)	(ST)⟵ (ST)

По различным причинам приходится заносить в память и восстанавливать отдельные компоненты текущего состояния процессора 8087. При этом наиболее часто необходимы следующие три действия:

1. Запомнить только регистр состояния (FSTSW) – применяется, когда ЦП должен проверить установку кодов условий после команды сравнения.

2. Запомнить процессорную среду, включающую в себя регистры управления, состояния и признаков, указатели команды и операнда (FSTENV), – применяется в процедуре обработки ошибки для идентификации причины ошибки.

3. Запомнить все состояние процессора, включающее в себя процессорную среду и регистровый стек (FSAVE), – применяется в подпрограммах, процедурах прерываний и при переключении процессов.

Чтобы окончить запросы процессора 8087, команды FSTENV и FSAVE после запоминания регистров в памяти устанавливают все маски ошибок в 1 (посредством инициализации). Команда FSAVE также устанавливает в 1 бит

маски разрешения прерываний IEM, так что все прерывания запрещаются. Запомненный образ можно восстановить командами FLDENV и FRSTOR. Однако в процедуре обработки ошибок необходимо установить все запомненные маски ошибок в 1 до загрузки состояния в процессор 8087. В противном случае сразу же возникает прерывание.

11.3.4. ПРИМЕР

В статистике очень важной мерой наблюдаемых случайных выборок служит стандартное отклонение, показывающее их разброс. Для заданного множества выборок $X_1, X_2, \ldots, X_N$ стандартное отклонение определяется как

$$\sqrt{\sum_{i=1}^{N} (X_i - \mathrm{MEAN})^2 / (N-1)},$$

где N – число выборок, а

$$\mathrm{MEAN} = \frac{\sum_{i=1}^{N} X_i}{N}$$

– выборочное среднее.

Предположим, что выборки введены, преобразованы в формат длинных вещественных данных и запомнены в массиве с начальным адресом ARRAY_X, а переменная N содержит число выборок. На рис. 11.28 приведена программа вычисления выборочного среднего и стандартного отклонения и запоминания их в MEAN и STD_DEV, соответственно.

```
ДИРЕКТИВЫ И КОМАНДЫ ОПРЕДЕЛЕНИЯ СЕГМЕНТОВ ДАННЫХ, ИНИЦИАЛИЗАЦИИ РЕГИСТРОВ,
ВВОДА И ПРЕОБРАЗОВАНИЯ ЧИСЕЛ

            FINIT                   ;УСТАНОВИТЬ ВСЕ МАСКИ ОШИБОК НА 1
            FLDZ                    ;ВКЛЮЧИТЬ 0.0 В ST - АККУМУЛЯТОР
            XOR     BX,BX           ;СБРОСИТЬ BX - ИНДЕКС
            MOV     CX,N            ;ИСПОЛЬЗОВАТЬ CX КАК СЧЕТЧИК
LOOPMEAN:   FADD    ARRAY_X[BX]     ;ПРИБАВИТЬ X(I) К ST
            ADD     BX,8            ;ПРОДВИНУТЬ НА СЛЕДУЮЩИЙ ЭЛЕМЕНТ
            LOOP    LOOPMEAN        ;ПОВТОРИТЬ N РАЗ; В ST - СУММА X(I)
            FIDIV   N               ;(ST) = (СУММА X(I))/N
            FST     MEAN            ;ЗАПОМНИТЬ СРЕДНЕЕ
            FLDZ                    ;ВКЛЮЧИТЬ 0.0 В ST
            XOR     BX,BX           ;СБРОСИТЬ ИНДЕКС
            MOV     CX,N            ;В CX СЧЕТЧИК
LOOPSTD:    FLD     ARRAY_X[BX]     ;ВКЛЮЧИТЬ В СТЕК X(I)
            ADD     BX,8            ;ПРОДВИНУТЬ УКАЗАТЕЛЬ
            FSUB    ST,ST(2)        ;(ST) =  X(I)-СРЕДНЕЕ
            FMUL    ST,ST(0)        ;(ST) = (X(I)-СРЕДНЕЕ)**2
            FADDP   ST(1),ST
            LOOP    LOOPSTD         ;(ST) = СУММА (X(I)-СРЕДНЕЕ)**2
            FLD1                    ;ВКЛЮЧИТЬ 1 В ST
            FISUBR  N               ;(ST) = N-1.0
            FDIVR   ST,ST(1)        ;(ST) = (СУММА (X(I)-СРЕДНЕЕ)**2)/(N-1)
            FSQRT                   ;(ST) = СТАНДАРТНОЕ ОТКЛОНЕНИЕ
            FST     STD_DEV         ;ЗАПОМНИТЬ (ST) В STD_DEV

КОМАНДЫ ВЫВОДА И ПРЕОБРАЗОВАНИЯ ЧИСЕЛ
```

Рис. 11.28. Программа вычисления выборочного среднего и стандартного отклонения

Первая команда инициализирует процессор 8087: все регистры отмечаются как пустые, флажки ошибок и прерываний сбрасываются, маски ошибок устанавливаются, ST устанавливается на 0, а также задается управление округлением, точностью и бесконечностью по умолчанию. Если этот фраг-

мент является частью процедуры, перед инициализацией необходимо запомнить состояние вызывающей программы.

Процессор 8087 выполняет зацикливание, условные переходы и индексирование с помощью ЦП. При правильном проектировании программы удается обеспечить максимально параллельную работу обоих процессоров. Например, лучше поместить команды XOR и MOV после команды FLDZ, так как время выполнения команды FINIT меньше, чем команды FLDZ.

Важно отметить, что операнд из памяти можно загрузить в процессор 8087 только операцией включения в стек. Следовательно, после каждой итерации цикла, начинающегося в LOOPSTD, первые три регистра от вершины стека содержат $(X_i - MEAN)^2$, частичную сумму $\sum_{i=1}^{N} (X_i - MEAN)^2$ и содержимое MEAN. Чтобы загрузить X_{i+1} в тот же самый верхний элемент стека в начале следующей итерации, необходимо воспользоваться командой сложения с извлечением из стека (FADDP). Эта команда удаляет $(X_i - MEAN)^2$ после того, как оно прибавляется к ST(1).

11.4. ПРОЦЕССОР ВВОДА-ВЫВОДА

Контроллеры и интерфейсные микросхемы, рассмотренные в гл. 9, значительно упрощают проектирование интерфейсов, но за исключением передач ПДП подготовку и саму передачу данных осуществляет ЦП. Для быстродействующих устройств данные передаются с применением ПДП, но все же ЦП должен подготовить контроллер устройства, инициировать передачу ПДП и контролировать состояние по завершению каждой операции ПДП. На рис. 11.29, *а* показана схема общего взаимодействия, когда вводом-выводом управляет ЦП.

Процессор ввода-вывода (ПВВ) 8089 специально предназначен для эффективного управления вводом-выводом. В отличие от контроллера ПДП он может выбирать и выполнять свои команды. Эти команды ориентированы на операции ввода-вывода, но кроме передач данных, они могут выполнять арифметические и логические операции, переходы, поиск и преобразование.

Центральный процессор взаимодействует с ПВВ посредством управляющих блоков в памяти. Он готовит управляющие блоки, которые описывают подлежащую выполнению задачу, а затем диспетчирует задачу ПВВ сигналом, напоминающим прерывание. Процессор ввода-вывода считывает управляющие блоки для локализации так называемой *канальной программы*, которая написана в командах ПВВ. Затем ПВВ выполняет предназначенную ему задачу, выбирая и выполняя команды канальной программы. Когда она завершается, ПВВ извещает ЦП посредством прерывания или модификации ячейки состояния в памяти.

Как показано на рис. 11.29, *б*, ПВВ выполняет все действия передач ввода-вывода, включая настройку устройства, программный ввод-вывод и операции ПДП, освобождая ЦП от операций ввода-вывода. В такой конфигурации на ЦП возлагаются задачи более высокого уровня, а ПВВ специализируется

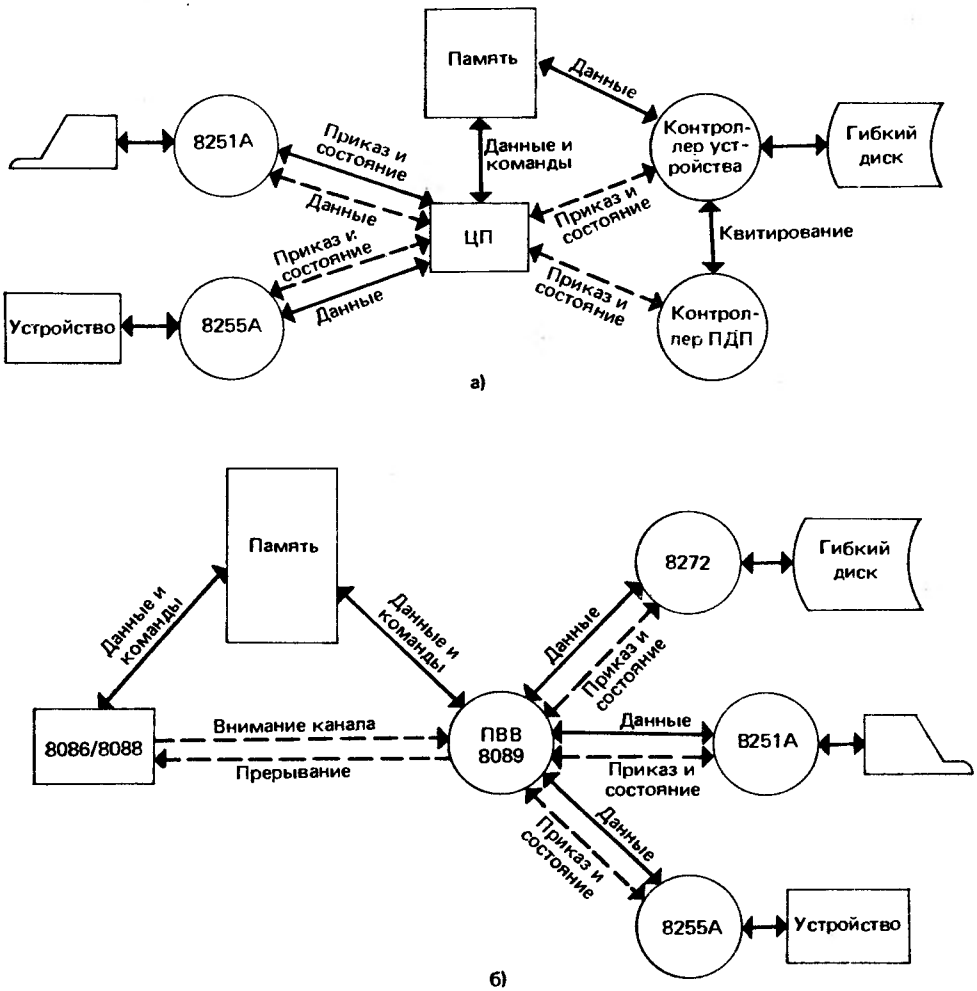


Рис. 11.29. Подсистемы ввода-вывода, когда вводом-выводом управляет микропроцессор (*а*) и ПВВ 8089 (*б*)

на реализации ввода-вывода. Распределенная обработка упрощает разработку аппаратных и программных средств и улучшает производительность и гибкость системы. Процессор ввода-вывода может работать в локальной (сильно связанной) или в дистанционной (слабо связанной) конфигурации. Как было показано на рис. 11.7 и 11.8, в локальной конфигурации ПВВ разделяет с главным процессором шинный интерфейс с помощью линий $\overline{RQ}/GT$. Доступ ко всем ресурсам осуществляется через системную шину.

В дистанционной конфигурации ПВВ имеет локальную шину ввода-вывода, а для доступа к разделенной системной шине необходимы арбитр шины,

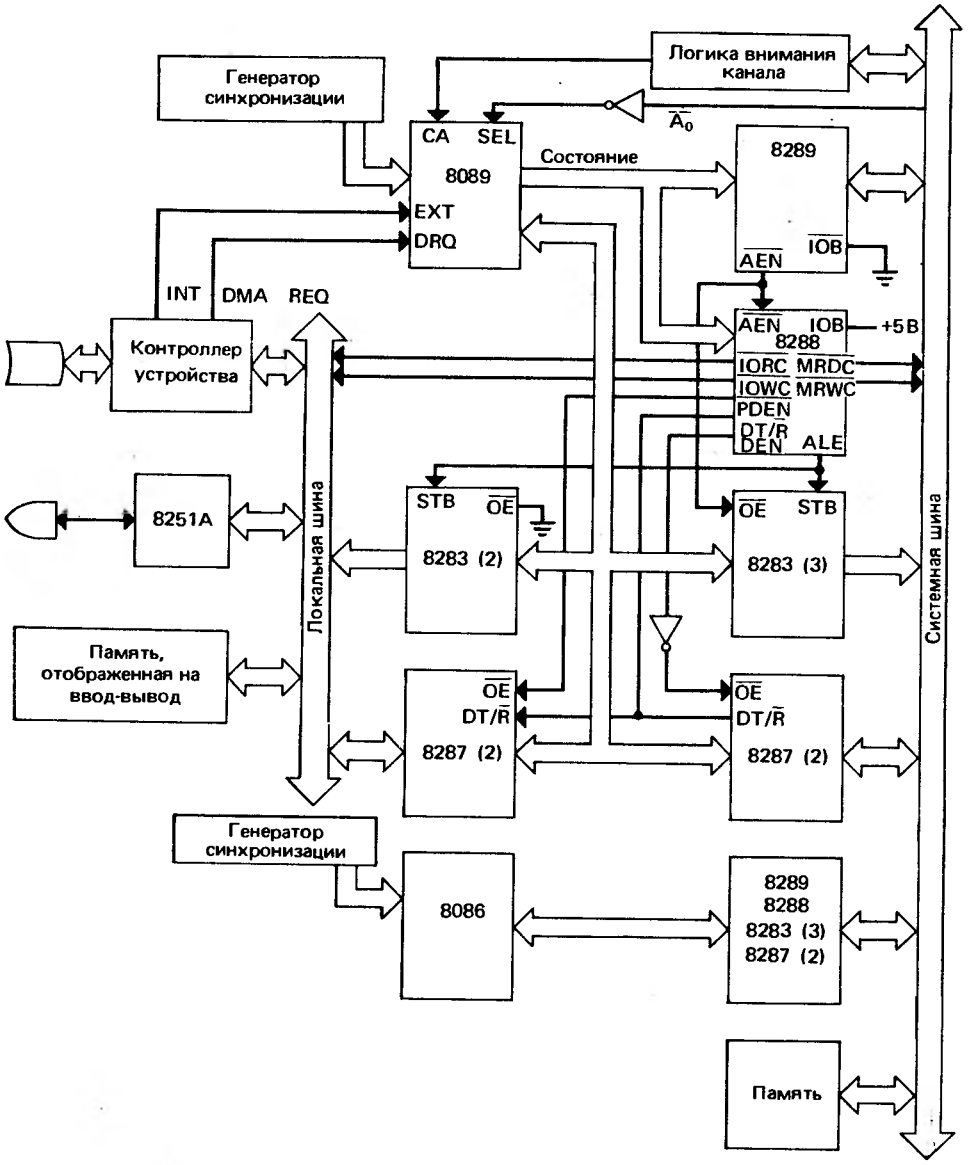


Рис. 11.30. Дистанционная конфигурация

контроллер шины, регистры-защелки адреса и приемопередатчики данных (рис. 11.30). Линию $\overline{RQ}/\overline{GT}$ ПВВ можно использовать для взаимодействия с другим ПВВ, который действует в качестве ведомого и разделяет шины с главным ПВВ. Процессор ввода-вывода обращается к выделенным для него

устройствам ввода-вывода по локальной шине, а с ЦП взаимодействует через системную память. Быстродействующий контроллер запрашивает передачу по одной из двух линий DRQ, а заканчивает передачу ПДП сигналом по одному из двух входов EXT. Для сокращения нагрузки на системную шину и увеличения степени параллельности локальная память должна хранить канальные программы и организовывать буферные области. Однако локальная память призвана реагировать на приказы шины ввода-вывода, а не на приказы считывания и записи в память (т. е. она должна быть памятью, отображенной на ввод-вывод). В отличие от микропроцессоров 8086/8088 шина ввода-вывода ПВВ не обязательно имеет ту же ширину данных, что и шина памяти. Это позволяет ПВВ передавать данные из 8-битного источника в 16-битный получатель и наоборот. Так как шина ввода-вывода имеет всего 16 линий адреса, емкость локального пространства (пространства ввода-вывода) составляет только 64К байт. С другой стороны, системное пространство (пространство памяти), которое адресуется системной шиной, имеет емкость 1М байт. Подчеркнем, что команды ПВВ обращаются к портам ввода-вывода, используя те же режимы адресации, что и для операндов в памяти. Принадлежность адреса к пространству ввода-вывода или к системному пространству определяется битом признака используемого регистра-указателя.

11.4.1. АРХИТЕКТУРА ПРОЦЕССОРА ВВОДА-ВЫВОДА

Внутренняя структура ПВВ показана на рис. 11.31. Каждый из двух каналов программируется и работает независимо, но оба они разделяют логику управления и АЛУ. Указатель управления каналом (ССР) пользователю недоступен. Он хранит адрес управляющего блока СВ канала 1 во время последовательности инициализации. Управляющий блок канала 2 начинается по адресу, равному содержимому ССР плюс 8. Для диспетчирования задачи любому каналу ЦП выдает сигнал внимания вместе с сигналом SEL, который выбирает канал 1 (SEL = 0) или канал 2 (SEL = 1). Так как каналы занимают два смежных адреса портов ввода-вывода, на вход SEL подключается линия адреса A0.

Каждый канал имеет идентичные наборы регистров, причем каждый набор делится на две группы в соответствии с длиной регистров. Указательная группа содержит регистры длиной 20 бит, а регистровая группа — 16 бит. С каждым указателем, кроме PP, связан бит признака (тзга). При обращении к операнду в памяти бит признака показывает, что представляет собой содержимое данного указателя: 20-битный адрес системного пространства (памяти) или 16-битный адрес локального пространства (ввода-вывода). В первом случае признак равен 0, а во втором — 1. При обращении к локальному пространству в качестве адреса используются 16 младших бит указателя. Регистр PP всегда содержит адрес в системном пространстве.

Регистры GA, GB, GC, BC, IX и MC можно использовать в канальных программах как регистры общего назначения в арифметических и логических операциях. Кроме того, при адресации операндов в памяти и выполнении операций ПДП они выполняют специальные функции. Операнд в памяти разре-

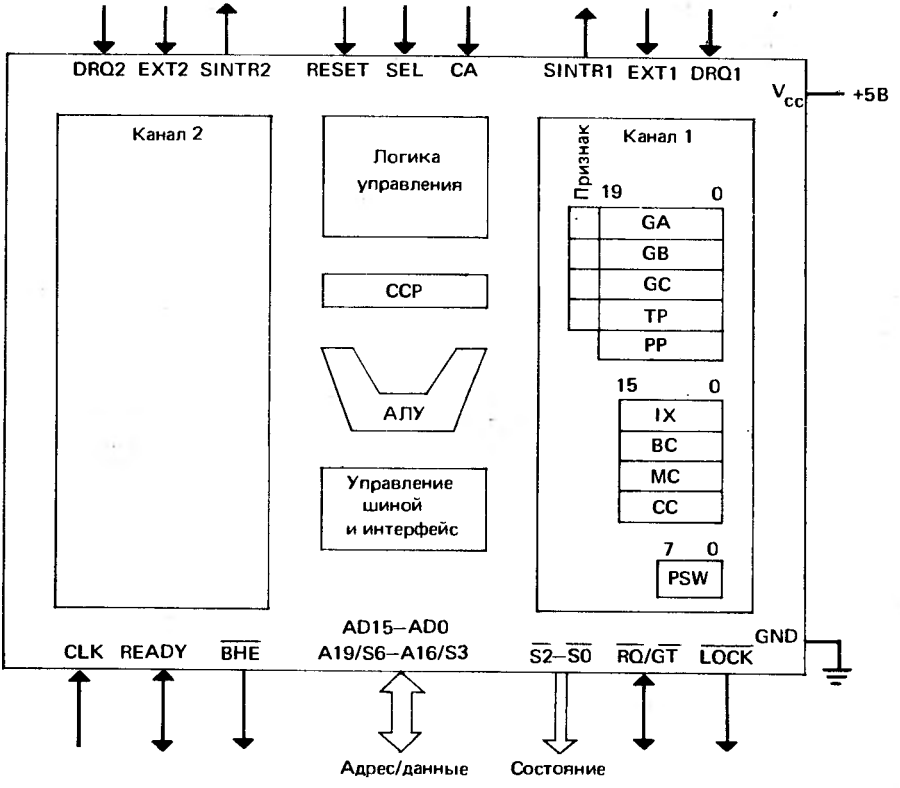


Рис. 11.31. Схема процессора ввода-вывода 8089

шается адресовать, используя в качестве базового регистра один из указателей GA, GB, GC или PP. В операции ПДП регистры GA и GB служат указателями источника и получателя. Если GA адресует источник, GB адресует получатель и наоборот. Когда вместе с передачей ПДП осуществляется операция преобразования, содержимое GC используется как базовый адрес 256-байтной таблицы преобразования. В передаче ПДП регистр BC служит счетчиком байт; после передачи байта производится декремент BC на 1, а после передачи слова — на 2. В операции маскированного сравнения регистр MC содержит в битах 7-0 сравниваемый двоичный набор, а в битах 15-8 — маску. Маскированное сравнение выполняется в соответствии с выражением

$$((\text{байт операнда}) \oplus (\text{MC})_{7-0}) \wedge (\text{MC})_{15-8}.$$

Результат можно использовать как условие окончания ПДП или как условие перехода в команде перехода по маскированному сравнению. Регистр IX используется как индексный регистр. В двух режимах адресации операнда в памяти его содержимое суммируется с содержимым базового регистра, образуя адрес операнда.

Указатель задачи TP хранит адрес следующей выполняемой команды и эквивалентен программному счетчику PC в ЦП. Он также имеет бит признака, показывающий, где находится следующая команда: в системном пространстве или в пространстве ввода-вывода. Указатель параметров PP не программируется пользователем, но он автоматически загружается процессором ввода-вывода при инициализации задачи. Он содержит адрес блока параметров, который рассматривается далее.

