

Л. Левенталь

ВВЕДЕНИЕ В МИКРОПРОЦЕССОРЫ

Программное
обеспечение,
аппаратные
средства,
программирование.



Л. Левенталь

ВВЕДЕНИЕ
В МИКРОПРОЦЕССОРЫ:
Программное
обеспечение,
аппаратные
средства,
программирование

ПЕРЕВОД С АНГЛИЙСКОГО
ПОД РЕДАКЦИЕЙ В. В. СТАШИНА



МОСКВА ЭНЕРГОАТОМИЗДАТ 1983



Scan AAW

ББК 32.97

Л35

УДК 681.3—181.4.06

Рецензент В. В. Сташин

LANCE A. LEVENTHAL
INTRODUCTION TO MICROPROCESSORS: SOFTWARE,
HARDWARE, PROGRAMMING

Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632

Левенталь Л.
Л35 Введение в микропроцессоры: Программное обеспечение, аппаратные средства, программирование. Пер. с англ. — М.: Энергоатомиздат, 1983. — 464 с., ил.

В пер.: 2 р. 70 к.

Рассмотрены вопросы, касающиеся построения микропроцессорных систем на базе микропроцессоров Intel 8080 и Motorola 6800 — аналогов выпускаемых в СССР и странах — членах СЭВ микропроцессоров. Обилие примеров и упражнений выгодно отличает книгу от ранее известных отечественных и переводных книг.

Для инженерно-технических работников в области вычислительной техники, автоматики, телемеханики и студентов вузов.

Л 2405000000-580
051(01)-83 206-83

ББК 32.97
6Ф7

Лэнс Левенталь

Введение в микропроцессоры:
Программное обеспечение, аппаратные средства,
программирование

Редактор издательства А. Н. Гусяцкая

Переплет художника И. Е. Сайко

Технический редактор В. В. Хапаева

Корректор Н. А. Смирнова

ИБ № 3301

Сдано в набор 27.05.83. Подписано в печать 15.11.83.

Формат 60×90^{1/16} Бумага типографская № 1. Гарнитура литературная
Печать высокая Усл. печ. 29,0. Усл. кр.-отг. 29,0. Уч.-изд. 33,73

Тираж 30 000 экз. Заказ 1644. Цена 2 р. 70 к.

Энергоатомиздат, 113114, Москва, М-114, Шлюзовая наб., 10

Московская типография № 4 Союзполиграфпрома
при Государственном комитете СССР
по делам издательств, полиграфии и книжной торговли
129041, Москва, Б. Перелазская, 46

© 1978 by Prentice-Hall, Inc., Englewood Cliffs; N. J. 07632
© Перевод на русский язык, Энергоатомиздат, 1983

ПРЕДИСЛОВИЕ РЕДАКТОРА ПЕРЕВОДА

Микропроцессоры, обладающие большими функционально-логическими возможностями, с успехом могут быть использованы для построения широкого класса средств цифровой автоматики. Как указывалось в отчетном докладе XXVI съезду КПСС, внедрение миниатюрных электронных управляющих машин, микропроцессоров и промышленных роботов открывает безграничные возможности в машиностроении и других областях инженерной деятельности.

Микропроцессоры являются неотъемлемыми компонентами современных устройств и систем автоматики прежде всего потому, что их высокая надежность, относительно низкая стоимость, гибкость применения (модифицируемость), возможность реализации сложных алгоритмов управления достаточно простыми средствами и возможность создания систем автоматики с качественно новыми «интеллектуальными» свойствами позволяют получить исключительно высокие технико-экономические и тактико-технические характеристики систем автоматики.

Микропроцессоры широко используются в автоматической аппаратуре, которая раньше создавалась на основе логических интегральных микросхем малой и средней степеней интеграции. Преимущества применения микропроцессоров в устройствах и системах автоматики по сравнению с «жесткими» цифровыми схемами тем очевиднее, чем сложнее разрабатываемая система и чем больше ее тиражность. Все это вызывает большой интерес специалистов по автоматике, телемеханике и связи. Удовлетворению этого интереса в большей степени способствует значительное число публикаций в периодических технических журналах, а также ряд книг отечественных и зарубежных авторов, в которых рассматриваются основные принципы построения микропроцессоров и особенности их применения.

Предлагаемая читателю книга американского автора Лэнса Левенталя представляет собой учебник, ориентированный на инженеров по автоматике и телемеханике. В книге в простой и достаточно строгой форме изложены основные вопросы теории цифровых устройств, особенности архитектуры микропроцессоров, а также средства и методы проектирования цифровых устройств автоматики на основе микропроцессоров. Книгу отличает продуманная последовательность изложения материала, большое число примеров и хорошие иллюстрации. Для советского читателя особый интерес представляет то, что в книге все примеры программ и микропроцессорной реализации функций в устройствах управления объектами и процессами приводятся для двух

типов микропроцессоров Intel 8080 и Motorola 6800, имеющих аналоги в отечественной практике и в практике стран — членов СЭВ.

Гл. 1 книги является вводной и знакомит читателя с общими принципами организации микропроцессоров и с основными понятиями из области интегральной схемотехники.

В гл. 2 описывается внутренняя организация микропроцессоров Intel 8080 и Motorola 6800.

В гл. 3 описываются системы команд микропроцессоров и методы адресации команд и данных.

В гл. 4 излагаются основные сведения по языкам ассемблера микропроцессоров, а в гл. 5 рассмотрено программирование на языках ассемблера для микропроцессоров Intel 8080 и Motorola 6800.

Гл. 6 содержит описание средств системного программного обеспечения микропроцессоров и рекомендации по разработке прикладных программ пользователей.

В гл. 7 излагаются вопросы организации памяти микропроцессорных систем.

В гл. 8 рассмотрены проблемы организации обмена информацией с микропроцессором. В ней содержится описание ряда стандартных аппаратных и программных решений для реализации ввода-вывода данных в микропроцессорных системах.

В гл. 9 рассматриваются особенности работы микропроцессоров в режиме прерывания и кратко описывается метод прямого доступа к памяти микропроцессорных систем.

В данной книге автор практически все примеры реализации процедур обработки данных приводит в двух версиях: для одноаккумуляторного микропроцессора Intel 8080 и для двухаккумуляторного микропроцессора Motorola 6800. Такое изложение материала позволяет читателю оценить достоинства и недостатки каждого из указанных типов микропроцессоров на основе анализа конкретных программ. Кроме того, противопоставление этих двух типов микропроцессоров дает возможность читателю оценить все многообразие структурных решений микропроцессорных систем, варианты распределения функций между аппаратными и программными средствами, способы организации межмодульных связей, методы распределения коммуникационных ресурсов системы и т. п.

Перечисленные достоинства книги позволяют рекомендовать ее широкому кругу специалистов для изучения элементной базы микропроцессорных систем, а также средств и методов проектирования цифровых систем автоматики и телемеханики на основе микропроцессоров. Книга может быть полезна студентам соответствующих специальностей.

Предисловие и гл. 1 переведены на русский язык ст. научным сотрудником К. В. Мышаком, гл. 2 и 3 — инж. Л. А. Сташиной, гл. 4—7 — канд. техн. наук, доц. В. Г. Черновым, гл. 8 и 9 — мл. науч. сотр. Ф. Ф. Химишиным.

ПРЕДИСЛОВИЕ

Почти в любой отрасли промышленности микропроцессоры уже оказали свое влияние. Создается впечатление, что ежедневно появляются новые процессоры, новые аппаратные и программные средства для работы с ними, новые устройства, в состав которых входят микропроцессоры. Обычно уровень материала, изложенного в учебниках, отстает от уровня техники. Руководства, написанные для конкретных микропроцессоров, предназначены для пользователей и отнюдь не являются учебниками. В статьях периодической печати и научных журналов рассматриваются ограниченные вопросы или частные области применения микропроцессоров. Данная книга является общим введением в теорию микропроцессоров и предназначена для студентов старших курсов, которые уже прослушали вводный курс программирования и цифровых систем. Она могла бы быть полезной и интересной для старшекурсников и выпускников не только электротехнических и других специальностей, но и для тех, кто будет работать в области физики и естественных наук, математики, вычислительной техники и здравоохранения. Эта книга будет полезна инженерам, техникам, официальным лицам, программистам, учителям и всем, кто хочет заняться самообразованием или иметь справочный материал по микропроцессорам.

Главная трудность в изучении микропроцессоров состоит в том, что их невозможно понять, рассмотрев только аппаратные средства и программное обеспечение. Программисты обнаружат, что микропроцессоры имеют архитектуру, систему команд и программное обеспечение подобные другим ЭВМ, с которыми они уже знакомы. Инженеры найдут, что микропроцессоры имеют те же физические особенности, сигналы ввода-вывода и рабочие характеристики, что и привычные для них интегральные микросхемы. Однако и программисты, и инженеры обнаружат, что разработка систем, основанных на микропроцессорах, требует понимания как аппаратных средств, так и программного обеспечения. В этой книге я исходил из того, что при использовании микропроцессоров основной упор в разработке приходится не на аппаратные средства, а на программное обеспечение, и поэтому уделил ему больше внимания. В то же время я подробно рассматривал микропроцессоры с точки зрения аппаратных средств, особо останавливаясь на конструировании блоков памяти, использовании стандартных и специальных интегральных микросхем, организации интерфейса простых периферийных устройств. Я не рассматривал проектирование цифровых схем, так как считаю, что достаточное число книг освещает этот вопрос. Таким образом, в написанной мною книге основное вни-

вание уделено программному обеспечению, но в то же время в ней рассмотрены вопросы, связанные с аппаратными средствами.

Чтобы не рассматривать широкий круг микропроцессоров и не изобретать какой-то образец для изучения, я выбрал два наиболее широко используемых микропроцессора: Intel 8080 и Motorola 6800. Эти микропроцессоры аналогичны большинству стандартных систем. Они также в достаточной степени отличаются друг от друга как в программировании, так и в технике организации интерфейса, чтобы быть характерными для целого ряда микропроцессоров. Я полагаю, что имеется соответствующий обзор микропроцессоров и что в учебниках дается подробный анализ одного или двух устройств. Я остановил свой выбор именно на этих микропроцессорах по коммерческим соображениям, но у меня нет оснований считать, что эти два процессора (или любые другие) имеют преимущества перед своими конкурентами. В этой книге иногда упоминаются некоторые особенности различных конкурирующих процессоров.

Книга построена следующим образом. Глава 1 является введением в предмет. В ней микропроцессоры сравниваются с мини-ЭВМ и большими ЭВМ, а также с другими БИС. Здесь описываются полупроводниковая технология и типы памяти, рассматриваются преимущества и недостатки микропроцессоров, перечисляются области применения микропроцессоров и дается несколько конкретных примеров устройств, разработанных на основе этих микропроцессоров.

В гл. 2 рассматривается архитектура микропроцессоров (МП). Кратко описаны различные устройства ЭВМ: центральный процессор (ЦП), память, устройство ввода-вывода. В остальном внимание здесь сконцентрировано на ЦП. Дается описание регистров, арифметического устройства и механизма дешифрации команд. В конце главы представлены архитектуры микро-ЭВМ Intel 8080 и Motorola 6800.

В гл. 3 описывается система команд. В первых параграфах этой главы содержится общее описание форматов команд, методов адресации и групп команд. В последних параграфах описываются системы команд МП Intel 8080 и Motorola 6800.

В гл. 4 рассматриваются ассемблеры. В начале главы рассматриваются преимущества и недостатки различных языковых уровней и основные особенности ассемблеров. В конце главы дается конкретное описание стандартных ассемблеров Intel 8080 и Motorola 6800.

В гл. 5 описывается технология программирования на языке ассемблера для МП Intel 8080 и Motorola 6800. Глава начинается с простых программ. Далее в ней описываются цикл, манипуляции с символами, преобразование кодов, арифметические операции, обработка списков, таблиц и работа с подпрограммами.

В гл. 6 рассмотрен весь процесс разработки программного обеспечения. В ней описываются постановка задачи, разработка программы, кодирование, отладка программы, тестовый контроль, документирование, эксплуатация и внесение поправок. В конце главы кратко описаны системы разработки.

В гл. 7 рассмотрена память. В ней описываются основные типы связи микропроцессора с памятью. Затем рассматривается организация

интерфейса простых блоков памяти и структура шин, которые требуются для более сложных блоков памяти. Здесь же описывается конструкция блоков памяти для МП Intel 8080 и Motorola 6800 и их работа.

В гл. 8 рассматриваются устройства ввода-вывода (УВВ). Начинается она с описания процедур ввода и вывода. Затем рассматриваются простые блоки УВВ и более сложные, разбираются некоторые схемы, которые широко используются в УВВ, и простые УВВ. В конце главы описывается специфика аппаратных средств и программного обеспечения, необходимых для организации интерфейса УВВ микропроцессоров Intel 808 и Motorola 6800.

В гл. 9 рассмотрены системы прерывания. Описываются их использование, преимущества и недостатки. Затем разбираются процедуры прерываний, особенности систем прерывания и управления источниками прерываний. Далее описывается программирование и организация интерфейса систем прерывания, принятые в МП Intel 8080 и Motorola 6800. Заканчивается глава кратким описанием метода прямого доступа к памяти.

Очевидно, что многие вопросы я оставил без внимания. Я кратко рассмотрел назначение микропрограммирования без описания техники микропрограммирования (микропрограммированию микропроцессоров следует посвятить отдельную книгу). Объяснено использование языков высокого уровня, но не приведены конкретные примеры. Такие технические методы, как мультиобработка, параллельное процессирование и виртуальная память, не рассматривались совсем. Периферийные устройства микропроцессоров и разнообразные интегральные цифровые и аналоговые схемы, которые попутно используются, описаны поверхностно. О памяти с прямым доступом имеется только краткое упоминание. Возможно, некоторые из этих тем впоследствии будут рассмотрены.

Эту книгу я попытался проиллюстрировать различными примерами реального использования микропроцессоров. Я стремился к такому логическому изложению этих примеров, чтобы читатель мог применять их непосредственно при решении исследовательских или инженерных задач. При описании аппаратных средств я использовал общепринятые символы, методы конструирования, интегральные микросхемы и документацию. При описании программного обеспечения я использовал стандартную символику блок-схем, составлял и документировал все программы в соответствии с методами, которые рекомендуют изготовители микропроцессоров. Я пытался дать не только полезные примеры, но также показать образцы структур и документации, которые, я полагаю, пригодятся при квалифицированной разработке систем, основанных на микропроцессорах.

Основная трудность в работе над книгой о новой технике заключается в быстром старении материала. Очевидно, что в течение времени, которое требуется для публикации и распространения этой книги, некоторая информация устаревает. Я не пытался предсказать будущее микропроцессорной технологии, но остановил внимание на процессорах, которые широко применяются, на их конкретных применениях, на том, как важно совместное рассмотрение аппаратных средств и про-

граммного обеспечения и какую роль микропроцессоры будут играть в последующие годы. Я пытался также обрисовать основные направления, текущие задачи и новые перспективы микропроцессоров, однако понимаю, что читатели должны получать дополнительную информацию из производственных материалов, технических документов, экономических статей и научных журналов, на специальных конференциях и учебных курсах.

Очевидно, что область применения микропроцессоров стремительно расширяется. Все время появляются книги, статьи и материалы конференций на эту тему. Мы даже начали замечать, что микропроцессорная техника используется в реальных изделиях. Возможность умного и гибкого управления простыми устройствами и системами теперь достигается за умеренную цену, а это, в свою очередь, есть призыв ко всем читателям открыть эту книгу.

*Лэнс Левенталь
Солна Бич, Калифорния*

ГЛАВА ПЕРВАЯ ВВЕДЕНИЕ В МИКРОПРОЦЕССОРЫ

1.1. ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА И МИКРОПРОЦЕССОРЫ

Краткая история ЭВМ отмечена быстрыми достижениями в области технологии, а также большим разнообразием их применения. Первые ЭВМ, созданные в конце 40-х и начале 50-х годов, использовались для решения сложных научных задач. Современные ЭВМ, которые намного мощнее первых машин, кроме того, имеют бытовое применение: электронные игры, кассовые аппараты, весы, калькуляторы, предметы домашнего обихода. Каждый может купить за несколько сотен долларов ЭВМ, показанную на рис. 1.1, и использовать ее дома для разнообразных целей.

Последним достижением электронно-вычислительной техники является *микропроцессор* — устройство, которое представляет собой один или несколько крохотных кусочков кремния и выполняет все функции центрального процессора электронно-вычислительной машины. Типичный МП показан на рис. 1.2. Такое устройство может выбирать команды из памяти, дешифровать и исполнять их, производить арифметические и логические операции, получать данные из устройств ввода и посылать результаты на устройства вывода. Микропроцессор вместе с памятью и каналами ввода-вывода для связи с внешним миром является полноправной ЭВМ или *микро-ЭВМ*. Простые микро-ЭВМ стоят всего 10, а целые *системы микро-ЭВМ* с шасси, передней панелью и источником питания стоят менее 1000 дол. и имеют такие же функциональные возможности, как и огромные ЭВМ 50-х годов, стоившие в сотни раз больше. Наблюдается резкий контраст между быстрым снижением стоимости ЭВМ и ростом стоимости большинства других изделий.

Микропроцессор появился как результат тенденции изготовления ЭВМ меньших размеров, проявившейся в середине 60-х годов. В начальный период выпуска ЭВМ упор делался на крупные и мощные машины. Электронно-вычислительные машины были такими дорогими, что их могли приобретать только крупные учреждения; только специально обученный персонал мог работать на них. Новые достижения технологии, такие как транзисторы и интегральные микросхемы, обеспечили большую скорость выполнения операций, но не снизили стоимость машин. Электронно-вычислительные машины оставались недоступными и загадочными.

Тенденция, наблюдающаяся в настоящее время, началась с появления мини-ЭВМ. Первые мини-ЭВМ были примитивны по сравнению с

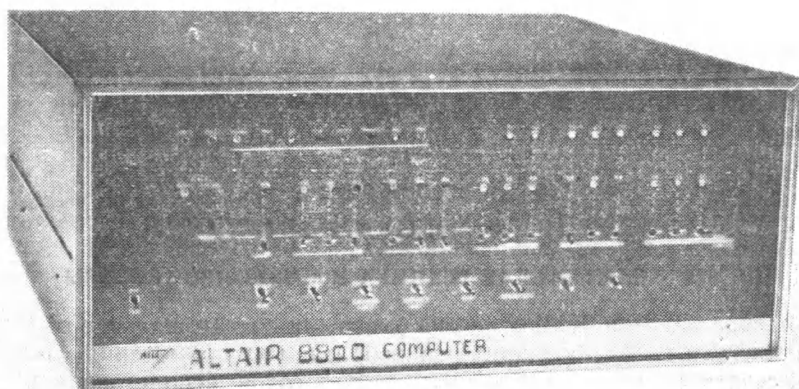


Рис. 1.1. Домашняя ЭВМ

большими ЭВМ и все еще стоили десятки тысяч долларов. Одного лаборатории, фабрики и небольшие учреждения, которые не могли позволить себе купить большие ЭВМ, теперь смогли покупать малые ЭВМ, такие как Digital Equipment PDP-8, Data General Nova, Scientific Data Systems 92 или IBM 1130. Появление более дешевых электронных схем привело к снижению стоимости ЭВМ. К 1970 г. небольшая мини-ЭВМ для работы в лаборатории, учреждении, на фабрике, складе или в кассе стоила несколько тысяч долларов.

Развитие интегральных схем привело к дальнейшему снижению стоимости мини-ЭВМ. Вскоре появилась возможность изготовления интегральной микросхемы, способной выполнять функции ЭВМ. Такие сложные схемы, получившие название *большие интегральные схемы* (БИС), можно изготавливать тысячами с издержками, ненамного большими, чем при изготовлении простых схем. Таким образом, дешевая ЭВМ, давно являвшаяся любимой мечтой писателей-фантастов, стала реальностью. В настоящее время микропроцессоров выпущено больше, чем всех других ЭВМ. К 1977 г. уже можно было купить домаш-

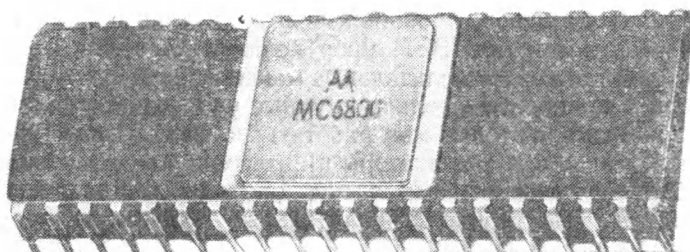


Рис. 1.2. Типичный микропроцессор

нюю ЭВМ с клавиатурой, дисплеем и кассетным запоминающим устройством менее чем за 1000 дол.

Помимо сравнения микро-ЭВМ с другими ЭВМ в этой главе описываются появление и история развития МП, даются краткое описание полупроводниковых технологий, на основе которых изготавливаются МП, общие характеристики МП и других интегральных микросхем, полупроводниковая память, используемая с МП и, наконец, некоторые области применения МП.

1.2. БОЛЬШИЕ И МАЛЫЕ ЭВМ

Появление дешевых ЭВМ открыло новые возможности для их применения. Эта тенденция, начавшаяся с появлением мини-ЭВМ, получила развитие вследствие применения микропроцессоров. На рис. 1.3 приводятся стоимости микро-, мини-ЭВМ и больших ЭВМ, а также их обычные количества, которые приобретает пользователь.

Большие ЭВМ общего назначения, такие как IBM 370, Univac 1100 или Burroughs 6700, выполняют две основные функции:

1) решение комплексных научных и инженерных задач, таких как управление космическим кораблем, прогноз погоды или проектирование различных систем;

2) обработка данных в больших масштабах (обработка отчетности для банков, страховых компаний, складских хозяйств, сферы коммунальных услуг и для государственных учреждений).

Для решения этих задач требуется исключительно большое число операций или пересылок данных. Решение типичных научных проблем требует решения сложных уравнений, которые нельзя осуществлять вручную. Решение хозяйственных задач требует обработки большого объема отчетов и работы с большим числом УВВ.

В этих областях мини- и микро-ЭВМ не заменяют больших ЭВМ, однако смогут решать аналогичные задачи при менее сложных вычислениях и с меньшими объемами информации. Так, на мини- или микро-ЭВМ можно производить лабораторные вычисления или обрабатывать отчетный материал для небольших фирм. Однако наиболее широкое применение мини- и микро-ЭВМ находят в областях за пределами обычного применения больших ЭВМ.

Обычное применение мини- и микро-ЭВМ характерно тем, что ЭВМ:

1) является компонентом системы. Система, которая может быть лабораторной установкой, станком или банковским терминалом,

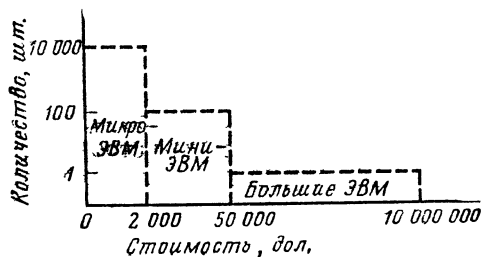


Рис. 1.3. Стоимость и количество ЭВМ, поступающих на рынок

имеет в своем составе малую ЭВМ. Ее даже может быть и не видно снаружи;

2) решает специальную задачу в конкретной системе. Машину не используют для решения разнообразных задач подобно большой ЭВМ, а она является частью определенной установки, такой как медицинский прибор, линотип или заводская установка;

3) имеет постоянную программу, которая редко меняется. В отличие от большой ЭВМ, которая может решать разнообразные хозяйственные и инженерные задачи, большинство малых ЭВМ выполняет только одну задачу, например такую, как управление системой охранной сигнализации, воспроизведение информации на дисплее, раскрой и гибка металлических листов. Программы при этом часто хранятся в в постоянном запоминающем устройстве (ПЗУ);

4) выполняет задачи в реальном масштабе времени. Примером такого применения могут служить станки, на которых в нужное время требуется сменить резец для получения правильного профиля, или система управления ракетой, где ЭВМ должна в нужные моменты включать двигатели для получения требуемой траектории;

5) выполняет скорее задачи управления, чем арифметические действия или обработку данных. Ее основной задачей может быть управление складом, транспортом или наблюдение за состоянием больного.

Использование мини- и микро-ЭВМ отличается от использования больших ЭВМ. Пользователь большой вычислительной системы пишет программы на удобном языке (например, ФОРТРАН, БЕЙСИК или ПЛ/1), передает их персоналу вычислительного центра и получает результаты в виде распечаток, на магнитной ленте или другом носителе информации. В распоряжение пользователя может предоставляться разнообразное периферийное оборудование ВЦ, стандартные программы и прочее, и ему нет необходимости знать, как функционирует ЭВМ или как она связана с памятью и устройствами ввода-вывода.

Те, кто захочет использовать мини- и микро-ЭВМ в качестве компонентов системы, обнаружат, что в этом случае положение совершенно другое. У мини- и микро-ЭВМ редко бывают программное обеспечение и периферийные устройства, требующиеся для конкретного применения. Разработка программ для малых ЭВМ — это утомительная работа, требующая много времени. Более того, острой проблемой является разработка связи между малой ЭВМ и всей системой. Чтобы эффективно использовать малую ЭВМ, проектировщик должен детально понимать, как она работает.

Таким образом, для применения малых ЭВМ требуются взаимосвязанные аппаратные средства и программное обеспечение. Разработчик должен распределить задачи между аппаратными средствами и программным обеспечением, учитывая быстродействие и стоимость, и составить программы для получения информации с устройств ввода и пересылки результатов на устройства вывода. Особо важное значение при разработке как аппаратных средств, так и программного обеспечения имеет фактор времени.

Малая ЭВМ, которая входит в готовое изделие в качестве компонента, обычно не может быть использована для разработки программ. Про-

граммы для больших ЭВМ разрабатываются на тех же ЭВМ, в которых они затем используются. Однако малые ЭВМ, являющиеся частью прибора или станка, имеют память, периферийные устройства и программное обеспечение, достаточные только для выполнения функций управления; наличие дополнительных устройств только увеличит их стоимость и не улучшит работу. В результате, у таких ЭВМ нет устройств считывания с перфокарт или телетайпов ввода-вывода информации, компиляторов или программ-отладчиков для упрощения разработки прикладных программ, памяти на магнитных лентах или дисках для хранения информации. Электронно-вычислительная машина, встраиваемая в систему, может реализовать только функции управления системой, но не может быть использована на этапе разработки.

На стадии разработки используется специальное оборудование, называемое *системой разработки*. Это оборудование может включать в себя ту же ЭВМ или другие ЭВМ (*симулятор* или *кросс-систему*). У типичной системы разработки имеются периферийные устройства для ввода программ и данных и для вывода результатов, довольно большая память для хранения программ пользователя и программ системы, программное обеспечение и аппаратные средства для программирования и обнаружения ошибок и внешняя память большой емкости. Такие системы разработки включают в себя программное обеспечение, аппаратные средства, периферийные устройства и интерфейсы, которые не имеют никакого отношения к конечному изделию; они только облегчают разработку.

Задача разработки программ, даже если они в конечном итоге предназначены для мини- или микро-ЭВМ, больше подходит для больших машин. Например, программа трансляции с языка ФОРТРАН на машинный язык может быть выполнена на большой ЭВМ быстрее, чем на малой, и может использовать периферийные устройства, являющиеся составной частью большой ЭВМ. Трансляция программы не обязательно должна осуществляться на ЭВМ, для которой она предназначена.

Пользователь мини- или микро-ЭВМ обнаруживает, что разработка системы довольно трудоемка. Программы часто разрабатываются на оборудовании, отличном от того, на котором они в итоге будут работать. Часто аппаратные средства и интерфейсы разрабатываются одновременно с программами, и пользователь сталкивается со сложной задачей «увязки» системы (комплексирования аппаратных и программных средств системы).

1.3. СРАВНЕНИЕ ТИПИЧНЫХ ЭВМ

В табл. 1.1 приведены сравнительные данные большой ЭВМ общего назначения, мини-ЭВМ общего назначения, встраиваемой мини- и микро-ЭВМ. Большие ЭВМ представлены моделью ЭВМ IBM 370/168 (рис. 1.4), которая широко используется для обработки информации, большие мини-ЭВМ — моделью PDP 11/45 фирмы Digital Equipment (рис. 1.5). Из встраиваемых мини-ЭВМ представлена часто используемая модель NAKED MINI фирмы Computer Automation (рис. 1.6), а из микро-ЭВМ — Intel MCS-80, основанная на широко применяемом МП

Т а б л и ц а 1.1. Сравнительные данные различных типов ЭВМ

Характеристика	Тип ЭВМ			
	IBM 370/168	DEC PDP 11/45	NAKED MINI	Intel MCS-80
Стоимость, дол.	$4,5 \cdot 10^6$	$50 \cdot 10^3$	2500	250
Разрядность, бит	32	16	16	8
Объем памяти, 8-разрядных байт	$8,4 \cdot 10^6$	256 К	64 К	64 К
Время суммирования, мкс	0,13	0,9	3,2	2,0
Максимальная скорость ввода-вывода, байт/с	$16 \cdot 10^6$	$4 \cdot 10^6$	$1,4 \cdot 10^6$	$0,5 \cdot 10^6$
Число регистров общего назначения	64	16	3	7
Периферийные устройства (представленные изготовителем)	Все типы	Широкое разнообразие	Диск, лента, перфокарта, телегайт, дисплей, кассетный накопитель	Устройство считывания с ленты, гибкий диск, программатор, ППЗУ
Программное обеспечение	Все типы	Широкое разнообразие	Операционная система, ассемблер, ФОРТРАН, БЕЙСИК	Ассемблер, монитор, ПЛ/М, редактор

Intel 8080 (рис. 1.7). Другие изготовители производят аналогичные машины различных уровней; данный сравнительный анализ, конечно, не предполагает, что описываемые ЭВМ по качеству выше своих конкурентов.

Стоимость

Большая ЭВМ общего назначения настолько дорога, что может использоваться только в качестве центральной вычислительной системы крупного учреждения. Для такой ЭВМ требуется штат специально

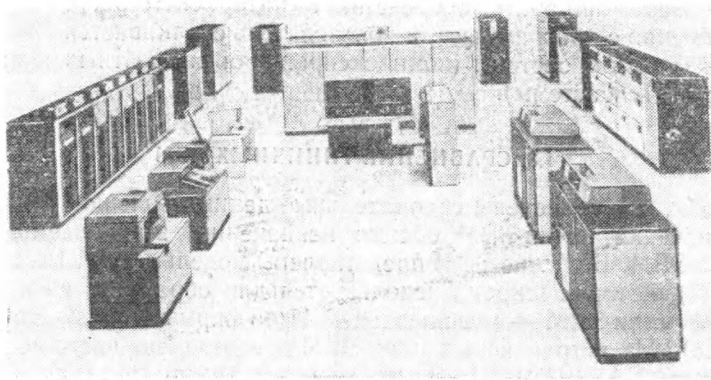


Рис. 1.4. ЭВМ IBM 370/168

Рис. 1.5. ЭВМ PDP 11/45 фирмы Digital Equipment

подготовленных программистов, инженеров и операторов. У нее большое число периферийных устройств, таких как УВВ, системы памяти на дисках и лентах, а также терминалы. На ЭВМ можно обрабатывать большие объемы данных и решать множество различных задач.

Большая мини-ЭВМ слишком дорога, чтобы быть составной частью готового изделия, но может использоваться лабораторией, небольшими организациями или промышленными предприятиями. Такие мини-ЭВМ могут также выполнять функции субпроцессоров для больших ЭВМ.

Встраиваемая мини-ЭВМ достаточно недорога, чтобы быть частью промышленной установки, банковского терминала или испытательной системы. Однако для того, чтобы стоимость ЭВМ была оправдана, необходимо, чтобы цена всего изделия превышала

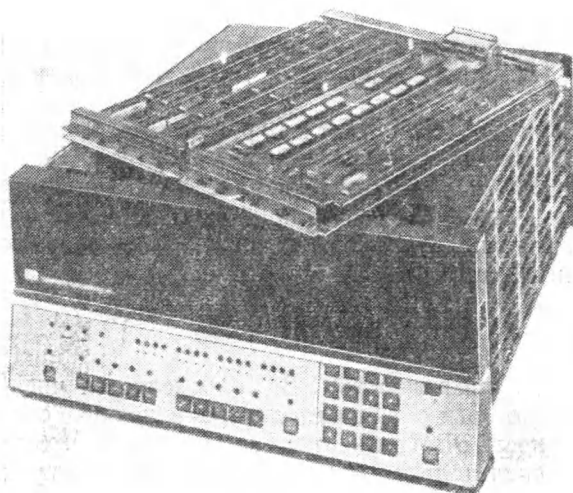
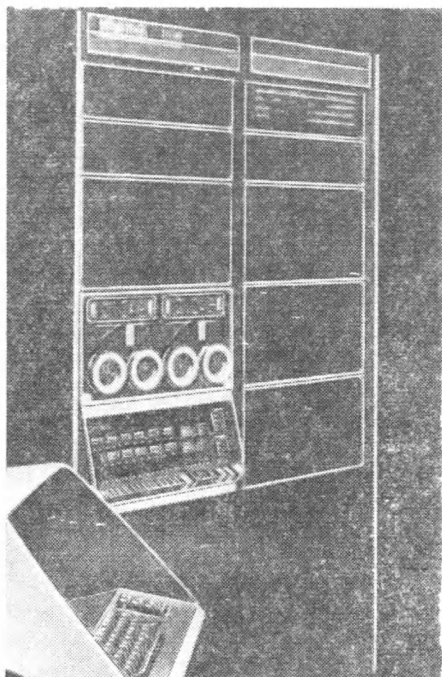


Рис. 1.6. Встраиваемая ЭВМ NAKED MINI фирмы Computer Automation

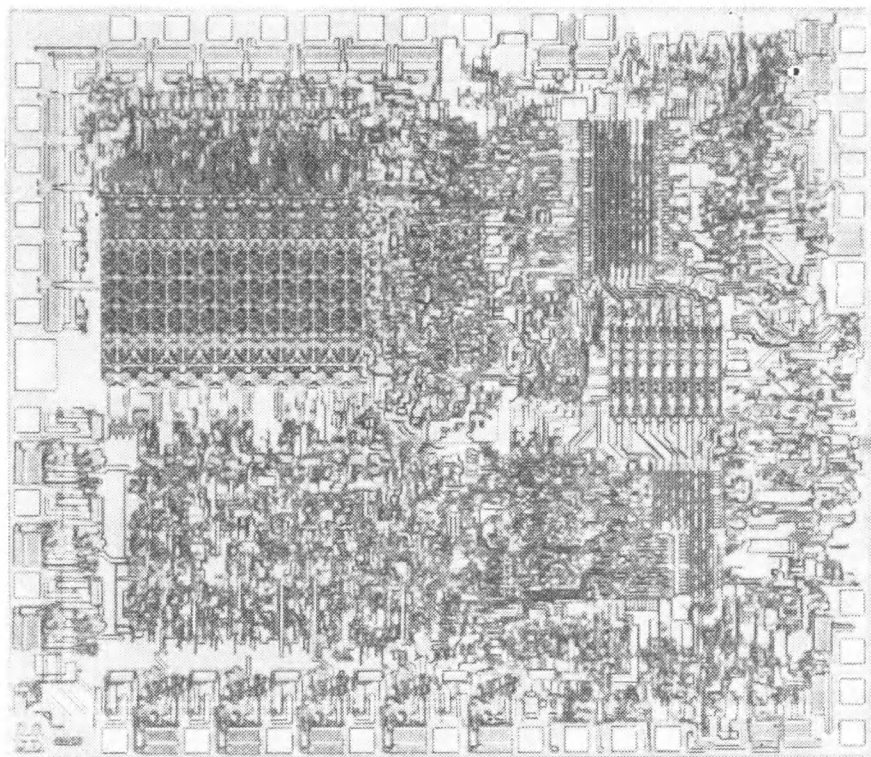


Рис. 1.7. Микрофотография микропроцессора Intel 8080

10 тыс. дол. Изделия с такой ценой обычно не выпускаются большой серией, и заказ, составляющий сотни мини-ЭВМ, можно считать очень крупным.

Кроме того, микро-ЭВМ стоит в 10 раз дешевле мини-ЭВМ. Поэтому она может быть частью системы, стоящей 1000 дол. Возможны применения микро-ЭВМ в автоматических кассовых аппаратах, дисплеях, регистраторах и измерительных приборах. Очевидно, что производитель таких изделий будет использовать большое число МП (10 000 изделий крупной серией).

Формат слова

Формат слова (разрядность) ЭВМ определяется числом двоичных знаков (*битов*), которые она может одновременно обработать (табл. 1.3). Производительность ЭВМ зависит от ее разрядности и быстродействия. Электронно-вычислительная машина, имеющая формат слова в 2 раза больший, чем другая ЭВМ, действующая с такой же скоростью, может за определенный отрезок времени выполнить в 2 раза большую работу.

Таблица 1.2

Тип ЭВМ	Стоимость, дол
IBM 370/168	4,5 · 10 ⁶
DEC PDP 11/45	50 000
NAKED MINI	2500
Intel MCS-80	250

Таблица 1.3

Тип ЭВМ	Формат, бит
IBM 370/168	32
DEC PDP 11/45	16
NAKED MINI	16
Intel MCS-80	8

У большой ЭВМ формат слова в 2 раза больше, чем у мини-ЭВМ, и в 4 раза больше, чем у микро-ЭВМ. Обычно у больших ЭВМ формат слова 32—64 бит, у мини-ЭВМ 12—32 бит, а у микро-ЭВМ 4—16 бит. Таким образом, формат слова является важным показателем вычислительной мощности ЭВМ.

Это особенно важно при выполнении арифметических операций.

Представьте себе, как трудно было бы перемножить два шестизначных числа, если бы оперировали только парой цифр; при таком выполнении умножения задача была бы сложнее не в 6 раз, а в большее число раз из-за необходимости позиционирования и множества переносов. Электронно-вычислительные машины с большим форматом слова могут выполнять сложные арифметические расчеты намного лучше, чем ЭВМ малой разрядности. Такие задачи, как прогноз погоды и имитация полета самолета, больше подходят для больших ЭВМ. Мини- и микро-ЭВМ больше приспособлены для задач управления.

Необходимо отметить, однако, что большой формат слова ЭВМ имеет преимущества, только когда входная информация в двоичной системе имеет большую длину. Если ЭВМ получает за 1 прием только 8-разрядную информацию, то тот факт, что она может обработать слова большей разрядности, преимущества не дает. Для задач управления часто требуется, чтобы ЭВМ в каждый момент времени оперировала данными небольшой разрядности. Эти данные поступают на ЭВМ со шкал, кнопок, конечных выключателей или сенсоров и после обработки передаются на дисплей, приводы и двигатели. В таких применениях преимуществами обладают ЭВМ с меньшим форматом слова. Электронно-вычислительная машина с меньшим форматом слова может успешно подготавливать данные для ввода в большую ЭВМ. Мини- или микро-ЭВМ, используемые в качестве терминалов, могут значительно уве-

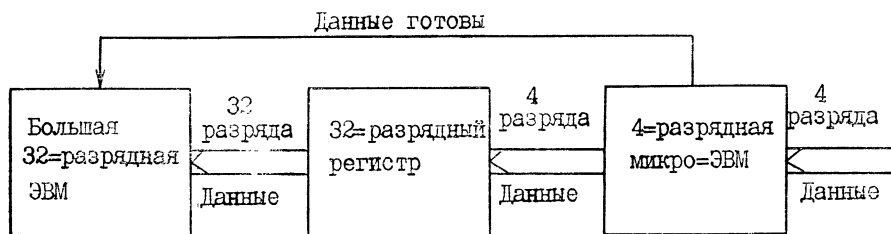


Рис. 1.8. Микропредпроцессор

личивать загрузку большой ЭВМ при очень небольших дополнительных затратах. На рис. 1.8 показана система, к которой 4-разрядный МП получает входные данные в виде 4-разрядных сегментов и размещает их в соответствующих местах 32-разрядного регистра. Когда МП набирает 32 разряда, он посылает сигнал в большую ЭВМ, которая за прием считывает все 32 разряда. Пока МП собирает данные, большая ЭВМ выполняет другие задачи.

Емкость памяти

Емкость памяти (в байтах) определяет длину программ и объем информации, с которыми ЭВМ может оперировать. Если программа или информация требует большего объема памяти, чем тот, которым располагает ЭВМ, то могут быть использованы дополнительные устройства памяти на магнитных дисках или лентах. Эти дополнительные системы памяти дороги и имеют намного меньшее быстродействие, чем основная память. Поэтому ЭВМ с ограниченной емкостью памяти потребуются большее время для выполнения больших программ, чем машинам, имеющим большую емкость памяти; использование дополнительной памяти аналогично пользованию справочником или таблицей — всегда, когда пользуешься справочником, требуется больше времени.

Большая ЭВМ может оперировать с огромными программами и большими объемами информации, не обращаясь к дополнительной памяти (табл. 1.4). Такая емкость памяти необходима при сложных вычислениях, обработке больших чисел и высокой точности расчетов. Большая мини-ЭВМ имеет значительно меньший объем памяти, чем большая ЭВМ, но все-таки намного больший, чем малые ЭВМ. Большие ЭВМ имеют достаточную память для больших операционных систем и разнообразных алгоритмических языков. Для таких программ могут потребоваться сотни тысяч байт памяти.

Мини- и микро-ЭВМ имеют значительно меньшую емкость памяти. Поэтому операционные системы, компиляторы и другое программное обеспечение, предназначенное для этих ЭВМ, должно занимать меньший объем памяти или использовать дополнительную память. Мини- и микро-ЭВМ обычно работают с короткими программами и небольшими объемами данных. Для микро-ЭВМ прикладная программа длиной 16 тыс. байт уже считается очень большой. Существуют определенные сложности адресации памяти в ЭВМ с малым форматом слова. В табл. 1.5 показаны объемы памяти, к которым можно обращаться в зависимости от разрядности адреса. Отметим, что 8-разрядный адрес позволяет работать только с 256-байтной памятью; немногие

Таблица 1.4

Тип ЭВМ	Емкость памяти, байт	Тип ЭВМ	Емкость памяти, байт
IBM 370/168	$8,4 \cdot 10^6$	NAKED MINI	$64 \cdot 10^3$
DEC PDP 11/45	$256 \cdot 10^3$	Intel MCS-80	$64 \cdot 10^3$

Таблица 1.5. Соотношение разрядности адреса и объема памяти

Разрядность адреса, бит	Объем памяти, байт	Разрядность адреса, бит	Объем памяти, байт	Разрядность адреса, бит	Объем памяти, байт
8	256	11	2 К	14	16 К
9	512	12	4 К	15	32 К
10	1 К	13	8 К	16	64 К

прикладные программы могут уложиться в такой малый объем. Однако 8-разрядная микро-ЭВМ может оперировать с более длинными адресами по слогам. Цикл команды 8-разрядных МП довольно сложный, так как процессор должен выбирать из памяти программ адреса побайтно и формировать их в ЦП. Если МП 4-разрядный, то операции с адресами будут еще сложнее. Следовательно, мини-ЭВМ имеют большую производительность, чем микро-ЭВМ, так как большая длина слова позволяет им более эффективно адресовать память.

Время выполнения команды

Производительность ЭВМ зависит от скорости выполнения команд (табл. 1.6). Скорость выполнения команд есть величина переменная и зависящая от сложности команды. Время выполнения операции суммирования является широко используемым показателем работы. Отметим, что объем работы в цикле выполнения команды зависит от формата слова ЭВМ.

Большая ЭВМ действует в 7 раз быстрее, чем большая мини-ЭВМ. Меньшие ЭВМ действуют намного медленнее. При простом сравнении получается, что большая ЭВМ в 60 раз производительнее микро-ЭВМ, так как она оперирует в 4 раза большим числом данных за отрезок времени, в 15 раз меньший.

Сравнение ЭВМ по производительности должно учитывать и два других фактора:

- 1) большее время выполнения команды в МП из-за меньшего формата слова и, следовательно, замедленной адресации;
- 2) более развитые системы команд больших ЭВМ.

Время суммирования — это время, требуемое для сложения содержимого двух регистров; обращение ЦП к памяти при этом не производится. Очевидно, что ЦП должен сначала получить операнды из памяти. Для выборки операндов требуется сформировать адреса, которые в МП формируются не самым лучшим способом из-за ограниченной разрядности шины данных.

Сложность и число команд ЭВМ также оказывают влияние на возможности ЭВМ. Одной ЭВМ может потребоваться много циклов команд для выполнения работы, которую на другой ЭВМ можно выполнить за один цикл. Например, операция вычитания на ЭВМ, имеющих и не имеющих команду вычитания, осуществляется следующим образом:

функция: $Z = X - Y$

ЭВМ I (с вычитанием):

$$Z = X - Y$$

Таблица 1.6

Тип ЭВМ	Время суммирования процессором, мкс
IBM 370/168	0,13
DEC PDP 11/45	0,9
NAKED MINI	3,2
Intel MCS-80	2,0

Таблица 1.7. Зависимость числа возможных команд от разрядности операции

Число бит для кодирования операций	Число возможных команд
3	8
4	16
5	32
6	64
7	128
8	256

ЭВМ II (без вычитания):

$$W = -Y;$$

$$Z = X + W.$$

Электронно-вычислительная машина II использует два цикла команд вместо одного. Команды деления и умножения, например, чаще имеются в больших ЭВМ, чем в малых. Отсутствие таких команд снижает производительность малых ЭВМ.

Число команд и среднее время их выполнения связаны с форматом слова ЭВМ. Число команд, которые может иметь ЭВМ, определяется числом бит, используемых для кодирования операций, как показано в табл. 1.7. Электронно-вычислительная машина со словом большого формата может использовать 7 или 8 бит для кодирования операций и оставшиеся биты отводить для адресации данных. Электронно-вычислительная машина со словом меньшего формата должна использовать дополнительные слова в памяти программ, чтобы иметь достаточно широкий набор команд, что приводит к необходимости многократного обращения к памяти и большему времени выполнения команд.

Максимальная скорость ввода-вывода данных

Максимальная скорость ввода-вывода данных (табл. 1.8) ограничивает число задач, которые может решать ЭВМ и типы периферийных устройств, которые она может эффективно использовать. Большая ЭВМ может передавать данные со значительно большей скоростью, чем малая, и использовать высокоскоростные дисковые системы и другие устройства, передающие миллионы бит в секунду. По этому критерию большая ЭВМ мощнее микро-ЭВМ более чем в 30 раз. При этом большие ЭВМ имеют обычно более содержательные команды и аппаратные средства для операций ввода-вывода и контроллеры, которые запускаются несколькими командами и могут затем передавать большие объемы данных без дальнейшего участия процессора. Малая ЭВМ часто должна передаваться по одному слову данных за 1 прием. Очевидно, что такая ЭВМ не может работать с высокоскоростными устройствами ввода-вывода.

Таблица 1.8

Тип ЭВМ	Максимальная скорость ввода-вывода, байт/с
IBM 370/168	$16 \cdot 10^6$
DEC PDP 11/45	$4 \cdot 10^6$
NAKED MINI	$1,4 \cdot 10^6$
Intel MCS-80	$500 \cdot 10^3$

Таблица 1.9

Тип ЭВМ	Число регистров общего назначения
IBM 370/168	64
DEC PDP 11/45	16
NAKED MINI	3
Intel MCS-80	7

Длина слова ЭВМ определяет и число УВВ. Каждое УВВ, как и ячейка памяти, должно иметь свой адрес. Электронно-вычислительная машина с большим форматом слова может адресовать большее число УВВ, чем ЭВМ с меньшим форматом слова. Из-за меньшей скорости передачи данных и формата слова малые мини- и микро-ЭВМ обычно используют несколько простых периферийных устройств, таких как панели управления, дисплеи с цифровой клавиатурой, телетайпы и устройства считывания с перфоленты. Высокоскоростные периферийные устройства, такие как магнитные ленты или диски, высокоскоростные печатающие устройства и линии связи чаще используются в больших ЭВМ.

Микро-ЭВМ отлично работают в системах взаимодействия с человеком (таких как электронные кассовые аппараты или видеоигры), так как время реакции человека составляет около 0,1 с. Другой областью применения микро-ЭВМ является управление медленно изменяющимися параметрами, такими как температура, давление или концентрация химического вещества. Области, где необходимо обеспечить высокие скорости изменения параметров и реакцию системы в диапазоне микросекунд, лучше оставить для больших ЭВМ или специализированных управляющих устройств.

Число регистров общего назначения

Регистры образуют небольшой блок памяти внутри ЦП и имеют такое же отношение к основной памяти, как основная память к памяти на дисках или магнитной ленте. Если у ЭВМ имеется несколько регистров общего назначения, они могут хранить часто используемые данные и промежуточные результаты вычислений. Чем больше регистров общего назначения, тем меньше непроизводительных затрат времени на перемещение данных между памятью и ЦП.

У больших ЭВМ имеется намного больше регистров общего назначения (табл. 1.9), чем у малых ЭВМ, и поэтому они могут быстрее оперировать данными. У них также могут быть дополнительные регистры, которые используются подпрограммами. Меньшие ЭВМ должны чаще обращаться к памяти и тратить больше времени на загрузку и хранение содержимого регистров. Подпрограммы вынужденно используют те же регистры, что и основная программа, и поэтому содержимое этих регистров должно направляться на хранение в память, а при возврате в основную память восстанавливаться.

Периферийные устройства

Периферийные устройства (табл. 1.10), которыми укомплектована ЭВМ, могут оказывать значительное влияние как на разработку прикладных программ, так и на интерфейс системы. Если имеется возможность получить разнообразные периферийные устройства, спроектированные специально для данной ЭВМ, то можно быстрее разработать программы и потратить меньше времени на проектирование интерфейса системы.

Разработка намного упрощается при наличии высокоскоростных УВВ и памяти большой емкости. Если таких периферийных устройств нет, то пользователь может действовать одним из двух способов:

а) разработать требуемые интерфейсы и б) использовать медленные и менее удобные периферийные устройства. В первом случае пользователю необходимо выполнить разработку интерфейсов между данной ЭВМ и высокоскоростными УВВ, во втором пользователь будет терять много времени на ввод и вывод информации. Отсутствие быстродействующего устройства ввода означает, что потребуются часы для того, чтобы ввести относительно длинную программу с телетайпа. Отсутствие памяти большой емкости приводит к необходимости частого повторения медленного ввода. Из-за отсутствия быстродействующих средств вывода сложной проблемой становится документирование программы или результатов вычислений.

Большие ЭВМ комплектуются разнообразными УВВ. Пользователь может купить быстродействующие периферийные устройства для целей разработки или отдельное периферийное устройство для конечного изделия. Для малых ЭВМ имеется очень немного периферийных устройств, подходящих по интерфейсу.

Стоимость периферийных устройств не снизилась в той же мере, как стоимость ЭВМ, так как в них используются дорогостоящие механические узлы и блоки. Хотя в настоящее время имеются периферийные устройства более высокого качества и по более низким ценам, чем в прошлом, их стоимость относительно стоимости ЭВМ возросла. Устройство считывания с перфоленты с перфоратором стоит столько же, сколько малая мини-ЭВМ и значительно больше, чем микро-ЭВМ.

Программное обеспечение

В области программного обеспечения (ПО) положение аналогично положению с периферийными устройствами (табл. 1.11) за исключением того, что стоимость ПО постоянно растет. Наличие или отсутствие ПО оказывает влияние как на процесс разработки, так и на объем требуемого нового программирования. Системное ПО упрощает задачу разработки программ пользователя. Поставляемое ПО может быть в состоянии выполнить некоторые или все задачи разрабатываемой системы.

Для больших ЭВМ имеется намного больший объем ПО, чем для малых. На IBM 370 может быть использован практически любой язык или даже программа, написанная для других систем. Для больших

Таблица 1.10

Тип ЭВМ	Имеющиеся периферийные устройства
IBM 370/168 DEC PDP 11/45 NAKED MINI	Все типы Широкое разнообразие Диск, лента, перфокарта, теле- таип, дисплей, кассетный нако- питель
Intel MCS-80	Перфолента, гиб- кий диск

Таблица 1.11

Тип ЭВМ	Программное обеспечение
IBM 370/168 DEC PDP 11/45 NAKED MINI	Все типы Широкое разно- образие Операционная си- стема, ассемблер, ФОРТРАН, БЕЙСИК
Intel MCS-80	Ассемблер, мони- тор, ПЛ/М, редак- тор

мини-ЭВМ имеется значительно меньший объем ПО. Изготовитель мини-ЭВМ и независимые фирмы поставляют пользователю некоторые операционные системы, компиляторы для большинства широко распространенных языков программирования и другие программы. Для больших ЭВМ имеются большие библиотеки прикладных программ, в состав которых входят разнообразные программы в диапазоне от программ обычных математических функций до таких высоко специализированных программ, как бухгалтерские системы для отдельных видов деятельности или программ решения отдельных групп инженерных задач. Наличие компиляторов для языков программирования означает, что обширный резерв программ, записанных на языках ФОРТРАН, БЕЙСИК, ПЛ/М, АПЛ, может быть прямо использован на больших ЭВМ.

При использовании малых ЭВМ обнаруживаем, что имеется намного меньший объем системного и прикладного ПО. Все, чего можно ожидать, это простая операционная система или монитор, ассемблер и несколько простых компиляторов или интерпретаторов. Иногда даже для этого ПО требуется память и периферийные устройства, не входящие в поставляемую систему минимальной конфигурации. Прикладное ПО малых ЭВМ довольно бедное. Пользователь должен разрабатывать с нуля большую часть прикладных программ.

Микро-ЭВМ, как правило, имеют еще меньший объем ПО, чем мини-ЭВМ, а именно: несколько операционных систем или компиляторов. Пользователь микро-ЭВМ редко может применить в своей системе разработанную ранее программу.

Программирование для МП усложняется тем, что МП используются для решения задач, отличных от задач больших ЭВМ. Большинство языков программирования предназначено для решения научных задач и обработки экономической информации, а не для решения задач управления объектами. Для эффективной разработки программного обеспечения МП требуются новые языковые средства программирования.

1.4. ИСТОРИЯ РАЗВИТИЯ МИКРОПРОЦЕССОРОВ

Хотя МП выполняют такие же функции, что и ЦП больших ЭВМ, они с физической точки зрения представляют собой полупроводниковые устройства, подобные кристаллам калькуляторов, электронных часов или полупроводниковой памяти. Появление микропроцессоров является результатом тех же технологических достижений, которые привели к появлению электронных часов и калькуляторов. Многие особенности МП определяют тем, что они являются продуктом полупроводникового производства.

Первые интегральные микросхемы (ИС) появились в начале 60-х годов. *Интегральная микросхема* — это комбинация элементов электронной схемы на единой подложке или кристалле кремния и конструктивно оформленная в виде отдельного блока. К середине 60-х годов появление новых технологических процессов позволило значительно увеличить число элементов схемы, которые могут быть размещены на кристалле. Наиболее широко используется технология *металл—окисел—полупроводник* (МОП). Простые интегральные микросхемы получили название *малых интегральных микросхем* (МИС), более сложные — *средних интегральных микросхем* (СИС) и еще более сложные — *больших интегральных микросхем* (БИС). На рис. 1.9, например, показано, как со временем росла емкость оперативной памяти на одном кристалле. За 10 лет, с 1966 по 1976 г., емкость таких устройств возросла в 64 раза, в то время как их стоимость за бит хранимой информации снизилась в 100 раз.

Первым применением МОП-технологии явились БИС памяти. Благодаря малым размерам и потребляемой мощности БИС памяти стали идеальным средством для построения блоков памяти в системах связи, военном снаряжении и ЭВМ.

В конце 60-х годов важными областями применения БИС, изготовленных на основе МОП-технологии, стали калькуляторы и терминалы. Как калькуляторы, так и терминалы выполняют простые задачи, для решения которых не требуется высокое быстродействие, большие объемы памяти или сложные операции.

Калькуляторы

Только по прошествии нескольких лет после разработки электронный калькулятор стал общепризнанным прибором. Его функции (рис. 1.10) просты: калькулятор получает от небольшой клавиатуры данные по одной десятичной цифре за цикл, выполняет требуемые арифметические действия и высвечивает результаты на линейном светодиодном цифровом дисплее. Программы калькулятора хранятся в ПЗУ; данные, которые вводит пользователь, запоминаются в ОЗУ.

Кристалл ЦП калькулятора должен обладать следующими свойствами:

- 1) иметь простой интерфейс с клавиатурой и светодиодным дисплеем;
- 2) обрабатывать десятичные цифры. Так как десятичные цифры кодируются четырьмя двоичными разрядами, то ЦП должен оперировать одновременно с одной (или более) 4-разрядной группой бит;
- 3) выполнять фиксированные программы, хранимые в ПЗУ, и запоминать данные в ОЗУ;
- 4) иметь возможность расширения числа реализуемых функций с тем, чтобы можно было относительно просто добавлять такие функции, как вычисление процентов, квадратных корней и тригонометрических функций;
- 5) обладать гибкостью в применении с тем, чтобы калькулятор можно было успешно применять (без существенных изменений в ЦП) как для инженерных и деловых расчетов, так и для программирования;
- 6) центральный процессор должен иметь малую стоимость, размеры и потребляемую мощность, для того чтобы калькулятор был портативным и недорогим.

Большая интегральная микросхема ЦП калькулятора может иметь малое быстродействие и небольшую по емкости память, так как с калькулятором работает человек, имеющий медленную реакцию, и в памяти необходимо хранить только несколько чисел. Калькуляторные БИС, изготовленные по МОП-технологии, недорогие, небольшие, потребляющие малую мощность, способные оперировать с десятичными цифрами и выполнять стандартные вычисления, были спе-

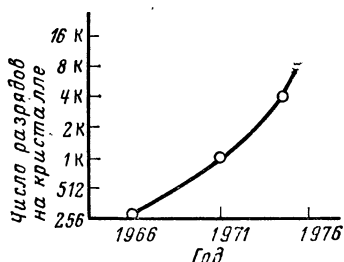
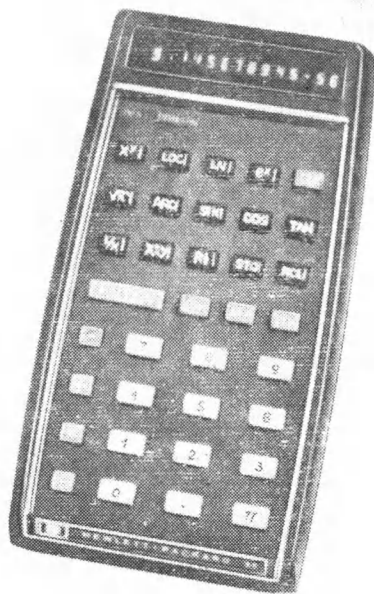


Рис. 1.9. Рост емкости памяти на одном кристалле по годам

Рис. 1.10. Калькулятор 35 фирмы Hewlett-Packard



циально спроектированы для обеспечения ввода с клавиатуры и вывода на светодиодный цифровой дисплей. По этой технологии изготовлены миллионы калькуляторов по несколько долларов за штуку.

Однако калькуляторные БИС не являются гибкими в применении. Изготовитель калькуляторов не может просто добавить новые функции по просьбе пользователя или подсоединить к калькулятору дополнительные УВВ, такие как печатающее устройство или графопостроитель. Калькуляторная БИС идеальна для простых вычислителей, но для более сложных аппаратов требуется более гибкое устройство.

Первый микропроцессор, Intel 4004, был разработан как БИС для калькулятора, который бы обладал некоторой гибкостью в применениях. Он отличается от предыдущих калькуляторных БИС тем, что пользователь может его запрограммировать для выполнения дополнительных функций и для работы с различными УВВ. Однако микропроцессор Intel 4004 сохранил основные свойства БИС калькулятора, т. е. он:

- 1) оперирует с 4 битами (одной десятичной цифрой);
- 2) имеет команды считывания с клавиатуры и выполнения арифметических операций над десятичными числами;
- 3) работает только по программам, хранящимся в ПЗУ;
- 4) может использоваться в простых недорогих системах, в состав которых входят дисплей и клавиатура.

Хотя БИС Intel 4004 очень похожа на БИС калькулятора, это уже ЭВМ, так как изготовитель системы может запрограммировать ее на выполнение новых функций. Одна и та же БИС может иметь множество областей применения.

Терминалы

Еще одной ведущей отраслью применения микропроцессорных БИС явилось производство терминалов. Терминалы являются средством общения с удаленными ЭВМ или средством отправки и получения сообщений по телеграфу или другим линиям связи. Они представляют собой широкую гамму устройств от простейших телетайпов до сложных устройств типа дисплея с дистанционным управлением, показанного на рис. 1.11. Данные вводятся с клавиатуры или пишущей машинки, обрабатываются посимвольно, индицируются на экране дисплея или фиксируются на печатающем устройстве и одновременно передаются в соответствующем формате в ЭВМ или на другой терминал. В ПЗУ хранятся рабочие программы и таблицы, обрабатывающие оче-

редность ввода символов на экран дисплея или печатающее устройство. Вводимые пользователем данные должны храниться в ОЗУ до тех пор, пока пользователь не даст команду передать их в ЭВМ или разместить их в памяти на резервное хранение.

Какой тип ЦП требуется для терминала? Наибольший интерес представляют следующие характеристики ЦП:

1. Возможность обеспечения простого интерфейса с разнообразными УВВ, включая клавиатуру, печатающие устройства, дисплей, линии связи, диски, каскеты и другие терминалы.

2. Способность обрабатывать символы. Так как символы обычно состоят из 8 бит, устройство должно быть в состоянии оперировать одновременно с одной 8-разрядной группой данных или более.

3. Способность выполнять стандартные программы, хранимые в ПЗУ, и программы пользователя, хранимые на кассетах, магнитных или перфокартах.

4. Достаточно большая емкость памяти, необходимая для того, чтобы система могла оперировать с разнообразными программами, а также хранить и проверять целую страницу данных до их передачи.

5. Способность обрабатывать текстовую информацию в пределах строки или строки текста. В этом случае пользователь имеет возможность вызвать строку текста из памяти и отредактировать ее.

6. Быстрый доступ к выбранным ячейкам памяти через указатели, содержащие адреса данных, а не сами данные. Пользователь может обеспечить доступ к определенному символу, поместив его адрес в указатель.

7. Расширяемость системы по функциональным возможностям. При этом изготовитель ЦП может добавить в систему новые свойства, такие как более широкие возможности редактирования текста, разнообразные интерфейсы для связи, проверка на ошибки, расширение памяти, вывод графической информации и т. п.

8. Гибкость применения. Терминал мог бы использоваться не только как средство ввода-вывода информации в ЭВМ, но и как управляющая ЭВМ.

9. Малые стоимость, размеры и потребляемая мощность.

Терминал должен иметь быстродействие, достаточное для того, чтобы успевать за человеком-оператором. Для терминалов требуется микропроцессорная БИС не такая, как для калькуляторов. Микропроцессор Intel 8008 был первым, разработанным для терминалов; его основные характеристики следующие:

1) операции с байтами;
2) наличие команд логических операций и команд сдвига, которые удачно используются для обработки символьной информации значительного объема памяти (16 К вместо 4 К в Intel 4004);

3) программы могут быть размещены как в ПЗУ, так и в ОЗУ;

4) возможность взаимодействия со сложными периферийными устройствами, такими как печатающие устройства или дисплей;

5) наличие внутреннего указателя памяти, который может использоваться для выбора и перемещения строки информации в процессе редактирования текста;



Рис. 1.11. Терминал Brilliant Bee фирмы Beehive

6) возможность перемещения и оперирования с блоками данных.

Хотя Intel 8008 специально разработан для применения в терминалах, ему в большей степени, чем Intel 4004, присущи признаки типичной ЭВМ. Набор команд, методы адресации и регистры общего назначения такие же, как и у мини-ЭВМ. Intel 8008 действует немного медленнее, чем мини-ЭВМ, но в то же время предоставляет пользователю такие вычислительные возможности, которые позволяют применять его не только в терминалах, но и в системах сбора данных, устройствах сигнализации, испытательном оборудовании, станках и навигационных системах.

1.5. ПОЛУПРОВОДНИКОВЫЕ ТЕХНОЛОГИИ

Микропроцессоры, являясь сложными полупроводниковыми приборами, имеют характеристики, во многом аналогичные характеристикам других устройств, изготовленных на основе той или иной интегральной технологии. Например, МОП-микропроцессоры имеют быстродействие, электрические характеристики, компоновку, потребляемую мощность и другие физические параметры, аналогичные параметрам устройств памяти МОП, сдвигающих регистров и калькуляторов БИС. Имеются МП, изготовленные на основе других интегральных технологий. Характеристики и возможности применения МП во многом зависят от особенностей технологии их производства.

Какими параметрами характеризуется технология? Некоторые из них приведены в табл. 1.12.

В табл. 1.13 приводятся сравнительные данные для различных полупроводниковых технологий, используемые в настоящее время при производстве МП.

Микропроцессор, изготовленный на основе *p*-МОП технологии, медленно действующий, но дешевый, имеющий высокую степень интеграции на кристалле, не сопрягаемый с ТТЛ и имеющий низкую нагрузочную способность по выходам (на выходах МП должны использоваться усилители мощности).

Микропроцессор, изготовленный на основе *n*-МОП технологии, более быстродействующий, но все еще уступающий по своим временным характеристикам стандартным схемам ТТЛ: его можно сделать сопрягаемым с ТТЛ, но он имеет низкую нагрузочную способность по выходам.

Микропроцессор, изготовленный на основе К-МОП технологии (комплементарной МОП-технологии), обладает умеренным быстродействием, невысокой степенью интеграции, повышенной стоимостью, но имеет такие достоинства, как простота сопряжения с ТТЛ, низкая потребляемая мощность и высокая помехоустойчивость.

Микропроцессор, изготовленный на основе биполярной технологии (ТТЛ-Шоттки), имеет более высокое быстродействие, чем МП, изготовленный по МОП-технологии, или стандартные схемы ТТЛ, но он дорог и имеет низкую степень интеграции.

Микропроцессор, изготовленный на основе ТЛЭС-технологии (транзисторная логика с эмиттерными связями), имеет высокое быстродействие, но очень дорог, потребляет большую мощность и сложно сопрягается с интегральными микросхемами, изготовленными по другой технологии.

Интегральная инжекционная логика (ИИЛ) — это новая полупроводниковая технология, которая, возможно, в скором времени позволит производить МП,

Таблица 1.12. Параметры полупроводниковой технологии

Быстродействие
Плотность упаковки (степень интеграции)
Стоимость
Потребляемая мощность
Помехоустойчивость
Безотказность
Сопрягаемость со схемами транзисторно-транзисторной логики
Опыт и традиции производства

Т а б л и ц а 1.13. Сравнение различных полупроводниковых технологий

Технология	Быстродействие (1=максимальному быстродействию)	Потребляемая мощность (1=наименьшая мощность)	Плотность (1=наименьшей степени интеграции)	Безотказность (1=наиболее высокая надежность)	Стоимость (1=самой минимальной стоимости)	Опыт производства (1=максимальному времени использования)	Сопрягаемость с ТТЛ
<i>p</i> -МОП	6	4	2	5	1	2	Нет
<i>n</i> -МОП	5	3	1	4	2	3	Иногда
К-МОП	3	1	3	1	3	4	Да
ТТЛ-Шоттки	2	5	5	2	3	1	»
ТЛЭС	1	6	6	6	6	5	Нет
ИИЛ (И ² Л)	3	2	3	3	3	6	—

Совмещающие в себе достоинства всех технологий (высокую плотность и низкую стоимость *n*-МОП-приборов, высокое быстродействие ТТЛ-схем и высокую помехоустойчивость К-МОП-приборов).

Выбор того или иного типа МП для конкретного применения часто осуществляется на основе данных о технологии, по которой он изготовлен. В табл. 1.14 приводятся некоторые возможные требования к разрабатываемой системе и указана технология, которая этим требованиям отвечает лучше всего.

Технологии *p*- и *n*-МОП позволяют получать МП на одном кристалле; такие процессоры являются самыми дешевыми и малогабаритными, они более всех остальных обеспечены аппаратными средствами и имеют развитое программное обеспечение. Технологии *p*- и *n*-МОП обеспечивают получение самой большой емкости памяти на одном кристалле (ПЗУ емкостью 64 Кбит и ОЗУ емкостью 16 Кбит). Однако микропроцессоры *p*- и *n*-МОП относительно медленно действующие, их трудно сопрягать со стандартными ТТЛ-схемами; они не имеют большого разнообразия и номенклатуры дополнительных схем (изготовленных по этой же технологии), которые можно было бы использовать для создания комплектных систем¹. Серьезным недостатком МП, изготовленных по *p*- и *n*-МОП-

Т а б л и ц а 1.14. Критерии выбора типа МП

Требования к системе	Наиболее подходящая технология
Низкая стоимость	<i>p</i> -МОП, <i>n</i> -МОП
Малые габариты	<i>p</i> -МОП, <i>n</i> -МОП
Высокое быстродействие	ТЛ-Шоттки, ТЛЭС
Малая потребляемая мощность	К-МОП
Высокая устойчивость по отношению к неблагоприятной окружающей среде	К-МОП
Сопрягаемость с: ТТЛ	ТТЛ-Шоттки, К-МОП
К-МОП	ТТЛ-Шоттки, К-МОП
ТЛЭС	ТЛЭС
Доступность	<i>p</i> -МОП, <i>n</i> -МОП
Наличие стандартных БИС, изготовленных по той же технологии	ТТЛ-Шоттки, К-МОП, ТЛЭС
Наличие БИС памяти большой емкости, изготовленной по той же технологии	<i>p</i> -МОП, <i>n</i> -МОП, ТТЛ-Шоттки
Наличие развитого ПО	<i>p</i> -МОП, <i>n</i> -МОП

¹ Сказанное не относится к МП Intel 8080 и Motorola 6800 которые хотя и изготовлены на основе *n*-МОП-технологии, тем не менее по своему интерфейсу прямо сопрягаемы с ТТЛ-схемами и, кроме того, имеют широкую номенклатуру дополнительных схем, позволяющих создавать законченные системы. (Прим ред.)

технологии, является низкое значение тока на выходе и неспособность работать на другие БИС без промежуточных усилителей мощности.

Микропроцессоры, изготовленные на основе К-МОП-технологии, прежде всего находят применение там, где помехи и агрессивная окружающая среда являются важным фактором (военное оборудование, транспорт), или в таких областях, как спутники и портативные системы связи, где очень важна малая потребляемая мощность. Хотя К-МОП-технология не обеспечивает получения такой высокой сложности схемы на кристалле или низкой стоимости, как *n*- или *p*-МОП-технология, она имеет большее быстродействие, может быть сопряжена со стандартной ТТЛ и имеет большое семейство вспомогательных БИС. Имеется только несколько микропроцессоров К-МОП и БИС с памятью средней емкости; однако последние достижения К-МОП-технологии (усложнение схемы на кристалле и повышение быстродействия) сделали ее очень привлекательной.

Технология ТТЛ-Шоттки (ТТЛ-Ш) используется главным образом в микрокристаллических МП из-за большой мощности потребления и малой сложности кристалла. Обычно эти процессоры копируют существующие мини-ЭВМ или используются как быстродействующие контроллеры в системах связи и средствах обработки сигналов. Технология ТТЛ-Ш позволяет получать БИС с меньшей плотностью упаковки и более дорогие, чем *p*- и *n*-МОП-схемы. Они потребляют больше энергии, а также более чувствительны к помехам чем К-МОП-схемы; главными преимуществами схем ТТЛ-Ш является высокое быстродействие и сопрягаемость с огромным семейством стандартных устройств ТТЛ. Имеются также напоминающие устройства довольно большой емкости, изготовленные по технологии ТТЛ-Ш.

Микропроцессоры, изготовленные на основе ТЛЭС-технологии, работают еще быстрее и потребляют большую мощность, чем схемы ТТЛ-Ш. С микропроцессорами, изготовленными на основе ТЛЭС-технологии, очень сложно соединить ТТЛ-схемы; они требуют специальной компоновки схем на печатных платах и межплатных соединений из-за большой мощности рассеяния. Микропроцессоры, изготовленные по технологии ТЛЭС, лучше всего подходят для применений в больших и быстродействующих ЭВМ, в быстродействующих системах связи и точных приборах. Имеется ограниченное число различных БИС и ЗУ малой емкости, изготовленных по этой же технологии.

При использовании технологии ИИЛ можно получить МП с высокой сложностью кристалла и низкой стоимостью, сравнимой со стоимостью БИС, изготовленных на основе *n*- и *p*-МОП-технологий, с малой мощностью потребления К-МОП-схем и быстродействием стандартной ТТЛ.

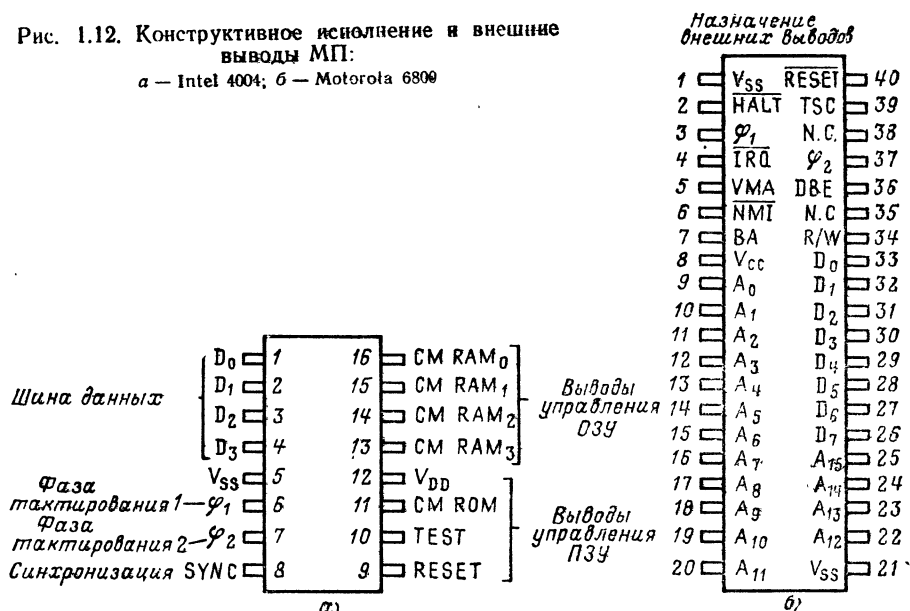
1.6. КОНСТРУКТИВНЫЕ ХАРАКТЕРИСТИКИ ИНТЕГРАЛЬНЫХ МИКРОСХЕМ

Многие характеристики МП такие же, как у других цифровых интегральных микросхем. Как и в других ИС, размер и сложность МП определяются максимальными размерами кристаллов, изготовление которых может обеспечить современная технология производства. Ограничения, связанные с размерами кристалла и корпуса, лимитируют число связей между устройством и внешним миром. Функциональные характеристики МП аналогичны характеристикам других ИС; типичными из них являются уровни сигналов на входе и выходе, потребляемая мощность, частота синхронизации, а также устойчивость к изменению питающих напряжений или условий окружающей среды.

Ограничения, связанные с размерами кристалла, означают, что большинство МП имеют простую структуру и малую длину слова. По мере усложнения схемы процессора усложняются его разработка, компоновка и изготовление. Для изготовления кристаллов, имеющих размеры, близкие к максимальным (в настоящее время около 6 мм²) требуются специальные производственные условия; даже в этом случае только один кристалл из 100 изготовленных работает правильно. Поэтому с приближением размеров кристаллов к максимальным стоимость МП сильно возрастает. Такие процессоры, как Motorola 6800, Intel 8080, Fairchild F-8 и Signetics 2650, изготовлены на кристалле размером 5×5 мм². Разработчики сохраняют малый размер кристалла процессора путем ограничения числа и

Рис. 1.12. Конструктивное исполнение и внешние выводы МП:

а — Intel 4004; б — Motorola 6800



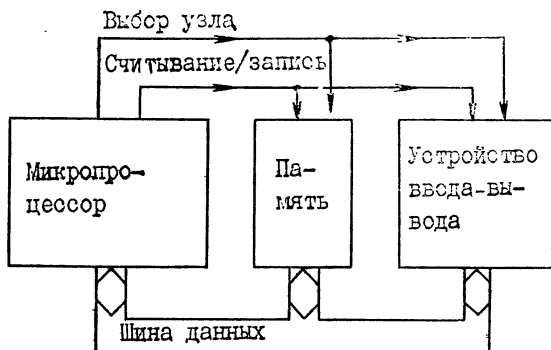
разрядности внутренних регистров и шин, числа соединений, числа и сложности системы команд и т. д. Тип и размеры корпуса МП оказывают влияние как на степень сложности монтажа МП, так и на его стоимость. Кристаллы БИС являются монолитными структурами — подсоединение можно осуществлять только через предусмотренные внешние выводы. К содержимому внутренних регистров, признаков или шин нельзя иметь непосредственный доступ, если не предусмотрены соответствующие выводы. Внутреннее состояние МП также нельзя изменить непосредственно, если нет соответствующих выводов. Эта проблема не возникает в больших ЭВМ, использующих дискретные схемы; там за внутренним состоянием ЦП можно следить с помощью осциллографа и изменять его путем подачи соответствующих сигналов. Для управления кристаллом БИС требуется сложная система вводов; для определения содержимого внутренних регистров требуются специальные программы с помощью которых содержимое регистров выводится из МП.



Рис. 1.13. Двух-
правленная шина
данных:

а — считывание дан-
ных; б — запись дан-
ных

Рис. 1.14. Использование общей шины данных для памяти и УВВ



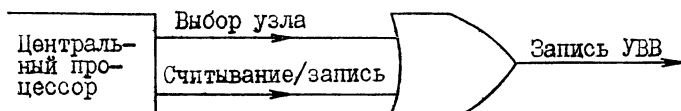
Большое число выводов БИС упрощает разработку системы, так как можно иметь большое число входных и выходных сигналов. Однако корпуса с большим числом внешних выводов имеют большую стоимость, занимают большую площадь на печатной плате и требуют много соединений. Более простые (и дешевые) МП имеют корпуса с 16—28 выводами. Пример такого МП дан на рис. 1.12, а. Стандартные МП, такие, как Motorola 6800, показанный на рис. 1.12, б, имеют корпуса с 40 внешними выводами.

Ограниченное число внешних выводов приводит к тому, что процессор должен использовать некоторые выводы для нескольких целей, например, для передачи данных к памяти и от памяти, как показано на рис. 1.13. Как система узнает, в каком направлении передавать данные? Для этого процессор выдает сигнал управления (READ/WRITE — «считывание/запись»), который имеет высокий уровень, когда процессор считывает сигнал из памяти, и низкий уровень, когда он записывает данные в память.

Процессор может использовать те же внешние выводы для передачи данных к устройствам и от устройств ввода-вывода, как показано на рис. 1.14. В этом случае процессору требуется другой сигнал управления (SECTION SELECT — «выбор устройства») для того, чтобы различить операции передачи для памяти и для УВВ. Этот сигнал может иметь высокий уровень для передачи в память и из памяти и низкий — для передачи в УВВ и из УВВ. Внешние схемы позволяют получать комбинации управляющих сигналов процессора. Например, схема ИЛИ (рис. 1.15) может выдать сигнал низкого уровня ЗАПИСЬ УВВ только тогда, когда процессор обращается к устройству вывода информации.

Кроме того, ограничение числа внешних выводов приводит к тому, что некоторые входы и выводы должны соединяться с различного вида внешними устройствами, например клавиатурой, кассетными накопителями или устройствами считывания с ленты. В некоторый период времени только одно устройство может быть подключенным к внешним выводам ЦП, а остальные устройства не должны ему мешать. Система дешифрирования команд выбирает определенное устройство, которое должно быть подключено к выводам ЦП. Чтобы другие устройства не мешали обмену с выбранным, используется специальная шина, наиболее часто — шина, построенная на основе схем с тремя состояниями (*трисабильная шина*). Незадействованные устройства в этом случае отключаются от шины.

Логические схемы могут быть также использованы для объединения входных сигналов. Сигнал RESET (гашение) может, например, войти или от удаленного концевого выключателя или с панели управления. На рис. 1.16 показано



Сигнал ЗАПИСЬ УВВ имеет низкий уровень, только тогда, когда сигналы ВЫБОР УЗЛА и СЧИТЫВАНИЕ/ЗАПИСЬ имеют низкий уровень.

Рис. 1.15. Внешняя схема для сигналов управления

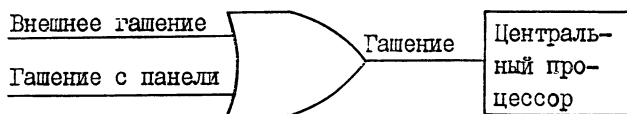


Рис. 1.16. Внешняя схема для сигналов управления

подсоединение этих источников сигнала гашения к вводу RESET через схему ИЛИ. Внешние логические схемы позволяют увеличивать число сигналов управления, поступающих от процессора, и уменьшать число сигналов управления, поступающих в процессор.

Кроме того, временное мультиплексирование внешних выводов приводит к тому, что выходные данные процессор формирует на шине данных только на очень короткое время и что процессор может не сразу вызвать данные для ввода. Внешние буферные регистры (Latches) должны сохранять данные до тех пор, пока их можно будет передавать в ЦП или из ЦП должным образом.

Важной характеристикой БИС являются уровни сигналов на входе и выходе. Уровни сигналов на входах должны быть таковы, чтобы схема идентифицировала их правильно с гарантией: понятие уровня выходного сигнала относится к уровню напряжения, которое схема гарантированно выдает. Эти уровни зависят от температуры окружающей среды и нагрузки схемы.

Чтобы две БИС (такие как МП и память) работали вместе, они должны правильно идентифицировать сигналы, генерируемые друг другом. Разработчик должен обратить внимание на защищенность системы от помех. Если требования помехоустойчивости не будут удовлетворены, то будут возникать ошибки и система будет вести себя неустойчиво. Правильность приема входных сигналов обеспечивается буферными схемами или преобразователями уровня. Однако такие устройства увеличивают стоимость системы и ее сложность, а также увеличивают время передачи сигналов. Преобразователи уровня могут понадобиться в системах, содержащих как ТТЛ, так и МОП-устройства, так как для МОП-устройства требуется более высокий уровень напряжения, чем для ТТЛ.

Кристаллы БИС (в частности, устройства, изготовленные по МОП-технологии) могут иметь различные энергетические и временные характеристики. Наиболее удобной ситуацией является такая, когда уровни напряжения питания и временные характеристики для схем всей системы будут одинаковыми. При применении МП и БИС памяти, изготовленных по ТТЛ-Шоттки- или К-МОП-технологии, так обычно и бывает. Однако более простые *n*-МОП- и *p*-МОП-системы требуют нескольких источников питания нестандартного уровня (т. е. уровня, отличного от напряжения 5 В) и сложных схем для временного согласования сигналов.

1.7. ПОЛУПРОВОДНИКОВАЯ ПАМЯТЬ

В большинстве микро-ЭВМ используется полупроводниковая память. Большие интегральные микросхемы памяти обеспечивают хранение программ, таблиц, символов и данных.

С МП обычно используются три типа БИС полупроводниковой памяти: постоянная, постоянное запоминающее устройство (ПЗУ), программируемая, программируемое постоянное запоминающее устройство (ППЗУ) и память для считывания и записи данных (обычно называемая оперативным запоминающим устройством — ОЗУ). Почти все системы, основанные на МП, используют все типы памяти на стадии разработки или изготовления. Большинство полупроводниковых технологий позволяет получать все три типа БИС памяти, однако в настоящее время БИС памяти, изготовленные на основе *n*-МОП-технологии, являются самыми дешевыми и используются наиболее широко.

Большие интегральные микросхемы постоянной памяти — наиболее простой и дешевый тип памяти. В технологическом процессе изготовления БИС содержимое памяти заносится в соответствии с шаблоном (или маской), который разрабатывает изготовитель БИС. Изготовитель требует определенную цену, *цену маски*, за каждый шаблон. Таким образом, замена ПЗУ или изготовление ПЗУ в не-

больших количествах обходится дорого. Постоянные запоминающие устройства используются для постоянных таблиц и для окончательных вариантов отлаженных программ в многотиражных изделиях. Стандартные ПЗУ могут закупаться без цены маски при использовании в системах воспроизведения символов на печатающих устройствах и дисплеях, а также для преобразования кодов. Постоянные запоминающие устройства *энергонезависимы*, т. е. их содержимое не меняется при отключении питающих напряжений.

Программируемые ПЗУ — это постоянные запоминающие устройства, содержимое которых не может быть изменено при нормальных условиях работы, но может быть запрограммировано при особых условиях. Изготовитель БИС производит ППЗУ, у которых все ячейки матрицы памяти находятся в одном и том же состоянии. Пользователь может изменить содержимое любой ячейки на противоположное путем подачи импульсов высокого напряжения в течение определенного периода времени. Имеется специальное оборудование, называемое устройством программирования ППЗУ (*программатор* ППЗУ), которое выдает необходимые импульсы для адресуемых пользователем ячеек.

Программируемое ПЗУ можно программировать только 1 раз. Одним из вариантов ППЗУ является стираемое ППЗУ или СППЗУ. Содержимое СППЗУ может быть стерто путем помещения БИС под источник ультрафиолетового света примерно на 10 мин после извлечения его из печатной платы. После этого ППЗУ можно снова программировать. Отметим, что если требуется заменить хотя бы 1 бит в СППЗУ, то необходимо стереть его содержимое полностью. Другой разновидностью ППЗУ является электроизменяемое ПЗУ или ЭИПЗУ (ЕАРОМ). У этого типа БИС постоянной памяти можно изменить содержимое одной ячейки, не снимая его с печатной платы. Как и ПЗУ, ППЗУ являются устройствами постоянного хранения. Они энергонезависимы и их содержимое не может быть изменено с помощью программы.

Большие интегральные микросхемы оперативной памяти (ОЗУ) являются наиболее сложными и дорогими видами памяти, но их содержимое менять легче всего. Полупроводниковые ОЗУ *энергозависимы*, т. е. они теряют содержимое при отключении напряжения питания. Микро-ЭВМ могут использовать ОЗУ для хранения программ на начальных этапах разработки программ или когда необходимы частые смены программ. В других случаях ОЗУ используется как память данных.

1.8. ОБЛАСТИ ПРИМЕНЕНИЯ МП

В настоящее время имеется множество изделий, использующих МП. Кроме общего описания областей применения МП, в этом параграфе даются также конкретные примеры по каждой области

Испытательное оборудование и приборы

Микропроцессоры используются в счетчиках, испытательном оборудовании, вычислительных осциллографах, цифровых вольтметрах, автоматических емкостных мостовых схемах, рентгеновских анализаторах, анализаторах крови, дальномерах, синтезаторах частот, системах сбора данных и в приборах спектрального анализа. Примером такого использования является комплексный генератор сигналов Fluke 6010A, приведенный на рис. 1.17, в котором применен МП Intel 4004 для программирования частот испытания и управления их последовательностью, для интерфейса с внешними испытательными системами, для выбора диапазонов (масштабирования) и для управления дисплеями. Микропроцессор является достаточно быстродействующим, чтобы самому генерировать сигналы для испытаний; вместо этого он действует как контроллер, подключающий по программе источники высокочастотных сигналов, генерируя сигналы управления, масштабируя данные от первичных преобразователей, управляя яркостью дисплея и осуществляя хранение вводимых значений так, чтобы к ним был обеспечен легкий доступ.

Для простых приборов и испытательных устройств МП предоставляет возможность программирования испытаний, может обеспечивать интерфейс, упрощать ввод данных, выдавать данные, предупредительные сигналы или команды

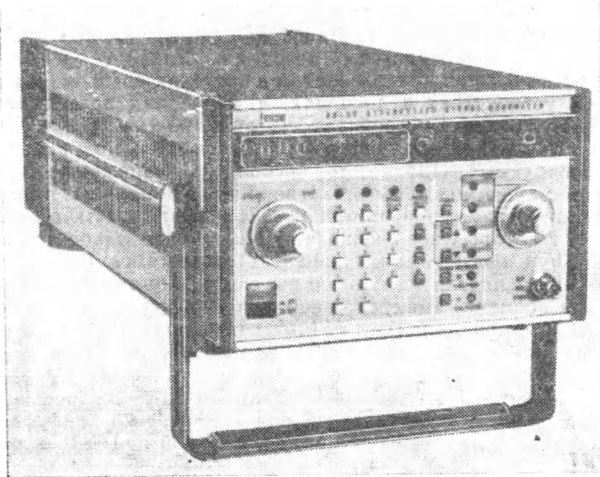


Рис. 1.17. Генератор сигналов 6010А фирмы Fluke

на дисплей в удобной форме, а также автоматически масштабировать данные параметры. Микропроцессор может также обеспечить самоиспытание и самокальбровку, проверку согласованности данных, связь с ЭВМ или приборами, управляемыми ЭВМ, и автоматическое усреднение показаний. В простых применениях МП заменяет логические схемы. Он более надежен, обеспечивает большую гибкость приборов и облегчает их использование. Вместе с тем приборы на основе МП обычно действуют медленнее, более дороги и требуют больших затрат на разработку.

В больших приборных системах и испытательном оборудовании МП может полностью или частично заменить мини-ЭВМ. Микропроцессор дешевле и меньше по размерам, его легче защитить от окружающих неблагоприятных условий, он является более надежным и потребляет меньшую мощность, чем мини-ЭВМ. Кроме того, МП действует медленнее, может оперировать с меньшим числом УВВ и имеет меньший объем стандартного программного обеспечения, номенклатуру периферийных устройств и возможности интерфейса, чем мини-ЭВМ.

Связь

Микропроцессоры используются в терминалах, сетях мини-ЭВМ, блоках коммутации сообщений, ретрансляторах, системах накопления и передачи данных, кодирующих и дешифрирующих устройствах, портативных системах связи и модемах. Типичным примером такого применения является телеконтроллер фирмы Action Communications Systems (рис. 1.18) — коммутирующая установка, используемая в системе индивидуальных терминалов. В телеконтроллере в качестве процессоров ввода-вывода для мини-ЭВМ используется МП (National IMP16); микропроцессоры преобразуют сообщения во внутренний код мини-ЭВМ, формируют последовательности сигналов управления, проверяют на предмет ошибок, редактируют текст сообщения и присваивают заголовок. В результате предварительной обработки информации в МП возможность загрузки мини-ЭВМ увеличивается на порядок.

В области связи МП могут выполнять некоторые рутинные функции больших ЭВМ, выполнять преобразования кодов или простые вычисления, обнаруживать сигналы, сопрягать линии, работающие с различными скоростями передачи или с различными протоколами и редактировать текст сообщения. Кроме того, в области контрольно-измерительной аппаратуры они могут обеспечивать программируемость, управлять дисплеями и сообщениями для оператора, связь с

клавиатурой или другими устройствами ввода и позволяют быстро освоить и совместить разнообразное оборудование. С помощью МП можно увеличить надежность и сократить стоимость систем связи.

Наиболее серьезным ограничением МП в области связи является быстродействие. МОП-процессоры не могут оперировать с данными со скоростью более 50 Кбит/с. Процессоры, реализованные на основе ТТЛ- и ТЛЭС-технологий, могут применяться в области связи.

Электронно-вычислительные машины

Микропроцессоры используются в развлекательных и деловых ЭВМ, в ЦП мини-ЭВМ и контроллерах ввода-вывода, а также во многих периферийных устройствах больших ЭВМ, включая интеллектуальные терминалы, устройства считывания с магнитных карт, графопостроители, перфораторы, оптические устройства считывания, кассетную память и гибкие диски. Типичным примером является терминал Pertec Model 7100 CRT (рис. 1.19), в котором применяется МП 8003 фирмы Intel для редактирования текста, исправления ошибок и подготовки данных для пересылки. Функции, требующие высокого быстродействия (такие как очистка экрана), выполняются аппаратными средствами, но инициируются микропроцессором. Микропроцессор предоставляет дополнительные возможности редактирования текста, позволяет изменять функциональное назначение клавиш клавиатуры, изменять наборы символов и использовать различные протоколы связи.

Микропроцессоры, используемые в периферийных устройствах ЭВМ, имеют такие же преимущества, как и МП, используемые в контрольно-измерительной аппаратуре. Что еще более важно, МП позволяют распределить вычислительные возможности системы. «Интеллект» на местах, в терминалах и других периферийных устройствах приводит к тому, что многие задачи могут выполняться на периферии, а не центральным процессором. Обработка данных не только снижает затраты на связь и увеличивает возможность загрузки ЦП, но также упрощает программное обеспечение и позволяет выполнять некоторую полезную работу.

Рис. 1.18. Телеконтроллер фирмы Action Communications Systems

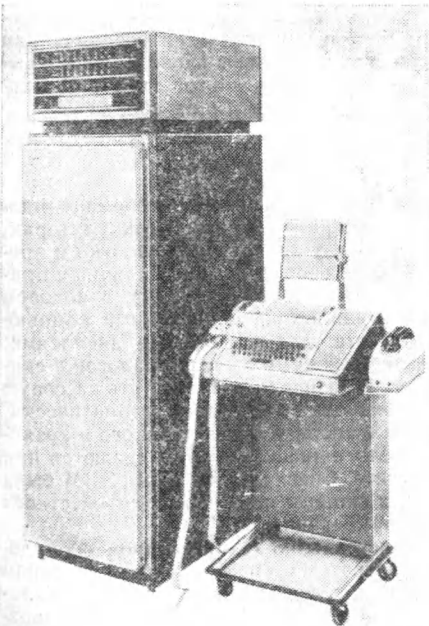


Рис. 1.19. Дисплей 7100 CRT фирмы Pertec Business Systems

даже в том случае, если откажет связь или нарушится работа ЦП. С теоретической точки зрения ЭВМ, состоящая полностью из микропроцессоров, в состоянии распределить ресурсы по определенным задачам и продолжать работу даже в том случае, если несколько элементов системы выйдет из строя. Такие распределенные системы находятся сейчас в стадии разработки.

Промышленность

Микропроцессоры используются в системах контроля данных, установках контроля качества, автоматических взвешивающих и дозирующих системах контроля узлов/машин, определения степени скручиваемости, контроллерах станков, растяжных и гибочных прессах, устройствах сортировки пиломатериалов, погрузочно-разгрузочных устройствах, фото- и кино-проявочных машинах, цифровых газовых счетчиках, оптических читающих устройствах, графических и чертежных системах, промышленных терминалах и автоматических тестерах. Типичным примером является автоматический тестер ALMA 720 (рис. 1.20) для проверки логических схем, в котором используется МП Motorola 6800 для генерирования испытательных программ, проверяющих основные параметры ИС. Микропроцессор также управляет электроникой внешних выводов и дисплеями и преобразует единицы измерения. Аналогичное оборудование, реализованное с использованием обычных логических схем или на основе мини-ЭВМ, будет намного дороже.

Основными преимуществами МП в промышленном использовании являются их низкая стоимость, высокая надежность и устойчивость к неблагоприятным условиям окружающей среды. Поэтому их можно применять там, где мини-ЭВМ будут слишком дорогими или ненадежными. В большинстве случаев промышленного использования МП их ограниченное быстродействие и вычислительные возможности не являются существенным недостатком. Однако в управлении технологическими процессами относительно большое число регулируемых параметров и сложность алгоритмов управления требуют применения мини-ЭВМ или очень мощного МП. Микропроцессоры могут также применяться в распределенных системах, где они реализуют алгоритмы управления объектами на местах и готовят данные для центрального процессора: «интеллект» на месте дает возможность повысить степень надежности системы в условиях производственных помех. Основным недостатком МП в промышленном применении является ограниченная возможность ввода-вывода информации. Из-за малых длины слова и числа выводов они не могут оперировать с очень большим числом портов входов и выходов, требующихся для применения в такой системе, как линия сборки.

Административно-хозяйственное оборудование

Микропроцессоры используются в кассовых аппаратах, информационно-справочных табло, системах сбора данных, программируемых калькуляторах, устройствах обработки чеков, терминалах банков, устройствах обработки и преобразования речевых сигналов и в упаковочном оборудовании. Типичным примером является терминал розничной торговли TRW 2001 (рис. 1.21), в котором используется МП Motorola 6800 для передачи данных на центральный компьютер для хозяйственного и бухгалтерского учета, для управления дисплеями, запроса по кредитным чекам, подсчета платежных сумм с учетом налогов и скидок, а также для передачи команд оператору. Терминал может ускорить кассовую работу, считывая цены прямо с бирок и производя вычисления автоматически. Он также немедленно вводит данные о сделке в систему бухгалтерского и хозяйственного учета и значительно снижает число ошибок, которые делаются при ручной записи данных и последующем подсчете. Очевидно, что мини-ЭВМ была бы слишком дорога для такого применения, а системы на основе схем «жесткой» логики не смогли бы обеспечить гибкость применения.

Основным преимуществом МП в административно-хозяйственном оборудовании является их способность при низких затратах адаптироваться к условиям конкретного применения. Микропроцессор может с легкостью выполнять большую часть работы клерков, продавцов, кассиров и секретарей; он может выполнять простые вычисления, регистрировать сделки, быстро и точно получать до-



Рис. 1.20. Автоматический тестер ALMA 720 фирмы ALMA

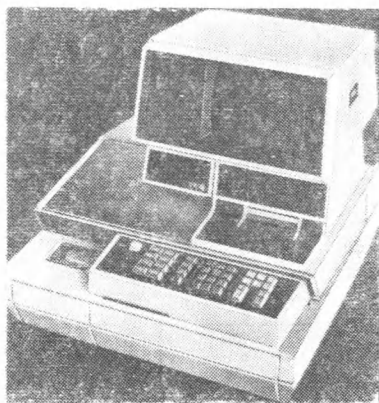


Рис. 1.21. Кассовый аппарат (терминал для розничной торговли) TRW 2001 фирмы TRW Communications Systems and Services

ступ к информации. Более того, МП может выдавать команды, предупредительные сигналы и сообщения об ошибках оператору. Таким образом, применение оборудования на основе МП может уменьшить ошибки в подсчетах, сэкономить затраты на обучение персонала и упростить внесение изменений в финансовые процедуры с изменением ставок налогообложения или наценок и скидок.

Другие преимущества этого оборудования связаны с его возможностью прямо связываться с центральной ЭВМ, в результате чего ускоряется обслуживание клиента, осуществляется более качественное управление операциями и облегчаются сбор и подготовка данных для ввода в систему. Микропроцессоры могут формировать, редактировать и проверять данные до передачи в ЭВМ. Они позволяют также приспособить административно-хозяйственное оборудование для удовлетворения потребностей пользователя и модифицировать его для решения новых задач. Недостатком МП в этой области применения является большой объем программирования, который должен выполнить изготовитель.

Транспорт

Микропроцессоры уже широко используются в контроллерах светофоров и в мобильных системах связи и терминалах. Они также используются в оборудовании электронного зажигания и систем впрыска топлива. В настоящее время проводятся экспериментальные работы в автомобильной промышленности для применения МП в контроллерах зажигания, системах торможения без проскальзывания, диагностических системах внутри автомашины, контроллерах температуры салона и системах цифровой информации и управления. Типичным примером такого прототипа является программируемый контроллер зажигания фирмы Ford (рис. 1.22), в котором используется 12-разрядный специально изготовленный МП для вычисления требуемого угла опережения зажигания автомобильного двигателя и управления положением клапана системы рециркуляции выхлопных газов. Ожидается, что система, основанная на МП, значительно сократит выброс продуктов сгорания и расход горючего. Важными факторами являются малые размеры, низкая энергоемкость, высокая надежность и долговечность микропроцессоров.

Основными проблемами, возникающими при применении МП на транспорте, являются исключительно неблагоприятные условия окружающей среды и отсутствие подходящего оборудования для их контроля и диагностики. Для автомоби-

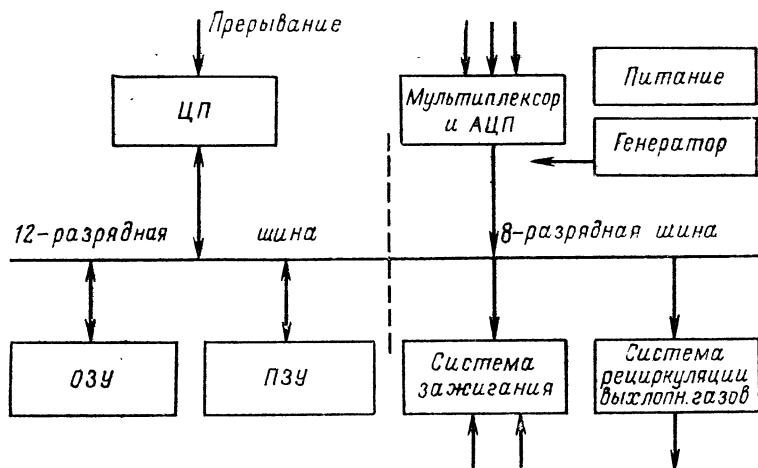


Рис. 1.22. Блок-схема контроллера опережения зажигания

лей, например, требуются устройства, способные выдерживать большие перепады температуры и влажности, значительные перепады тока и напряжения в системе электропитания, механические удары и вибрацию, загрязнение химическими веществами и газами, а также электромагнитные помехи. На судах и самолетах окружающие условия еще хуже, включая брызги соленой воды, высокие уровни вибрации, колебания наружного давления и ядерную радиацию. Микропроцессор и другие вспомогательные схемы должны быть изготовлены на основе подходящей полупроводниковой технологии (обычно К-МОП) и скомпонованы так, чтобы удовлетворить требованиям окружающей среды. Первичные преобразователи и исполнительные механизмы, имеющиеся в настоящее время, или слишком дороги, или не обеспечивают необходимой точности, или неспособны выдерживать сложные условия окружающей среды. Более того, для МП-систем требуется новое контрольно-ремонтное оборудование. Хотя МП могли бы помочь сократить расход топлива и выхлоп газов в автомобилях, а также создать улучшенные панели приборов и предупредительно-диагностические системы, технические и экономические проблемы, связанные с применением их на транспорте, далеки от решения.

Космическое и военное оборудование

Микропроцессоры используются в навигационных системах, системах мобильной связи, тренажерах, системах спутниковой связи и контроллерах радиоуправляемых самолетов. Типичным примером является тактический блок ввода-вывода фирмы Litton Data Systems (рис. 1.23), в котором используется МП Intel 8008 для управления дисплеем, лицевой панелью, работой клавиатуры, а также для осуществления самоконтроля. В военном применении особенно важными являются повышенная степень надежности и гибкости, а также способность к самодиагностике оборудования, основанного на МП. Важными также являются малый размер, низкий уровень потребления энергии, прочность и возможность специализации дисплея для конкретной задачи и оператора. Как и на транспорте, в военном применении требуются устройства, способные выдержать неблагоприятные окружающие условия. Сложностью для военного применения является также отсутствие стандартизации на микропроцессоры.

Потребительские товары (торговля)

Микропроцессоры применяются в таких потребительских товарах, как игры, счетные линейки-калькуляторы, домашние ЭВМ, автоматы для розлива спиртных напитков, бытовые приборы, стереоаппаратура и игральные автоматы.

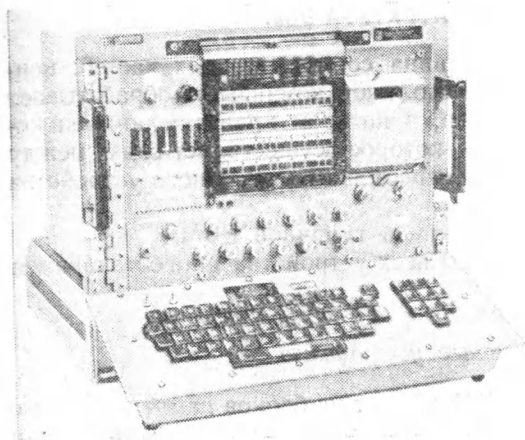


Рис. 1.23. Блок ввода-вывода тактической информации фирмы Litton Data Systems

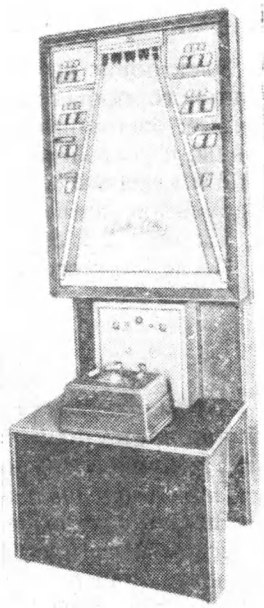


Рис. 1.24. Кегельбан фирмы Bally Manufacturing

Типичным примером является оживленная игра в шары — кегельбай, разработанная фирмой Bally Manufacturing Company (рис. 1.24), в котором применяется МП Intel 4004 для управления перемещением шаров и кеглей, ведения счета, разрешения премиальной игры и займов. Микропроцессор обеспечивает оживление игры, что делает ее реалистичной и интересной; традиционные модели (типа обычных бильярдных игр) не могут обеспечить такого разнообразного и сложного оживления, а мини-ЭВМ является слишком дорогой. Применение МП в потребительской и торговой областях только начинается: имеются трудности, связанные с условиями окружающей среды, а также требуются дешевые и совместимые с МП исполнительные механизмы, датчики и аналого-цифровые преобразователи.

Число и разнообразие применения МП демонстрирует преимущества, предоставляемые новой технологией, однако, как и с любой новой технологией, имеется много трудностей и ограничений. В то же время у дешевых ЭВМ имеются несомненные перспективы. Электронные системы могут быть более гибкими, надежными, дешевыми, точными, и ими проще пользоваться.

ГЛАВА ВТОРАЯ АРХИТЕКТУРА МИКРОПРОЦЕССОРОВ

В этой главе рассматриваются вопросы внутренней организации или *архитектуры* МП.

В первом параграфе описывается структура ЭВМ. Затем рассматривается блок центрального процессора (ЦП); при этом основное внимание обращается на число и типы регистров, схему арифметического устройства и механизм реализации команд. В последних параграфах обсуждаются конкретные особенности архитектуры МП, таких как МП 8080 фирмы Intel и МП 6800 фирмы Motorola.

2.1. ОБЩАЯ АРХИТЕКТУРА ЭВМ

Электронно-вычислительная машина состоит из трех частей: центрального процессора, который исполняет команды и обрабатывает данные; запоминающего устройства, в котором хранятся данные и команды, и устройства ввода-вывода, которое осуществляет связь между ЭВМ и внешним миром. Сигналы пересылаются из одного устройства в другое по шинам.

На рис. 2.1 представлена структура типичной ЭВМ.

После краткого описания работы всех устройств и шин остановимся на особенностях работы ЦП.

Запоминающее устройство

Запоминающее устройство ЭВМ состоит из блоков памяти, которые обычно представляют собой набор блоков магнитных сердечников или БИС полупроводниковой памяти.

Память предназначена для хранения двоичных переменных.

Память организована по *байтам*, которые являются наименьшей адресуемой группой битов, с которой ЭВМ может одновременно оперировать, или по *словам*, имеющим такой же формат в битах, что и рабочие регистры, шина данных и арифметическое устройство ЭВМ.

Байт обычно состоит из 8 бит, в то время как слово может иметь длину от 4 до 64 бит.

Память организована в массив ячеек, каждая из которых имеет свой собственный адрес. Адрес слова в памяти не следует путать с его содержимым. Например, ячейка памяти 0 может содержать любое число. Когда ссылаемся на содержимое ячейки памяти, ее адрес помещаем в круглые скобки. Так, если X — адрес, то (X) — содержимое ячейки памяти по этому адресу.

На рис. 2.2 приведена организация типичной памяти.

Дешифратор адреса получает адрес из ЦП и выбирает соответствующую ячейку памяти.

На рис. 2.3 показан простой однолинейный дешифратор. Если на входной линии 0, то на выходной линии A_0 , а на выходной линии B_1 ; если на входной линии 1, то на выходной линии A_1 , а на выходной линии B_0 . Таким образом, одна линия адресации может осуществлять выборку из двух ячеек памяти: 0 из ячейки памяти B , а 1 из ячейки памяти A .

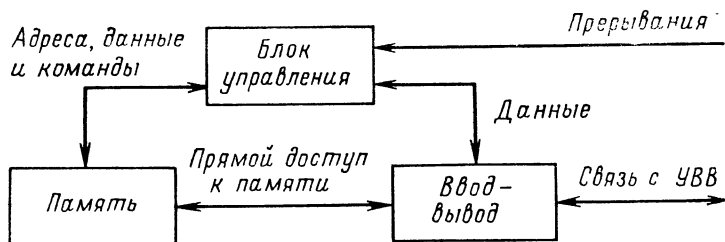


Рис. 2.1. Структурная схема ЭВМ

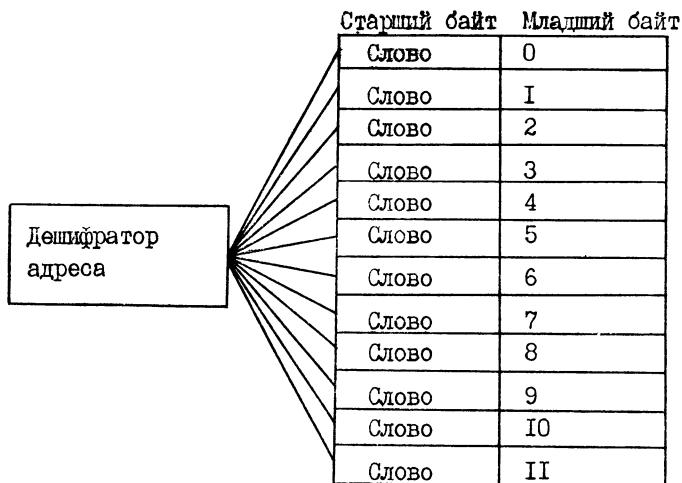


Рис. 2.2. Структура памяти (1 слово=2 байта)
(соотношение между словами и байтами зависит от разрядности ЭВМ)

Выборка нужной ячейки памяти и извлечение ее содержимого требуют определенного времени, которое называется *временем доступа к памяти*.

Время доступа влияет на быстродействие ЭВМ, так как ЭВМ получает команды и большинство данных из памяти. Память ЭВМ обычно является оперативной, так что все ячейки памяти имеют одинаковое время доступа. Как правило, ЭВМ должна переходить в режим ожидания всякий раз, когда ЦП обращается к памяти; время доступа обычно находится в пределах от 100 нс до нескольких микросекунд.

Запоминающие устройства часто подразделяются на блоки, называемые *страницами*. Все устройство может иметь емкость миллионы слов, в то время как страница содержит от 256 до 4 Кбайт.

Электронно-вычислительная машина может получить доступ к ячейке памяти, выбрав сначала определенную страницу, а затем уже ячейку на этой странице. Этот процесс показан на рис. 2.4. Преимущество страничной организации памяти состоит в том, что ЭВМ может иметь доступ к нескольким ячейкам памяти на одной и той же странице, указав только адрес на странице. Этот процесс похож на адресацию домов на улице — сначала называется улица, а потом указываются номера домов.

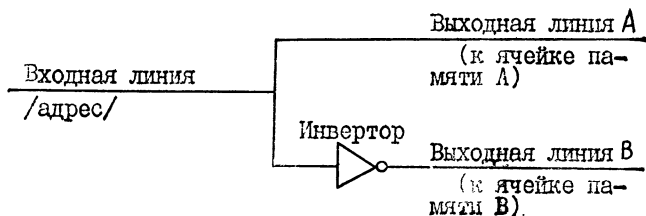


Рис. 2.3. Простейший дешифратор

Центральный процессор осуществляет обмен данными в ЭВМ следующим образом:

- 1) управляющее устройство пересылает адрес в память;
- 2) управляющее устройство генерирует сигнал СЧИТЫВАНИЕ/ЗАПИСЬ и передает его в блок памяти для указания направления передачи;

- 3) управляющее устройство ожидает завершения пересылки. Эта задержка при вводе предшествует пересылке самих данных, а при выводе следует за ней.

Процессор и запоминающее устройство связываются несколькими шинами (рис. 2.5). Шина адреса передает адрес ячейки памяти, к которой происходит обращение. Сигнал СЧИТЫВАНИЕ/ЗАПИСЬ определяет направление передачи. Шина данных пересылает данные между блоками. Некоторые из шин могут быть общими, например одна шина может пересылать данные в разных направлениях или передавать данные и адреса в различные периоды времени. Шина, используемая более чем для одной цели, называется *мультиплексированной*. Дополнительные сигналы управления определяют, что именно находится на этой шине в данный момент времени.

Важной особенностью памяти ЭВМ является то, что она может содержать либо данные, либо команды, причем и то, и другое представлено в двоичном коде. Машина, которая пользуется одним и тем же

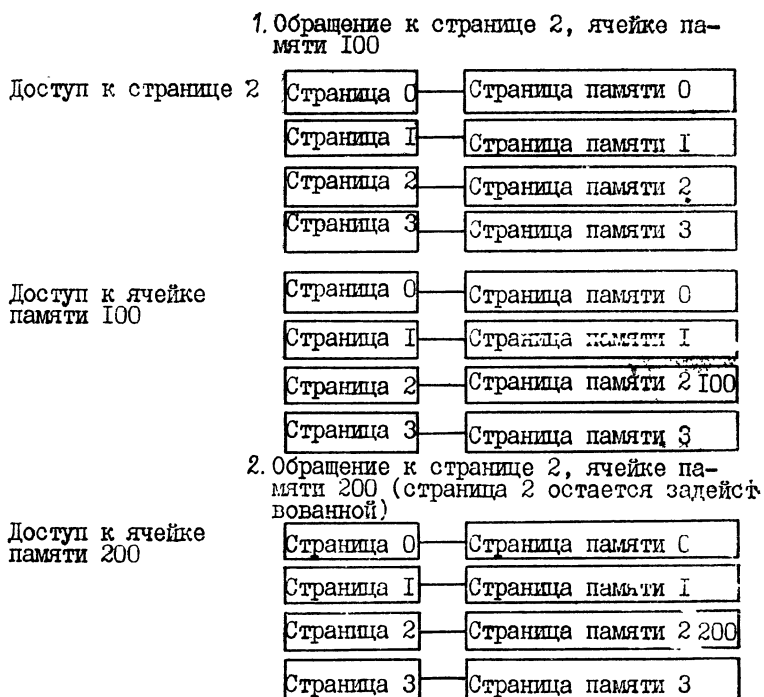
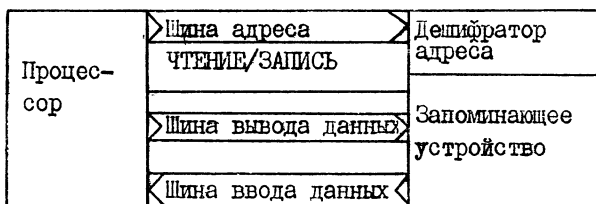


Рис. 2.4. Постраничная адресация

Рис. 2.5. Связи между процессором и памятью



форматом памяти для данных и команд, называется машиной Фон-Неймана, по имени математика, первым предложившим такие машины. Каким образом ЦП узнает, имеет он дело с командой или с единицей данных? Ответ заключается в том, что ЦП только знает, что он рассчитывает получить в определенный момент времени. Если программист сделает ошибку, то ЦП может принять данные за команду или наоборот.

Устройство ввода-вывода

Устройство ввода-вывода осуществляет передачу данных, а также сигналов состояния и управляющих сигналов между ЭВМ и внешним или периферийными устройствами. Процесс передачи включает в себя обмен сигналами состояния и управления и вслед за тем собственно пересылку данных. Устройство ввода-вывода должно регулировать временные различия между ЭВМ и периферийными устройствами, формировать должным образом формат данных, управлять сигналами состояния и управления и обеспечивать требуемый уровень тока и напряжения. Нерегулярные передачи могут управляться сигналами прерывания, которые сразу привлекают внимание ЦП и вызывают приостановку его нормальной работы.

Сама передача данных между ЭВМ и периферийными устройствами происходит быстро, но обеспечение правильной передачи занимает намного больше времени. Типичная операция ввода происходит следующим образом (рис. 2.6):

1) периферийное устройство сигнализирует ЦП о том, что имеются новые данные. Устройство ввода-вывода должно соответствующим образом сформировать сигнал и держать его до тех пор, пока ЦП его не примет;

2) периферийное устройство посылает данные в ЦП. Устройство ввода-вывода должно хранить их до тех пор, пока ЦП не будет готов их считать;

3) центральный процессор считывает данные. Устройство ввода-вывода должно иметь блок дешифрирования, который выбирает определенную часть УВВ (или *порт*). Считывание данных должно снять сигнал, свидетельствующий о том, что данные имеются; результатом этого может быть также подтверждение, посланное периферийному устройству, о том что оно может посылать новые данные.

Операции вывода во многом похожи на операции ввода. Периферийное устройство оповещает ЦП, что оно готово принять данные. После этого ЦП направляет данные вместе с сигналом (стробом), который ука-

зывает периферийному устройству, что данные имеются. Устройство ввода-вывода формирует соответствующим образом данные и сигналы управления и сохраняет данные в течение времени, необходимого для их использования периферийным устройством. Данные вывода должны храниться намного дольше, чем данные ввода, так как механические устройства, отображающие их, реагируют намного медленнее, чем ЭВМ.

Устройство ввода-вывода должно выполнять множество задач простого интерфейса. Оно должно придать сигналам подходящий формат как для управляющего, так и для периферийного устройства. Центральному процессору требуются сигналы с определенными уровнями напряжения. Периферийные устройства могут использовать много различных типов сигналов, включая непрерывные (аналоговые) сигналы различного тока и напряжения. Для сигналов, идущих на большие расстояния или работающих на большие нагрузки, требуются усилители.

Устройство ввода-вывода может также выполнять некоторые функции, которые выполняет ЦП. Эти функции включают в себя преобразование данных из последовательного кода в параллельный, включение или исключение специальных символов, отмечающих начало или конец передачи данных, а также преобразование кодов обнаружения ошибок таких как проверка на четность.

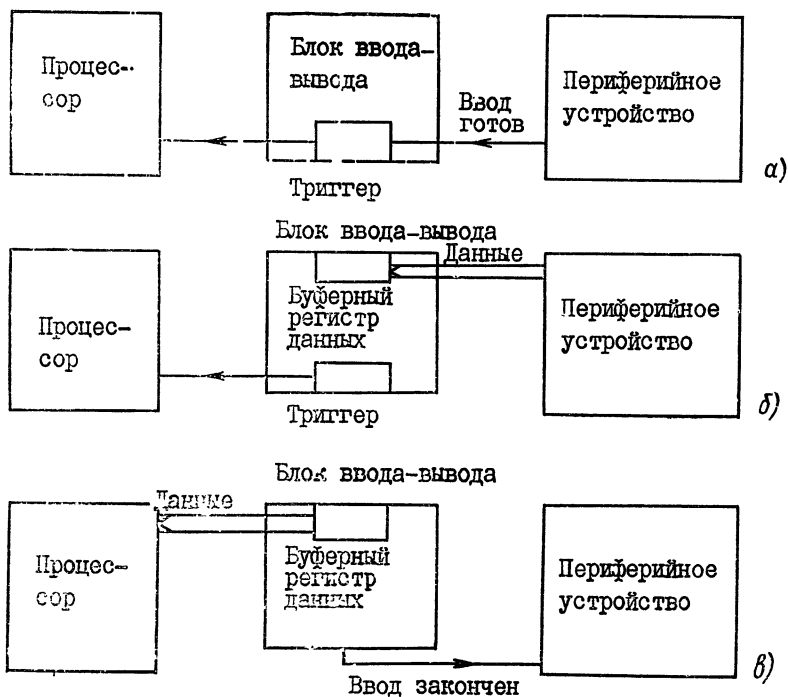
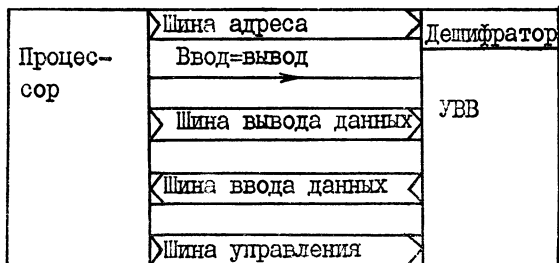


Рис 26. Операция ввода:

а — генерация сигнала о наличии данных; б — перемещение данных; в — считывание данных

Рис. 2.7. Связи между процессором и УВВ



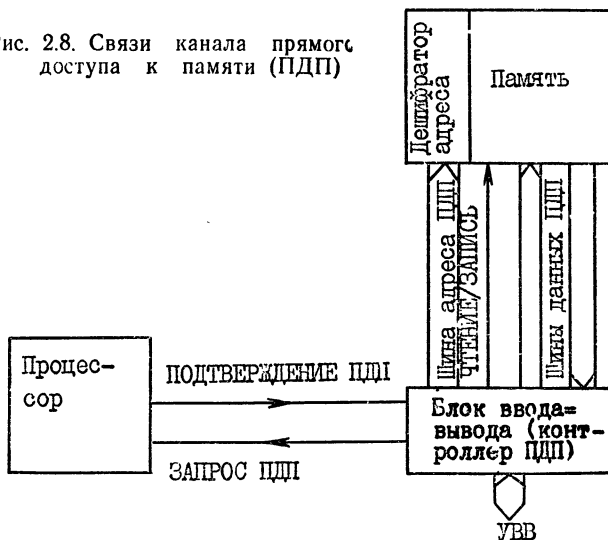
Устройство ввода-вывода может выполнить эти задачи аппаратными средствами быстрее, чем ЦП может выполнить их программными методами. Устройство ввода-вывода ЭВМ может быть программируемым и даже содержать процессор для реализации его некоторых задач.

На рис. 2.7 показаны связи между ЦП и УВВ. Адресная шина передает адрес порта ввода или вывода, который нужен для использования ЦП. Сигнал ввода-вывода определяет направление передачи. По шине данных осуществляется передача информации между устройствами. Шина управления передает сигналы, указывающие, что данные готовы и что передача завершена. Что касается шин между ЦП и ЗУ, то некоторые из них могут быть одними и теми же, но разделенными во времени для выполнения различных операций.

Более того, шины могут соединять ЦП как с памятью, так и с УВВ. Одна линия управления может определять назначение блоков. Действительно, некоторые ЭВМ (например, Motorola 6800) полностью совмещают по адресному полю память и УВВ; они обращаются к устройствам ввода или вывода так же, как к ячейкам памяти.

Современные ЭВМ имеют прямую связь между памятью и УВВ, что позволяет осуществлять передачу данных к периферийным уст-

Рис. 2.8. Связи канала прямого доступа к памяти (ПДП)



роЙствам и обратно без участия ЦП. Этот метод передачи данных называется *прямым доступом к памяти* (ПДП). Преимуществом ПДП является то, что скорость передачи ограничивается только временем доступа к памяти (обычно менее 1 мкс). Для передачи данных через ЦП требуется несколько команд, и на это уходит в 10—20 раз больше времени. Прямой доступ к памяти применяется с быстродействующими периферийными устройствами, такими как магнитные диски, быстродействующие линии связи или дисплей.

Связи, требующиеся для организации ПДП, показаны на рис. 2.8. Контроллер ПДП, являющийся частью УВВ, управляет передачей данных так же, как ЦП. Сигнал управления ЗАПРОС ПДП исключает возможность управляющему устройству пользоваться памятью в одно и то же время с контроллером ПДП.

Центральный процессор

Центральный процессор обрабатывает данные. Он выбирает команды из памяти, дешифрирует их и выполняет. Он вырабатывает временные сигналы и сигналы управления, передает данные в память и из памяти и устройств ввода-вывода, выполняет арифметические и логические операции и идентифицирует внешние сигналы. На рис. 2.9 показан типичный ЦП.

В течение каждого цикла команды ЦП выполняет много управляющих функций:

- 1) помещает адрес команды в адресную шину памяти;
- 2) получает команду из шины ввода данных и дешифрирует ее;
- 3) выбирает адреса и данные, содержащиеся в команде; адреса и данные могут находиться в памяти или в регистрах;

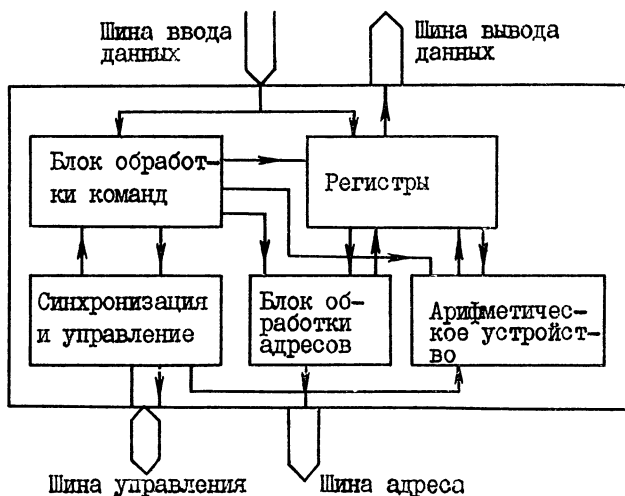


Рис. 2.9. Процессор

4) выполняет операцию, определенную в коде команды. Операцией может быть арифметическая или логическая функция, передача данных или функция управления;

5) следит за управляющими сигналами, такими как прерывание, и реагирует соответствующим образом;

6) генерирует сигналы состояния, управления и времени, которые необходимы для нормальной работы УВВ и памяти.

Таким образом, ЦП является «мозгом», определяющим действия ЭВМ.

2.2. РЕГИСТРЫ

Основу большинства ЦП образуют рабочие регистры. Регистры представляют собой сверхоперативное ЗУ небольшой емкости. Регистры состоят из триггеров и адресуются подобно ячейкам памяти. Число регистров, однако, очень невелико. Данные могут храниться в регистре до тех пор, пока шина или некоторый блок не будут готовы принять их или пока они не потребуются по программе. Использование в программе рабочих регистров выгодно, так как ЦП может получить содержащиеся в них данные, не обращаясь к памяти. Регистры, содержимое которых не изменяется под воздействием программы, позволяют сохранить данные для последующего использования.

С помощью внутренних шин регистры связаны друг с другом. С другими блоками системы связь осуществляется под управлением программы.

Стоимость изготовления на кристалле большого числа регистров и внутренних соединений ограничивает их число в БИС.

Если ЦП имеет большое число регистров, программе не потребуется большого числа пересылок данных в память и из памяти. Благодаря этому уменьшается число операций обращения к памяти и формат команд. Наличие большого числа внутрипроцессорных регистров приводит к расширению возможностей дешифрирования и адресации команд и данных, и эта тенденция имеет будущее.

На рис. 2.10 показан типовой набор регистров ЦП. Регистры могут иметь много различных назначений; некоторые ЭВМ даже позволяют программисту присваивать регистрам разнообразные частные функции. Большинство ЭВМ, однако, содержит несколько основных регистров: счетчик команд, регистр команд, регистр адреса памяти, аккумулятор, регистры общего назначения, индексные регистры, регистр условий, указатель стека.

Счетчик команд (СК) содержит адрес ячейки памяти, в которой находится очередная команда. Цикл выполнения команды начинается с того, что ЦП посылает содержимое счетчика команд в шину адреса; таким образом ЦП извлекает из памяти первое слово команды. При этом увеличивается на единицу содержимое счетчика команд и, таким образом, в следующем цикле команды из памяти будет извлечена следующая из последовательности команд. Если команда многобайтная, то ЦП увеличивает на 1 содержимое счетчика команд (инкрементирует) столько раз, сколько это нужно. Таким образом, ЦП извлекает из

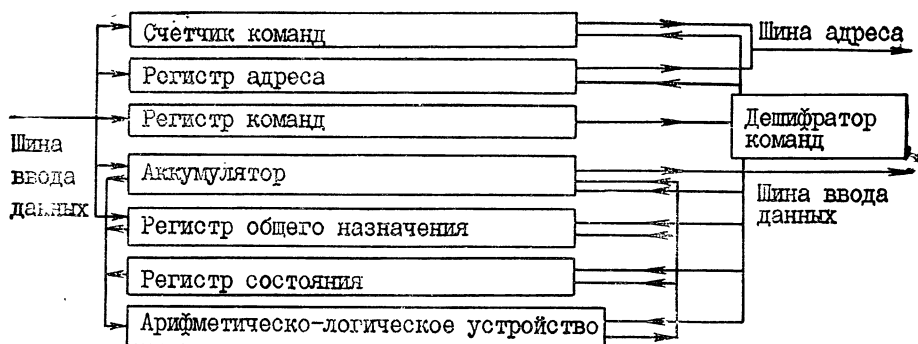


Рис. 2.10. Блок регистров

памяти и реализует команды последовательно, если только команда ^a ПЕРЕДАЧА УПРАВЛЕНИЯ ИЛИ УСЛОВНЫЙ ПЕРЕХОД не изменит содержимое счетчика команд.

Регистр команд сохраняет код команды до тех пор, пока она не будет дешифрована.

Регистр адреса памяти содержит адрес данных в памяти. Адреса могут представлять собой часть команд или данные. На рис. 2.11 показаны возможности использования регистра адреса памяти. Команда ЗАГРУЗИТЬ В АККУМУЛЯТОР ЧИСЛО ИЗ ПАМЯТИ загружает аккумулятор содержимым ячейки памяти с адресом 300. Содержимое аккумулятора инкрементируется (ПРИБАВИТЬ 1 К СОДЕРЖИМОМУ

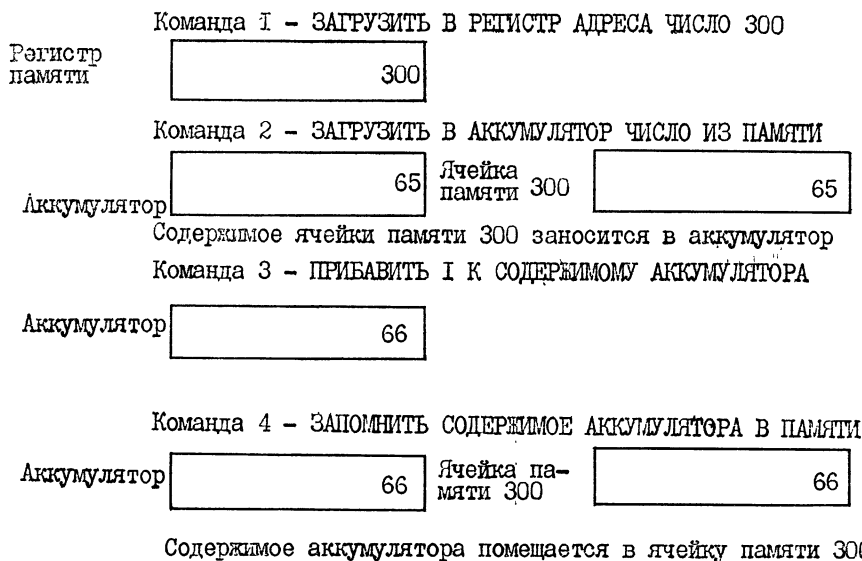


Рис. 2.11. Использование регистра адреса для увеличения на 1 содержимого ячейки памяти

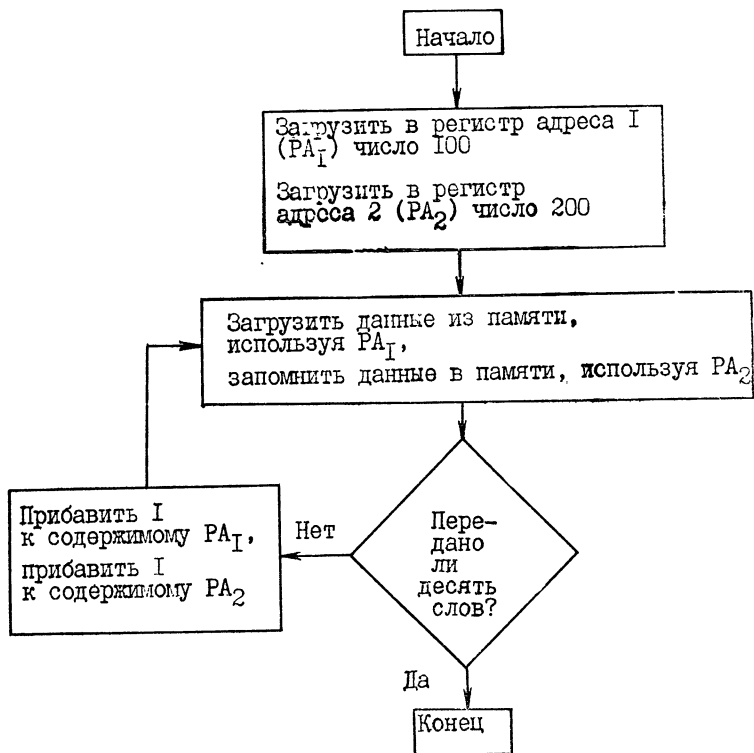


Рис. 2.12. Использование двух регистров адреса для передачи данных (задача: передать десять слов из ячеек памяти 100—109 в ячейки памяти 200—209)

АККУМУЛЯТОРА) и запоминается в ячейке 300 (ЗАПОМНИТЬ СОДЕРЖИМОЕ АККУМУЛЯТОРА В ПАМЯТИ). Таким образом, ЭВМ может использовать содержимое ячейки памяти 300 несколько раз без указания ее адреса в каждой команде и использовать следующую ячейку памяти (301) путем добавления 1 к содержимому регистра адреса.

Многие ЭВМ имеют по несколько регистров адреса. На рис. 2.12 представлена блок-схема программы, которая использует регистры адреса памяти для того, чтобы переслать десять слов данных из одной части памяти в другую. На рис. 2.12 команды ЗАГРУЗИТЬ ДАННЫЕ и ЗАПОМНИТЬ ДАННЫЕ не требуют адресов памяти; 1 бит в команде мог бы определить, использует ЦП первый регистр адреса памяти или второй.