Каждый канал имеет также 8-битный регистр состояния программы PSW, который содержит текущее состояние канала. Состояние показывает ширину шины источника и получателя, действие канала, управление прерываниями, предел загрузки шины и приоритет. Пользователь не может оперировать PSW, но его может модифицировать канальный приказ. Когда канальная программа приостанавливается, PSW запоминается вместе с указателем TP и четырьмя битами признаков в первых двух словах блока параметров. Это позволяет каналу возобновить приостановленную канальную программу при поступлении приказа возобновления.

Одной из уникальных возможностей ПВВ является выполнение передач ПДП со множеством вариантов. Можно определить направление передачи память – память, ввод-вывод – ввод-вывод или память – ввод-вывод, а также ввод-вывод – память. В каждой передаче ПВВ считывает байт или слово, запоминает данные в получателе и соответственно модифицирует регистры GA, GB и BC. Если данные передаются из 8-битного источника в 16-битный получатель, ПВВ может считать два байта и запомнить их как одно слово. Когда же передача осуществляется в противоположном направлении, перед засылкой в получатель слово разделяется на два байта. Между циклами считывания и запоминания можно сравнить или преобразовать данные. Операцию ПДП можно окончить по внешнему запросу, по соответствию или несоответствию при маскированном сравнении или по достижению счетчиком нуля. Эти варианты определяются содержимым регистра управления канала CC, который имеет следующий формат:

1. Управление функцией (биты 15 и 14). Определяет один из четырех режимов передач данных: память – память (11), порт ввода-вывода в память (10), память в порт ввода-вывода (01), порт ввода-вывода в порт ввода-вывода (00). При передаче память – память производится автоинкремент указателей источника и получателя, но во время и после передачи ввод-вывод – ввод-вывод оба указателя не изменяются. В большинстве обычных контроллеров ПДП такие режимы отсутствуют. Они удобны при пересылке блока кода или данных из одной области памяти в другую, а также при прямом взаимодействии устройств.

2. Режим преобразования (бит 13). Показывает, что при передаче ПДП байты данных преобразуются с помощью 256-байтной таблицы (если бит содержит 1). Базовый адрес таблицы преобразования должен находиться в регистре GC.

3. Управление синхронизацией (биты 12 и 11). Определяет, каким образом синхронизируется передача данных. В несинхронизированной передаче (00) следующий цикл передачи начинается, как только будет доступной ши-

на. Передача с синхронизацией источником (01) запускает операцию считывания следующего цикла передачи при получении сигнала DRO; передача с синхронизацией получателем (10) инициирует операцию записи следующего цикла передачи при получении сигнала DRQ.

4. Индикатор источника/получателя (бит 10). Определяет регистр GA как указатель источника (0) или как указатель получателя (1). В любом случае регистр GB служит другим указателем.

5. Управление блокировкой (бит 9). Формирует в цикле передачи ПДП процессора ввода-вывода активный сигнал $\overline{\text{LOCK}}$ (если бит содержит 1).

6. Управление зацеплением (бит 8). Придает другому каналу наивысший приоритет (этот бит не обязательно используется в операции ПДП).

7. Режим однократной передачи (бит 7). Оканчивает ПДП после одной передачи (если бит содержит 1), а затем вызывает выполнение следующей команды, адресуемой ТР.

8. Управление окончанием (биты 6-0). Определяет, каким образом заканчивается ПДП и откуда выбирать следующую команду после окончания операции ПДП. Как показано на рис. 11.32, передачу ПДП можно окончить после текущего цикла передачи с учетом внешнего управления (биты 6 и 5), счетчика байт (биты 4 и 3), сравнения (биты 2, 1 и 0) или комбинации перечисленных условий. Если задано внешнее управление, канал оканчивает ПДП при активном сигнале EXT (внешнее окончание). Когда указан счетчик байт, передача ПДП заканчивается при достижении нуля в регистре BC данного канала. Биты 2-0 определяют окончание ПДП по результату сравнения с получением соответствия или несоответствия.

УПРАВЛЯЮЩИЕ БИТЫ CC		УСЛОВИЕ ОКОНЧАНИЯ И СМЕЩЕНИЕ
БИТ 6	БИТ 5	
0	0	НЕТ ВНЕШНЕГО ОКОНЧАНИЯ
0	1	ОКАНЧИВАЕТСЯ ПРИ ВЫСОКОМ УРОВНЕ EXT; СМЕЩЕНИЕ УСТАНАВЛИВАЕТСЯ НА 0
1	0	ОКАНЧИВАЕТСЯ ПРИ ВЫСОКОМ УРОВНЕ EXT; СМЕЩЕНИЕ УСТАНАВЛИВАЕТСЯ НА 4
1	1	ОКАНЧИВАЕТСЯ ПРИ ВЫСОКОМ УРОВНЕ EXT; СМЕЩЕНИЕ УСТАНАВЛИВАЕТСЯ НА 8
БИТ 4	БИТ 3	
0	0	НЕТ ОКОНЧАНИЯ ПО СЧЕТЧИКУ БАЙТ
0	1	ОКАНЧИВАЕТСЯ, КОГДА BC=0; СМЕЩЕНИЕ УСТАНАВЛИВАЕТСЯ НА 0
1	0	ОКАНЧИВАЕТСЯ, КОГДА BC=0; СМЕЩЕНИЕ УСТАНАВЛИВАЕТСЯ НА 4
1	1	ОКАНЧИВАЕТСЯ, КОГДА BC=0; СМЕЩЕНИЕ УСТАНАВЛИВАЕТСЯ НА 8
БИТ 2	БИТ 1	БИТ 0	
0	0	0	НЕТ ОКОНЧАНИЯ МАСКИРОВАННЫМ СРАВНЕНИЕМ
0	0	1	ОКАНЧИВАЕТСЯ, КОГДА СРАВНЕНИЕ СООТВЕТСТВУЕТ; СМЕЩЕНИЕ УСТАНАВЛИВАЕТСЯ НА 0
0	1	0	ОКАНЧИВАЕТСЯ, КОГДА СРАВНЕНИЕ СООТВЕТСТВУЕТ; СМЕЩЕНИЕ УСТАНАВЛИВАЕТСЯ НА 4
0	1	1	ОКАНЧИВАЕТСЯ, КОГДА СРАВНЕНИЕ СООТВЕТСТВУЕТ; СМЕЩЕНИЕ УСТАНАВЛИВАЕТСЯ НА 8
1	0	0	НЕ ВЛИЯЕТ
1	0	1	ОКАНЧИВАЕТСЯ, КОГДА НЕТ СООТВЕТСТВИЯ; СМЕЩЕНИЕ УСТАНАВЛИВАЕТСЯ НА 0
1	1	0	ОКАНЧИВАЕТСЯ, КОГДА НЕТ СООТВЕТСТВИЯ; СМЕЩЕНИЕ УСТАНАВЛИВАЕТСЯ НА 4
1	1	1	ОКАНЧИВАЕТСЯ, КОГДА НЕТ СООТВЕТСТВИЯ; СМЕЩЕНИЕ УСТАНАВЛИВАЕТСЯ НА 8

Рис. 11.32. Окончание операции прямого доступа к памяти

При завершении операции ПДП канал выполняет команду, адрес которой равен содержимому ТР плюс смещение. Следовательно, при задании нескольких условий окончания ПДП смещение вместе с командой перехода позволяет перейти к различным процедурам завершения ПДП в зависимости от фактической причины окончания ПДП. Если одновременно возникает

несколько условий, используется наибольшее смещение, соответствующее удовлетворенному условию. Для инициирования передачи ПДП в канальной программе необходимы команды загрузки указателей источника и получателя, регистров CC, BC и при необходимости GC и MC и указания ширины шины ввода-вывода.

11.4.2. ВЗАИМОДЕЙСТВИЕ ЦЕНТРАЛЬНОГО ПРОЦЕССОРА И ПРОЦЕССОРА ВВОДА-ВЫВОДА

Межпроцессорное взаимодействие, включая инициализацию ПВВ и диспетчирование задачи, опирается на память и реализуется посредством связанного списка управляющих блоков. Первый управляющий блок в связанном списке хранится, начиная с фиксированной ячейки FFFF6, а остальные размещаются в определяемых пользователем областях, каждая из которых адресуется предыдущим управляющим блоком.

Области взаимодействия ПВВ показаны на рис. 11.33. Блок указателя системной конфигурации (SCPB) содержит три слова, начиная с ячейки FFFF6 системной памяти. Младший байт (SYSBUS) определяет ширину си-

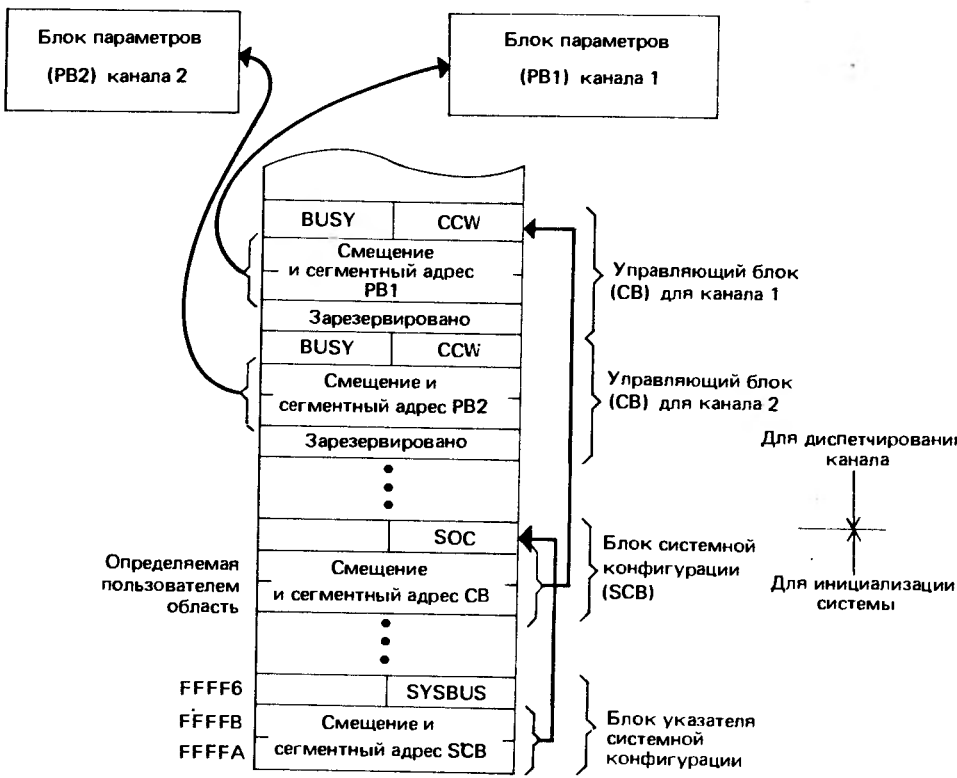


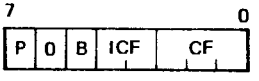
Рис. 11.33. Структура взаимодействия

стемной шины: 16 бит, если SYSBUS = 1, и 8 бит, если SYSBUS = 0. Следующие два слова хранят смещение и сегментный адрес блока системной конфигурации (SCB).

Блок SCB не требуется хранить в фиксированной области, но он должен находиться в системном пространстве. Младший байт SCB представляет собой приказ системной операции SOC. Биты 0 и 1 этого приказа определяют ширину шины ввода-вывода и режим следующим образом:

бит 0 = 0 показывает 8-битную шину ввода-вывода,
бит 0 = 1 показывает 16-битную шину ввода-вывода,
бит 1 = 0 показывает стандартный режим $\overline{RQ/GT}$,
бит 1 = 1 показывает модифицированный режим $\overline{RQ/GT}$ для использования с несколькими ПВВ.

Последние два слова в SCB содержат смещение и сегментный адрес начала двух смежных управляющих блоков канала (CB) в системном пространстве. Для каждого канала имеется свой управляющий блок и первый его байт, называемый командным словом канала CCW, показывает действие, предпринимаемое каналом. Следующий байт BUSY показывает состояние занятости канала (FF – канал занят, 00 – канал не занят), а последние два слова содержат адрес блока параметров. Блок параметров предназначен для указания начального адреса канальной программы и передачи информации в эту программу и из нее.



CF	Поле приказа
000	Модифицировать PSW
001	Запустить канальную программу, находящуюся в пространстве ввода-вывода
010	(Зарезервировано)
011	Запустить канальную программу, находящуюся в системном пространстве
100	(Зарезервировано)
101	Возобновить приостановленную операцию канала
110	Приостановить операцию канала
111	Остановить операцию канала
ICF	Поле управления прерыванием
00	Игнорируется, не воздействует на прерывания
01	Снять запрос прерывания; прерывание подтверждается
10	Разрешить прерывания
11	Запретить прерывания
B	Предел загрузки шины
0	Нет предела загрузки шины
1	Предел загрузки шины
P	Бит приоритета

Рис. 11.34. Определение управляющего слова канала

Формат CCW определен на рис. 11.34. Он содержит 3-битное поле приказа, 2-битное поле управления прерыванием, бит предела загрузки шины и бит приоритета. Поле приказа определяет один из следующих шести допустимых приказов:

Модифицировать PSW. Вызывает модификацию PSW.

Запустить канальную программу (пространство ввода-вывода). Инициирует выполнение канальной программы, находящейся в пространстве ввода-вывода.

Запустить канальную программу (системное пространство). Инициирует выполнение канальной программы, находящейся в системном пространстве.

Приостановить операцию канала. Приостанавливает текущую операцию канала до получения приказа возобновления. (Приказ обычно применяется не для вложения канальных программ, а для простой приостановки операции ввода-вывода.)

Возобновить операцию канала. Вызывает продолжение приостановленной операции с точки останова.

Остановить операцию канала. Аннулирует текущую операцию канала.

Поле управления прерыванием служит для разрешения и запрещения запросов прерываний и снятия предыдущих запросов. Когда ПВВ выдает запрос прерывания, он устанавливает бит обслуживания в PSW. Пока прерывание обслуживается, этот бит необходимо сбросить посредством выдачи в ПВВ приказа с комбинацией 01 в поле управления прерыванием; в противном случае запрос будет блокировать другие запросы.

Когда бит предела загрузки шины установлен в 1, ПВВ может выполнять только одну команду за каждые 128 тактов синхронизации. Это предотвращает монополизацию шины процессором ввода-вывода в тех ситуациях, когда он не должен быть доминирующим процессором. Бит приоритета в PSW устанавливается или сбрасывается в соответствии с битом приоритета в CCW.

Схемы, показывающие взаимодействие ЦП и ПВВ, приведены на рис. 11.35. Обязательные действия программы ЦП отражены на рис. 11.35, *а*, а действия, предпринимаемые ПВВ, — на рис. 11.35, *б*. Из рисунка видно, что фазы инициализации и задачи инициируются сигналом внимания канала CA. Как уже указывалось, ПВВ ассоциируется с двумя адресами и линия A0 подключается на вход SEL. Внимание канала проявляется в выдаче на линии адреса одного из двух адресов ПВВ; сигналы на линиях данных игнорируются. Выдачу CA можно реализовать командой OUT. За исключением фазы инициализации, когда SEL = 0 определяет ПВВ ведущим, а SEL = 1 — ведомым, вход SEL показывает, какой канал должен принять внимание (SEL = 0 выбирает канал 1, а SEL = 1 — канал 2).

После сброса ПВВ (обычно в последовательности включения) его до выполнения первой задачи необходимо инициализировать. Для инициализации ПВВ программа ЦП должна выполнить следующие действия:

сформировать SCPB и SCB,
установить байты BUSY каналов 1 и 2 на FF и 00 соответственно,
выдать CA, определяющий ПВВ ведущим или ведомым,
ожидать до тех пор, пока ПВВ не сбросит байт BUSY канала 1.

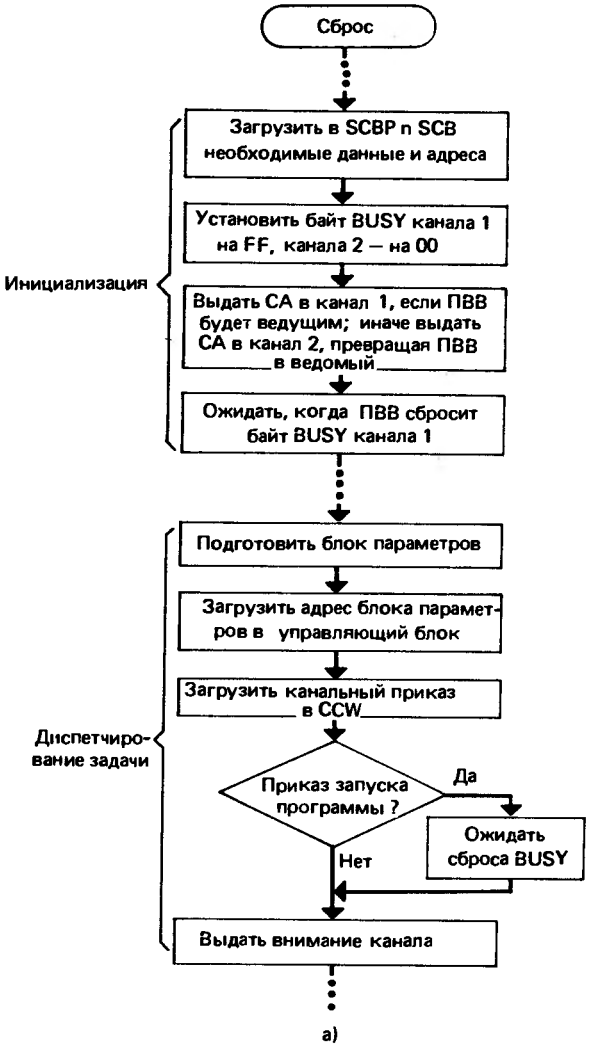
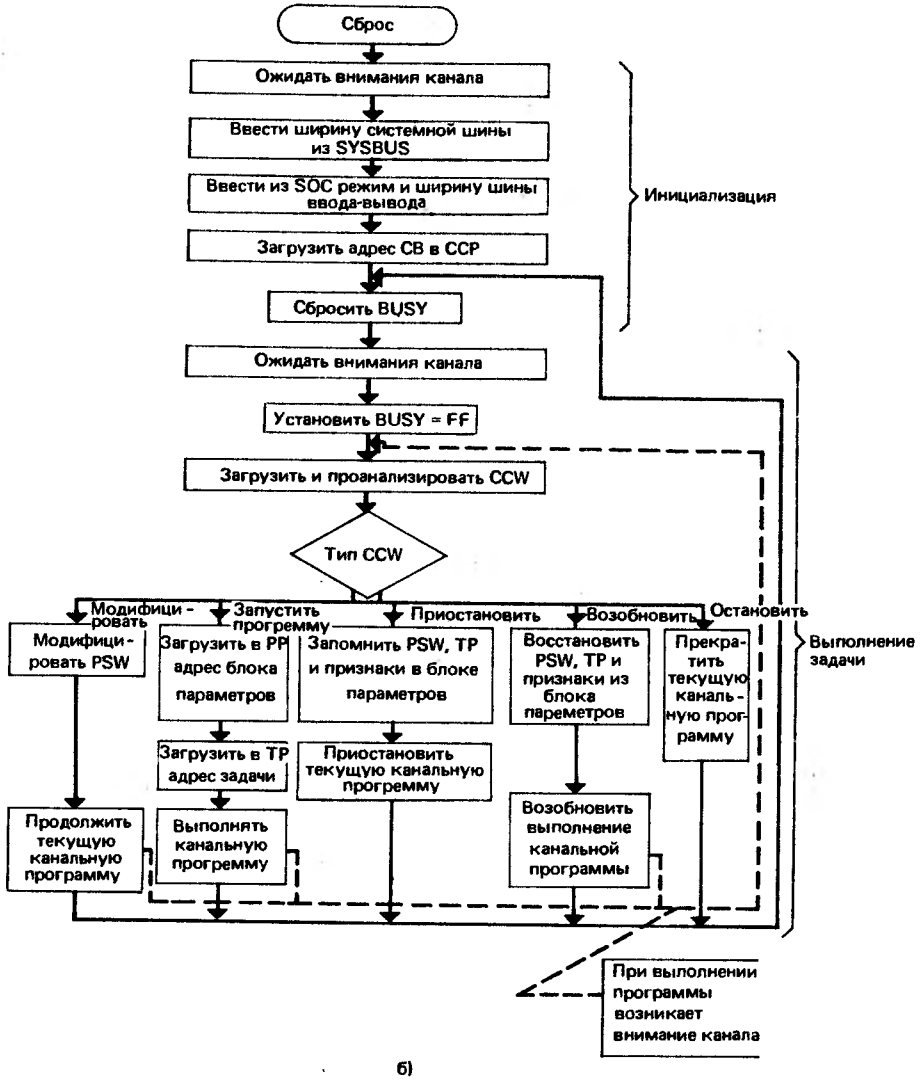


Рис. 11.35. Схемы инициализации и работы процессора ввода-вывода 8089:
***а* — действия, предпринимаемые программой ЦП;**

Ассемблерный фрагмент типичного процесса инициализации представлен на рис. 11.36. В нем предполагается, что ПВВ имеет адреса 0080 и 0081 и работает в режиме ведущего. Предполагается также, что ширина системной шины 16 бит, а ширина шины ввода-вывода 8 бит. Если бы адреса портов ПВВ находились в системном пространстве, а не в пространстве ввода-вывода, команду OUT следовало бы заменить на команду MOV.