Аккумуляторы — это регистры временного хранения, которые используются в процессе вычисления. В большинстве ЭВМ, таких как калькуляторы, в аккумуляторе всегда содержится один из операндов арифметических операций. Электронно-вычислительная машина может также использовать аккумуляторы при выполнении логических операций. Таким образом, аккумуляторы в ЭВМ — это в основном наибо-

лее часто используемые регистры. Многие из широко используемых ЭВМ имеют по одному аккумулятору; программы для таких ЭВМ затрачивают много команд и времени на пересылку данных в аккумулятор и из него. Большинство более совершенных ЭВМ имеет по несколько аккумуляторов; поэтому у программы нет необходимости многократно пересылать данные. Например, на рис. 2.13 показана последовательность команд, которая реализует выражение

$$A \times B + C \times D$$

для ЭВМ, имеющей один аккумулятор, и для ЭВМ с двумя аккумуляторами. При наличии только одного аккумулятора программа должна запомнить первый промежуточный результат и затем вновь обратиться к нему; с двумя аккумуляторами программа может выполнять вычисления раздельно. Однако в машине с двумя аккумуляторами каждая команда должна содержать признак для определения того, какой из аккумуляторов должен использоваться ЦП.

Регистры общего назначения выполняют различные функции. Они могут служить в качестве регистров временного хранения данных или адресов. Программисту предоставляется возможность определять их как аккумуляторы или как счетчики команд.

Индексные регистры используются для адресации данных. Содержимое индексного регистра складывается с адресом ячейки памяти, который содержится в команде. Затем сумма образует действительный адрес данных или исполнительный адрес. Если содержимое индексного регистра изменяется, одна и та же команда может быть использована для обработки данных из ячеек памяти с различными адресами. Можно пересылать данные из одной области памяти в другую подобно тому, как если бы использовались индексные регистры. На рис. 2.14 иллюстрируется этот способ. Команды ЗАГРУЗИТЬ и ЗАПОМНИТЬ должны содержать адреса ячеек памяти. Однако эта программа использует один индексный регистр вместо двух регистров

Один аккумулятор	Два аккумулятора
ЗАГРУЗИТЬ В АККУМУЛЯТОР ЧИСЛО А УМНОЖИТЬ НА В ЗАПОМНИТЬ РЕЗУЛЬТАТ В РЕГИСТРЕ ВРЕМЕННОГО ХРАНЕНИЯ ЗАГРУЗИТЬ В АККУМУЛЯТОР ЧИСЛО С УМНОЖИТЬ НА D ПРИБАВИТЬ К СОДЕРЖИМОМУ РЕГИСТРА ВРЕМЕННОГО ХРАНЕНИЯ	ЗАГРУЗИТЬ В АККУМУЛЯТОР 1 ЧИСЛО А УМНОЖИТЬ НА В ЗАГРУЗИТЬ В АККУМУЛЯТОР 2 ЧИСЛО С УМНОЖИТЬ НА D СЛОЖИТЬ СОДЕРЖИМОЕ ДВУХ АККУМУЛЯТОРОВ

Рис. 2.13. Пример программы для МП с одним и с двумя аккумуляторами

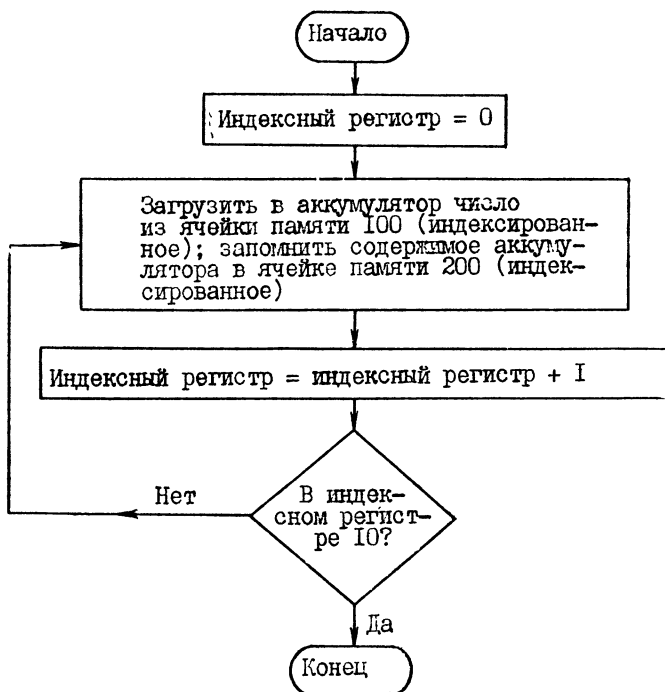


Рис. 2.14. Использование индексного регистра для передачи данных (задача: передать десять слов из ячеек памяти 100—109 в ячейки памяти 200—209)

адресов памяти. Некоторые ЭВМ, такие как DEC PDP-8 и PDP-11, имеют автоиндексацию, при помощи которой индексный регистр каждый раз, когда используется, автоматически увеличивает (автоинкрементация) или уменьшает (автодекрементация) содержимое на 1. Эта особенность очень важна в программных циклах, таких как представленные на рис. 2.12 и 2.14. Каждая команда в ЭВМ с индексными регистрами должна содержать признаки, указывающие, используется ли в данной команде индексация. Если ЭВМ имеет более чем один индексный регистр, команда должна также содержать указание, какой из регистров в данной команде используется.

Регистр кода условий или регистр состояния содержит набор одноразрядных признаков, которые отображают состояние ЦП или нескольких внешних входов или выходов. Эти признаки — основа для работы ЭВМ, принимающей решение. Различные ЭВМ имеют различное число и назначение признаков. Большинство устаревших ЭВМ имеет один или два признака, так как это связано, прежде всего, со стоимостью аппаратных средств. Новейшие ЭВМ имеют по несколько признаков. Наиболее распространенными признаками являются следующие:

ПЕРЕНОС—1, если при выполнении последней операции в старшем

разряде регистра образуется признак переноса. Признак ПЕРЕНОС может сохранять свое значение или может принимать участие в переносе от одного слова к другому в арифметических операциях многократной точности.

НУЛЬ—1, если результат последней операции равен нулю. Это часто используется в управлении циклом и в процессе поиска некоторого адресуемого числа.

ПЕРЕПОЛНЕНИЕ—1, если последняя операция выдает дополнительный код с переполнением. Бит ПЕРЕПОЛНЕНИЕ определяет, не превышает ли результат арифметической операции разрядности процессора.

ЗНАК—1, если старший значащий бит результата последней операции был 1 (иногда называется ОТРИЦАТЕЛЬНЫМ, так как 1 означает, что число в дополнительном коде отрицательное). Бит ЗНАК используется в арифметических операциях и при анализе знакового разряда в пределах многобайтного слова.

ЧЕТНОСТЬ—1, если в результате последней операции число единичных бит (число единиц) в слове было нечетным (нечетный паритет) или в противном случае четным (четный паритет).

ВСПОМОГАТЕЛЬНЫЙ ПЕРЕНОС—1, если в результате последней операции возник признак переноса в младшем полуслове, что используется для выполнения арифметических операций с двоично-кодированными десятичными числами.

РАЗРЕШЕНИЕ ПРЕРЫВАНИЯ—1, если прерывание разрешается, 0 — если прерывание не разрешено программой. Центральный процессор может автоматически запретить прерывания в периоды запуска сервисных программ или выхода из них. Программист может запретить прерывания в течение циклов операций со многими словами. Электронно-вычислительная машина может иметь несколько признаков разрешения прерывания, если имеется несколько источников или уровней прерывания.

Центральный процессор может иметь признаки, которые могут быть изменены или сохранены извне при операциях ввода или вывода.

2.3. АРИФМЕТИЧЕСКОЕ УСТРОЙСТВО

Арифметическая часть ЦП может варьироваться от простого сумматора до сложного блока, который выполняет многие арифметические и логические функции. Если арифметическое устройство не может выполнить операцию аппаратным способом, то для получения желаемого результата необходимо использовать последовательность команд.

Все современные ЭВМ имеют двоичные сумматоры, принимающие два двоичных входных сигнала и получающие двоичную сумму и признак переноса в старшем значащем разряде суммы. Так как большинство ЭВМ оперирует с дополнительным кодом, то вычитание может быть выполнено путем подачи на один из входов сумматора числа в дополнительном коде. Умножение и деление может быть выполнено с помощью повторяющихся операций сложения и вычитания соответ-

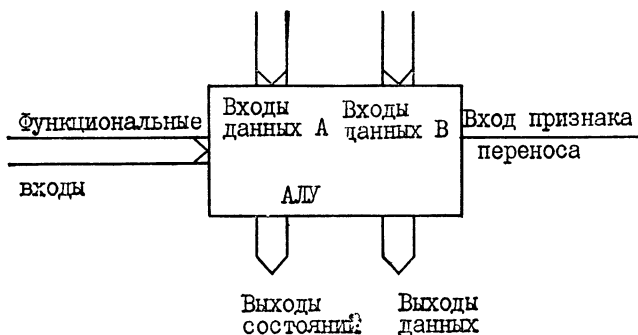


Рис. 2.15. Арифметическо-логическое устройство

ственно. Дополнительные схемы могут сформировать другие признаки результата, такие как, например, нулевое значение.

Широко применяются в микроэлектронной технике однокристалльные арифметическо-логические устройства (АЛУ). Эти устройства состоят из двоичного сумматора и вспомогательных логических схем. Типичное АЛУ показано на рис. 2.15. Оно имеет два входа данных, функциональные входы, вход признака переноса для выполнения операций умножения и деления, выходы данных и состояний, которые представляют собой различные признаки, описанные в предыдущем параграфе. Функциональные входы определяют, какую операцию решает АЛУ. Типичные операции:

сложение, вычитание, логическое И, логическое (включающее) ИЛИ, логическое ИСКЛЮЧАЮЩЕЕ ИЛИ, логическое НЕ (дополнение), инкрементирование (увеличение на 1), декрементирование (уменьшение на 1), сдвиг влево (сложение входного сигнала с самим собой), обнуление (результат равен 0).

Функциональные входы также определяют режим работы шины. Например, число, представленное в шине А, может быть увеличено на 1 путем маскирования шины В (число, поступающее в АЛУ по шине В, равно 0), установки в 1 входа признака переноса и сложения. Таким образом,

$$\text{OUTPUT} = A + B + \text{CARRY} = A + 0 + 1 = A + 1$$

Другие операции позволяют выполнять сравнение входной или выходной величины с целью решения задач типа $A + B$ или $A \cdot B$. Таким образом, АЛУ может решать любую из разнообразных задач в течение одного цикла под управлением сигналов на функциональных входах.

При помощи специальных дополнительных схем могут также решаться другие арифметические задачи, такие как умножение и деление, нахождение синуса и косинуса, определение логарифмов или экспонент. Эти специальные схемы способны обеспечивать высокое быстродействие, но арифметические блоки, которые могут непосредственно решать специализированные задачи, стоят гораздо дороже, чем стандартные блоки АЛУ.

2.4. РЕАЛИЗАЦИЯ КОМАНД

Центральный процессор должен преобразовать код команды, полученной им из памяти, в управляющие сигналы, которые осуществляют в ЦП определенные действия над данными. Центральный процессор должен определить адреса ячеек памяти и регистров, которые использует команда, сформировать сигналы на функциональных входах АЛУ и скоммутировать шины таким образом, чтобы команда была выполнена правильно. Например, если имеется команда СЛОЖИТЬ СОДЕРЖИМОЕ РЕГИСТРА 1 С СОДЕРЖИМЫМ РЕГИСТРА 2 И РЕЗУЛЬТАТ ПОМЕСТИТЬ В РЕГИСТР 3, то, если ЭВМ построена, как показано на рис. 2.16, а, центральный процессор реализует цикл команд следующим образом:

1) содержимое регистра 1 помещает в регистр временного хранения 1 (рис. 2.16,б). Этот шаг необходим потому, что рабочие регистры и АЛУ связаны общей шиной. Таким образом, ЦП должен запомнить содержимое регистра 1 в регистре временного хранения, который непосредственно связан с АЛУ;

2) помещает содержимое регистра 2 в регистр временного хранения 2 (рис. 2.16,в);

3) выполняет операцию сложения и результат помещает в регистр 3 (рис. 2.16,г).

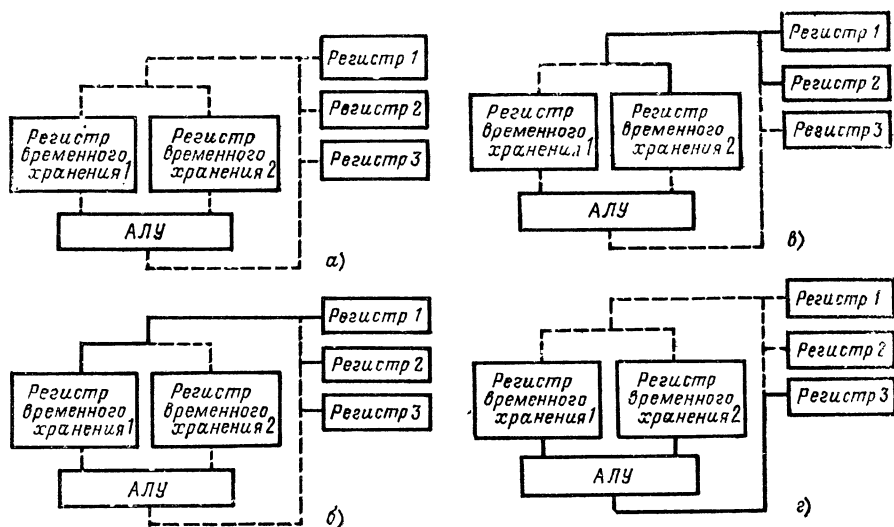


Рис. 2.16. Выполнение команды СЛОЖИТЬ СОДЕРЖИМОЕ РЕГИСТРА 1 С СОДЕРЖИМЫМ РЕГИСТРА 2 И РЕЗУЛЬТАТ ПОМЕСТИТЬ В РЕГИСТР 3:
 а — основная организация; б — из регистра 1 в регистр временного хранения 1; в — из регистра 2 в регистр временного хранения 2; г — из регистров временного хранения в АЛУ, результат — в регистр 3

Центральный процессор выполняет эти шаги в определенном порядке и должен завершить очередной шаг, прежде чем приступить к последующему.

Центральный процессор должен также извлечь команду из памяти, поместить ее в регистр команд и произвести выборку очередной команды. Центральный процессор реализует цикл команды следующим образом:

1) передает содержимое счетчика команд в шину адреса памяти с целью выборки очередной команды;

2) увеличивает на 1 содержимое счетчика команд, чтобы быть готовым к выборке последующей команды;

3) выбирает команду с входной шины данных по адресу и помещает ее в регистр команд;

4) дешифрирует код команды и исполняет соответствующую операцию.

Первые три шага в этом цикле одинаковые для любой команды.

Прежде всего ЦП выполняет требуемую последовательность операций. Так работает ЦП с устройством управления, построенным по принципу жесткой логики. Однако цикл команды — это последовательность операций. Устройство управления инициирует серию операций, которые, в свою очередь, исполняют команду ЭВМ. Написание последовательности операций, которые выполняют команды ЭВМ, называется *микропрограммированием*.

Микропрограммирование аналогично переводу с французского языка на английский путем перевода сначала с французского на немецкий, а затем с немецкого на английский.

Достигается ли что-либо этим процессом в действительности или это просто усложняет задачу? Ответ: микропрограммирование заменяет аппаратные средства программными и обеспечивает высокую гибкость ценой потери быстродействия. Заметим, что ЭВМ могла бы выполнить команду как последовательность микрокоманд:

Микрокоманда 1:

ЗАНЕСТИ СОДЕРЖИМОЕ СЧЕТЧИКА КОМАНД В АДРЕСНУЮ ШИНУ.

Микрокоманда 2:

УВЕЛИЧИТЬ НА 1 СОДЕРЖИМОЕ СЧЕТЧИКА КОМАНД.

Микрокоманда 3:

ДАННЫЕ С ВХОДНОЙ ШИНЫ ЗАНЕСТИ В РЕГИСТР КОМАНД.

Центральный процессор должен использовать код команды, чтобы найти последовательность микрокоманд, требуемую для ее исполнения. Например, ЭВМ могла бы выполнить команду (см. рис. 2.16) следующим образом:

Микрокоманда 4:

СОДЕРЖИМОЕ РЕГИСТРА 1 ЗАНЕСТИ В РЕГИСТР ВРЕМЕННОГО ХРАНЕНИЯ 1.

Микрокоманда 5:

СОДЕРЖИМОЕ РЕГИСТРА 2 ЗАНЕСТИ В РЕГИСТР ВРЕМЕННОГО ХРАНЕНИЯ 2.

Микрокоманда 6:

СЛОЖИТЬ.

Микрокоманда 7:

РЕЗУЛЬТАТ ИЗ АЛУ ЗАНЕСТИ В РЕГИСТР 3.

Микрокоманда 8:

ВЕРНУТЬСЯ К МИКРОКОМАНДЕ 1.

Действия, выполненные ЦП под управлением микрокоманды, много проще, чем действия, выполняемые по командам ЭВМ. Кроме того, одни и те же микрокоманды могут быть частью многих различных команд. Например, для команды ВЫЧЕСТЬ потребуется только изменение в микрокоманде 6. Изменяя микропрограммы, можно создать полностью новую систему команд; изменить аппаратную часть ЦП для того, чтобы получить модернизированную ЭВМ, нельзя. Действительно, ЭВМ, которая выполняет команды с помощью микропрограмм, может быть предназначена для того, чтобы оперировать с системой команд другой ЭВМ. Этот процесс называется *эмуляцией*.

Естественно, микропрограммирование вносит некоторые неудобства. Этот процесс более медленный, чем аппаратное дешифрование кодов команд.

Микрокоманды трудны для написания и проверки, а средства автоматизации микропрограммирования слабо развиты.

Заметим, что нет необходимости устанавливать «родство» между микропрограммированием и микропроцессорами, несмотря на схожесть в названиях. Микропроцессор подобно любому другому управляющему устройству ЭВМ или может быть микропрограммируемым или может иметь неизменяемую систему команд.

2.5. СТЕК

Большинство современных ЭВМ использует стек для того, чтобы обрабатывать данные в определенном порядке. Стеки являются устройствами магазинной памяти (LIFO); можно только или добавлять элемент данных в его верхнюю ячейку, или извлекать его оттуда. Добавление элемента данных в стек есть операция PUSH (занести в стек), извлечение данных из стека—POP или PULL. На рис. 2.17 показаны результаты операций занесения в стек и извлечения из него.

Некоторые ЭВМ имеют стеки, которые физически похожи на распределители тарелок в кафетериях, т. е. действительно все элементы данных физически сдвигаются, когда ЭВМ заносит или выбирает элементы. Стек, показанный на рис. 2.17, один из тех, в которых двигаются все элементы. Этот тип стека может быть построен на основе реверсивного сдвигового регистра.

В стеках большинства ЭВМ элементы данных в действительности не перемещаются. Единственное изменение, которое имеет место, приходится на указатель стека, который содержит адрес верхнего элемента стека. Электронно-вычислительная машина добавляет один элемент в

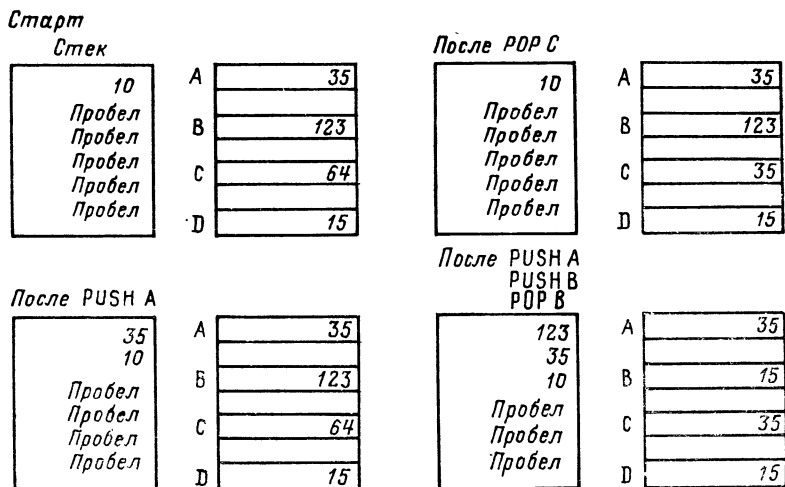


Рис. 2.17. Работа стека

стек, помещая элемент в ячейку памяти, адресуемую указателем стека, а затем увеличивает на 1 содержимое указателя стека. Извлечение элемента из стека производится путем уменьшения на 1 содержимого указателя стека и извлечения этого элемента из соответствующей ячейки памяти.

На рис. 2.18 показаны примеры этих операций. Заметим, что элементы данных в стеке совсем не передвигаются. Этот тип стека использует обычное ОЗУ.

Основное преимущество стека состоит в том, что можно заносить в него данные (увеличивать емкость стека), не разрушая структуру уже записанных в нем данных. Если данные запоминаются в ячейках памяти или в регистре, то теряется предыдущее содержимое этого участка памяти. Таким образом, прежде чем использовать одну и ту же

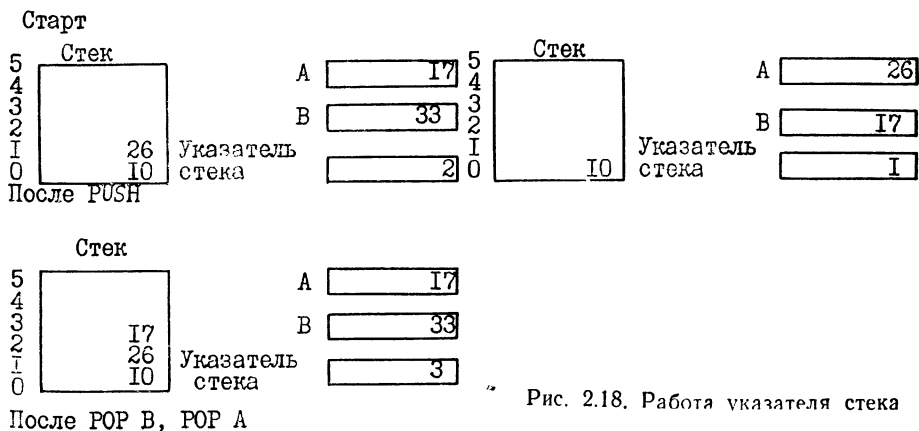


Рис. 2.18. Работа указателя стека

ячейку памяти или регистр снова, необходимо где-то запомнить ее содержимое. Вместе с тем можно использовать стек снова и снова, так как предыдущее содержимое его автоматически сохраняется. Кроме того, ЦП может легко и быстро передавать данные в стек и из стека, так как адрес содержится в указателе стека, а не является частью команды. Формат команд операций со стеком очень короткий.

Стек можно использовать для того, чтобы иметь возможность очень просто запоминать адреса возвратов при работе с подпрограммами, как это показано на рис. 2.19.

Каждая команда ПЕРЕХОД К ПОДПРОГРАММЕ автоматически засылает адрес возврата из счетчика команд в стек. Каждая команда ВОЗВРАТ извлекает адрес возврата в программу из стека и помещает его в счетчик команд.

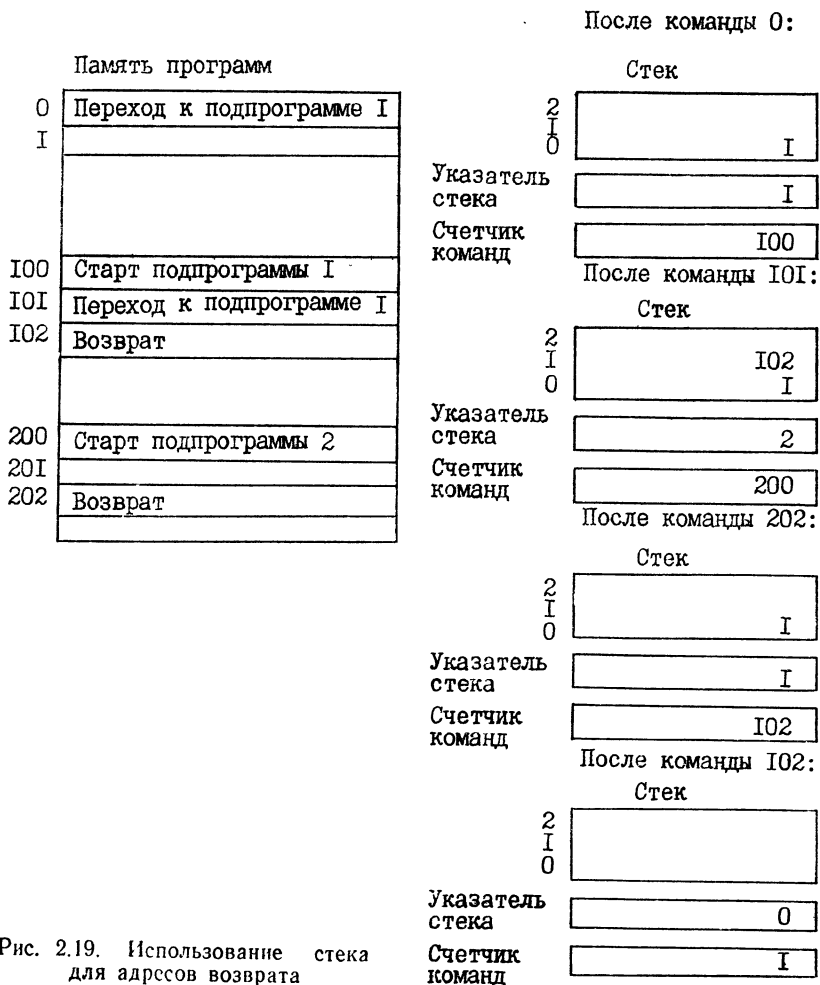


Рис. 2.19. Использование стека для адресов возврата

Основной недостаток использования стековой памяти состоит в сложности отладки и документирования программ, оперирующих со стеком. Так как элемент данных в стеке не имеет фиксированного в программе адреса, то его содержимое может быть трудно для распознавания. Ошибки в использовании стека программистом очень трудны для обнаружения. Типичными примерами ошибок являются: перемещение из стека данных в неверном порядке, помещение в стек или выборка из него избыточных данных, переполнение стека или потеря данных в стеке.

2.6. ОСОБЕННОСТИ АРХИТЕКТУРЫ МИКРОПРОЦЕССОРОВ

В этом параграфе излагаются вопросы организации блоков регистров и арифметического устройства МП.

Регистры микропроцессоров

Регистры микропроцессоров отличаются от регистров больших ЭВМ следующими характеристиками:

1) ограниченным размером кристалла. Однокристалльный микропроцессор может иметь только ограниченное число узкоформатных регистров и шин;

2) использованием ПЗУ для хранения программ. Микропроцессоры в результате этого не могут хранить адреса или данные в программной памяти;

3) ограниченной емкостью ОЗУ;

4) узким форматом слов. Адрес памяти может занимать несколько слов данных;

5) механизмом обработки прерываний. Регистры должны быть приспособлены к быстрому распознаванию и обслуживанию прерываний;

6) структурами специального назначения. Многие микропроцессоры имеют регистры, предназначенные для специальных применений, таких как калькуляция, управление терминалом или процессом.

Очевидно, что каждый из перечисленных факторов предъявляет различные требования к МП по числу и типам регистров.

Предельный размер кристалла МП и необходимость быстро запомнить содержимое регистров при обработке прерываний ограничивают число регистров. Короткий формат слова означает, что для простой адресации регистров их число должно быть невелико. Вместе с тем использование ПЗУ для хранения программ и ограниченная емкость ОЗУ создают необходимость иметь на кристалле МП память для временного хранения данных и адресов. Сложности адресации внешней памяти при коротком формате команды также делают желательным большой объем регистровой памяти в микропроцессоре.

Ниже приводятся типичные особенности архитектуры микропроцессоров.

1. Большинство микропроцессоров имеет несколько регистров общего назначения. Микропроцессор Fairchild F8 имеет 64, Intel 8080 и Signetics 2650—6, Intel 4040 — 24. Некоторые процессоры, включая

Motorola 6800, совсем не имеют регистров общего назначения. Однако Motorola 6800 обеспечивает очень быстрый доступ в определенные ячейки памяти (по нулевой странице).

2. Почти все микропроцессоры имеют по одному аккумулятору. Однако у МП Motorola 6800 их два, а у National PACE четыре, один из которых выполняет функции аккумулятора только для ограниченного набора команд.

3. Почти все микропроцессоры имеют стековую память для сохранения адресов возврата при работе с подпрограммами.

Некоторые МП, такие как Intel 4040 и Signetics 2650, имеют на кристалле стек ограниченной емкости (обычно 4—10 уровней вложений). Другие, включая Intel 8080 и Motorola 6800, организуют стеки во внешней оперативной памяти, а на кристалле имеют только указатель стека, что позволяет сделать стек такой длины, какая требуется. Только некоторые микропроцессоры, такие как Scientific Micro Systems Interpreter, не имеют стека.

4. Большинство микропроцессоров имеет особенности в организации регистров для обработки прерываний. Особенность может заключаться в использовании различных способов сохранения и восстановления содержимого регистров в процессе прерываний. Такая особенность имеется в МП Intel 4040 и Signetics 2650. Микропроцессоры, которые имеют всего несколько регистров (например, Motorola 6800), могут реагировать на прерывания без существенных временных затрат. Узкий формат слова микропроцессоров затрудняет операции с адресами. Специальная организация регистров позволяет преодолеть эти трудности.

5. Переменная длина регистров. Часто некоторые регистры, особенно счетчики команд, регистры адреса, указатели стеков и индексные регистры, бывают длиннее обычного формата слова процессора. Специальные команды позволяют манипулировать с содержимым этих длинных регистров.

6. Некоторые регистры адреса программно доступны. Эта особенность позволяет программе помещать в эти регистры начальные адреса блоков данных, а затем простыми средствами использовать их в последовательностях команд. Микропроцессоры Zilog-80, Intel 4040, Intel 8080, RCA CDR 1802 и Fairchild-8 используют этот метод для операций с адресами.

7. Использование спаренных регистров. Многие МП имеют регистры, которые могут адресоваться либо в одиночку, либо в парах. Тогда в программе можно использовать эти регистры либо по одному для представления десятичных цифр или символов, либо в паре для адресов или данных удвоенного формата. Эту особенность имеют МП Intel 4040, Intel 8080 и RCA CDR 1802.

8. Длинные индексные регистры. Некоторые микропроцессоры, в том числе и Motorola 6800, имеют индексный регистр, который используется для адресации памяти. Индексный регистр имеет достаточную разрядность для того, чтобы охватить все адресное пространство; команды с индексной адресацией выполняют короткие смещения от этого адреса.

Это использование индексного регистра отличается от обычного тем, что в МП реализуются короткие смещения.

Арифметическое устройство микропроцессора

Почти все микропроцессоры имеют простые арифметические устройства. Только некоторые (например, Data, General micro-Nova and Texas Instruments 9900) имеют аппаратно реализуемые операции умножения и деления. Некоторые процессоры, такие как Intel 4040 и Texas Instruments TMS 1000NC, не имеют логических операций. Некоторые процессоры, включая Intersil 6100 и Scientific Micro Systems Interpreter, даже не выполняют непосредственно операцию вычитания; они должны сначала представить вычитаемое в дополнительном коде, а затем произвести сложение. Таким образом, арифметические и логические возможности большинства микропроцессоров сильно ограничены по сравнению с большими ЭВМ.

Большинство МП имеет арифметическое устройство с простой шинной структурой. Обычно один операнд выбирается из аккумулятора, а другой — из регистра временного хранения; результат операции засылается в аккумулятор.

Многие МП имеют специальные блоки ПЗУ или специализированные схемы для решения некоторых типовых задач [десятичное (BCD) сложение — наиболее распространенная задача]; другие также имеют десятичное (BCD) вычитание, ввод с клавиатуры, управление дисплеем и т. д.

2.7. ПРИМЕРЫ АРХИТЕКТУРЫ МИКРОПРОЦЕССОРОВ

Структурная схема МП Intel 8080.

На рис. 2.70 показана структурная схема МП Intel 8080. В правой части рисунка расположен блок регистров и буферный регистр адреса. В центре рисунка расположен буфер шины данных, под ним регистр команд и дешифратор команд. В левой части рисунка показано арифметическо-логическое устройство, в состав которого входит комбинационная схема АЛУ, регистр признаков, схема десятичной коррекции, аккумулятор и регистр временного хранения. Устройство синхронизации и управления, а также работа шин ввода/вывода будут рассмотрены в гл. 7 и 8.

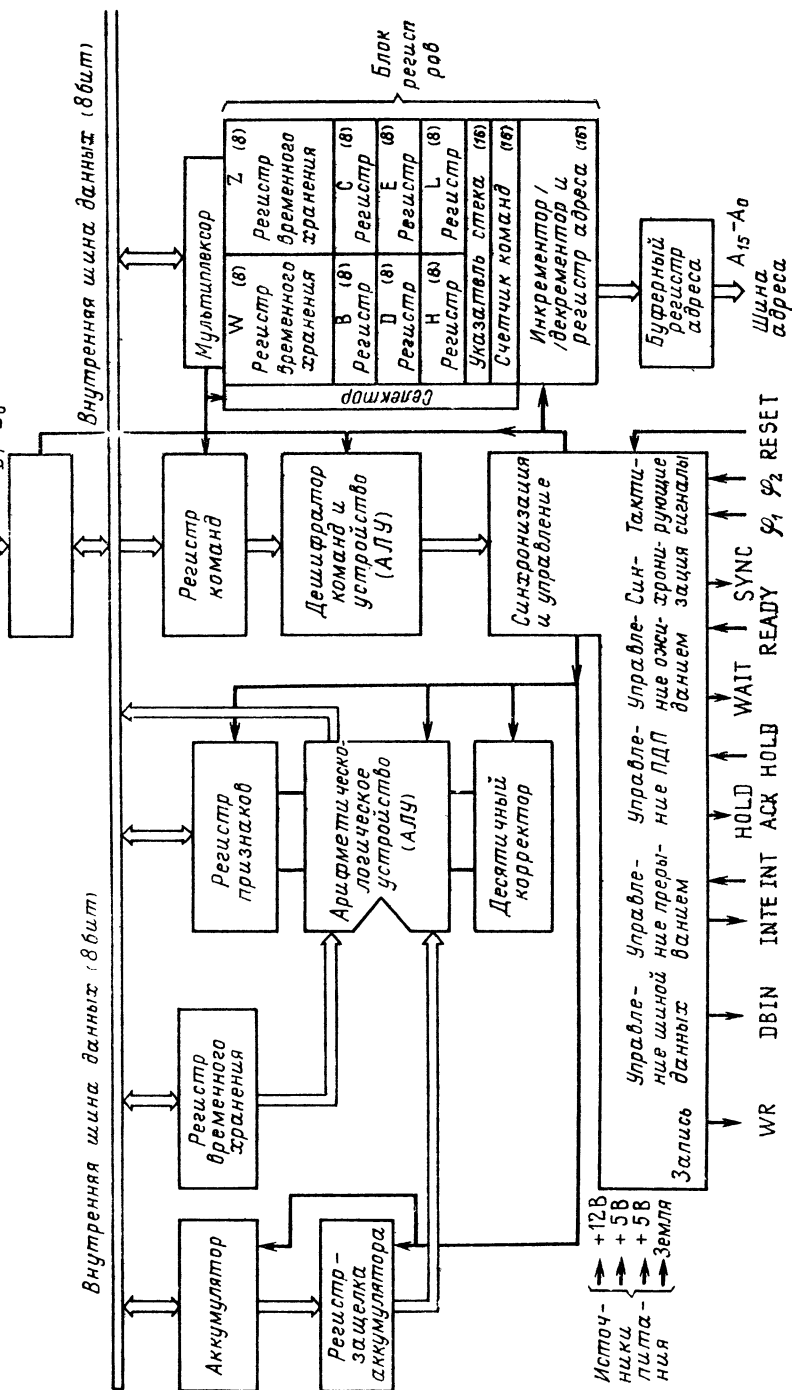
Регистры МП Intel 8080. Микропроцессор Intel 8080 содержит следующие регистры: шесть 8-битных регистров общего назначения с именами В, С, D, E, H, L, один 16-битный указатель стека, один 16-битный счетчик команд, два 8-битных регистра временного хранения (W и Z).

Шесть 8-битных регистров могут быть сгруппированы в три 16-битных регистровых пары (В и С, D и E, H и L), первый из названных в паре регистров содержит восемь старших значащих бит, и вся пара может носить его имя (В, D и H).

Регистровая пара H (регистры H и L) используется в качестве регистра начального адреса.

Ячейка памяти, адресуемая содержимым H и L, может быть использована в качестве регистра общего назначения, кроме тех случаев, когда центральному процессору необходим осуществить передачу данных в схему ИЛИ из этой регистровой пары. Другие пары регистров также могут использоваться в качестве регистра адреса, но только с целью загрузки или запоминания содержимого аккумулятора.

В блоке регистров имеется возможность выполнять простые арифметические операции. Специальными командами можно увеличивать или уменьшать на 1



содержимое 16-битного указателя стека и любой 16-разрядной регистровой пары. Два 8-битных регистра временного хранения и схема инкремента/декремента позволяют манипулировать с 16-битными адресами без участия аккумулятора и АЛУ. Счетчик команд также без участия АЛУ автоматически инкрементируется после каждого цикла выборки. Указатель стека автоматически инкрементируется после того, как байт данных выдается из стека в шину, и декрементируется перед тем, как байт данных будет передан из шины в стек. Стек загружается данными по содержимому регистра—указателя стека в порядке убывания адресов (от старшего к младшему).

Устройство управления МП Intel 8080 состоит из 8-битного регистра команд и дешифратора команд. Команда загружается из шины данных в регистр команд через буферную схему шины данных.

Арифметическое устройство МП Intel 8080 состоит из 8-битного АЛУ, схемы десятичного корректора, пяти триггеров признаков, аккумулятора (или регистра А) и регистра временного хранения данных.

Арифметическо-логическое устройство содержит схемы для выполнения операций сложения, вычитания, реализации четырех основных логических функций (И, ИЛИ, ИСКЛЮЧАЮЩЕЕ ИЛИ и НЕ) и операций сдвига. При выполнении арифметических и логических операций один операнд всегда извлекается из аккумулятора, а другой — из регистра временного хранения. Последний может загружаться содержимым любого из регистров общего назначения либо из ячейки памяти, адресуемой регистровой парой H и L. Он загружается через 8-битную внутреннюю шину, что является частью цикла выполнения команды.

Схема десятичной коррекции включается специальной командой и позволяет АЛУ, использующему двоичную арифметику, выполнять сложение по правилам десятичной двоично-кодированной арифметики.

В МП Intel 8080 имеются пять признаков: ПЕРЕНОС, НУЛЬ, ЗНАК (самый старший разряд) ЧЕТНОСТЬ (НЕЧЕТНОСТЬ), ВСПОМОГАТЕЛЬНЫЙ ПЕРЕНОС (или ПОЛУПЕРЕНОС).

Микропроцессор не имеет признака переполнения при получении дополнительного кода числа.

Структурная схема МП Motorola 6800

На рис. 2.21 показана структурная схема МП Motorola 6800. В правой части рисунка расположены регистры и арифметическо-логическое устройство, в левой части — устройство управления, буферный регистр адреса — сверху, буферный регистр данных — внизу.

Регистры МП Motorola 6800 Микропроцессор Motorola 6800 содержит следующие регистры: два 8-битных аккумулятора, один 16-битный индексный регистр, один 16-битный счетчик команд, один 16-битный указатель стека, один 8-битный регистр признаков два 8-битных регистра временного хранения данных.

Шестнадцатибитный индексный регистр служит для адресации памяти. К его содержимому может быть добавлено 8-битное смещение для получения исполнительного адреса.

Указатель стека содержит адрес очередной свободной ячейки в ОЗУ стека: в отличие от МП Intel 8080, МП Motorola 6800 уменьшает на 1 содержимое указателя стека после засылки байта данных в стек и увеличивает его на 1 перед выборкой байта данных из стека. Стек загружается данными в порядке убывания адресов.

Устройство управления МП Motorola 6800 состоит из 8-битного регистра команд и дешифратора команд. Из шины данных через буфер код команды загружается в регистр команд и поступает на дешифратор команд.

Арифметическое устройство МП Motorola 6800 состоит из 8-битного АЛУ и регистра шести признаков. Входные данные из шины данных передаются чаще прямо в АЛУ, реже — в регистр временного хранения.

Арифметическо-логическое устройство выполняет операции сложения, вычитания и четыре логических операции. Содержимое одного из аккумуляторов может быть использовано как один из операндов; результат возвращается в исходный аккумулятор.

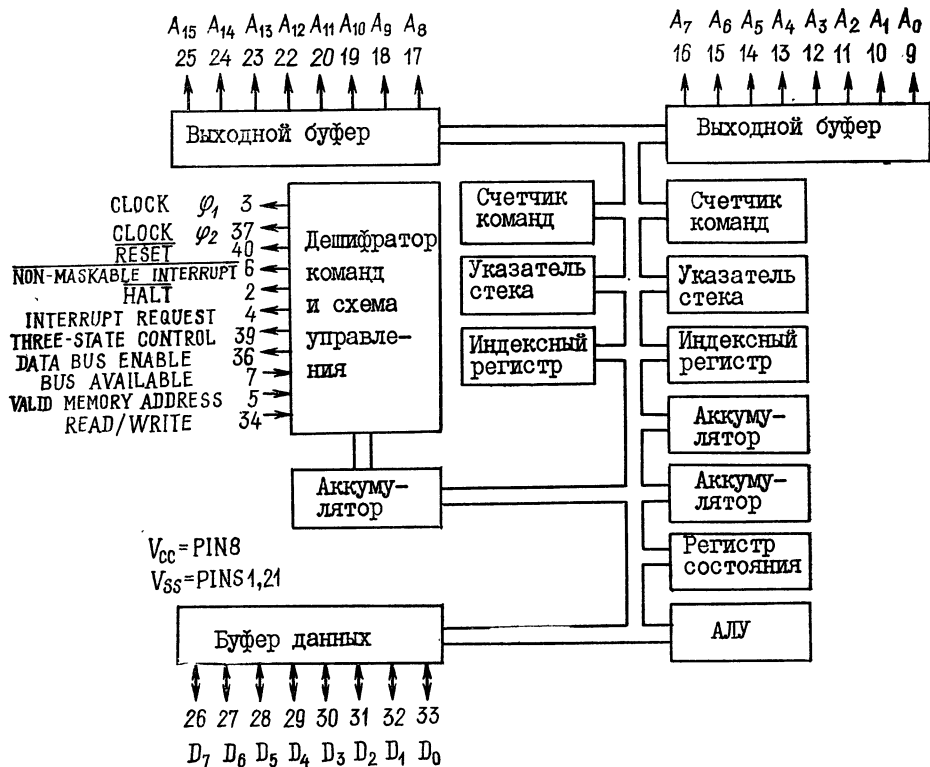


Рис. 2.21. Структурная схема МП Motorola 6800

В МП Motorola 6800 имеются шесть признаков: ПЕРЕНОС, НУЛЬ, МИНУС (отрицательный знак или старший значащий разряд), ПЕРЕПОЛНЕНИЕ (дополнительный код), ВСПОМОГАТЕЛЬНЫЙ ПЕРЕНОС, ПРЕРЫВАНИЕ (ЗАПРЕТ, т. е. прерывание невозможно, если в разряде — единица).

2.8. ВЫВОДЫ

Электронно-вычислительные машины имеют три основные составные части; центральный процессор, который определяет последовательность и производит преобразования данных;

запоминающее устройство, которое хранит данные и команды;

устройство ввода вывода, которое обеспечивает связь с внешним миром.

Центральный процессор состоит из блока регистров, арифметического устройства, дешифратора команд и схем синхронизации и управления.

Регистры могут выполнять различные операции: хранить и преобразовывать данные и команды, а также полные или частичные адреса. Арифметические устройства МП могут быть самыми разнообразными по сложности — от простых сумматоров до сложных арифметическо-логических устройств.

Стек представляет собой удобное средство для временного хранения последовательностей данных и адресов, но их довольно сложно использовать при написании, отладке и документировании программ.

ГЛАВА ТРЕТЬЯ

СИСТЕМЫ КОМАНД МИКРОПРОЦЕССОРОВ

В следующих четырех главах описано программное обеспечение микропроцессора: гл. 3 содержит описание системы команд, гл. 4 — описание ассемблеров, в гл. 5 излагаются вопросы программирования на языке ассемблера и в гл. 6 — развитие программного обеспечения. Главы 3 и 4 содержат основную информацию о форматах команд, методах адресации, типах команд, языковых уровнях и особенностях ассемблеров. Глава 5 содержит короткие программы на языке ассемблера для процессоров Intel 8080 и Motorola 6800. В гл. 6 показывается, как сформулировать задачу в виде программы и каким образом отладить, протестировать и оформить документацию на программу. Таким образом, начнем с общей информации, дадим примеры написания коротких программ и закончим обсуждение изложением методов программирования с использованием системных средств развития программного обеспечения.

В данной главе описываются системы команд микропроцессоров. В начале рассматриваются форматы команд и методы адресации, а затем различные группы команд и их применение. В главу включено описание особенностей системы команд микропроцессора и подробное знакомство с системами команд микропроцессоров Intel 8080 и Motorola 6800.

3.1. ФОРМАТЫ КОМАНД

Электронно-вычислительная машина должна получать данные из внешнего мира, обрабатывать их и отсылать результаты обратно. Она выполняет определенные операции в соответствии с определенными двоичными входными воздействиями. Эти входные воздействия, благодаря которым ЭВМ выполняет специфические операции, называются *командами*. Последовательность команд, которая заставляет ЭВМ выполнять эту задачу, называется *программой*, а совокупность команд, известных ЭВМ, называется ее *системой команд*.

Команды и данные в ЭВМ представлены в одинаковой форме, т. е. в виде двоичных чисел, которые ЭВМ заносит в память и передает по шине данных в ЦП. Таким образом, ЭВМ обрабатывает команды, которые представляют собой слова такой же длины, что и данные. Разница только в том, что команды ЭВМ засылает в регистр команд и в дешифратор команд, в то время как данные отправляются ею в регистры данных или в АЛУ (более подробно см. в гл. 7).

Каждая команда должна содержать значительную информацию. Она должна определить:

1) саму операцию. Часть команды, которая определяет эту операцию, называется *кодом операции*, или КОП;

2) источник данных. Эту информацию содержит поле адреса. Такие команды, как СЛОЖИТЬ или УМНОЖИТЬ (ADD или MULTIPLY), требуют двух операндов; команды, подобные СДВИНУТЬ

<i>Код операции</i>	<i>Адрес операнда 1</i>	<i>Адрес операнда 2</i>	<i>Адрес результата</i>	<i>Адрес следующей команды</i>
-------------------------	---------------------------------	---------------------------------	-----------------------------	--

Рис. 3.1. Формат команды

или ДОПОЛНИТЬ (SHIFT или COMPLEMENT), — только одного;

3) место назначения результата;

4) источник следующей команды.

Очевидно, что команда, которая содержит всю эту информацию (рис. 3.1), должна быть очень длинной. Например, если бы поле кода операции составляло 4 бита (что позволяет закодировать 2^4 , или 16, различных операций) и каждый из адресов составлял 12 бит (что позволяет адресовать 2^{12} , или 4 Кбайт памяти), то общая длина команды составила бы 40 бит. Очевидно, что с такой командой трудно было бы работать микро-ЭВМ, которая оперирует с 8- или 16-битными словами. Тем не менее многие программы требуют более 4 Кбайт памяти и более 16 различных команд. Таким образом, часть информации, в которой нуждается ЭВМ, должна быть задана неявно и не зависеть от особенностей конкретной команды.

Уменьшение формата команды

Существует множество методов уменьшения форматов команд ЭВМ. Среди наиболее употребительных следующие:

1. Использование программного счетчика, содержащего адрес команды. Центральный процессор увеличивает содержимое программного счетчика после каждого обращения к памяти программ и таким образом вызывает из программной памяти следующую команду с очередным, более старшим адресом.

2. Использование адресов источника и места назначения информации не явно выраженных, а неточных. Эти неявно выраженные адреса могут быть записаны в регистры или в ячейки памяти, адресуемые через регистры.

3. Использование адреса, по которому записывается результат, принадлежащего одному из исходных, т. е. когда результат записывается на место одного из операндов.

4. Ограничение адресации регистрами, или адресации по содержимому регистров, вместо использования полных адресов ячеек памяти.

С помощью перечисленных методов удается сократить формат команд ценой некоторой потери гибкости их использования. Необходимы специальные команды, с помощью которых программист может изменить неявно адресуемую информацию, предотвратить автоматическое получение следующего адреса и загрузить или заполнить содержимое неявно выраженных адресов. Такой прием оправдан, если число требуемых дополнительных специальных команд в программе мало, т. е. неявно адресуемая информация используется большую часть времени работы программы.

Например, использование программного счетчика представляет собой метод выборки последующей команды. Нет необходимости специфицировать адрес следующей команды, если он занимает в памяти ячейку с адресом, на 1 большим, но при этом необходимо использовать дополнительные команды (например, ПЕРЕХОД, УСЛОВНЫЙ ПЕРЕХОД, ПРОПУСК или ОСТАНОВ — JUMP, BRANCH, SKIP или HALT). Вопрос в том, может ли программист составлять программы так, чтобы ЭВМ большую часть времени работала последовательно. Ответ: такое написание программы не только возможно, но и желательно, так как делает программы более легкими для ввода и отладки.

Использование неявной адресации (с помощью аккумулятора или записи адреса в указатель стека или в адресный регистр) означает, что данной команде нет необходимости содержать этот адрес, но нужны специальные команды для загрузки или запоминания содержимого этих регистров. Может ли программист ограничиться небольшим числом специальных команд? Да, если программист размещает операнды в последовательных ячейках памяти, а программа извлекает операнды в их собственном порядке.

Если местом назначения информации сделать ее источник, то один адрес в команде можно опустить, но при этом прежнее содержимое источника будет уничтожаться. Может ли программист составлять программу так, чтобы реже возникала необходимость в запоминании промежуточных результатов? Да, в большинстве случаев промежуточные результаты вычислений не нужны.

Аналогично использование только адресации регистров означает, что дополнительные команды нужны для того, чтобы загрузить эти регистры и запомнить их содержимое. Здесь программист тоже может уменьшить число дополнительных команд в программе путем разумной организации последовательностей данных и операций и путем отказа от вывода промежуточных результатов. Использование стека также позволяет сократить формат команд, так как указатель стека автоматически изменяется при каждом обращении к стеку. Однако при этом программист должен упорядочить информацию в стеке. Отметим, что все эти методы уменьшают формат команд, но усложняют работу программиста.

Одноадресные команды

В ЭВМ с ограниченной разрядностью обычно используют аккумулятор в качестве источника и места назначения информации. В команде нет необходимости специально указывать, что должен быть использован аккумулятор. Типичная команда ADD B, означает, что требуется сложить содержимое регистра B с содержимым аккумулятора и результат поместить в аккумулятор. Такие команды называются одноадресными. Они производят действия, аналогичные операциям, выполняемым калькуляторами, в которых в качестве операндов используют самую последнюю вводимую информацию и содержимое аккумулятора, а результат операции вновь помещают в аккумуля-

Код операции	Адрес операнда
-----------------	-------------------

Рис. 3.2. Формат одноадресной команды

лятор. На рис. 3.2 показан формат одноадресной команды; очевидно, что такая команда может быть короткой.

Однако программы, содержащие одноадресные команды, требуют дополнительных команд, которые предварительно помещают информацию в аккумулятор, а затем размещают результаты в памяти или в регистрах общего назначения (РОН). Это вспомогательные команды ЗАГРУЗИТЬ АККУМУЛЯТОР и ЗАПОМНИТЬ СОДЕРЖИМОЕ АККУМУЛЯТОРА (LOAD ACCUMULATOR и STORE ACCUMULATOR). Большинство программ содержит много одноадресных команд, так как по каждой команде выполняется незначительное действие. Так, вычисление выражения

$$R = \frac{X + Y}{W + Z}$$

требует следующей последовательности команд:

Адрес	Команда	Адрес	Команда
START	ЗАГРУЗИТЬ W	START+4	СЛОЖИТЬ C Y
START+1	СЛОЖИТЬ C Z	START+5	РАЗДЕЛИТЬ НА TEMP 1
START+2	ЗАПОМНИТЬ TEMP 1	START+6	ЗАПОМНИТЬ R
START+3	ЗАГРУЗИТЬ X	START+7	ОСТАНОВ

Команда ЗАГРУЗИТЬ помещает содержимое ячейки памяти W в аккумулятор. Команда ЗАПОМНИТЬ R помещает содержимое аккумулятора в ячейку памяти R.

Хотя исключение адресов делает команды короче, само программирование усложняется. Программист должен следить за содержимым аккумулятора и тщательно документировать программы для того, чтобы избежать типичных ошибок, например таких, как не поместить исходную информацию в аккумулятор или не запомнить в памяти его содержимое после серии операций. Программы, содержащие одноадресные команды, трудны для понимания, так как каждая команда выполняет только часть операций решения задачи и цель последовательности команд часто неясна.

Использование одноадресных команд приводит к тому, что аккумулятор становится «узким местом» ЭВМ. На загрузку и запоминание содержимого аккумулятора может тратиться больше времени, чем на выполнение полезной работы. В некоторых ЭВМ, имеющих короткие команды, эту проблему решают путем использования не одного аккумулятора. При этом в действительности каждая команда содержит два адреса, но один адрес, определяющий источник и место назначения информации, может задавать один из двух, четырех или даже вось-

ми аккумуляторов. В команде требуется несколько бит для адресации аккумулятора. Motorola 6800, National PACE и Signetics 2650 — примеры микропроцессоров, имеющих более одного аккумулятора.

3.2. МЕТОДЫ АДРЕСАЦИИ

Адрес ячейки памяти или адрес регистра, с которыми оперирует команда, можно указать многими различными способами. В каждом конкретном случае метод адресации выбирается, исходя из следующих соображений:

1. Использование команды с возможно более коротким адресом. Эти команды требуют меньшего объема памяти и имеют меньшее время выборки.

2. Обеспечение простого доступа к возможно большему объему памяти. Очевидно, что это противоречит требованию предыдущего пункта. Однако многие программы временно используют сначала одну область памяти, а затем другую. Тогда появляется возможность легко определять любую часть памяти, используя укороченные адреса, осуществлять доступ к отдельным ячейкам этой части.

3. Возможность изменения содержимого адресной части без изменения команды. Одна и та же последовательность команды может тогда быть использована с целью обработки всех элементов массива, таблицы или строки. Такое изменение команд создает трудности в документировании и отладке программы. К тому же, если память программы постоянная, то команды не могут быть модифицированы. Такой подход позволил бы по одной и той же программе обрабатывать массивы или таблицы любого формата.

4. Обеспечение наиболее быстрой адресации. Предпочтительнее тот метод, который требует меньшего числа арифметических операций или дополнительных обращений к памяти.

5. Использование наиболее простого метода адресации.

Использование более сложных методов адресации приводит к появлению ошибок в программе.

Наиболее используемыми являются следующие методы адресации: прямая, косвенная, непосредственная, индексная, прямая регистровая, косвенная регистровая, стековая.

В ЭВМ эти методы адресации используются в различных сочетаниях.

Прямая адресация

Прямая адресация означает, что действительный адрес является частью команды. Одноадресная команда ADD 100 заставляет ЦП сложить содержимое ячейки памяти 100 с содержимым аккумулятора, т. е. $(A) = (A) + (100)$. Заметим, что ячейка памяти 100 не обязательно содержит число 100. На рис. 3.3 показана последовательность команд, использующих прямую адресацию, которая складывает содержимое ячеек памяти 100 и 101 и сумму помещает в ячейку памяти 102.

Прямая адресация не требует вычислений, легка для понимания и позволяет всегда выбрать любую ячейку памяти. Однако для прямой ад-

ресации необходим большой формат команды. Например, 8-битный процессор с памятью 64 Кбайт требует 16-битного адреса, который должен выбираться из двух 8-битных ячеек.

Формат команд с прямой адресацией может быть уменьшен путем разделения памяти на секции, или страницы. Тогда эту часть адреса можно держать в регистре страницы и только по команде включать ее в полный адрес. На рис. 3.4 показаны два примера прямой адресации с регистром страницы. Естественно, отдельные команды должны загружать адрес в регистр страницы, но если изменять номера страниц требуется редко, то это делает программу короче.

Если ограничить число страниц, к которому может обращаться ЦП, то регистр страниц можно исключить. Например, если ЦП может

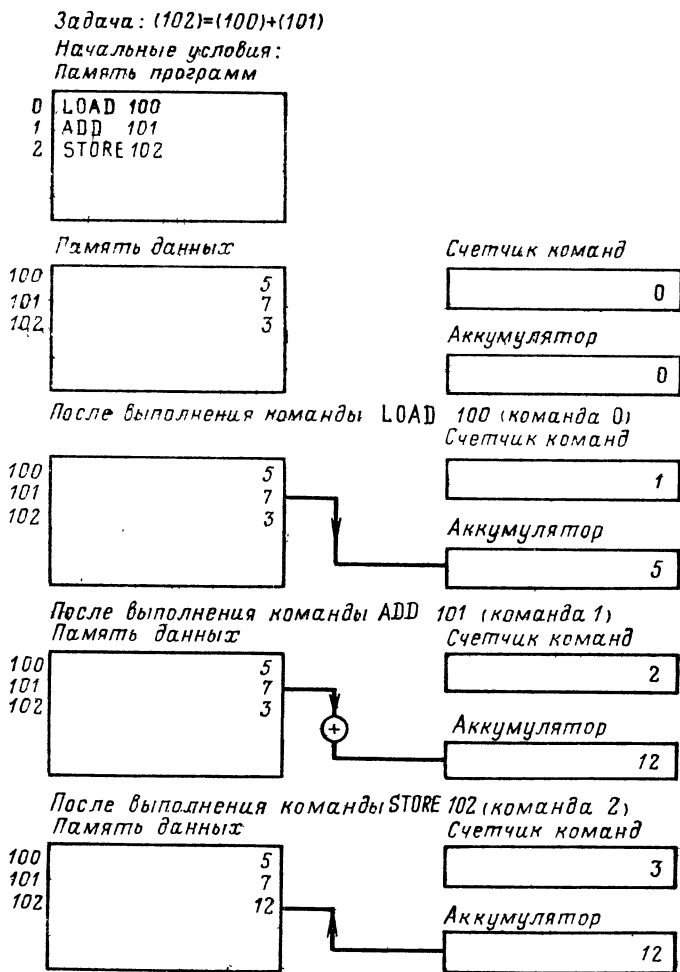


Рис. 3.3. Последовательность команд, использующих прямую адресацию

1. Команда

ADD 7

Регистр страницы

2

Результат: исполнительный адрес = P007, где
P – содержимое регистра страниц,
(аккумулятор) = (аккумулятор) + (2007)

2. Команда

STORE 36

Регистр страницы

4

Результат: исполнительный адрес = P036, где
P – содержимое регистра страниц,
(4036) = (аккумулятор)

Рис. 3.4. Прямая адресация в пределах страницы

обращаться только к ячейкам памяти на странице 0, то этот метод называется *адресацией к нулевой странице*. Для выборки из ячеек памяти на других страницах потребуются дополнительные методы адресации. Можно ограничить ЦП возможностью обращаться только к одной странице памяти. Этот метод называется *адресацией к текущей странице*. Оба эти метода могут быть использованы совместно путем добавления одного бита в каждую команду, как показано на рис. 3.5. Эти методы не требуют регистра страниц, но позволяют ЦП работать с одной или с двумя страницами памяти.

Прямой адресации не хватает гибкости, необходимой для обработки массивов данных. Окончательные команды могут обеспечивать доступ только к отдельным ячейкам памяти. Заметим, что прямая адресация требует дополнительных обращений к памяти, выдает прямой адрес в шину адреса и использует его для передачи данных.

Косвенная адресация

При косвенной адресации используется адрес адреса операнда, что предпочтительнее, чем собственно адрес как часть команды. Одноадресная команда ADD @ 100 (знак @ указывает на косвенную адресацию) заставляет ЦП сложить содержимое аккумулятора с содержимым ячейки памяти, адресуемой ячейкой памяти 100, т. е. $(A) = (A) +$

Команда	Бит страницы
Адрес 4013 ADD 35	0

Результат: исполнительный адрес = 0035,
(аккумулятор) = (аккумулятор) + (35)

Рис. 3.5. Прямая адресация, использующая текущую или нулевую страницу (бит страницы равен 0 при адресации к нулевой странице и равен 1 при адресации к текущей странице)

Команда	Бит страницы
Адрес 4013 STORE 27	1

Результат: исполнительный адрес = P027 = 4027, здесь команда адресует к странице 4.

+ (100)) (двойные скобки указывают на содержимое содержимого). По адресу, который является частью команды, содержится тоже адрес. Если, например, ячейка памяти 100 содержит 227, то команда ADD @ 100 аналогична команде ADD227. На рис. 3.6 показана последовательность команд, которая использует косвенную адресацию для сложения двух чисел и запоминания результата. Эта последовательность более сложна и требует больше времени для исполнения, чем последовательность на рис. 3.3. Однако она позволяет производить обращение к любой части памяти, а программа может простыми средствами изменять содержимое ячейки ОЗУ, которая содержит действительный адрес данных.

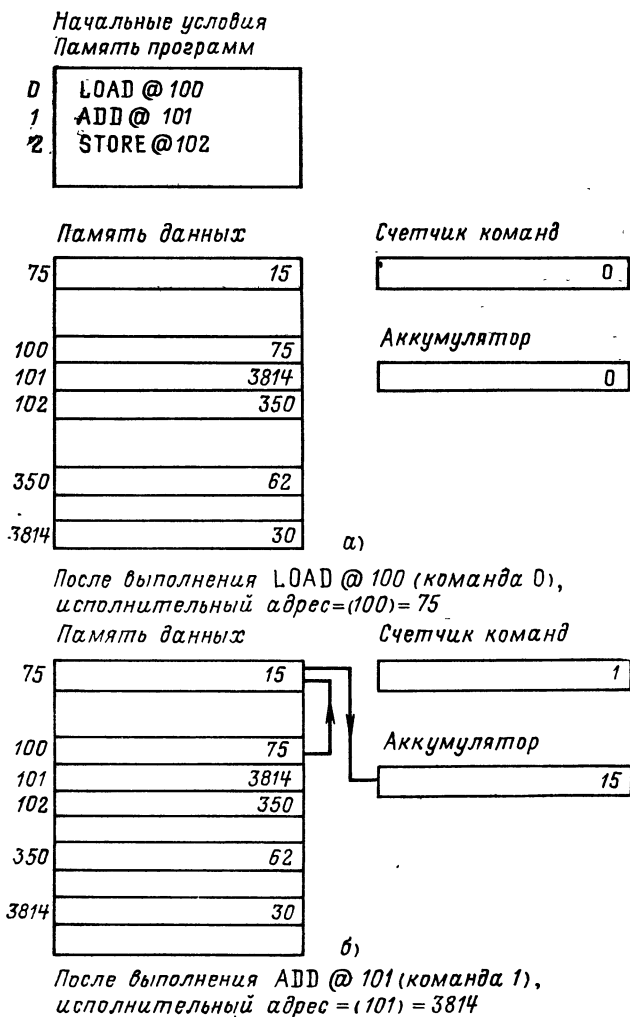
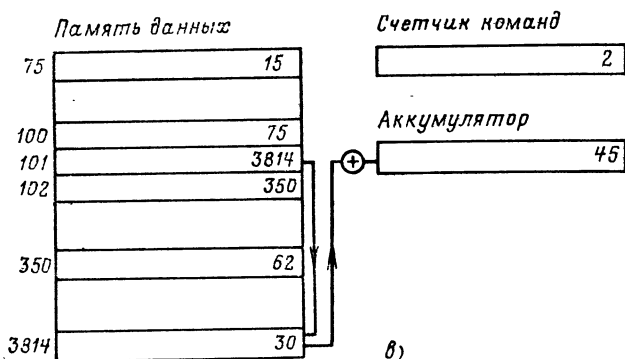


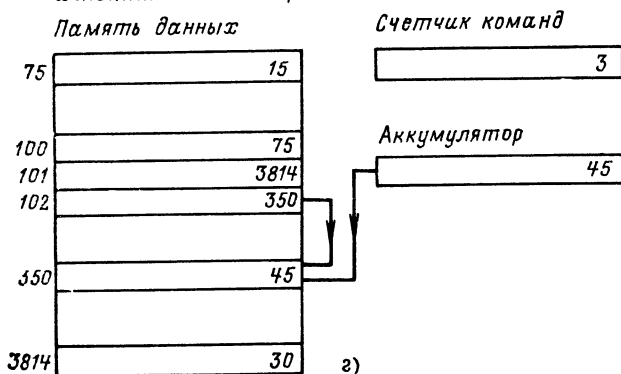
Рис. 3.6. Последовательность команд.

Косвенная адресация медленнее прямой. По сравнению с прямой адресацией ЦП должен выполнить дополнительное обращение к памяти. Ему необходимо сначала выбрать из памяти команду, содержащую косвенный адрес, затем использовать косвенный адрес для получения исполнительного адреса и только потом использовать его для обработки данных. Большинство программистов считают косвенную адресацию сложной и ведущей к ошибкам. Однако, несмотря на кажущуюся сложность, косвенная адресация является более гибкой, чем прямая. Например, программа может занести результат операции в память для его дальнейшего использования по команде ЗАПОМНИТЬ РЕЗУЛЬТАТ (STORE @ = RESULT).

Прежде чем запомнить последующий результат, к содержимому ячейки ОЗУ с символическим адресом RESULT добавляется 1. По той же самой команде можно занести очередной результат в следующую ячейку памяти. Очевидно, что при прямой адресации каждый раз требуется отдельная команда. Косвенная адресация особенно удобна для



После выполнения STORE @ 102 (команда 2), исполнительный адрес = 1002 = 350



Заметим, что программа может находиться в ПЗУ, в то время как адреса данных находятся в ОЗУ

использующих косвенную адресацию

подпрограмм, так как она позволяет одной и той же подпрограмме обрабатывать данные из различных массивов памяти. На рис. 3.7 приведены два варианта подпрограммы, определяющей наибольший элемент в массиве. Если используется прямая адресация, как показано на рис. 3.7,а, то необходимо разместить исходные данные массива в определенных ячейках памяти. Если используется косвенная адресация, как это показано на рис. 3.7,б, то необходимо только поместить начальный адрес массива данных в ячейках памяти (100) для косвенного адреса.

Непосредственная адресация

При непосредственной адресации действительные данные являются частью команды. Одноадресная команда `ADD #100` (знак `#` означает непосредственную адресацию) заставляет ЦП сложить число 100 с предыдущим содержимым аккумулятора. Команды с непосредственной адресацией используются для инициализации счетчиков, загрузки косвенных адресов и для введения констант, необходимых при вычислениях. Например, можно было использовать в подпрограмме сортировки (рис. 3.7,б) для просмотра массива, начиная с ячейки 5000, следующую последовательность команд:

```
LOAD #5000
STORE 100
CALL MAX
```

Можно было бы также пересчитать футы в дюймы, используя непосредственную адресацию:

```
LOAD FEET
MULTIPLY #12
STORE INCHES
```

Команды с непосредственной адресацией выполняются быстро. Непосредственная адресация легка для использования программистом. Однако она обладает наименьшей гибкостью по сравнению с другими

Задача: подпрограмме MAX определить наибольший элемент в массиве данных

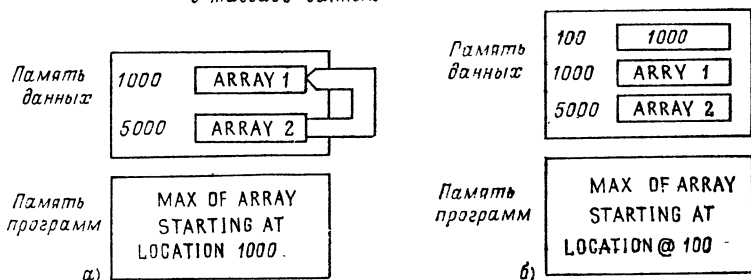


Рис. 3.7. Использование в подпрограммах косвенной адресации

а — прямая адресация (если требуется определить максимум массива 2, используя подпрограмму MAX, то необходимо ввести входной массив данных); б — косвенная адресация (если требуется определить максимум массива 2, используя подпрограмму MAX, то помещают 5000 в ячейку памяти 100)

методами адресации, так как фиксированы и адрес, и данные. Таким образом, непосредственная адресация применима, но не позволяет решить многих проблем программирования.

Индексная адресация

Индексная адресация означает, что для того, чтобы определить исполнительный адрес, ЦП складывает содержимое индексного регистра с адресом, содержащимся в команде. Одноадресная команда `LOAD 100, X` (знак, `X` означает индексацию) заставляет ЦП сначала вычислить исполнительный адрес путем сложения 100 с содержимым индексного регистра, а затем загрузить в аккумулятор исполнительный адрес. Если индексный регистр содержит число 35, то команда `LOAD 100, X` выполняет то же действие, что и команда `LOAD 135`. Индексная адресация используется главным образом для обработки массивов и таблиц. На рис. 3.8 показано, как вычислить среднее значение списка чисел, используя индексную адресацию. Рисунок 3.9 демонстрирует, каким образом можно получить таблицу квадратов чисел, используя индексную адресацию.

Индексная адресация медленнее прямой, так как ЦП должен выполнить дополнительные операции для получения исполнительного адреса. Индексная адресация более гибкая, чем прямая, так как по одной и той же команде, можно обрабатывать все элементы массива или таблицы. Большинство программистов считает индексную адресацию легкой в работе, так как стандартная система записи программ позволяет оперировать с массивами по индексам, X_i — это i -й элемент мас-

Шаг 1:

`LOAD # 0`

`LOAD INDEX REGISTER # 0`

Шаг 2:

`ADD NUMBER, X`

Шаг 3:

`INCREMENT INDEX REGISTER BY 1`

Шаг 4:

Если не желательно использовать команду сложения для всех чисел, то возвращаются к шагу 2

Шаг 5:

`DIVIDE # N` (N — число элементов)

`STORE AVG`

Примечание. Символ `#` означает индексную адресацию.

Рис. 3.8. Использование индексной адресации для расчета среднего значения

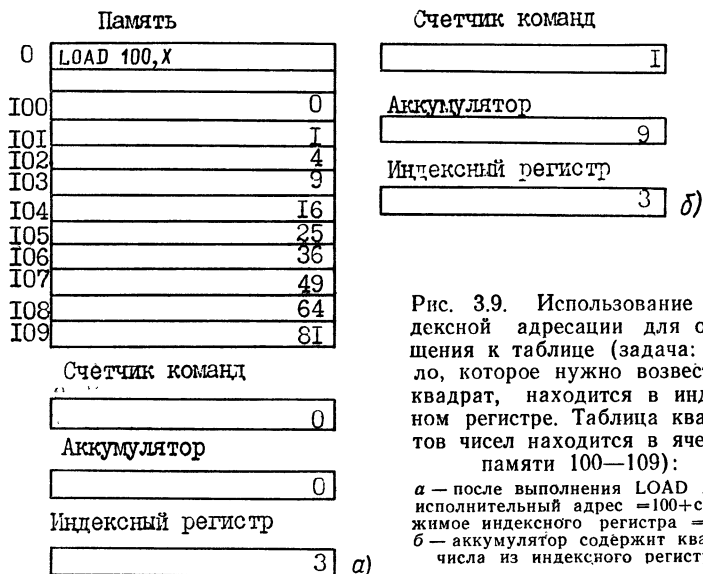


Рис. 3.9. Использование индексной адресации для обращения к таблице (задача: число, которое нужно возвести в квадрат, находится в индексном регистре. Таблица квадратов чисел находится в ячейках памяти 100—109):

a — после выполнения LOAD 100, X исполнительный адрес = 100 + содержимое индексного регистра = 103;
б — аккумулятор содержит квадрат числа из индексного регистра

сива X. Чтобы обработать X-массив, используя индексную адресацию, исходный (базовый) адрес помещают в команду, а индекс *i* — в регистр индексов.

Относительная индексация

При относительной индексации ЦП для определения исполнительного адреса прибавляет содержимое счетчика команд к адресу, содержащемуся в команде. Одноадресная команда LOAD * + 100 (знак * или \$ означает текущее содержимое счетчика команд) заставляет ЦП вычислить исполнительный адрес путем прибавления числа 100 к со-

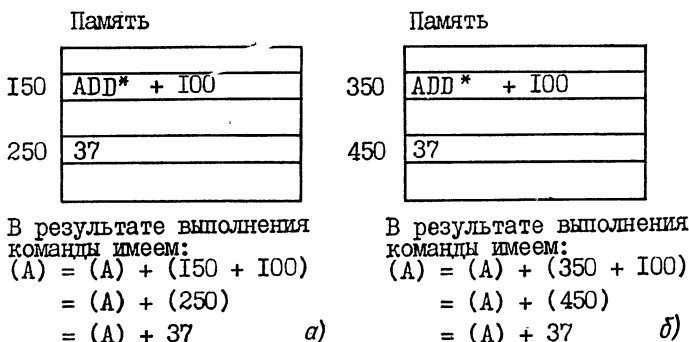


Рис. 3.10. Использование относительной адресации для операции перемещения
a — начальные условия; *б* — после перемещения программы и данных в памяти вперед на 200 ячеек

держимому счетчика команд, а затем загрузить в аккумулятор исполнительный адрес. Если бы команда `LOAD*+100` находилась в ячейке памяти с адресом 2000, то ее действие было бы равноценно действию команды `LOAD2100`. Как следует из названия, относительная адресация определяет скорее относительный, чем абсолютный адрес текущей команды.

Относительная адресация позволяет перемещать программы в различные области памяти. На рис. 3.10, например, показано, как можно передвинуть в памяти команду `ADD` и данные на 200 ячеек и при этом сохранить их относительное расположение. Такая программа называется перемещаемой. Перемещаемые программы удобны, так как они могут находиться в любой возможной области памяти и не мешать другим программам.

Кроме того, относительная адресация позволяет использовать более короткие адреса, если исполнительный адрес содержится в текущей ячейке программы. Часго используют относительные адреса в командах переходов, так как большинство из них имеют узкий формат.

Прямая регистровая адресация

Это та же самая процедура, что и прямая адресация, за исключением того, что адресуется не ячейка памяти, а регистр. Типичной одноадресной командой, использующей прямую регистровую адресацию, является команда `ADDR1`, которая складывает содержимое регистра общего назначения 1 (`POH1`) с содержимым аккумулятора. Некоторые ЭВМ должны использовать прямую регистровую адресацию для выполнения арифметических и логических операций.

Очевидно, что ей присущи достоинства и недостатки обычной прямой адресации. Регистровая прямая адресация даже быстрее, чем прямая адресация, так как ЦП нет необходимости извлекать данные из памяти. К тому же команды могут иметь более короткий формат, так как регистров меньше, чем ячеек памяти. Таким образом, прямая регистровая адресация — самый быстрый способ адресации, пригодный для большинства ЭВМ.

Ее очевидным недостатком является то, что сначала необходимо загрузить регистр общего назначения операндом, а после выполнения операции занести его обратно в память. Такой процесс требует дополнительных затрат времени и не пригоден, если программа использует содержимое регистра только 1 раз. Например, если необходимо сложить содержимое ячейки памяти 35 с содержимым аккумулятора, одиночная команда `ADD 35` будет выполнена быстрее, чем последовательность

```
LOAD R1, 35  
ADD R1
```

По команде `LOAD R1, 35` содержимое ячейки памяти 35 засылается в регистр 1. Однако, если необходимо сложить содержимое ячейки памяти 35 с сотней различных чисел, то число 35 следует загрузить в регистр 1 только 1 раз. Команда `ADD R1` будет выполнена гораздо

быстрее, чем команда ADD35, так как процессору не придется каждый раз извлекать из памяти содержимое ячейки 35. Таким образом, прямая регистровая адресация имеет преимущества, когда программа использует многократно одни и те же данные. Естественно, при этом необходимо внимательно произвести распределение ограниченного числа регистров.

Косвенная регистровая адресация

Это то же самое, что и косвенная адресация, за исключением того, что адрес команды находится не в ячейке памяти, а в регистре. Типичной одноадресной командой, использующей косвенную адресацию, является команда ADD @ R1, которая складывает содержимое ячейки памяти, адресуемой содержимым регистра 1, с содержимым аккумулятора. Если регистр 1 содержит число 1200, команда ADD @ R1 аналогична команде ADD 1200.

Этому типу адресации свойственны достоинства и недостатки обычной косвенной адресации. Регистровая косвенная адресация более быстрая, чем простая косвенная, так как у процессора нет необходимости выбирать из памяти исполнительный адрес. Кроме того, адрес регистра короче адреса памяти. Однако сначала требуется загрузить регистр, и поэтому косвенная адресация предпочтительнее тогда, когда программа многократно использует один и тот же адрес ячейки памяти или соединение адреса. Например, передают массив данных из одной области памяти в другую и модифицируют содержимое адресных регистров следующие команды:

```
LOAD @ R1  
STORE @ R2  
INCREMENT R1  
INCREMENT R2
```

Для выполнения этой программы предварительно необходимо загрузить регистры 1 и 2 начальными адресами областей памяти, участвующих в обмене. Регистровая косвенная адресация может заменять простую косвенную адресацию, что иллюстрируется на рис. 3.11. Первый вызов подпрограммы отыскивает наибольший элемент массива,

Память программы

```
LOAD R1, # 1000  
CALL SUBROUTINE MAX  
  
LOAD R1, # 6000  
CALL SUBROUTINE MAX
```

Подпрограмма MAX определяет наибольший элемент в массиве, который начинается с ячейки, адресуемой содержимым регистра 1

Рис. 3.11. Использование косвенной регистровой адресации в подпрограммах

начинающегося с ячейки памяти 100, тогда как второй вызов подпрограммы определяет наибольший элемент массива, начинающегося с ячейки памяти 6000.

Стековая адресация

Стековая адресация означает, что содержимое специального регистра—указателя стека является адресом данных. Команды, в которых используется стековая адресация, как правило, имеют меньший формат, чем команды с любым другим методом обращения к памяти. Этот метод адресации может быть использован вместо косвенной или индексной адресации, но преимущество его состоит в том, что адрес операнда содержится в самом коде операции. Кроме того, стековая адресация не требует от ЦП выполнения операции сложения для получения исполнительного адреса. Так как добавление новых данных к хранимым в стеке не разрушает хранимой информации, а только скрывает ее, то этот метод позволяет использовать вложенные программы, которые могут выполняться, когда выполнение текущей программы прервано или приостановлено, или рекурсивные программы, которые могут многократно использоваться или многократно обращаться к подпрограммам.

Комбинированная адресация

Каждый из методов адресации имеет свои достоинства и недостатки. Некоторые программы используют только один метод адресации. Очевидно, что использование тех или иных методов адресации зависит от целевых функций аргументов программы. Можно использовать индексную, стековую, прямую регистровую или косвенную регистровую адресацию в тех частях программы, к которым ЭВМ обращается неоднократно; однако прямая, косвенная или непосредственная адресация необходима для того, чтобы загрузить счетчики, регистры общего назначения, адресные регистры и регистр — указатель стека.

Если в ЭВМ реализуется несколько методов адресации, каждая команда должна точно определить метод, который ею используется (рис. 3.12). Очевидно, что в большинстве команд необходимо иметь поле для кодирования метода адресации. Некоторые ЭВМ допускают использование только определенных методов адресации с определенными командами (например, относительная адресация используется только с командами перехода).

Следует отметить важность того, насколько легко и удобно использовать программисту тот или иной способ адресации. Электронно-вычислительная машина может отлично работать с любым методом адре-

Рис. 3.12. Формат одноадресной команды с указанием метода адресации

Поле кода операции	Поле кода метода адресации	Поле адреса
-----------------------	----------------------------------	----------------

сации, но большинство программистов считает, что, пользуясь определенными методами, они делают больше ошибок в программе. Прямая, непосредственная и индексная адресации — наиболее легкие для использования и наиболее популярные. Косвенная и стековая адресации приводят в замешательство многих программистов. Оба этих метода могут использоваться в программе, но программисту (особенно начинающему) следует обращаться с ними с осторожностью. Отношение человека к программному обеспечению ЭВМ представляет собой предмет, требующий серьезного внимания.

3.3. ТИПЫ КОМАНД

Далее рассматриваются коды операций, которые определяют действия ЭВМ при выполнении команд. Система команд разбивается на несколько групп по функциональному признаку, а затем описываются формат команд и их работы.

Группы команд

Система команд ЭВМ может быть разбита на группы различными способами. Один из возможных способов разбиения системы команд на четыре основные группы был предложен Кушманом¹. Это следующие группы:

- 1) команды преобразования данных;
- 2) команды передачи данных;
- 3) команды управления программой;
- 4) команды управления состояниями.

Команды преобразования данных выполняют операции обработки данных. В основном они используются в арифметическо-логическом устройстве ЭВМ и включают в себя следующие команды:

арифметических операций; логических операций; операций сдвига; операций сравнения; специальных.

Команды передачи данных пересылают данные из одной части ЭВМ в другую без изменения этих данных и включают в себя следующие команды: обращения к памяти; ввода-вывода; внутрипроцессорного обмена; оперирующие со стеком.

Команды управления программой передают управление программой из одной области памяти в другую. Они изменяют состояние счетчика команд так, что команды выполняются не в естественной последовательности, и включают в себя следующие команды: безусловного перехода; условного перехода; оперирующие с подпрограммами; останов и отсутствие операций.

Команды управления состояниями изменяют значения признаков ЭВМ без изменения данных или порядка, в котором выполняются команды. Таким образом, эти команды скорее влияют на функции управления, чем на функции обработки данных. Они составляют незначительную часть большинства программ.

¹R. H. Cushman. Microprocessor Instruments Sets: The Vocabulary of Programming, EDN, vol. 20, № 6, March 20, 1975, p. 35—41.

Команды преобразования данных

Команды арифметических операций. В табл. 3.1 перечислены команды арифметических операций в одноадресной ЭВМ, где А — содержимое аккумулятора, а М — содержимое адресуемой ячейки памяти.

Наиболее простая арифметическая операция — это сложение в двоичном коде. Сложение с повышенной точностью требует нескольких операций суммирования с участием признака переноса. Команда **ADD WITH CARRY** используется именно в этой ситуации. Так, для сложения чисел 4327 и 5096 в ЭВМ, которая за цикл команды может сложить только две значащие цифры, используется команда **ADD** для сложения двух младших значащих цифр:

$$\begin{array}{r} + 27 \\ 96 \\ \hline 1 \quad 23 \end{array}$$

CARRY RESULT

Затем используется команда **ADD WITH CARRY** для сложения двух старших значащих цифр и признака переноса:

$$\begin{array}{r} + 43 \\ + 50 \\ \hline \text{CARRY} \\ + 43 \\ + 50 \\ \hline 1 \\ \hline 94 \end{array}$$

RESULT

Таким образом, общий результат равен 9423.

Другой простой арифметической операцией является двоичное вычитание в двух видах. Как и при сложении, специальная команда (**SUBTRACT WITH CARRY**) необходима для выполнения вычитания с повышенной точностью.

Таблица 3.1. Команды арифметических операций

Название команды		Операция
ADD	(СЛОЖИТЬ)	$A = A + M$
ADD WITH CARRY	(СЛОЖИТЬ С ПЕРЕНОСОМ)	$A = A + M + \text{CARRY}$
SUBTRACT	(ВЫЧЕСТЬ)	$A = A - M$
SUBTRACT WITH CARRY	(ВЫЧЕСТЬ С ЗАЕМОМ)	$A = A - M - \text{CARRY}$
INCREMENT	(УВЕЛИЧИТЬ НА 1)	$M = M + 1$
DECREMENT	(УМЕНЬШИТЬ НА 1)	$M = M - 1$
MULTIPLY	(УМНОЖИТЬ)	$A = A \times M$
DIVIDE	(РАЗДЕЛИТЬ)	$A = A \div M$

Большинство ЭВМ имеют также команды INCREMENT и DECREMENT. Эти команды используются для добавления 1 или вычитания 1 из содержимого счетчиков, регистров индексов и регистров косвенного адреса. Команды INCREMENT и DECREMENT короче по формату и быстрее по выполнению, чем команды ADD и SUBTRACT. Кроме того, команды INCREMENT и DECREMENT не воздействуют на признак переноса, и, таким образом, они могут быть использованы в циклах, осуществляющих арифметические операции с повышенной точностью. Последовательность команд

```
ADD WITH CARRY @ R1
INCREMENT R1
DECREMENT R2
```

прибавляет 1 к косвенному адресу, находящемуся в регистре 1, и вычитает 1 из счетчика, содержимое которого записано в регистре 2, без воздействия на признак переноса, полученный при сложении. В этом случае можно использовать значение признака переноса при следующем повторении последовательности команд.

Некоторые ЭВМ имеют команды MULTYPLY и DIVIDE. Такие команды требуют операций с двойным словом, так как в результате умножения двух однословных чисел получается слово удвоенной разрядности и, наоборот, при делении чисел делимое должно быть длиной в два слова, если делитель, частное и остаток — все слова обычной длины. Если ЭВМ не имеет команд MULTYPLY и DIVIDE, то эти команды можно выполнить путем многократного повторения операций сложения и вычитания.

Команды логических операций. В табл. 3.2 представлено несколько команд простых логических операций. Наиболее простая логическая операция — логическое И. Логическое И используется для того, чтобы исследовать (проанализировать) части слова. Эта операция носит название операции маскирования, а число, которое вместе с данными участвует в операции логического И, называется маской, так как его задача состоит в том, чтобы исключить те биты, которые не должны участвовать в дальнейших операциях. Например, если 8-битное слово содержит два десятичных двоично-кодированных числа, можно отделить последние значащие разряды, осуществив операцию логического И при помощи двоичной маски 00001111. Таким образом,

Таблица 3.2. Команды логических операций

Название команды		Операция
AND	(И)	$A = A \cdot M$
OR	(ИЛИ)	$A = A + M$
EXCLUSIVE OR	(ИСКЛЮЧАЮЩЕЕ ИЛИ)	$A = A \oplus M$
COMPLEMENT	(ИНВЕРТИРОВАТЬ)	$M = M$

Таблица 3.3 Команды логического сдвига

Название команды		Операция
LOGICAL SHIFT	(ЛОГИЧЕСКИЙ СДВИГ)	М сдвигается на 1 бит, опустевший бит обнуляется
ARITHMETIC SHIFT	(АРИФМЕТИЧЕСКИЙ СДВИГ)	М сдвигается на 1 бит, знак бита сохраняется, опустевший бит обнуляется
ROTATE	(ЦИКЛИЧЕСКИЙ СДВИГ)	М циклически сдвигается на 1 бит, старший и младший значащие разряды расположены рядом
ROTATE WITH CARRY	(ЦИКЛИЧЕСКИЙ СДВИГ С ПЕРЕНОСОМ)	М циклически сдвигается на 1 бит через разряд ПЕРЕНОС; старший и младший значащие разряды расположены рядом

если первоначальное слово было 10010110, то в результате операции имеем:

$$\begin{array}{r}
 10010110 \\
 00001111 \\
 \hline
 00000110
 \end{array}$$

Старшие значащие разряды можно было бы отделить путем операции логического И над исходным словом двоичной маской 11110000. Команда логическое И может также очищать разряды. Например, чтобы очистить пятый бит слова, производим операцию логического И над этим словом и маской, которая в пятом бите имеет 0, а в остальных — 1. В результате пятый бит этого слова будет обнулен, в то время как остальные биты слова останутся без изменения.

Более сложная операция — логическое ИЛИ (иногда ее называют ВКЛЮЧАЮЩЕЕ ИЛИ). Команда логического ИЛИ может устанавливать (включать) биты. Например, чтобы установить пятый бит слова, производят операцию логического ИЛИ над словом и установкой, которая имеет 1 в пятом бите, а в остальных битах — 0. В результате в пятом бите слова будет 1; остальные биты останутся без изменения.

Другие логические операции сложнее. Операция ИСКЛЮЧАЮЩЕЕ ИЛИ используется для формирования контрольной суммы, а операция ДОПОЛНЕНИЕ — для инвертирования двоичного кода слова.

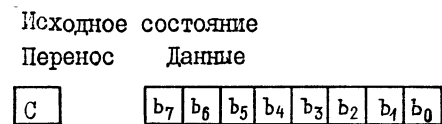
Команды операций сдвига. В табл. 3.3 дан перечень и описание команд операций сдвига. Сдвиги используются для преобразования данных, нормализации масштабирования, подготовки данных для запоминания или обработки, исключения отдельных бит или частей слов,

при выполнении операций умножения или деления, а также для выделения и анализа отдельных бит.

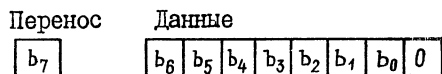
Команда ЛОГИЧЕСКИЙ СДВИГ сдвигает слово и обнуляет освободившиеся биты. На рис. 3.13 показано, как логический сдвиг на один разряд воздействует на 8-битное слово, если старший бит слова перемещается в разряд ПЕРЕНОС.

Если, например, старший знак десятичной цифры в 8-битном слове необходимо поместить в младшие разряды, то это можно сделать четырехкратным логическим сдвигом вправо. Тогда, если первоначально код слова был равен 10010110, то после сдвига он будет равен 00001001, т. е. старшие четыре двоичных разряда переместились на место младших.

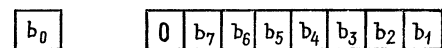
Команда ARITHMETIC SHIFT (АРИФМЕТИЧЕСКИЙ СДВИГ) обнуляет освободившийся разряд справа (если сдвиг влево), но сохраняет слева содержимое знакового разряда. На рис. 3.14 показано выполнение команд арифметического сдвига 8-битного числа. При арифметическом сдвиге вправо значение знакового разряда записывается в соседний разряд. Эта процедура называется расширением числа и используется для нормализации и масштабирования чисел в дополни-



После логического сдвига влево

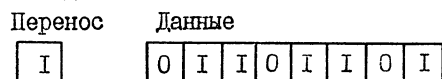


После логического сдвига вправо

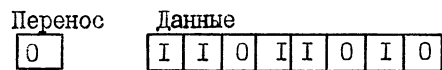


Пример

Исходное состояние



После логического сдвига влево



После логического сдвига вправо

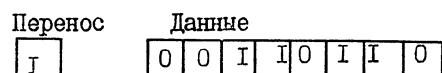
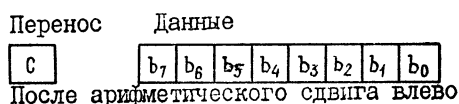
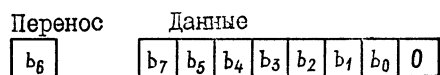


Рис. 3.13. Команда логического сдвига

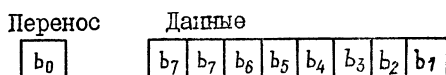
Исходное состояние



После арифметического сдвига влево

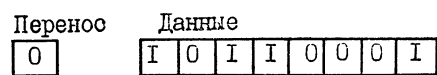


После арифметического сдвига вправо

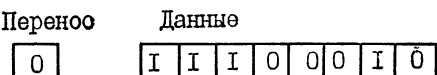


Пример

Исходное состояние



После арифметического сдвига влево



После арифметического сдвига вправо

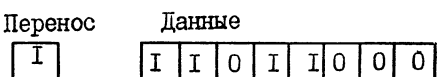


Рис. 3.14. Команда арифметического сдвига

Рис. 3.15. Команда циклического сдвига

тельном коде. Арифметический сдвиг влево производит умножение на два числа в дополнительном коде, а арифметический сдвиг вправо — деление числа на два.

Команда ROTATE (ЦИКЛИЧЕСКИЙ СДВИГ) (рис. 3.15) интерпретирует данные как содержимое кольцевого сдвигающего регистра, в котором старший и младший разряды соединены между собой. Эта команда (иногда ее называют CIRCULAR SHIFT) сохраняет значения разрядов слова, но изменяет положение разрядов и, как правило, содержимое разряда ПЕРЕНОС.

Команда ROTATE WITH CARRY (ЦИКЛИЧЕСКИЙ СДВИГ С ПЕРЕНОСОМ) аналогичен команде ROTATE с той разницей, что в состав кольцевого сдвигающего регистра включен разряд ПЕРЕНОС.

Следовательно, при сдвиге все разряды исходного слова сохраняются, за исключением разряда ПЕРЕНОС. Эта команда используется при выполнении операций умножения и деления на ЭВМ, которые не имеют соответствующих команд.

Команды операций сравнения. Операции сравнения позволяют сравнить или проверить данные без их изменения. Такие операции воздействуют на разряд признаков, но не запоминают результат, что позволяет использовать их при условных переходах. В табл. 3.4 приведены наиболее часто используемые команды операций сравнения и операции, которые они выполняют в одноадресной ЭВМ. Команды срав-

Исходное состояние

Перенос

С

Данные

b₇ b₆ b₅ b₄ b₃ b₂ b₁ b₀

После циклического сдвига влево

Перенос

Данные

b₇

b₆ b₅ b₄ b₃ b₂ b₁ b₀ b₇

После циклического сдвига вправо

Перенос

b₀

b₀ b₇ b₆ b₅ b₄ b₃ b₂ b₁

Пример

Исходное состояние

Перенос

Данные

0

1 1 0 0 1 0 0 1

После циклического сдвига влево

Перенос

Данные

1

1 0 0 1 0 0 1 1

После циклического сдвига вправо

Перенос

Данные

1

1 1 1 0 0 1 0 0

Таблица 3.4. Команды операций сравнения

Название команды		Операция
COMPARE	(СРАВНИТЬ)	РАЗ- Вычислить A—M Вычислить A · M
BIT TEST	(ПРОВЕРИТЬ РЯД)	
TEST	(ПРОВЕРИТЬ)	Вычислить M—0 Отыскать элемент в ряду подобных
SCAN	(СКАНИРОВАТЬ)	

нения позволяют программе многократно выполнять операции сравнения без перегрузки аккумулятора. Обычно эти команды используются для обнаружения первого значащего символа в строке или для интерпретации командных символов.

По команде COMPARE (СПАВНИТЬ) выполняется операция вычитания, но результат не фиксируется в аккумуляторе. Чтобы найти символ пробела в строке (шестнадцатиричное число 40 в коде EBCDIC), используют последовательность команд

```
LOAD # 40  
COMPARE BASE, X
```

Вслед за этим можно увеличить на 1 содержимое индексного регистра и снова выполнить команду COMPARE, чтобы проверить следующий символ.

По команде BIT TEST (ПРОВЕРИТЬ РАЗРЯД) выполняется операция логического И без запоминания результата. По этой команде можно проверить содержимое отдельных бит или группы бит в слове без запоминания промежуточных результатов (без создания копий слов после каждой операции). Команда TEST (ПРОВЕРИТЬ) устанавливает признаки в соответствии с содержимым адресуемых ячеек памяти без выполнения каких-либо вычислений. При использовании этой команды можно анализировать данные (устанавливать признаки) без перемещения данных и без изменения содержимого регистров.

Команда SCAN (СКАНИРОВАТЬ), используемая в процессе редактирования текста, по задаваемому признаку отыскивает символ в строке символов и, если поиск успешен, фиксирует на регистре адрес соответствующей ячейки памяти.

Специальные команды. Многие ЭВМ в своих системах команд имеют специальные команды, такие как команды десятичной арифметики или команды сканирования клавиатуры (в калькуляторах), команды для обработки чисел с плавающей запятой или команды контроля и паритета (в системах связи), команды сортировки или объединения (в системах обработки коммерческой информации). Наличие специальных команд может быть существенным фактором при выборе типа ЭВМ для конкретного применения. Электронно-вычислительной машине без специальных команд для решения тех же самых задач требуется целая серия команд.

Команды передачи данных

Команды обращения к памяти осуществляют обмен данными между регистрами и ячейками памяти. Содержимое источника информации не изменяется. Наиболее простые команды обращения памяти — это LOAD и STORE. Специальные команды CLEAR и SET (ОЧИСТИТЬ и УСТАНОВИТЬ) часто используются для обнуления разряда признаков, регистра или ячейки памяти, а также для засылки единицы в разряд признака.

Команды ввода-вывода. Операции ввода-вывода аналогичны операциям обращения к памяти, за исключением того, что источником или

местом назначения информации здесь являются порты ввода или вывода информации, а не ячейки памяти. Некоторые ЭВМ обращаются к портам ввода-вывода как к ячейкам памяти и не имеют специальных команд ввода-вывода.

Самая простая команда ввода — READ или INPUT, которая передает одно слово данных из входного порта в регистр. Команда WRITE или OUTPUT передает одно слово данных регистра в выходной порт. Передача более одного слова требует последовательности команд ввода-вывода для того, чтобы запомнить вводимые данные или извлечь из памяти выводимые.

Например, выдать через аккумулятор в периферийное устройство последовательность слов может следующая программа:

```
LOAD BUFFER, X  
WRITE DEVICE  
INCREMENT INDEX REGISTER
```

Каждый раз, когда процессор выполняет эту последовательность команд, он посылает одно слово данных в периферийное устройство и готовится выбрать следующее слово из буфера в памяти. Программа должна считать число переданных слов или использовать специальный маркер для индикации конца буфера. Некоторые ЭВМ используют команды BLOCK TRANSFER, по которым автоматически решаются все эти задачи. При этом необходимые параметры могут содержаться в команде.

Команды внутрипроцессорного обмена используются для пересылки данных из одного регистра в другой. Такие операции чаще всего используются для того, чтобы загрузить аккумулятор или индексный регистр данными из регистров общего назначения или чтобы запомнить содержимое аккумулятора в одном из регистров общего назначения.

Команды, оперирующие со стеком, осуществляют обмен данными между стеком и регистрами процессора (см. § 2.5). Команда PUSH помещает содержимое регистра в стек; команды POP или PULL пересылают верхний элемент данных из стека в регистр. Эти команды изменяют содержимое регистра — указателя стека.

Команды управления программой

Команды безусловного перехода (JUMP или BRANCH) изменяют обычную последовательность выполнения команд. Команда JUMP 150 засылает число 150 в счетчик команд; следующую команду процессор будет выбирать из этой ячейки. Эти команды воздействуют только на счетчик команд. Команда SKIP позволяет пропустить очередную команду в последовательности команд.

Некоторые ЭВМ используют в качестве счетчика команд один из регистров общего назначения. В этом случае передача данных в память, межрегистровая передача или операции со стеком могут изменять содержимое счетчика команд. Например, если регистр 4 — это счетчик команд, то команда LOAD REGISTER 4 аналогична команде безусловного перехода. Электронно-вычислительные машины, в ко-

торых управление счетчиком команд осуществляется таким способом, могут быть очень гибкими. Однако программист затруднится определить порядок, в котором выполняются команды, так как команды передачи управления не отличаются от других команд.

Команды условного перехода по программе являются важной составной частью большинства программ. Такие команды позволяют ЭВМ повторять последовательности команд, искать и выделять определенные символы, выявлять ошибки и контролировать состояние периферийных устройств. Команды условного перехода являются ключевыми в решении задач на ЭВМ, когда необходимо выбирать последовательности команд на основе информации, полученной от входных данных и в процессе выполнения операций. Именно команды условного перехода превращают ЭВМ в «интеллигентный» контроллер.

Наиболее простая команда условного перехода — это команда **ПЕРЕХОД ПО УСЛОВИЮ (JUMP ON CONDITION)**. Эта команда вызывает переход, если имеет место выполнение условия. Если условие не выполнено, ЭВМ выполняет очередную команду из последовательности. Условие, включенное в команду, может иметь различные формы. Это может быть состояние отдельных разрядов регистра признаков, таких как **ПЕРЕНОС (CARRY)**, **НУЛЬ (ZERO)**, **ЗНАК (SIGN)** или **ПЕРЕПОЛНЕНИЕ (OVERFLOW)**.

В табл. 3.5 содержится несколько примеров команд **ПЕРЕХОД ПО УСЛОВИЮ** и состояния отдельных разрядов регистра признаков. Условием перехода может быть и внешней входной сигнал, который вводится в ЭВМ, например **ПЕРЕХОД ПО ПРОВЕРЯЕМОМУ НУЛЮ (JUMP ON TEST ZERO)**. Некоторые ЭВМ допускают комбинации условий (например, **ПЕРЕХОД ПО ПЕРЕНОСУ и НУЛЮ ПЕРЕПОЛНЕНИЯ (JUMP ON CARRY AND OVERFLOW ZERO)**).

Иногда команды условного перехода действуют так, как будто они всегда следуют за командой **СРАВНИТЬ**. Так, команда **ПЕРЕХОД ПО РАВЕНСТВУ (JUMP ON EQUAL)** вызывает по программе переход, если две сравнимые величины оказались равными. Такие команды условного перехода, как **ПЕРЕЙТИ ПРИ ОТСУТСТВИИ РАВЕНСТВА (JUMP ON NOT EQUAL)**, **ПЕРЕЙТИ, ЕСЛИ ВЕЛИЧИНА БОЛЬ-**

Т а б л и ц а 3.5. Команды условных переходов

Адрес	Команда	Результат
100	JUMP ON CARRY 150	(PC) = 150, если ПЕРЕНОС = 1
135	JUMP ON NOT ZERO 139	(PC) = 101, если ПЕРЕНОС = 0 (PC) = 139, если НУЛЬ = 0
160	JUMP ON NEGATIVE 120	(PC) = 136, если НУЛЬ = 1 (PC) = 120, если ЗНАК = 1
145	JUMP ON OVERFLOW 147	(PC) = 161, если ЗНАК = 0 (PC) = 147, если ПЕРЕПОЛНЕНИЕ = 0 (PC) = 146, если ПЕРЕПОЛНЕНИЕ = 1

ШЕ (JUMP ON GREATER THAN), ПЕРЕЙТИ, ЕСЛИ ВЕЛИЧИНА МЕНЬШЕ (JUMP ON LESS THAN), и т. д., осуществляют переход, если число, записанное в аккумуляторе, находится в определенном отношении с числом, с которым оно должно сравниваться.

В некоторых ЭВМ используются только команды условного перехода типа ПРОПУСТИТЬ (SKIP). В таких ЭВМ процессор не может осуществить условный переход по программе, но может по условию пропустить очередную команду в последовательности. В таких ЭВМ операция условного перехода выполняется по двум командам. Последовательность команд

```
SKIP ON NOT CONDITION  
JUMP LOCATE
```

производит то же действие, что и одиночная команда

```
JUMP ON CONDITION LOCATE
```

Команда условного пропуска SKIP заставляет ЭВМ пропустить команду безусловного перехода, если условие не выполняется. Эта техника неудобна, но имеет то преимущество, что только команда безусловного перехода требует адресации памяти.

Недостающие для данной ЭВМ команды условного перехода можно получить простым способом. Например, если ЭВМ имеет команду ПЕРЕХОД ПО НУЛЕВОМУ РЕЗУЛЬТАТУ (JUMP ON ZERO), но не имеет команды ПЕРЕХОД ПО НЕНУЛЕВОМУ РЕЗУЛЬТАТУ (JUMP ON NOT ZERO), то последовательность команд

```
JUMP ON ZERO *+2  
JUMP ADD R
```

эквивалентна одиночной команде

```
JUMP ON NOT ZERO ADD R
```

Осуществляя управление программным циклом, часто приходится выполнять определенное число итераций. Ниже приводится типичная последовательность команд, предназначенная для суммирования десяти чисел:

Адрес

Команда

- 0 LOAD INDEX REGISTER #10 (ЗАГРУЗИТЬ В ИНДЕКСНЫЙ РЕГИСТР НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД 10)
- 1 CLEAR ACCUMULATOR (ОЧИСТИТЬ АККУМУЛЯТОР)
- 2 ADD 100, X (ПРИБАВИТЬ 100+X)
- 3 DECREMENT INDEX REGISTER (ВЫЧЕСТЬ 1 ИЗ СОДЕРЖИМОГО ИНДЕКСНОГО РЕГИСТРА)
- 4 JUMP ON NOT ZERO 2 (ПЕРЕЙТИ К 2 ПО НЕНУЛЕВОМУ РЕЗУЛЬТАТУ)

После каждого цикла (адреса 2—4) команда ADD 100, X прибавляет другое число (из ячеек 101—100) к сумме, находящейся в аккумуляторе. Индексный регистр, кроме своих прямых функций, выполняет

еще и функции счетчика итераций. Заметим, что подсчет в индексном регистре позволяет признаку НУЛЬ (ZERO) выступать в качестве условия выхода из цикла.

Команды, оперирующие с подпрограммами. Операции с подпрограммами отличаются от обычных переходов по программе тем, что вычислительный процесс должен вернуться после исполнения подпрограммы к первоначальной программе. Например, программе ЗАРПЛАТА нужны подпрограммы, которые занимаются выборкой файлов данных, сортировкой данных, подсчетом заработной платы, налогов и печатают результаты. Очевидно, что многие из этих подпрограмм могут использоваться другими программами или в различных местах этой же программы. Электронно-вычислительная машина должна выполнить подпрограмму и затем вернуться к основной программе в то место, где ее выполнение было приостановлено подпрограммой.

Основные команды с подпрограммами — это команды CALL (ВЫЗОВ ПОДПРОГРАММЫ) и RETURN (ВОЗВРАТ К ОСНОВНОЙ ПРОГРАММЕ). Команда CALL передает управление подобно команде JUMP; разница заключается в том, что по команде CALL сохраняется содержимое счетчика команд с тем, чтобы иметь возможность вернуться к основной программе. По команде RETURN ЭВМ возвращается к первоначальной программе, т. е. в счетчике команд восстанавливается значение, запомненное командой CALL.

В некоторых ЭВМ по команде CALL запоминается содержимое счетчика, команд в ячейке памяти, к которой осуществляется переход. Эта разновидность команд CALL называется JUMP AND MARK

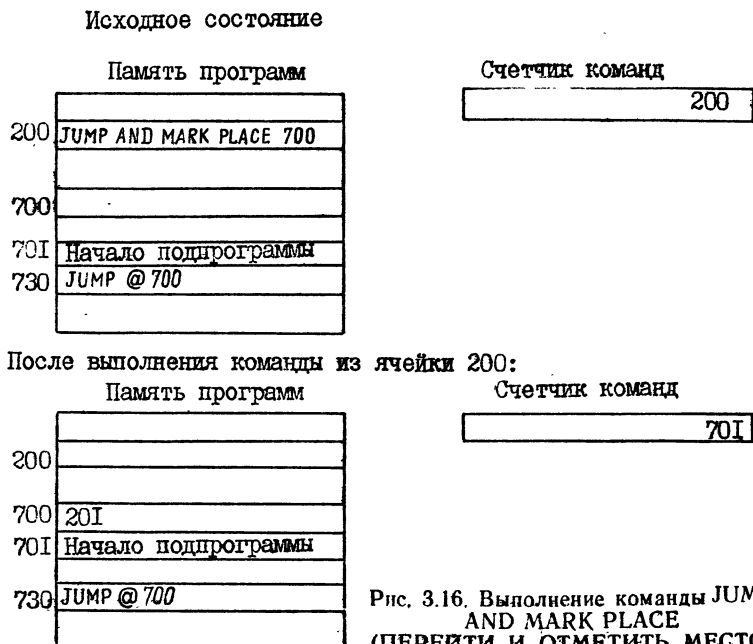
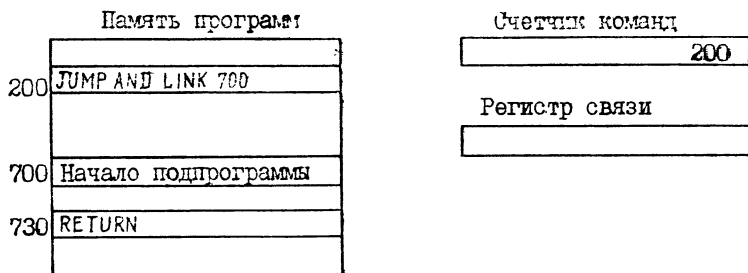


Рис. 3.16. Выполнение команды JUMP AND MARK PLACE (ПЕРЕЙТИ И ОТМЕТИТЬ МЕСТО)

Начальное состояние



После выполнения команды из ячейки памяти 200

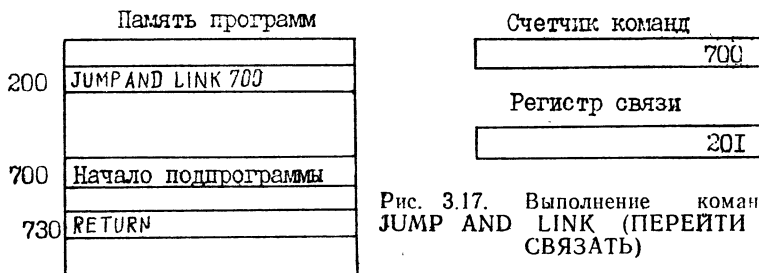


Рис. 3.17. Выполнение команды JUMP AND LINK (ПЕРЕЙТИ И СВЯЗАТЬ)

PLACE или JUMP AND SAVE (ПЕРЕЙТИ И ОТМЕТИТЬ МЕСТО или ПЕРЕЙТИ И ЗАПОМНИТЬ). На рис. 3.16 показано выполнение команды JUMP AND MARK PLACE. Электронно-вычислительная машина помещает прежнее содержимое счетчика команд в ячейку памяти 700 и начинает выполнение подпрограммы с ячейки 701. Чтобы вернуться к основной программе, нет необходимости в специальной команде RETURN. Команда косвенного перехода (JUMP @ 700) в конце подпрограммы обеспечить переход к адресу, сохраняемому в ячейке памяти 700. Так как ячейка памяти 700 содержит адрес команды, следующей непосредственно за JUMP AND MARK PLACE, ЭВМ возвращается к основной программе.

Этот метод управления подпрограммами может быть использован многократно. Подпрограмма, в свою очередь, может вызывать другую подпрограмму. Однако подпрограмма не может вызывать сама себя так как в этом случае был бы разрушен адрес возврата, хранимый в первой ячейке подпрограммы.

Другой разновидностью команды CALL является команда JUMP AND LINK (ПЕРЕЙТИ И СОЕДИНИТЬ), которая помещает старое содержание счетчика команд в соединительный регистр. На рис. 3.17 дан пример команды JUMP AND LINK. Когда ЦП выполняет команду, извлеченную из ячейки памяти 200, он засылает число 700 в счетчик команд, а инкрементированное содержимое счетчика команд (201) — в регистр. Команда RETURN в конце подпрограммы (ячейка памяти 730) помещает содержимое регистра в счетчик команд и, таким образом, возвращается управление основной программе. Этот метод управ-

ления подпрограммами быстрее метода, использующего команду JUMP AND MARK PLACE, так как ЦП не нужно запоминать старое содержимое счетчика команд или выполнять длительный косвенный переход в конце подпрограммы. Однако этот метод не позволяет осуществлять вложение подпрограмм, т. е. подпрограмма не может вызвать другую подпрограмму, пока она загружает регистр. Регистры, которые используются для соединения подпрограмм, непригодны для других целей.

Еще одной разновидностью команды CALL является команда, по которой запоминается адрес возврата в стеке (см. рис. 2.19). В этом случае команда RETURN является командой безусловного перехода, которая использует стековую адресацию. Команда CALL запоминает адрес возврата в стеке; команда RETURN восстанавливает его в счетчике команд по окончании выполнения подпрограммы. Этот метод требует дополнительного времени, так как происходит обращение к стеку. Использование стека позволяет иметь много уровней вложения подпрограммы (в зависимости от емкости стека) и обрабатывать рекурсивные выражения.

Многие ЭВМ имеют специальную команду для вызова подпрограмм, размещенных по нескольким специальным адресам. Эта команда называется TRAP или SOFTWARE (ЛОВУШКА или ПРОГРАММНОЕ ПРЕРЫВАНИЕ). Команды TRAP имеют короткий формат и выполняются быстро, так как их адресные части обычно фиксируются либо изготовителем ЭВМ, либо операционной системой. В некоторых ЭВМ подпрограммы обслуживания программных прерываний размещаются в специальном ПЗУ, к которому ЭВМ может обращаться при возникновении ошибок или сбоев в вычислительном процессе.

Команды ОСТАНОВ И ОТСУТСТВИЕ ОПЕРАЦИЙ. Команды ОСТАНОВ (HALT) заставляет ЭВМ приостановить на время работу до поступления внешнего сигнала. По команде ОТСУТСТВИЕ ОПЕРАЦИЙ (NO OPERATION) не выполняется никакой операции, она только занимает цикл команды и увеличивает содержимое счетчика команд на единицу. Эта команда очень полезна. Команды NO OPERATION могут обеспечивать короткие задержки, замещать ошибочные команды, они предоставляют возможность для внесения исправлений в программу и замещают подпрограммы, которые возможно еще и не написаны.

Команды управления состоянием

Наиболее простыми командами управления состоянием являются команды РАЗРЕШИТЬ ПРЕРЫВАНИЕ (ENABLE INTERRUPT) и ЗАПРЕТИТЬ ПРЕРЫВАНИЕ (DISABLE INTERRUPT). Большинство ЭВМ автоматически запрещает работу системы прерывания во время команды RESET или когда прерывание принято. Можно разрешить прерывание в процессе инициализации программы и запретить его в течение временных циклов или других программ, которые не могут быть правильно возобновлены. У некоторых ЭВМ в регистре состояний имеются биты, разрешающие прерывание. В этом случае, из-

меняя содержимое регистра, можно разрешить или запретить работу всей системы прерывания или части ее. Другие команды управления состоянием включают в себя разделение страниц памяти, обозначение регистров для частных задач (таких как счетчики команд или регистры адреса) и разрешение или запрет работы аварийных сигналов, концевых выключателей и других внешних устройств.

Комбинированные команды

Некоторые ЭВМ имеют команды, которые в одном цикле выполняют несколько операций. Одна команда может не только выполнять арифметическую или логическую операцию, но также сформировать сигнал переноса, сдвинуть результат и при определенных условиях пропустить очередную команду. Такие команды увеличивают производительность и уменьшают требования к объему памяти. Однако комбинированные команды сложны для использования программистом и документирования. Появление микропрограммирования сделало комбинированные команды более легкими для использования, но составление полезных комбинаций связано с некоторыми трудностями.

Команды и признаки состояний

Выработка командами значений признаков состояний очень разнообразна в различных ЭВМ. Команды ADD и SUBTRACT всегда изменяют признаки. Команды INCREMENT и DECREMENT обычно не влияют на признаки состояния ПЕРЕНОС (CARRY) и поэтому могут участвовать в циклах, выполняющих арифметические операции с повышенной точностью. Логические операции, такие как ИЛИ, И и ИСКЛЮЧАЮЩЕЕ ИЛИ, всегда воздействуют на признаки ЗНАК (SIGN) и НУЛЬ (ZERO). Операции сдвига могут изменять или не изменять любые признаки, кроме признака ПЕРЕНОС. Команды сравнения служат только для установки признаков состояний. Операции специального назначения и такие команды, как УМНОЖИТЬ (MULTIPLY), ДЕЛИТЬ (DIVIDE) и ВЗЯТЬ ДОПОЛНИТЕЛЬНЫЙ КОД (COMPLEMENT), в различных ЭВМ воздействуют на признаки состояний по-разному.

Важно, влияют ли на значения признаков состояния команды передачи данных, управления программой, управления состояниями. Во многих ЭВМ на признаки воздействуют только команды обработки данных. В этом случае они имеют специальные команды, которые устанавливают признаки в соответствии с содержимым регистров или ячеек памяти. В других ЭВМ признаки изменяются только под воздействием специальных команд.

Наборы команд

Нет сомнения в том, что ЭВМ со всеми возможными командами не существует. Большинство ЭВМ содержит от 20 до 200 отдельных команд, многие из которых могут отличаться только используемым методом

адресации. Число действительно различных команд нередко трудно определить, так как ЭВМ может иметь комбинированные команды или необычную архитектуру. Широкий набор команд делает программы короче, а быстродействие выше. Однако такой набор команд требует более длинных команд и более сложного декодирования; кроме того, он более труден для изучения и эффективного использования. Программисты редко используют большое число команд из набора; как правило, только немногие программисты могут эффективно применять наборы, в которых более 100 команд. Вместе с тем малое число команд приводит к необходимости выполнять неудобные манипуляции для реализации простых операций. Наиболее оптимальными следует считать наборы из 40—80 команд.

В большинстве ЭВМ наборы команд имеют некоторые ограничения, связанные с методами адресации. Хотя в ЭВМ могут применяться многие методы адресации, в основных командах используются только некоторые из них. Приведено несколько типичных ограничений:

1. Арифметические и логические операции могут выполняться только над содержимым регистров.

Чтобы сложить содержимое ячейки памяти 50 с содержимым аккумулятора, необходимо использовать последовательность команд

```
LOAD R2,50  
ADD R2
```

вместо одиночной команды ADD50. Это ограничение проявляется в программах, требующих большого числа команд передачи данных.

2. В однооперандных командах, таких как СДВИГ, ВЗЯТЬ ОБРАТНЫЙ КОД, УВЕЛИЧИТЬ или УМЕНЬШИТЬ НА 1, разрешается использовать только аккумулятор.

В этом случае данные необходимо засылать в аккумулятор и после операции снова запоминать их в исходной ячейке. Это ограничение также сказывается в том, что в программе появляется много команд передачи данных.

3. Обмен данными с памятью и операции ввода-вывода можно осуществить только через аккумулятор.

В этом случае снова появится необходимость в дополнительных командах. Чтобы переслать содержимое регистра 2 в ячейку памяти 50, потребуются последовательность команд

```
MOVE ACC, R2  
STORE 50
```

Заметим, что при этом необходимо запомнить предыдущее содержимое аккумулятора.

4. Команды условных переходов могут использовать только короткие относительные адреса. Более длинные условные переходы требуют последовательностей команд. Например, последовательность команд

```
JUMP ON NOT CONDITION* +2  
JUMP ADD R
```

выполняет то же действие, что и неразрешенная команда
JUMP ON CONDITION ADD R

Все эти ограничения упрощают декодирование команд, но они же делают длинные программы труднее для написания и понимания.

3.4. СИСТЕМЫ КОМАНД МИКРОПРОЦЕССОРОВ

Форматы команд

Каждая микро-ЭВМ характеризуется разрядностью слова, объемом ЗУ и ограниченным числом шин и регистров, что и определяет выбор ее системы команд. Микропроцессоры имеют значительно менее мощные системы команд, чем новые микро-ЭВМ.

Методы адресации

Ограниченный формат слова большинства МП не только влияет на уменьшение числа возможных методов адресации, но также ухудшает пригодность некоторых из них. Так как большинство МП имеет 4- или 8-битные слова, использование значительного числа бит команды для поля адресации не желательно. Поэтому большинство процессоров имеет ограниченное число методов адресации, используемое в некоторых популярных МП.

Короткий формат слова МП делает неудобной прямую адресацию, большинство МП используют ее редко. Чтобы сформировать прямой адрес, нужно иметь не только одно или два слова адреса в памяти, но и дважды обратиться к памяти для их извлечения.

Индексная адресация имеет те же самые недостатки, что и прямая, так как требует одно или два дополнительных слова в памяти программ. Кроме того, индексная адресация требует увеличения формата адреса, что приводит к увеличению времени выполнения команды в процессоре, так как последний вынужден использовать 12- или 16-битные адреса, имея только 4- или 8-битные арифметические устройства.

В МП широко используется относительная адресация для уменьшения формата адреса. Однако применение однословного адреса означает, что возможно только короткое перемещение по полю адресов памяти. Относительная адресация также требует дополнительного машинного времени, как и индексная.

В МП удобна и часто используется регистровая и косвенная адресации, так как они позволяют иметь короткий формат команд. Число регистров обычно невелико, так как имеют место команды длиной в одно слово.

Регистровая косвенная адресация является основным типом адресации в памяти в таких процессорах, как Intel 4040, Intel 8008, Intel 8080, RCA CDP 1802 и Fairchild F-8. Косвенная регистровая адресация требует, чтобы программист упорядочивал данные в памяти. Если данные будут разбросаны по разным, далеко расположенным друг от друга областям памяти, то в программе потребуется содержимое адресных регистров; программист должен так составлять программы и формировать структуру данных, чтобы минимизировать число таких дополнительных команд.

Стековая адресация, хотя и кажется идеально подходящей для МП, пока широко не используется. Большинство МП применяют стековую адресацию только для запоминания адресов возврата из подпрограмм и для запоминания содержимого регистров во время исполнения подпрограмм и обработки прерываний. Многие МП имеют стек только для хранения адресов возврата; такой стек расположен на кристалле процессора и, очевидно, не может использоваться широко. Такие процессоры, как Intel 8080 и Motorola 6800, имеют внешние по отношению к процессору стеки, и обращение к ним осуществляется во время обработки прерываний и при выполнении подпрограмм.

3.5. ПРИМЕРЫ СИСТЕМ КОМАНД МИКРОПРОЦЕССОРОВ

В этом параграфе описываются системы команд МП Intel 8080 и Motorola 6800. Для обеих систем команд будем использовать символику языка ассемблера и пояснять ее специфику.

Форматы команд

Как Intel 8080, так и Motorola 6800 в основном используют одноадресные команды. В МП Motorola 6800 имеются два аккумулятора и в большинстве команд 1 бит, адресующий один из них. В МП Intel 8080 для выполнения некоторых операций передачи данных используются двухадресные команды.

Методы адресации

Как показано в табл. 3.6, МП Intel 8080 и Motorola 6800 используют различные методы адресации. В Intel 8080 в основном применяется прямая регистровая и косвенная регистровая адресации. Три бита во многих командах указывают один из семи регистров или метод косвенной адресации (адрес в регистрах H и L). Таблица 3.7 содержит адреса регистров Intel 8080. Процессор имеет 1-байтные команды, которые оперируют с регистрами или ячейками памяти, при этом 16-битный адрес находится в регистрах H и L. Микропроцессор Intel 8080 использует непосредственно адресацию для инициирования счетчиков и адресных регистров, прямую — для доступа к разрозненным данным или к косвенным адресам и стековую — для сохранения и восстановления содержимого регистров во время выполнения подпрограмм или программы обслуживания прерывания.

Микропроцессор Motorola 6800 использует преимущественно прямую и косвенную адресации. Нулевая страница памяти используется для прямых адресов часто выбираемых данных. Имеется несколько 1-байтных команд и много 2-байтных команд с адресами на нулевой странице, индексными и относительными адресами (относительная адресация используется только в командах передачи управления). Микропроцессор Motorola 6800 использует непосредственную адресацию для загрузки констант в регистры и начальных значений индексного ре-

Таблица 3.6. Методы адресации наиболее известных МП

Метод адресации	Intel 4040*	Intel 8080	Motorola 6800**	Signetics 2650***	Fairchild F-8
Прямая:	+	+	+	+	—
нулевая страница	—	—	+	—	—
текущая страница	+	—	—	—	—
Косвенная	—	—	—	—	—
Индексная	—	—	+	+	—
Относительная	—	—	+	+	+
Непосредственная	+	+	+	+	+
Прямая регистровая	+	+	+	+	+
Косвенная регистровая	+	+	—	—	+
Стековая	—	+	+	—	—

* Перед использованием регистровой косвенной адресации необходимо запомнить содержимое регистра;

** имеется 16-битный индексный регистр;

*** для четырех индексных регистров имеется автоматическое уменьшение и увеличение на 1.

Таблица 3.7. Адреса регистров МП Intel 8080

Адрес	Регистр	Адрес	Регистр
000	B	101	L
001	C	110	Регистр косвенного адреса HL, т. е. регистр адреса M
010	D		
011	E		
100	H	111	A

гистра и регистра — указателя стека. Стековая адресация используется для временного хранения данных, а также во время выполнения подпрограмм и программ обслуживания прерываний.

Команды преобразования данных

Команда арифметических операций. В табл. 3.8 приведены арифметические операции МП Intel 8080 и Motorola 6800. Оба МП имеют команды: СЛОЖИТЬ (ADD), СЛОЖИТЬ С ПЕРЕНОСОМ (ADD WITH CARRY), ВЫЧЕСТЬ (SUBTRACT), ВЫЧЕСТЬ С ПЕРЕНОСОМ (SUBTRACT WITH CARRY) или (BORROW) УВЕЛИЧИТЬ НА 1 (INCREMENT), УМЕНЬШИТЬ НА 1 (DECREMENT). Микропроцессор Intel 8080 использует прямую регистровую, косвенную регистровую (через регистры H и L) или непосредственную адресацию с командами СЛОЖИТЬ и ВЫЧЕСТЬ. Команды УВЕЛИЧИТЬ НА 1 или УМЕНЬШИТЬ НА 1 могут выполняться с любым регистром или ячейкой памяти, адресуемой регистрами H и L. Типичные примеры команд МП Intel 8080

1. ADD B

$$(A) = (A) + (B)$$

2. INR M

$$(H \text{ и } L) = (H \text{ и } L) + 1$$

Команды УВЕЛИЧИТЬ НА 1 и УМЕНЬШИТЬ НА 1 не изменяют значение разряда ПЕРЕНОС

Таблица 3.8. Команды арифметических операций МП Intel 8080 и Motorola 6800

Команда		Код операции	
		Intel 8080	Motorola 6800
ADD	(СЛОЖИТЬ)	ADD или ADI *	ADD ** или ABA
ADD WITH CARRY	(СЛОЖИТЬ С ПЕРЕНОСОМ)	ADC или ACI	ADC
ADD DOUBLE WORDS	(СЛОЖИТЬ 2-БАЙТНЫЕ СЛОВА)	DAD	
DECREMENT	(УМЕНЬШИТЬ НА 1)	DCR	DEC
DOUBLE DECREMENT	(УМЕНЬШИТЬ НА 1 2-БАЙТНОЕ СЛОВО)	DCX	DES или DEX
INCREMENT	(УВЕЛИЧИТЬ НА 1)	INR	INC
DOUBLE INCREMENT	(УВЕЛИЧИТЬ НА 1 2-БАЙТНОЕ СЛОВО)	INX	INS или INX
SUBTRACT	(ВЫЧЕСТЬ)	SUB или SUI	SUB или SBA
SUBTRACT WITH CARRY	(ВЫЧЕСТЬ С ЗАЕМОМ)	SBB или SBI	SBC

* Для непосредственной адресации в МП Intel используются мнемокоды операций, оканчивающиеся на букву I (Immediate); ** в МП Motorola используются в конце многих кодов операций символы A и B для указания используемого аккумулятора.

Микропроцессор Motorola 6800 использует адресацию к нулевой странице, индексную, непосредственную или прямую¹ для команд СЛОЖИТЬ и ВЫЧЕСТЬ. Команды УВЕЛИЧИТЬ НА 1 или ВЫЧЕСТЬ 1 могут выполняться либо с аккумулятором,

¹ Микропроцессоры фирмы интерпретируют адресацию к нулевой странице как прямую адресацию, а обычную прямую — как расширенную

Таблица 3.9. Адреса регистровых пар МП Intel 8080

Адрес	Регистровая пара	Адрес	Регистровая пара
00	BC	10	HL
01	DE	11	Указатель стека SP

Примечание. Код 11 адресует содержимое аккумулятора и регистра признаков в командах PUSH и POP.

мулятором, либо с ячейкой памяти с использованием индексной или прямой адресации. Команды УВЕЛИЧИТЬ НА 1 и ВЫЧЕСТЬ 1 не влияют на признак ПЕРЕНОС. Типичные примеры команд МП Motorola 6800:

1. ADA 50

$$(A) = (A) + (50)$$

2. DEC 1000

$$(1000) = (1000) - 1$$

Микропроцессор Motorola 6800 имеет также специальные 1-байтные команды для сложения и вычитания содержимого аккумулятора [ABA означает $(A) = (A) + (B)$, SBA означает $(A) = (A) - (B)$].

Как Intel 8080, так и Motorola 6800 могут инкрементировать или декрементировать 16-битные адреса за один цикл команды. Команды МП Intel 8080 используют адреса регистровой пары (табл. 3.9). Первый регистр в паре содержит восемь старших значащих бит. Команды двойного формата МП Motorola 6800 могут воздействовать или на индексный регистр, или на регистр — указатель стека.

Микропроцессор Intel 8080 также имеет 2-байтную команду ADD (DAD), которая использует регистры H и L как 16-битный аккумулятор и регистровую пару (по адресу из табл. 3.9) для представления второго операнда. Эта команда может быть использована для индексирования (базовый адрес помещается в одну регистровую пару, а индекс — в другую) или для сдвига влево 16-битного числа из регистровой пары H и L (DAD H).

Микропроцессор Intel 8080 имеет несколько команд арифметических операций специального назначения. Среди них:

1. SUB A

$$(A) = (A) - (A) = 0$$

Эта команда очищает аккумулятор.

2. ADD A

$$(A) = (A) + (A)$$

По этой команде содержимое аккумулятора сдвигается влево на 1 бит.

Команды логических операций, выполняемые МП Intel 8080 и Motorola 6800, представлены в табл. 3.10. Команды ИЛИ (OR), И (AND) и ИСКЛЮЧАЮЩЕЕ ИЛИ (EXCLUSIVE OR) выполняются аналогично командам СЛОЖИТЬ и ВЫЧЕСТЬ обоими процессорами. В МП Motorola 6800 эти команды не влияют на разряд ПЕРЕНОС; в МП Intel 8080 они обнуляют его. Команды ВЗЯТЬ ОБРАТНЫЙ или ДОПОЛНИТЕЛЬНЫЙ КОД ЧИСЛА (COMPLEMENT ONES или COMPLEMENT TWOS) в МП Motorola 6800 используются для получения обратного или дополнительного кода числа как из аккумулятора, так и из ячейки памяти (короткая адресация к нулевой странице памяти не разрешается). Эти же команды в МП Intel 8080 могут оперировать только с аккумулятором или знаком ПЕРЕНОС. В МП Motorola 6800 команды ВЗЯТЬ ДОПОЛНИТЕЛЬНЫЙ КОД действуют на все признаки; в МП Intel 8080 команда COMPLEMENT ACCUMULATOR не действует ни на один признак.

Таблица 3.10. Команды логических операций МП Intel 8080 и Motorola 6800

Команда		Код операции	
		Intel 8080	Motorola 6800
AND COMPLEMENT (one's)	(Логическое И) (ПОЛУЧЕНИЕ ОБРАТНОГО КО- ДА ЧИСЛА)	ANA или ANI CMA или CMC*	AND COM
COMPLEMENT (two's)	(ПОЛУЧЕНИЕ ДОПОЛНИ- ТЕЛЬНОГО КО- ДА ЧИСЛА)		NEG
EXCLUSIVE OR	(ИСКЛЮЧАЮ- ЩЕЕ ИЛИ)	XRA или XRI	EOR
OR	(Логическое ИЛИ)	ORA или ORI	ORA

* CMA — complement accumulator, CMC — complement carry.

Некоторые логические операции в МП Intel 8080 имеют специальное назначение:

1. XRA A

$$(A) = (A) \oplus (A) = 0$$

Эта команда обнуляет аккумулятор.

2. ORA A или ANA A

$$(A) = (A) + (A) \text{ или } (A) \cdot (A)$$

Эти команды устанавливают все признаки в соответствии с числом, находящимся в аккумуляторе.

Типичные логические операции, выполняемые МП Intel 8080:

1. ANI 11110000 B

$$(A) = (A) \cdot 11110000$$

Эта команда маскирует четыре младших значащих разряда в аккумуляторе.

2. ORA M

$$(A) = (A) + (H \text{ и } L)$$

Эта команда логического ИЛИ над содержимым ячейки памяти, адресуемой регистровой парой H и L.

Типичные логические операции, выполняемые МП Motorola 6800:

1. AND A # % 00001111 (символ # означает непосредственную адресацию, а % на языке ассемблера Motorola 6800 — двоичный код)

$$(A) = (A) \cdot 00001111$$

Эта команда маскирует четыре старших значащих разряда аккумулятора.

2. COM 2,X

$$((\text{INDEX REGISTER}) + 2) = \overline{((\text{INDEX REGISTER}) + 2)}$$

Эта команда представляет собой исполнительный адрес в дополнительном коде.

Команды операций сдвига. В табл. 3.11 даны команды операций сдвига, выполняемые МП Intel 8080 и Motorola 6800. Микропроцессор Intel 8080 может выполнять операцию сдвига только над содержимым аккумулятора; МП Motorola 6800 осуществляет сдвиг содержимого аккумулятора или ячейки памяти. Команды операций сдвига в МП Intel 8080 не влияют ни на какие признаки, кроме признака ПЕРЕНОС; команды операций сдвига в МП Motorola 6800 влияют на все признаки.

Таблица 3.11. Команды операций сдвига, выполняемые МП Intel 8080 и Motorola 6800

Команда		Код операции	
		Intel 8080	Motorola 6800
ARITHMETIC SHIFT RIGHT	(АРИФМЕТИЧЕСКИЙ СДВИГ ВПРАВО)	(ADD A)	ASR
LOGICAL SHIFT LEFT	(ЛОГИЧЕСКИЙ СДВИГ ВЛЕВО)		ASL
LOGICAL SHIFT RIGHT	(ЛОГИЧЕСКИЙ СДВИГ ВПРАВО)		LSR
ROTATE LEFT	(ЦИКЛИЧЕСКИЙ СДВИГ ВЛЕВО)	RLC	
ROTATE RIGHT	(ЦИКЛИЧЕСКИЙ СДВИГ ВПРАВО)	RRC	
ROTATE WITH CARRY LEFT	(ЦИКЛИЧЕСКИЙ СДВИГ ВЛЕВО С ПЕРЕНОСОМ)	RAL	ROL
ROTATE WITH CARRY RIGHT	(ЦИКЛИЧЕСКИЙ СДВИГ ВПРАВО С ПЕРЕНОСОМ)	RAR	ROR
OTHERS	(ПРОЧИЕ)	(DAD H)	

Команды сравнения, выполняемые МП Intel 8080 и Motorola 6800, приведены в табл. 3.12. В МП Intel 8080 и Motorola 6800 по команде СРАВНИТЬ (COMPARE) выполняется вычитание, но результат не помещается в аккумулятор. Команда действует на все признаки, не изменяя содержимое какого-либо регистра. Микропроцессор Motorola 6800 имеет команду, аналогичную команде СРАВНИТЬ, включая специальную форму для сравнения содержимого аккумуляторов, а также команды ПРОВЕРИТЬ (TEST) и ПРОВЕРИТЬ БИТ (BIT TEST). Команда ПРОВЕРИТЬ вычитает нуль из числа, которое может находиться в аккумуляторе или ячейке памяти. Таким образом, команда ПРОВЕРИТЬ устанавливает признаки в соответствии с числом, находящимся в аккумуляторе или ячейке памяти, не изменяя самого числа. Команда ПРОВЕРИТЬ БИТ выполняет операцию логического И над содержимым аккумулятора и адресуемым числом, не изменяя содержимого в каком-либо регистре. Таким образом, команда ПРОВЕРИТЬ БИТ позволяет программисту проверить содержимое отдельного бита с помощью операции логического И над словом и соответствующей маской.