б — действия ПВВ

На фазе инициализации СА заставляет ПВВ автоматически выполнить следующие действия:

1. Ввести SYSBUS из ячейки FFFF6, чтобы ПВВ мог настроиться на системную шину.
2. Загрузить адрес SCB из ячеек FFFF8 и FFFFA.

```
SCPBLOCK    SEGMENT   AT FFFFH
                ORG     6
    SYSBUS      DB      1               ;ШЕСТНАДЦАТИРАЗРЯДНАЯ СИСТЕМНАЯ ШИНА
                DB      ?
    SCBP        DD      SOC
SCPBLOCK    ENDS
SCBLOCK     SEGMENT
    SOC         DB      ?
                DB      ?
    CBP         DW      ?,?
SCBLOCK     ENDS
CCBIOPX     SEGMENT
    CCW1        DB      ?
    BUSY1       DB      ?
    PBP1        DW      ?,?
                DW      ?
    CCW2        DB      ?
    BUSY2       DB      ?
    PBP2        DW      ?,?
                DW      ?
CCBIOPX     ENDS
INIT_CODE   SEGMENT
  ASSUME  CS:INIT_CODE, DS:CCBIOPX, ES:SCBLOCK
                MOV     AX,SCBLOCK
                MOV     ES,AX
                MOV     ES:SOC,0        ;ШИРИНА ШИНЫ ВВОДА-ВЫВОДА
                MOV     AX,CCBIOPX      ;РАВНА 8, СТАНДАРТНЫЙ RQ/GT
                MOV     DS,AX
                MOV     ES:CBP+2,AX     ;ЗАПОМНИТЬ АДРЕС В CBP+2
                MOV     AX,OFFSET CCW1  ;СЕГМЕНТНЫЙ АДРЕС CCW1
                MOV     ES:CBP,AX       ;ЗАПОМНИТЬ В CBP СМЕЩЕНИЕ CCW1
                MOV     BUSY1,OFFH      ;УСТАНОВИТЬ BUSY1 НА FF
                MOV     BUSY2,0         ;УСТАНОВИТЬ BUSY2 НА 00
                OUT     80H,AL          ;СДЕЛАТЬ ПРОЦЕССОР ВЕДУЩИМ
    WAITING:    TEST    BUSY1,OFFH      ;ПРОВЕРЯТЬ, КОГДА НЕ БУДЕТ
                JNE     WAITING         ;ЗАНЯТ
                 .
                 .
                 .
```

Рис. 11.36. Фрагмент инициализации процессора ввода-вывода

3. Ввести SOC из SCB, чтобы ПВВ определили режим и ширину шины ввода-вывода.

4. Загрузить адрес управляющего блока из SCB + 2 и SCB + 4 в регистр CCP.

5. Установить BUSY в CB + 1 на 00.

6. Ожидать от ЦП следующего сигнала CA, инициирующего выполнение задачи.

При наличии в системе нескольких ПВВ центральный процессор инициализирует их по очереди, пользуясь той же процедурой и теми же самыми областями для SCPB и SCB. Разумеется, перед инициализацией каждого ПВВ необходимо модифицировать SCB, так как каждый ПВВ должен иметь отдельный управляющий блок. Могут изменяться также режим и ширина шины ввода-вывода.

Так как инициализированный ПВВ запомнил начальный адрес управляющего блока в своем регистре CCP и учел другую информацию о конфигурации системы, теперь в любом из каналов можно запустить канальную программу. Центральный процессор диспетчирует задачу любому каналу с помощью следующих операций:

1. Подготовить блок параметров.

2. Поместить адрес блока параметров в CB + 2 и CB + 4 для канала 1 или в CB + 10 и CB + 12 для канала 2.

3. Загрузить требуемый канальный приказ в CCW канала.

4. Выдать в канал CA.

Связи блоков сообщений показаны на рис. 11.37. Отметим, что в блоке параметров первые два слова всегда адресуют канальную программу, а остальные определяются пользователем. (Как и при вызове подпрограммы, они содержат либо сами параметры, либо адреса параметров.) Блок параметров всегда размещается в системном пространстве.

В качестве примера рассмотрим канальную программу считывания строки текста из интерфейса 8251A, подключенного к процессору ввода-вывода. Точка входа канальной программы есть INPUTPG, а цепочка текста запоми-

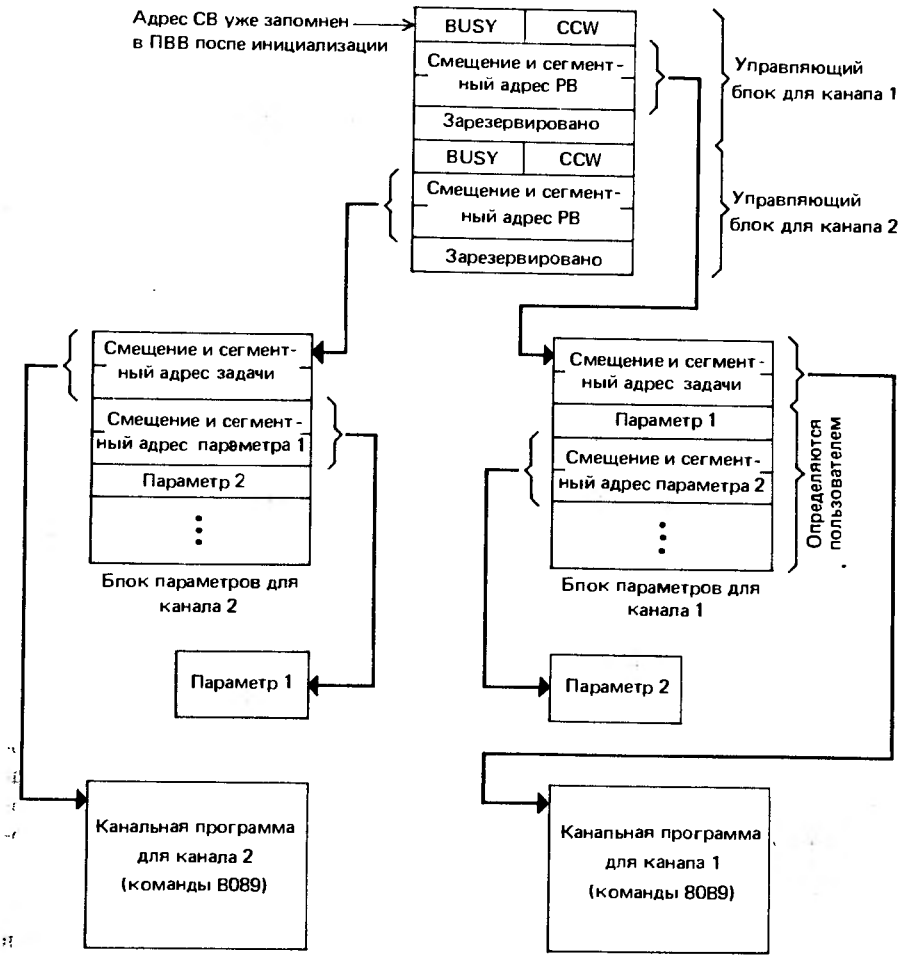


Рис. 11.37. Связи блоков сообщений

нается в MSG. Предположим, что управляющий блок определен в соответствии с рис. 11.36 и что блок параметров для канала 1 имеет вид

```
PB1     SEGMENT
 TB1     DW      ?, ? ; Указатель канальной программы
 MSGP    DW      ?, ? ; Указатель буферной области
 NUM     DW      ?    ; Число символов
PB1     ENDS
```

Следующий фрагмент инициирует канальную программу в канале 1, адрес порта ввода-вывода которого равен 0080:

```
EXTRN    INPUTPG:FAR
      .
      .
      .
ASSUME  DS:CCBIOPX,ES:PB1          ; PB1 — начало блока параметров
      MOV       AX,CCBIOPX
      MOV       DS,AX
      MOV       AX,PB1
      MOV       ES,AX
      MOV       ES:TB1, OFFSET INPUTPG
      MOV       ES:TB1+2, SEG INPUTPG
      MOV       ES:MSGP, OFFSET MSG
      MOV       ES:MSGP+2, SEG MSG
      MOV       PBP1, OFFSET TB1
      MOV       PBP1+2, AX
TY:   TEST      BUSY1,0FFH       ; Ожидать освобождения канала
      JNE       TY
      MOV       CCW1,13H         ; Запустить канальную
                                 ; программу из системного пространства
                                 ; и разрешить прерывания
      OUT       B0H,AL           ; Выдать внимание канала в канал 1
      .
      .
      .
```

После того как ПВВ инициализирован, на последующий сигнал внимания канала он загружает CCW из CB + 0 или CB + 8 в зависимости от состояния входа SEL. Если в CCW загружен приказ запуска канальной программы, адресуемый канал загружает адрес канальной программы из блока параметров в указатель задачи TP и начинает выполнение. Центральный процессор фиксирует окончание канальной программы либо через прерывание (если бит разрешения прерываний в CCW установлен), либо посредством проверки состояния BUSY.

11.4.3. СИСТЕМА КОМАНД

Процессор ввода-вывода имеет 53 команды, которые приведены на рис. 11.38. Команды классифицируются на следующие группы:

общие передачи данных;
8- или 16-битные арифметические операции;
8- или 16-битные логические операции;
команды загрузки и запоминания указателей;
условные и безусловные переходы и вызовы подпрограмм;
операции манипуляций битами и проверки;
команды управления процессором.

НАЗВАНИЕ	МНЕМОНИКА		ПРИМЕЧАНИЯ	ОПЕРАЦИЯ
ПРИБАВИТЬ СЛОВО ИЗ ПАМЯТИ	ADD	DST,SRC	(1)	(DST)← (DST)+(SRC)
ПРИБАВИТЬ БАЙТ ИЗ ПАМЯТИ	ADDB	DST,SRC	(2)	(DST)← (DST)+(SRC)
ПРИБАВИТЬ НЕПОСРЕДСТВЕННЫЙ БАЙТ	ADDBI	DST,I8	(6)	(DST)← (DST)+I8
ПРИБАВИТЬ НЕПОСРЕДСТВЕННОЕ СЛОВО	ADDI	DST,I16	(5)	(DST)← (DST)+I16
ОБ'ЕДИНИТЬ ПО "И" СО СЛОВОМ ИЗ ПАМЯТИ	AND	DST,SRC	(1)	(DST)← (DST)∧(SRC)
ОБ'ЕДИНИТЬ ПО "И" С БАЙТОМ ИЗ ПАМЯТИ	ANDB	DST,SRC	(2)	(DST)← (DST)∧(SRC)
ОБ'ЕДИНИТЬ ПО "И" С НЕПОСРЕДСТВЕННЫМ БАЙТОМ	ANDBI	DST,I8	(6)	(DST)← (DST)∧ I8
ОБ'ЕДИНИТЬ ПО "И" С НЕПОСРЕДСТВЕННЫМ СЛОВОМ	ANDI	DST,I16	(5)	(DST)← (DST)∧ I16
ВЫЗВАТЬ ПОДПРОГРАММУ	CALL	M,DST	(7)	(M)← (TP) + БИТ ПРИЗНАКА (TP)← (TP)+[DST-(TP)]
СБРОСИТЬ ВЫБРАННЫЙ БИТ В НУЛЬ	CLR	M,I3		(M)← 0 (СБРОСИТЬ БИТ I3 В M)
ДЕКРЕМЕНТ СЛОВА НА 1	DEC	DST	(5)	(DST)← (DST)-1
ДЕКРЕМЕНТ БАЙТА В ПАМЯТИ НА 1	DECB	M	(5)	(M)← (M)-1
ОСТАНОВИТЬ КАНАЛЬНУЮ ПРОГРАММУ	HLT			(BUSY В CB)← 0 И ОСТАНОВИТЬ КАНАЛЬНУЮ ПРОГРАММУ
ИНКРЕМЕНТ СЛОВА НА 1	INC	DST	(5)	(DST)← (DST)+1
ИНКРЕМЕНТ БАЙТА В ПАМЯТИ НА 1	INCB	M		(M)← (M)+1
ПЕРЕЙТИ, ЕСЛИ ВЫБРАННЫЙ БИТ ИСТИНЕН	JBT	M,I3,DST	(7)	ЕСЛИ $(M)_{I3}$ = 1, (TP)← (TP)+[DST-(TP)]
ПЕРЕЙТИ, ЕСЛИ МАСКИРОВАННОЕ СРАВНЕНИЕ РАВНО	JMCE	M,DST	(7)	ЕСЛИ ((M) ⊕ $(MC)_{7-0}$)∧ $(MC)_{15-8}$ =0, (TP)← (TP)+[DST-(TP)]
ПЕРЕЙТИ, ЕСЛИ МАСКИРОВАННОЕ СРАВНЕНИЕ НЕ РАВНО	JMCNE	M,DST	(7)	ЕСЛИ ((M) ⊕ $(MC)_{7-0}$)∧ $(MC)_{15-8}$ ≠0, (TP)← (TP)+[DST-(TP)]
ПЕРЕЙТИ БЕЗУСЛОВНО	JMP	DST	(7)	(TP)← (TP)+[DST-(TP)]
ПЕРЕЙТИ, ЕСЛИ ВЫБРАННЫЙ БИТ НЕ ИСТИНЕН	JNBT	M,I3,DST	(7)	ЕСЛИ $(M)_{I3}$ ≠ 0, (TP)← (TP)+[DST-(TP)]

Рис. 11.38. Система команд процессора ввода-вывода

ПЕРЕЙТИ, ЕСЛИ СЛОВО НЕ НУЛЬ	JNZ	SRC,DST	(7),(10)	ЕСЛИ (SRC) = 0, (TP)← (TP)+[DST-(TP)]
ПЕРЕЙТИ, ЕСЛИ БАЙТ В ПАМЯТИ НЕ НУЛЬ	JNZB	M,DST	(7)	ЕСЛИ (M) = 0, (TP)← (TP)+[DST-(TP)]
ПЕРЕЙТИ, ЕСЛИ СЛОВО РАВНО НУЛЮ	JZ	SRC,DST	(7),(10)	ЕСЛИ (SRC) ≠ 0, (TP)← (TP)+[DST-(TP)]
ПЕРЕЙТИ, ЕСЛИ БАЙТ В ПАМЯТИ РАВЕН НУЛЮ	JZB	M,DST	(7)	ЕСЛИ (M) ≠ 0, (TP)← (TP)+[DST-(TP)]
ВЫЗВАТЬ ПОДПРОГРАММУ (ДЛИННЫЙ ВЫЗОВ)	LCALL	M,DST	(8)	(M)← (TP) + БИТ ПРИЗНАКА (TP)← (TP)+[DST-(TP)]
ПЕРЕЙТИ, ЕСЛИ ВЫБРАННЫЙ БИТ ИСТИНЕН (ДЛИННЫЙ ПЕРЕХОД)	LJBT	M,I3,DST	(8)	ЕСЛИ $(M)_{I3}$ = 1, (TP)← (TP)+[DST-(TP)]
ПЕРЕЙТИ, ЕСЛИ МАСКИРОВАННОЕ СРАВНЕНИЕ РАВНО (ДЛИННЫЙ ПЕРЕХОД)	LJMCE	M,DST	(8)	ЕСЛИ $((M) \oplus (MC)_{7-0}) \wedge (MC)_{15-8}=0$, (TP)← (TP)+[DST-(TP)]
ПЕРЕЙТИ, ЕСЛИ МАСКИРОВАННОЕ СРАВНЕНИЕ НЕ РАВНО (ДЛИННЫЙ ПЕРЕХОД)	LJMCNE	M,DST	(8)	ЕСЛИ $((M) \oplus (MC)_{7-0}) \wedge (MC)_{15-8}\neq 0$, (TP)← (TP)+[DST-(TP)]
ПЕРЕЙТИ БЕЗУСЛОВНО (ДЛИННЫЙ ПЕРЕХОД)	LJMP	DST	(8)	(TP)← (TP)+[DST-(TP)]
ПЕРЕЙТИ, ЕСЛИ ВЫБРАННЫЙ БИТ НЕ ИСТИНЕН (ДЛИННЫЙ ПЕРЕХОД)	LJNBT	M,I3,DST	(8)	ЕСЛИ $(M)_{I3}$ = 0, (TP)← (TP)+[DST-(TP)]
ПЕРЕЙТИ, ЕСЛИ СЛОВО НЕ НУЛЬ (ДЛИННЫЙ ПЕРЕХОД)	LJNZ	SRC,DST	(8),(10)	ЕСЛИ (SRC) ≠ 0, (TP)← (TP)+[DST-(TP)]
ПЕРЕЙТИ, ЕСЛИ БАЙТ В ПАМЯТИ НЕ НУЛЬ (ДЛИННЫЙ ПЕРЕХОД)	LJNZB	M,DST	(8)	ЕСЛИ (M) ≠ 0, (TP)← (TP)+[DST-(TP)]
ПЕРЕЙТИ, ЕСЛИ СЛОВО РАВНО НУЛЮ (ДЛИННЫЙ ПЕРЕХОД)	LJZ	SRC,DST	(8),(10)	ЕСЛИ (SRC) = 0, (TP)← (TP)+[DST-(TP)]
ПЕРЕЙТИ, ЕСЛИ БАЙТ В ПАМЯТИ РАВЕН НУЛЮ (ДЛИННЫЙ ПЕРЕХОД)	LJZB	M,DST	(8)	ЕСЛИ (M) = 0, (TP)← (TP)+[DST-(TP)]
ЗАГРУЗИТЬ УКАЗАТЕЛЬ ИЗ ПАМЯТИ	LPD	P,M		(P)← 20-БИТНЫЙ АДРЕС, ОБРАЗОВАННЫЙ СМЕЩЕНИЕМ И СЕГМЕНТНЫМ АДРЕСОМ, КОТОРЫЕ ХРАНЯТСЯ, НАЧИНАЯ С M. БИТ ПРИЗНАКА← 0
ЗАГРУЗИТЬ УКАЗАТЕЛЬ ИЗ НЕПОСРЕДСТВЕННЫХ ДАННЫХ	LPDI	P,SRC	(11)	(P)← 20-БИТНЫЙ АДРЕС, ОБРАЗОВАННЫЙ СМЕЩЕНИЕМ SRC И СООТВЕТСТВУЮЩИМ СЕГМЕНТНЫМ АДРЕСОМ. БИТ ПРИЗНАКА← 0
ПЕРЕДАТЬ СЛОВО	MOV	DST,SRC	(3)	(DST)← (SRC)
ПЕРЕДАТЬ БАЙТ	MOVB	DST,SRC	(4)	(DST)← (SRC)
ПЕРЕДАТЬ НЕПОСРЕДСТВЕННЫЙ БАЙТ	MOVBI	DST,I8	(6)	(DST)← I8
ПЕРЕДАТЬ НЕПОСРЕДСТВЕННОЕ СЛОВО	MOVI	DST,I16	(5)	(DST)← I16
ПЕРЕДАТЬ УКАЗАТЕЛЬ	MOVP	DS ,SRC	(12)	(DST)← (SRC) (ВКЛЮЧАЯ 20-БИТНЫЙ АДРЕС И БИТ ПРИЗНАКА. ОПЕРАНД В ПАМЯТИ ИМЕЕТ ТРИ БАЙТА.)
ХОЛОСТАЯ КОМАНДА	NOP			ХОЛОСТАЯ КОМАНДА
ИНВЕРТИРОВАТЬ СЛОВО	NOT DST ИЛИ		(5)	(DST)← $\overline{(DST)}$
	NOT R,M		(5)	ИЛИ (R)← $\overline{(M)}$
ИНВЕРТИРОВАТЬ БАЙТ	NOTB M			(M)← $\overline{(M)}$
	ИЛИ NOTB R,M			ИЛИ (R)← $\overline{(M)}$ С РАСШИРЕНИЕМ ЗНАКА
ОБ'ЕДИНИТЬ ПО "ИЛИ" СЛОВО	OR	DST,SRC	(1)	(DST)← (DST)∨(SRC)

ОБ'ЕДИНИТЬ ПО "ИЛИ" БАЙТ	ORB	DST,SRC	(2)	(DST)$\leftarrow$ (DST)$\vee$(SRC)
ОБ'ЕДИНИТЬ ПО "ИЛИ" НЕПОСРЕДСТВЕННЫЙ БАЙТ	ORBI	DST,I8	(6)	(DST)$\leftarrow$ (DST)$\vee$ I8
ОБ'ЕДИНИТЬ ПО "ИЛИ" НЕПОСРЕДСТВЕННОЕ СЛОВО	ORI	DST,I16	(5)	(DST)$\leftarrow$ (DST)$\vee$ I16
УСТАНОВИТЬ ВЫБРАННЫЙ БИТ В ЕДИНИЦУ	SETB	M,I3		$(M)_{I3} \leftarrow 1$ (УСТАНОВИТЬ БИТ I3 В M)
УСТАНОВИТЬ БИТ ЗАПРОСА ПРЕРЫВАНИЯ	SINTR			СФОРМИРОВАТЬ СИГНАЛ SINTR, ЕСЛИ $(CCW)_{4,3} = 10$
ПРОВЕРИТЬ И УСТАНОВИТЬ С БЛОКИРОВКОЙ	TSL	M,I8,DST	(9)	ЕСЛИ (M)=0, (M)$\leftarrow$ I8, ИНАЧЕ (TP)$\leftarrow$ (TP)+[DST-(TP)]. ВЫДАТЬ СИГНАЛ $\overline{LOCK}$ ПРИ ВЫПОЛНЕНИИ КОМАНДЫ
УСТАНОВИТЬ ЛОГИЧЕСКУЮ ШИРИНУ ШИНЫ	WID	W1,W2		УСТАНОВИТЬ ШИРИНУ ДАННЫХ ИСТОЧНИКА ПДП НА W1 (W1: 8 ИЛИ 16). УСТАНОВИТЬ ШИРИНУ ДАННЫХ ПОЛУЧАТЕЛЯ ПДП НА W2 (W2: 8 ИЛИ 16)
ЗАПУСТИТЬ ПДП ПОСЛЕ СЛЕДУЮЩЕЙ КОМАНДЫ	XFER			ПОСЛЕ ВЫПОЛНЕНИЯ СЛЕДУЮЩЕЙ КОМАНДЫ НАЧИНАЕТСЯ ПЕРЕДАЧА ПДП

ПРИМЕЧАНИЯ ПО ФОРМАМ ОПЕРАНДОВ:

(1)	DST,SRC	R,M(СЛОВО) ИЛИ M(СЛОВО),R
(2)	DST,SRC	R,M(БАЙТ) ИЛИ M(БАЙТ),R
(3)	DST,SRC	R,M(СЛОВО) ИЛИ M(СЛОВО),R ИЛИ M(СЛОВО),M(СЛОВО)
(4)	DST,SRC	R,M(БАЙТ) ИЛИ M(БАЙТ),R ИЛИ M(БАЙТ),M(БАЙТ)
(5)	DST	R ИЛИ M(СЛОВО)
(6)	DST	R ИЛИ M(БАЙТ)
(7)	DST	МЕТКА. РАССТОЯНИЕ [DST-(TP)] МОЖЕТ ИМЕТЬ ДЛИНУ 8 ИЛИ 16 БИТ
(8)	DST	МЕТКА. РАССТОЯНИЕ [DST-(TP)] ИМЕЕТ 16 БИТ
(9)	DST	МЕТКА. РАССТОЯНИЕ [DST-(TP)] ДОЛЖНО ИМЕТЬ ДЛИНУ 8 БИТ
(10)	SRC	R ИЛИ M(СЛОВО)
(11)	SRC	16-БИТНАЯ КОНСТАНТА ИЛИ АДРЕС ПАМЯТИ
(12)	DST,SRC	P,M (3 БАЙТА) ИЛИ M,P

ОБОЗНАЧЕНИЯ:

R: GA, GB, GC, BC, IX, MC, CC, TP

P: GA, GB, GC, TP

M: ОПЕРАНД В ПАМЯТИ

In: ЦЕЛОЕ, КОТОРОЕ МОЖЕТ БЫТЬ ПРЕДСТАВЛЕНО n БИТАМИ

В арифметических и логических командах операндом может быть указатель (20 бит), регистр (16 бит), операнд в памяти (слово или байт) или константа (8 или 16 бит), причем источник и получатель не обязательно имеют одинаковую длину. В арифметической операции операнды в байт или слово расширяются со знаком до 20 бит. Если получателем определен не указатель, старшие 4 бита результата перед запоминанием отбрасываются. В логической операции операнд-байт при необходимости расширяется со знаком до 16 бит. Когда результат логической операции запоминается в указателе, старшие 4 бита не определены.

Команда передачи (MOV, MOVB, MOVI и MOVBI) также расширяет бит знака или отбрасывает старшие 4 бита операнда, чтобы соответствовать размеру получателя. Если получателем является указатель, бит признака устанавливается в состояние 1. Этим обеспечивается эффективная загрузка в указатель адреса в пространстве ввода-вывода.

Операнд в памяти должен адресоваться косвенно через указатель (GA, GB, GC или PP). Как уже показано, когда задача диспетчируется каналу, в ре-

гистр PP автоматически загружается адрес канального блóка параметров. Регистр PP в основном используется для доступа к параметрам или к их адресам. Загрузку указателей GA, GB и GC осуществляет команда MOVP.

Имеются четыре режима адресации данных в памяти:

базовой адресации. Эффективным адресом является содержимое определенного указателя;

адресации со смещением. Эффективный адрес равен сумме содержимого определенного указателя и 8-битного беззнакового смещения. Смещение допускается определять именем, представляющим собой элемент в структуре;

индексной адресации. Эффективный адрес равен сумме содержимого регистра IX (беззнаковое целое) и содержимого определенного указателя;

автоинкрементной индексной адресации. Эффективный адрес формируется так же, как в предыдущем режиме. Однако после вычисления эффективного адреса регистр IX модифицируется в соответствии с операцией: инкремент равен 1 в операции над байтами, 2 в операции над словами и 3 в команде передачи указателя.

Получающийся эффективный адрес содержит 20 бит. Если бит признака соответствующего указателя содержит 0, все 20 бит используются для обращения к операнду в системном пространстве. Если же бит признака содержит 1, для обращения к операнду в пространстве ввода-вывода используются только младшие 16 бит. Форматы режимов адресации в языке ассемблера ASM-89 и примеры кодирования показаны на рис. 11.39.

РЕЖИМ АДРЕСАЦИИ	ОБОЗНАЧЕНИЕ	ПРИМЕР
БАЗОВЫЙ	[УКАЗАТЕЛЬ]	[GA]
СО СМЕЩЕНИЕМ	[УКАЗАТЕЛЬ].СМЕЩЕНИЕ	[GA].5
ИНДЕКСНЫЙ	[УКАЗАТЕЛЬ+IX]	[PP+IX]
АВТОИНКРЕМЕНТНЫЙ ИНДЕКСНЫЙ	[УКАЗАТЕЛЬ+IX+]	[GA+IX+]

Рис. 11.39. Режимы адресации операнда в памяти

Во всех командах переходов, включая и вызовы подпрограмм, применяется режим относительной адресации. Ассемблер ASM-89 вычисляет расстояние между назначением и адресом следующей по порядку команды (т. е. адрес метки, которой передается управление, минус содержимое TP) и помещает его в команду перехода. Если переход выполняется, то смещение прибавляется к (TP).

11.4.4. ПРИМЕРЫ

Рассмотрим два примера, иллюстрирующих программирование процессора ввода-вывода. Программы носят учебный характер, чтобы избежать ненужных деталей, зависящих от конкретной системы. Программа из первого примера считывает сообщение из терминала подключенного к интерфейсу 8251A, запоминает сообщение в буферной области и возвращает число считанных символов. Если вводится символ "забоя" (удаления), предыдущий символ уничтожается. Каждый введенный символ возвращается в терминал

для эхо-индикации. Предполагается, что адрес регистра данных 8251A равен 1000, адрес регистра управления и состояния равен 1001 и что интерфейс 8251A инициализирован на требуемые формат данных, скорость передачи и режим ввода-вывода. Канальная программа, написанная на языке ассемблера ASM-89, приведена на рис. 11.40. Эту программу можно связать с программным модулем микропроцессоров 8086/8088, который диспетчирует задачу процессору ввода-вывода (см. раздел 11.4.2).

```
IOPROG1     SEGMENT
  PUBLIC  INPUTPG
    INPUTPG:    LPD     QA,[PP].4       ;ЗАГРУЗИТЬ АДРЕС БУФЕРА ИЗ ТАБЛИЦЫ ПАРАМЕТРОВ
                MOVI    QB,1000H        ;ЗАГРУЗИТЬ В QB АДРЕС РЕГИСТРА ДАННЫХ 8251A
                MOVI    QC,1001H        ;ЗАГРУЗИТЬ В QC АДРЕС РЕГИСТРА СОСТОЯНИЯ 8251A
                MOV     IX,0            ;СБРОСИТЬ ИНДЕКСНЫЙ РЕГИСТР
    INPWAIT:    JNBT    [QC],1,INPWAIT  ;ВХОДНОЙ СИМВОЛ ГОТОВ? ЕСЛИ НЕТ, ОЖИДАТЬ;
                MOVB    BC,[QB]         ;ИНАЧЕ ПЕРЕДАТЬ ЕГО В BC
                MOVI    MC,7F7FH        ;СРАВНИТЬ С СИМВОЛОМ ЗАБОЯ
                JMCNE   [QB],STORE      ;ЕСЛИ НЕ СИМВОЛ ЗАБОЯ, ПЕРЕЙТИ К STORE
                DEC     IX              ;АДРЕС ПРЕДЫДУЩЕГО СИМВОЛА В БУФЕРЕ
                MOVBI   BC,08H          ;ЗАГРУЗИТЬ В BC КОД ВОЗВРАТА НА ШАГ
    OUTWAIT:    JNBT    [QC],0,OUTWAIT  ;ЕСЛИ ВЫХОДНЫЕ ДАННЫЕ НЕ ПУСТЫЕ, ОЖИДАТЬ
                MOVB    [QB],BC         ;ИНАЧЕ ВЫВЕСТИ ВОЗВРАТ
                JMP     INPWAIT         ;ВВЕСТИ СЛЕДУЮЩИЙ СИМВОЛ
    STORE:      MOVB    [QA+IX],BC      ;ЗАПОМНИТЬ СИМВОЛ В БУФЕРНОЙ ОБЛАСТИ
    ECHO:       JNBT    [QC],0,ECHO     ;ВЫХОДНОЙ РЕГИСТР ДАННЫХ ПУСТОЙ?
                MOVB    [QB],BC         ;ЕСЛИ ДА, ПОВТОРИТЬ СИМВОЛ
                MOVI    MC,7F0DH        ;СРАВНИТЬ С ВОЗВРАТОМ КАРЕТКИ
                JMCNE   [QA+IX+],INPWAIT;ВОЗВРАТ КАРЕТКИ ? ЕСЛИ НЕТ, ВВЕСТИ
                                        ;СЛЕДУЮЩИЙ СИМВОЛ И ИНКРЕМЕНТ IX
                MOV     [PP].8,IX       ;ДЛИНА СИМВОЛЬНОЙ ЦЕПОЧКИ
                SINTR                   ;ПРЕРВАТЬ 8086/8088
                HLT                     ;ОЖИДАТЬ СЛЕДУЮЩЕГО CA ДЛЯ ЗАПУСКА
                                        ;ДРУГОЙ КАНАЛЬНОЙ ПРОГРАММЫ
IOPROG1     ENDS
```

Рис. 11.40. Канальная программа ввода строки символов

Данный пример показывает реализацию программного ввода-вывода и сканирование символьной цепочки операцией маскированного сравнения. После обнаружения возврата каретки канальная программа возвращает в блок параметров число принятых символов, извещает ЦП запросом прерывания, сбрасывает состояние занятости и ожидает следующего сигнала внимания канала.

Во втором примере заданное число байт записывается в накопитель на гибком диске, управляемый контроллером 8272. Предполагается, что блок параметров инициализирован следующим образом:

указатель канальной программы (4 байта),
указатель буфера (4 байта),
число записываемых байт (2 байта),
номер накопителя (1 байт),
номер цилиндра (1 байт),
адрес дорожки (1 байт),
запись (1 байт),
число байт в секторе (1 байт),
конец дорожки (1 байт),
длина промежутка (1 байт),
длина данных (1 байт).

Следующие 3 байта предназначены для запоминания байт состояния ST0, ST1 и ST2 после окончания записи. Предполагается, что основной регистр состояния, регистр данных и счетчик байт в контроллере 8272 имеют адреса 8000, 8001 и 8002 соответственно. Как показано на рис. 11.41, до выполнения команды XFER программа инициализирует указатели источника и получателя, определяет содержимое регистра управления канала и ширину шин, а также загружает в контроллер 8272 все параметры, кроме последнего. Команда XFER заставляет канал запустить операцию ПДП после выполнения следующей за ней команды. Этот прием оставляет ПВВ достаточно времени для перехода в режим ПДП, даже если устройство начинает передачу данных сразу же при получении последнего параметра.

```
DISKWT      SEGMENT
  PUBLIC  WRITE
    WRITE:      MOVI    QC,8000H        ;QC АДРЕСУЕТ РЕГИСТР СОСТОЯНИЯ
                MOVI    MC,0C0H         ;УСТАНОВИТЬ БАЙТ МАСКИРОВАННОГО СРАВНЕНИЯ
                LPD     QA,[PP].4       ;QA АДРЕСУЕТ ИСТОЧНИК
                MOVI    QB,8001H        ;QB АДРЕСУЕТ ПОЛУЧАТЕЛЬ
                MOVI    CC,5008H        ;ЗАДАТЬ ПРАВИЛЬНУЮ ОПЕРАЦИЮ ПДП
                WID     16,8            ;ШИНА ИСТОЧНИКА=16, ШИНА ПОЛУЧАТЕЛЯ=8
    TRY1:       JMCNE   [QC],TRY1       ;8272 ГОТОВ ВОСПРИНИМАТЬ СЛЕДУЮЩИЙ ПРИКАЗ ?
                MOVBI   [QC].1,3        ;ЗАГРУЗИТЬ ПРИКАЗ ЗАПИСИ
                MOVI    IX,10           ;УСТАНОВИТЬ ИНДЕКСНЫЙ РЕГИСТР НА ЗАГРУЗКУ
                MOVI    BC,7            ;В 8272 СЛЕДУЮЩИХ 7 ПАРАМЕТРОВ
    TRY2:       JMCNE   [QC],TRY2       ;8272 ГОТОВ К СЛЕДУЮЩЕМУ ПАРАМЕТРУ ?
                MOVB    [QC].1,[PP+IX+] ;ЗАГРУЗИТЬ СЛЕДУЮЩИЙ ПАРАМЕТР
                DEC     BC
                JNZ     BC,TRY2         ;ПОВТОРИТЬ 7 РАЗ
                MOV     BC,[PP].8       ;ЗАГРУЗИТЬ ЧИСЛО БАЙТ В СЧЕТЧИК
    TRY3:       JMCNE   [QC],TRY3
                XFER                    ;ЗАПУСТИТЬ ПДП ПОСЛЕ СЛЕДУЮЩЕЙ КОМАНДЫ
                MOV     [QC].1,[PP].17  ;ЗАГРУЗИТЬ В 8272 ПОСЛЕДНИЙ ПАРАМЕТР
                                        ;И ЗАПУСТИТЬ ПДП
                MOVBI   [QC].2,0        ;ВЫДАТЬ ОКОНЧАТЕЛЬНЫЙ ОТСЧЕТ
                MOVI    BC,3
                MOVI    IX,18
                MOVI    MC,0C0C0H       ;УСТАНОВИТЬ БАЙТ МАСКИРОВАННОГО СРАВНЕНИЯ
    TRY4:       JMCNE   [QC],TRY4
                MOVB    [PP+IX+],[QC].1 ;ЗАПОМНИТЬ ST0, ST1 И ST2
                DEC     BC
                JNZ     BC,TRY4
                MOVI    BC,4
                MOVI    IX,11
    TRY4:       JMCNE   [QC],TRY5
                MOVB    [PP+IX+],[QC].1 ;МОДИФИЦИРОВАТЬ ПАРАМЕТРЫ C, H, R И N
                DEC     BC
                JNZ     BC,TRY5
                SINTR                   ;ПРЕРВАТЬ ЦЕНТРАЛЬНЫЙ ПРОЦЕССОР
                HLT                     ;ОЖИДАТЬ СЛЕДУЮЩИЙ ЗАПРОС КАНАЛА
DISKWT      SEGMENT
```

Рис. 11.41. Канальная программа записи на диск блока байт

В регистр управления канала загружается код 5008H, показывающий, что операция ПДП выполняется из памяти во ввод-вывод без преобразования, синхронизируется получателем и использует регистр GA в качестве указателя источника. После загрузки последнего параметра начинается передача ПДП. Когда ПДП останавливается при достижении счетчиком нуля, выдается сигнал последнего отсчета, байты состояния ST0, ST1 и ST2 передаются в блок параметров и модифицируются параметры C, H, R и N контроллера 8272.

Упражнения

1. Постройте таблицу архитектурных средств микропроцессоров 8086/8088, предназначенных для мультипроцессорных систем. В первом столбце необходимо перечислить

все специальные линии и команды, а во втором – указать их основное назначение в мультипроцессорной системе.

2. Пусть слабо связанная мультипроцессорная система состоит из следующих трех модулей:

A. Процессор 8086 с локальной памятью;

B. Процессор 8086 и 8087;

C. Два процессора ввода-вывода 8089 с локальной шиной ввода-вывода.

Определите необходимые для каждого модуля интерфейсные микросхемы.

3. Модифицируйте рис. 11.14 так, чтобы микропроцессор 8086 мог обращаться к двум системным шинам.

4. Поясните, почему очередь команд улучшает коэффициент использования процессора в мультипроцессорной системе.

5. Предположим, что выполняются следующие однооперандные команды:

а) 500 команд общего назначения; каждая команда требует два цикла на выполнение и по два цикла на выборку каждой команды и обращение к операнду;

б) 300 команд с плавающей точкой; каждая команда требует 15 циклов на выполнение и по 4 цикла на выборку каждой команды и обращение к операнду;

в) 200 команд, относящихся к вводу-выводу; каждая команда требует один цикл на выполнение и по два цикла на выборку команды и обращение к операнду.

Предполагая, что полоса пропускания шины составляет 10^6 циклов в секунду, для каждой из следующих конфигураций определите время выполнения всех команд и коэффициент использования шины:

а) система, реализованная на одном процессоре;

б) система состоит из двух процессоров – один предназначен для выполнения команд общего назначения и ввода-вывода, а второй ориентирован на команды с плавающей точкой. Считайте, что оба процессора работают идеально параллельно;

в) система состоит из трех процессоров, ориентированных на команды каждого класса. Считайте, что все процессоры работают идеально параллельно.

6. Пусть следующие 16-ричные числа представляют короткие вещественные числа:

CA5B2000 3C630000 4C341200

Определите десятичные эквиваленты этих чисел.

7. Преобразуйте следующие десятичные числа в короткие вещественные числа (ответы запишите в 16-ричной системе):

$$-123450000 \quad -148.4375 \times 10^{-3} \quad 1.2345625 \times 10^{4}$$

8. Напишите фрагмент ассемблерной программы на языке 8086/8087, который преобразует беззнаковое десятичное число в эквивалентное короткое вещественное число. Считайте, что целая часть, содержащая до 10 десятичных цифр, находится в массиве INTARY, дробная часть, которая тоже содержит до 10 десятичных цифр, хранится в массиве FRARY, а результат необходимо запомнить в REALVAR. Предположите, что 10-байтный массив WORK зарезервирован в качестве рабочей области.

9. Пусть N вещественных чисел хранятся в коротком вещественном формате, начиная с ячейки REALARY. Напишите фрагмеит ассемблерной программы на языке 8086/8087, который умножает сумму положительных чисел иа сумму отрицательных чисел и запоминает результат в PRODUCT.

10. Напишите фрагмент ассемблерной программы на языке 8086/8087, который вычисляет $\sin\theta$ и $\operatorname{tg}\theta$, где $0 < \theta < \pi/2$. Считайте, что THETA содержит угол в коротком вещественном формате, а результаты запоминаются в длинном вещественном формате.

11. Рассмотрите решение проблемы критической секции при использовании в мультипроцессорной системе процессора 8087.

12. Для примеров инициализации ПВВ и диспетчирования задачи из раздела 11.4.2 покажите, как входы SEL и CA подключаются к системной шине.

13. Напишите канальную программу реализации приоритетного опроса, приведенного на рис. 6.7.

14. Предположим, что ПВВ управляет клавиатурой и индикатором, показанными на рис. 9.35. Напишите две канальные программы для замены процедур ввода с клавиатуры и вывода на индикатор, рассмотренных в разделе 9.4.3. Адреса KEYS и DIGITS необходимо передавать канальным программам как параметры.

15. Для примера программы на рис. 11.41 покажите, какую схему необходимо подключить к контроллеру 8272, чтобы контроллер диска воспринимал сигнал окончания при завершении операции ПДП.

16. Рассмотрите решение проблемы критической секции при использовании в мультипроцессориой системе процессора ввода-вывода 8089.

12. НОВЫЕ СВЕРХБОЛЬШИЕ ИНТЕГРАЛЬНЫЕ СХЕМЫ

При выпуске новых сверхбольших интегральных схем или при появлении возможности разместить больше схем на одном кристалле приходится решать вопрос о наиболее эффективном использовании дополнительной логики. Здесь необходимо проанализировать следующие направления:

1. Увеличить число рабочих регистров и (или) их разрядность. При увеличении разрядности регистров целесообразно расширить число контактов данных и (или) адреса с тем, чтобы повысить скорость передачи данных и емкость памяти. Изменения такого рода обычно требуют значительно больше логики, но именно они оказываются самыми серьезными и обычно приводят к появлению нового семейства процессоров и вспомогательных микросхем.

2. Улучшить вычислительные возможности процессора. Для этого обычно разрабатывается отдельный быстродействующий числовой процессор, совместимый с базовым ЦП. Такой процессор, выполняющий операции с плавающей точкой и многократной точности, значительно повышает производительность системы при вычислениях.

3. Усовершенствовать средства доступа к быстродействующей внешней памяти. Для этого можно разместить на кристалле ЦП контроллер ПДП, но при интенсивном вводе-выводе требуется отдельное устройство, способное выполнять операции ввода-вывода и передачи ПДП.

4. Разместить специализированные или системные программные средства в ПЗУ, находящемся на кристалле ЦП или в отдельной микросхеме с быстрым доступом, которая может также содержать и специальные системные аппаратные компоненты.

5. Разместить на кристалле ЦП такие компоненты, как интерфейсы ввода-вывода, таймеры, контроллеры ПДП, блоки управления прерываниями и секции памяти, что сокращает число корпусов и контактов в системе. (Однако число контактов на корпусе ЦП увеличивается из-за дополнительных линий, которые подключаются непосредственно к различным интерфейсам и

устройствам.) При этом большинство малых систем (за исключением устройств ввода-вывода) можно сократить до одной микросхемы, что ведет к удешевлению системы и повышению надежности. Такой подход характерен для контроллерных применений с минимальным числом устройств ввода-вывода и внешней памяти.

6. Разместить на кристалле ЦП логику (устройство) управления памятью или разработать отдельную микросхему, обладающую широкими воэможностями управления памятью. В первом случае устраняются задержки, вызванные прохождением сигналов через отдельную микросхему управления памятью, и сокращается общее число контактов (хотя число контактов на корпусе ЦП приходится увеличивать), но обычно этот подход оказывается более ограниченным. Такого рода усовершенствование предназначено для мультипрограммных систем.

В соответствии с направлением 1 несколько фирм разработали однокристальные 32-битные процессоры, примером которых служит процессор 80386 фирмы Intel. Направлениям 2 и 3 удовлетворяют процессоры 8087 и 8089, рассмотренные в гл. 11. Данная глава посвящена микросхемам фирмы Intel, соответствующим направлениям 4 – 6. Следует указать, что значительные усилия в этих областях предпринимают и другие фирмы, но ограниченный объем книги не позволяет рассмотреть их изделия. Приборы фирмы Intel представлены здесь как типичные представители.

12.1. МИКРОСХЕМА 80130

Микросхема 80130 специально разработана для поддержки операционной системы iRMX 86, рассмотренной в § 7.1. Она применяется вместе с микропроцессорами 8086/8088, работающими в максимальном режиме, и вместе с ЦП образует то, что фирма Intel называет процессором операционной системы (OSP). Внутренняя архитектура микросхемы 80130 представлена на рис. 12.1. Главной компонентой является ПЗУ емкостью 16К байт, в котором находится код, необходимый для 35 системных вызовов iRMX 86. Системные вызовы реализуются программными прерываниями, поэтому указатели прерываний в младших адресах памяти необходимо инициализировать для адресации соответствующих областей в ПЗУ. Кроме ПЗУ, в микросхеме 80130 имеется логика управления прерываниями, которая очень похожа на контроллер 8259А и может обрабатывать до семи внешних прерываний. Предусмотрены схемы синхронизации, формирующие временные интервалы, определяемые в приказе SLEEP и других приказах, а также схемы, программирующие скорость передачи. Наконец, на кристалле размещена вся необходимая логика управления и буферирования шины. Логику прерывания можно каскадировать с ведомыми контроллерами 8259А, образуя систему, обеспечивающую обработку до 56 прерываний.