Микропроцессор Motorola 6800 также имеет команду СРАВНИТЬ двойного формата, которая позволяет сравнить содержимое индексного регистра с другим 16-битным числом. Эта команда выполняет 16-разрядное вычитание, не изменяя содержимого какого-либо регистра.

Таблица 3.12. Команды операций сравнения, выполняемые МП Intel 8080 и Motorola 6800

Команда		Код операций	
		Intel 8080	Motorola 6800
BIT TEST	(ПРОВЕРИТЬ БИТ)	CMR или CPI	BIT
COMPARE	(СРАВНИТЬ БАЙТЫ)		CMR или CBA
DOUBLE TEST	(СРАВНИТЬ 2-БАЙТНОЕ СЛОВО) (ПРОВЕРИТЬ)		CPX TST

Типичные операции сравнения, выполняемые МП Intel 8080:

1. CMP B

Устанавливает признаки после выполнения операции (A) — (B).

2. CPI 10

Устанавливает признаки после выполнения операции (A) — 10.

Типичные операции сравнения, выполняемые МП Motorola 6800:

1. CMPA # 30

Устанавливает признаки после выполнения операции (A) — 30.

2. TST 2000

Устанавливает признаки после выполнения операции (2000) — 0.

Специальные команды Единственная специальная команда, выполняемая МП Intel 8080 или Motorola 6800, — это так называемая команда десятичной коррекции, DAA (DECIMAL ADJUST ACCUMULATOR). Эта команда позволяет вместо сложения в двоичном коде выполнить сложение в двоично-десятичном коде (BCD addition). Последовательность команд

ADD

DAA

аналогична команде сложения в десятичном коде.

Десятичная коррекция использует как основной признак ПЕРЕНОС, так и ВСПОМОГАТЕЛЬНЫЙ ПЕРЕНОС (HALT-CARRY или AUXILIARY CARRY) для выполнения коррекции результата. Команда DAA — единственная команда, выполняемая как МП Intel 8080, так и Motorola 6800, которая использует разряд ВСПОМОГАТЕЛЬНЫЙ ПЕРЕНОС (HALT-CARRY или AUXILIARY CARRY).

Команды передачи данных

Команды обращения к памяти. Как МП Motorola 6800, так и МП Intel 8080 имеют 1- и 2-байтные команды ЗАГРУЗИТЬ и ЗАПОМНИТЬ для обмена данными между регистрами и памятью. Эти команды приведены в табл. 3.13.

Команды передачи данных, выполняемые МП Motorola 6800, узконаправленные. Команды ЗАГРУЗИТЬ и ЗАПОМНИТЬ передают 8-битные числа между памятью и одним из двух аккумуляторов. Двухбайтные команды ЗАГРУЗИТЬ и ЗАПОМНИТЬ передают 16-битные числа между памятью и индексным регистром — указателем стека. Эти команды могут использовать любой метод адресации, имеющийся в МП Motorola 6800. Так, можно загрузить регистры по команде ЗАГРУЗИТЬ, используя непосредственную адресацию, адресацию нулевой страницы, индексную или прямую адресацию.

В МП Intel 8080 команды передачи данных в память более сложные. Здесь 8-битное число можно пересылать через любой регистр и регистровую пару, используя команды косвенной регистровой адресации (коды операций MOV REG, M; MOV M, REG; LDAX и STAX), команды с непосредственной адресацией (MVI) или команды с прямой адресацией (STA, LDA). Программист обычно использует метод косвенной регистровой адресации с регистровой парой H и L для обработки массивов или таблиц, команды LDAX и STAX, чтобы обрабатывать два массива одновременно, MVI, чтобы загружать счетчики и другие установочные данные и STA и LDA, чтобы иметь доступ к рассредоточенным данным.

Передача 16-битных данных в МП Intel 8080 возможна различными способами. Обычно используют команды с непосредственной адресацией (LXI) для загрузки адресных регистров фиксированными величинами, и команды с прямой адресацией (LHLD, SHLD), чтобы обеспечить возможность косвенной адресации и загрузить адресные регистры переменными величинами.

Ниже показаны типичные команды передачи данных в память, выполняемые МП Motorola 6800:

1. LDAA # 35

(A) = 35

2. STX 1515

(1515) = (XH)
(1516) = (XL)

Таблица 3.13. Команды передачи данных, выполняемые МП Intel 8080 и Motorola 6800

Команда		Код операции	
		Intel 8080	Motorola 6800
LOAD	(ЗАГРУЗИТЬ)	LDA (прямая адресация) MOV r, M (косвенная адресация с помощью регистров H и L) MVI r, VAL (непосредственный операнд) LDAX (косвенная адресация с помощью регистров B и C или D и E)	LDA
LOAD DOUBLE-WORD	(ЗАГРУЗИТЬ ДВОЙНОЕ СЛОВО)	LXI (непосредственный операнд) LHLD (прямая адресация)	LDS, LDX
STORE	(ЗАПОМНИТЬ)	STA (прямая адресация) MOV M, r (косвенная адресация с помощью регистров H и L) STAX (косвенная адресация с помощью регистров B и C или D и E) SHLD	STA
STORE DOUBLE-WORD	(ЗАПОМНИТЬ ДВОЙНОЕ СЛОВО)		STS, STX

Здесь IХN и IXL — соответственно старшие и младшие значащие разряды индексного регистра.

Типичные команды передачи данных в память, выполняемые МП Intel 8080:

1. MOV C, M

$$(C) = ((H \text{ и } L))$$

Центральный процессор загружает регистр C числом из ячейки памяти, адресуемой содержимым регистровой пары H и L.

2. LXI H 160

$$(H \text{ и } L) = 160$$

Число 160 помещается в регистровую пару H и L.

Оба процессора осуществляют обмен в память данными, имеющими двойной формат, по 1 байту за цикл обращения к памяти. Преимущество обмена двойными словами над обменом по одному слову состоит в том, что процессору нужно извлечь из памяти только одну команду, чтобы выполнить пересылку двойного слова, а не две

Основное отличие команд пересылки данных, выполняемых МП Intel 8080 и Motorola 6800, состоит в том, что у первого они не влияют на значения признаков состояния, в то время как у второго — воздействуют на них. В МП Intel 8080 команды пересылки данных могут заслать нуль в регистр без изменения разряда признака НУЛЬ (ZERO); любое число, помещенное в регистр МП Motorola 6800, воздействует на признаки.

Микропроцессор Motorola 6800 имеет также команду ОЧИСТИТЬ (CLEAR) для каждого из двух аккумуляторов, а также для разрядов ПЕРЕНОС (CARRY) и ПЕРЕПОЛНЕНИЕ (OVERFLOW). Микропроцессор Intel 8080 имеет только

7	6	5	4	3	2	1	0
S Признак знака	Z Признак нуля	O Посто- янный 0	AS Вспомо- гатель- ный пе- ренос	O Постоян- ный 0	P Чет- кость	I Посто- янная	C Пере- нос

Рис. 3.18. Слово состояния МП Intel 8080

команду УСТАНОВИТЬ (SET) для разряда ПЕРЕНОС. Ячейки памяти и регистры должны очищаться с помощью логических и арифметических операций, или операций пересылок, как это описывалось ранее.

Команды ввода-вывода. Микропроцессор Intel 8080 имеет простые команды ввода-вывода. Команда ВВОД (IN) передает байт из адресуемого входного устройства в аккумулятор. Команда ВЫВОД (OUT) передает байт из аккумулятора в выходное адресуемое устройство. Обе команды (ВВОД и ВЫВОД) используют 8-битные адреса этих устройств, а не полные 16-битные адреса памяти (Zilog-80, улучшенный вариант Intel 8080, имеет команды ввода-вывода, которые передают блок данных).

Микропроцессор Motorola 6800 совсем не имеет команд ввода-вывода. Он рассматривает устройство ввода-вывода как ячейки памяти, и поэтому любая команда, которая передает данные в память или из памяти, может служить командой ввода или вывода. Такими наиболее простыми командами являются ЗАГРУЗИТЬ (LOAD) и ЗАПОМНИТЬ (STORE); они действуют как команды ввода-вывода, когда их адреса действительно являются адресами устройств ввода-вывода.

Другие команды МП Motorola 6800 могут использоваться аналогично. Например, можно заслать нуль в выходное устройство путем обнуления связанной с ним ячейки памяти. Можно исследовать входные данные, используя команды СРАВНИТЬ (COMPARE) и ПРОВЕРИТЬ (TEST), в которых адресуется не ячейка памяти, а устройство ввода.

Команды, оперирующие со стеком. Как Motorola 6800, так и Intel 8080 имеют 16-разрядный регистр — указатель стека и стек, который расположен во внешнем ОЗУ. Регистр—указатель стека МП Intel 8080 содержит адрес последнего байта, помещенного в стек; регистр—указатель стека МП Motorola 6800 содержит адрес очередной пустой ячейки.

Микропроцессор Intel 8080 использует стековые операции с двойными словами.

Команда PUSH помещает содержимое пары регистров в стек, а команда POP извлеченное из стека содержимое передает паре регистров. Содержимое стека наращивается вниз от верхушки памяти, т. е. по команде PUSH центральный процессор уменьшает на два содержимое регистра—указателя стека. Этот прием адресации стека позволяет использовать младшие адреса ОЗУ для размещения данных, а остальные отводить для стека. Микропроцессор Intel 8080 может оперировать командами PUSH и POP с любой регистровой парой. Здесь регистровая пара 3 — это специальный 16-разрядный регистр, называемый *регистром слова состояния процессора* (PSW); его старшие восемь разрядов есть содержимое аккумулятора, а младшие восемь — признаки, распределенные, как показано на рис. 3.18. Когда МП Intel 8080 помещает содержимое регистровой пары в стек, он начинает с восьми старших разрядов. Конечно, команды POP изменяют этот порядок на противоположный.

Микропроцессор Motorola 6800 использует однословные операции со стеком. Команда PSN помещает содержимое аккумулятора в стек, а PUL помещает верхний байт из стека в аккумулятор. Команды PSN и PUL применимы и для работы с остальными регистрами МП.

Приведем несколько типичных примеров операций со стеками:

1. Intel 8080
- а) PUSH D

Начальные условия:
(STACK POINTER) = 173
(D) = 55
(E) = 37

Конечные условия:
(STACK POINTER) = 171
(172) = 55
(171) = 37
6) POP H

Начальные условия:
(STACK POINTER) = 200
(200) = 31
(201) = 0

Конечные условия:
(STACK POINTER) = 202
(H) = 0
(L) = 31
2 Motorola 6800

а) PSHA
Начальные условия:
(STACK POINTER) = 250
(A) = 64

Конечные условия:
(STACK POINTER) = 249
(250) = 64
6) PULB

Начальные условия:
(STACK POINTER) = 180
(181) = 72

Конечные условия:
(STACK POINTER) = 181
(B) = 72

Команды внутрипроцессорного обмена. Микропроцессор Motorola 6800 имеет несколько команд межрегистровых пересылок. Наиболее простыми являются TBA, передающая данные из B в A, и TAB, передающая данные из A в B.

Микропроцессор Intel 8080 использует команды внутрипроцессорного обмена чаще, так как он имеет большее число регистров. По команде MOV можно передать данные из любого регистра в любой другой регистр. Однако чаще всего используется передача данных в аккумулятор и из него. По команде XCHG регистровые пары L и H обмениваются содержимым.

Приведем несколько типичных команд внутрипроцессорного обмена для МП Intel 8080:

1. MOV B, A
(B) = (A). Содержимое A не изменяется.
2. MOV A, D
(A) = (D). Содержимое D не изменяется.

Команды управления программой

Команды безусловного перехода. Микропроцессор Intel 8080 имеет две команды безусловного перехода: команда JMP использует 16-разрядный прямой адрес; команда PCHL помещает содержимое регистровой пары HL в счетчик команд и таким образом осуществляет безусловный переход по программе.

Микропроцессор Motorola 6800 также имеет две команды безусловного перехода. Команда JMP может использовать как прямую, так и косвенную адресацию. Можно осуществить косвенный переход, используя индексную адресацию с нулевым смещением. По команде JMP 0, X (или просто JMPX) центральный процессор передает управление по адресу, содержащемуся в индексном регистре BRA (BRANCH ALWAYS), использует относительную адресацию. Восемь раз-

Таблица 3.14. Команды условного перехода, выполняемые МП Intel 8080 и Motorola 6800

Команда		Код операции	
		Intel 8080	Motorola 6800
JUMP IF CARRY=0 CARRY=1	(ПЕРЕЙТИ, ЕСЛИ ПЕРЕНОС=0 ПЕРЕНОС=1)	JNC JC	BCC BCS
JUMP IF OVERELOW=0 OVERFLOW=1	(ПЕРЕЙТИ, ЕСЛИ РЕПОЛНЕНИЕ=0 ПЕРЕПОЛНЕНИЕ=1)		BVC BVS
JUMP IF PARITY=0 PARITY=1	(ПЕРЕЙТИ, ЕСЛИ ЧЕТНОСТЬ=0 ЧЕТНОСТЬ=1)	JPO JPE	
JUMP IF SIGH=0 SIGH=1	(ПЕРЕЙТИ, ЕСЛИ ЗНАК=0 ЗНАК=1)	JP JM	BPL BMI
JUMP IF ZERO=0 ZERO=1	(ПЕРЕЙТИ, ЕСЛИ НУЛЬ=0 НУЛЬ=1)	JNZ JZ	BNE BEQ
JUMP IF <0 ≤0 >0 ≥0	ПЕРЕЙТИ, ЕСЛИ <0 ≤0 >0 ≥0		BLT BLS, BLE BHI, BGT BGE

рядов относительного адреса разрешают переход на 128 ячеек в любом направлении (фактически на 129 вперед или на 126 назад, так как отсчет начинается с адреса второго байта 2-байтной команды BRANCH). Таким образом, JMP используется для длинных или косвенных переходов, а BRA — для коротких переходов.

Команды условного перехода. В табл. 3.14 приведены команды условного перехода, используемые МП Intel 8080 и Motorola 6800. Микропроцессор Intel 8080 выполняет условные переходы по любому значению признаков: 1 или 0. Все команды условных переходов используют 16-разрядный прямой адрес.

Наиболее простыми командами условных переходов являются ПЕРЕХОД ПО НУЛЮ JZ (JUMP ON ZERO) и ПЕРЕХОД ПРИ ОТСУТСТВИИ НУЛЯ JNZ (JUMP ON NOT ZERO). Команда JNZ вызывает переход по программе, если признак НУЛЬ равен 0 (предыдущий результат не был равен 0). В большинстве программных циклов используется команда JZ или JNZ. Типичная последовательность команд

```
DCR C
JNZ LOOP
```

заставляет ЦП выполнять программу (с начальным адресом LOOP), пока содержимое регистра C остается равным 0. На рис. 3.19 показан типичный цикл программы, в которой требуемое число итераций (COUNT) определено в регистре C. По ходу основной программы содержимое регистра C уменьшается на 1 (декрементируется) и команда JNZ используется для определения числа повторения основной программы.

Команды JZ и JNZ можно также использовать для обращения к отдельным байтам данных. Последовательность

```
CPI 100
JZ F100
```

вызывает переход по адресу F100, если содержимое аккумулятора равно 100.

Другие простые команды условного перехода — это ПЕРЕХОД ПО ПРИЗНАКУ ПЕРЕНОСА JC (JUMP ON CARRY) и ПЕРЕХОД ПРИ ОТСУТСТВИИ ПРИЗНАКА ПЕРЕНОСА JNC (JUMP ON NOT CARRY). Эти команды часто используются вместе с командой СРАВНИТЬ (COMPARE). Команда СРАВНИТЬ устанавливает признак ПЕРЕНОС в 1 в том случае, если 8-битное число без

START COUNTER
LOOP.

MVIC, COUNT

Основная программа
в цикле

DCR C
JNZ LOOP

Следующая часть
программы

Рис. В.19. Типичный цикл программы

знака в аккумуляторе меньше числа, с которым оно сравнивается. При сравнении $CARRY = 1$ означает, что для выполнения вычитания нужен заем. Последовательность команд

CPI 10
JC LSTEN

вызывает переход к ячейке с адресом LSTEN, если в аккумуляторе содержится число без знака, которое меньше 10 ($CARRY = 10$, если содержимое аккумулятора равно 10).

Таким же образом последовательность команд

CPI 64
JNC LARGE

вызывает переход к ячейке с адресом LARGE, если в аккумуляторе содержится число без знака, значение которого лежит между 64 и 255

Команды JC и JNC можно использовать для проверки значения разряда, сдвигаемого на место разряда ПЕРЕНОС. Заметим, что в МП Intel 8080 команды сдвига не влияют ни на какие другие признаки. Таким образом, для того чтобы определить, четное ли число содержится в аккумуляторе (последний значащий разряд есть 0), используется последовательность

RAR
JNC EVEN

По команде RAR последний значащий разряд аккумулятора перемещается в разряд ПЕРЕНОС, и команда JNC вызывает переход к ячейке с адресом EVEN, если ПЕРЕНОС = 0.

Микропроцессор Motorola 6800 в командах условных переходов использует только относительную адресацию. Поэтому, как и в команде безусловного перехода BRANCH (BRA), возможность выполнения безусловных переходов ограничена 129 ячейками вперед или 126 назад. Более длинные переходы по программе могут быть получены при использовании последовательности

BRANCH ON NOT CONDITION OVER
JMP ADDR
OVER

Микропроцессор МП Motorola 6800 подобно Intel 8080 имеет команды условного перехода BRANCH, но с дополнительными комбинациями. Названия большинства команд перехода отражают тот факт, что исполняются операции сравне-

ния. Команда BEQ (ПЕРЕЙТИ ПРИ УСЛОВИИ РАВЕНСТВА BRANCH ON EQUAL) вызывает переход, если признак НУЛЬ равен единице, т. е. при сравнении числа оказались равными. Команда BNE (BRANCH ON NOT EQUAL) вызывает переход, если признак НУЛЬ равен нулю. Как и в МП Intel 8080, эти команды могут управлять циклами и выполнять разнообразные проверки.

Типичный пример организации цикла (счетчик содержит адрес ячейки памяти 40):

```
DEC 40
BNE LOOP
```

Типичный пример организации поиска:

```
CMRA #100
BEQ F100
```

Как и в МП Intel 8080, можно использовать разряд ПЕРЕНОС при выполнении команды СРАВНИТЬ. Микропроцессор Motorola 6800 имеет не только команды ВС (ПЕРЕЙТИ ПРИ НАЛИЧИИ ПЕРЕНОСА — BRANCH IF CARRY SET) и ВСС (ПЕРЕЙТИ ПРИ ОТСУТСТВИИ ПЕРЕНОСА — BRANCH IF CARRY CLEAR), но и дополнительную пару команд — ВНІ (ПЕРЕЙТИ, ЕСЛИ БОЛЬШЕ, — BRANCH IF HIGHER) и ВLS (ПЕРЕЙТИ, ЕСЛИ МЕНЬШЕ ИЛИ РАВНО, — BRANCH IF LOWER OR SAME). Команда ВLS вызывает переход, если содержимое аккумулятора меньше или равно заданному числу, а команда ВНІ, если содержимое аккумулятора больше заданного числа. Так, после выполнения команды CMRA # 64 возможны следующие варианты:

1. BCS DONE (PC) = DONE, если (A) < 64
2. BLS DONE (PC) = DONE, если (A) ≤ 64
3. BCC DONE (PC) = DONE, если (A) ≥ 64
4. BHI DONE (PC) = DONE, если (A) > 64

Так как МП Motorola 6800 имеет разряд ПЕРЕПОЛНЕНИЕ (OVERFLOW), арифметическая операция получения числа в дополнительном коде у него проще чем у МП Intel 8080. Используя команды BVC (ПЕРЕЙТИ ПРИ ОТСУТСТВИИ ПЕРЕПОЛНЕНИЯ — BRANCH IF OVERFLOW CLEAR) и BVS (ПЕРЕЙТИ ПРИ НАЛИЧИИ ПЕРЕПОЛНЕНИЯ — BRANCH IF OVERFLOW SET), можно определить, имеет ли место переполнение при получении дополнительного кода чисел. Команды BGE (ПЕРЕЙТИ, ЕСЛИ БОЛЬШЕ ИЛИ РАВНО НУЛЮ — BRANCH IF GREATER THAN OR EQUAL TO ZERO), BLT (ПЕРЕЙТИ, ЕСЛИ МЕНЬШЕ НУЛЯ, — BRANCH IF LESS THAN ZERO), BGT (ПЕРЕЙТИ, ЕСЛИ БОЛЬШЕ НУЛЯ, — BRANCH IF GREATER THAN ZERO) и BLE (ПЕРЕЙТИ, ЕСЛИ МЕНЬШЕ ИЛИ РАВНО НУЛЮ, — BRANCH IF LESS THAN OR EQUAL TO ZERO) также анализируют возможность переполнения. Логические функции признаков (табл. 3.15) определяют различные условия переходов по программе.

Таблица 3.15. Команды комбинированных условных переходов, выполняемые МП Intel 8080 и Motorola 6800

Команда	Обозначение	Проверяемое условие
JUMP IF < 0 (ПЕРЕЙТИ, ЕСЛИ < 0)	BLT	$N \oplus V = 1$
JUMP IF ≤ 0 (ПЕРЕЙТИ, ЕСЛИ ≤ 0)	BLE	$Z + (N \oplus V) = 1$
JUMP IF ≥ 0 (ПЕРЕЙТИ, ЕСЛИ ≥ 0)	BLS	$C + Z = 1$
JUMP IF ≥ 0 (ПЕРЕЙТИ, ЕСЛИ ≥ 0)	BGE	$N \oplus V = 0$
JUMP IF > 0 (ПЕРЕЙТИ, ЕСЛИ > 0)	BGT	$Z + (N \oplus V) = 0$
JUMP IF > 0 (ПЕРЕЙТИ, ЕСЛИ > 0)	BHT	$C + Z = 0$

Команды, оперирующие с подпрограммами. Как Intel 8080, так и Motorola 6800 для работы с подпрограммами используют стек. Прежде чем начальный адрес подпрограммы будет помещен в счетчик команд, в стек засылается содержимое счетчика команд по команде ВЫЗВАТЬ (CALL) в МП Intel 8080 или по командам JSR и BSR (JUMP; BRANCH TO SUBROUTINE) в МП Motorola 6800. Команда RETURN, выполняемая Intel 8080, и команда RTS (RETURN FROM SUBROUTINE), выполняемая Motorola 6800, реализуют возврат к прерванной программе и восстанавливают прежнее содержимое счетчика команд.

Микропроцессор Intel 8080 имеет команды условного перехода CALL и RETURN. Несмотря на свою простоту, программистами они используются редко. Микропроцессор Intel 8080 имеет также специальную 1-байтную команду CALL для обработки прерываний. Это специальная команда BST (ПОВТОРНЫЙ ПУСК — RESTART) аналогична команде, описанной в гл. 9.

Микропроцессор Motorola 6800 также имеет несколько специальных команд прерывания. Это команда SWI (ПРЕРЫВАНИЕ ПРОГРАММЫ — SOFTWARE INTERRUPT), которая вызывает переход по адресу, содержащемуся в ячейках памяти с адресами FFF8 и FFF9, и запоминает содержимое всех регистров в стеке, и команда RTI (ВОЗВРАТ ПОСЛЕ ПРЕРЫВАНИЯ — RETURN FROM INTERRUPT), по которой из стека извлекается содержимое всех регистров. Система прерываний МП Motorola 6800 описана в гл. 9.

Команды ОСТАНОВ и ОТСУТСТВИЕ ОПЕРАЦИЙ. Как МП Intel 8080, так и МП Motorola 6800 имеют команду ОТСУТСТВИЕ ОПЕРАЦИЙ (NO OPERATION), которая просто увеличивает на единицу содержимое счетчика команд. Микропроцессор Intel 8080 имеет команду HLT (ОСТАНОВ — HALT), по которой процессор останавливается и может быть вновь запущен только по сигналам прерывания или сброса. Микропроцессор Motorola 6800 имеет команду WAI (ОЖИДАНИЕ ПРЕРЫВАНИЯ — WAIT FOR INTERRUPT), по которой запоминается содержимое всех регистров, и МП ждет сигнала прерывания. Оба МП имеют также несколько неопределенных кодов операций; воздействие этих кодов операций на центральный процессор аналогично выполнению им команды NO OPERATION.

Команды управления состоянием

Команды управления состоянием, выполняемые МП Intel 8080 и Motorola 6800, только разрешают или запрещают работу системы прерывания. В МП Intel 8080 — это команды EI (РАЗРЕШЕНИЕ ПРЕРЫВАНИЯ — ENABLE INTERRUPTS) и DI (ЗАПРЕТ ПРЕРЫВАНИЙ — DISABLE INTERRUPTS). Микропроцессор Motorola 6800 имеет разряд маскирования прерывания I; команда CLI (ОБНУЛЕНИЕ РАЗРЯДА ПРЕРЫВАНИЯ — CLEAR INTERRUPT) обнуляет маскирующий разряд и разрешает прерывания, а команда SEI (SET INTERRUPT) устанавливает разряд маскирования в единицу и запрещает прерывания. Оба процессора автоматически запрещают работу системы прерывания на период первоначальной установки в нуль и после приема сигнала запроса прерывания. Система прерывания МП Intel 8080 после этого должна приводиться в действие программой обслуживания прерывания. Система прерывания МП Motorola 6800 будет автоматически приведена в действие командой RTI, так как по этой команде извлекается старое значение разряда маскирования прерываний из стека.

Выводы

Команда представляет собой двоичный входной код, на который центральный процессор отвечает определенной последовательностью действий. Каждая команда должна определить операцию, источники операндов, место назначения результата и адрес следующей команды. Во многих малых ЭВМ большая часть этой командной информации выражена неявно, содержится в коде команды. Результатом использования неявно выраженной информации является более короткий формат команд, но это означает, что для выполнения операций обработки данных требуется большое число команд. Стандартным форматом является одноадресная команда, по которой ЭВМ извлекает один операнд из аккумулятора, re-

зультат отправляет обратно в аккумулятор и автоматически увеличивает содержимое счетчика команд на 1.

Для задания адреса операндов может быть использовано много различных методов. Прямая адресация проста и может оперировать с отдельными независимыми байтами данных. Косвенный и индексный методы адресации более сложны, но позволяют программе так модифицировать действительный или исполнительный адрес, чтобы имелась возможность обрабатывать массивы и таблицы данных. Непосредственная адресация используется для задания констант, в то время как относительная адресация позволяет перемещать фрагменты программы из одной части памяти в другую. Использование стековой адресации затруднительно для программиста. Регистровая адресация позволяет ему поместить данные или адреса в рабочие регистры МП и многократно использовать их без дополнительных обращений к памяти.

Система команд может быть разделена на несколько основных категорий:

- а) команды преобразования данных;
- б) команды передачи данных, которые только передают данные из одного места в другое, но не преобразуют их;
- в) команды управления программой, которые изменяют обычную последовательность команд;
- г) команды управления состояниями ЭВМ.

Типичными командами преобразования данных являются команды арифметических и логических операций, операции сдвига и сравнения или операции специального назначения. Команды передачи данных выполняют операции обмена с памятью, межрегистровые передачи, операции ввода-вывода и обмена со стеком. Типичными командами управления программой являются команды условного и безусловного переходов, оперирующие с подпрограммами, команды **ОСТАНОВ** и **ОТСУТСТВИЕ ОПЕРАЦИЙ**. Команды управления состоянием разрешают или запрещают прерывание программы и определяют некоторые другие виды операций.

Обычно система команд МП содержит от 40 до 80 различных команд. Короткий формат данных большинства микро-ЭВМ предопределяет преимущественное использование коротких команд и неявного метода адресации. Методы регистровой и стековой адресации более эффективны, чем методы прямой, косвенной или индексной адресации, при которых необходимо использовать длинные адреса.

Команды МП Intel 8080 и Motorola 6800 имеют различные методы адресации, но их системы команд похожи. Микропроцессор Intel 8080 использует адресный регистр HL для быстрого обращения к памяти; имеются специальные команды для загрузки, запоминания, увеличения и уменьшения на 1 содержимого этих регистров. Микропроцессор Motorola 6800 использует метод адресации к нулевой странице и 16-разрядный индексный регистр для обращения к памяти; имеются специальные команды для загрузки, запоминания, увеличения и уменьшения на 1 содержимого индексного регистра. Оба процессора используют стек во внешнем ОЗУ для запоминания адреса возврата из подпрограммы и для временного хранения содержимого рабочих регистров.

ГЛАВА ЧЕТВЕРТАЯ

МИКРОПРОЦЕССОРНЫЕ АССЕМБЛЕРЫ

В данной главе рассматриваются особенности программ-ассемблеров, которые обычно используются для разработки программного обеспечения (ПО) систем, построенных на базе микропроцессоров.

В первом параграфе этой главы рассматриваются различные уровни языковых средств, с помощью которых могут быть написаны программы, и обсуждаются их достоинства и недостатки. В последующих параграфах приводятся основные характеристики ассемблеров и специ-

фические особенности микропроцессорных ассемблеров. В последнем параграфе описаны стандартные ассемблеры для микропроцессоров Intel 8080 и Motorola 6800.

4.1. СРАВНЕНИЕ ЯЗЫКОВ РАЗЛИЧНОГО УРОВНЯ

Программирование на машинном языке

Команды, выполняемые ЭВМ, представляют собой двоичные числа, которые центральный процессор (ЦП) выбирает из памяти точно так же, как и любые другие данные, а затем декодирует их и выполняет требуемые операции. Например, команда МП Intel 8080, которая выполняет сложение содержимого регистра общего назначения (РОН) и аккумулятора, представляет собой 8-разрядное двоичное число

100 000 00.

Эта команда выглядит точно так же, как дополнительный код числа 128 или восемь младших двоичных разрядов адреса 128. Электронно-вычислительная машина определяет, является ли выбранное из памяти число командой, частью адреса или просто частью данных в зависимости от того, в какой фазе цикла выполнения команды оно находится. Таким образом, ЭВМ может интерпретировать одно и то же число из одной и той же ячейки памяти тремя совершенно различными способами.

Одинаковое представление данных, адресов и команд в памяти ЭВМ является одной из многих трудностей, связанных с составлением программ непосредственно в той форме, в которой машина могла бы их выполнять. Подобный способ составления программ называется *программированием на машинном языке*. Например, на рис. 4.1 приведена программа на машинном языке МП Intel 8080, которая пересылает содержимое десяти последовательных ячеек памяти, начиная с адреса 40₁₆, в десять новых ячеек, начиная с адреса 60₁₆. Очевидно, что разобраться в работе такой программы, внести в нее изменения и использовать ее будет достаточно сложно.

Адрес	Содержимое	Адрес	Содержимое
0	00100001	8	01111110
1	01000000	9	00010010
2	00000000	10	00100011
3	00010001	11	00010011
4	01100000	12	00000101
5	00000000	13	11000010
6	00000110	14	00001000
7	00001010	15	00000000
		16	01110110

Рис. 4.1. Программа на машинном языке

	LXI	H, BLK1	;Указатель памяти 1 = начало блока 1
	LXI	D, BLK2	;Указатель памяти 2 = начало блока 2
	MVI	B, COUNT	;Счетчик = длина блока
TRANS:	MOV	A, M	;Загрузить элементы блока 1
	STAX	D	;Послать элементы в блок 2
	INX	H	
	INX	D	
	DCR	B	
	JNZ	LOOP	
	HLT		

Рис. 4.2. Программа на языке ассемблера

Чтобы следить за изменением содержимого ячеек памяти, регистров и признаков, программисту потребуется таблица двоичных кодов команд и лист бумаги. Он также должен хорошо владеть двоичной арифметикой и хорошо разбираться в организации МП Intel 8080. Работать с двоичными числами достаточно трудно, а различать, что ячейка 6 содержит команду, ячейка 7 — данные, ячейка 14 — часть адреса, почти невозможно. Поэтому большинство программистов не составляет свои программы непосредственно на машинном языке.

Простой альтернативой программированию на машинном языке является использование в программах вместо двоичных кодов символических имен для обозначения команд, регистров, данных, ячеек памяти. Программист должен построить таблицы двоичных эквивалентов различных символических имен и использовать их для перевода программы в форму, в которой на может быть выполнена ЭВМ.

Чтобы упростить описание ЭВМ, изготовитель обозначает команды и регистры символическими именами.

Символические имена являются мнемоническими; это значит, что они вызывают ассоциации с функциями, выполняемыми командами, или с назначением регистров.

Приведем типичные примеры мнемонических обозначений:

ADD — сложить
 SUB — вычесть
 LD — загрузить
 JMP — перейти
 A — аккумулятор
 X — индексный регистр

Программист обязан, естественно, присвоить символические имена данным и адресам, используемым в своей программе, и построить таблицы двоичных эквивалентов этих имен. Программу, показанную на рис. 4.1, можно записать в форме, приведенной на рис. 4.2. При этом были использованы правила составления программ на языке ассемблера. В табл. 4.1 показан перевод символических имен, определенных в программе, в их двоичные эквиваленты. Подобный способ составления программ называется *программированием на языке ассемблера*, а про-

Таблица 4.1. Таблица символических имен для программы, приведенной на рис. 4.2

Имена	Значения (двоичные)	Имена	Значения (двоичные)
BLK1	01000000	COUNT	00001010
BLK2	01100000	TRANS	00001000

цесс преобразования программ в двоичную форму — *ручным ассемблированием*.

Поскольку различные команды МП Intel 8080 занимают различное число ячеек памяти, необходимо определить действительный адрес, соответствующий имени TRANS. Это можно сделать, либо непосредственно определив, какое число ячеек занимают все предшествующие этой точке команды, либо предварительно преобразовав программу в двоичную форму и затем отыскав нужные адреса в программе на машинном языке.

Очевидно, что в программе, показанной на рис. 4.2, легче разобраться, чем в программе, приведенной на рис. 4.1. Данным, адресам и командам даны различные обозначения. Мнемонические коды команд, выбранные изготовителем, соответствуют выполняемым машинным командам. Для данных и адресов выбраны такие имена, которые определяют смысл их использования в программе. Правильность программы можно проверить перед загрузкой ее в ЭВМ.

И все-таки процесс программирования еще не очень удобен. Необходимо переводить код каждой команды в двоичный эквивалент, запоминать, требует ли команда дополнительных слов для указания данных и адресов, и, если требует, представлять себе, каков формат этих дополнительных ячеек. Необходимо создать таблицу, подобную табл. 4.1, присвоить значения различным именам или определить их значения из двоичной программы и преобразовать данные и адреса в двоичную форму. Очевидно, при этом приходится повторять много одинаковых действий, что утомляет и может привести к незначительным, но имеющим катастрофические последствия ошибкам. Если потребуется использовать буквенные или цифровые коды, относительные адреса или программы, написанные кем-либо другим, возникнут дополнительные трудности. Программирование на языке ассемблера с ручным ассемблированием легче программирования на машинном языке, но, естественно, это не лучший способ писать сложные программы.

Ручное ассемблирование затрудняет также и процесс отладки. Кроме распознавания таких ошибок, как неправильные коды команд или ошибочные двоичные преобразования, программист, использующий ручное ассемблирование, обязан следить за всеми последствиями каждого изменения (в тексте программы). Например, если в программе на рис. 4.2 по ошибке была опущена команда MVI B, COUNT, которая задает начальное значение счетчика 10, то программа будет работать неверно. Однако простое добавление этой команды не сделает программу правильной. Так как новая команда занимает две дополнительные

ячейки памяти, необходимо скорректировать также все адреса в программе, расположенные ниже вставленной команды. Например, адрес TRANS следовало бы изменить с 6 на 8. Таким образом, любое изменение, которое предполагает вставку или удаление команд или использование разных областей памяти под данные, приводит к необходимости повторить ручное ассемблирование. Трудности и большая вероятность появления ошибок при составлении и отладке программ, ассемблируемых вручную, очевидны.

Машинное ассемблирование и программа-ассемблер

Процесс ручного ассемблирования несложен; однако операции преобразования мнемонических кодов операций и десятичных чисел в их двоичные эквиваленты, введение счетчика с целью определить значения адресов, а также корректное размещение данных и адресов в формате команд занимают много времени и имеют повторяющийся характер. Эти операции легко может выполнять ЭВМ, которая никогда не делает ошибок из-за небрежности, никогда не устает и никогда не покидает своего рабочего места. Следовательно, для решения задач, возникающих в процессе программирования, можно воспользоваться услугами ЭВМ.

Простым примером подобного использования ЭВМ является работа восьмеричного или шестнадцатиричного монитора. Очевидно, что программы на машинном языке проще писать, если воспользоваться восьмеричной или шестнадцатиричной системой счисления вместо двоичной. При этом соответственно втрое или вчетверо уменьшается объем вводимых данных и существенно сокращается количество ошибок. Получаемые в результате ассемблирования программы становятся более короткими и простыми с точки зрения контроля. Но как заставить ЭВМ воспринимать информацию в восьмеричной или шестнадцатиричной системе счисления? Это можно сделать, написав для ЭВМ программу, которая преобразует восьмеричные и шестнадцатиричные числа в двоичные. Эта задача проста; можно написать программу самим (на машинном языке) или использовать готовую программу, поставляемую изготовителем. Написав программу преобразования 1 раз, можно использовать ее для упрощения составления других программ. Фактически большинство изготовителей ЭВМ поставляют простой восьмеричный или шестнадцатиричный монитор (обычно размещаемый в ПЗУ), который избавляет пользователя от необходимости вводить информацию в двоичном виде.

ЭВМ может выполнить более сложную работу, чем преобразование восьмеричных или шестнадцатиричных чисел в двоичные. Она может выполнить все операции, которые должен выполнять программист при ручном ассемблировании. Необходимая для этого программа называется *ассемблером*. Ассемблер преобразует мнемонические коды в двоичные коды команд, строит таблицы имен и их значений и заменяет все ссылки на имена соответствующими двоичными числами. Очевидно, что ЭВМ может ассемблировать программы быстрее и точнее программиста.

Электронно-вычислительная машина никогда не выберет неверный код команды, не перепутает числа и не ошибется при преобразовании адреса или данных в двоичную форму. Она не забудет, сколько дополнительных слов адреса или данных требуется задать в команде или в каком формате этот адрес или данные должны быть представлены. Как будет показано далее, ассемблер может обладать также и другими полезными свойствами. Ассемблер — это программа, воспринимающая исходный текст на языке ассемблера в качестве входных данных и выдающая на выходе программу в объектном коде на машинном языке.

Естественно, такие возможности представляются не бесплатно. Пользователь должен купить или сам написать ассемблер или использовать его за плату в системе разделения времени. Ассемблер, в свою очередь, предъявляет собственные требования при составлении программ. Эти требования представляют собой совокупность правил и форматов, которые необходимо изучить. Программирование на языке ассемблера с использованием программы-ассемблера более удобно и более производительнее по сравнению с программированием на машинном языке или ручным ассемблированием.

Однако весь процесс составления программ в терминах системы команд конкретной ЭВМ не эффективен. Во-первых, можно использовать полученные программы только на ЭВМ этого типа. Эти программы нельзя легко преобразовать на язык ассемблера другой ЭВМ, так как архитектура, системы команд и методы адресации в ЭВМ очень разнообразны. Кроме того, нельзя просто воспользоваться существующими программами, написанными для других ЭВМ. Можно оказаться привязанным к конкретной ЭВМ, поскольку при изменении типа ЭВМ потребуется переписать и заново протестировать программы.

Главная (основная) проблема при составлении программ состоит в том, что язык ассемблера более тесно связан со структурой ЭВМ, чем с особенностями инженерных задач. Программируя на языке ассемблера, программист затрачивает больше времени на манипулирование регистрами и продумывание порядка выполнения команд, чем решение этой задачи. Программист должен хорошо знать систему команд, архитектуру и методы адресации ЭВМ. Чтобы написать простую программу, подобную той, которая показана на рис. 4.2, необходимо знать Intel 8080. Таким образом, программирование на языке ассемблера требует много дополнительных усилий, которые непосредственно не направлены на решение поставленной при проектировании задачи.

Система команд ЭВМ отражает недостатки технологии ее изготовления. Изготовитель создает систему команд, которая может быть легко и недорого реализована аппаратными средствами. Таким образом, пользователь редко находит отдельную команду, которая будет выполнять содержательную операцию. Даже такие простые операции, как суммирование чисел, сравнение или поиск символов, реализуются последовательностями команд. Поэтому программирование на языке ассемблера требует много времени. Большое число команд увеличивает возможность появления ошибок и затрудняет документирование про-

грамм. Производительность программиста, пишущего на языке ассемблера, низка, а полученные, в конце концов, программы имеют мало общего с инженерным описанием системы.

Процедурно-ориентированные языки

Программист может избежать зависимости своих программ от архитектуры и системы команд конкретных ЭВМ, используя *процедурно-ориентированные языки*. Процедурно-ориентированные языки или языки *высокого уровня* обладают такими возможностями, что стандартные задачи могут быть легко и естественно представлены в форме, которая понятна специальной программе-транслятору, которая называется *компилятором*. Подобно ассемблеру компилятор воспринимает как входные данные программы, написанные на процедурно-ориентированном языке, и выдает в качестве выходных данных программы на машинном языке или языке ассемблера. Вместе с тем процедурно-ориентированный язык не зависит от особенностей конкретной ЭВМ. Одна и та же программа будет выполняться на любой ЭВМ, имеющей компилятор для данного языка. Компилятор зависит от особенностей машины, а язык программирования нет. Кроме того, задачу для ЭВМ можно описать в удобной форме. При этом программа выглядит так, что становится понятным, что именно требуется сделать, а не состоит из совокупности плохо осознаваемых операций. На языке высокого уровня можно формулировать задачи более квалифицированно. При этом не требуется точного понимания архитектуры ЭВМ.

Существует много процедурно-ориентированных языков. Некоторые из них являются универсальными, тогда как другие ориентированы на решение специальных задач, например обработка данных, проектирование схем или анализ тестов. До сих пор наиболее распространенным процедурно-ориентированным языком является ФОРТРАН, появившийся в начале 50-х годов и получивший свое название от аббревиатуры Formula Translation Language (язык для трансляции формул). Другие процедурно-ориентированные языки, такие как ПЛ/1, АПЛ, БЕЙСИК, АЛГОЛ и ПАСКАЛЬ, также используются для решения инженерных задач. Однако компиляторы с ФОРТРАНа имеются на большинстве ЭВМ. Стандартный вариант языка ФОРТРАН является широко распространенным и на нем написаны тысячи программ. Таким образом, программист, работающий на ФОРТРАНе, может легко переносить программы с одной ЭВМ на другую и использовать огромные библиотеки фортрановских программ.

Если можно представить задачу в алгебраической форме, то ее легко можно записать с помощью нескольких операторов на ФОРТРАНе. Для программы, приведенной на рис. 4.1 и 4.2, которая состоит из 16 операторов на машинном языке или 10 операторов на языке ассемблера, потребуется только два оператора на ФОРТРАНе (рис. 4.3).

Очевидно, что эта программа короче, проще и легче для чтения и понимания, чем ее вариант на языке ассемблера или на машинном языке. Операторы ФОРТРАНа похожи на описание задачи, которое составляется как часть обычного процесса проектирования. Чтобы сде-

лать понятной программу на ФОРТРАНе, требуется выполнить меньше работы по документированию, чем для программы на ассемблере или машинном языке. Программисту, работающему на ФОРТРАНе, нет необходимости знать методы адресации или другие особенности ЭВМ, так как программа на ФОРТРАНе не зависит от них.

Тем не менее ФОРТРАН обладает некоторыми серьезными недостатками. Компилятор с ФОРТРАНа более сложен и дорог по сравнению с ассемблером. Он требует для своей работы более сложной конфигурации, чем минимальная вычислительная система. ФОРТРАН — сложный язык с большим числом правил, которым должен следовать программист.

Компилятор с ФОРТРАНа создает объектные программы, которые работают медленнее и требуют больше памяти по сравнению с программами на ассемблере или на машинном языке. Компилятор транслирует операторы ФОРТРАНа точно, но прямолинейно; он не обнаружит тех способов улучшения качества программы, которые очевидны опытному программисту. Большее время выполнения и больший расход памяти для программ на ФОРТРАНе являются серьезными недостатками для его применения в микропроцессорных системах. ФОРТРАН облегчает программирование, беря на себя многие функции детальной обработки данных, которые пришлось бы программировать на ассемблере, но при этом программист не может использовать своих знаний специфики конкретной ЭВМ и особенностей задачи, чтобы сделать программы более эффективными.

ФОРТРАН первоначально был создан для класса задач, выходящих за область применения микропроцессоров. Как видно из названия, ФОРТРАН разработан для решения научных задач, которые могут быть описаны математическими формулами. Микропроцессоры редко используются для решения подобных задач. ФОРТРАН не очень подходит для программирования задач реального времени, а ведь именно для решения таких задач широко используются МП. Требования к формированию, обработке двоичных данных и облегчению интерфейса, характерные для задач управления, трудно реализуются средствами ФОРТРАНа. Для решения подобных задач более подходят такие языки, как АЛГОЛ и ПЛ/1, но эти языки используются не слишком широко, что затрудняет распространение программ на другие ЭВМ. Кроме того, указанные языки имеют малые библиотеки программ. Ближайшее будущее покажет, будет ли ФОРТРАН приспособлен для нужд пользователей микропроцессоров или его заменят другие (может быть специально разработанные) языки.

При составлении программ пользователь может использовать любой из указанных в данном параграфе методов. Программирование на машинном языке или ручное ассемблирование требует наименьших средств обеспечения и позволяет программисту осуществлять непосредственный контроль над ЭВМ. Вместе с тем такие методы требуют

больших затрат времени при программировании и создают благоприятные условия для возникновения ошибок. Ассемблер освобождает программиста от повторных операций по ручному ассемблированию, но делает его зависимым от конкретной ЭВМ. Процедурно-ориентированный язык, подобный ФОРТРАНУ, упрощает программирование, делает его более производительным и независимым от ЭВМ, но в общем случае дает медленные программы, требующие много памяти. В настоящее время (с точки зрения использования микропроцессоров) недостатки процедурно-ориентированных языков преобладают над их достоинствами. Программирование для большинства микропроцессоров осуществляется на языке ассемблера, поэтому остановимся именно на этом языке. Однако в будущем большая эффективность процедурно-ориентированных языков несомненно приведет к их широкому распространению.

4.2. ХАРАКТЕРИСТИКИ АССЕМБЛЕРОВ

Основное назначение ассемблера — перевод мнемонических кодов языка ассемблера в двоичные коды машинного языка.

Некоторые ассемблеры этим и ограничиваются. В таких случаях, чтобы привести программу к соответствующему виду, программист вынужден выполнять большую работу по ручному ассемблированию.

Однако большинство ассемблеров в настоящее время позволяет использовать метки, символическую адресацию, форматные преобразования, распределение памяти, генерацию данных и выполнение арифметических операций на этапе ассемблирования. По мере расширения возможностей ассемблеров различие между ними и компиляторами фактически стирается.

Структура оператора

Операторы ассемблера состоят из нескольких частей или полей. На рис. 4.4 показана типовая структура оператора.

Пример.

LAST: JUMP START; ВОЗВРАТ К НАЧАЛУ ПРОГРАММЫ

Некоторые ассемблеры используют *фиксированный формат*, при котором элементы оператора занимают строго определенные позиции на перфокарте или на другом входном носителе. Преимущество использования фиксированных форматов состоит в том, что для разделения полей нет необходимости использовать специальные символы. Ассемблеры большинства микропроцессоров используют *свободный формат*, в котором поля могут быть размещены в любом месте строки. При этом

Метка	Мнемонический код операции	Адрес	Комментарии
-------	----------------------------	-------	-------------

Рис. 4.4. Структура оператора языка ассемблера

они разделены специальными символами, называемыми *разделителями*. Обычно в качестве разделителей используются: пробел, двоеточие, точка с запятой, запятая, с наклоном влево косая черта, вопросительный знак и другие символы, имеющиеся на стандартной клавиатуре и не используемые в программах для других целей.

Метки

Поле метки позволяет задать в символическом виде адрес команды или элемента данных. В этом случае можно использовать имя в качестве адреса или в качестве данных в других командах. Большинство ассемблеров позволяет использовать метки, состоящие из букв, цифр и некоторых других символов. Способ выбора меток рассмотрен в гл. 6 при обсуждении методов программирования. Длина метки обычно ограничена 5—6 символами. Часто первый символ должен быть буквой, так что ассемблер может легко отличить метку от числа. Данная метка может быть использована в программе только 1 раз. Использование кодов машинных команд и команд ассемблера в качестве меток нецелесообразно и часто недопустимо. Поле метки может быть пустым. Оно используется только при необходимости.

Мнемонические коды операции

В поле кода операции обычно содержится мнемонический код команды. Это единственное поле, которое никогда не может быть пустым.

Ассемблер отыскивает соответствующий код в массиве, хранимом в памяти. Соответствующие записи в других таблицах содержат двоичный эквивалент кода и адреса программы, которая будет отыскивать требуемые адреса или данные. Таблицы могут быть размещены в ПЗУ.

Псевдокоманды

В поле кода операции могут содержаться также директивы ассемблера. Они называются *псевдокомандами*, поскольку эти директивы появляются в поле кода операций, но не транслируются в двоичные коды команд. Псевдокоманды могут выделять память для программ и данных, определять символические имена, выделять память для переменных, генерировать фиксированные таблицы и данные, помечать конец программы и задавать формат листинга программы. Псевдокоманды также фигурируют в постоянных таблицах мнемонических кодов ассемблера. Они не имеют двоичных эквивалентов, но им соответствуют подпрограммы ассемблера, выполняющие предписанные действия.

Стандартными псевдокомандами являются (табл. 4.2):

ORIGIN
EQUATE или DEFINE
RESERVE
DATA

END
LIST
PAGE
SPACE
NAME или TITLE

Конкретные мнемоники, используемые в ассемблерах для обозначения псевдокоманд, могут существенно отличаться от приведенных.

Псевдокоманда ORIGIN (сокращенно ORG) позволяет размещать программы или данные, начиная с определенного места памяти. Программы инициализации, программы обработки прерываний и программа обработки других ситуаций (TRAP-программы) должны начинаться с определенных адресов памяти. Основная программа, подпрограммы и данные не должны накладываться друг на друга и на фиксированные адреса памяти ЭВМ. Как следствие в одной программе может быть объявлено несколько точек начала. На рис. 4.5 приведен типичный пример.

Псевдокоманда EQUATE (сокращенно EQU) или DEFINE определяет символические имена, которые будут затем использоваться в программе. Определяемое имя записывается в поле метки, а его значение — в поле адреса. Например:

```
COUNT EQU 10
THRSH EQU 200
KBD EQU 2
```

Таблица 4.2. Наиболее распространенные псевдокоманды (команды ассемблера)

Псевдокоманда	Краткое обозначение	Выполняемое действие
DATA (ДАННЫЕ)	DATA, DB, DW, DDB, FM FCC, FCB	Формирует значение констант в памяти
DEFINE (ОПРЕДЕЛИТЬ)	EQU, SET, DEF	Определяет имена
END (КОНЕЦ)	END	Обозначает конец ассемблерной программы
LIST (ПЕЧАТЬ)	LIST	Обеспечивает печать текста ассемблерной программы
NAME (ИМЯ)	NAME	Приписывает имя программе
ORIGIN (НАЧАЛО)	ORG	Задаёт точку в памяти, начиная с которой размещается следующая секция программы
PAGE (СТРАНИЦА)	PAGE	Осуществляет переход на следующую страницу при печати
RESERVE (ОТВЕСТИ)	RES, DS, RMB	Резервирует память для размещения переменных данных
SPACE (ПРОПУСК)	SPACE	Осуществляет переход на следующую строку при печати

```

Программа инициализации
ORG RESET
: Программа обработки прерывания # 1
ORG INT1
: Главная программа
ORG MAIN
: ПЗУ для данных
ORG VARS
: Поле данных ОЗУ

```

Рис. 4.5. Использование псевдооперации ORG

Обычно все определения размещаются в начале программы, поэтому легко можно найти их и использовать при документировании программы. Все определения можно разделить на группы определения: устройства ввода-вывода, имена переменных, фиксированные адреса памяти и параметры.

Псевдокоманда RESERVE (сокращенно RES или DS для DEFINE STORAGE, или RM для RESERVE MEMORY) резервирует для переменных место в оперативной памяти и присваивает символическому имени переменной значение адреса первой ячейки области памяти.

Псевдокоманда TEMP RESERVE 1 резервирует одно слово в памяти и приписывает этому адресу имя TEMP. Впоследствии имя TEMP можно использовать в программе. Аналогично псевдокоманда SYMTB RESERVE 100 резервирует 100 ячеек оперативной памяти и приписывает адресу первой ячейки имя SYMTB. Заметим, что программа может изменять данные, хранимые по этим адресам.

Некоторые ассемблеры позволяют программисту вводить начальные значения в ячейке оперативной памяти. Но не будем пользоваться этой возможностью, так как она основана на предположении, что программа (вместе с начальными значениями) будет загружаться в память каждый раз во время ее исполнения. Большинство микропроцессорных программ записано в ПЗУ. Они должны корректно выполняться каждый раз при включении питания, а содержимое ОЗУ в этот момент однозначно не определено.

Псевдокоманда DATA (часто встречается в вариантах DB-DEFINE BYTE, DW-DEFINE WORD или FCB-FORM CONSTANT BYTE) позволяет размещать в памяти таблицы или константы. Обычно имеется возможность приписывать символическое имя первой ячейке памяти в зарезервированной области. Так, псевдокоманда

```
TFAC DATA 32
```

обеспечивает размещение в очередной ячейке памяти числа 32 и приписывает (значение) адреса этой ячейки имени TFAC. Можно разместить в памяти таблицу, используя псевдокоманду типа

```
SQTAB DATA 0, 1, 4, 9, 16, 25, 36, 49
```

которая формирует таблицу квадратов восьми последовательных натуральных чисел и приписывает имени SQTAB значение адреса первого элемента.

Следует обратить внимание на то, что псевдокоманда DATA помещает в память программ постоянные величины, в то время как псевдокоманда RESERVE обеспечивает выделение области памяти данных (в ОЗУ). Поэтому в поле операндов псевдокоманды DATA указываются конкретные числовые значения; а в поле операндов псевдокоманды RESERVE — число выделяемых ячеек.

Псевдокоманда END служит признаком конца программы на языке ассемблера. Некоторые ассемблеры позволяют в поле адреса оператора END указывать метку, которая задает начальный адрес выполнения программы, если она была оттранслирована без ошибок. Псевдооперация

END FIRST

сигнализирует о конце программы на языке ассемблера и позволяет начать выполнение программы с оператора, имеющего метку FIRST (если программа не содержала ошибок).

Псевдооперация LIST, PAGE, SPACE, TITLE оказывает влияние только на форму выдачи листинга ассемблера.

Таблица символов

Ассемблер собирает все имена, использованные в ассемблерной программе, в таблицу символов, подобную постоянной таблице мнемонических кодов. Методы организации таблицы символов выходят за рамки данной книги, но с ними можно познакомиться по литературе, список которой приведен в конце главы. Разумеется, для создания таблицы символов ассемблер использует ОЗУ.

Адреса

Большинство ассемблеров позволяет программисту задавать данные и адреса самыми различными способами. Обычно способ адресации указывается с помощью специальных символов, например:

@ — КОСВЕННАЯ адресация;
— НЕПОСРЕДСТВЕННЫЙ операнд;
,I или ,X — ИНДЕКСАЦИЯ;
,* , \$ или ◇ — ОТНОСИТЕЛЬНАЯ адресация

Иногда метод адресации указывается путем модификации мнемонического кода операции, например:

ADDI	ADD	IMMEDIATE (ПРИБАВИТЬ НЕПОСРЕДСТВЕННЫЙ)
ADDX	ADD	INDEXED (ПРИБАВИТЬ ИНДЕКСИРОВАННЫЙ)
ADDA	ADD	TO ACCUMULATOR A (ПРИБАВИТЬ К АККУМУЛЯТОРУ A)

Большинство ассемблеров допускает задание адресов и чисел в следующем виде:

1. Десятичные числа:

ADD 136 — прибавить содержимое ячейки памяти адресом 136 к содержимому аккумулятора.

MULTIPLY #5 — умножить содержимое аккумулятора на 5.

В большинстве ассемблеров числа предполагаются десятичными, если программист не оговорил противного.

2. Восьмеричные, шестнадцатеричные или двоичные числа. Числа в указанных системах счисления идентифицируются специальным знаком в конце строки цифр (так, В означает двоичное, О или Q — восьмеричное, Н — шестнадцатеричное число). Ассемблер, разумеется, преобразует их в двоичную форму. При задании шестнадцатеричных чисел, чтобы ассемблер мог отличить их от символических имен, может дополнительно потребоваться незначащий начальный разряд (например, записать 0А, а не просто А).

SUBTRACT #ОВЗН — вычесть шестнадцатеричное число ВЗ из содержимого аккумулятора.

JUMP 53Q — перейти к ячейке с адресом 53₈ (43₁₀).

AND #00001111В — выполнить логическую операцию И над содержимым аккумулятора и двоичным числом 00001111.

Рекомендуется записывать данные и адреса в той системе счисления, которая обеспечивает наибольшую ясность. Так, числовые константы удобно задавать в десятичной системе счисления, двоично-десятичные числа — в шестнадцатеричной (так как всякая двоично-кодированная десятичная цифра является одновременно шестнадцатеричной), маски — в двоичной (так как при этом они более наглядны, хотя для задания длинных масок более удобны шестнадцатеричные числа).

3. Символические имена. Символические имена могут использоваться вместо значений, которые они представляют. Эти имена могут представлять собой адреса и данные. Если имя желательно использовать в качестве данных, его следует задать как непосредственный операнд.

Обычно в ассемблере для регистров зарезервированы специальные символические имена, например:

А — для аккумулятора,

Х — для индексного регистра.

Иногда символические имена регистров содержат букву R, например R3 или R15.

Примеры.

ADD INT

означает, что содержимое ячейки памяти, адрес которой задан именем INT, следует сложить с содержимым аккумулятора

ONE EQU 1

SUBTRACT ONE

означает, что содержимое ячейки памяти с адресом 1 (оно не обязательно равно 1) вычитается из содержимого аккумулятора:

ONE EQU 1
SUBTRACT # ONE

означает вычитание единицы (# означает непосредственный операнд) из содержимого аккумулятора.

JUMP TRAP

означает передачу управления ячейке памяти, адрес которой задан именем TRAP.

4. Значение счетчика адреса. В большинстве ассемблеров текущее значение счетчика адреса указывается символом \$ или символом *. Эту форму задания счетчика адреса можно использовать, даже если в ЭВМ отсутствует относительная адресация; ассемблер сам вычислит фактическое значение адреса.

Пример.

JUMP \$ +8

означает переход на восемь ячеек вперед.

5. Арифметические выражения. Во многих ассемблерах можно использовать в качестве адреса арифметические выражения, составленные из имен, знаков арифметических операций и чисел. Пример (\$ + 8), приведенный выше, представляет собой очень распространенный тип выражения. Способ вычисления этих выражений существенно отличается в разных ассемблерах. Поэтому программист должен пользоваться этой возможностью осмотрительно.

Примеры.

MULTIPLY ADDR+2

означает умножение содержимого сумматора на содержимое ячейки, которая находится на два номера дальше ячейки ADDR.

JUMP COUNT-1

означает переход к адресу, на единицу меньшему COUNT.

Ассемблер допускает использование таких операций, как умножение, деление без остатка и возведение в степень. В некоторых ассемблерах операции имеют такой же приоритет, как в ФОРТРАНе. Это означает, что умножение и деление выполняются раньше сложения и вычитания (операции одного приоритета вычисляются в порядке слева направо). В других ассемблерах операции не имеют приоритетов и выполняются в порядке слева направо. Заметим, что именно ассемблер осуществляет вычисление выражения; ЭВМ не может сама вычислять выражения непосредственно. Использование очень сложных выражений делает программы запутанными и затрудняет их тестирование и отладку.

6. Коды символов. Часто данные могут быть заданы кодами символов, обычно ASCII или EBCDIC. Чаще всего символьные данные заключаются в одинарные или двойные кавычки, хотя в некоторых ассемб-

лерах символьная строка начинается или завершается специальным символом, например С, А или Е.

Примеры.

COMPARE #'E'

означает сравнить содержимое аккумулятора с внутренним представлением символа Е.

LOAD #','

означает послать в аккумулятор внутреннее представление символа ",," (запятая).

7. Другие выражения. В некоторых ассемблерах при задании адресов и данных можно использовать логические операции (И, ИЛИ, НЕ, СЛОЖЕНИЕ ПО МОДУЛЮ 2), операции сдвига и другие операции. Разумеется, на практике сложные логические выражения используются редко.

Заметим, что в адресном поле псевдооперации DATA можно использовать любое из перечисленных выражений, например:

ERRM DATA 'ERROR'
PROD DATA C * D

Следует быть особенно осторожным в тех случаях, когда данные оказываются длиннее или короче адресного поля; в различных ассемблерах используются различные правила отбрасывания лишних или дописывания недостающих разрядов

Условное ассемблирование

В некоторых ассемблерах имеются средства, которые позволяют в зависимости от условий, возникающих в момент ассемблирования, включать или не включать в объектную программу отдельные участки программы. В некоторых ассемблерах имеются даже операторы условного и безусловного переходов, цикла, вызова подпрограммы; эти операторы здесь не рассматриваются, информацию о них можно почерпнуть из литературы, приведенной в списке литературы. Условное ассемблирование осуществляется с помощью псевдокоманд IF и ENDIF. Приведены типичные примеры условного ассемблирования.

Пример. Выбор способа адресации.

```
IF      X1 < 256
      ADD    X1
ENDIF
IF      X1 ≥ 256
      LOAD   # X1
      STORE  TEMP1
      ADD    @TEMP1
ENDIF
```

Этой конструкцией можно воспользоваться, если данная ЭВМ позволяет осуществлять прямую адресацию только в пределах нулевой страницы (длина страницы равна 256 ячейкам).

Если переменная X1 находится за пределами нулевой страницы, ассемблер будет использовать косвенную адресацию через ячейку памяти TEMP1, находящуюся на странице 0 и содержащую адрес X1. Значение переменной X1 (заданной с помощью псевдокоманды DEFINE или в качестве метки) в момент ассемблирования определяет, какая из двух ветвей будет включена в оттранслированную программу.

Пример. Выбор числа операндов.

```
ADD      X1
IF       X2  $\neq$  0
ADD      X2
ENDIF
IF       X3  $\neq$  0
ADD      X3
ENDIF
```

Дополнительные операнды можно исключить, если в начале программы задать их адреса равными нулю. Одну и ту же программу на языке ассемблера можно использовать при наличии дополнительных операндов и без них. Условное ассемблирование также удобно использовать для включения в программу операторов, используемых только в процессе отладки.

Макрокоманды

Если данная последовательность команд часто встречается в программах (а также в некоторых других не очень очевидных случаях), оказывается удобным обозначить ее символическим именем. Такое средство ассемблера, которое позволяет подставить вместо символического имени соответствующую последовательность команд, называется макросредством, а сама поименованная совокупность команд — макрокомандой. Ассемблер автоматически заменяет каждую ссылку на макрокоманду последовательностью команд, взятой из макроопределения.

Простейшая макрокоманда состоит из одной машинной команды. Например, на ЭВМ, где нет специальной команды ЛОГИЧЕСКИЙ СДВИГ ВЛЕВО, эта операция может быть заменена сложением содержимого аккумулятора с самим собой по команде ADD A. Если составить макроопределение, то можно присвоить команде сдвига свой собственный мнемонический код:

```
SLL      MACRO
ADD      A
ENDM
```

Имя SLL, указанное в поле метки псевдооперации MACRO, есть имя макрокоманды. Как только эта макрокоманда определена, мнемоническое имя SLL можно использовать так же, как и код любой ма-

шинной команды: каждое вхождение имени SLL ассемблер заменит двоичным кодом команды ADD A.

Макрокоманда может иметь параметры. Например, для логической операции NAND (И—НЕ) можно задать следующее макроопределение:

```
NAND  MACRO  ADDR
      AND    ADDR
      ИНВЕРТИРОВАТЬ АККУМУЛЯТОР
      ENDM
```

Символ NAND есть имя макрокоманды, а символ ADDR, указанный в поле адреса, — параметр макрокоманды. При использовании макрокоманды NAND следует задавать значение адреса. Ассемблер переработает это значение адреса в ту совокупность команд, которая определяет макрокоманду. Например, вместо макровывоза NAND 100 ассемблер вставит команды

```
AND 100
ИНВЕРТИРОВАТЬ АККУМУЛЯТОР
```

Разумеется, макрокоманды могут быть гораздо сложнее, чем в рассмотренных примерах. У макрокоманды может быть несколько параметров и она может состоять из многих команд.

Пример. Сумма элементов массива.

```
SUM  MACRO  TOTAL, ELEM, NUM
      LOAD  # 0
      ЗАГРУЗИТЬ В ИНДЕКСНЫЙ РЕГИСТР NUM
      LOOP  ADD  ELEM, X
      ДЕКРЕМЕНТИРОВАТЬ ИНДЕКСНЫЙ РЕГИСТР
      ПЕРЕЙТИ К LOOP, ЕСЛИ НЕ НУЛЬ
      ЗАПОМНИТЬ РЕЗУЛЬТАТ В TOTAL
      ENDM
```

Макрокоманда SUM имеет три параметра: адрес TOTAL, куда помещается сумма, начальный адрес массива ELEM и адрес SUM, в котором находится длина массива. Программист может воспользоваться макрокомандой, задав в своей программе оператор вида

```
SUM  TOT1, A1, N1
```

Ассемблер расширит этот оператор приведенной ранее последовательностью команд, в которой параметры будут заменены заданными в макровывозе значениями.

Макрокоманды и подпрограммы различаются между собой, хотя похожи по форме и аналогичны по назначению. Ассемблер заменяет каждый макровывоз заданной последовательностью команд. При использовании макрокоманд не требуется использовать команды CALL и RETURN. Подпрограмма представляет собой единственный экземпляр совокупности команд, к которой можно обращаться из различных частей главной программы или других подпрограмм. Подпрограм-

ма хранится в памяти в одном экземпляре. Для обращения к подпрограмме используется команда `CALL`, а для возврата в главную программу — команда `RETURN`.

Подпрограммы

Во многих ассемблерах имеются специальные средства, облегчающие работу с подпрограммами. Ассемблер дает возможность транслировать подпрограммы отдельно. Затем он собирает информацию обо всех ссылках на подпрограмму в главной программе и передает ее специальной программе-загрузчику (называемой *связывающим загрузчиком*), которая заменяет эти ссылки действительными адресами. Если в ассемблере нет средств установления связей, то подпрограммы должны ассемблироваться совместно с основной программой. Некоторые ассемблеры автоматически обрабатывают ссылки на общие подпрограммы, поставляемые изготовителем ЭВМ (называемые *библиотекой* подпрограмм). В состав библиотеки могут входить тригонометрические функции, процедуры ввода-вывода или другие широко используемые программы.

Достоинства и недостатки подпрограмм и макрокоманд

Использование подпрограмм и макрокоманд имеет следующие преимущества:

- 1) программист может написать некоторую последовательность команд 1 раз и многократно использовать ее;
- 2) программы становятся более наглядными и понятными;
- 3) все изменения вносятся в одну последовательность команд, а не в каждое вхождение этой последовательности;
- 4) можно использовать библиотечные программы, другие стандартные программы или последовательности команд, которые ранее были отлажены и протестированы.

По сравнению с подпрограммами макрокоманды имеют то преимущество, что они не требуют выполнения команд перехода, поскольку ассемблер помещает команды, взятые из макроопределения, в том месте, где находится макрокоманда. В результате макрокоманды выполняются быстрее подпрограмм. Кроме того, программа, которая прерывает другую программу, может использовать ту же макрокоманду, которая была прервана. При этом не возникает никаких сложностей, так как оба макровывода порождают две физически различные последовательности команд. Вместе с тем использование макрокоманд приводит к большим затратам памяти, чем использование подпрограмм, поскольку на место каждого макровывода подставляется своя последовательность команд, а подпрограмма присутствует только в одном экземпляре. Макрокоманды обычно используются для относительно коротких последовательностей команд, когда вопрос использования памяти не является главным. Для более длинных последовательностей (десять команд и более) предпочтительно применять технику подпрограмм. Программист должен осмотрительно прибегать к использованию

как макрокоманд, так и подпрограмм, поскольку их однократное использование может приводить к выполнению многих дополнительных команд и большим затратам времени.

Локальные или глобальные переменные

При использовании макрокоманд и подпрограмм возникает проблема использования в них имен, определенных в главной программе, и наоборот. Обычно такое использование недопустимо, если только эти имена не являются параметрами макрокоманды или подпрограммы. Переменная, определенная в некоторой отдельной подпрограмме, называется *локальной* переменной; переменная, которая определена в пределах всей программы, называется *глобальной*. Обычно имена и метки, используемые в макрокомандах и подпрограммах, являются локальными; следовательно, их нельзя использовать за пределами этой макрокоманды или подпрограммы. Так, например, в описанной выше макрокоманде SUM метка LOOP является локальной по отношению к макрокоманде. Отсюда следует, что ни одна команда за пределами макрокоманды не имеет права ссылаться на эту метку. Вместе с тем повторное использование макрокоманды не означает, что эта метка окажется дважды определенной. Здесь опять необходимо предостеречь от возможных недоразумений: метка, которую можно использовать как внутри макрокоманды или подпрограммы, так и за их пределами должна быть включена в список параметров.

Комментарии

Почти все ассемблеры предоставляют возможность задавать комментарии, что позволяет документировать программы. Обычно комментарий в ассемблере отмечается специальным символом или отделяется пробелом. Поле комментария может быть пустым, однако наличие комментариев помогает пониманию программ, поэтому они являются важной частью документации. Комментарии не влияют на объектный код, генерируемый ассемблером.

Переместимость

Некоторые ассемблеры позволяют получить программу на машинном языке, которая является *переместимой*, т. е. может быть помещена загрузчиком в любое место памяти. Ассемблер вырабатывает информацию, которая используется загрузчиком для настройки всех переместимых адресов. Следует избегать использования в программах абсолютных адресов, за исключением тех случаев, когда это действительно необходимо для задания фиксированных ячеек памяти, адресов (подпрограмм) обработки прерываний и т. п. Загрузчик прибавляет значение константы перемещения ко всем адресам, которые не являются абсолютными. Следует избегать выполнения арифметических операций над переместимыми адресами, поскольку такие операции либо могут быть ошибочными (например, суммы, содержащие две константы

перемещения), либо приведут к абсолютным величинам (например, разности). Переместимые подпрограммы удобны потому, что в этом случае можно использовать одни и те же подпрограммы с различными главными подпрограммами и загрузчик может размещать их должным образом в доступной памяти.

Одно- и двухпроходной ассемблеры

Большинство ассемблеров являются *двухпроходными*, так как для получения правильного объектного кода они осуществляют два просмотра исходной программы. Во время первого прохода ассемблер создает таблицу символов и собирает все имена, определенные в программе; во время второго прохода он транслирует программу, используя информацию, собранную при первом проходе. В общем случае символические имена могут быть определены в любом месте программы, поскольку в любом случае ассемблер просматривает всю программу. Однако программисту бывает удобно размещать все определения в начале программы. Большинство двухпроходных ассемблеров временно записывает транслируемую программу на ленту или диск, так что повторный ввод данных с внешнего носителя не требуется. Если рабочая память отсутствует, то двухпроходный ассемблер должен будет дважды ввести программу с перфоленты или перфокарт. Для выдачи результата трансляции на перфоленту или кассету может понадобиться третий проход ассемблера. На рис.4.6 иллюстрируется принцип работы двухпроходного ассемблера.

Первый проход. Создание таблицы символических имен

Программа			Таблица символических имен	
123	LOAD	X1+7	X1	60
124	JUMP	THRU	THRU	
125 GR:	ADD	#1	GR	125
126	DIVIDE	#10	ST	127
127 ST:	STORE	TEMP	TEMP	62

Метка THRU появляется в программе далее

Второй проход. Использование таблицы символических имен

Программа			Программа с числовыми адресами	
123	LOAD	X1+7	LOAD	67
124	JUMP	THRU	JUMP	327
125 GR:	ADD	#1	ADD	#1
126	DIVIDE	#10	DIVIDE	#10
127 ST:	STORE	TEMP	STORE	62

Рис. 4.6. Двухпроходный ассемблер

Кроме двухпроходных ассемблеров часто используется *однопроходный ассемблер*, который работает быстрее, так как должен просматривать программу 1 раз. Однако в однопроходных ассемблерах возникает проблема *ссылок вперед*; часто такие ссылки обрабатывает загрузчик. В общем случае однопроходные ассемблеры предоставляют меньше возможностей, чем двухпроходные. Кроме того, они вносят дополнительные ограничения на способы задания адресов, использование имен, распределение памяти или требуют более сложного загрузчика, который выполняет часть функций ассемблера в процессе загрузки программы в память.

Вход и выход ассемблера. Обычно ассемблеры вводят исходную программу с любого носителя. Очевидно, что скорость процесса ассемблирования зависит от быстродействия устройства ввода (особенно если ассемблер должен читать программу дважды).

Обычно ассемблер выдает на выходе все или некоторые из перечисленных ниже документов:

а) листинг ассемблерной программы вместе с сгенерированным объектным кодом;

б) список ошибок ассемблирования;

в) таблицу символических имен, используемых в программе, с указанием их значений;

г) таблицу перекрестных ссылок, содержащую перечень имен, и список всех команд, в которых они используются;

д) список внешних ссылок (перечень имен подпрограмм или переменных, которые определены за пределами данной программы);

е) список подпрограмм или макрокоманд с указанием их длины.

По требованию ассемблер выдает также копию сгенерированной машинной программы на перфоленте или перфокартах. В некоторых ассемблерах предусмотрена возможность размещения на носителе перед объектным кодом простого самонастраивающегося загрузчика. В результате объектная программа становится самозагружаемой.

Некоторые ассемблеры сразу помещают программу в память и инициализируют ее выполнение с указанной точки входа. Такие ассемблеры называются ассемблерами *типа загрузки и выполнения*. Другие помещают машинную программу на диск или ленту (или оставляют ее в памяти) и ожидают дальнейших команд.

Ошибки. Ассемблеры выдают множество различных сообщений об ошибках. Эти сообщения описаны в соответствующих руководствах по ассемблеру. Наиболее типичные ошибки перечислены в табл. 4.3. Обычно ассемблер идентифицирует ошибку с помощью специальной буквы или цифры и печатает в конце программы список ошибок с указанием номеров строк, к которым они относятся. Иногда ассемблер отмечает строки, в которых обнаружена ошибка, также и в выходном листинге.

Кросс-ассемблер и собственный ассемблер

Ассемблер или компилятор может работать не обязательно на той ЭВМ, для которой он генерирует объектную программу. Ассемблер, работающий на ЭВМ того же типа, на которой будет выполняться про-

Т а б л и ц а 4.3. Наиболее распространенные ошибки в ассемблерных программах

Сообщение об ошибке	Смысл сообщения об ошибке
DEFINED SYMBOL NOT USED (символ определен, но не используется)	Символическое имя было определено, но в программе не используется. Большинство ассемблеров фиксирует эту ситуацию, но продолжает трансляцию программы
ERRONEOUS LABEL (метка не нужна)	Меткой снабжен оператор, который не может быть помечен. К таким операторам обычно относятся ORG, END, IF, ENDF, PAGE, NAME или другие псевдокоманды
ILLEGAL CHARACTER (недопустимый символ)	Некорректный символ в данных определенного типа, например символ, отличный от 0 или 1 в двоичном числе
ILLEGAL FORMAT (неверный формат)	Ошибка формата: неверный разделитель, слишком много операндов или неверный тип операндов
ILLEGAL LABEL (неверная метка)	Использованная метка не допускается правилами ассемблера: недопустимый символ, слишком большая длина, недопустимый первый символ
ILLEGAL NUMERIC (ошибка в числе)	В изображении числа встречается недопустимый символ. Чаще всего это ошибка перфорации
ILLEGAL OPERAND или INVALID REGISTER (недопустимый операнд, недопустимый регистр)	Операнд или регистр указаны неверно: например, указанный регистр не существует или регистр (или операнд) не может указываться в данной команде
ILLEGAL VALUE (недопустимое значение)	Заданное число невозможно разместить в отведенной для него ячейке
INVALID ASSEMBLER OPERATION (неверная псевдокоманда)	Неверно использованная команда ассемблера: например, ENDM встречается без MACRO (или наоборот) или ENDF встречается без IF
INVALID EXPRESSION (неверное выражение)	Ошибка в адресном выражении. Наиболее частыми являются следующие ошибки: два знака операции подряд, несуществующая операция (ошибка перфорации), несовпадающее число открывающихся и закрывающихся скобок, пропущенные параметры, параметры недопустимого типа или неверной длины, отсутствие кавычек или разное число открывающихся и закрывающихся кавычек
LIMIT EXCEEDED (превышение предела) или SYMBOL TABLE OVERFLOW (переполнение таблицы имен)	Число символьческих имен, число внешних ссылок, значение счетчика адреса, число команд или других элементов программы превышает предельное значение для данного ассемблера
MACRO DEPTH EXCEEDED (превышена глубина вложений макро)	Внутри макроопределения имеется другое макроопределение или макроопределение ссылается на себя (рекурсия)
MISSING LABEL (отсутствует метка)	Отсутствует метка у оператора, который должен быть обязательно помечен. Обычно это относится к псевдокомандам DEFINE и MACRO, которые не имеют смысла без метки

Сообщение об ошибке	Смысл сообщения об ошибке
MISSING OPERAND FIELD (нет поля операнда) MULTIPLE DEFINITION (неоднозначное определение)	В команде, требующей указания адреса ячейки памяти, отсутствует операнд Метка или символическое имя определены более 1 раза. Чаще всего это означает, что одна и та же метка использована для разных целей. Эта ошибка может возникнуть, если длинные символические имена имеют одинаковые начальные символы (в большинстве ассемблеров воспринимаются первые пять или шесть символов имени)
UNDEFINED OPERATION CODE (несуществующий код операции)	Несуществующий код операции или несуществующее имя макрокоманды. Обычно это ошибка при наборе данных
UNDEFINED SYMBOL (символическое имя не определено)	Использованное имя в программе не определено: программист забыл определить это имя, или ошибка набора данных, или символ локализован в макрокоманде

грамма, называется *собственным ассемблером* или *резидентным ассемблером*. Ассемблер, работающий на другой ЭВМ, называется *кросс-ассемблером*. Аналогичный смысл имеют термины *собственный компилятор* и *кросс-компилятор*.

На первый взгляд кросс-ассемблер представляется чем-то таинственным. Однако его использование бывает необходимо при составлении микропроцессорных программ, так как для работы собственного ассемблера МП часто не хватает быстродействия, объема памяти, периферийных устройств и программного обеспечения (ПО). Как отмечалось ранее, разработка программ представляет собой задачу, которую лучше всего решать на больших вычислительных системах, имеющих быстродействующие устройства ввода-вывода, массовые ЗУ, редакторы, компиляторы и другие удобные программные и аппаратные средства.

Конечно, кросс-ассемблирование и кросс-компиляция создают дополнительные сложности. Ассемблер не может просто поместить результирующую программу на машинном языке в память и исполнить ее. Вместо этого он выводит программу на перфоленду или другой носитель. После этого необходимо ввести информацию с промежуточного носителя в микро-ЭВМ, на которой программа должна выполняться. В результате необходимо иметь загрузчик и монитор в микро-ЭВМ, а также кросс-ассемблер или кросс-компилятор в большой ЭВМ. Делая выбор между кросс-ассемблером и собственным ассемблером, следует сопоставлять те неудобства и затраты, которые возникают в связи с использованием кросс-ассемблеров, с большим быстродействием и более широкими возможностями больших ЭВМ. Программу необходимо отладить и протестировать на самой микро-ЭВМ. Однако нередко

легче выполнить большую часть работ по созданию программного обеспечения на большой ЭВМ, оставив для микро-ЭВМ этап фактического тестирования.

Размер ассемблера

Размер ассемблера зависит от предоставляемых им возможностей и от допустимого размера таблицы символов. Простейший ассемблер может занимать 8 К ячеек 8-битной памяти, а более развитый ассемблер от 8 К до 16 К ячеек 16-битной памяти. Существует определенная взаимосвязь между количеством памяти, требуемой для работы ассемблера, затратами на разработку ассемблера, наличием дополнительных возможностей в ассемблере и скоростью его работы. При оценке ассемблера следует принимать во внимание все перечисленные факторы. Если система не располагает большой памятью и быстродействующими периферийными устройствами, целесообразно использовать простейший ассемблер.

4.3. ОСОБЕННОСТИ МИКРОПРОЦЕССОРНЫХ АССЕМБЛЕРОВ

Большинство микропроцессорных ассемблеров являются достаточно простыми. У них нет таких широких возможностей, как у ассемблеров, созданных для более мощных ЭВМ типа IBM 370 или PDP-11 фирмы DEC. Некоторые возможности ассемблеров больших ЭВМ в микропроцессорных ассемблерах не нужны, так как микропроцессоры редко используются для реализации больших систем обработки данных.

Сравнение кросс-ассемблеров и собственных ассемблеров

Большинство микропроцессорных ассемблеров существует в двух вариантах: кросс-ассемблер и собственный ассемблер. Причиной этого является сравнительно малое быстродействие микропроцессоров и их преимущественное использование для управления, а не для решения сложных задач обработки текстов и символьной информации, на которые ориентированы классические ассемблеры. К тому же ассемблеру требуется больше памяти и периферийных устройств, чем их обычно имеется в составе микро-ЭВМ.

Некоторые типы собственных ассемблеров могут работать только на специальных системах разработки, базирующихся на том же самом типе микропроцессора. Иногда собственный ассемблер размещается в ПЗУ, и тогда нет необходимости вводить его при каждом использовании. Однако, несмотря на это, собственный ассемблер работает медленно, так как скорость работы системы разработки определяется быстродействием базового микропроцессора. Если в системе нет массовой памяти, то в процессе работы собственного ассемблера выполняются два просмотра программы с повторным вводом ее с внешнего носителя.

Как кросс-ассемблер, так и собственный ассемблер предполагают использование при разработке программ некоторой дополнительной ЭВМ. При этом кросс-ассемблер предполагает использование ЭВМ, отличной от той, для которой создается программа. Объектный код, получаемый на выходе, затем должен быть помещен в память целевой микро-ЭВМ. Собственный ассемблер предполагает наличие системы разработки, имеющей достаточный объем памяти, программное обеспечение и периферийные устройства. При выполнении отладки можно выполнить объектную программу непосредственно на системе разработки, хотя окончательный этап тестирования программы должен выполняться на прототипе. Методы разработки программ для систем, основанных на микропроцессорах, более подробно излагаются в гл. 6.

Почти все кросс-ассемблеры для микропроцессоров написаны на ФОРТРАНе, так как их несложно запрограммировать, и они могут работать на любой ЭВМ, имеющей компилятор с ФОРТРАНа. Однако кросс-ассемблер, запрограммированный на ФОРТРАНе, требует много памяти и работает медленно. Для повышения скорости ассемблирования и уменьшения требуемого объема памяти созданы варианты кросс-ассемблеров, разработанные на языке ассемблера некоторых ЭВМ (чаще всего IBM 370).

Особые характеристики

Некоторые возможности ассемблера особенно удобны при программировании микропроцессоров, поскольку последние имеют короткое слово, программы ПЗУ и используются в системах, требующих обработки прерываний. Среди этих возможностей следует отметить, следующие:

1. Возможность обрабатывать команды длиной в несколько слов. У микропроцессоров имеется много таких длинных команд. Ассемблер не только выбирает правильную длину и корректно устанавливает счетчик команд, но также выделяет и верно адресует длинные операнды.

2. Обеспечение нескольких точек отсчета. Большинство микропроцессорных программ должно иметь несколько точек отсчета, чтобы размещать в нужном месте памяти программы, выполняющие операции сброса и обработки прерываний; а также чтобы обходить области памяти, отведенные для хранения данных или выполнения операций ввода-вывода.

3. Раздельное выделение памяти под постоянные и переменные данные. Большинство микропроцессорных программ предполагает подобное разделение, поскольку постоянные данные хранятся в ПЗУ, а переменные — в ОЗУ. Псевдооперация DATA размещает данные в ПЗУ, а псевдооперация RESERVE выделяет для размещения переменных память в ОЗУ.

Ввод-вывод при ассемблировании

Кросс-ассемблеры для микропроцессоров могут вводить исходную программу с любого носителя, используемого на ЭВМ разработки. Чаще всего ввод осуществляется с перфокарт или перфоленты. Собственные ассемблеры почти всегда вводят данные с перфоленты или мини-кассеты. Как кросс-ассемблеры, так и собственные ассемблеры обычно выводят программу на машинном языке на мини-кассету или на перфоленту. Для тестирования и отладки содержимое перфоленты можно ввести в ОЗУ, его также можно использовать для программирования ПЗУ.

Получение и использование микропроцессорных ассемблеров

Микропроцессорные ассемблеры поставляются изготовителями микропроцессоров. Собственные ассемблеры обычно входят в состав системы разработки. Кросс-ассемблеры могут быть приобретены либо на магнитной ленте, либо в виде колоды перфокарт. Как собственные ассемблеры, так и кросс-ассемблеры для распространенных микропроцессоров поставляются изготовителями микропроцессоров или систем разработки, фирмами, специализирующимися на создании ПО, и другими организациями. Кросс-ассемблеры для таких распространенных мини-ЭВМ, как PDP-8 и PDP-11 фирмы DEC, NOVA фирмы Data General и Hewlett-Packard 2100, являются широко распространенными.

Таким образом, услугами кросс-ассемблера можно воспользоваться на большинстве ЭВМ. К нему можно получить доступ также и через системы разделения времени. За особую плату пользователь системы разделения времени может получить в свое распоряжение массовую память и развитые средства ПО.

4.4. АССЕМБЛЕРЫ МИКРОПРОЦЕССОРОВ INTEL 8080 И MOTOROLA 6800

Ассемблеры для МП Intel 8080 и Motorola 6800 имеются в системах разработки, в системах разделения времени и в ПО многих ЭВМ. В данном параграфе рассмотрим возможности стандартных кросс-ассемблеров, предоставляемых фирмами Intel и Motorola. Другие разработчики предлагают ассемблеры с аналогичными возможностями.

Структура оператора языка ассемблера

В языке ассемблера операторы записываются по формату, показанному на рис. 4.4. Каждый оператор имеет поля метки, кода операции, адреса и комментариев. Поля метки и комментариев являются необязательными. Исключение составляют некоторые псевдооперации, например EQU или MACRO, для которых наличие метки обязательно, так как их назначение состоит в определении некоторого имени. Оба ассемблера допускают запись операторов в свободном формате. При этом в ассемблере для МП Motorola поле метки должно начинаться с колонки 1. В качестве разделителей используются:

В МП Intel 8080:

- 1) двоеточие в поле метки; исключение составляют псевдооперации EQU и MACRO, в которых метка отделяется пробелом;
- 2) пробел после кода операции;
- 3) запятая между операндами в поле адреса;
- 4) точка с запятой перед комментарием. Этот символ может отмечать начало целой строки комментариев.

В МП Motorola 6800:

- 1) пробел после метки;
- 2) пробел после имени аккумулятора в поле кода операции. Имя аккумулятора часто добавляется к коду операции на конце; например, ADDB означает «сложить с содержимым аккумулятора В»;
- 3) запятая между 8-битным смещением и символом X, указывающим индексирование, например ADDB 5,X;
- 4) пробел перед комментарием. Полная строка комментариев начинается с символа «звездочка» (*).

Типичными примерами операторов на языке ассемблера МП Intel 8080 являются:

```
EXTR:    ADI    30    ;Прибавить смещение
          MVI    C,5
          RAR
```

Типичными примерами операторов на языке ассемблера МП Motorola 6800 являются:

```
EXTR    ADDA    #30    Прибавить смещение
          LDAB    #5
          RORA
```

Метки

Оба ассемблера предоставляют широкие возможности для задания меток. В МП Intel 8080 допускаются метки длиной до пяти символов; при этом первым

символом должна быть буква, символ @ или «?». В качестве меток нельзя

использовать имена, зарезервированные для обозначения регистров, кодов операций или псевдокоманд. В МП Motorola 6800 метка может иметь длину до шести символов; первым символом метки должна быть буква. В качестве меток нельзя использовать символы A, B или X, которые зарезервированы для обозначения аккумуляторов и индексного регистра. В обоих ассемблерах запрещено повторное определение меток.

Если длина метки оказывается больше максимально допустимой, ассемблер будет игнорировать лишние символы. Обычно следует избегать использования меток, которые усекаются при трансляции, могут оказаться недопустимыми в других ассемблерах, совпадают с именами кодов команд или псевдоопераций или близки по начертанию к другим меткам (например, ANTI и ANTI)¹. Наилучший способ избежать ошибки — не использовать неясных и двусмысленных имен. Допустимыми практически во всех ассемблерах являются следующие имена:

LAST, SUM, DROP4, CHECK, ADD 15.

¹ Наиболее часто путают букву O и цифру 0, букву I и цифру 1, букву Z и цифру 2.

Ассемблер сразу обнаруживает неверные и многократно определенные имена. При этом, если программист выбрал содержательные обозначения, возникшие ошибки легко исправить.

Псевдокоманды

В табл. 4.4 приведен перечень псевдокоманд, имеющихся в ассемблерах МП Intel 8080 и Motorola 6800. По форме отличаются только те псевдокоманды, которые обеспечивают генерацию в памяти 8- или 16-битных данных или адресов (например, DEFINE BYTE и DEFINE WORD в МП Intel 8080 и FORM CONSTANT BYTE и FORM DOUBLE CONSTANT BYTE в МП Motorola 6800). В МП Motorola 6800 имеется, кроме того, возможность использовать псевдооперацию FORM CONSTANT CHARACTER, которая специально предназначена для занесения в память программ символов в коде ASCII.

Таблица 4.4. Псевдокоманды, используемые в ассемблерах МП Intel 8080 и Motorola 6800

Тип псевдокоманды	Intel 8080	Motorola 6800
DATA	DB — определить байт (8-битные данные) DW — определить слово (16-битные данные)	FCB — сформировать байтную константу (8-битные данные) FDB — сформировать двухбайтную константу (16-битные данные) FCC — сформировать символьную константу (данные в коде ASCII) EQU
DEFINE	EQU SET (может быть изменено) END	END Много операций ORG
END LIST ORIGIN RESERVE	ORG DS — определить память	RMB — зарезервировать область памяти в байтах SPACE
SPACE		

Типичными примерами псевдокоманд МП Intel 8080 являются следующие:

```

ORG 1000
FACT EQU 35      (двоеточие после метки отсутствует)
ZRO: DB 0
EMES: DB 'ERROR' (символы в коде ASCII заключаются в кавычки)
BUFR: DS 100

```

Эквивалентными псевдокомандами в МП Motorola 6800 являются следующие:

```

ORG 1000
FACT EQU 35      (метка должна начинаться в колонке 1)
ZRO FCB 0
EMES FCC (ERROR) (строка символов в коде ASCII может быть ограничена справа и слева любыми одинаковыми разделителями)
BUFP RMB 100

```

Поле адреса

Поле адреса в МП Intel 8080 может быть пустым, как например, в командах STC (установить в 1 признак ПЕРЕНОС) или HLT (ОСТАНОВ). Адресное поле может быть определено несколькими способами:

- 1) как число в шестнадцатиричной, десятичной, восьмеричной или двоичной системе счисления;
- 2) с использованием текущего значения счетчика адреса (символ \$);
- 3) строкой символов в коде ASCII;
- 4) с помощью символических имен;
- 5) с использованием арифметических или логических выражений.

Если не оговорено противное, предполагается, что все числа, используемые в ассемблерной программе, десятичные. В конце записи шестнадцатиричных, восьмеричных или двоичных чисел должна присутствовать соответствующая буква (H — для шестнадцатиричных, O или Q — для восьмеричных, B — для двоичных). Шестнадцатиричные числа должны начинаться с десятичной цифры (0—9), чтобы можно было отличать их от символических имен.

Стандартные примеры использования числовых значений следующие:

- 1) STRT: MVI C,16; Загрузить в регистр C число 16.

Поскольку специальные указания отсутствуют, ассемблер воспринимает 16 как число в десятичной системе счисления.

- 2) M1 EQU OFFH.

Значение имени M1 (минус единица) задано с помощью шестнадцатиричного числа (следует обратить внимание на то, что оно начинается с нуля, чтобы отличить числовое значение от символического имени FFH).

- 3) INIT: LXI B,30DEH; Послать 30DE в регистры B, C.

Данные или адреса, имеющие длину 16 бит, удобно задавать в шестнадцатиричной системе счисления.

- 4); Маски для выделения 4-битных цифр

MASKS:DB 11110000 B, 00001111 B

Маски, предназначенные для выделения двоичных полей, наиболее наглядны в двоичном коде.

В поле адреса можно использовать адрес текущей команды. Он обозначается символом \$ (знак доллара). Команда

JMP \$ + 6

будет транслирована таким образом, что будет обеспечиваться передача управления команде, которая расположена на шесть ячеек дальше, чем та ячейка, с которой начинается сама команда JMP. В МП Intel данный способ адресации не рекомендуется использовать, поскольку в этом микропроцессоре многие команды занимают более одной ячейки памяти программ. В этих условиях программист может легко ошибиться, вычисляя значение смещения по отношению к текущему адресу; кроме того, для читателя будет затруднительно определять адрес перехода. Поскольку в МП Intel 8080 отсутствует относительная адресация, рассматриваемый способ задания адреса не экономит ни память, ни время выполнения. Поэтому лучше при указании адреса воспользоваться символическим именем.

В ассемблере МП Intel 8080 данные можно задавать как символы в коде ASCII (7-битные символы, дополняемые слева нулевым битом).

Ниже приведены характерные примеры использования символов в коде ASCII.

- 1) CPI'E'; Сравнить символ с символом E.

По этой команде содержимое аккумулятора сравнивается с кодом ASCII символа E (45₁₆).

- 2) ; Сообщение об ошибке, если поле длиннее
; допустимого

ERRM: DB' DATA TOO LARGE'

Выдаваемое сообщение хранится как строка символов в коде ASCII, начиная с ячейки с меткой ERRM.

В поле операнда можно использовать любое имя. При этом программист должен быть внимательным и не путать адреса данных и сами данные. Ниже приведено несколько характерных примеров.

1) MASK EQU 00001111B
ANI MASK ; Выделение младших 4 бит

В данном случае имя MASK обозначает ячейку с данными.

2) ALPH EQU 5300H
LDA ALPH; Послать ALPH в аккумулятор

По этой команде в аккумулятор посылается содержимое ячейки ALPH (5300₁₆).

3) PNCH EQU 5
OUT PNCH

В этой команде значение имени PNCH используется как адрес выходного устройства.

Использование содержательных обозначений для всех устройств ввода-вывода делает программу не только более понятной, но и упрощает ее корректировку. Появление нового номера устройства потребует при этом изменения только одного описания, а не многочисленных ссылок на номер соответствующего устройства.

Ассемблер МП Intel 8080 позволяет также использовать в адресных выражениях арифметические и логические операции для всех типов данных. Во всех операциях операнды рассматриваются как 16-битные числа и результат также представляет собой 16-битное число. Допускается использование следующих операций:

«+» : арифметическое сложение,
«-» : арифметическое вычитание или отрицательная величина,
«*» : умножение,
«/» : частное при делении,
MOD : остаток при делении,
NOT : логическая операция НЕ (инверсия),
AND : логическая операция И (поразрядная),
OR : логическая операция ИЛИ (поразрядная),
XOR : логическая операция СЛОЖЕНИЕ ПО МОДУЛЮ 2 (поразрядная),
SHL : логический сдвиг влево,
SHR : логический сдвиг вправо.

Использование скобок позволяет указывать порядок выполнения операций и выражения более понятными. Первым вычисляется самое внутреннее выражение в скобках. Порядок выполнения операций при отсутствии скобок определяется их приоритетом (причем операции одного приоритета выполняются по порядку слева направо):

- 1) ^, /, MOD, SHL, SHR;
- 2) +, -, * ;
- 3) NOT;
- 4) AND;
- 5) OR, XOR.

Это значит, что операции умножения и деления выполняются раньше операций сложения и вычитания, а арифметические операции раньше логических. Для пояснения смысла адресных выражений следует использовать скобки. Следует избегать сложных адресных выражений, так как понять такие выражения трудно, а допустить в них ошибку легко.

Программист должен убедиться, что в результате вычисления выражения получается число, уместящееся в адресном поле данной команды. В табл. 4.5 приведены сведения о разрядности адресов операндов для различных команд.

Ассемблер МП Motorola 6800 допускает использование в адресном поле аналогичных, но несколько более простых выражений. При этом информацию можно задавать в виде.

1) числовых значений в двоичной, десятичной, восьмеричной или шестнадцатеричной системе счисления;

Таблица 4.5. Типы операндов у команд МП Intel 8080

Коды операций	Операнд 1	Операнд 2
ACI, ADI, ANI, CPI, ORI, SBI, SUI, XRI	8-битный непосредственный операнд	—
ADC, ADD, ANA, CMP, DCR, INR, ORA, SBB, SUB, XRA	Регистр	—
CALL, CC, CM, CNC, CNZ, CP, CPE, CPO, CZ, JC, JM, JMP, JNC, JNZ, JP, JPE, JPO, JZ, LDA, LHLD, SHLD, STA	16-битный прямой адрес	—
CMA, CMC, DAA, DI, EI, HLT, NOP, PCHL, RAL, RAR, RC, RET, RLC, RM, RNC, RNZ, RP, RPE, RPO, RRC, RZ, SPHL, XCHG, XTHL	Отсутствует	—
DAD, DCX, INX, POP, PUSH	Регистровая пара	—
IN, OUT	8-битный номер устройства	—
LDAX, STAX	Регистровая пара (только В или D)	—
LXI	Регистровая пара	16-битный непосредственный операнд
MOV	Регистр	Регистр
MVI	Регистр	8-битный непосредственный операнд
RST	3-битный адрес (00b ₂ b ₀ 000)	—

2) текущего значения счетчика адреса (*);

3) символьных строк (в коде ASCII);

4) имен;

5) арифметических выражений.

Тип числовой константы определяется следующими специальными символами, стоящими перед ее изображением:

\$ — шестнадцатеричная,

% — двоичная,

@ — восьмеричная.

Непомеченные числовые значения воспринимаются ассемблером как десятичные. Кроме того, с помощью апострофа (') обозначаются 7-битные символы в коде ASCII (старший бит устанавливается равным нулю), а с помощью символа # — помечается непосредственный операнд.

Ассемблер МП Motorola 6800 допускает только арифметические выражения, использующие операции «+», «-», «*» (умножение) и «/». Он вычисляет выражения слева направо: скобки не допускаются и все операции имеют одинаковый приоритет. Результат операции — целое число. Таким образом, ассемблер МП Motorola 6800 допускает только простейшие арифметические выражения. Это не является существенным ограничением так как уже отмечалось что сложные выражения используются редко.

В любом языке ассемблера вычисление выражений, которые записаны в программе, осуществляется в процессе ассемблирования, а не во время выполнения программы. Так, например, ассемблер может выполнить операцию деления и умножения, хотя ни в МП Motorola 6800, ни в МП Intel 8080 нет соответствующих команд.

Ассемблер МП Intel 8080 позволяет предусмотреть условное ассемблирование с помощью псевдоопераций IF и ENDIF; в МП Motorola 6800 такая возможность не предусмотрена. В псевдооперации IF выражение записывается в соответствии с изложенными правилами. Если значение соответствующего выражения на этапе ассемблирования оказывается не равным нулю, то ассемблер включит в результирующую программу операторы, заключенные между псевдооперациями IF и ENDIF. Если выражение оказывается равным нулю, то соответствующие операторы в программу не включаются. Приведем несколько характерных примеров.

Пример 1. Обратный или дополнительный код.

```
CMA
IF      TWOS
INR    A      ; Прибавить 1 для получения дополнительного кода
ENDIF
```

Если переменная TWOS = 0, то ассемблер включит в программу только команду CMA, которая формирует обратный код. Если TWOS ≠ 0, ассемблер включит также команду INR A, выполнение которой превратит обратный код в дополнительный. С помощью одной и той же ассемблерной программы можно выполнить арифметические операции в обратном или дополнительном коде. Выбор режима осуществляется с помощью переменной TWOS, значение которой устанавливается перед ассемблированием.

Пример 2. 8- или 16-битное сложение.

```
IF      L8;      8-битное сложение
LDA     OP1
MOV     B,A
LDA     OP2
ADD     B
STA     RES
ENDIF
IF      L16      ; 16 битное сложение
LHLD    OP1
XCHG
LHLD    OP2
DAD
SHLD    RES
ENDIF
```

Данная программа позволяет выбрать 8- или 16-битный вариант сложения в зависимости от значений переменных L8 и L16. Если L8 ≠ 0 и L16 = 0, ассемблер транслирует команды, реализующие 8-битное сложение. Если L8 = 0 и L16 ≠ 0, будут транслированы команды, обеспечивающие сложение 16-битных чисел.

Программистам не рекомендуется злоупотреблять средствами условного ассемблирования. Их использование делает программы более запутанными и усложняет их отладку. Лучше написать две отдельные программы для двух различных случаев, чем прибегать к сложным средствам условного ассемблирования. Средства условного ассемблирования целесообразно использовать для облегчения отладки в первоначальных вариантах программы.

Макрокоманды

В МП Intel 8080 имеются макросредства, в МП Motorola 6800 они не предусмотрены. Макрокоманды можно использовать для того, чтобы ввести удобные обозначения для существующих команд, расширить систему команд, обозначить одним оператором целую совокупность команд. Для каждой макрокоманды требуется составить определение и присвоить ей уникальное имя. Макрокоманды не могут включать в себя ни определения других макрокоманд, ни ссылки на самих себя. Вместе с тем они могут содержать ссылки на другие макрокоманды. Для

описания макрокоманд в МП Intel 8080 используются псевдооперации MACRO, ENDM. Приведем несколько примеров макрокоманд

Пример 1. Обнулить аккумулятор и установить признак ПЕРЕНОС.

```
CLR    MACRO
      SUB    A
      ENDM
```

Использование команды SUB A позволяет просто выполнить очистку аккумулятора и установку в 1 признака ПЕРЕНОС. Введение с помощью макро-средств мнемоники CLR делает использование этой команды более наглядным.

Пример 2. Операция NOR (ИЛИ — НЕ).

```
NOR    MACRO    REG
      ORA      REG
      CMA
      ENDM
```

В макрокоманде NOR используется операнд REG. Определив эту макрокоманду, можно использовать логическую операцию NOR так же, как любую другую, входящую в состав системы команд МП Intel 8080. Например, NOR C выполняет логическую операцию ИЛИ — НЕ над содержимым аккумулятора и регистра C.

Пример 3. Косвенная адресация (ЗАГРУЗИТЬ АККУМУЛЯТОР С КОСВЕННОЙ АДРЕСАЦИЕЙ).

```
LIND    MACRO    ADDR
      LHL    ADDR
      MOV    A, M
      ENDM
```

С помощью приведенного макроопределения в МП Intel 8080 можно реализовать косвенную адресацию.

При использовании макросредств программист должен соблюдать следующие правила:

- 1) каждая макрокоманда должна иметь уникальное имя;
- 2) макроопределение должно начинаться с псевдооперации MACRO и заканчиваться псевдооперацией ENDM;
- 3) имена, определенные в макрокоманде, локализованы в ней и не определены в главной программе;
- 4) макроопределения не должны включать других макроопределений.

Вывод результатов ассемблирования

Ассемблеры МП Intel 8080 и Motorola 6800 выдают программу на перфоленте. Ассемблер МП Intel 8080 использует так называемый BNPF-формат. Символы в этом формате имеют следующее значение:

- B — начало 8-битного слова,
- N — двоичный ноль,
- P — двоичная единица,
- F — конец 8-битного слова.

Данные в этом формате используются для программирования ПЗУ и ППЗУ фирмы Intel.

Использование ассемблеров

Ассемблер МП Intel 8080 включен в состав Intel 8/Mod 80 и системы разработки MDS фирмы Intel. Имеются также собственные ассемблеры систем разработки для МП Intel 8080, поставляемые другими фирмами. Существуют различные версии кросс-ассемблеров, написанные на языке ФОРТРАН и поставляемые фирмой Intel и другими, которые могут работать на ЭВМ, имеющих транслятор с ФОРТРАНа. Кросс-ассемблеры есть также на большинстве систем разделения времени.

Ассемблер МП Motorola 6800 есть в системе разработки Motorola Exerciser. Кроме того, он есть на других системах разработки, в системах, разделяющих время и в удобном виде на большинстве ЭВМ, имеющих компилятор с ФОРТРАНА.

4.5. ВЫВОДЫ

Программы могут разрабатываться на языках различных уровней. Машинный язык и язык ассемблера не требуют сложного программного и аппаратного обеспечения и позволяют получать очень эффективные программы. Однако эти языки являются машиннозависимыми и программирование на них трудоемко и занимает много времени. Процедурно-ориентированные языки, или языки, высокого уровня, являются машиннонезависимыми и значительно упрощают программирование, но требуют больших программных и аппаратных ресурсов и обычно порождают неэффективные программы. В настоящее время большинство систем, использующих микропроцессоры, программируется на языке ассемблера.

Основная задача ассемблера состоит в преобразовании мнемонических кодов команд в их двоичные эквиваленты. Кроме того, большинство ассемблеров предоставляет другие средства: использование символических имен, комментариев, арифметические операции на этапе ассемблирования, условное ассемблирование, макрокоманды. Пользователи микропроцессоров обычно применяют большинство этих средств только в их простейшем варианте. Стандартные микропроцессорные ассемблеры достаточно просты и не предоставляют широких возможностей при составлении программ. Несмотря на это, для их использования требуется другая ЭВМ (кросс-ассемблер) или специальная система разработки (собственный ассемблер).

ГЛАВА ПЯТАЯ

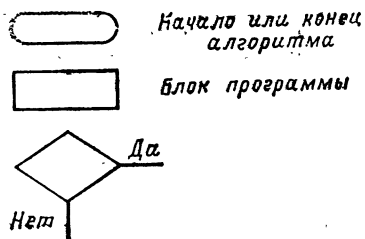
ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ АССЕМБЛЕРА

В данной главе рассмотрены особенности программирования на языке ассемблера микропроцессоров Intel 8080 и Motorola 6800. Вместе с тем изложенные в ней общие приемы программирования применимы и при программировании микропроцессоров других типов с учетом особенностей, обусловленных своеобразием методов адресации и системы команд. В § 5.1 приведены примеры простых программ на языке ассемблера. В следующих параграфах рассмотрена организация циклов, работа с массивами, реализация арифметических операций, обработка символьной информации и использование подпрограмм. В главе приведены примеры подробных листингов программ на языке ассемблера и в машинных кодах, блок-схемы и трассы выполнения программ.

Все приведенные примеры удовлетворяют следующим общим соглашениям.

1. Используются правила записи ассемблерных программ для МП Intel 8080 и Motorola 6800 в соответствии с их описанием в гл. 4.
2. Для задания команд и адресов используется шестнадцатиричная система счисления.
3. При задании исходных данных используется та система счисления, в которой данные представляются наиболее наглядно. Все десятичные числа изображаются в соответствии с правилами ассемблеров. В языке ассемблера МП Intel 8080 символами H и B обозначают соответственно шестнадцатиричные и двоичные числа; при использо-

Рис. 5.0. Условные обозначения, применяемые в блок-схемах



вании ассемблера МП Motorola 6800 шестнадцатиричные числа отмечаются символом \$, двоичные — символом %.

4. Для каждой программы, содержащей разветвления вычислительного процесса, приводится блок-схема. Для изображения блок-схем используются стандартные обозначения, приведенные на рис. 5.0.

5. Приводятся трассы выполнения программ. Трасса программы иллюстрирует, каким образом в результате выполнения каждой команды меняется содержимое регистров и ячеек памяти.

6. Листинг каждой ассемблерной программы снабжен комментариями. Комментарии более пространны, чем это принято при создании реальной документации, но выдержаны в общепринятом стиле. В гл. 6 рассматриваются способы составления комментариев и другие формы документирования.

В языке ассемблера МП Intel 8080 каждый комментарий начинается с символа «точка с запятой»; в соответствии с правилами ассемблера МП Motorola 6800 комментарий отделяется от команды пробелом и, если комментарий занимает отдельную строку, начинается с символа «звездочка».

7. При подготовке программы основной упор был сделан на их ясность, а не на достижение максимального быстродействия или экономии памяти. Программы составлены достаточно эффективно, однако без использования приемов, позволяющих любой ценой минимизировать время выполнения или затраты памяти. В гл. 6 рассматриваются методы достижения компромиссов между объемом памяти и быстродействием программ.

8. Основное внимание уделено типовым задачам, решаемым с помощью микропроцессоров, и широко используемым командам.

9. Рассмотрена работа только с наиболее часто используемыми признаками: ПЕРЕНОС, ЗНАК и НУЛЬ в МП Intel 8080, ПЕРЕНОС, ОТРИЦАТЕЛЬНЫЙ и НУЛЬ в МП Motorola 6800.

Остальные признаки (ЧЕТНОСТЬ и ВСПОМОГАТЕЛЬНЫЙ ПЕРЕНОС в МП Intel 8080 и ПЕРЕПОЛНЕНИЕ и ПОЛУПЕРЕНОС в МП Motorola 6800) рассмотрены только в связи с их конкретным использованием в программах.

10. Соблюдены стандартные соглашения по использованию памяти: все короткие программы располагаются в памяти с ячейки 0, а для размещения данных используют ячейки начиная с адреса 40₁₆. В результате эти программы смогут выполняться на любой микро-ЭВМ, если пользователь позаботится о корректном задании соответствующих адресов.

11. Каждая программа заканчивается командой останова или программного прерывания: HLT (HALT—ОСТАНОВ) в МП Intel 8080

и SWI (SOFTWARE INTERRUPT—ПРОГРАММНОЕ ПРЕРЫВАНИЕ) в МП Motorola 6800.

12. Для синхронизации используются тактовые генераторы, работающие со стандартной частотой 1 МГц в МП Motorola 6800 и 2 МГц в МП Intel 8080. При этом время выполнения команд может быть указано в микросекундах.

5.1. ПРОСТЫЕ ПРОГРАММЫ

Пример 1. Сложение 8-битных чисел.

По-видимому, самой простой программой, которую можно себе представить, является программа сложения двух чисел. Предположим, что нужно сложить содержимое ячеек 40₁₆ и 41₁₆ и поместить результат в ячейку 42₁₆. В этом примере для простоты не будем учитывать возможность возникновения переноса.

На большинстве ЭВМ такое сложение будет выполняться за несколько шагов.

Шаг 1. Послать в аккумулятор содержимое ячейки 40₁₆.

Шаг 2. Прибавить к содержимому аккумулятора содержимое ячейки 41₁₆.

Шаг 3. Послать содержимое аккумулятора в ячейку 42₁₆.

Motorola 6800 (пример 1)

В МП Motorola 6800 программа сложения будет в точности повторять эти шаги.

М6800 Пример 1

Сложение двух чисел

LDAA \$40 Загрузить операнд 1
ADDA \$41 Прибавить операнд 2
STAA \$42 Запомнить результат
SWI

Рассмотрим выполнение программы по шагам.

1. LDAA \$ 40

Шестнадцатиричный адрес ячейки памяти	Мнемонический код операции	Шестнадцатиричный код содержимого памяти
00	LDAD \$ 40	96
01		40
02	ADAA \$ 41	9B
03		41
04	STAA \$ 42	97
05		42
06	SWI	3F

Рис. 5.1. Результат ассемблирования программы сложения для МП Motorola 6800

Исходное состояние:

Память данных		Регистры	
40	05	PC	0000
41	03		

После выполнения первой команды, LDAA \$40:

Память данных		Регистры	
40	05	PC	0002
41	03	A	05

Признаки

Нуль ☐ Отрицательный ☐

Команда, LDAA не изменяет значения признака ПЕРЕНОС

После выполнения второй команды, ADDA \$41:

Память данных		Регистры	
40	05	PC	0004
41	03	A	08

Признаки

Нуль ☐ Отрицательный ☐ Перенос ☐

После выполнения третьей команды, STAA \$42:

Память данных		Регистры	
40	05	PC	0006
41	03	A	08
42	08		

Признаки

Нуль ☐ Отрицательный ☐

Команда STAA не изменяет значения признака ПЕРЕНОС

Рис. 5.2. Трасса выполнения программы сложения на МП Motorola 6800

Данная команда посылает в аккумулятор А содержимое ячейки 40. Поскольку ячейка 40 находится на нулевой странице (ее адрес меньше 100_{16}), можно использовать формат этой команды с прямой адресацией (96_{16}), при котором она занимает два слова программной памяти и выполняется за три такта (за 3 мкс в МП Motorola 6800 со стандартной частотой тактового генератора 1 МГц).

2. ADDA \$ 41

Данная команда прибавляет к содержимому аккумулятора А содержимое ячейки 41. Команда имеет формат с прямой адресацией ($9B_{16}$), занимает две ячейки программной памяти и выполняется за 3 мкс.

3. STAA \$42

По этой команде содержимое аккумулятора А посылается в ячейку 42. Эта команда прямого формата (97_{16}) занимает два слова программной памяти и выполняется за 3 мкс.

4. SWI

Этой командой завершаются примеры всех программ для МП Motorola 6800.

Используя таблицу кодов команд, можно выполнить ассемблирование этой программы вручную (рис. 5.1). На рис. 5.2 приведена трасса выполнения данной программы при условии, что первоначально в ячейке 40 хранилось число 5, в ячейке 41 — число 3, а содержимое счетчика команд было равно 0. Следует обратить особое внимание на то, как выполнение каждой команды меняет содержимое счетчика команд и признаки. Каждая из первых трех команд занимает две ячейки памяти программ: в одной ячейке хранится код операции, в другой — 8-битный адрес ячейки на нулевой странице.

Intel 8080 (пример 1)

Программа для МП Intel 8080 выглядит сложнее, поскольку в МП Intel 8080 невозможно выполнить сложение с использованием прямой адресации. В результате требуется вспомогательная команда для пересылки одного операнда в РОН. Последующая команда сложения может непосредственно адресовать регистр. В следующем примере будет показано, как обойти это неудобство при использовании косвенной регистровой адресации. В данном случае программа состоит из следующих шагов:

Intel 8080 пример 1
Сложение двух чисел
LDA 40H ; Загрузить операнд 1
MOV B, A
LDA 41H ; Загрузить операнд 2
ADD B ; Сложить операнды
STA 42H ; Сохранить результат
HLT

Программа для МП Intel 8080 имеет длину пять команд, в то время как программа для МП Motorola 6800 — только три. Действительно, программа для МП Intel 8080 оказывается эффективней при обработке больших совокупностей данных, чем при обработке отдельных чисел. Причиной этого является то, что регистровая прямая и косвенная адресации эффективны только тогда, когда содержимое регистра используется многократно.

Шестнадцатиричный адрес ячейки памяти	Мнемонический код операции	Шестнадцатиричный код содержимого памяти	Шестнадцатиричный адрес ячейки памяти	Мнемонический код операции	Шестнадцатиричный код содержимого памяти
00	LDA 40H MOV B, A	3A	06	ADD B STA 42H HLT	00
01		40	07		80
02		00	08		32
03		47	09		42
04		3A	0A		00
05	LDA 41H	41	0B		76

Рис. 5.3. Результат ассемблирования программы сложения для МП Intel 8080

Рассмотрим выполнение программы для МП Intel 8080 по шагам, не обращая внимание на команды, встречающиеся ранее.

1. LDA 40H

По этой команде содержимое ячейки памяти с адресом 40 посылается в аккумулятор. Поскольку в Intel 8080 отсутствует специальная адресация нулевой страницы, в команде должен быть задан полный 16-битный адрес. Команда LDA занимает три ячейки памяти и выполняется за 13 тактов (или за 6,5 мкс на МП Intel 8080 со стандартной тактовой частотой 2 МГц).

2. MOV B,A

По этой команде в регистр B посылается содержимое аккумулятора. Команда занимает одну ячейку памяти и выполняется за 2 мкс.

3. ADD B

По этой команде к содержимому аккумулятора прибавляется содержимое регистра B. Команда занимает одну ячейку памяти и выполняется за 2 мкс.

4. STA 42H

По этой команде содержимое аккумулятора посылается в ячейку 42. Команда занимает три ячейки памяти и выполняется за 6,5 мкс.

5. HLT

Эта команда является последней во всех примерах программ для МП Intel 8080.

Эту программу можно ассемблировать вручную (рис. 5.3), используя таблицу кодов операций МП Intel 8080. Трасса выполнения программы показана на рис. 5.4. Программа выполняется при тех же начальных условиях, что и ранее: (40) = 5, (41) = 3, (PC) = 0. Заметим, что в МП Intel 8080 на значение признаков влияет только выполнение команд ADD (сложить). Следует обратить также внимание на то, что содержимое счетчика команд меняется с непостоянным шагом, так как команды имеют различную длину.

Команды LDA и STA требуют задания 16-битного адреса. В МП Intel 8080 младшие 8 бит 16-битного адреса помещаются в первой ячейке (младший адрес), а старшие 8 бит — во второй ячейке (старший ад-

рес); этот метод задания адреса отличается от принятого в МП Motorola 6800 и почти во всех других ЭВМ.

Пример 2. Выделение шестнадцатиричной цифры.

При использовании любой ЭВМ возникает проблема обработки данных, длина которых не совпадает с размером ячейки памяти. В ЭВМ за один прием всегда обрабатывается ячейка или байт. Но как выйти из положения, если информация занимает не всю ячейку? С помощью 1,

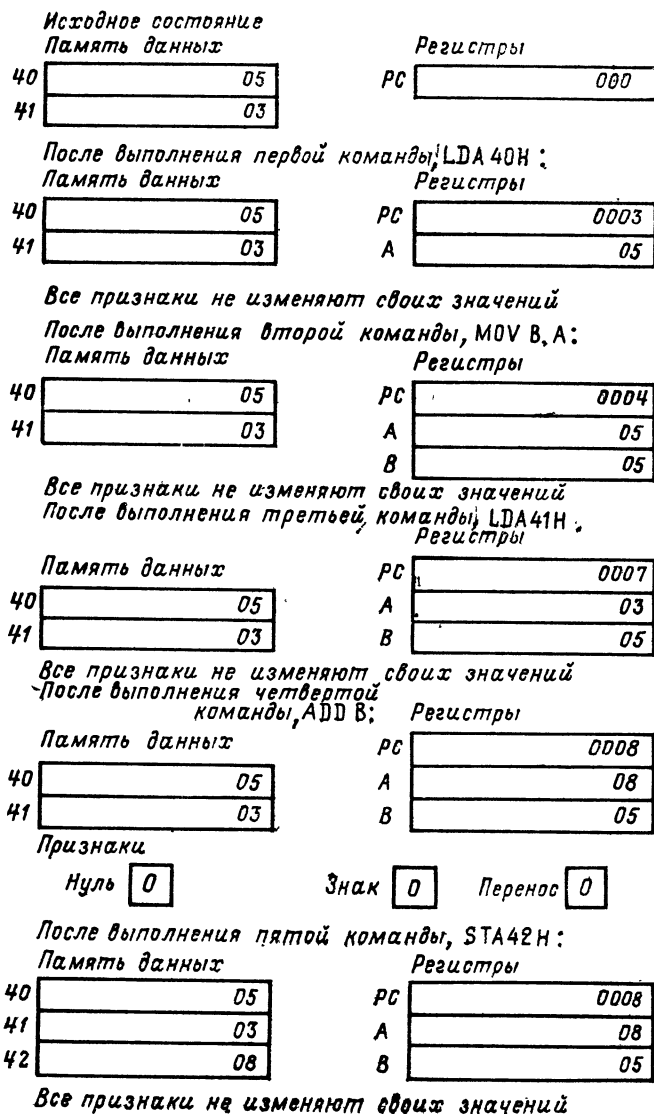


Рис. 5.4. Трасса выполнения программы сложения на МП Intel 8080

2 или 4 бит можно кодировать состояние, задаваемое тумблером, индикатором, клавишей на пульте или двигателем. Вместе с тем, что делать, если данные не умецаются в одной ячейке. Для задания многозначного десятичного числа, управляющего сообщения или информации о совокупности переключателей или индикаторов может потребоваться много бит. Рассмотрим сначала подобную ситуацию на простом примере и перейдем к более сложному в следующем параграфе.

Чтобы обрабатывать короткие информационные поля, программа должна удалить неиспользуемые биты ячейки памяти, чтобы они не влияли на результат выполнения операций. Такое удаление может быть выполнено с помощью логической операции И. Предположим, например, что нас интересуют только младшие 4 бита (младшая шестнадцатиричная цифра) ячейки памяти. Приведенная ниже программа будет выбирать содержимое ячейки 40₁₆, выделять младшие 4 бита и посылать их по адресу 41₁₆. Выделение будет выполняться в следующем порядке.

Шаг 1. Послать в аккумулятор содержимое ячейки памяти 40₁₆.

Шаг 2. Выполнить над содержимым аккумулятора и маской 00001111 (двоичной) логическую операцию И.

В результате младшие 4 бита исходного слова сохранятся без изменений (поскольку $0 \wedge 1 = 0$ и $1 \wedge 1 = 1$), а старшие биты станут равными нулю (поскольку $0 \wedge 0 = 0$ и $1 \wedge 0 = 0$).

Шаг 3. Запомнить содержимое аккумулятора по адресу 41₁₆.

Motorola 6800 (пример 2)

Программа выделения для МП Motorola 6800 состоит из трех команд.

M6800 пример 2

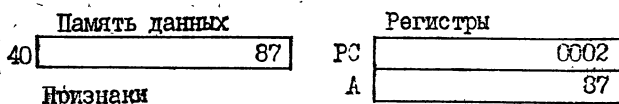
Выделение шестнадцатиричной цифры

LDAА	\$ 40	Загрузить операнд 1
ANDA	#%00001111	Обнулить старшие 4 бита
STAA	\$ 41	Сохранить результат
SWI		

Шестнадцатиричный адрес ячейки памяти	Мнемонический код операции	Шестнадцатиричный код содержимого памяти
00	LDDA \$ 40	96
01		40
02	ANDA # % 00001111	84
03		0F
04	STAA \$ 41	97
05		41
06	SWI	3F

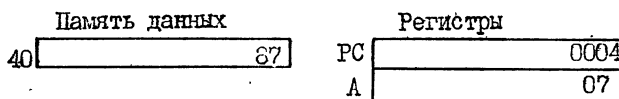
Рис. 5.5. Результат ассемблирования программы выделения шестнадцатиричной цифры для МП Motorola 6800

После выполнения первой команды, LDA #40:



Нуль ☐ 0 Отрицательный ☐ 1

После выполнения второй команды, ANDA #00001111



Нуль ☐ 0 Отрицательный ☐ 0

Значение признака ПЕРЕНОС не изменяется

Рис. 5.6. Трасса выполнения программы выделения шестнадцатиричной цифры на МП Motorola 6800

Единственной новой командой в данной программе является команда ANDA #00001111. Она выполняет логическую операцию И над двоичным числом 00001111 и содержимым аккумулятора А. Следует обратить внимание на символ #, который указывает, что задан непосредственный операнд, и на символ %, который означает, что этот операнд — двоичное число. Данная команда с непосредственным операндом (84₁₆) занимает две ячейки памяти и выполняется за 2 мкс.

Ассемблированный вариант данной программы показан на рис. 5.5, а трасса ее выполнения — на рис. 5.6. Предполагается, что в ячейке 40₁₆ находится шестнадцатиричное число 87. Эта программа аналогична программе сложения из примера 1.

Intel 8080 (пример 2)

При выполнении операции выделения в программе для МП Intel 8080 используется возможность применения косвенной регистровой адресации, которая в программе на языке ассемблера указывается кодом регистра М. Программист может использовать регистр М точно так же, как и все остальные; однако фактически МП Intel 8080 выбирает данные из ячеек памяти, которые адресуются регистрами Н и L. Регистры Н и L используются в качестве указателей ячеек памяти или данных, поскольку они содержат не сами данные, а их адреса. Изменяя содержимое регистров Н и L, изменяем адресный указатель. Например, выполнение команды INX Н приводит к тому, что указатель будет ссылааться на ячейку памяти с большим адресом. Разумеется перед ис-

пользованием регистра для косвенной адресации содержимое регистров Н и L должно быть установлено с помощью некоторой команды (чаще всего команды LXI). Программа выделения имеет следующий вид:

Intel 8080 пример 2

Выделение шестнадцатичной цифры

LXI H, 40H

MOV A, M ; Загрузить данные

ANI 00001111B ; Обнулить старшие 4 бита

INX H

MOV M, A ; Сохранить результат

HLT

В этом примере программа также состоит из пяти команд в отличие от программы из трех команд для МП Motorola 6800. Косвенная адресация с помощью регистров оказывается эффективной только при обработке массивов данных.

Рассмотрим выполнение программы по шагам.

1. LXI H, 40H

По этой команде в регистровую пару (в данном случае Н и L) посылается содержимое указанных во втором операнде двух ячеек памяти программ. Содержимое ячейки, адрес которой следует сразу за кодом операции, посылается в регистр L, а содержимое следующей ячейки — в регистр H. Команда занимает три ячейки памяти и выполняется за 5 мкс. Заметим, что по этой команде в регистровую пару Н и L посылается 16-битный адрес 0040.

2. MOV A, M

По этой команде в аккумулятор посылается содержимое ячейки памяти, адрес которой указан в Н и L. Команда занимает одну ячейку памяти и выполняется за 3,5 мкс.

3. ANI 00001111B

По этой команде выполняется логическая операция И над двоичным числом 00001111 и содержимым аккумулятора. Команда занимает две ячейки памяти и выполняется за 3,5 мкс.

4. INX H

По этой команде 16-битный адрес, размещенный в регистровой паре Н и L, увеличивается на единицу. Команда занимает одну ячейку памяти и выполняется за 2,5 мкс.

5. MOV M, A

Команда выполняет действия, обратные действию команды MOV A, M.

Результат ассемблирования данной программы показан на рис. 5.7; трасса выполнения программы представлена на рис. 5.8. Были заданы те же начальные условия, что и в программе для МП Motorola 6800: (40) = 87, (PC) = 0. Следует обратить внимание на то, как работают команды, использующие косвенную адресацию с помощью регистров (MOV, A, M и MOV M, A). Эти команды целесообразно использовать вместо LDA и STA, если данные хранятся в смежных ячейках памяти, так как в результате уменьшаются затраты времени и памяти.

Шестнадцатиричный адрес ячейки памяти	Мнемонический код операции	Шестнадцатиричный код содержимого памяти
00	LXI H, 40H	21
01		40
02		00
03	MOV A, M	7E
04	ANI 00001111B	E6
05		0F
06	INX H	23
07	MOV M, A	77
08	HLT	76

Рис. 5.7. Результат ассемблирования программы выделения шестнадцатиричной цифры для МП Intel 8080

Следует также обратить внимание на то, что только команда ANI изменяет состояние признаков; команда INX на состояние признаков влияния не оказывает. Данная программа заметно короче программы, приведенной на рис. 5.3, в которой не используется косвенная адресация с помощью регистров.

Пример 3. Разделение слова на части.

Пример 2 можно усложнить, потребовав, чтобы программа выделяла из исходного слова две шестнадцатиричные цифры и помещала их в отдельные ячейки памяти. Новая программа, как и раньше, посылает четыре младших разряда в ячейку памяти 41, а четыре старших разряда — в ячейку памяти 42. Программа помещает четыре старших бита исходного слова в младшие разряды ячейки 42, чтобы затем их можно было обработать как обычное число. Операция разделения слова на части выполняется в следующей последовательности.

Шаг 1. Послать в аккумулятор содержимое ячейки 40₁₆.

Шаг 2. Выполнить логическую операцию И над содержимым аккумулятора и двоичной константой 00001111.

Шаг 3. Послать содержимое аккумулятора в ячейку 41₁₆.

Шаг 4. Послать в аккумулятор содержимое ячейки 40₁₆.

Шаг 5. Логически сдвинуть содержимое аккумулятора на 4 бита вправо.

Шаг 6. Послать содержимое аккумулятора в ячейку 42₁₆.

Операция ЛОГИЧЕСКИЙ СДВИГ ВПРАВО обнуляет старшие 4 бита аккумулятора, перемещая их прежнее содержимое в младшие 4 бита,

Motorola 6800 (пример 3)

Программа разделения исходного слова на две части для МП Motorola 6800 имеет вид:

M6800 пример 3

После выполнения первой команды, LXI H, 40H :

Память данных		Регистры	
40	87	PC	0003
Ни один из признаков не меняет своего значения		H	00
		L	40

После выполнения второй команды, MOV A, M :

Память данных		Регистры	
40	87	PC	0004
После выполнения третьей команды, ANI 00001111B :		A	87
		H	00
		L	40

Память данных		Регистры	
40	87	PC	0006
Признаки		A	07
		H	00
		L	40

Нуль ☐ Знак ☐ Перенос ☐

Обратите внимание, что команда ANI обнуляет в МП Intel 8080 признак ПЕРЕНОС (точно так же влияют на него все логические команды)

После выполнения четвертой команды, INX H :

Память данных		Регистры	
40	87	PC	0007
Ни один из признаков не меняет своего значения		A	07
		H	00
		L	41

После выполнения пятой команды, MOV M, A :

Память данных		Регистры	
40	87	PC	0008
41	07	A	07
Ни один из признаков не меняет своего значения		H	00
		L	41

Рис. 5.8. Трасса выполнения программы выделения шестнадцатирочной цифры на МП Intel 8080

Шестнадцатиричный адрес ячейки памяти	Мнемонический код операции	Шестнадцатиричный код содержимого памяти
00	LDAA \$ 40	96
01		40
02	ANDA # % 00001111	84
03		0F
04	STAA \$ 41	97
05		41
06	LDAA \$ 40	96
07		40
08	LSRA	44
09	LSRA	44
0A	LSRA	44
0B	LSRA	44
0C	STAA \$42	97
0D		42
0F	SWI	3F

Рис. 5.9. Результат ассемблирования программы разделения слова для МП Motorola 6800

Разделение слова на две части

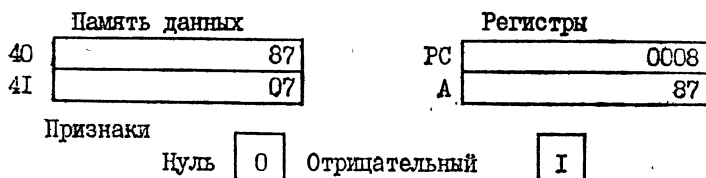
LDAA \$40	Загрузить данные
ANDA # %00001111	Замаскировать старшую цифру
STAA \$41	Сохранить младшую цифру
LDAA \$40	
LSRA	Четырехкратный
LSRA	сдвиг вправо
LSRA	самой старшей
LSRA	цифры
STAA \$42	Сохранить старшую цифру
SWI	

Единственной новой командой в данной программе является команда LSRA, которая выполняет логический сдвиг аккумулятора А на один бит вправо, т. е. обнуляет самый старший значащий бит. Команда LSRA занимает одну ячейку памяти и выполняется за 2 мкс. Результат ассемблирования этой программы показан на рис. 5.9, на рис. 5.10 показано содержимое регистров и памяти после логического сдвига вправо на один разряд. Попробуйте с карандашом в руках проследить, как будет меняться содержимое ячеек и регистров при выполнении оставшейся части программы.

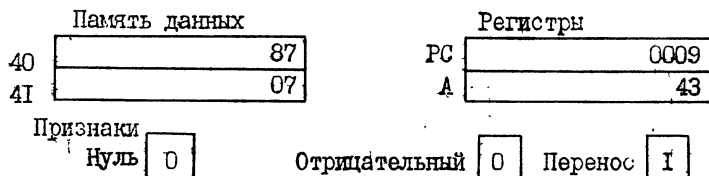
Intel 8080 (пример 3)

Программа разделения слова на части для МП Intel 8080 несколько длиннее соответствующей программы для МП Motorola 6800, поскольку в МП Intel 8080 отсутствует команда ЛОГИЧЕСКИЙ СДВИГ ВПРАВО.