Рис. 12.2 показывает разводку контактов микросхемы 80130, а рис. 12.3 – ее подключение в типичной системе. Отметим, что микросхема 80130 нахо-

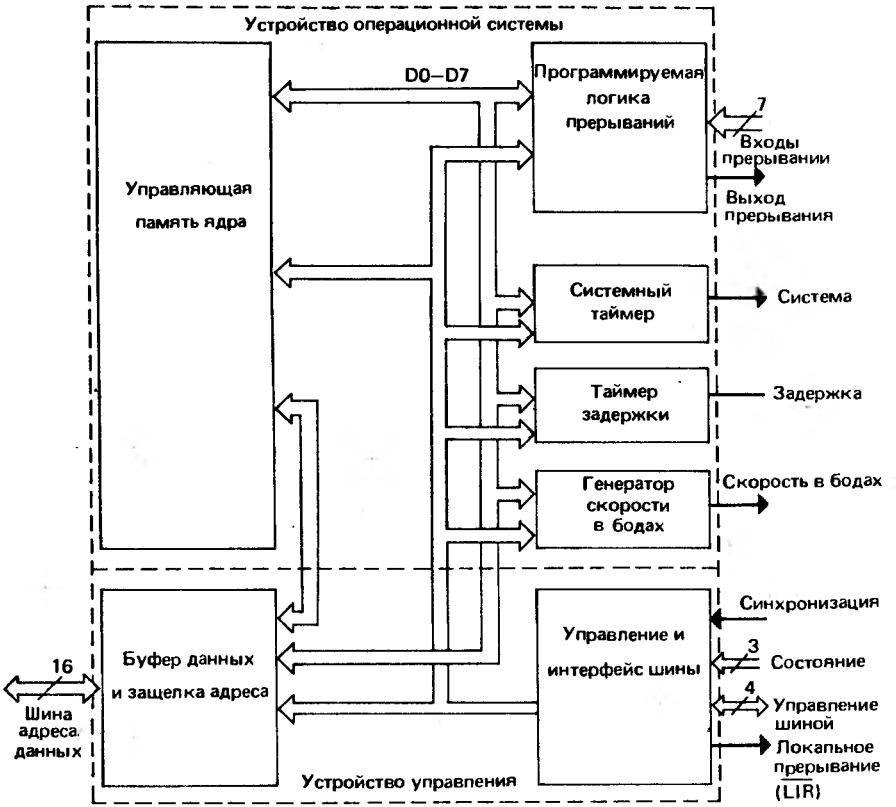


Рис. 12.1. Внутренняя архитектура 80130

дится со стороны ЦП относительно логики управления шиной и поэтому должна дешифрировать сигналы состояния $\overline{S2}$-$\overline{S0}$. Она выдает три сигнала синхронизации: сигнал системной синхронизации SYSTICK, задержанный сигнал синхронизации (который пока не используется) и выход скорости передачи в бодах BAUD. Один из имеющихся восьми входов запросов прерываний применяется для системной синхронизации и подключается к SYSTICK; для восприятия запросов прерываний остается семь входов. Выход $\overline{LIR}$ предназначен для программного управления локальной ведомой системой прерываний. О наличии запроса прерывания микросхема 80130 сигнализирует 8086/8088 по выходу INT и фиксирует подтверждение, когда на всех трех линиях состояния $\overline{S2}$-$\overline{S0}$ имеются низкие уровни.

Для доступа к памяти и регистрам ввода-вывода микросхемы 80130 в логику дешифрирования адреса подается шина адреса (включая сигнал $\overline{BHE}$). Так как регистры, относящиеся к логике управления прерываниями и син-

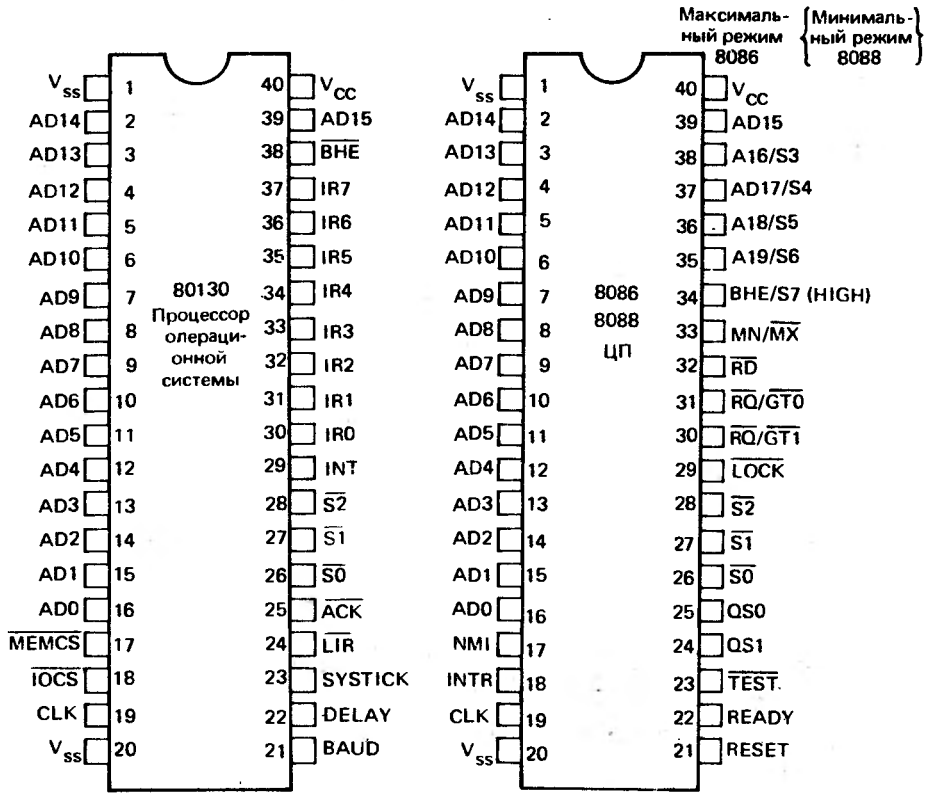


Рис. 12.2. Разводка контактов корпусов процессоров 80130 и 8086/8088

хронизации, находятся в адресном пространстве ввода-вывода, а ПЗУ – в адресном пространстве памяти, логика дешифрирования формирует два сигнала выбора кристалла, которые подаются на входы $\overline{IOCS}$ и $\overline{MEMCS}$. Сигнал $\overline{ACK} = 0$, когда осуществляется доступ к ресурсу микросхемы 80130, и объединяется по И с сигналом DEN для управления приемопередатчиками данных 8286. Чтобы передать данные в или из регистров ввода-вывода микросхемы 80130 или в ее ПЗУ, линии AD15-AD0 от процессора 8086 (или линии AD7-AD0 от процессора 8088) подключены прямо на контакты AD15-AD0 (AD7-AD0) микросхемы 80130.

Хотя на рис. 12.3 не показан процессор 8087, его легко ввести в систему, пользуясь схемами из гл. 11. По существу, таким образом можно улучшить имеющуюся систему на базе микропроцессоров 8086/8088 в максимальном режиме. Кроме того, схемы с процессором ввода-вывода 8089 из гл. 11 возможно трансформировать для включения микросхемы 80130.

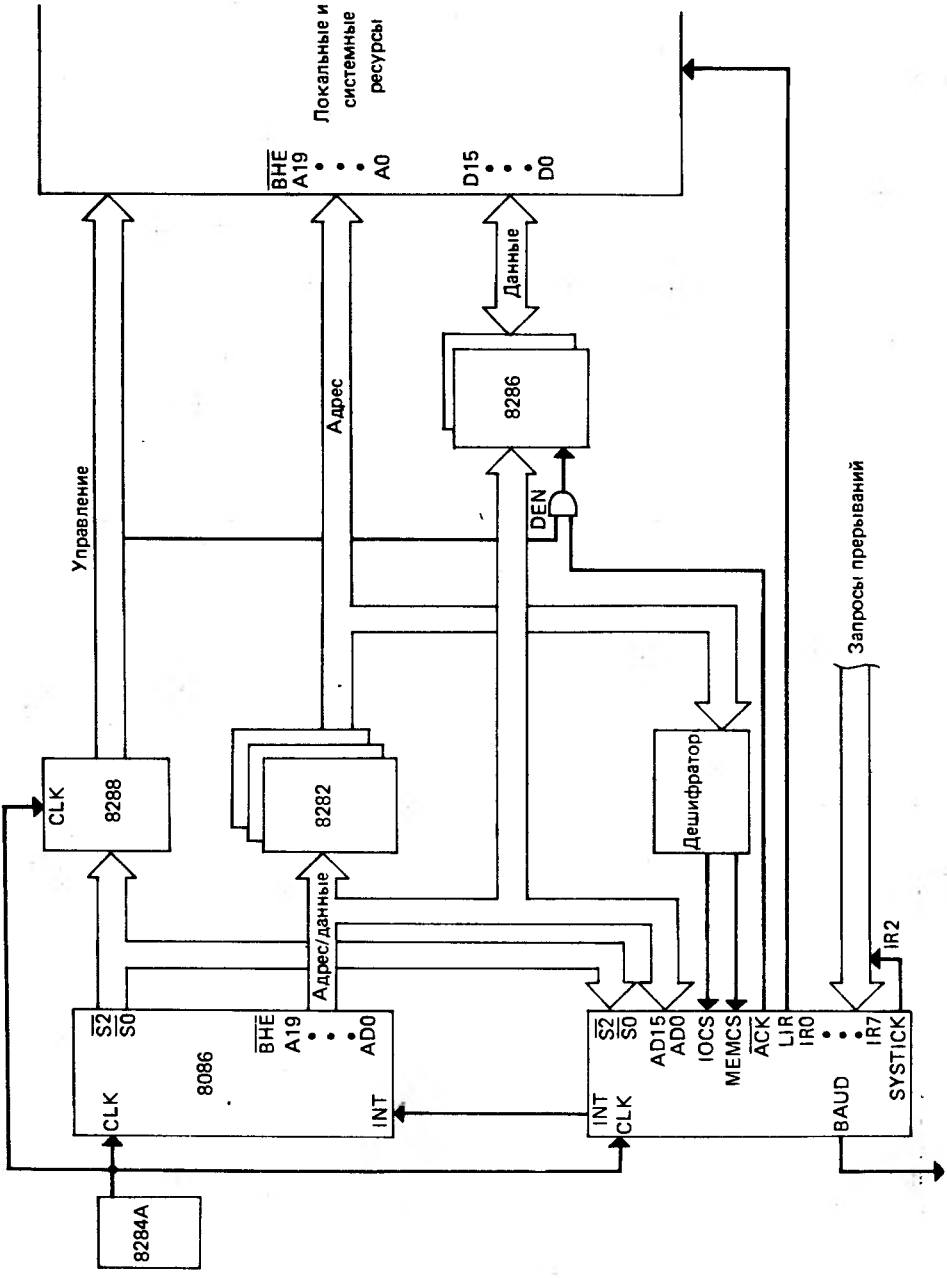


Рис. 12.3. Типичная конфигурация микросхемы 80130 с процессором операционной системы

12.2. МИКРОСХЕМА 80186

Микросхема 80186 представляет собой усовершенствованный вариант процессора 8086 с внутренними генератором синхронизации, логикой управления прерываниями, схемой таймеров, контроллерами ПДП и программируемыми схемами выбора кристалла. Разводка контактов и схема процессора 80186 приведены на рис. 12.4, откуда видно, что он имеет 68 контактов и размещен в корпусе с 4-сторонним расположением контактов (выводов).

Сигналы на линиях $\overline{\text{TEST}}$, NMI, A19/S6-A16/S3, AD15-AD0, $\overline{\text{BHE}}$/S7, $\overline{\text{LOCK}}$ и $\overline{\text{S2}}$-$\overline{\text{S0}}$ имеют такие же смысл и временные диаграммы, что и у микропроцессора 8086 в максимальном режиме. Благодаря такой совместимости в системах с процессором 80186 можно применять микросхемы управления шиной 8288 и 8289, а также внешние процессоры 8087 и 8089, которые разработаны для семейства 8086/8088. Сигналы DT/$\overline{\text{R}}$, $\overline{\text{DEN}}$, HOLD и HLDA выполняют такие же функции, что и у микропроцессора 8086 в минимальном режиме. Назначения сигналов ALE/QS0, $\overline{\text{WR}}$/QS1 и $\overline{\text{RD}}$/$\overline{\text{QSMD}}$ определяются в процессе сброса по результату опроса сигнала $\overline{\text{RD}}$/$\overline{\text{QSMD}}$ Если он имеет низкий уровень (т. е. подключен на землю), процессор 80186 переводится в режим состояния очереди и сигналы ALE/QS0 и $\overline{\text{WR}}$/QS1 отражают состояние очереди (как в максимальном режиме микропроцессора 8086), а сигнал $\overline{\text{RD}}$/$\overline{\text{QSMD}}$ при обычной работе не играет роли; в противном случае (RD/QSMD = 1) сигналы ALE, $\overline{\text{WR}}$ и $\overline{\text{RD}}$ действуют как в минимальном режиме микропроцессора 8086. Вход синхронной готовности SRDY аналогичен входу READY микропроцессора 8086 и должен синхронизироваться извне. Процессор 80186 имеет одно напряжение питания +5 В, которое подается на вход V_{CC}. Контакты V_{SS} предназначены для системной земли. Все остальные сигналы процессора 80186 в микропроцессоре 8086 отсутствуют.

Линии ARDY, $\overline{\text{RES}}$, RESET, CLKOUT, X1 и X2 подключены к внутреннему генератору синхронизации, похожему на генератор 8284A. Сигнал на входе асинхронной готовности ARDY аналогичен входу RDY генератора 8284A и синхронизируется внутренней схемой генератора синхронизации. Вход $\overline{\text{RES}}$ предназначен для подачи сигнала сброса, а сигнал RESET представляет собой синхронизированный выход сигнала $\overline{\text{RES}}$. Выходной сигнал синхронизации CLKOUT требуется в логике управления шиной и интерфейсах. Осциллятор (кварц), определяющий частоту синхронизации (она установлена равной 8 МГц), подключается на входы X1 и X2.

Линии INT0, INT1, INT2/$\overline{\text{INTA0}}$ и INT3/$\overline{\text{INTA1}}$ подключаются к логике управления прерываниями. Использование их зависит от режима, который определяется битами в регистрах управления, находящихся в контроллере прерываний. В режиме iRMX 86 контроллер 8259A должен применяться как основное устройство управления прерываниями и необходима специальная программа инициализации. Режим iRMX 86 делает контроллер совместимым с операционной системой iRMX 86, а три других режима не относятся к iRMX 86: Это режимы:

Вложенный. Все четыре линии применяются как входы запросов прерываний, а внутренняя работа контроллера примерно напоминает работу микро-

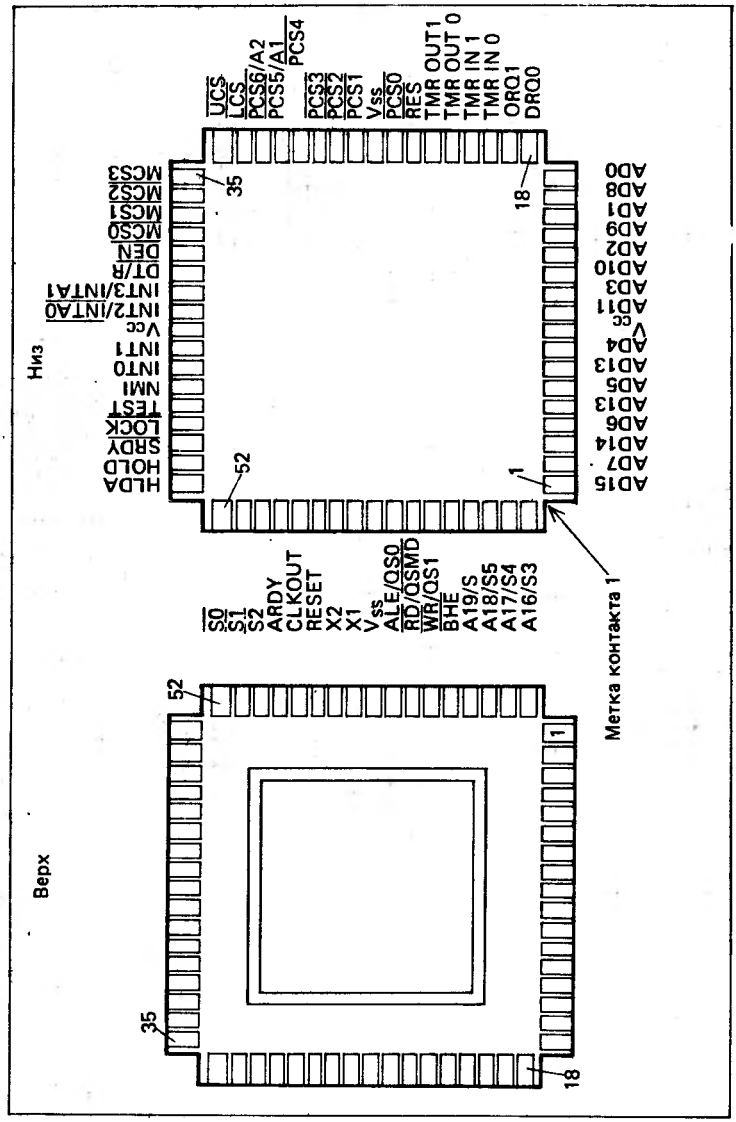


Рис. 12.4. Разводка контактов корпуса и схема процессора 80186

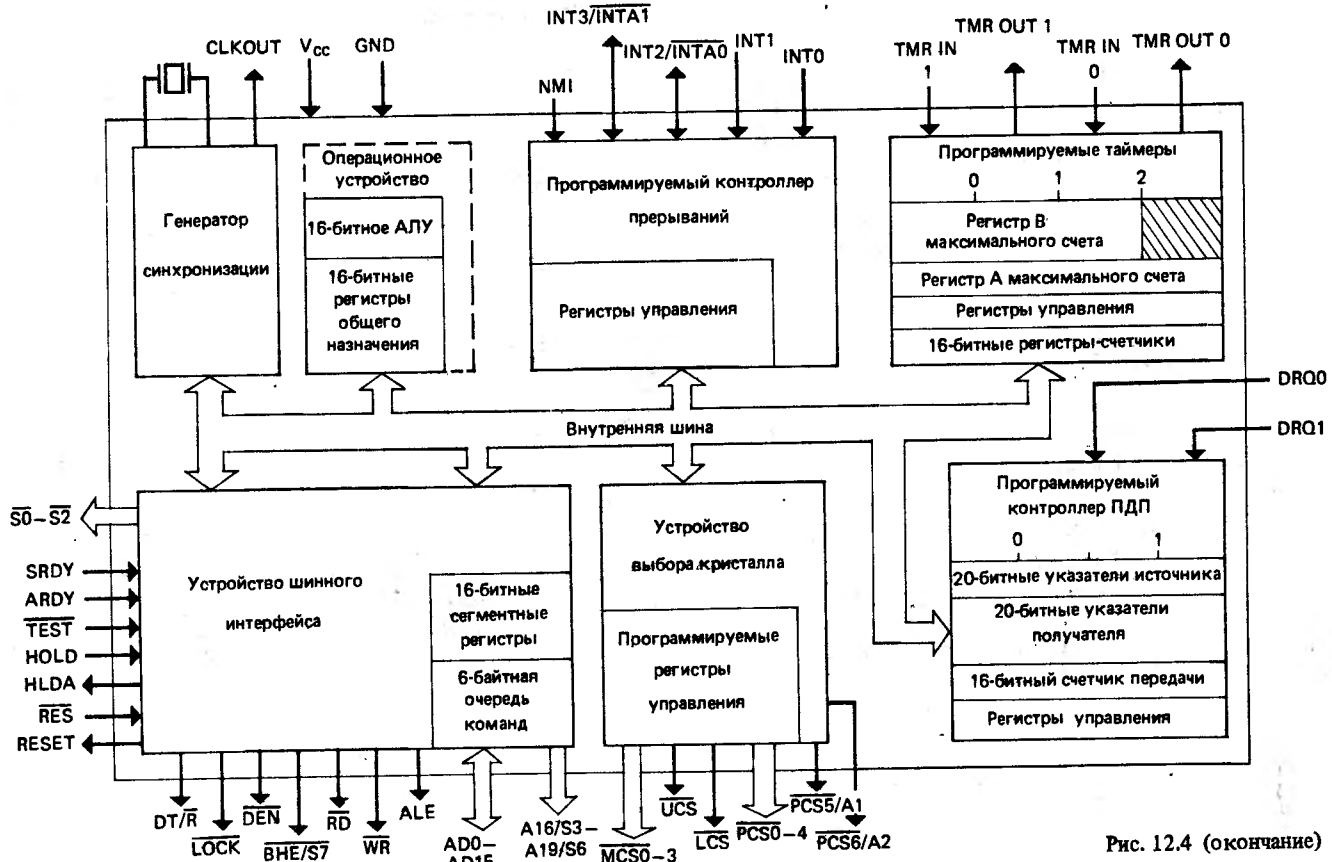


Рис. 12.4 (окончание)

схемы 8259A, но имеются четыре входа запросов прерываний вместо восьми и не разрешается подключать ведомые контроллеры 8259A.

Каскадный. Пары INT0-INT2/$\overline{\text{INTA0}}$ и INT1-INT3/$\overline{\text{INTA1}}$ действуют как пары запрос/подтверждение. Каждую из них можно подключить к отдельному устройству, приоритетной цепочке, контроллеру 8259A или каскадно включенным 8259A. В последнем варианте каждая пара может обслуживать до 64 линий прерываний.

Специальный вложенный. Аналогичен предыдущему режиму, но каскадированные ведущие контроллеры 8259A работают в специальном вложенном режиме (см. раздел 8.3.2).

Процессор 80186 имеет три таймера, один из которых подключен к паре TMR IN 0 и TMR OUT 0, другой – к паре TMR IN 1 и TMR OUT 1, а третий внешне недоступен, но может быть делителем первых двух таймеров или источником запросов в одном из внутренних контроллеров ПДП. Выполняемая таймером функция зависит от режима работы. Внешне доступные таймеры можно перевести в один из нескольких режимов, в которых входы TMR IN используются для подачи сигналов управления или синхронизации, а выходы TMR OUT формируют одиночные импульсы или непрерывные сигналы.

Линии $\overline{\text{UCS}}$, $\overline{\text{LCS}}$, $\overline{\text{MCS3}}$-$\overline{\text{MCS0}}$, $\overline{\text{PCS4}}$-$\overline{\text{PCS0}}$, $\overline{\text{PCS5}}$/A1 и $\overline{\text{PCS6}}$/A2 подключены к логике выбора кристалла и позволяют интерфейсам получать сигналы выбора кристалла непосредственно от процессора 80186. При этом логика дешифрирования адреса в этих интерфейсах не требуется. Первые шесть из указанных линий предназначены для выбора модулей, адреса которых находятся в пространстве памяти, а последние семь формируют сигналы выбора кристаллов для интерфейсов с адресами в пространстве ввода-вывода.