После выполнения четвертой команды, LDA A S 40:



После выполнения пятой команды, LSR A:



Прежнее значение самого младшего бита помещено в признак ПЕРЕНОС

Рис. 5.10. Трасса выполнения программы разделения слова на МП Motorola 6800

Однако в МП 8080 можно выполнить требуемое действие (логический сдвиг вправо на 4 бита) с помощью четырехкратного выполнения команды ЦИКЛИЧЕСКИЙ СДВИГ ВПРАВО и команды ЛОГИЧЕСКОЕ И, которая маскирует старшую (шестнадцатиричную) цифру. Программа обеспечивает сохранение выбранных из памяти данных в PОН В. Это позволяет выполнить повторную загрузку исходного числа в аккумулятор из регистра, а не из памяти. Программа имеет вид: Intel 8080 пример 3

Разделение слова на две части

```

LXI  H, 40H
MOV  A,M      ;Загрузить данные
MOV  B, A      ;Запомнить данные в PОН В
ANI  00001111B ;Замаскировать старшую цифру
INX  H
MOV  M, A      ;Запомнить младшую цифру
MOV  A, B
RRC          ;Четырехкратный
RRC          ;циклический
RRC          ;сдвиг
RRC          ;вправо
ANI  00001111B ;Замаскировать младшую цифру
INX  H
MOV  M, A      ;Запомнить старшую цифру
HLT
    
```


Единственной новой командой в этой программе является команда RRC, которая циклически сдвигает содержимое аккумулятора на 1 бит вправо. Она занимает одну ячейку памяти и выполняется за 2 мкс. Следует обратить внимание на то, что команда MOV A, B посылает содержимое регистра В в аккумулятор, а команда MOV B, A выполняет пересылку в обратном направлении. Команда межрегистровой пересылки занимает одну ячейку памяти и выполняется за 2,5 мкс.

Результат ассемблирования программ показан на рис. 5.11, а результат выполнения первой операции сдвига иллюстрируется на рис. 5.12. Начальные условия трассы несколько отличаются от конечной ситуации, показанной на рис. 5.8, из-за выполнения команды MOV B, A, которая запоминает копию данных. Можно отметить, что в программе имеется довольно большое число команд длиной в одно слово. Отметим также, что в Intel 8080 операция сдвига изменяет только содержимое признака ПЕРЕНОС.

Рассмотренные примеры простых программ иллюстрируют следующие особенности программирования на языке ассемблера.

1. Почти все операции обработки данных осуществляются над содержимым аккумуляторов. Программа начинается с отправки данных из памяти в аккумулятор и завершается запоминанием результата из аккумулятора в память.

Шестнадцатиричный адрес ячейки памяти	Мнемонический код операции	Шестнадцатиричный код содержимого памяти
00	LXI H, 40H	21
01		40
02		00
03	MOV A, M	7E
04	MOV B, A	47
05	ANI 00001111B	E6
06		0F
07	INX H	23
08	MOV M, A	77
09	MOV A, B	78
0A	RRC	0F
0B	RRC	0F
0C	RRC	0F
0D	RRC	0F
0E	ANI 00001111B	E6
0F		0F
10	INX H	23
11	MOV M, A	77
12	HLT	76

Рис. 5.11. Результат ассемблирования программы разделения слова для МП Intel 8080

После выполнения седьмой команды, MOV A,B:

Память данных		Регистры	
40	87	PC	000A
41	07	A	87
		B	87
		H	00
		L	41

После выполнения восьмой команды, RRC:

Память данных		Регистры	
40	87	PC	0008
41	07	A	C3
		B	87
		H	00
		L	41

Признак

Перенос



Рис. 5.12. Трасса выполнения программы разделения слова на МП Intel 8080

2. Программист старается предусмотреть как можно меньше обращений процессора к памяти. Число обращений к памяти можно уменьшить, используя нулевую страницу или косвенную адресацию с помощью регистров (что дает возможность работать с короткими командами), а также запоминая данные в РОН.

3. Следует внимательно анализировать влияние команд на состояние признаков. Характер изменения признаков существен, но меняется в зависимости от команд и типов процессоров.

4. С помощью команды ЛОГИЧЕСКОЕ И можно обнулить некоторые разряды ячейки, что дает возможность работать с данными, имеющими длину, меньшую длины ячейки.

5. С помощью команд сдвига можно осуществлять перемещение данных из одних разрядов в другие в целях как обработки, так и экономии памяти.

5.2. ОРГАНИЗАЦИЯ ЦИКЛОВ И ОБРАБОТКА МАССИВОВ

Разумеется, реальные задачи, решаемые на ЭВМ, не сводятся к обработке отдельного элемента данных с помощью одной операции. Напротив, они требуют обработки многих элементов данных (например, массива или блока данных) или отдельных элементов, которые занимают несколько ячеек памяти. Программа может выполнять одну и ту же операцию над содержимым нескольких ячеек или элементов данных. Многократное выполнение группы команд осуществляется с помощью программных циклов, числом повторений цикла управляют счетчики, а указатели или индексы показывают, какой именно элемент данных обрабатывается при данном проходе цикла

Пример 4. Сумма ряда чисел.

Задача состоит в том, чтобы осуществить суммирование нескольких чисел сразу. Эти числа могут представлять собой совокупности вход-

ных сигналов определенного типа, находящихся под управлением системы, число изделий, проданных в некоторой торговой операции, число очков, набранных в игре, или число сообщений, принятых за некоторый интервал времени. Предположим, что длина суммируемого ряда хранится в ячейке 41 и что сам ряд размещен, начиная с ячейки 42. Предположим также, что сумма не превышает 256 и для ее хранения достаточно одной 8-битной ячейки памяти с адресом 40.

Типичный пример

(41) = 03;
(42) = 35;
(43) = 72;
(44) = 1D

Ряд содержит три числа, так как в ячейке 41 находится число 3. Сумма равна

$(42) + (43) + (44) = 35 + 72 + 1D = C4$ (все числа являются шестнадцатиричными). Итак, результат равен (40) = C4.

Процесс сложения выполняется следующим образом:

Шаг 1.

COUNT = (41)
SUM = 0
POINTER = 42

Шаг 2.

SUM = SUM + (POINTER)

Шаг 3.

COUNT = COUNT - 1
POINTER = POINTER + 1

Шаг 4.

ЕСЛИ COUNT = 0 ИДТИ К ШАГУ 2

Шаг 5.

(41) = SUM

Из блок-схемы (рис. 5.13) видно, что программа состоит из четырех различных блоков:

1. Блок присвоения начальных значений переменным, счетчикам и указателям данных. Указатели представляют собой адреса данных.

2. Блок обработки, который фактически выполняет требуемые вычисления.

3. Блок управления циклом, в котором изменяются значения счетчиков и указателей перед выполнением следующей операции, а также осуществляется проверка числа выполнений цикла.

4. Заключительный блок выполняет запоминание результата.

Собственно суммирование выполняется в блоке 2. Однако наличие всех остальных блоков также существенно для обеспечения правильного выполнения блока 2. На рис. 5.14 приведена обобщенная блок-схема циклической программы. Первый и четвертый блоки выполняются только 1 раз, поэтому большая часть машинного времени затрачивается на выполнение второго и третьего блоков. Программа сможет

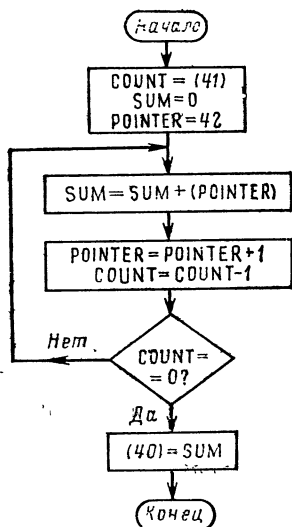


Рис. 5.13. Блок-схема программы суммирования массива данных

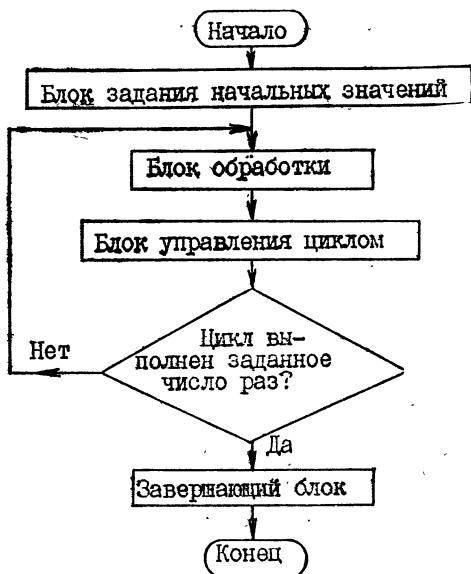


Рис. 5.14. Стандартная блок-схема циклической программы

выполняться существенно быстрее только в том случае, если программист сумеет уменьшить время работы второго и третьего блоков. Влияние первого и четвертого блоков на время выполнения программы незначительно.

Motorola 6800 (пример 4)

Программа сложения чисел для МП Motorola 6800 использует оба аккумулятора и индексный регистр. В аккумуляторе А содержится сумма, в аккумуляторе В—счетчик, а в индексном регистре — указатель данных, т. е. адрес того элемента данных, который сейчас прибавляется к сумме.

М6800 пример 4

Сумма данных

CLRA	
LDAB	\$41
LDX	\$42
SUMD	ADDA X
INX	
DECB	
BNE	SUMD
STAA	\$40
SWI	

Сумма=0

Счетчик=длина массива

Начальная точка массива

Сумма = сумма + элемент массива

Запоминание суммы

Новыми в данной программе являются команды: CLRA

Эта команда обнуляет аккумулятор А. Она занимает одну ячейку памяти и выполняется за 2 мкс.

LDX # 42

Эта команда посылает в 16 битный индексный регистр содержимое двух последовательных ячеек памяти. Старшие 8 бит находятся в первой ячейке, младшие — во второй (это стандартный способ, который, однако, отличается от способа, используемого МП Intel 8080). Команда занимает три ячейки памяти и выполняется за 3 мкс.

ADDA X (или 0, X)

Данная команда осуществляет сложение содержимого ячейки памяти, адрес которой находится в индексном регистре, и содержимого аккумулятора. В ассемблерной программе можно не указывать нулевое смещение; фактический адрес равен нулю плюс содержимое индексного регистра. Команда занимает две ячейки памяти и выполняется за 5 мкс; центральный процессор должен затратить дополнительное время на прибавление 16-битного значения смещения, даже если это значение равно нулю.

INX

Команда прибавляет 1 к содержимому 16-битного индексного (сокращенно X) регистра. Она занимает одну ячейку памяти и выполняется за 5 мкс. Команда INX инкрементирует указатель данных так, что в нем оказывается следующий, больший на 1 адрес.

DECB

Шестнадцатиричный адрес ячейки памяти	Мнемонический код операции	Шестнадцатиричный код содержимого памяти
00	CLRA	4F
01	LDAB \$ 41	D6
02		41
03	LDX # \$ 42	CE
04		00
05		42
06	SUMD ADDAX	AB
07		00
08	INX	08
09	DECB	5A
0A	BNE SUMD	26
0B		FA
0C	STAA \$ 40	97
0D		40
0E	SWI	3F

Рис. 5.15. Результат ассемблирования программы суммирования массива для МП Motorola 6800

После выполнения третьей команды, LDX # \$42:

Память данных		Регистры	
41	03	PC	0006
42	35	A	00
43	72	B	03
44	ID	X	0042

Признаки

Нуль	<input type="checkbox"/>	Отрицательный	<input type="checkbox"/>
------	--------------------------	---------------	--------------------------

После выполнения четвертой команды, ADRA X:
(исполнительный адрес равен (X) = 0042)

Память данных		Регистры	
41	03	PC	0008
42	35	A	35
43	72	B	03
44	ID	X	0042

Признаки

Нуль	<input type="checkbox"/>	Отрицательный	<input type="checkbox"/>	Перенос	<input type="checkbox"/>
------	--------------------------	---------------	--------------------------	---------	--------------------------

После выполнения седьмой команды, BNE SUMD:

Память данных		Регистры	
41	03	PC	0006
42	35	A	35
43	72	B	02
44	ID	X	0043

Так как содержимое признака НУЛЬ равно нулю,
программа передаст управление в ячейку SUMD (0006)

Рис. 5.16. Трасса выполнения программы суммирования массива на МП Motorola 6800

По этой команде из содержимого аккумулятора В вычитается 1. Команда занимает одну ячейку памяти и выполняется за 2 мкс. В аккумуляторе В содержится число повторений цикла, которое осталось выполнить.

BNE SUMD

Эта команда в случае равенства нулю признака НУЛЬ вызывает передачу управления ячейке, помеченной именем SUMD. В команде используется относительная адресация; значение смещения, записанное в дополнительном коде длиной 8 бит, представляет собой расстояние от ячейки памяти, находящейся непосредственно за командой перехода, до ячейки, куда передается управление. Команда BNE занимает две ячейки памяти и выполняется за 4 мкс. Как и при индексации, центральный процессор затрачивает дополнительное время на прибав-

ление 16-битного смещения к счетчику команд. Команда BNE приводит к выполнению следующих действий:

(PC) = SUMD, если признак НУЛЬ = 0

(PC) = (PC) — 2, если признак НУЛЬ = 1

Результат ассемблирования данной программы приведен на рис. 5.15. Новыми особенностями этой программы являются следующие:

1. Для загрузки индексного регистра зарезервированы две ячейки памяти (в четвертой и пятой ячейках размещены числа 00 и 42 соответственно).

2. В шестой и седьмой ячейках размещены код индексируемой команды ADDA X и нулевое смещение.

3. В ячейках памяти А и В размещен код команды BNE с относительным смещением. Значение смещения можно определить, если вычесть из адреса ячейки программной памяти, расположенной непосредственно за командой условного перехода, адрес ячейки, которая указана в качестве операнда в команде передачи управления, т. е.

06 — 0C = 06 + F4 = FA.

Шестнадцатичное вычитание выполняется путем сложения значения уменьшаемого с дополнительным кодом вычитаемого.

На рис. 5.16 приведена часть трассы программы при использовании данных из приведенного примера. Следует обратить внимание на индексирование и косвенную адресацию.

Intel 8080 (пример 4)

В программе суммирования ряда чисел для МП Intel 8080 в качестве счетчика использован PCH, а в качестве адресного указателя данных регистровая пара H и L.

Intel 8080 пример 4

Сумма данных

	SUB	A	;Сумма=0
	LXI	H, 41H	;Счетчик=длина массива
	MOV	B, M	
SUMD:	INX	H	
	ADD	M	;Сумма = сумма + данные
	DCR	B	
	INZ	SUMD	
	STA	40H	;Запомнить сумму
	HLT		

Новыми в данной программе являются команды:

SUBA

Данная команда обнуляет аккумулятор, вычитая его содержимое из него самого. Команда занимает одну ячейку памяти и выполняется за 2 мкс.

Шестнадцатиричный адрес ячейки памяти	Мнемонический код операции	Шестнадцатиричный код содержимого памяти
00	SUB A	97
01	LXI H, 41H	21
02		41
03		00
04	MOV B, M	46
05	SUMD: INX H	23
06	ADD M	86
07	DCR B	05
08	JNZ SUMD	C2
09		05
0A		00
0B	STA 40H	32
0C		40
0D		00
0E	HLT	76

Рис. 5.17. Результат ассемблирования программы суммирования массива для МП Intel 8080

ADD M

По этой команде содержимое ячейки памяти, адрес которой задан в регистровой паре H и L, прибавляется к содержимому аккумулятора. Команда занимает одну ячейку памяти и выполняется за 3,5 мкс.

DCR B

По этой команде из содержимого регистра B вычитается 1. Команда занимает одну ячейку памяти и выполняется за 2,5 мкс.

JNZ SUMD

По этой команде в случае равенства нулю признака НУЛЬ осуществляется переход к ячейке памяти, помеченной именем SUMD. Для указания адреса требуется 16 бит, из которых младшие 8 бит расположены в ячейке, идущей непосредственно за кодом команды, а старшие 8 бит — в следующей по порядку ячейке. Команда JNZ занимает три ячейки памяти и выполняется за 5 мкс.

На рис. 5.17 показан результат ассемблирования данной программы; частичная трасса одной из итераций приведена на рис. 5.18. Следует обратить особое внимание на использование регистра для косвенной адресации в командах MOV B, M и ADD M.

Программы для МП Intel 8080 и Motorola 6800 очень похожи. Они занимают одинаковое число ячеек памяти, так как косвенная адресация с помощью регистров, использованная при программировании данной задачи, оказывается более эффективной, чем в более простых задачах. На самом деле программа для МП Intel 8080 работает несколько быстрее (см. сравнительные оценки времени выполнения, приведенные на

рис. 5.19), поскольку в микропроцессоре этого типа не требуется выполнения операций сложения 16-битных адресов, которые необходимы в МП Motorola 6800 в связи с использованием косвенной адресации и индексации.

Пример 5. Пересылка данных.

Данная задача состоит в пересылке массива данных из одного места памяти в другое. Такая операция может понадобиться при инициализации элементов массива, при пересылке данных в (из) область ввода (вывода) или при формировании строки данных, выводимой на экран дисплея. Пусть длина массива задана в ячейке 40, данные располага-

После выполнения третьей команды, MOV B, M:

Память данных		Регистры	
41	03	PC	0005
42	35	A	00
43	72	B	03
44	1D	H	00

Исполнительный адрес = (H и L) = L
= 0041

После выполнения команды, ADD M:

Память данных		Регистры	
41	03	PC	0007
42	35	A	35
43	72	B	03
44	1D	H	00
		L	42

Исполнительный адрес = (H и L) = 0042

Признаки

Нуль ☐ Знак ☐ Перенос ☐

После выполнения седьмой команды, JNZ SUMD:
Память данных

Память данных		Регистры	
41	03	PC	0005
42	35	A	35
43	72	B	02
44	1D	H	00
		L	42

Программа возвращается в ячейку памяти SUMD (0005), так как признак NZ равен нулю

Рис. 5.18. Трасса выполнения программы суммирования массива на МП Intel 8080

Команды Intel 8080	Время выполне- ния, мкс	Команды Motorola 6800	Время выполне- ния, мкс
Часть 1. Задание начальных значений			
SUB A	2	CLRA	2
LXI H, 41H	5	LDAB \$41	3
MOV B, M	2,5	LDX # \$ 42	3
Всего	9,5		8
Часть 2. Программный цикл			
INX H	2,5	ADDA X	5
ADD M	3,5	INX	4
DCR B	2,5	DECB	2
JNZ SUMD	5	BNE SUMB	4
Всего	13,5		15
Часть 3. Завершающий блок			
STA 40H	6,5	STAA \$ 40	3
Всего	6,5		3

Рис. 5.19. Сравнение временных затрат

ются начиная с ячейки 41 и область, в которую они пересылаются, начинается с ячейки 51.

Типичный пример.

(40) = 0,2;

(41) = 7E;

(42) = 55.

Массив состоит из двух элементов, расположенных в ячейках 41 и 42. В результате пересылки

(51) = 7E;

(52) = 55.

Содержимое ячейки 41 пересылается в ячейку 51, а содержимое ячейки 42 — в ячейку 52.

На рис. 5.20 приведена блок-схема соответствующей программы. У этой программы отсутствует заключительный блок, но имеются блоки инициализации, обработки и управления циклом. Для работы требуются два указателя данных: в одном указателе содержится адрес исходной области, а во втором — адрес той области, куда данные пересылаются. Заметим, что разность соответствующих адресов остается всегда постоянной. Это позволяет использовать для работы с обеими областями единственный индексный регистр.

Motorola 6800 (пример 5)

В программе для МП Motorola 6800 фиксированное расстояние между исходной и результирующей областями используется в качестве значения смещения в индексируемой команде. Это дает возможность получить доступ к элементам обоих массивов с помощью одного индексного регистра. Однако при этом массивы могут располагаться на расстоянии не более 256 ячеек, так как значение смещения задается полем длиной 8 бит. Программа имеет следующий вид:

Motorola 6000 пример 5

Пересылка данных

	LDX	# \$41	Начальный адрес массива
	LDAB	\$ 40	Счетчик = длина массива
TRANS	LDAA	X	Взять данные из исходной области
	STAA	\$ 10, X	Послать данные в область результата
	INX		
	DECB		
	BNE	TRANS	
	SWI		

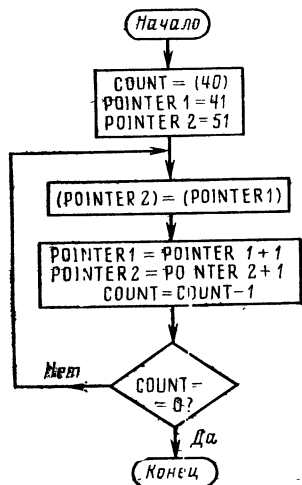


Рис. 5.20. Блок-схема программы пересылки данных

Шестнадцатиричный адрес ячейки памяти	Мнемонический код операции	Шестнадцатиричный код содержимого памяти
00	LDX # \$ 41	CE
01		00
02		41
03	LDAB \$ 40	D6
04		40
05	TRANS: LDAA X	A6
06		00
07	STAA \$10, X	A7
08		10
09	INX	03
0A	DECB	5A
0B	BNE TRANS	26
0C		F8
0D	SWI	3F

Фактическая величина смещения в команде BNE TRANS равна $05 - 0D = 05 + F3 = F8$.

Рис. 5.21. Результат ассемблирования программы пересылки данных для МП Motorola 6800

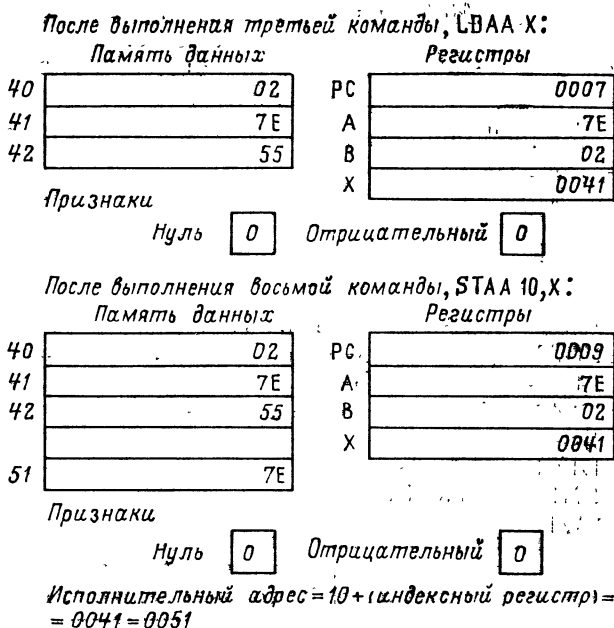


Рис. 5.22. Часть трассы выполнения программы пересылки данных на МП Motorola 6800

Единственной новой особенностью данной программы является совместное использование для адресации смещения и индексного регистра. При выполнении команды STAA \$ 10, X исполнительный адрес будет равен 10_{16} плюс содержимое индексного регистра. Следует обратить внимание на то, что в ассемблированной программе (рис. 5.21) значение смещения, равное 10_{16} , располагается непосредственно за кодом команды STAA. Трасса программы для одного прохода приведена на рис. 5.22; исходные данные взяты из примера.

Intel 8080 (пример 5)

В программе для МП Intel 8080 в качестве адресного регистра используется регистровая пара H и L. В этом микропроцессоре в качестве адресных регистров можно также использовать регистры B и C (регистровая пара B) или регистры D и E (регистровая пара D), но только для выполнения операций пересылки данных из памяти в аккумулятор или обратно (с помощью команд LDAX и STAX). Эти адресные регистры нельзя использовать в командах, выполняющих арифметические или логические операции, а также операции пересылки данных в PОН или из PОН. Упомянутое ограничение редко оказывается существенным, так как обычно процессор перед выполнением операций обработки должен пересылать данные в аккумулятор. Вместе с тем использование различных кодов команд для различных адресных регистров может

привести к ошибкам. Заметим, что при каждом проходе программа должна изменять содержимое обоих адресных регистров и что один и тот же набор регистров должен совместно использоваться в качестве счетчиков, указателей адреса и рабочих ячеек.

Intel 8080 пример 5

Пересылка данных

LXI H, 40H ;Счетчик = длина массива

MOV B, M

LXI D, 50H ;Начальный адрес результирующего массива

TRANS: INX H
 MOV A, M ;Загрузить данные из исходного массива
 STAX D ;Послать данные в результирующий массив
 DCR B
 JNZ TRANS
 HLT

Новой в этой программе является команда STAX D, по которой содержимое аккумулятора посылается в ячейку памяти, адрес которой содержится в регистровой паре D. Команда занимает одну ячейку па-

Шестнадцатиричный адрес ячейки памяти	Мнемонический код операции	Шестнадцатиричный код содержимого памяти
00	LXI H, 40H	21
01		40
02		00
03	MOV B, M	46
04	LXI D, 50H	11
05		50
06		00
07	TRANS: INX D	13
08	INX H	23
09	MOV A, M	7E
0A	STAX D	12
0B	DCR B	05
0C	JNZ TRANS	C2
0D		
0E		07
0F	HLT	00
		76

Рис. 5.23. Результат ассемблирования программы пересылки данных для МП Intel 8080

После выполнения шестой команды, MOV A, M:

Память данных

40	02
41	7E
42	55

Регистры

PC	000A
A	7E
B	02
D	00
E	51
H	00
L	41

Исполнительный адрес = (H и L) = 0041

После выполнения седьмой команды, STAX D

Память данных

40	02
41	7E
42	55
51	7E

Регистры

PC	000B
A	7E
B	02
D	00
E	51
H	00
L	41

Исполнительный адрес = (H и L) = 0051

Рис. 5.24. Часть трассы выполнения программы пересылки данных (первый проход) на МП Intel 8080

мента и выполняется за 3,5 мкс. Команда STAX D аналогична команде MOV M, A и отличается от нее только тем, что использует иной адресный регистр; подобная аналогия существует между командами LDAX D и MOV A, M.

На рис. 5.23 приведен результат ассемблирования данной программы для МП Intel 8080. На рис. 5.24 показана часть трассы для первого прохода. Центральный процессор должен изменять содержимое обоих адресных регистров, но при этом для вычисления исполнительного адреса не требуется суммировать 16-битные числа. В МП Intel 8080 массивы могут располагаться в произвольных местах памяти; не обязательно располагать их на расстоянии не более 256 ячеек, как это имеет место в МП Motorola 6800.

Пример 6. Нахождение максимума.

Разумеется, блоки обработки в циклических программах редко бывают столь простыми, как в примерах 4 и 5. Обычно в них выполняется много операций, предусмотрены разветвления и иногда даже встроены другие программные циклы. При этом остальные блоки циклической программы могут остаться неизменными, какова бы ни была сложность самого блока обработки.

Для отыскания максимального элемента массива требуется разработать несколько более сложный, чем в предыдущих примерах, блок обработки. Подобная операция может понадобиться при сортировке данных, при анализе тестовых результатов, при выборе следующей записи, если обработка ведется на приоритетной основе, или при мас-

штабирований входных или выходных данных для выполнения их анализа и выдачи на устройство отображения. Пусть длина массива данных задана в ячейке 41, а сам массив располагается с ячейки 42. Программа помещает максимальное значение в ячейку 40. Элементы массива представляют собой 8-битные числа без знака.

Типичный пример.

(41) = 03;
(12) = 37;
(43) = F2;
(44) = C6.

Массив состоит из трех элементов, расположенных в ячейках памяти 42—44. В результате работы алгоритма (40) = F2, так как максималь-

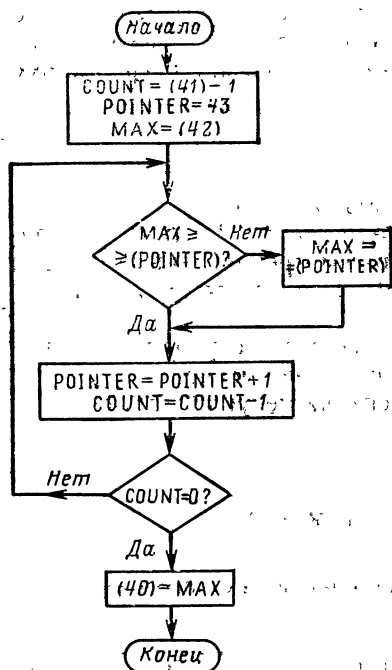


Рис. 5.25. Блок-схема программы поиска максимума

Шестнадцатиричный адрес ячейки памяти	Мнемонический код операции	Шестнадцатиричный код содержимого памяти
00	LDAB \$41	D6
01		41
02	DECB	5A
03	LDAA \$42	96
04		42
05	LDX #43	CE
06		00
07		43
08	MAXM CMPSX	A1
09		00
0A	BCC NOCHG	24
0B		02
0C	LDAA X	A6
0D		00
0E	NOCHG INX	08
0F	DECB	5A
10	BNE MAXM	26
11		F6
12	STAA \$40	97
13		40
14	SWI	3F

Фактическое значение смещения в команде BCC NOCHG равно $0E - 0C = 0E + F4 = 02$.

Фактическое значение смещения в команде BNE MAXM равно $08 - 12 = 08 + EE = F6$.

Рис. 5.26. Результат ассемблирования программы поиска максимума для МП Motorola 6800

ным из трех заданных чисел без знака длиной 8 бит является число F 2 (содержимое ячейки 43).

На рис. 5.25 изображена блок-схема программы. Блок обработки содержит операцию условного перехода, с помощью которой осуществляется проверка того, является ли новая запись большей, чем та, которая до этого считалась максимальной. Если это так, то прежнее максимальное значение заменяется новым. Вначале предполагается, что максимальным является первый элемент массива. Программа уменьшает содержимое счетчика на единицу, поскольку в дальнейшем анализировать первый элемент массива не нужно. Структура программы не отличается от структуры программ, рассмотренных ранее.

Motorola 6800 (пример 6)

В программе для МП Motorola 6800 для определения того, является ли запись, которая ранее считалась максимальной, большей, чем вновь выбранная, используется команда сравнения (CMPA X). Если в результате сравнения признак ПЕРЕНОС не устанавливается в единицу (что свидетельствует о том, что уменьшаемое больше вычитаемого), то программа оставляет в аккумуляторе A прежнее максимальное значение. В противном случае максимальное значение заменяется значением текущего элемента. Следует обратить внимание на то, что достоинством команды СРАВНИТЬ является то, что прежнее максимальное значение в аккумуляторе A сохраняется и может вновь участвовать в сравнениях, если очередной элемент не оказался большим. Программа имеет следующий вид:

Motorola 6800 пример 6

Поиск максимума

	LDAB	\$ 41	Счетчик=длина массива
*	DECB		Счетчик = счетчик -1, чтобы исключить из рассмотрения первый элемент
	LDAA	\$ 42	Максимум равен первому элементу
	LDX	# \$ 43	
MAXM	CMPA	X	Максимум больше текущего элемента?
	BCC	NOCNG	Да, максимум не меняется
	LDAA	X	Нет, замена на текущий элемент
NOCNG	INX		
	DECB		
	BNE	MAXM	
	STAA	\$ 40	Запомнить максимум
	SWI		

Новыми в данной программе являются следующие команды:

CMPA X

По этой команде значения признаков формируются точно так же, как если бы содержимое указанной в команде ячейки памяти вычиталось из содержимого аккумулятора A. В индексированном варианте данная команда занимает две ячейки памяти и выполняется за 5 мкс.

Если М обозначает адресуемую ячейку памяти, то признаки НУЛЬ и ПЕРЕНОС устанавливаются следующим образом:

НУЛЬ = 1, если (А) = (М);

ПЕРЕНОС = 1, если (М) > (А) при условии, что оба числа не имеют знака

BCC NOCHG

По этой команде, если признак ПЕРЕНОС равен нулю, происходит передача управления в ячейку NOCHG. Как и в команде BNE, в команде BCC используется относительная адресация; команда занимает две ячейки памяти и выполняется за 4 мкс.

На рис. 5.26 показан результат ассемблирования данной программы. В ней имеются две команды условного перехода с относительными адресами. На рис. 5.27 показаны трассы двух проходов блока обработки данной программы. Следует обратить внимание на выполнение команды СРАВНИТЬ и на влияние, которое оказывают в приведенных двух проходах значения признаков на процесс выполнения программы.

Intel 8080 (пример 6)

Программа нахождения максимального элемента для МП Intel 8080 аналогична соответствующей программе для МП Motorola 6800. Для реализации соответствующих проверок в ней используются команды СРАВНИТЬ и ПЕРЕЙТИ ПРИ ОТСУТСТВИИ ПЕРЕНОСА. Программа имеет следующий вид:

Intel 8080 пример 6

Поиск максимума

	LXI	H, 41H	
	MOV	B, M	; Счетчик = длина массива
	DCR	B	; Уменьшить содержимое счетчика на 1,
			; чтобы исключить из рассмотрения
			; первый элемент
	INX	H	
	MOV	A, M	; Максимум равен первому элементу
MAXM:	INX	H	
	CMP	M	; Максимум больше текущего элемента?
	JNG	NOCHG	; Да, сохранить прежнее значение
			; максимума
	MOV	A, M	; Нет, заменить максимум на текущий
			; элемент
NOCHG:	DCR	B	
	JNZ	MAXM	
	STA	40H	; Сохранить максимум
	HLT		

Новыми в данной программе являются следующие команды:

Первый проход:

После выполнения команды, С МРАХ:

Память данных

Регистры

41	03
42	37
43	F2
44	C6

PC	000A
A	37
B	02
X	0043

Признаки

Нуль ☐ Отрицательный ☐ Перенос ☐

Значения признаков формируются так, как будто ЦП выполняет операцию $(A)-(X)=(A)-(0043)=37-F2=37-0E=45$. Обратите внимание на то, что ЦП инвертирует значение признака ПЕРЕНОС, возникающее при сложении, поэтому значение признака ПЕРЕНОС фактически представляет собой заем.

После выполнения шестой команды, ВСС НОСНГ:

Память данных

Регистры

41	03
42	37
43	F2
44	C6

PC	000C
A	F2
B	02
X	0034

Так как ПЕРЕНОС=1, передача управления не реализуется второй проход:

После выполнения команды, С МРАХ:

Память данных

Регистры

41	03
42	37
43	F2
44	C6

PC	000A
A	F2
B	01
X	0044

Признаки

Нуль ☐ Отрицательный ☐ Перенос ☐

Значения признаков формируются так, как будто ЦП выполняет операцию

$$(A)-(X)=(A)-(0044)=F2-C6=2C$$

После выполнения шестой команды, ВСС НОСНГ:

Память данных

Регистры

41	03
42	37
43	F2
44	C6

PC	000E
A	F2
B	01
X	0044

Происходит передача управления, так как ПЕРЕНОС=0

Рис. 5.27. Часть трассы выполнения программы поиска максимума на МП Motorola 6800

CMP M

При выполнении этой команды признаки устанавливаются так, как если бы содержимое ячейки памяти, адрес которой содержится в регистрах H и L, вычиталось из содержимого сумматора. Эта команда занимает одну ячейку памяти и выполняется за 3,5 мкс.

JNG NOCHG

По этой команде управление передается в ячейку с символическим адресом NOCHG в том случае, если признак ПЕРЕНОС равен нулю. Команда занимает три ячейки памяти и выполняется за 5 мкс.

На рис. 5.28 приведен результат ассемблирования рассматриваемой программы для МП Intel 8080. На рис. 5.29 приведена трасса двух проходов блока обработки данной программы.

В трех последних примерах можно выявить некоторые общие закономерности ассемблерных программ.

Шестнадцатиричный адрес ячейки памяти	Мнемонический код операции	Шестнадцатиричный код содержимого памяти
00	LXI H, 41H	21
01		41
02		00
03	MOV B, M	46
04	DCR B	05
05	INX H	23
06	MOV A, M	7E
07	MAXM:	23
08	INX H	
09	CMP M	BE
	JNC	D2
	NOCHG	
0A		0D
0B		00
0C	MOV A, M	7F
0D	NOCHG:	05
	DCR B	
0E	JNZ	C2
	MAXM	
0F		07
10		00
11	STA 40H	32
12		40
13		00
14	HLT	76

Рис. 5.28. Результат ассемблирования программы для МП Intel 8080

Первый проход:

После выполнения седьмой команды, CMP M:

Память данных

Регистры

41	03
42	37
43	F2
44	C6

PC	0009
A	37
B	02
H	00
L	43

Признаки

Ноль ☐

Знак ☐

Перенос ☐

Значения признаков формируются так, как будто ЦП выполняет операцию $(A) - ((H \text{ и } L)) = (A) - (0043) = 37 - F2$

ЦП инвертирует значения признака ПЕРЕНОС, возникающее при вычитании, и формирует в признаке ПЕРЕНОС заем.

После выполнения восьмой команды, JNC NOCHG:

Память данных

Регистры

41	03
42	37
43	F2
44	C6

PC	000C
A	37
B	02
H	00
L	43

Передача управления не происходит, так как ПЕРЕНОС=1.

Второй проход:

После выполнения седьмой команды, CMP M:

Память данных

Регистры

41	03
42	37
43	F2
44	C6

PC	0009
A	F2
B	01
H	00
L	44

Признаки

Ноль ☐

Знак ☐

Перенос ☐

Значения признаков формируются так, как будто ЦП выполняет операцию $(A) - ((H \text{ и } L)) = (A) - (0044) = F2 - C6$

После выполнения восьмой команды, JNC NOCHG:

Память данных

Регистры

41	03
42	37
43	F2
44	C6

PC	000D
A	F2
B	01
H	00
L	44

Происходит передача управления, так как ПЕРЕНОС=0

Рис. 5.29. Часть трассы выполнения программы поиска максимума на МП Intel 8080

1. *Структура простого цикла.* Каждая программа состоит из блоков установки начальных значений (инициализации), обработки и управления циклом. Электронно-вычислительная машина выполняет блок инициализации всего 1 раз перед входом в цикл. Блок управления циклом определяет число повторений цикла; этот блок может быть одинаковым для многих задач обработки данных. Блок обработки может включать в себя всего один операнд или быть сложной программой.

2. *Использование указателей для доступа к элементам массива.* Использование адресных указателей уменьшает число адресов памяти, выбираемых центральным процессором. Такое сокращение выборки адресов особенно важно для микропроцессоров, в которых адреса занимают обычно по две ячейки памяти. Наличие многих адресных регистров и индексирование позволяют повысить гибкость применения указателей. В блоке инициализации следует задать начальные значения всех адресных указателей, а блок управления должен осуществлять их модификацию после каждого повторения цикла.

3. *Использование команд условного перехода для организации циклов и разветвления вычислительного процесса.* В типовой структуре цикла предполагается установка начального значения счетчика и его уменьшение после каждого прохода. Команда ПЕРЕЙТИ ПО НЕ НУЛЮ обеспечивает выполнение цикла заданное число раз. Команда СРАВНИТЬ выполняет вычитание и устанавливает значение признаков, не меняя содержимого регистров. В зависимости от результата сравнения с помощью команд ПЕРЕХОД ПО НУЛЮ; ПЕРЕХОД ПО НЕ НУЛЮ, ПЕРЕХОД ПО ПЕРЕНОСУ и ПЕРЕХОД ПО ОТСУТСТВИЮ ПЕРЕНОСА можно выбрать соответствующую ветвь программы.

Работа с массивами и организация циклов требуют большого внимания ко всем деталям. Программист должен проверить:

- а) правильно ли заданы начальные значения всех переменных перед входом в цикл;
- б) верно ли заданы адреса в командах условного перехода;
- в) верно ли выбраны проверяемые условия в точках разветвления;
- г) верно ли выполняется первый и последний проходы цикла.

К этим вопросам вернемся при рассмотрении методов отладки программ в гл. 6.

5.3. ВЫЧИСЛИТЕЛЬНЫЕ ПРОГРАММЫ

Нахождение суммы ряда чисел в примере 4 представляет собой простую вычислительную задачу. Для большинства вычислений точность, обеспечиваемая одной ячейкой памяти, недостаточна; кроме того, необходима работа с десятичными числами и требуются более сложные операции, чем сложение и вычитание. В данном параграфе рассмотрим арифметику для чисел, занимающих несколько ячеек, десятичную арифметику и табличное задание функций. Программы имеют циклическую структуру, рассмотренную в предыдущем параграфе.

Пример 7. Арифметика для чисел, занимающих несколько ячеек памяти.

Задача состоит в том, чтобы сложить два числа длиной более 8 бит каждое. Подобными числами могут быть результаты работы высокоточных АЦП, значения тригонометрических или показательных функций, полюсы или корни уравнений, значения токов, результаты обработки или другие данные, для представления которых недостаточно слова длиной 8 бит. Пусть длина чисел (в ячейках) задана в ячейке 40, а сами числа располагаются, начиная с ячеек 41 и 51 соответственно, таким образом, что сначала идут младшие разряды. Программа помещает результат в ячейки, где хранилось первое исходное число

Типичный пример.

(40) = 03;
 (41) = 29;
 (42) = A4;
 (43) = 50;
 (51) = FB;
 (52) = 37;
 (53) = 28.

Задача состоит в том, чтобы сложить 24-битные числа 50A429 и 2837FB. Результат будет следующим:

(41) = 24 = 29 + FB
 (42) = DC = A4 + 37 + ПЕРЕНОС (1)
 (43) = 78 = 50 + 29 + ПЕРЕНОС (0)

Возникающие разряды переноса должны участвовать в сложении.

Блок-схема программы приведена на рис. 5.30. Как и ранее, в программе имеются блоки инициализации, обработки и управления циклом. Единственным новым элементом является использование признака ПЕРЕНОС для учета возникающего переноса в следующем проходе цикла.

Motorola 6800 (пример 7)

В программе для МП Motorola 6800 сначала команда ОБНУЛИТЬ ПЕРЕНОС (CLC) обнуляет признак, а команда СЛОЖИТЬ С ПЕРЕНОСОМ (ADC) выполняет сложение. Программа имеет вид:

Motorola 6800 пример 7

Сложение длинных чисел

	LDAB	\$ 40	Счетчик = длина чисел
	CLC		Перенос=0
	LDX	# \$ 41	Установить указатель на первое число
MPADD	LDAA	X	Взять 8 бит первого числа
	ADCA	\$ 10, X	Прибавить 8 бит второго числа
	STAA	X	Поместить результат в ячейку памяти, занимаемую первым числом
	INX		
	DECB		
	BNE	MPADD	
	SWI		

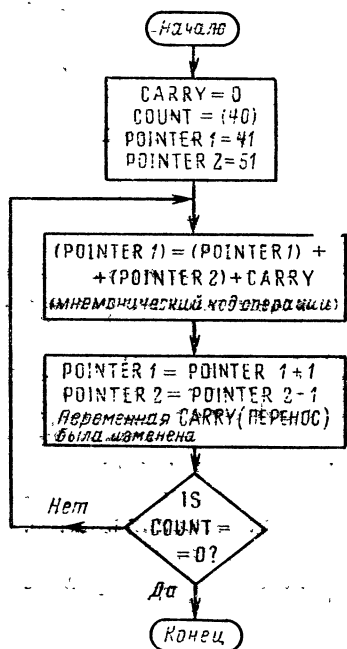


Рис. 5.30. Блок-схема программы сложения двоичных чисел

Шестнадцатиричный адрес ячейки памяти	Мнемонический код операции	Шестнадцатиричный код содержимого памяти
00	LDAB	6
01	\$40	40
02	CLC	0C
03	LDX	CE
04	# \$ 41	00
05		41
06	MPADD	A6
	LDAA	
	X	
07		00
08	ADCA	A9
	\$10, X	
09		10
0A	STAA X	A7
0B		00
0C	INX	08
0D	DECB	5A
0E	BNE	26
	MPADD	
0F		F6
10	SWI	3F

Значение относительного адреса в команде BNE MPADD равно $06 - 10 = 06 + F0 = F6$.

Рис. 5.31. Результат ассемблирования программы сложения длинных чисел для МП Motorola 6800

По команде ADCA X к содержимому аккумулятора A прибавляется содержимое адресуемой ячейки памяти и содержимое признака ПЕРЕНОС.

На рис. 5.31 приведен результат ассемблирования данной программы. На рис. 5.32 показано, к каким результатам приводит выполнение команды СЛОЖИТЬ С ПЕРЕНОСОМ (ADC). Команды INX и DECB, обеспечивающие управление циклом, не влияют на значение признака ПЕРЕНОС. В результате значение этого признака может использоваться в следующем проходе цикла.

Проход I:

После выполнения четвертой команды, LDAAX:

Память данных		Регистры	
40	03	PC	0008
41	29	A	29
42	A4	B	03
43	50	X	0041
51	FB		
52	37		
53	28		

Признаки

Ноль ☐ Отрицательный ☐ Перенос ☐

После выполнения пятой команды, ADCA \$40, X:

Память данных		Регистры	
40	03	PC	000A
41	29	A	24
42	A4	B	03
43	50	X	0041
51	FB		
52	37		
53	28		

Признаки

Ноль ☐ Отрицательный ☐ Перенос ☐

Прекнее значение признака ПЕРЕНОС было равно нулю и он не влиял на результат сложения; новое же значение признака ПЕРЕНОС равно единице

Проход 2:

После выполнения четвертой команды, LDAAX:

Память данных		Регистры	
40	03	PC	0008
41	24	A	A4
42	A4	B	02
43	50	X	0042
51	FB		
52	37		
53	28		

Признаки

Ноль ☐ Отрицательный ☐ Перенос ☐

После выполнения пятой команды, ADCA \$10, X:

Память данных		Регистры	
40	03	PC	000A
41	24	A	DC
42	A4	B	02
43	50	X	0042
51	FB		
52	37		
53	28		

Признаки

Ноль ☐ Отрицательный ☐ Перенос ☐

Рис. 532. Трасса выполнения операции СЛОЖИТЬ С ПЕРЕНОСОМ в программе сложения длинных чисел на МП Motorola 6800

Intel 8080 (пример 7)

В программе для МП Intel 8080 для обнуления признака ПЕРЕНОС используется команда SUB A (специальная команда ОБНУЛИТЬ ПРИЗНАК ПЕРЕНОС отсутствует), а для выполнения сложения — команда СЛОЖИТЬ С ПЕРЕНОСОМ (ADC). Программа имеет вид: Intel 8080 пример 7

Сложение длинных чисел

	SUB	A	; Перенос = 0
	LXI	H, 40H	; Счетчик = длина чисел
	MOV	B, H	
	LXI	D, 60H	; Установить указатель на второе ; число
MPADD.	INX	D	
	INX	H	
	LDAX	D	; Взять 8 бит второго числа
	ADC	M	; Прибавить 8 бит первого числа
	MOV	M, A	; Поместить сумму в ячейку памяти; ; ти, занимаемую первым числом
	DCR	B	
	JNZ	MPADD	
	HLT		

Порядок слагаемых при выполнении операций сложения МП в Intel 8080 обратный, поскольку процессор может выполнять сложение, только используя косвенную адресацию через регистры H и L. В данном случае, как и ранее, выполнение команд INX D, INX H и DCR B не влияет на состояние признака ПЕРЕНОС и информация о переносе может использоваться в следующем проходе цикла. По команде ADC M содержимое ячейки памяти, адрес которой содержится в регистрах H и L, и содержимое признака ПЕРЕНОС прибавляются к содержимому аккумулятора.

На рис. 5.33 показана программа для МП Intel 8080 после ассемблирования. Команда СЛОЖИТЬ С ПЕРЕНОСОМ работает одинаково на МП Intel 8080 и Motorola 6800.

Пример 8. Десятичная арифметика.

У большинства 8-битных микропроцессоров имеется специальная команда для реализации сложения десятичных чисел. Эта команда (которая обычно называется ДЕСЯТИЧНАЯ КОРРЕКТИРОВКА) использует биты ПЕРЕНОС и ПОЛУПЕРЕНОС (или ВСПОМОГАТЕЛЬНЫЙ ПЕРЕНОС).

Десятичная арифметика используется в таких распространенных приложениях микропроцессоров, как терминалы в торговых точках, банковские терминалы, калькуляторы, игры и навигационные системы. Рассмотрим, например, ту же задачу, что и в примере 7, с той только разницей, что слагаемые являются двоично-кодированными десятичными числами. Длина чисел хранится в ячейке 40, а сами числа в ячейках 41 и 51 соответственно, результат помещается на место первого числа.

Шестнадцатиричный адрес ячейки памяти	Мнемонический код операции	Шестнадцатиричный код содержимого памяти
00	SUB A	97
01	LXI H, 40H	21
02		40
03		00
04	MOV B, M	46
05	LXI D, 60H	11
06		60
07		00
08	MPADD:	13
09	INX D	23
0A	INX H	1A
0B	LDAX D	8E
0C	ADC M, A	77
0D	DCR B	05
0E	JNZ MPADD	C2
0F		08
10		00
11	HLT	76

Рис. 5.33. Результат ассемблирования программы сложения длинных чисел для МП Intel 8080

Типичный пример.

(40) = 03;
(41) = 29;
(42) = 65;
(43) = 37;
(51) = 88;
(52) = 43;
(53) = 22.

Задача состоит в том, чтобы сложить две строки, каждая из которых состоит из шести десятичных цифр:

376529 + 224388.

Результатом сложения является число 600917:

(41) = 17;
(42) = 09;
(43) = 60.

Программа сложения десятичных чисел - аналогична программе, блок-схема которой приведена на рис. 5.30. Отличие состоит в том, что операция сложения является десятичной, а не двоичной.

Motorola 6800 и Intel 8080 (пример 8)

Единственное изменение, которое нужно внести в рассмотренные в примере 7 программы для МП Motorola 6800 и Intel 8080, состоит в том, чтобы после команды СЛОЖИТЬ С ПЕРЕНОСОМ вставить команду DAA (ДЕСЯТИЧНАЯ КОРРЕКТИРОВКА АККУМУЛЯТОРА). На рис. 5.34 показано, как работает команда DAA в первых двух проходах цикла. Каждая операция десятичного сложения выполняется с по-

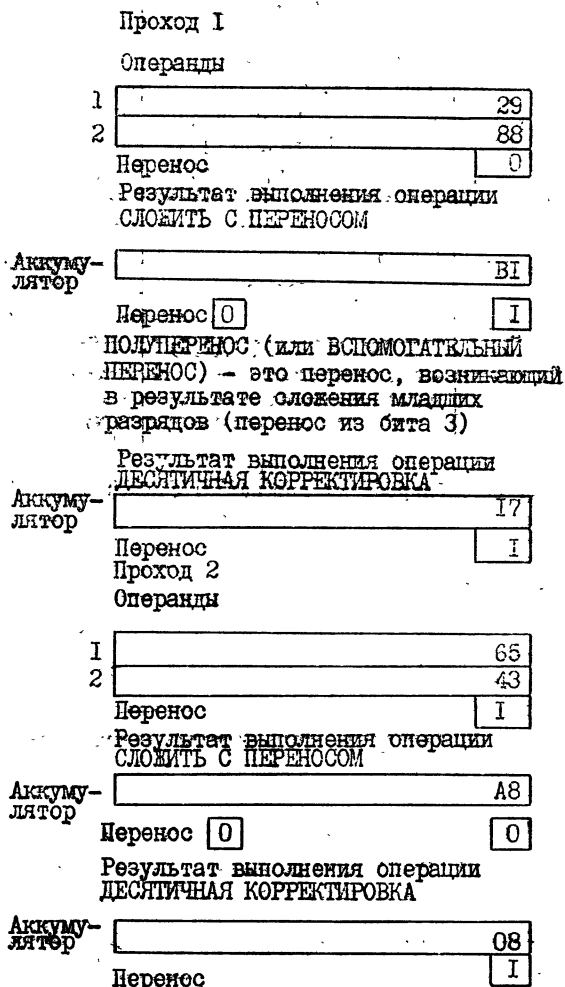


Рис. 5.34. Трасса выполнения команды ДЕСЯТИЧНАЯ КОРРЕКТИРОВКА

мощью двух команд: команды сложения и идущей непосредственно за ней команды ДЕСЯТИЧНАЯ КОРРЕКТИРОВКА. В МП Motorola 6800 команда ДЕСЯТИЧНАЯ КОРРЕКТИРОВКА используется только для корректировки содержимого аккумулятора А. В других микропроцессорах эта задача решается несколько иначе. В микропроцессоре PACE фирмы National имеется команда ДЕСЯТИЧНОЕ СЛОЖЕНИЕ, а в MOS Technology 6502 существует десятичный режим (включаемый командой УСТАНОВИТЬ ДЕСЯТИЧНЫЙ РЕЖИМ), в котором все арифметические операции выполняются в десятичной системе счисления: Motorola 6800 — пример 8

Десятичное сложение

	LDAB	\$ 40	Счетчик = длина чисел
	CLC		Перенос = 0
	LDX	# \$ 41	Установить указатель на первое число
DCADD	LDA	X	Взять две цифры первого числа
	ADCA	\$ 10, X	Прибавить две цифры второго числа
	DAA		Десятичная корректировка
	STA	X	Послать сумму на место первого числа
	INX		
	DECB		
	BNE	DCADD	
	SWI		

Intel 8080 пример 8

Десятичное сложение

	SUB	A	; Перенос = 0
	LXI	H, 40H	
	MOV	B, M	; Счетчик = длина чисел
	LXI	D, 50H	; Установить указатель на второе число
DCADD:	INX	D	
	INX	H	
	LDAX	D	; Взять две цифры второго числа
	ADC	M	; Прибавить две цифры первого числа
	DAA		; Десятичная корректировка
	MOV	M, A	; Послать сумму на место первого числа
	DCR	B	
	JNZ	DCADD	
	HLT		

Пример 9. Таблица квадратов.

Решение более сложных вычислительных задач часто могут упростить таблицы с готовыми результатами. В таких таблицах в определенном порядке хранятся все возможные результаты вычислений. В каждом конкретном случае программа должна уметь выбирать верный результат. При таком способе вычислений достигается высокое быстродействие за счет расхода дополнительной памяти: использование таблицы уменьшает время расчета, так как результат не нужно вычис-

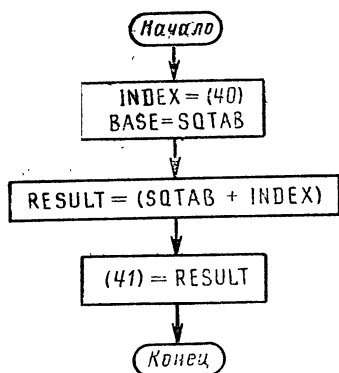


Рис. 5.35. Блок-схема программы поиска значения в таблице квадратов

лять, но требует больших затрат памяти, поскольку в памяти должны быть заранее зафиксированы все возможные результаты. По мере того как стоимость памяти уменьшается, табличное задание функций становится все более целесообразным. Таблицы функций и таблицы перекодировки доступны даже в стандартных ПЗУ.

Для примера рассмотрим программу вычисления квадрата 3-битного числа. Подобная задача возникает при обработке сигналов, в системах связи и при управлении производственными процессами. Таблица состоит из всех квадратов 3-битных чисел, расположенных в порядке возрастания. Такую таблицу можно разме-

стить в памяти с помощью ассемблерной псевдооперации DATA следующим образом:

SQTAB DATA 0, 1, 4, 9, 16, 25, 36, 49

Индексом при обращении к такой таблице является число, квадрат которого отыскивается: нулевой элемент содержит квадрат нуля, первый — квадрат единицы и т. д. Чтобы найти адрес нужного элемента таблицы, программа должна прибавить 3-битное число к базовому адресу SQTAB. Пусть исходное число, заключенное между 0 и 7, находится в ячейке 40, а результат следует поместить в ячейку 41.

Типичный пример.

(40) = 04.

Нужно возвести в квадрат число 4. Результат получается равным

(41) = $10_{16} = 16_{10}$.

Интересующий нас элемент таблицы расположен в ячейке с адресом SQTAB + 4.

Блок-схема программы представлена на рис. 5.35. Чтобы вычислить адрес нужного элемента таблицы, по которому затем можно выбрать элемент, программа должна выполнить операцию сложения.

Motorola 6800 (пример 9)

Поскольку в МП Motorola 6800 имеются возможности индексации, получение нужного элемента таблицы может показаться простым. Очевидное решение состоит в том, чтобы поместить базовый адрес таблицы в индексный регистр, а индекс использовать в качестве смещения. Тогда с помощью индексной адресации можно выбрать нужный элемент. Однако этот способ не годится, так как смещение задается как постоянная величина в памяти программ (в ПЗУ) и не может изменяться в за-

висимости от значения возводимой в квадрат величины. Базовый адрес и индекс также нельзя поменять местами, так как обычно базовый адрес имеет длину 16 бит, а для представления смещения отводится 8 бит. Решение является не очень изящным. При достаточно короткой таблице можно поместить базовый адрес в индексный регистр и увеличивать его на единицу число раз, равное значению индекса или исходного числа. Если считать, что индекс находится в ячейке 40, получается следующая программа:

	LDX	# BASE	Загрузить базовый адрес таблицы
	LDAA	\$ 40	Счетчик = индекс
LOOKT	BEQ	DONE	
*	INX		Инкрементировать базовый адрес
			k раз (k — содержимое счетчика)
	DECA		
	BRA	LOOKT	
DONE	LDAB	X	Загрузить элемент таблицы

Очевидно, подобный метод оказывается очень медленным из-за длинных таблиц. Вместо этого можно переслать через память значение индекса и старшие 8 бит базового адреса в индексный регистр и использовать младшие 8 бит базового адреса в качестве смещения. В МП Motorola 6800 нет возможности непосредственно пересылать данные между аккумуляторами и индексным регистром. Пусть BASEU и BASEL обозначают соответственно восемь старших и восемь младших разрядов базового адреса таблицы. Программа имеет следующий вид:

LDAA	#BASEU	Загрузить старшие 8 бит адреса таблицы
STAA	TEMP	
LTAА	\$ 40	Загрузить индекс
STAA	TEMP+1	
LDX	TEMP	Пересылка смещенного адреса в индексный регистр через память
*		
LDAA	BASEL, X	Загрузить элемент из таблицы

Ассемблер может вычислить BASEU и BASEL из следующих выражений:

BASEU	EQU	BASE/256
BUPP	EQU	BASE * 256
BASEL	EQU	BASE — BUPP

Величина BASE/256 — это старшие 8 бит величины BASE (так как деление на 256 равноценно сдвигу на 8 бит вправо, поскольку $256 = 2^8$). Величина BUPP имеет те же значения старших 8 бит, что и BASE, при этом младшие 8 бит равны нулю. Вычитание BUPP из BASE дает величину BASEL, т. е. младшие 8 бит величины BASE.

Программа выборки квадратов из таблицы с использованием второго способа адресации для МП Motorola 6800 имеет вид:

Motorola 6800 пример 9

Шестнадцатиричный адрес ячейки памяти	Мнемонический код операции	Шестнадцатиричный код содержимого памяти
00	LDAA #SQTBU	86
01		00
02	STAA TEMP	97
03		
04	LDAA \$ 40	97
05		40
06	STAA TEMP+1	97
07		10
08	LDX TEMP	DE
09		0F
0A	LDAA SQTBL, X	A6
0B		11
0C	STAA \$ 41	97
0D		41
0E	SWI	3F
0F	TEMP RMB 2	
10		
11	SQTAB FCB 0 (0 ²)	00
12		
13	1 (1 ²)	01
14	4 (2 ²)	04
15	9 (3 ²)	09
16	16 (4 ²)	10
17	25 (5 ²)	19
18	36 (6 ²)	24
	49 (7 ²)	31

Значение SQTAB равно 0011, поэтому SQTBU=00, а SQTBL=11.

Рис. 5.36. Результат ассемблирования программы поиска в таблице квадратов для МП Motorola 6800

Таблица квадратов

LDAA # SQTBU

Загрузить старшие 8 бит адреса таблицы квадратов

STAA TEMP

LDAA \$ 40

Загрузить индекс

STAA TEMP + 1

LDX TEMP

Пересылка смещенного адреса в индексный регистр через память

LDAA SQTBL, X

Загрузить элемент из таблицы квадратов

*

*

*

* STAA	\$ 41	Запомнить результат
SWI		
TEMP RMB	2	
SQTAB FCB	0, 1, 4, 9, 16, 25, 36, 49	
SQTBU EQU	SQTAB/256	
SQUPP EQU	SQTBU * 256	
SQTBL EQU	SQTAB — SQUPP	

Следует обратить внимание на то, что в индексном регистре содержится 8-битный индекс и 8 бит базового адреса

В программе использованы следующие псевдокоманды:

TEMP RMB 2

Эта команда резервирует две ячейки памяти и приспосабливает метку TEMP адресу первой ячейки. Никакие величины в ячейки не посылаются.

SQTAB FCB 0, 1, 4, 9, 16, 25, 36, 49

Эта команда размещает таблицу квадратов в памяти и присписывает метку SQTAB адресу первой ячейки этой таблицы.

На рис. 5.36 приведена программа после ассемблирования. Следует обратить внимание на то, как таблица квадратов размещена в памяти. Не нужно забывать о том, что элементы таблицы закодированы шестнадцатиричными числами. На рис. 5.37 представлена трасса программы доступа к таблице.

После выполнения пятой команды, `LDX TEMP:`

Память данных		Регистры	
40	04	PC	000A
0F	00	A	04
10	04	X	0004

После выполнения шестой команды, LDA SQTBL,X:

Память данных		Регистры	
40	04	PC	000C
0F	00	A	10
10	04	X	0004

$$(A) = ((X) + SQTBL) = (0004 + II)$$

$$= (0015) = 10$$

Рис. 5.37. Трасса программы поиска в таблице квадратов, выполняемой на МП-
Motorola 6800

Intel 8080 (пример 9)

Хотя в МП Intel 8080 отсутствуют средства индексирования, задача нахождения нужного элемента в таблице решается сравнительно просто. Чтобы сложить базовый адрес с индексом, в программе используется 16-битная команда сложения DAD. Сложение реализовано не наилучшим образом, поскольку адрес имеет длину 16 бит, а индекс только 8 бит. Перед использованием команды DAD индекс следует расширить до 16 бит путем обнуления старшей половины регистровой пары. Программа имеет следующий вид:

Intel 8080 пример 9

Шестнадцатиричный адрес ячейки памяти	Мнемонический код операции	Шестнадцатиричный код содержимого памяти
00	LXI D, SQTAB	11
01		0F
02		00
03	LDA 40H	3A
04		40
05		00
06	MOV L, A	6F
07	MVI H, 0	
08		26
09		00
0A	MOV DAD D A, M	19
0B		7E
0C	STA 41H	
0D		32
0E		41
0F	HLT	00
10	SQTAB: DB 0(0 ²)	76
11	1(1 ²)	00
12	4(2 ²)	00
13	9(3 ²)	04
14	16(4 ²)	09
15	25(5 ²)	10
16	36(6 ²)	19
	49(7 ²)	24
		31

Рис. 5.38. Результат ассемблирования программы поиска в таблице квадратов для МП Intel 8080

После выполнения четвертой команды, **MVI H,0**:

Память данных		Регистры	
40	04	PC	0009
		A	04
		D	00
		E	0
		H	00
		L	04

После выполнения пятой команды, **DAD D**:

Память данных		Регистры	
40	04	PC	000A
		A	04
		D	00
		E	0
		H	00
		L	13

После выполнения шестой команды, **MOV A,M**:

Память данных		Регистры	
40	04	PC	000B
		A	10
		D	00
		E	0
		H	00
		L	13

$$\begin{aligned}
 (A) - ((H \text{ и } L)) &= \\
 &= (13) = 10
 \end{aligned}$$

Рис. 5.39. Трасса выполнения на МП Intel 8080 программы поиска в таблице квадратов

Таблица квадратов

```

LXI  D, SQTAB ; Загрузить базовый адрес таблицы
                ; квадратов
LDA  40H      ; Загрузить данные
MOV  L, A     ; Послать данные в младшие 8 бит
                ; индекса
MVI  H, 0     ; Обнулить старшие 8 бит индекса
DAD  D        ; Индекс таблицы квадратов
MOV  A, M     ; Загрузить квадрат данных
STA  41H, H   ; Запомнить квадрат данных
HLT

```

SQTAB: DB 0, 1, 4, 9, 16, 25, 36, 49

Здесь использованы следующие новые команды:

MVI H, 0

Эта команда обнуляет регистр H. Она занимает две ячейки памяти (во второй ячейке помещается ноль) и выполняется за 3,5 мкс.

DAD D

Эта команда прибавляет содержимое регистровой пары D (регистры D и E) к содержимому регистровой пары H (регистры H и L). Результат помещается в регистровую пару H. Команда DAD D занимает одну ячейку памяти и выполняется за 5 мкс.

На рис. 5.38 показан результат ассемблирования данной программы для МП Intel 8080, а на рис. 5.39 приведена трасса доступа к таблице. Базовый адрес таблицы помещен в регистры D и E и при необходимости может использоваться для выборки другого элемента. Если таблица всегда помещается на одной странице длиной 256 ячеек, то приведенную процедуру можно упростить. В этом случае старшие 8 бит у всех адресов одинаковы и можно использовать вместо 16-битной операции сложения 8-битную.

В последних трех примерах рассматривались особенности программирования арифметических операций для микропроцессоров. Используя типовую структуру циклического процесса, можно реализовать двоичную арифметику для чисел, расположенных в нескольких ячейках, а с использованием специальной команды ДЕСЯТИЧНАЯ КОРРЕКТИРОВКА — десятичную арифметику для длинных чисел. Более сложные арифметические задачи могут реализовываться с помощью таблиц; в большинстве микропроцессоров извлечение нужного результата из таблицы требует выполнения нескольких команд.

5.4. ОБРАБОТКА СИМВОЛЬНОЙ ИНФОРМАЦИИ

Во многих областях применения микропроцессоров требуется обработка символьной информации. Это нужно не только при работе с такими распространенными устройствами, как клавишные пульты, телетайпы, печатающие устройства и дисплеи, но и при взаимодействии с ЭВМ, линиями связи и измерительными приборами. В большинстве микропроцессоров для представления символьной информации используется код ASCII. Коды символов в ASCII могут обрабатываться как обычные числа с использованием арифметических операций и операций сравнения. В данном параграфе приводятся примеры анализа строки символов в коде ASCII, преобразования данных в код ASCII и обратно, сравнения строк символов в коде ASCII.

Пример. Длина строки символов в коде ASCII.

Требуется определить длину строки символов в коде ASCII, которая размещается в памяти, начиная с ячейки 41. Конец строки отмечен символом «точка» (шестнадцатиричный код 2E). Результат следует поместить в ячейку 40.

Типичный пример.

- (41) = 43 C;
- (42) = 41 A;
- (43) = 54 T;
- (44) = 2E.

Конец строки обозначен символом «точка». Этот символ расположен в ячейке 44. Длина строки (не считая признака конца) равна 3; таким образом в результате должно получиться

(40) = 03

На рис. 5.40 приведена блок-схема программы. В данном случае выбран другой способ организации цикла, чем ранее, так как число повторений цикла заранее неизвестно. Для простоты предполагается, что в конце каждой строки обязательно имеется символ «точка».

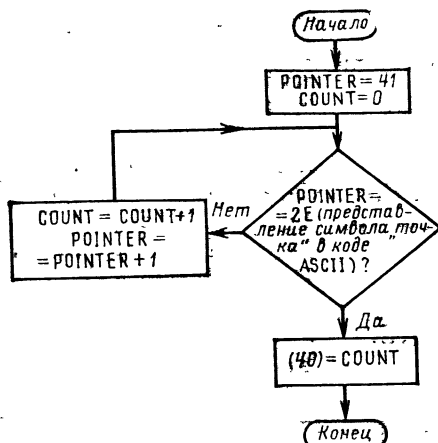


Рис. 5.40. Блок-схема программы определения длины строки символов

Motorola 6800 (пример 10)

В программе для МП Motorola 6800 для обнаружения символа «точка» в коде ASCII используется команда СРАВНИТЬ. Следует обратить внимание на то, что команда СРАВНИТЬ оставляет неизменным содержимое аккумулятора во всех проходах. Программа имеет следующий вид:

Motorola 6800 пример 10

Длина строки

	CLRB		Длина строки = 0
	LDA #		Ввод символа «точка» для сравнения
	LDX #	\$41	Указатель начала строки
CHPER	CMPA X		Текущий символ — точка?
	BEQ	DONE	Да, идти к DONE
	INCB		Нет, увеличить длину строки на 1
	INX		
	BRA	CHPER	
DONE	STAB	\$40	Запомнить длину строки
	SWI		

Апостроф (') означает задание данных в коде ASCII.