Для сигналов выбора памяти начальные адреса и размеры модулей, воспринимающих сигналы, программируются с помощью регистров управления в секции выбора кристалла. Однако модуль, подключаемый к $\overline{\text{UCS}}$ должен находиться в самой верхней части памяти (которая кончается адресом FFFFF), а модуль, подключаемый к $\overline{\text{LCS}}$, должен начинаться с адреса 00000. Линия $\overline{\text{UCS}}$ обычно применяется для выбора ПЗУ, содержащего код инициализации, выполняемый после сброса. С сигналом $\overline{\text{LCS}}$ ассоциируется память, которая содержит указатели прерываний и, возможно, секцию системного кода. Модуль, подключаемый к любой из этих линий, может иметь размер 2^n К байт, $n = 0 \ldots 8$. Линии $\overline{\text{MCS3}}$-$\overline{\text{MCS0}}$ можно использовать для выбора модулей с размерами 2^nК байт, $n = 1 \ldots 7$, но размеры всех модулей должны быть одинаковыми. Модули должны быть смежными в адресном пространстве с модулем, соответствующим $\overline{\text{MCS0}}$ и имеющим начальный адрес, который кратен 4×2^nК, где 2^nК – длина модулей.

Каждая линия периферийного выбора кристалла ассоциируется со 128-байтным блоком адресного пространства и все блоки должны быть смежными. Соответствующий $\overline{\text{PCS0}}$ начальный адрес можно запрограммировать на любой адрес, кратный 128, но после его задания начальный адрес, соответствующий $\overline{\text{PCS1}}$, равен сумме этого адреса и 128 и т. д. Разумеется, при задании начальных адресов линий выбора периферийных устройств и памяти необходимо избегать конфликтных назначений. Двухфункциональные линии

$\overline{PCS5}$/A1 и $\overline{PCS6}$/A2 можно использовать для периферийных выборов кристалла либо как защелки линий адреса A1 и A2. Во второй ситуации они обычно подключаются на входы A0 и A1 8-битных интерфейсных микросхем.

Число состояний ожидания (0, 1, 2 или 3) допускается программировать для каждой группы линий выбора. Для этого предназначены 3 младших бита соответствующего регистра управления.

Процессор 80186 имеет два независимых контроллера ПДП, причем запросы этих контроллеров осуществляются по входам DRQ0 и DRQ1. Линии явных сигналов подтверждения ПДП отсутствуют. Подтверждение можно реализовать с помощью сигналов считывания или записи, а также одной или двух линий выбора кристалла. Каждый контроллер может пересылать блоки размером до 64К байт или слов. Передача байт или слов определяется битом в регистре управления контроллера. Еще один бит в этом регистре определяет, будет прерывание или нет при достижении счетчиком нуля. Внешний сигнал об этом условии отсутствует.

Все регистры управления находятся в 256-байтном блоке адресного пространства (рис. 12.5). Базовый адрес этого блока определяется регистром управления, у которого смещение в блоке равно FE, а обращения ко всем регистрам управления осуществляются в соответствии с содержимым данного регистра. При сбросе содержимое этого регистра устанавливается таким, что базовый адрес блока равен FF00 в пространстве ввода-вывода. Это обыч-

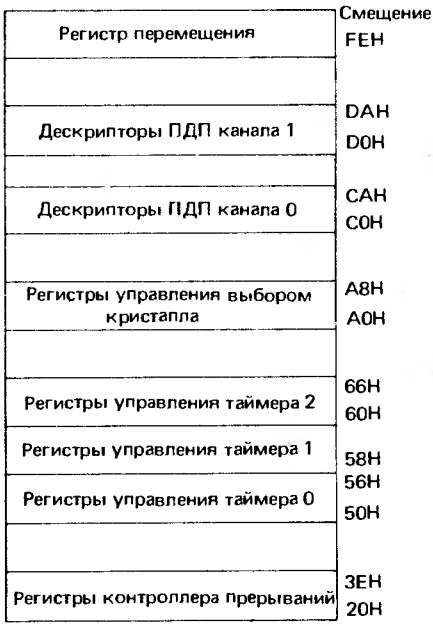


Рис. 12.5. Карта регистров управления

ное размещение блока регистров управления, но его можно переместить во время инициализации системы, изменяя содержимое регистра со смещением FE (т. е. базового адреса регистров управления).

Рис. 12.6 иллюстрирует типичную конфигурацию малой системы на базе

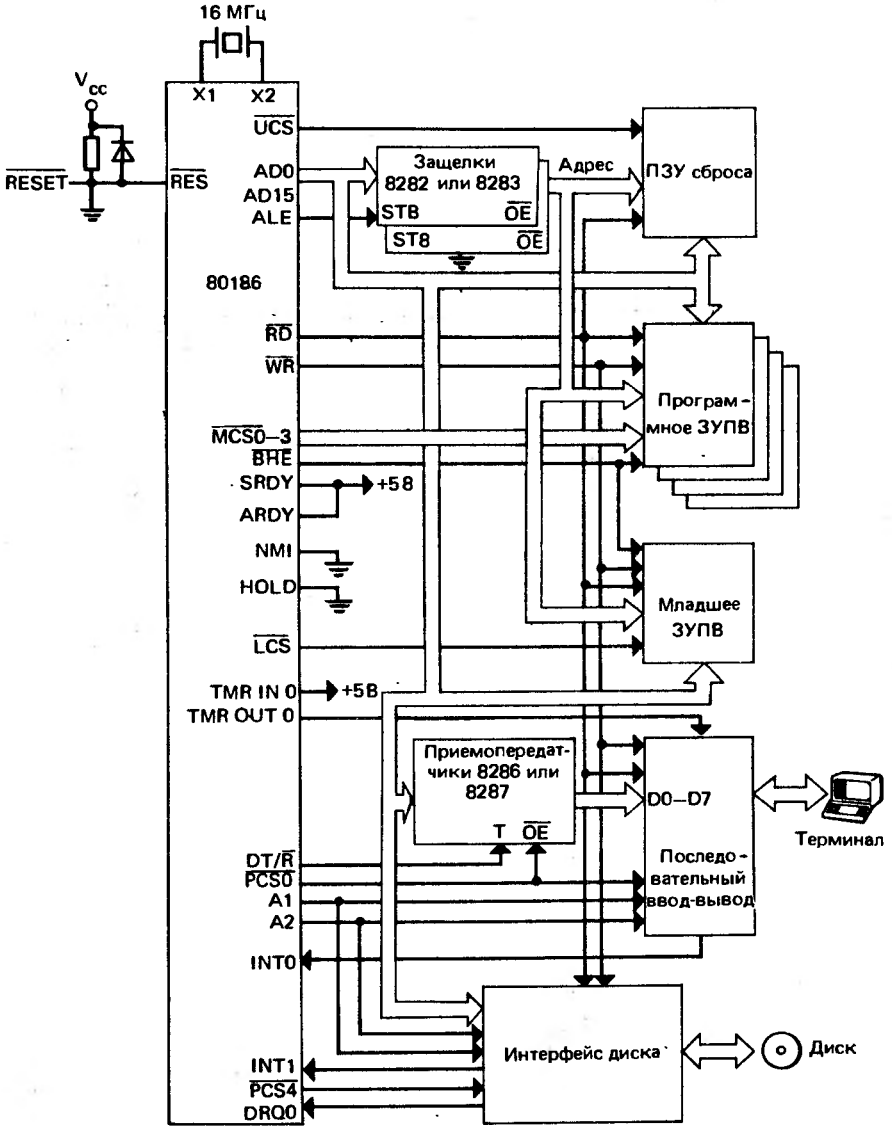


Рис. 12.6. Небольшая система на базе процессора 80186

процессора 80186, а более сложная система, которая может подключаться к шине с несколькими ведущими, показана на рис. 12.7. Отметим в обоих случаях сокращение числа микросхем (корпусов).

Кроме наличия внутренних компонент, реализующих разнообразные функции, процессор 80186 имеет вдвое более высокую производительность

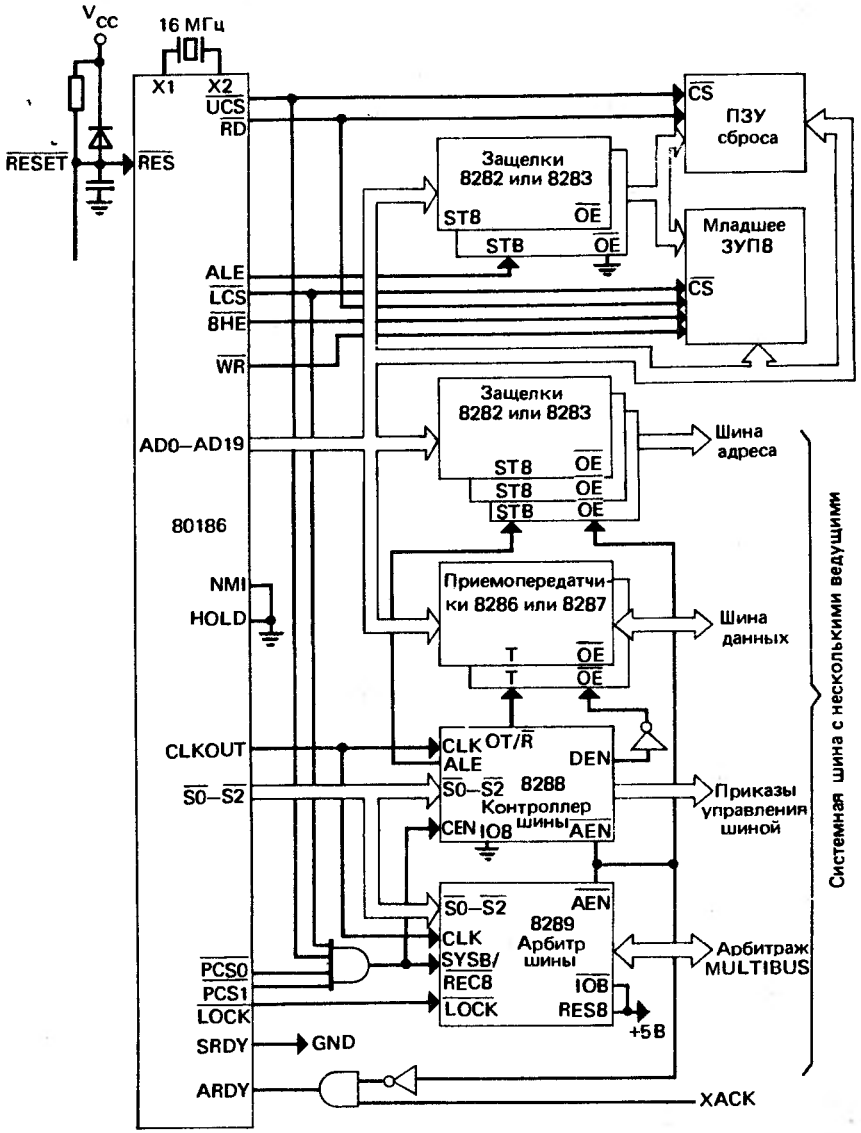


Рис. 12.7. Процессор 80186 в мультипроцессорной системе с несколькими ведущими

по сравнению с микропроцессором 8086, а его линии адреса/данных и некоторых сигналов управления имеют вдвое большую нагрузочную способность. Наконец, в ней предусмотрены следующие дополнительные команды:

Непосредственно включить (в стек) и извлечь (из стека). Для включения и извлечения непосредственных значений.

Включить все (PUSHA) и извлечь все (POPA). Для включения и извлечения всех регистров одной командой.

Знаковое непосредственное умножение. Для умножения на непосредственный операнд.

Сдвиг/циклический сдвиг на счетчик. Счетчиком является непосредственный операнд.

Ввод цепочки (INS) и вывод цепочки (OUTS). Позволяет вводит и выводить цепочки, пользуясь префиксом повторения REP.

Войти (ENTER) и выйти (LEAVE) из процедуры. Команда ENTER определяет, сколько байт динамической памяти распределить стековому кадру для вызываемой процедуры. Она также определяет уровень вложения процедуры и задает, сколько указателей ЦП будут копироваться в новом кадре из предыдущего кадра. Команда LEAVE выполняет действия, обратные действиям команды ENTER. Данные команды помогают реализовать языки высокого уровня с блоковой структурой.

Обнаружить значение вне диапазона (BOUND). Контролирует содержимое заданного в команде регистра в границах, определяемых адресуемым командой двойным словом. Применяется в основном для задания границ массивов.

12.3. МИКРОСХЕМА 80286

Процессор 80286 является усовершенствованным вариантом микропроцессора 8086, содержащим схемы управления памятью и ее защиты. Как и микросхема 80186, он оформлен в корпусе с 68 контактами, расположенными в четыре ряда. Выпускаются модели процессоров с частотой синхронизации 8 и 10 МГц. Все внутренние регистры процессора 80286 показаны на рис. 12.8. Помимо регистров микропроцессора 8086 введены следующие регистры:

Слово состояния машины (MSW). Для хранения дополнительных бит состояния, связанных с расширенными возможностями процессора 80286. Бит разрешения защиты (бит 0) в этом регистре определяет работу процессора в режиме реального адреса или в режиме защищенного виртуального адреса. После сброса процессора этот бит содержит 1, что соответствует работе в режиме реального адреса. Для перехода в режим с защитой этот бит необходимо сбросить. После операции сброса вновь перевести его в состояние 1 может только сброс процессора.

Расширения сегментных регистров. Для хранения текущих сегментных дескрипторов.

Регистр задачи (TR). Адресует системный дескриптор сегмента, который определяет базовый адрес, права доступа и размер сегмента состояния текущей задачи.

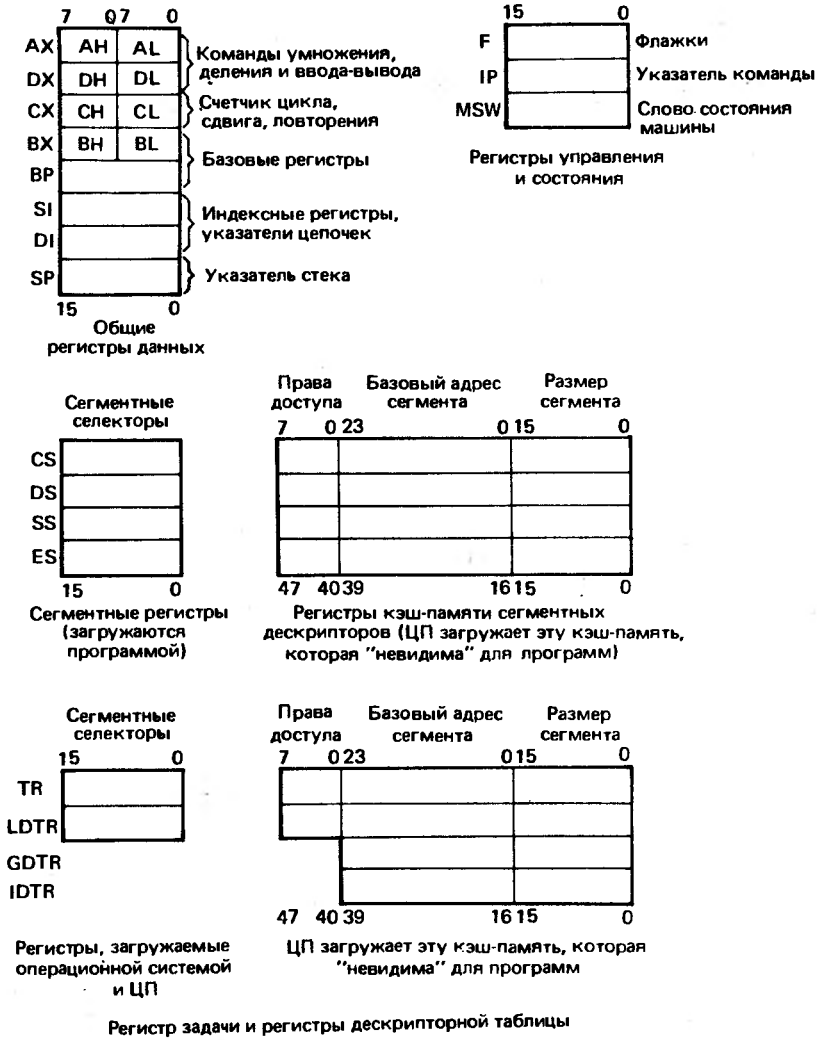


Рис. 12.8. Регистры процессора 80286

Регистры дескрипторной таблицы (LDTR, GDTR и IDTR). Адресуют дескрипторные таблицы, которые содержат дескрипторы различных сегментов.

В режиме реального адреса процессор 80286 функционирует так же, как микропроцессор 8086, и дополнительные регистры не используются. В режиме с защитой, если при обращении к памяти используемый сегментный регистр не изменялся, адрес формируется сложением эффективного адреса и 24-битного базового адреса из расширения (см. § 7.5). Если же сегментный

регистр изменялся, старшие 13 бит сегментного регистра используются как индекс, который прибавляется к содержимому соответствующего регистра дескрипторной таблицы для получения местоположения дескриптора, загружаемого в расширение сегментного регистра. После загрузки нового расширения эффективный адрес (как и раньше) прибавляется к базовому адресу для получения физического адреса операнда в памяти.

Регистр задачи используется при переключении задач. При переключении все состояние машины запоминается в области, определяемой системным дескриптором сегмента, на который указывает регистр задачи. Затем в регистр задачи загружается адрес области запоминания новой задачи, а из этой области загружается новое состояние машины.

Система команд процессора 80286 включает в себя все команды микропроцессора 8086, дополнительные команды, приведенные выше для процессора 80186, и 16 команд для работы со средствами управления памятью. Эти 16 команд показаны на рис. 12.9.

Разводка контактов корпуса процессора 80286 приведена на рис. 12.10. Отметим, что в отличие от микропроцессоров 8086/8088 линии данных и адресов не мультиплексируются: имеется 24 линии адреса A_{23}-A_0 и 16 линий данных D_{15}-D_0. (В режиме реального адреса, когда процессор 80286 формирует 20-битный адрес, линии A_{23}-A_{20} не используются.) Имеются четыре линии состояния: COD/$\overline{\text{INTA}}$, M/$\overline{\text{IO}}$, $\overline{\text{S1}}$ и $\overline{\text{S0}}$ (см. табл. 12.1). Сигналы $\overline{\text{BHE}}$, $\overline{\text{LOCK}}$, CLK, INTR, NMI, RESET и $\overline{\text{READY}}$ выполняют те же функции, что и в микропроцессоре 8086, работающем в максимальном режиме (но сигнал готовности инвертирован). Сигналы HOLD и HLDA аналогичны соответствующим сигналам микропроцессора 8086 в минимальном режиме. Сигналы

LGDT – Загрузить регистр глобальной дескрипторной таблицы
LIDT – Загрузить регистр дескрипторной таблицы прерываний
LLDT – Загрузить регистр локальной дескрипторной таблицы
LMSW – Загрузить регистр слова состояния машины
LTR – Загрузить регистр задачи
SGDT – Запомнить регистр глобальной дескрипторной таблицы
SIDT – Запомнить регистр дескрипторной таблицы прерываний
SLDT – Запомнить регистр локальной дескрипторной таблицы
SMSW – Запомнить регистр слова состояния машины
STR – Запомнить регистр задачи
ARPL – Скорректировать запрошенный уровень привилегий
CTS – Сбросить флажок переключения задачи
LAR – Загрузить права доступа
ISL – Загрузить предел сегмента
VERR – Проверить доступ по считыванию
VERW – Проверить доступ по записи

Рис. 12.9. Команды, относящиеся к управлению памятью

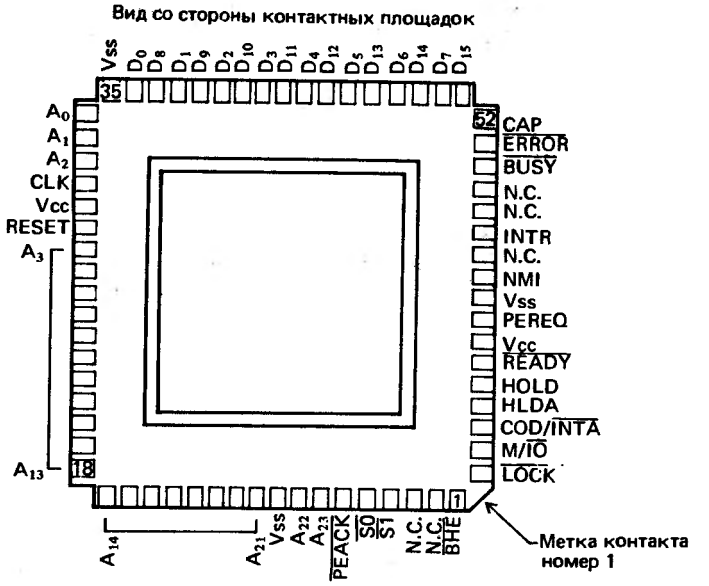


Рис. 12.10. Разводка контактов корпуса и определения состояний процессора 80286

Примечание. Площадки, отмеченные N.C, подключать нельзя

Таблица 12.1.

Определение состояния цикла шины процессора 80286

COD/$\overline{\text{INTA}}$	M/$\overline{\text{IO}}$	$\overline{\text{S1}}$	$\overline{\text{S0}}$	Инициируемый цикл шины
0 (низкий	0	0	0	Подтверждение прерывания
0 уровень)	0	0	1	Зарезервирован
0	0	1	0	Зарезервирован
0	0	1	1	Отсутствует
0	1	0	0	Если A1 = 1, то останов; иначе выключение
0	1	0	1	Считывание из памяти данных
0	1	1	0	Запись в память данных
0	1	1	1	Отсутствует
1 (высокий	0	0	0	Зарезервирован
1 уровень)	0	0	1	Считывание ввода-вывода
1	0	1	0	Запись ввода-вывода
1	0	1	1	Отсутствует
1	1	0	0	Зарезервирован
1	1	0	1	Считывание из памяти команд
1	1	1	0	Зарезервирован
1	1	1	1	Отсутствует

Состояние цикла шины показывает инициирование цикла шины, а вместе с сигналами M/$\overline{\text{IO}}$ и COD/$\overline{\text{INTA}}$ определяет тип цикла шины. Шина находится в состоянии T_S, когда любой один или оба их этих сигналов имеют низкие уровни. Сигналы $\overline{\text{S1}}$ и $\overline{\text{S0}}$ имеют активный низкий уровень и переводятся в высокоимпедансное состояние во время подтверждения запроса шины.

PEREQ, $\overline{\text{PEACK}}$, $\overline{\text{BUSY}}$ и $\overline{\text{ERROR}}$ применяются, когда процессор 80286 подключается к микросхеме расширения процессора, которую мы не рассматриваем.

Процессор 80286 имеет одно напряжение питания +5 В, которое подается на входы V_{CC}, а входы V_{SS} заземляются. На контакт CAP подключается кон-

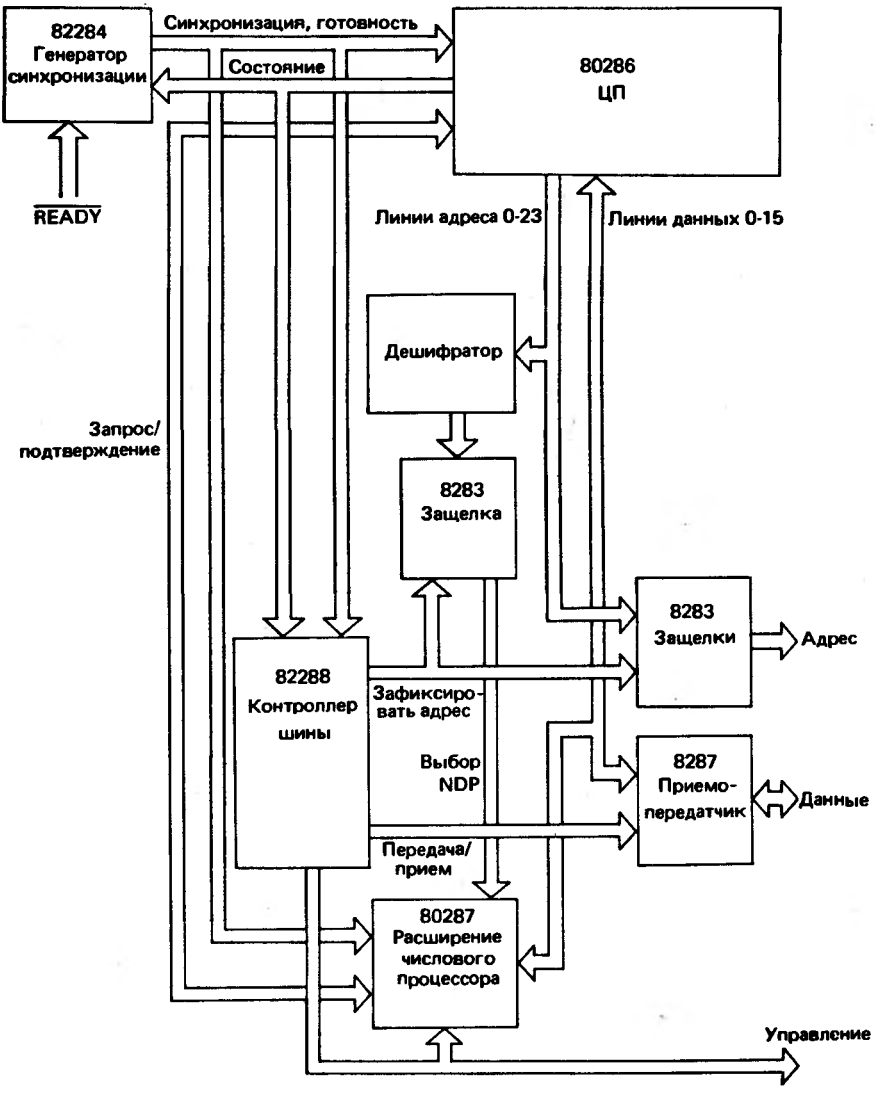


Рис. 12.11. Система обработки числовых даниых на базе процессора 80286

денсатор 0,047 мкФ ± 20 % (с напряжением 12 В), который служит фильтром подложки.

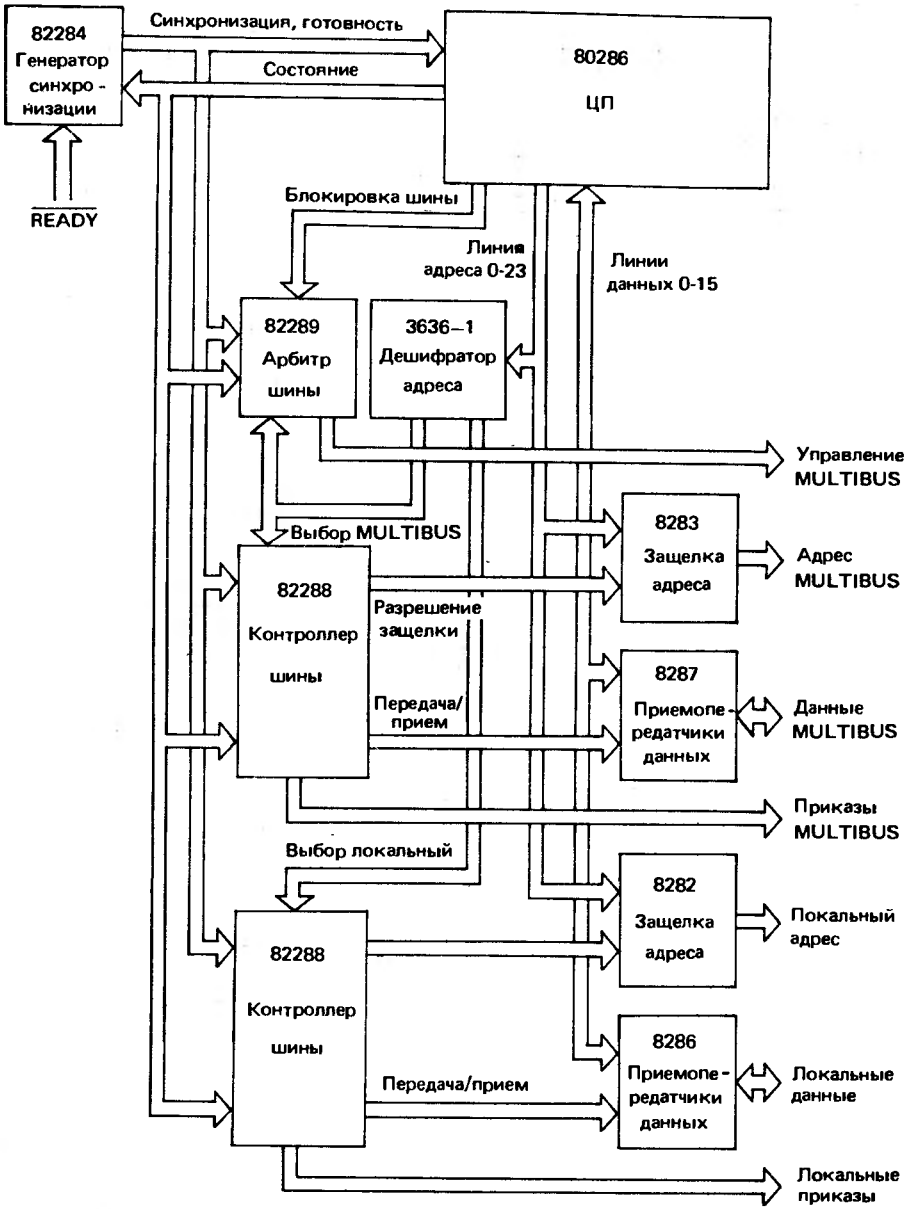


Рис. 12.12. Процессор 80286 в системе с несколькими ведущими шины

Кроме средств управления памятью, увеличенной емкости памяти (благодаря 24-битным адресам) и дополнительных команд, производительность процессора 80286 в мультипрограммной среде в шесть раз выше производительности микропроцессора 8086. Имеется процессор числовых данных 80287, совместимый с процессором 80286. На рис. 12.11 и 12.12 показано, каким образом процессор 80286 объединяется с процессором числовых данных и работает в конфигурации с несколькими ведущими шины. Генератор синхронизации 82284, контроллер шины 82288 и арбитр шины 82289 являются аналогами рассмотренных нами микросхем 8284A, 8288 и 8289.

ПРИЛОЖЕНИЕ

СИСТЕМА КОМАНД МИКРОПРОЦЕССОРОВ 8086/8088

В первых двух столбцах табл. П1 приведены мнемоника (столбец 1) и краткое описание (столбец 2) каждой команды. Столбец 3 показывает число тактов синхронизации, необходимых для выполнения команды. В микропроцессоре 8086 на передачу каждого слова с нечетным адресом следует добавить четыре такта синхронизации, а в микропроцессоре 8088 — четыре такта синхронизации на передачу каждого слова. Для команд условных переходов и циклов приведены по два числа тактов синхронизации; одно из них, находящееся перед наклонной чертой, соответствует ситуации, когда переход не предпринимается, а второе — когда переход выполняется. Столбец 4 содержит число байт или диапазон числа байт команды, а столбец 5 показывает возможные состояния флажков со следующими условными обозначениями:

— флажок не модифицируется,
0 сбрасывается в нуль,
1 устанавливается в единицу,
x устанавливается или сбрасывается в соответствии с результатом,
u не определен,
r восстанавливается прежнее запомненное состояние.

ТАБЛИЦА П1. СИСТЕМА КОМАНД МИКРОПРОЦЕССОРОВ 8086/8088

МНЕМОНИКА	ОПИСАНИЕ	ЧИСЛО ТАКТОВ СИНХРОНИЗАЦИИ	ЧИСЛО БАЙТ	ФЛАЖКИ O D I T S Z A P C
AAA	ASCII КОРРЕКЦИЯ ДЛЯ СЛОЖЕНИЯ	4	1	U — — — U U X U X
AAD	ASCII КОРРЕКЦИЯ ДЛЯ ДЕЛЕНИЯ	60	2	U — — — X X U X U
AAM	ASCII КОРРЕКЦИЯ ДЛЯ УМНОЖЕНИЯ	83	2	U — — — X X U X U
AAS	ASCII КОРРЕКЦИЯ ДЛЯ ВЫЧИТАНИЯ	4	1	U — — — U U X U X
ADC	ASCII КОРРЕКЦИЯ ДЛЯ СЛОЖЕНИЯ	4	1	U — — — U U X U X
ADC	СЛОЖИТЬ С ПЕРЕНОСОМ			X — — — X X X X X
	РЕГИСТР - РЕГИСТР	3	2	
	ПАМЯТЬ - РЕГИСТР	9+EA	2-4	
	РЕГИСТР - ПАМЯТЬ	16+EA	2-4	
	НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД - РЕГИСТР	4	3-4	
	НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД - ПАМЯТЬ	17+EA	3-6	
	НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД - АККУМУЛЯТОР	4	2-3	

МНЕМОНИКА	ОПИСАНИЕ	ЧИСЛО ТАКТОВ СИНХРОНИЗАЦИИ	ЧИСЛО БАЙТ	ФЛАЖКИ O D I T S Z A P C
ADD	СЛОЖИТЬ:			X - - - X X X X X
	РЕГИСТР - РЕГИСТР	3	2	
	ПАМЯТЬ - РЕГИСТР	9+EA	2-4	
	РЕГИСТР - ПАМЯТЬ	16+EA	2-4	
	НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД - РЕГИСТР	4	3-4	
	НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД - ПАМЯТЬ	17+EA	3-6	
	НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД - АККУМУЛЯТОР	4	2-3	
AND	ОБ'ЕДИНИТЬ ПО "И" (КОН'ЮНКЦИЯ)			0 - - - X X U X 0
	РЕГИСТР - РЕГИСТР	3	2	
	ПАМЯТЬ - РЕГИСТР	9+EA	2-4	
	РЕГИСТР - ПАМЯТЬ	16+EA	2-4	
	НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД - РЕГИСТР	4	3-4	
	НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД - ПАМЯТЬ	17+EA	3-6	
	НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД - АККУМУЛЯТОР	4	2-3	
CALL	ВЫЗОВ ПРОЦЕДУРЫ			- - - - - - - - -
	ВНУТРИСЕГМЕНТНЫЙ ПРЯМОЙ	19	3	
	ВНУТРИСЕГМЕНТНЫЙ КОСВЕННЫЙ ЧЕРЕЗ РЕГИСТР	16	2	
	ВНУТРИСЕГМЕНТНЫЙ КОСВЕННЫЙ ЧЕРЕЗ ПАМЯТЬ	21+EA	2-4	
	МЕЖСЕГМЕНТНЫЙ ПРЯМОЙ	28	5	
	МЕЖСЕГМЕНТНЫЙ КОСВЕННЫЙ	37+EA	2-4	
CBW	ПРЕОБРАЗОВАТЬ БАЙТ В СЛОВО	2	1	- - - - - - - - -
CLC	СБРОСИТЬ ФЛАЖОК ПЕРЕНОСА	2	1	- - - - - - - - 0
CLD	СБРОСИТЬ ФЛАЖОК НАПРАВЛЕНИЯ	2	1	- 0 - - - - - - -
CLI	СБРОСИТЬ ФЛАЖОК ПРЕРЫВАНИЯ	2	1	- - 0 - - - - - -
CMC	ИНВЕРТИРОВАТЬ ФЛАЖОК ПЕРЕНОСА	2	1	- - - - - - - - X
CMP	СРАВНИТЬ			X - - - X X X X X
	РЕГИСТР - РЕГИСТР	3	2	
	ПАМЯТЬ - РЕГИСТР	9+EA	2-4	
	РЕГИСТР - ПАМЯТЬ	9+EA	2-4	
	НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД - РЕГИСТР	4	3-4	
	НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД - ПАМЯТЬ	10+EA	3-6	
	НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД - АККУМУЛЯТОР	4	2-3	
CMPS/ CMPSB/ CMPSW	СРАВНИТЬ ЦЕПОЧКИ/ СРАВНИТЬ ЦЕПОЧКИ БАЙТ/ СРАВНИТЬ ЦЕПОЧКИ СЛОВ	1		X - - - X X X X X
	БЕЗ ПОВТОРЕНИЯ	22		
	С ПОВТОРЕНИЕМ	9+22/ПОВТОР.		
CWD	ПРЕОБРАЗОВАТЬ СЛОВО В ДВОЙНОЕ СЛОВО	5	1	- - - - - - - - -
DAA	ДЕСЯТИЧНАЯ КОРРЕКЦИЯ ДЛЯ СЛОЖЕНИЯ	4	1	U - - - X X X X X
DAS	ДЕСЯТИЧНАЯ КОРРЕКЦИЯ ДЛЯ ВЫЧИТАНИЯ	4	1	U - - - X X X X X
DEC	ДЕКРЕМЕНТ НА 1			X - - - X X X X -
	16-БИТНЫЙ РЕГИСТР	2	1	
	8-БИТНЫЙ РЕГИСТР	3	2	
	ПАМЯТЬ	15+EA	2-4	
DIV	РАЗДЕЛИТЬ БЕЗ ЗНАКА			U - - - U U U U U
	8-БИТНЫЙ РЕГИСТР	80-90	2	
	16-БИТНЫЙ РЕГИСТР	144-162	2	
	8-БИТНАЯ ПАМЯТЬ	(86-96) +EA	2-4	
	16-БИТНАЯ ПАМЯТЬ	(150-168) +EA	2-4	
ESC	ПЕРЕКЛЮЧИТЬСЯ НА СОПРОЦЕССОР			- - - - - - - - -
	РЕГИСТР	2	2	
	ПАМЯТЬ	8+EA	2-4	
HLT	ОСТАНОВИТЬ	2	1	- - - - - - - - -
IDIV	РАЗДЕЛИТЬ СО ЗНАКОМ			U - - - U U U U U
	8-БИТНЫЙ РЕГИСТР	101-112	2	
	16-БИТНЫЙ РЕГИСТР	165-184	2	
	8-БИТНАЯ ПАМЯТЬ	(107-118) +EA	2-4	
	16-БИТНАЯ ПАМЯТЬ	(171-190) +EA	2-4	
IMUL	УМНОЖИТЬ СО ЗНАКОМ			X - - - U U U U X
	8-БИТНЫЙ РЕГИСТР	80-98	2	
	16-БИТНЫЙ РЕГИСТР	128-154	2	
	8-БИТНАЯ ПАМЯТЬ	(86-104) +EA	2-4	
	16-БИТНАЯ ПАМЯТЬ	(134-160) +EA	2-4	
IN	ВВЕСТИ ИЗ ПОРТА ВВОДА-ВЫВОДА			- - - - - - - - -
	ФИКСИРОВАННЫЙ ПОРТ	10	2	
	ПЕРЕМЕННЫЙ ПОРТ	8	1	

МНЕМОНИКА	ОПИСАНИЕ	ЧИСЛО ТАКТОВ СИНХРОНИЗАЦИИ	ЧИСЛО БАЙТ	ФЛАЖКИ O D I T S Z A P C
INC	ИНКРЕМЕНТ НА 1			X - - - X X X X -
	16-БИТНЫЙ РЕГИСТР	2	1	
	8-БИТНЫЙ РЕГИСТР	3	2	
	ПАМЯТЬ	15+EA	2-4	
INT	ПРЕРЫВАНИЕ			- - 0 0 - - - - -
	ТИП = 3	52	1	
	ТИП ≠ 3	51	2	
INTO	ПРЕРЫВАНИЕ ПРИ ПЕРЕПОЛНЕНИИ			- - 0 0 - - - - -
	ПРЕРЫВАНИЕ ЕСТЬ	53		
	ПРЕРЫВАНИЯ НЕТ	4		
IRET	ВОЗВРАТИТЬСЯ ИЗ ПРЕРЫВАНИЯ	24	1	R R R R R R R R R
JA/	ПЕРЕЙТИ, ЕСЛИ ВЫШЕ/	16/4	2	- - - - - - - - -
JNBE	ПЕРЕЙТИ, ЕСЛИ НЕ НИЖЕ ИЛИ РАВНО			
JAE/	ПЕРЕЙТИ, ЕСЛИ ВЫШЕ ИЛИ РАВНО/	16/4	2	- - - - - - - - -
JNB/	ПЕРЕЙТИ, ЕСЛИ НЕ НИЖЕ/			
JNC	ПЕРЕЙТИ, ЕСЛИ НЕТ ПЕРЕНОСА			
JB/	ПЕРЕЙТИ, ЕСЛИ НИЖЕ/	16/4	2	- - - - - - - - -
JNAE/	ПЕРЕЙТИ, ЕСЛИ НЕ ВЫШЕ ИЛИ РАВНО/			
JC	ПЕРЕЙТИ, ЕСЛИ ЕСТЬ ПЕРЕНОС			
JBE/	ПЕРЕЙТИ, ЕСЛИ НИЖЕ ИЛИ РАВНО/	16/4	2	- - - - - - - - -
JNA	ПЕРЕЙТИ, ЕСЛИ НЕ ВЫШЕ			
JCXZ	ПЕРЕЙТИ, ЕСЛИ CX РАВЕН НУЛЮ	18/6	2	- - - - - - - - -
JE/	ПЕРЕЙТИ, ЕСЛИ РАВНО/	16/4	2	- - - - - - - - -
JZ	ПЕРЕЙТИ, ЕСЛИ НУЛЬ			
JG/	ПЕРЕЙТИ, ЕСЛИ БОЛЬШЕ/	16/4	2	- - - - - - - - -
JNLE	ПЕРЕЙТИ, ЕСЛИ НЕ МЕНЬШЕ ИЛИ РАВНО			
JGE/	ПЕРЕЙТИ, ЕСЛИ БОЛЬШЕ ИЛИ РАВНО/	16/4	2	- - - - - - - - -
JNL	ПЕРЕЙТИ, ЕСЛИ НЕ МЕНЬШЕ			
JL/	ПЕРЕЙТИ, ЕСЛИ МЕНЬШЕ/	16/4	2	- - - - - - - - -
JNGE	ПЕРЕЙТИ, ЕСЛИ НЕ БОЛЬШЕ ИЛИ РАВНО			
JLE/	ПЕРЕЙТИ, ЕСЛИ МЕНЬШЕ ИЛИ РАВНО/	16/4	2	- - - - - - - - -
JNG	ПЕРЕЙТИ, ЕСЛИ НЕ БОЛЬШЕ			- - - - - - - - -
JMP	ПЕРЕХОД			
	ВНУТРИСЕГМЕНТНЫЙ ПРЯМОЙ КОРОТКИЙ	15	2	
	ВНУТРИСЕГМЕНТНЫЙ ПРЯМОЙ	15	3	
	МЕЖСЕГМЕНТНЫЙ ПРЯМОЙ	15	5	
	ВНУТРИСЕГМЕНТНЫЙ КОСВЕННЫЙ ЧЕРЕЗ ПАМЯТЬ	18+EA	2-4	
	ВНУТРИСЕГМЕНТНЫЙ КОСВЕННЫЙ ЧЕРЕЗ РЕГИСТР	11	2	
	МЕЖСЕГМЕНТНЫЙ КОСВЕННЫЙ	24+EA	2-4	
JNE/	ПЕРЕЙТИ, ЕСЛИ НЕ РАВНО/	16/4	2	- - - - - - - - -
JNZ	ПЕРЕЙТИ, ЕСЛИ НЕ НУЛЬ			
JNO	ПЕРЕЙТИ, ЕСЛИ НЕТ ПЕРЕПОЛНЕНИЯ	16/4	2	- - - - - - - - -
JNP/	ПЕРЕЙТИ, ЕСЛИ НЕТ ПАРИТЕТА/	16/4	2	- - - - - - - - -
JPO	ПЕРЕЙТИ, ЕСЛИ ПАРИТЕТ НЕЧЕТНЫЙ			
JNS	ПЕРЕЙТИ, ЕСЛИ НЕТ ЗНАКА	16/4	2	- - - - - - - - -
JO	ПЕРЕЙТИ, ЕСЛИ ЕСТЬ ПЕРЕПОЛНЕНИЕ	16/4	2	- - - - - - - - -
JP/	ПЕРЕЙТИ, ЕСЛИ ЕСТЬ ПАРИТЕТ/	16/4	2	- - - - - - - - -
JPE	ПЕРЕЙТИ, ЕСЛИ ПАРИТЕТ ЧЕТНЫЙ			
JS	ПЕРЕЙТИ, ЕСЛИ ЕСТЬ ЗНАК	16/4	2	- - - - - - - - -
LAHF	ЗАГРУЗИТЬ AH ИЗ ФЛАЖКОВ	4	1	- - - - - - - - -
LDS	ЗАГРУЗИТЬ УКАЗАТЕЛЬ, ИСПОЛЬЗУЯ DS	16+EA	2-4	- - - - - - - - -
LEA	ЗАГРУЗИТЬ ЭФФЕКТИВНЫЙ АДРЕС	2+EA	2-4	- - - - - - - - -
LES	ЗАГРУЗИТЬ УКАЗАТЕЛЬ, ИСПОЛЬЗУЯ ES	16+EA	2-4	- - - - - - - - -
LOCK	ЗАБЛОКИРОВАТЬ ШИНУ	2	1	- - - - - - - - -
LODS/	ЗАГРУЗИТЬ ЦЕПОЧКУ/		1	- - - - - - - - -
LODSB/	ЗАГРУЗИТЬ ЦЕПОЧКУ БАЙТ/			
LODSW	ЗАГРУЗИТЬ ЦЕПОЧКУ СЛОВ			
	БЕЗ ПОВТОРЕНИЯ	12		
	С ПОВТОРЕНИЕМ	9+13/ПОВТОР.		
LOOP	ЗАЦИКЛИТЬ	17/5	2	- - - - - - - - -
LOOPE/	ЗАЦИКЛИТЬ, ЕСЛИ РАВНО/	18/6	2	- - - - - - - - -
LOOPZ	ЗАЦИКЛИТЬ, ЕСЛИ НУЛЬ			
LOOPNE/	ЗАЦИКЛИТЬ, ЕСЛИ НЕ РАВНО/	19/5	2	- - - - - - - - -
LOOPNZ	ЗАЦИКЛИТЬ, ЕСЛИ НЕ НУЛЬ			
MOV	ПЕРЕСЛАТЬ			- - - - - - - - -
	АККУМУЛЯТОР - ПАМЯТЬ	10	3	
	ПАМЯТЬ - АККУМУЛЯТОР	10	3	
	РЕГИСТР - РЕГИСТР	2	2	
	ПАМЯТЬ - РЕГИСТР	8+EA	2-4	
	РЕГИСТР - ПАМЯТЬ	9+EA	2-4	
	НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД - РЕГИСТР	4	2-3	
	НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД - ПАМЯТЬ	10+EA	3-6	
	РЕГИСТР - РЕГИСТРЫ SS, DS ИЛИ ES	2	2	
	ПАМЯТЬ - РЕГИСТРЫ SS, DS ИЛИ ES	8+EA	2-4	
	СЕГМЕНТНЫЙ РЕГИСТР - РЕГИСТР	2	2	
	СЕГМЕНТНЫЙ РЕГИСТР - ПАМЯТЬ	9+EA	2-4	

МНЕМОНИКА	ОПИСАНИЕ	ЧИСЛО ТАКТОВ СИНХРОНИЗАЦИИ	ЧИСЛО БАЙТ	ФЛАЖКИ O D I T S Z A P C
MOVS/	ПЕРЕСЛАТЬ ЦЕПОЧКУ/		1	- - - - - - - - - -
MOVSB/	ЗАГРУЗИТЬ ЦЕПОЧКУ БАЙТ/			
MOVSW	ЗАГРУЗИТЬ ЦЕПОЧКУ СЛОВ			
	БЕЗ ПОВТОРЕНИЯ	18		
	С ПОВТОРЕНИЕМ	9+17/ПОВТОР.		
MUL	УМНОЖИТЬ БЕЗ ЗНАКА			X - - - U U U U X
	8-БИТНЫЙ РЕГИСТР	70-77	2	
	16-БИТНЫЙ РЕГИСТР	118-133	2	
	8-БИТНАЯ ПАМЯТЬ	(76-83)		
		+EA	2-4	
	16-БИТНАЯ ПАМЯТЬ	(124-139)		
		+EA	2-4	
NEG	ИЗМЕНИТЬ ЗНАК			X - - - X X X X X
	РЕГИСТР	3	2	
	ПАМЯТЬ	16+EA	2-4	
NOP	ХОЛОСТАЯ КОМАНДА	3	1	- - - - - - - - - -
NOT	ИНВЕРТИРОВАТЬ (ОТРИЦАНИЕ)			- - - - - - - - - -
	РЕГИСТР	3	2	
	ПАМЯТЬ	16+EA	2-4	
OR	ОБ'ЕДИНИТЬ ПО "ИЛИ" (ДИЗ'ЮНКЦИЯ)			0 - - - X X U X 0
	РЕГИСТР - РЕГИСТР	3	2	
	ПАМЯТЬ - РЕГИСТР	9+EA	2-4	
	РЕГИСТР - ПАМЯТЬ	16+EA	2-4	
	НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД - АККУМУЛЯТОР	4	2-3	
	НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД - РЕГИСТР	4	3-4	
	НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД - ПАМЯТЬ	17+EA	3-6	
OUT	ВЫВЕСТИ В ПОРТ ВВОДА-ВЫВОДА			- - - - - - - - - -
	ФИКСИРОВАННЫЙ ПОРТ	10	2	
	ПЕРЕМЕННЫЙ ПОРТ	8	1	
POP	ИЗВЛЕЧЬ СЛОВО ИЗ СТЕКА			- - - - - - - - - -
	В РЕГИСТР	8	1	
	В СЕГМЕНТНЫЙ РЕГИСТР SS, DS ИЛИ ES	8	1	
	В ПАМЯТЬ	17+Ea	2-4	
POPF	ИЗВЛЕЧЬ ФЛАЖКИ ИЗ СТЕКА	8	1	R R R R R R R R R
PUSH	ВКЛЮЧИТЬ СЛОВО В СТЕК			- - - - - - - - - -
	РЕГИСТР	11	1	
	СЕГМЕНТНЫЙ РЕГИСТР	10	1	
	ПАМЯТЬ	16+EA	2-4	
PUSHF	ВКЛЮЧИТЬ ФЛАЖКИ В СТЕК	10	1	- - - - - - - - - -
RCL	ЦИКЛИЧЕСКИ СДВИНУТЬ ВЛЕВО ЧЕРЕЗ ПЕРЕНОС			X - - - - - - - - X
	РЕГИСТР (СДВИГ НА ОДИН БИТ)	2	2	
	РЕГИСТР (ПЕРЕМЕННЫЙ СДВИГ)	8+4/БИТ	2	
	ПАМЯТЬ (СДВИГ НА ОДИН БИТ)	15+EA	2-4	
	ПАМЯТЬ (ПЕРЕМЕННЫЙ СДВИГ)	20+EA+4/БИТ	2-4	
RCR	ЦИКЛИЧЕСКИ СДВИНУТЬ ВПРАВО ЧЕРЕЗ ПЕРЕНОС			X - - - - - - - - X
	РЕГИСТР (СДВИГ НА ОДИН БИТ)	2	2	
	РЕГИСТР (ПЕРЕМЕННЫЙ СДВИГ)	8+4/БИТ	2	
	ПАМЯТЬ (СДВИГ НА ОДИН БИТ)	15+EA	2-4	
	ПАМЯТЬ (ПЕРЕМЕННЫЙ СДВИГ)	20+EA+4/БИТ	2-4	
REP	ПОВТОРИТЬ ЦЕПОЧЕЧНУЮ ОПЕРАЦИЮ	2	1	- - - - - - - - - -
REPE/	ПОВТОРЯТЬ ОПЕРАЦИЮ, ПОКА РАВНО/	2	1	- - - - - - - - - -
REPZ	ПОВТОРЯТЬ ОПЕРАЦИЮ, ПОКА НУЛЬ			
REPNE/	ПОВТОРЯТЬ ОПЕРАЦИЮ, ПОКА НЕ РАВНО/	2	1	- - - - - - - - - -
REPNZ	ПОВТОРЯТЬ ОПЕРАЦИЮ, ПОКА НЕ НУЛЬ			
RET	ВОЗВРАТ ИЗ ПРОЦЕДУРЫ			- - - - - - - - - -
	ВНУТРИСЕГМЕНТНЫЙ	8	1	
	ВНУТРИСЕГМЕНТНЫЙ С КОНСТАНТОЙ	12	3	
	МЕЖСЕГМЕНТНЫЙ	18	1	
	МЕЖСЕГМЕНТНЫЙ С КОНСТАНТОЙ	17	3	
ROL	ЦИКЛИЧЕСКИ СДВИНУТЬ ВЛЕВО			X - - - - - - - - X
	РЕГИСТР (СДВИГ НА ОДИН БИТ)	2	2	
	РЕГИСТР (ПЕРЕМЕННЫЙ СДВИГ)	8+4/БИТ	2	
	ПАМЯТЬ (СДВИГ НА ОДИН БИТ)	15+EA	2-4	
	ПАМЯТЬ (ПЕРЕМЕННЫЙ СДВИГ)	20+EA+4/БИТ	2-4	
ROR	ЦИКЛИЧЕСКИ СДВИНУТЬ ВПРАВО			X - - - - - - - - X
	РЕГИСТР (СДВИГ НА ОДИН БИТ)	2	2	
	РЕГИСТР (ПЕРЕМЕННЫЙ СДВИГ)	8+4/БИТ	2	
	ПАМЯТЬ (СДВИГ НА ОДИН БИТ)	15+EA	2-4	
	ПАМЯТЬ (ПЕРЕМЕННЫЙ СДВИГ)	20+EA+4/БИТ	2-4	
SAHF	ЗАПОМНИТЬ АН ВО ФЛАЖКАХ	4	1	- - - - R R R R R
SAL/	СДВИНУТЬ АРИФМЕТИЧЕСКИ ВЛЕВО/			X - - - X X U X X
SHL	СДВИНУТЬ ЛОГИЧЕСКИ ВЛЕВО			
	РЕГИСТР (СДВИГ НА ОДИН БИТ)	2	2	
	РЕГИСТР (ПЕРЕМЕННЫЙ СДВИГ)	8+4/БИТ	2	
	ПАМЯТЬ (СДВИГ НА ОДИН БИТ)	15+EA	2-4	
	ПАМЯТЬ (ПЕРЕМЕННЫЙ СДВИГ)	20+EA+4/БИТ	2-4	

МНЕМОНИКА	ОПИСАНИЕ	ЧИСЛО ТАКТОВ СИНХРОНИЗАЦИИ	ЧИСЛО БАЙТ	ФЛАЖКИ O D I T S Z A P C
SAR	СДВИНУТЬ АРИФМЕТИЧЕСКИ ВПРАВО			X – – – X X U X X
	РЕГИСТР (СДВИГ НА ОДИН БИТ)	2	2	
	РЕГИСТР (ПЕРЕМЕННЫЙ СДВИГ)	8+4/БИТ	2	
	ПАМЯТЬ (СДВИГ НА ОДИН БИТ)	15+EA	2-4	
	ПАМЯТЬ (ПЕРЕМЕННЫЙ СДВИГ)	20+EA+4/БИТ	2-4	
SBB	ВЫЧЕСТЬ С ЗАЕМОМ			X – – – X X X X X
	РЕГИСТР – РЕГИСТР	3	2	
	ПАМЯТЬ – РЕГИСТР	9+EA	2-4	
	РЕГИСТР – ПАМЯТЬ	16+EA	2-4	
	НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД – АККУМУЛЯТОР	4	2-3	
	НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД – РЕГИСТР	4	3-4	
	НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД – ПАМЯТЬ	17+EA	3-6	
SCAS/ SCASB/ SCASW	СКАНИРОВАТЬ ЦЕПОЧКУ/ СКАНИРОВАТЬ ЦЕПОЧКУ БАЙТ/ СКАНИРОВАТЬ ЦЕПОЧКУ СЛОВ			X – – – X X X X X
	БЕЗ ПОВТОРЕНИЯ	15		
	С ПОВТОРЕНИЕМ	9+15/ПОВТОР.		
SHR	СДВИНУТЬ ЛОГИЧЕСКИ ВПРАВО			X – – – X X U X X
	РЕГИСТР (СДВИГ НА ОДИН БИТ)	2	2	
	РЕГИСТР (ПЕРЕМЕННЫЙ СДВИГ)	8+4/БИТ	2	
	ПАМЯТЬ (СДВИГ НА ОДИН БИТ)	15+EA	2-4	
	ПАМЯТЬ (ПЕРЕМЕННЫЙ СДВИГ)	20+EA+4/БИТ	2-4	
STC	УСТАНОВИТЬ ФЛАЖОК ПЕРЕНОСА	2	1	– – – – – – – – 1
STD	УСТАНОВИТЬ ФЛАЖОК НАПРАВЛЕНИЯ	2	1	– 1 – – – – – – –
STI	УСТАНОВИТЬ ФЛАЖОК ПРЕРЫВАНИЯ	2	1	– – 1 – – – – – –
STOS/ STOSB/ STOSW	ЗАПОМНИТЬ ЦЕПОЧКУ/ ЗАПОМНИТЬ ЦЕПОЧКУ БАЙТ/ ЗАПОМНИТЬ ЦЕПОЧКУ СЛОВ		1	– – – – – – – – –
	БЕЗ ПОВТОРЕНИЯ	11		
	С ПОВТОРЕНИЕМ	9+10/ПОВТОР.		
SUB	ВЫЧЕСТЬ			X – – – X X X X X
	РЕГИСТР – РЕГИСТР	3	2	
	ПАМЯТЬ – РЕГИСТР	9+EA	2-4	
	РЕГИСТР – ПАМЯТЬ	16+EA	2-4	
	НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД – АККУМУЛЯТОР	4	2-3	
	НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД – РЕГИСТР	4	3-4	
	НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД – ПАМЯТЬ	17+EA	3-6	
TEST	ПРОВЕРИТЬ			0 – – – X X U X 0
	РЕГИСТР – РЕГИСТР	3	2	
	ПАМЯТЬ – РЕГИСТР	9+EA	2-4	
	НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД – АККУМУЛЯТОР	4	2-3	
	НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД – РЕГИСТР	5	3-4	
	НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД – ПАМЯТЬ	11+EA	3-6	
WAIT	ОЖИДАТЬ АКТИВНОГО СИГНАЛА $\overline{\text{TEST}}$	3+5	1	– – – – – – – – –
XCHG	ОБМЕНЯТЬ			– – – – – – – – –
	РЕГИСТР – АККУМУЛЯТОР	3	1	
	РЕГИСТР – ПАМЯТЬ	11+EA	2-4	
	РЕГИСТР – РЕГИСТР	4	2	
XLAT/ XLATB	ПРЕОБРАЗОВАТЬ	11	1	– – – – – – – – –
XOR	СЛОЖИТЬ ПО МОДУЛЮ 2 (ИСКЛЮЧАЮЩЕЕ-ИЛИ)			0 – – – X X U X 0
	РЕГИСТР – РЕГИСТР	3	2	
	ПАМЯТЬ – РЕГИСТР	9+EA	2-4	
	РЕГИСТР – ПАМЯТЬ	16+EA	2-4	
	НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД – АККУМУЛЯТОР	4	2-3	
	НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД – РЕГИСТР	4	3-4	
	НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД – ПАМЯТЬ	17+EA	3-6	

СПИСОК ЛИТЕРАТУРЫ

К главе 1

1. Krutz, Ronald L., *Microprocessors and Logic Design* (New York: John Wiley & Sons, Inc., 1980).
2. Wollesen, Donald L., "The Computer Component Process of the 80's," *Computer*, 13, no. 2 (February, 1980), 59–67.
3. Sumney, Larry W., "VLSI with a Vengeance," *IEEE Spectrum*, 17, no. 4 (April, 1980), 24–27.
4. *The 8086 Family User's Manual* (Santa Clara, Calif.: Intel Corporation, 1979).
5. Morse, Stephen P., Bruce W. Romenel, Stanley Mazor, and William B. Pohlman, "Intel Microprocessors—8008 to 8086," *Computer*, 13, no. 10 (October, 1980), 42–60.
6. Shima, Masatoshi, "Demystifying Microprocessor Design," *IEEE Spectrum*, 16, no. 7 (July, 1979), 22–30.
7. Balde, J. W., "Computer Packaging Workshop Explores Impact of LSI," *Computer*, 13, no. 11 (November, 1980), 100–102.
8. Sequin, C. H., "Instruction in MOS LSI Systems Design," *Computer*, 13, no. 3 (March, 1980), 67–73.
9. Gibson, Glenn A., and Yu-cheng Liu, *Microcomputers for Engineers and Scientists* (Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1980).
10. Blakeslee, Thomas R., *Digital Design with Standard MSI and LSI* (New York: John Wiley & Sons, Inc., 1975).

К главе 2

1. Morse, Stephen P., Bruce W. Ravenel, Stanley Mazor, and William P. Pohlman, "Intel Microprocessors—8008 to 8086," *Computer*, no. 10 (October, 1980), 42–60.
2. *MCS-86 User's Manual* (Santa Clara, Calif.: Intel Corporation, 1979).

К главе 3

1. *MCS-86 User's Manual* (Santa Clara, Calif.: Intel Corporation, 1979).
2. *The 8086 Family User's* Manual (Santa Clara, Calif.: Intel Corporation, 1979).
3. *MCS-86 Macro Assembly Language Reference Manual* (Santa Clara, Calif.: Intel Corporation, 1979).
4. Rector, Russell, and George Alexy, *The 8086 Book* (Berkeley, Calif.: Osborne/McGraw-Hill, 1980).
5. Morse, Stephen P., *The 8086 Primer: An Introduction To Its Architecture, System Design, And Programming* (Rochelle Park, N.J.: Hayden Book Company, Inc., 1978).
6. Donovan, John J., *Systems Programming* (New York: McGraw-Hill Book Company, 1972).

К главе 4

1. Myers, Glenford J., *Software Reliability: Principles and Practices* (New York: Wiley-Interscience, 1976).
2. Bohl, Marilyn, *Tools for Structured Design* (Chicago: Science Research Associates, Inc., 1978).

3. Jenson, Randall W., and Charles C. Tonies, *Software Engineering* (Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1979).
4. *MCS-86 Macro Assembler Reference Manual* (Santa Clara, Calif.: Intel Corporation, 1979).
5. *The 8086 Family User's Manual* (Santa Clara, Calif.: Intel Corporation, 1979).

К главе 6

1. Gibson, Glenn A., and Yu-cheng Liu, *Microcomputers for Engineers and Scientists* (Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1980).
2. Andrews, Michael, *Programming Microprocessor Interfaces for Control and Instrumentation* (Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1982).
3. Morse, Stephen P., *The 8086 Primer: An Introduction to Its Architecture, System Design, and Programming* (Rochelle Park, N.J.: Hayden Book Company, Inc., 1980).
4. *1982 Data Components Catalog* (Santa Clara, Calif.: Intel Corporation, 1982).

К главе 7

1. Alexy, George, *Multiprogramming with the iAPX 88 and iAPX 86 Microsystems*, Application Note Ap-106 (Santa Clara, Calif.: Intel Corporation, 1980).
2. *1982 Data Component Catalog* (Santa Clara, Calif.: Intel Corporation, 1982).
3. *iRMX 86 Nucleus Reference Manual* (Santa Clara, Calif.: Intel Corporation, 1981).
4. *Introduction to the iAPX 286* (Santa Clara, Calif.: Intel Corporation, 1982).
5. Wharton, John, *Using Operating System Firmware Components to Simplify Hardware and Software Design*, Application Note Ap-130 (Santa Clara, Calif.: Intel Corporation, 1982).
6. Dijkstra, E. W., "Cooperating sequential processes," Technological University, Eindhoven, The Netherlands, 1965. (Reprinted in *Programming Languages*, F. Genuys, ed., Academic Press, Inc., New York, 1968.)
7. Shaw, Alan C., *The Logical Design of Operating Systems* (Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1974).
8. Madnick, Stuart E., and John J. Donovan, *Operating Systems* (New York: McGraw-Hill Book Company, 1974).

К главе 8

1. *1982 Data Component Catalog* (Santa Clara, Calif.: Intel Corporation, 1982).
2. *The 8086 Family User's Manual* (Santa Clara, Calif.: Intel Corporation, 1979).
3. *Intel MULTIBUS Interfacing*, Application Note AP-28A (Santa Clara, Calif.: Intel Corporation, 1979).

К главе 9

1. McNamara, John E., *Technical Aspects of Data Communications* (Bedford, Mass.: Digital Equipment Corporation, 1978).
2. *1981 Data Component Catalog* (Santa Clara, Calif.: Intel Corporation, 1981).
3. Peatman, John B., *Microcomputer-Based Design* (New York: McGraw-Hill Book Company, 1977).

4. Hall, Douglas V., *Microprocessors and Digital Systems* (New York: McGraw-Hill Book Company, 1980).

К главе 10

1. *1982 Data Components Catalog* (Santa Clara, Calif.: Intel Corporation, 1982).
2. Gibson, Glenn A., and Yu-cheng Liu, *Microcomputers for Engineers and Scientists* (Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1980).
3. Hall, Douglas V., *Microprocessors and Digital Systems* (New York: McGraw-Hill Book Company, 1980).
4. Givone, Donald D., and Robert P. Roesser, *Microprocessors/Microcomputers: An Introduction* (New York: McGraw-Hill Book Company, 1980).
5. *Intel Memory Design Handbook* (Santa Clara, Calif.: Intel Corporation, 1977).
6. Luecke, Gerald, Jack P. Mize, and William N. Carr, *Semiconductor Memory Design and Application* (New York: McGraw-Hill Book Company, 1973).

К главе 11

1. *The 8086 Family User's Manual, Numerics Supplement* (Santa Clara, Calif.: Intel Corporation, 1980).
2. *8086/8087/8088 Macro Assembly Language Reference Manual* (Santa Clara, Calif.: Intel Corporation, 1980).
3. *The 8086 Family User's Manual* (Santa Clara, Calif.: Intel Corporation, 1979).
4. *8089 Assembler User's Guide* (Santa Clara, Calif.: Intel Corporation, 1979).

К главе 12

1. *iAPX 86/30, iAPX 88/30 Operating System Processors* (Santa Clara, Calif.: Intel Corporation, 1982).
2. *iAPX 186 High Integration 16-Bit Microprocessor* (Santa Clara, Calif.: Intel Corporation, 1982).
3. *Introduction to the iAPX 286* (Santa Clara, Calif.: Intel Corporation, 1982).
4. *iAPX 286/10, High Performance Microprocessor with Memory Management and Protection* (Santa Clara, Calif.: Intel Corporation, 1982).

СПИСОК КНИГ, ПЕРЕВЕДЕННЫХ НА РУССКИЙ ЯЗЫК

Гибсон Г. А., Лю Ю-Ч. Аппаратные и программные средства микро-ЭВМ. – М.: Финансы и статистика, 1983. – 255 с.

Майерс Г. Д. Надежность программного обеспечения. М.: Мир, 1980. – 360 с.

Мэдник С., Донован Д. Операционные системы. – М.: Мир, 1978. – 792 с.

Шоу А. Логическое проектирование операционных систем. – М.: Мир, 1981.– 360 с.

СОДЕРЖАНИЕ