На рис. 5.41 показана программа после ассемблирования. Информация в коде ASCII обрабатывается так же, как и всякая другая. Команда BRA (БЕЗУСЛОВНЫЙ ПЕРЕХОД) представляет собой безусловный переход, в котором используется относительная адресация.

Относительный адрес в команде BEQ DONE равен

0E — 0A = 04

Относительный адрес в команде BRA CHPER равен 0,6 — 0E = — 06 + F2 = F8

Intel 8080 пример 10

Длина строки

	MVI	B,0	; Длина строки = 0
	MVI	A,'.'	; Загрузка символа «точка» для сравнения
	LXI	H, 41H	; Начало строки
CHPER:	CMP	M	; Текущий символ—точка?
	JZ	DONE	; Да, идти к DONE
	INR	B	; Нет, увеличить длину строки на 1
	INX	H	
	JMP	CHPER	
DONE:	MOV	A, B	
	STA	40H	; Запомнить длину строки
	HLT		

Шестнадцатичный адрес ячейки памяти	Мнемонический код операции	Шестнадцатичный код содержимого памяти
00	CLRB	5F
01	LDA A #'	86
02		2A
03	LDA #41	CE
04		00
05		41
06	CHPER CMP A X	A1
07		00
08	BEQ DONE	27
09		04
0A	INCB	5C
0B	INX	08
0C	BRA CHPER	20
0D		F8
0E	DONE STAB \$ 40	D7
0F		40
10	SWI	3F

Рис. 5.41. Результат ассемблирования программы определения длины строки символов для МП Motorola 6800

Шестнадцатичный адрес ячейки памяти	Мнемонический код операции	Шестнадцатичный код содержимого памяти
00	MVI B, 0	06
01		00
02	MVI A, '.'	3E
03		2A
04	LXI H, 41H	21
05		41
06		00
07	CHPER: CMP M	BE
08	JZ DONE	CA
09		10
0A		00
0B	INR B	04
0C	INX H	23
0D	JMP CHPER	C3
0E		07
0F		00
10	DONE: MOV A, B	78
11	STA 40H	32
12		40
13		00
14	HLT	76

Рис. 5.42. Результат ассемблирования программы определения длины строки для МП Intel 8080

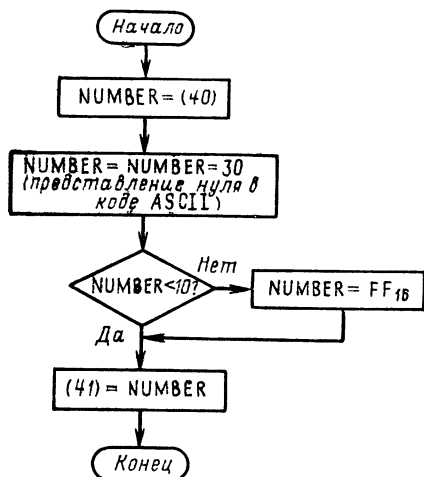


Рис. 5.43. Блок-схема алгоритма преобразования цифр из кода ASCII в десятичную систему

Шестнадцатеричный адрес ячейки памяти	Мнемонический код операции	Шестнадцатеричный код содержимого памяти
00	LDAА \$40	96
01		40
02	SUBА #0	80
03		30
04	СМРА #10	81
05		0А
06	BCS DONE	25
07		02
08	LDAА #\$FF	86
09		FF
0А	DONE STAA \$41	97
0В		41
0С	SWI	3F

Рис. 5.44. Результат ассемблирования программы преобразования цифр из кода ASCII в десятичную систему для МП Motorola 6800

В апострофы заключены данные в коде ASCII. На рис. 5.42 приведен результат ассемблирования программы для МП Intel 8080.

Пример 11. Преобразование из кода ASCII в десятичную систему.

Часто в программе требуется до начала обработки преобразовать числовые данные, представленные в коде ASCII, в десятичную или двоичную форму или выполнить обратное преобразование перед выдачей данных на печать. Такое преобразование выполнить просто, так как коды ASCII сохраняют числовую упорядоченность цифр. Чтобы преобразовать цифру из кода ASCII в десятичную форму, достаточно вычесть из ее кода значение 30_{16} (ноль в коде ASCII). Обратное преобразование выполняется путем прибавления нуля в коде ASCII. В программе следует также проверить, является ли преобразуемый символ в коде ASCII цифрой. Пусть символ в коде ASCII находится в ячейке памяти 40, результат преобразования в десятичную форму посылается в ячейку 41. Если преобразуемый символ не есть цифра, то программа помещает в ячейку 41 код FF_{16} .

Типичные примеры.

1. $(40) = 36$; 36 в коде ASCII соответствует десятичной цифре 6
В результате $(41) = 0,6$;

2. $(40) = 5E$; 5E в коде ASCII не является цифрой. Поэтому $(41) = FF$.

На рис. 5.43 приведена блок-схема данной программы. Процедура заключается в вычитании из исходного значения нуля в коде ASCII.

Если результат меньше десяти, то исходное значение кода соответствует цифре и получен ее десятичный эквивалент. В противном случае исходное значение не является цифрой в коде ASCII и программа помещает в ячейку результата код ошибки (FF₁₆).

Motorola 6800 (пример 11)

Программа для МП Motorola 6800, преобразующая цифровой символ из кода ASCII в десятичный, написана в строгом соответствии с блок-схемой. На рис. 5.44 приведен результат ее ассемблирования.

Motorola 6800 пример 11

Преобразование из ASCII в десятичный код

LDA	\$40	Загрузить символ в код ASCII
SUBA	#0	Вычесть ноль в код ASCII
CMPA	#10	Результат меньше 10?
BCS	DONE	Да, запомнить десятичную цифру
LDA	#\$FF	Нет, результат=FF (символ не есть десятичная цифра)
* DONE	STAA \$41	Запомнить результат
	SWI	

Intel 8080 (пример 11)

Для МП Intel 8080 программа преобразования цифровых символов из кода ASCII в десятичный также проста. Результат ее ассемблирования приведен на рис. 5.45.

Intel 8080 пример 11

Преобразование из кода ASCII в десятичный

LDA	40 H	; Загрузить символ в код ASCII
SUI	0	; Вычесть ноль в код ASCII
CPI	10	; Результат меньше 10?
JC	DONE	; Да, запомнить десятичную цифру
MVI	A, 0FFH	; Нет, результат=FF (символ не есть десятичная цифра)
DONE:	STA 41 H	; Запомнить результат
	HLT	

Пример 12. Сравнение символьных строк.

Во многих задачах требуется распознавать строки символов в коде ASCII. Эти строки могут представлять собой команды, идентификаторы, имена, сообщения или числа. Распознавание состоит в проверке совпадения некоторой символьной строки с заданной. Пусть две строки символов расположены в памяти, начиная с ячеек 42 и 52 соответственно. В ячейке 41 указана длина строк. Программа посылает в ячейку 40 ноль, если строки совпадают, и FF₁₆ в противном случае.

Типичный пример.

(41) = 04;
 (42) = 43 C;
 (43) = 41 A;
 (44) = 54 T;
 (45) = 53;
 (52) = 43 C;
 (53) = 41 A;
 (54) = 50 P;
 (55) = 45 E.

Две данные строки символов не идентичны, так как третий и четвертый символы в них не совпадают. Результат сравнения равен

(40) = FF.

На рис. 5.46 показана блок-схема программы. Очевидно, что эта программа аналогична программе пересылки данных (см. пример 5) и программам выполнения арифметических операций над длинными числами (примеры 7 и 8). В данном случае имеется два выхода из цикла:

Шестнадцатичный адрес ячейки памяти	Мнемонический код операции	Шестнадцатичный код содержимого памяти
00	LDA 40H	3A
01		40
02		00
03	SUI '0'	D6
04		30
05	CPI 10	FF
06		0A
07	JC DONE	A
08		0C
09		00
0A	MVI A, OFFH	3E
0B		FF
0C	DONE STA 41H	32
0D		41
0E		00
0F	HLT	76

Рис. 5.45. Результат ассемблирования программы преобразования цифр из кода ASCII в десятичную систему для МП Intel 8080

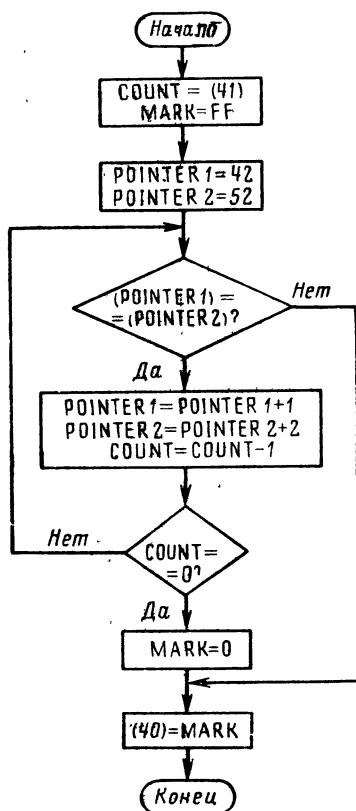


Рис. 5.46. Блок-схема программы сравнения строк символов

один при несовпадении очередной пары символов, второй — при тождественности строк. В этой программе использованы все те приемы, которые иллюстрировались примером 10, а также примерами 4—8.

Motorola 6800 (пример 12)

В программе сравнения двух строк символов для МП Motorola 6800 предполагается, что строки расположены в памяти на расстоянии не более 256 ячеек друг от друга. Это предположение позволяет указывать расстояние между строками в виде 8 -битного смещения. На рис. 5.47 приведен результат ассемблирования программы. Следует обратить внимание на то, что в команде CLR \$40 требуется задавать 16-битный адрес (0040), даже если ячейка 40 находится на нулевой странице. В МП Motorola 6800 для любой из команд с одним операндом ОЧИСТИТЬ, ДОПОЛНЕНИЕ, ОТРИЦАНИЕ, ЦИКЛИЧЕСКИЙ СДВИГ ВЛЕВО, ЦИКЛИЧЕСКИЙ СДВИГ ВПРАВО, АРИФМЕТИЧЕСКИЙ СДВИГ ВЛЕВО, АРИФМЕТИЧЕСКИЙ СДВИГ ВПРАВО И ТЕСТ

Шестнадцатиричный адрес ячейки памяти	Мнемонический код операции	Шестнадцатиричный код содержимого памяти
00	LDAA #\$SFF	86
01		FF
02	STAA \$40	97
03		40
04	LDAB \$41	D6
05		41
06	LDX #\$42	CE
07		00
08		42
09	PATTC LDAA X	A6
0A		00
0B	CMPA \$10, X	A1
0C		10
0D	BNE DONE	26
0E		07
0F	INX	08
10	DECB	5A
11	BNE PATTC	26
12		F6
13	CLR \$40	7F
14		00
15		40
16	DONE SWI	3F

Рис. 5.47. Результат ассемблирования программы сравнения строк символов для МП Motorola 6800

нельзя использовать адресацию нулевой страницы. В МП Motorola 6800 любая из этих команд может работать с данными, находящимися в памяти (в отличие от МП Intel 8080), но при этом в команде должен быть задан 16-битный прямой адрес или индексируемое смещение. Motorola 6800 пример 12

Сравнение строк символов

	LDAA	#\$FF	
	STAA	\$40	Метка = FF (несовпадение)
	LDAB	\$41	Счетчик = длина строк
	LDX	#\$42	Начало строки
PATTC	LDAA	X	Загрузить элемент строки 1
	CMPA	\$10,X	Совпадает с элементом строки 2?
	BNE	DONE	Нет, строки не идентичны
	INE		
	DECB		Все символы проанализированы?
	BNE	PATTC	Нет, сравнить следующие символы
	CLR	\$40	Да, метка = 0 (совпадение)
DONE	SWI		

Intel 8080 (пример 12)

Программа для МП Intel 8080 позволяет располагать сравниваемые строки в любом месте памяти. Результат ассемблирования этой программы показан на рис. 5.48.

Intel 8080 пример 12

Сравнение строк символов

	MVI	C, 0FFH	; Метка = FF (несовпадение)
	LXI	H, 41H	; Счетчик = длина строк
	MOV	B, M	
	LXI	D, 51H	; Начало строки 2
PATTC:	INX	H	
	INX	D	
	LDAX	D	; Загрузить элемент строки 2
	CMP	M	; Совпадает с элементом строки 1?
	JZ	DONE	; Нет, не совпадает
	DCR	B	; Все символы проанализированы?
	JNZ	PATTC	; Нет, сравнить следующие символы
	MVI	C, 0	; Да, метка = 0 (совпадение)
DONE:	MOV	A, C	
	STA	40H	; Запомнить метку
	HLT		

В данном параграфе была рассмотрена обработка символьной информации. Было продемонстрировано, как отыскать нужный символ в строке, как преобразовать данные из символьного представления в цифровое и обратно, как сравнить две строки символов. Простота соответствующих программ свидетельствует о том, что 8-битные процессоры хорошо приспособлены для обработки символьной информации.

Шестнадцатиричный адрес ячейки памяти	Мнемонический код операции	Шестнадцатиричный код содержимого памяти
00	MVI C, 0FFH	0E
01		FF
02	LXI H, 41H	21
03		41
04		00
05	MOV B, M	46
06	LXI D, 51H	11
07		51
08		00
09	PATTC: INX H	23
0A	INX D	13
0B	LDAX D	1A
0C	CMP M	BE
0D	JZ DONE	CA
0E		16
0F		00
10	DCR B	05
11	JNZ PATTC	C2
12		09
13		00
14	MVI C, 0	0E
15		00
16	DONE: MOV A, 0	79
17	STA 40H	32
18		40
19		00
1A	HLT	76

Рис. 5.48. Результат ассемблирования программы сравнения строк символов для МП Intel 8080

При этом программист может воспользоваться тем, что символы закодированы так, что как цифры, так и буквы оказываются упорядоченными естественным образом

5.5. ПОДПРОГРАММЫ

Любую программу можно превратить в подпрограмму, пометив ее первую команду или точку входа и поставив в конце программы команду RETURN (ВОЗВРАТ). Главная программа может вызывать подпрограмму и так устанавливать указатель стека, чтобы обеспечить правильное сохранение адресов возврата.

Однако если подпрограмму написать специальным образом, то ею будет удобнее пользоваться и не возникает никаких ограничений на размещение ее в памяти. В простейшем случае главная программа перед

обращением к подпрограмме размещает исходные данные и адреса в регистрах; подпрограмма, в свою очередь, помещает полученные результаты в регистры. Данные и адреса, которые требуются подпрограмме, называются *параметрами*; процесс, делающий параметры доступными подпрограмме, называется *передачей параметров*. Для работы более сложных подпрограмм может потребоваться слишком много параметров, чтобы их можно было разместить в доступных регистрах. Такие подпрограммы могут использовать для хранения параметров и результатов специально выделенную область памяти или стек.

К реентерабельным подпрограммам можно обращаться при обработке прерываний, при этом сохраняется возможность продолжить правильное выполнение прерванной программы. Такие подпрограммы используют для передачи параметров, результатов и для рабочих ячеек либо стек, либо области памяти, выделенные в каждой вызывающей программе. Использование стека упрощает задачу, особенно в тех случаях, когда имеется несколько вложенных подпрограмм или прерываний.

Программу преобразования из кода ASCII в десятичный (см. пример 11) удобно оформить в виде подпрограммы. Очевидно, что для любой программы, которая вводит числа с телетайпа, требуется такое преобразование. Чтобы превратить рассмотренные в предыдущих примерах программы в подпрограммы, достаточно присвоить метку первой команде (например, ASDEC) и заменить последнюю команду на BO3-BPAT (RET в МП Intel 8080 или RTS в МП Motorola 6800). Модифицированные соответствующим образом программы для МП Intel 8080 и МП Motorola 6800 оформлены в виде примера 13.

Пример 13. Подпрограмма преобразования из кода ASCII в десятичный простейший вариант.

Motorola 6800 пример 13

Простейшая подпрограмма преобразования из кода ASCII в десятичный

ASDEC	LDA	\$40	Загрузить символ ASCII
	SUBA	#'0	Вычесть 0 в коде ASCII
	CMPA	#10	Результат меньше 10?
	BCS	DONE	Да, запомнить десятичную цифру
	LDA	#\$FF	Нет, результат = FF (не десятичная цифра)

DONE	STAA	\$41	Запомнить результат
	RTS		

Intel 8080 пример 13

Простейшая подпрограмма преобразования из кода ASCII в десятичный

ASDEC:	LDA	40H	; Загрузить символ ASCII
	SUI	0	; Вычесть ноль в коде ASCII
	CPI	10	; Результат меньше 10?
	JC	DONE	; Да, запомнить десятичную цифру
	MVI	A, OFFH	; Нет, результат = FF (не десятичная цифра)
DONE:	STA	41H	; Запомнить результат
	RET		

Главная программа должна поместить исходные данные в ячейку 40, вызвать подпрограмму и интерпретировать полученный результат. Пример 14 иллюстрирует, как происходит обращение к подпрограмме. В МП Motorola 6800 имеются две команды вызова подпрограммы: BSR, использующая относительную адресацию, и JSR, использующая прямую (16-битный адрес) адресацию или индексированную адресацию. В МП Intel 8080 имеются как безусловная, так и условная форма команды CALL и команды RETURN.

Пример 14. Обращение к простейшей подпрограмме преобразования из кода ASCII в десятичный.

Motorola 6800 пример 14

Обращение к простейшей подпрограмме преобразований из кода ASCII в десятичный

STAA	\$ 40	Запомнить символ ASCII для подпрограммы
JCR	ASDEC	Преобразовать в десятичный
LDAA	\$ 41	Результат — цифра?
CMPA	# \$FF	
BEQ	ERROR	Нет, ошибка в данных

Intel 8080 пример 14

Обращение к простейшей подпрограмме преобразования из кода ASCII в десятичный

STA	40H	; Запомнить символ ASCII для подпрограммы
CALL	ASDEC	; Преобразовать в десятичный
LDA	41H	; Результат — цифра?
CPI	OFFH	
IZ	ERROR	; Нет, ошибка в данных

Поскольку команды CALL и JSR помещают адрес возврата в стек, главная программа должна правильно установить значение указателя стека. В МП Motorola 6800 эта операция выполняется по команде LDS (ЗАГРУЗИТЬ УКАЗАТЕЛЬ СТЕКА); в Intel 8080 ее можно выполнить либо по команде LXI SP (ЗАГРУЗИТЬ В УКАЗАТЕЛЬ СТЕКА НЕПОСРЕДСТВЕННЫЙ ОПЕРАНД), либо по команде SPHL (ПЕРЕСЛАТЬ H и L В УКАЗАТЕЛЬ СТЕКА). Обычно стек располагается в старших адресах оперативной памяти. Если, например, оперативная память системы занимает адреса от 1000₁₆ до 13FF₁₆, то начальное значение указателя стека обычно устанавливают равным 1400 для МП Intel 8080 или 13FF для Motorola 6800.

Если при обращении к подпрограмме для передачи параметров и результатов используются регистры, то отпадает необходимость выбирать данные из памяти и посылать их в память. Такую подпрограмму, как преобразование символа из кода ASCII в десятичный, легко модифицировать, так как для ее выполнения требуется передача одного входного параметра и она возвращает в вызывающую программу один выходной параметр. Модифицированные подпрограммы и обращение к ним иллюстрируются в примере 15. Организация передачи параметров поясняется комментариями.

Выполнение подпрограммы может приводить к изменению состояния МП. Почти всегда она меняет состояние всех признаков и содержимое аккумулятора. Она может менять содержимое других регистров и ячеек памяти. В документации должно быть четко определено, какие изменения вызывает выполнение подпрограммы. Очевидно, подпрограммой проще пользоваться, если она меняет состояние только нескольких регистров и в качестве рабочей памяти использует стек.

Пример. 15. Модифицированный вариант подпрограммы преобразования из кода ASCII в десятичный.
Motorola 6800 пример 15

Модифицированный вариант преобразования из кода ASCII в десятичный

*

- * Подпрограмма ASDEC преобразует
- * символ ASCII из аккумулятора A
- * в десятичную цифру в аккумуляторе A.
- * Если в аккумуляторе был не цифровой символ ASCII,
- * то ASDEC помещает в аккумулятор A код FF.
- * Используемые регистры: A.

*

ASDEC	SUBA	#'0	Вычесть 0 в коде ASCII
	CMPA	# 10	Результат меньше 10?
	BCS	DONE	Да, преобразование сделано
	LDA	# \$FF	Нет, установить признак ошибки

DONE RTS

* Пример обращения к подпрограмме

LDA	ASDAT	Загрузить символ ASCII
JSR	ASDEC	Преобразовать в десятичный
CMPA	# \$FF	Ошибка?
BEQ	ERROR	Да, переход к процедуре обработки ошибок

*

Intel 8080 пример 15

Модифицированный вариант преобразования из кода ASCII в десятичную систему

;

- ; Подпрограмма ASDEC преобразует
- ; символ ASCII из аккумулятора A
- ; в десятичную цифру в аккумуляторе A
- ; Если в аккумуляторе был не цифровой
- ; символ ASCII, то ASDEC помещает
- ; в аккумулятор A код FF.
- ; Используемые регистры: A

ASDEC:	SUI	'0'	; Вычесть 0 в коде ASCII
	CPI	10	; Результат меньше 10?
	JC	DONE	; Да, преобразование сделано
	MVI	A, OFFH	; Нет, установить признак ошибки
DONE:	RET		

; Пример обращения к подпрограмме		
LDA	ASDAT	; Загрузить символ ASCII
CALL	ASDEG	; Преобразовать в десятичный
CPI	OFFH	; Ошибка?
JZ	ERROR	; Да, переход к процедуре обработки ; ошибки

Подпрограмма преобразования из кода ASCII в десятичную систему имеет один входной и один выходной параметр и не использует рабочую память. В программе разделения слова на две части (см. пример 3) получаются два результирующих значения и для хранения копии исходных данных используется рабочая память. Пример 16 иллюстрирует, как преобразовать эту программу в подпрограммы для МП Motorola 6800 и Intel 8080.

В программе для МП Motorola 6800 для хранения результата используются два аккумулятора, а в качестве рабочей памяти — стек. Для работы со стеком используются команды PSH и PUL.

Команда PSH помещает содержимое аккумулятора в ячейку памяти, на которую ссылается указатель стека, и декрементирует указатель стека:

$$\begin{aligned} ((SP)) &= (A \text{ или } B) \\ (SP) &= (SP) - 1 \end{aligned}$$

Команда PUL инкрементирует указатель стека и помещает содержимое ячейки памяти, на которую ссылается указатель стека, в аккумулятор:

$$\begin{aligned} (SP) &= (SP) + 1 \\ (A \text{ или } B) &= ((SP)) \end{aligned}$$

Пример 16. Подпрограмма разделения слова на части.
Motorola 6800 пример 16

Подпрограмма разделения слова на части

- * Подпрограмма WDDIS расчленяет слово
- * из аккумулятора A на две шестнадцатичные
- * цифры, помещаемые в аккумуляторы A и B.
- * Старшие 4 бита аккумулятора A помещаются
- * в младшие 4 бита аккумулятора A.
- * Младшие 4 бита аккумулятора A помещаются
- * в младшие 4 бита аккумулятора B.
- * Все остальные биты обнуляются.
- * Используемые регистры: A, B.
- *

WDDIS	PSHA		Копирование данных в стек
	ANDA	#%00001111	Маскирование старших 4 бит
	TAB		Младшая цифра → B
	PULA		Старшая цифра → A
	LSRA		Логический сдвиг
	LSRA		вправо

LSRA
LSRA
RTS

разряда

Подпрограмма WDDIS изменяет содержимое регистра В. Содержимое этого регистра можно сохранить, если перед обращением к WDDIS запомнить его в стеке, т. е. выполнить команды

PSHB
JSR WDDIS
PULB

В программе для МП Intel 8080 для передачи второго выходного параметра используется еще один регистр, одновременно он используется в качестве рабочей памяти. В этом случае подпрограмма меняет содержимое регистров В и С. Если это потребуется, можно запомнить хранимые в этих регистрах данные одной командой PUSH В.

В Intel 8080 команды, работающие со стеком, сохраняют или восстанавливают содержимое регистровых пар. Команда PUSH декрементирует указатель стека и помещает старшие 8 бит регистровой пары в ячейку памяти, адресуемую указателем стека, а затем повторяет эту операцию над младшими 8 бит регистровой пары. Другими словами,

$(SP) = (SP) - 1$

$((SP)) = (\text{старшая половина регистровой пары})$

$(SP) = (SP) - 1$

$((SP)) = (\text{младшая половина регистровой пары})$

Команда PUL помещает содержимое ячейки памяти, адресуемой указателем стека, в младшие 8 бит регистровой пары, инкрементирует указатель стека, а затем повторяет ту же операцию для старших 8 бит.

$(\text{Младшая половина регистровой пары}) = ((SP))$

$(SP) = (SP) + 1$

$(\text{Старшая половина регистровой пары}) = ((SP))$

$(SP) = (SP) + 1$

Intel 8080 пример 16

Подпрограмма разделения слова на части

- ; Подпрограмма WDDIS расчленяет слово
- ; из аккумулятора на две шестнадцатиричные
- ; цифры, помещаемые в регистры А и В.
- ; Старшие 4 бит аккумулятора помещаются
- ; в младшие четыре разряда регистра В.
- ; Младшие 4 бит аккумулятора помещаются
- ; в младшие 4 разряда регистра С.
- ; Все остальные биты обнуляются.
- ; Используемые регистры: А, В, С.


```

WDDIS:  MOV    B, A
        ANI    00001111B    ; Обнулить старшие 4 бит
        MOV    C, A          ; Запомнить младшую цифру
        MOV    A, B          ; Восстановить исходное слово
        RRC     ; Сдвиг вправо
        RRC     ; на
        RRC     ; четыре
        RRC     ; разряда
        ANI    00001111B    ; Обнулить младшие 4 бит
        MOV    B, A          ; Запомнить старшую цифру
        RET

```

Многие подпрограммы работают с большим количеством данных. Попробуйте, например, превратить в подпрограмму программу поиска максимального элемента (пример 6). Если подпрограмма имеет дело с данными, расположенными в определенном месте памяти, то перед обращением к подпрограмме их следует поместить туда. Более разумно не делать этого, а поместить в регистры начальный адрес массива, его длину и найденное максимальное значение. Пример 17 является иллюстрацией того, каким образом подобная подпрограмма реализуется в МП Motorola 6800 и Intel 8080. Логика работы этой подпрограммы немного другая, чем у программы из примера 6. Отличие состоит в том, что в данной подпрограмме первый элемент массива обрабатывается так же, как и все остальные.

В подпрограмме для МП Motorola 6800 используются оба аккумулятора и индексный регистр. К сожалению, перед обращением к подпрограмме невозможно просто сохранить в стеке прежнее содержимое индексного регистра. Поэтому необходимо запоминать его в фиксированных ячейках памяти. Подпрограмма изменяет содержимое регистра В: при возврате в вызывающую программу его содержимое равно нулю.

В подпрограмме для МП Intel 8080 в качестве регистровой пары используется регистровая пара H и L. Используя команду PUSH H, можно легко сохранить в стеке содержимое регистров H и L. Для обоих микропроцессоров каждое обращение к подпрограмме требует выполнения нескольких подготовительных команд, которые обеспечивают засылку в регистры соответствующих значений параметров. Если содержимое регистров сохраняется, то обе подпрограммы являются реентерабельными, так как они не используют фиксированных ячеек памяти.

Некоторые подпрограммы требуют задания большего числа параметров. Так, для программы сложения длинных чисел требуется задать:

- 1) начальные адреса исходных массивов,
- 2) длину массивов,
- 3) начальный адрес массива, куда помещается результат, если он не совпадает с одним из исходных.

В микропроцессорах с ограниченным числом регистров манипулирование большим количеством параметров оказывается затруднительным. Без использования большого числа вспомогательных команд процедуры, имеющие много параметров, редко можно сделать реентерабельными. Если для передачи параметров используется стек, то про-

граммист должен обратить особое внимание на адрес возврата (который по команде CALL помещается на вершину стека): его нельзя уничтожать или помещать в такое место, откуда его сложно будет извлечь.

Пример 17. Подпрограмма поиска максимума.

Motorola 6800 Пример 17

Подпрограмма поиска максимума

```

*
* Подпрограмма MXVAL отыскивает самое большое
* число в массиве 8-битных чисел без знака и
* помещает его в аккумулятор А.
* Начальный адрес массива задается в
* индексном регистре, а длина массива в
* аккумуляторе В.
* Массив должен содержать не менее
* одного элемента.
* Используемые регистры: А, В, Х.
*
MXVAL  LDAA    X           Максимум = текущий элемент
MAXM   DECB
       BEQ     DONE
       INX
       CMPA   X           Максимум больше текущего элемента?
       BCC    MAXM        Да, идти к следующему элементу
       BRA    MXVAL       Нет, заменить максимум на текущий
                               элемент
DONE    RTS
*

```

*Пример обращения к подпрограмме

```

LDAB  COUNT  Задать длину массива
IDX   #BASE  Задать базовый адрес
JSR   MXVAL  Найти максимум

```

Intel 8080 пример 17

Подпрограмма поиска максимума

```

;
; Подпрограмма MXVAL отыскивает наибольшее значение
; в массиве 8-битных целых чисел без знака и
; помещает его в аккумулятор А.
; Начальный адрес массива задается в регистрах H и L,
; а длина массива в регистре В.
; Массив должен иметь не менее одного элемента.
; Используемые регистры: А, В, H, L

```

```

MXVAL: MOV    A, M        ; Максимум = текущий элемент
MAXM:  DCR    B
       JZ     DONE
       INX    H
       CMP    H           ; Максимум больше текущего
                               ; элемента?

```

```
JNC  MAXM      ; Да, идти к следующему элементу
JMP  MXVAL     ; Нет  заменить  максимум  теку-
                ; щим элементом
```

```
DONE:  RET
```

;Пример обращения к подпрограмме

```
LDA  COUNT     ; Задать длину массива
MOV  B, A
LXI  H, BASE   ; Задать базовый адрес
CALL MXVAL     ; Найти максимум
```

Для передачи параметров можно использовать некоторую область памяти. Однако задание постоянной области делает подпрограмму не-реентерабельной, а при использовании указателя области памяти, помещаемого в адресный регистр, возникают неудобства, связанные с его загрузкой и сохранением содержимого. С точки зрения программиста может оказаться более удобным располагать параметры в рабочей области памяти перед обращением к подпрограмме и предоставлять ей возможность свободно распоряжаться этой рабочей областью.

В примере 18 иллюстрируется реализация на МП Intel 8080 подпрограммы, которая суммирует два длинных числа и запоминает результат. Для передачи параметров используются пять регистров. В подпрограмме использована полезная для различных расширений команда PCHL, которая выполняет загрузку в счетчик команд содержимого регистровой пары H и L, а также команда XTHL, которая осуществляет обмен информацией между вершиной стека и регистрами H и L, что позволяет освободить дополнительный адресный регистр.

Пример 18. Подпрограмма сложения длинных чисел.

Intel 8080 пример 18

Подпрограмма сложения длинных чисел

```
; Подпрограмма суммирует два длинных числа
; и помещает результат в память.
; Начальный адрес одного из чисел (и результата)
; находится в регистровой паре H.
; Начальный адрес второго числа находится
; в регистровой паре D.
; Длина чисел указывается в регистре B.
; Используемые регистры: A, B, D, E, H, L.
```

```
MPADD:  SUB     A           ; Перенос = 0
ADD8:   LDAX   D           ; Загрузить 8 бит первого числа
        ADC    M           ; Прибавить 8 бит второго числа
        MOV    M, A        ; Запомнить результат
        INX    D
        INX    H
        DCR    B
        JNZ    ADD8
        RET
```

; Пример обращения к подпрограмме

```
LXI    D, NUM1 ; Начало первого числа
LXI    H, NUM2 ; Начало второго числа
                     ; (и результата)
LDA    COUNT    ; Задать длину
MOV     B, A
CALL   MPADD     ; Сложение длинных чисел
```

В МП Motorola 6800 программа, использующая два указателя адреса, была бы чересчур громоздкой, так как в этом микропроцессоре имеется единственный индексный регистр. В тех же целях можно использовать несколько рабочих ячеек памяти, как это сделано в примере 18 для МП Motorola 6800. Данная программа не является реентерабельной, так как она работает с фиксированной областью памяти TEMPА. Написать реентерабельный вариант этой программы затруднительно. Motorola 6800 пример 18

Подпрограмма сложения длинных чисел

- * Подпрограмма суммирует два длинных двоичных
- * числа и помещает результат в память.
- * Начальный адрес одного из чисел (и результата)
- * находится в индексном регистре.
- * Адрес второго числа записан в ячейках ANUM2
- * и ANUM2 + 1.
- * Длина чисел указывает в аккумуляторе В.
- * Используемые регистры: А, В, Х.

```
MPADD  CLC          Перенос = 0
ADD8   LDAA  X       Загрузить 8 бит первого числа
        STX   TEMPА  Запомнить первый указатель
        IDX  ANUM2   Загрузить второй указатель
        ADCA  X       Прибавить 8 бит второго числа
        INX
        STX   ANUM2  Запомнить второй указатель
        LDX   TEMPА  Восстановить первый указатель
        INX
        DECB
        BNE   ADD8
        RTS
TEMPА  RES    2
*
```

*Пример обращения к подпрограмме

```
LDX    #NUM2  Задать начало второго числа
STX    ANUM2
LDX    #NUM1  Задать начало первого числа
LDAB   COUNT  Задать длину
JSR    MPADD  Сложение длинных чисел
```

Подводя итоги, можно отметить, что в данном параграфе были рассмотрены организация и использование подпрограмм. Техника применения подпрограмм позволяет экономить память и использовать ранее отлаженные и протестированные программы. Программы с небольшим

количеством входных параметров и результатов несложно написать и использовать. Исходные данные и результаты могут передаваться через регистры. Перед обращением к подпрограмме главная программа должна сохранить прежнее содержимое используемых регистров и поместить в них значения входных параметров. Перед обращением к подпрограмме необходимо установить указатель стека. Программы обработки массивов можно преобразовать в подпрограммы, используя регистр для передачи начального адреса массива. В качестве рабочей памяти подпрограммы могут использовать стек. В более сложных задачах требуется тщательная проработка вопроса использования подпрограмм. Следует иметь в виду, что при использовании подпрограмм затраты времени и памяти начинают быстро расти, как только объем данных начинает превышать непосредственные возможности процессора.

5.6. ВЫВОДЫ

В данной главе были рассмотрены методы программирования микропроцессоров Intel 8080 и Motorola 6800 на языке ассемблера, методы организации циклов, обработки массивов, способы реализации вычислительных операций, обработки символьной информации и организации подпрограмм. Хотя рассматривалось программирование только для двух конкретных типов микропроцессоров, большая часть предложенных приемов пригодна при программировании других микропроцессоров. Можно сформулировать следующие общие рекомендации.

Простейшие программы

1. Размещать исходные данные в регистрах и выполнять обработку желательнее без обращений к памяти, так как на такие обращения расходуется много времени.
2. Для выделения из содержимого ячейки части данных в виде, удобном для последующей обработки, целесообразно применять логические команды и команды сдвига.

Обработка массивов и организация циклов

3. Необходимо использовать простые структуры цикла. Для управления числом повторений цикла целесообразно применять счетчики и команды условного перехода.
4. При обработке массивов целесообразно устанавливать начальное значение адреса массива, а затем просматривать его элементы, инкрементируя при каждом проходе индексный или адресный регистр.

Арифметические операции

5. Простые арифметические операции целесообразно реализовывать побайтно, используя для переноса информации между ячейками значение признака ПЕРЕНОС. Во многих процессорах имеются специальные команды для десятичной арифметики.
6. При выполнении более сложных вычислений разумно использовать табличную форму реализации операций. Подобная таблица, размещенная в памяти программ, содержит все возможные результаты операций. В результате выполнение вычислительной операции заменяется поиском нужного элемента в таблице.

Обработка символьных данных

7. Символьные данные следует обрабатывать так же, как любую другую двоично-кодированную информацию. Для выполнения преобразований необходимо использовать двоичную арифметику или таблицы.
8. Для упрощения процессов обработки следует использовать естественную упорядоченность кодов цифр и букв.

9. Простейшие программы можно преобразовывать в подпрограммы, если ввести символическую метку точки входа и команду ВОЗВРАТ. Использование регистров для передачи параметров делает программы более универсальными.

10. Чтобы сделать подпрограммы реентерабельными, для передачи параметров и хранения промежуточных и окончательных результатов следует использовать регистры и стек.

ГЛАВА ШЕСТАЯ

РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ МИКРОПРОЦЕССОРА

В данной главе будут рассмотрены различные этапы разработки программного обеспечения (ПО) для микропроцессоров и методы, которые можно использовать на каждом этапе разработки. Рассматриваемый круг вопросов включает в себя постановку задачи, проектирование программ, кодирование, отладку, тестирование, документирование, сопровождение и повторное проектирование. Основное внимание уделено не изложению какого-то одного метода разработки, а описанию самых разных методов и полезных рекомендаций. Проектировщики ПО должны представлять себе весь комплекс проблем, возникающих при разработке ПО, и уметь выбирать те методы разработки, которые окажутся наиболее простыми и эффективными. В данной главе приведен краткий обзор многих современных методов программирования. Следует, однако, иметь в виду, что ни один из них не является панацеей. В заключение рассмотрены некоторые системы разработки и аппаратные средства тестирования, которые широко используются при проектировании микропроцессорных систем.

6.1. ЗАДАЧИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Многие студенты и авторы книг часто путают разработку ПО с кодированием. *Кодирование* представляет собой процесс составления программы на языке, который воспринимает ЭВМ. Фактически кодирование — небольшая часть процесса разработки ПО. Хотя в этой области проводилось мало количественных исследований, в нескольких работах показано, что на этап кодирования уходит менее одной пятой общего времени, затраченного на программирование. Остальное время затрачивается на проектирование программ, их тестирование и другие этапы разработки ПО. Довольно распространенным является мнение, что программист, принимающий участие в реализации крупного проекта по созданию ПО, должен иметь производительность от двух до десяти отлаженных операторов в день. Поскольку составление даже десяти операторов требует всего нескольких минут, становится очевидно, что кодирование не представляет собой самый трудоемкий этап разработки ПО.

Разработку ПО можно разделить на несколько этапов.

1. *Постановка задачи.* На этом этапе задача формализуется. Сюда входят определение входов и выходов, определение сущности процессов обработки, формулирование системных ограничений (время выполнения, точность, время реакции) и разработка методов обработки ошибок.

2. *Проектирование программы.* На этом этапе проектируется программа, удовлетворяющая требованиям постановки задачи. При этом целесообразно использовать такие методы, как проектирование сверху вниз, структурное программирование, модульное программирование и создание блок-схемы.

3. *Кодирование (собственно программирование).* На этом этапе спроектированная ранее программа переводится на язык машинных команд. Вопросы программирования на языке ассемблера были рассмотрены в гл. 5, а использование языков высокого уровня — в гл. 4. Вопросы кодирования программ рассматриваются во многих руководствах и справочниках, поэтому в данной главе рассмотрим их очень кратко и уделим основное внимание другим этапам разработки ПО.

4. *Отладка.* На этом этапе осуществляется поиск и исправление программных ошибок (иногда этот процесс называют верификацией). Весьма незначительное число программ работает верно с первого раза, поэтому процесс отладки представляет собой важный и требующий больших затрат времени этап разработки ПО. При отладке полезно прибегать к таким средствам, как пакеты отладки, программные имитаторы, логические анализаторы и контрольные точки. В данной главе рассмотрены наиболее распространенные ошибки в микропроцессорных программах и некоторые простые приемы, позволяющие сократить их число.

5. *Тестирование.* На этом этапе проверяется корректность программы. Тестирование позволяет убедиться в том, что программа правильно выполняет возложенные на нее функции. Очень важным на данном этапе является правильный выбор тестовых данных и разработка методов тестирования.

6. *Документирование.* На этом этапе разрабатывается программная документация, которая дает возможность тем, кто должен использовать и сопровождать программу, разобраться в ее работе с целью расширения программы для других приложений. Для документирования широко используются такие средства, как блок-схемы, программные комментарии и карты памяти.

7. *Сопровождение.* На этом этапе осуществляется корректировка программ при изменении условий или места их использования. Надлежащие средства тестирования и хорошая документация должны существенно уменьшить частоту и трудоемкость операций сопровождения.

8. *Расширение и повторное проектирование.* На этом этапе происходит переработка программы с тем, чтобы она могла решать задачи, не укладывающиеся в рамки исходной постановки. Разумеется, при этом проектировщики всегда стремятся использовать программы, созданные для решения ранее поставленных задач. Проектировщики ПО

не должны рассматривать любую задачу совершенно изолированно от задач, которые могут возникнуть в будущем.

Каждый этап разработки ПО влияет на другие этапы. Постановка задачи должна включать в себя некоторые соображения по поводу плана тестирования, стандарта документирования, методам сопровождения и возможным расширениям на другие задачи. Проект программы должен содержать положения, относящиеся к отладке, тестированию и документированию. В каждый момент времени программист выполняет работы, соответствующие сразу нескольким этапам. Таким образом, кодирование, отладка, тестирование и документирование нередко оказываются тесно переплетенными. Конечно, в рамках одной главы невозможно полностью рассмотреть даже один из этапов, а тем более все этапы сразу. Читатель может найти ответы на интересующие его вопросы в литературе, список которой приведен в конце книги.

Сначала сформулируем критерии оценки качества программ. Эти критерии позволяют определить цели, методы и относительную важность различных этапов разработки ПО. При разработке программ для микропроцессоров следует учитывать следующие факторы.

1. *Надежность.* Самым важным показателем качества программы является надежность ее работы. Если программа не работает, то нет никакой пользы от изящности ее структуры, экономного использования памяти и высокого быстродействия, небольшого срока разработки и полноты ее документации. В этом смысле тестирование является ключевым этапом разработки, на котором фактически устанавливается, работает программа или нет. Следует убедиться в том, что программа работает правильно в тестовых условиях, соответствующих реальным. Выбор и выполнение плана тестирования не является простой задачей. На этапах постановки задачи и проектирования программы следует предусмотреть такой план и разработать такую программу, которая может быть сравнительно просто и исчерпывающим образом протестирована.

2. *Скорость работы.* Более быстрая программа нередко может выполнить большую работу, чем медленная. Скорость работы может служить критерием работоспособности программы, так как иногда время работы является критичным. Если скорость работы всей системы в целом зависит от таких внешних факторов, как время реакции оператора, скорость ввода и вывода данных, или от постоянных времени устройств типа сенсоров, дисплеев, тумблерных коммутаторов или преобразователей, то в этих условиях скорость работы программы будет не столь важна. Нередко в системах, основанных на микропроцессорах, ограничивающим является не время выполнения программы, а перечисленные выше внешние факторы.

3. *Стоимость аппаратных средств.* Каждый дополнительный модуль памяти увеличивает стоимость системы. Разумеется, длинная программа потребует дополнительных модулей памяти только в том случае, если имеющиеся модули уже заполнены (каждый модуль памяти имеет емкость порядка тысяч байт). Подключение дополнительной памяти приводит к дополнительным соединениям, платам, схемам дешифрирования и требует соответствующих источников тока и напря-

жения и средств отвода тепловой энергии. По мере того как становятся доступными полупроводниковые ЗУ с меньшей стоимостью хранения одного бита информации, важность вопроса использования памяти существенно снижается. Однако до сих пор необходимо учитывать требуемый объем памяти, особенно при создании простых средств, когда стоимость каждого модуля оказывается существенной.

Кроме памяти для работы программы могут потребоваться и другие аппаратные средства. Среди них можно отметить буферные элементы, сдвиговые регистры, защелки, счетчики, сдвигатели уровня, приемно-передающие модули, мультиплексоры, демультимплексоры, блоки управления прерываниями, компараторы и преобразователи. Раньше программисты старались минимизировать используемую память и внешние аппаратные средства, так как память и аппаратура стоили дороже программного обеспечения. В настоящее время стоимость памяти и внешних аппаратных средств существенно уменьшилась и поэтому требования к расходу оборудования стали менее важными. С одной стороны, благодаря снижению стоимости памяти и процессоров задачи, которые ранее реализовывались аппаратными средствами, сейчас могут реализовываться программными средствами; с другой стороны, дополнительные аппаратные средства, использование которых позволяет существенно снизить трудоемкость программирования, также доступны и имеют сравнительно низкую стоимость. Следует также отметить, что стоимость ПО остается неизменной независимо от числа экземпляров разработанной системы, в то время как стоимость аппаратных средств пропорциональна числу выпущенных единиц. Разумеется, рациональное соотношение между аппаратурой и ПО в настоящее время отличается от того, которое существовало тогда, когда программисты старались максимально использовать дорогостоящее оборудование.

4. *Время программирования и стоимость создания ПО.* Время и средства, затраченные на создание ПО, являются важными факторами. Хотя процессоры, ЗУ и другие аппаратные средства становятся все дешевле, затраты на программное обеспечение непрерывно растут. Изменение соотношения затрат на аппаратуру и людские ресурсы является главной причиной того, что сейчас много внимания уделяется применению методов структурного программирования и нисходящего проектирования, которые повышают производительность труда программистов. Правильно организованное проектирование, отладка, тестирование и документирование программ позволяют уменьшить совокупные затраты на программирование. Использование при составлении программ языков высокого уровня может существенно повысить производительность труда программиста и упростить многие этапы разработки ПО. Здесь также прослеживается тесная взаимосвязь между затратами на ПО и аппаратные средства: на языке высокого уровня программист может быстрее писать и отлаживать программы, но при этом полученная в результате машинная программа потребует больше памяти, чем программа, написанная на языке ассемблера. Практически нет убедительных результатов, которые позволили бы ответить на вопрос, какие языки и методы проектирования дают на-

илучшие результаты. По мере того как оплата труда программиста будет расти по сравнению с затратами на разработку аппаратных средств, большое значение будет уделяться психологическим исследованиям в области программирования (по-видимому, это случится в недалеком будущем).

5. *Простота использования.* Программа, которую проще использовать другим программистам или пользователям-непрограммистам, имеет большую ценность по сравнению с той, с которой работать сложнее. Жестко заданные и сложные форматы данных, неясные сообщения об ошибках и плохо продуманные формы выдачи результатов усложняют и делают более дорогим процесс отладки, использования и сопровождения программы. Здесь, как и ранее, относительно высокая стоимость труда программистов по сравнению с затратами на аппаратуру делает этот фактор важным. С точки зрения обеспечения удобства использования программы особенно важна роль этапов проектирования и документирования. Учет психологических и физиологических факторов часто оказывается важным, если при использовании программы предполагается взаимодействие ее с человеком. Очень многие микропроцессорные системы предназначены для того, чтобы упростить оператору выполнение типовых задач.

6. *Устойчивость по отношению к ошибкам.* Программу, которая допускает появление ошибок, легче сопровождать, чем программу, не обладающую подобным свойством. Программа должна быть сконструирована таким образом, чтобы обеспечить разумную реакцию даже на те ошибки, появление которых не может быть предусмотрено. Устойчивость по отношению к ошибкам является очень важной характеристикой в тех случаях, когда ввод данных осуществляется оператором или проектируемая программа должна обеспечить выполнение жизненно важных функций. Программа должна уметь оповещать оператора и соответствующий компонент системы об ошибке ввода или неправильной работе, не выводя систему в целом из рабочего состояния.

7. *Расширяемость.* При длительной эксплуатации программа, применение которой может быть расширено на случаи использования, отличные от тех, для которых она была предназначена, окажется лучше программы, которая может использоваться только для выполнения частной задачи. Чтобы программа обладала свойством расширяемости, следует обратить особое внимание на этапы конструирования и документирования. Для достижения поставленной цели целесообразно использовать модульное программирование, хотя структурное программирование и проектирование сверху вниз также позволяют улучшить расширяемость программы.

Надежность и стоимость программирования в настоящее время являются наиболее важными показателями при разработке ПО для микропроцессоров. В качестве начальной цели ставится задача за приемлемое время и с не очень большими затратами создать работоспособную программу. Оптимизацию программы можно выполнить позднее.

В большинстве проектов с использованием микропроцессоров основные затраты обусловлены сроками программирования, поэтому очень важно использовать методы, которые позволяют минимизиро-

вать время, необходимое для завершения разработки программы. Ввиду наличия ЗУ большого размера, более дешевых аппаратных средств и быстродействующих процессоров ограничения по времени выполнения, затратам памяти и расходу оборудования при разработке ПО для микропроцессоров являются не такими важными, как соответствующие факторы при разработке ПО для больших ЭВМ.

В заключение данной главы основное внимание будет уделено тому, каким образом за разумные сроки создавать надежные программы и тестировать и документировать их. Значительно меньше внимания будет уделено вопросу о том, как составлять короткие и быстро работающие программы.

6.2. ПОСТАНОВКА ЗАДАЧИ

Большинство разработок, основанных на применении микропроцессоров, касаются всей системы в целом, а не отдельных задач и поэтому требуют тщательной проработки. Обычно пользователь хочет, чтобы микропроцессор выполнял управление электрической или механической системой (например, весами, терминалом, устройством ввода с перфокарт или осциллографом), осуществлял разнообразные вычисления и выдавал ряд выходных сигналов. В редких случаях пользователю требуется решать некоторый тип уравнения, находить отдельную запись данных или выполнять какую-либо другую четко определенную задачу. На начальном этапе разработки ПО нужно точно определить список выполняемых задач и требований к системе.

Основным вопросом, решаемым при постановке задачи, является форма представления входов и выходов. Следует определить, с какими устройствами будет взаимодействовать микропроцессор и в каком виде они будут посылать и принимать информацию. Основными характеристиками устройств ввода-вывода являются максимальная и средняя скорости обмена данными, процедуры обнаружения ошибок, управляющие сигналы, с помощью которых устройства ввода-вывода сообщают о наличии данных или о готовности принять данные, типичная длина слова, требования к форматированию данных, тактовые или стробирующие сигналы и протоколы. Микропроцессоры часто используются в системах управления, в которых требования к вводу-выводу являются главными при постановке задачи.

Следующий вопрос, решаемый при постановке задачи, касается требований к процессу обработки. Следует четко определить, какие операции должны быть выполнены над входными данными и в каком порядке будут выполняться решаемые в системе задачи. Порядок выполнения операций в ряде случаев должен строго соблюдаться, так как сигналы ввода-вывода должны приниматься и посылаться в строго определенной последовательности. Система может предъявлять требования к временным соотношениям: к скоростям обмена в периферийных устройствах, временам задержек в механических или электрических компонентах, временам, в течение которых данные должны оставаться стабильными, временам стабилизации, восстановления, перехода в состояние готовности и выхода из него. Для выполнения этих ограничений

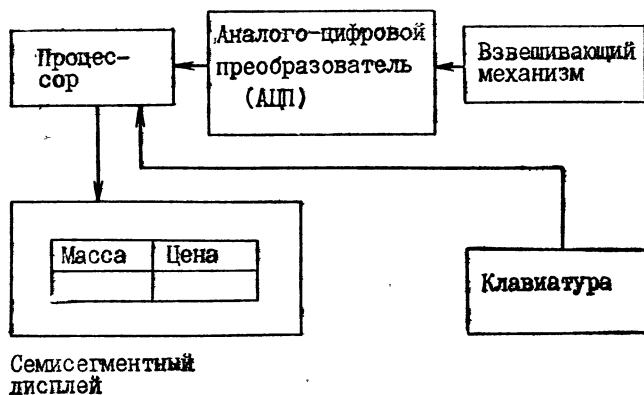


Рис. 6.1. Структурная схема цифровых весов

иногда используются защелки и схемы синхронизации. Размеры доступной памяти могут накладывать ограничения на объем программ и данных и на размер буферов. Часто требуется выполнять обработку данных за определенный промежуток времени, а также успевать реагировать на сигналы о состоянии объектов прежде, чем эти сигналы успеют измениться.

Еще одним вопросом, решаемым при постановке задачи, является обработка ошибок. Следует предусмотреть средства восстановления при возникновении некорректных последовательностей операций или ошибочных сигналов. Типовые ошибки могут быть специальным образом обработаны в программе; другие ошибки можно ликвидировать путем повторного запуска управляющей процедуры. Следует тщательно описать наиболее вероятные ошибки.

В постановку задачи должны быть включены также вопросы взаимодействия с другими программами или таблицами, специфика которых должна быть учтена.

Постановка задачи может учитывать множество факторов. Вопросы ввода-вывода, временные ограничения, требования к процессу обработки, точность, ограничения по памяти, обработка ошибок и связь с другими программами следует рассматривать всегда. Далее приведен простой пример, иллюстрирующий этап постановки задачи.

Пример. Цифровые весы.

Назначение: микропроцессор должен управлять работой весов в соответствии со структурной схемой, показанной на рис. 6.1.

Метод: оператор кладет взвешиваемый продукт на весы и вводит с клавиатуры стоимость единицы товара. С помощью семисегментных дисплеев (аналогичных тем, что используются в калькуляторах) весы показывают общую массу и стоимость продукта.

Вход.

1. Первый входной сигнал поступает от АЦП, подключенного к взвешивающему устройству. Этот входной сигнал обычно является двоичным или двоично-десятичным числом. Процессор должен иници

проводить процесс преобразования, зафиксировать момент его завершения и принять данные от преобразователя (см. § 8.5). Точная последовательность выполнения этих действий зависит от типа используемого преобразователя. Процессор и сам может выполнить некоторые операции, связанные с задачей преобразования. Например, процессор может формировать синхронизирующие сигналы или уровни сравнения (через ЦАП). Это значит, что часть преобразователя может быть выполнена программно.

2. Второй входной сигнал поступает с клавиатуры. Клавиатура состоит из матрицы переключателей. Нажатие клавиши приводит к замыканию контакта и соединению строки и столбца. В § 8.5 описано, каким образом по номерам взаимно соединенных строки и столбца матрицы можно идентифицировать нажатую клавишу.

В данном случае процессор также может взять на себя выполнение некоторых интерфейсных функций. Процессор может стабилизировать положение клавиш (подождать, пока произойдет замыкание контакта — первоначально клавиша будет колебаться около положения равновесия). Процессор может также осуществлять опрос клавиатуры, означать многократные замыкания клавиши и отличать одно длинное замыкание от двух отдельных коротких. Разделение функций между аппаратурой и ПО будет вновь рассмотрено в гл. 8 и 9.

Выход. Существует семисегментный код, используемый для индикации. Результат может быть закодирован с помощью программы или аппаратных средств и послан на элементы соответствующих дисплеев, что дает возможность представить его в удобном для визуального восприятия формате.

Обработка. Программа должна выполнить умножение стоимости единицы продукта на его массу. Перемножаемые числа являются десятичными.

Точность. Масса продукта задана с точностью до сотой доли фунта. Программа должна округлить вычисленный результат до одного пенса.

Временные ограничения. Время реакции системы не должно быть очень большим. В него входит время преобразования в АЦП, время, которое необходимо для отжатия и возврата клавиш, время, затрачиваемое на переключение дисплеев. Общая скорость работы системы не является самой важной характеристикой, так как в любом случае оператор должен набрать входное число, проанализировать и отметить полученный результат. Однако программа должна учитывать физические ограничения, накладываемые устройствами ввода-вывода.

Ограничения по памяти. Требуемая память очень мала, так как программа проста и обрабатывает всего несколько цифровых разрядов. Стоимость взвешивающего устройства, АЦП и системы упаковки будет превышать стоимость процессора и памяти.

Обработка ошибок. При вводе данных возможны самые различные ошибки, которые обусловлены присутствием в системе человека или возникают в электрических цепях. Среди них можно отметить следующие:

1. Ошибки в данных на входе или выходе АЦП.

2. Ошибки в данных, набранных на клавиатуре:

- а) слишком мало разрядов,
- б) слишком много разрядов,
- в) лишние десятичные точки,
- г) все нули,
- д) одновременно нажаты две или более клавиши.

3. Значения массы или суммарной стоимости слишком велики и не помещаются на дисплеях.

4. Сбой в работе дисплея.

5. Сбой в работе процессора.

6. Сбой в работе клавиатуры.

Разумеется, наиболее распространенной является ошибка в данных, вводимых с клавиатуры. Программа может автоматически исправлять последствия некоторых ошибок. Например, она может просто игнорировать лишние разряды, лишние десятичные точки или одновременное нажатие нескольких клавиш. Оператор обнаружит эти ошибки, глядя на дисплей. Программа должна только обеспечить продолжение процесса ввода, а ошибки исправит сам оператор.

Однако некоторые из ошибок не так просто обнаружить. Например, результат может не укладываться в разрядность индикатора. Чтобы предупредить оператора о возникновении ошибки, программа должна предусмотреть в этом случае специальную сигнализацию, например высвечивая в качестве результата все нули или все десятичные точки.

Ошибки в электрических и механических компонентах, естественно, будут возникать реже ошибок набора данных. Неисправности в работе дисплея могут быть замечены не сразу (так, цифры 0 и 8 на семисегментном дисплее отличаются только одним сегментом). Для решения этой проблемы может использоваться специальный режим тестирования, в котором сегменты дисплеев зажигаются так, что оператор может проверить правильность их работы.

Ошибки в взвешивающем устройстве и АЦП могут быть неочевидными. Центральный процессор может сам проконтролировать работу АЦП, подавая на него известный входной сигнал. Для проверки работы взвешивающего устройства требуется иметь стандартную контрольную массу. Программа может предусмотреть задачу специальной индикации для оповещения оператора о таких ошибках.

Ошибки в работе клавишного устройства (заклинивание клавиш или отсутствие контакта при нажатии) могут быть замечены оператором визуально. Выход процессора из строя будет также сразу замечен. Однако возникшую неисправность не всегда просто обнаружить, для этого может потребоваться специальный механизм тестирования.

Следует обратить внимание на то, какой широкий круг проблем должен рассмотреть проектировщик в очень простой системе. Разумеется, хорошая оценка аппаратных и человеческих факторов является решающей при разработке ПО. Другие системы, основанные на микропроцессорах, могут потребовать обеспечения интерфейса с электрическими и механическими устройствами без прямого участия человека.

В этом случае ПО должно обеспечить не только соответствующий интерфейс, но также работоспособность системы, пока это возможно, и возможность обнаружения возникшей ошибки.

6.3. КОНСТРУИРОВАНИЕ ПРОГРАММЫ

После того, как задача в целом поставлена, выполняется этап конструирования программы. Традиционно конструирование программы связывается с созданием *блок-схемы*. Однако фактически блок-схемы более полезны при документировании, чем при проектировании. Несмотря на призывы многих авторов книг, только немногие программисты сначала составляют подробную блок-схему, а затем пишут по ней программу. Обычно программисты полагают, что разработка подробной блок-схемы такая же трудоемкая работа, как описание первоначального варианта работающей программы, причем более бесполезная. Техника разработки блок-схем полезна при описании структуры программы и объяснении ее работы. Однако на этапе разработки программы более эффективны другие методы проектирования программ. К ним относятся:

1. *Модульное программирование*. Это метод, при котором длинные программы расчленяются на короткие, которые называются *модулями* и могут быть спроектированы, закодированы и отлажены отдельно с привлечением минимальных сведений о других программах.

2. *Нисходящее проектирование*. Это метод, при котором разрабатываемая задача расчленяется сначала на несколько обобщенных подзадач, которые затем, в свою очередь, детализируются далее. Этот процесс продолжается сверху вниз до тех пор, пока получаемые подзадачи не оказываются поставленными в таком виде, что их можно просто реализовать на ЭВМ.

Противоположным данному методу является метод проектирования программ снизу вверх, при котором сначала кодируются все подзадачи, а затем происходит их объединение в более крупные части общего проекта.

3. *Структурное программирование*. Согласно этому методу программы пишутся по специально разработанным правилам. Эти правила позволяют использовать только определенные типы программных операторов, хотя для описания сложных процессов допускается включение

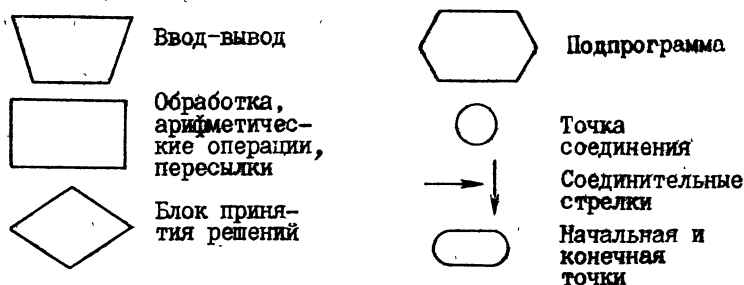


Рис. 6.2. Стандартные символы, используемые в блок-схемах

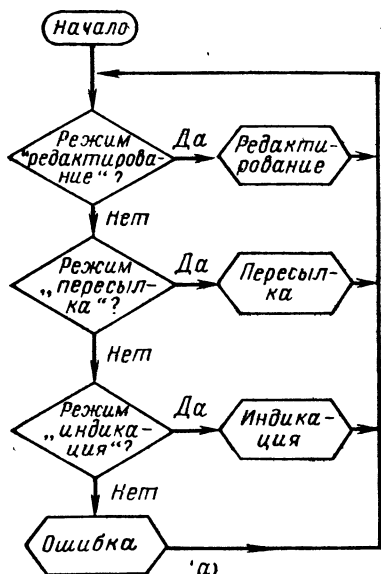
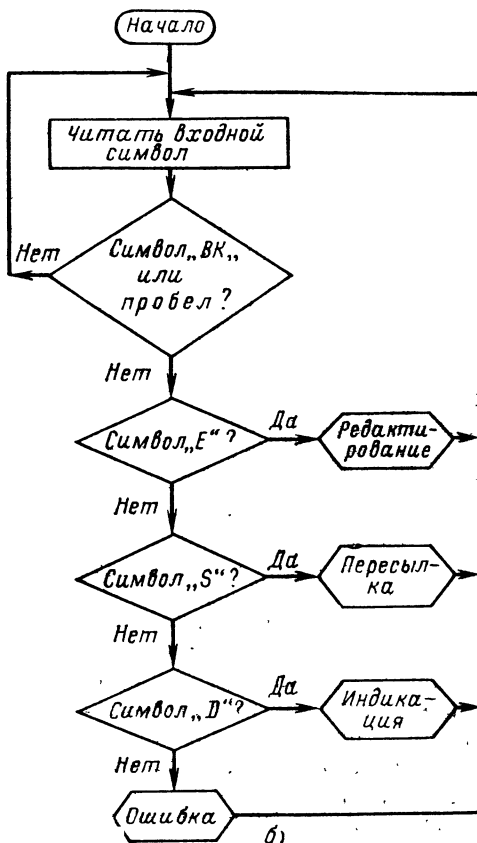


Рис. 6.3. Обобщенная (а) и детальная (б) блок-схема



ние одних операторов внутри других. Структурное программирование часто определяют как метод разработки программ, при котором каждый блок программы имеет только один вход и один выход.

Методы структурного программирования первоначально были разработаны для составления очень длинных программ, но их можно использовать и при создании более коротких программ, которые обычно разрабатываются для микропроцессоров. Сначала рассмотрим технику разработки блок-схемы, а затем остальные методы проектирования программ.

Преимущество блок-схем перед листингами программ состоит в том, что они:

- а) ясно указывают порядок выполнения операций и взаимосвязь блоков программы;
- б) выделяют ключевые точки принятия решений;
- в) не зависят от конкретной ЭВМ или языка программирования;
- г) используют стандартный набор символов (рис. 6.2).

Блок-схема подтверждает утверждение, что образы выразительнее слов. Несмотря на то что блок-схемы имеют ограниченное применение,

они остаются хорошей иллюстрацией, облегчающей процесс программирования. Кроме того, они являются эффективным посредником при общении программистов с непрограммистами.

Обычно желательно иметь блок-схемы двух типов: 1) дающие представление об общей структуре программы и 2) сообщающие подробности, представляющие главный интерес для программистов. На рис. 6.3 приведен пример типичной блок-схемы, описывающей алгоритм работы программы-редактора. Слишком большое число деталей затрудняет составление и восприятие блок-схемы. Слишком подробная блок-схема не имеет особых преимуществ перед листингом программы.

Блок-схемы полезны только тогда, когда они имеют достаточно обобщенный характер. Однако разработка блок-схем связана с дополнительными затратами и они не содержат информации о структуре обрабатываемых данных или используемых аппаратных средств. Блок-схемы описывают только последовательность выполнения операции в программе; они не отражают связи со структурой данных или с аппаратными элементами.

Нисходящее проектирование

Нисходящее проектирование состоит в том, что программы разрабатываются, начиная с системного уровня, путем последовательной замены общих положений конкретными программами. Этот метод проектирования отличается от традиционного метода восходящего проектирования, при использовании которого сначала разрабатываются программы, выполняющие отдельные задачи, а затем они объединяются в общую систему.

Нисходящее проектирование выполняется в следующем порядке.

1. Разрабатывается и тестируется общая управляющая программа, которая вызывает главные подпрограммы. Ненаписанные подпрограммы заменяются программными заглушками, которые либо регистрируют, что произошло обращение к подпрограмме, либо формируют результат, который ожидается при работе будущей программы. Программная заглушка должна приводить к появлению таких же результатов, как будущая программа.

2. Выполняется детализация каждой программной заглушки. На каждом этапе, где заглушка заменяется работающей программой, должны выполняться отладка и тестирование.

3. Тестируется вся система в целом.

Нисходящее проектирование имеет то преимущество, что процесс тестирования и включения элементов в систему происходит в течение всего периода разработки, а не в конце. При этом несоответствия в программе могут обнаруживаться ранее. Тестирование может проводиться в реальной системной обстановке и не требует составления специальных вспомогательных программ (например, программы генерации чисел, которые будут перемножаться в программе, обслуживающей весы), которые впоследствии должны быть исключены из системы.

Нисходящее проектирование имеет и свои недостатки. Проектирование всей системы в целом часто не позволяет использовать возмож-

ности аппаратуры; при таком подходе может оказаться, что на аппаратуру возложены операции, которые она выполняет неэффективно, например, выполнение операций над 16-битными данными на 12-битном процессоре. Реализация метода нисходящего проектирования программ затруднена в том случае, когда одна и та же задача возникает в нескольких различных местах; программа, которая выполняет эту задачу, должна быть должным образом привязана к каждому из этих мест. Может быть затруднено написание подходящей программной заглушки. Не все программы имеют простую древовидную структуру, которая хорошо согласуется с методом нисходящего проектирования. Могут возникнуть сложности и в связи с использованием несколькими программами общих данных.

Ошибки на верхнем уровне разрабатываемой системы, возникшие на этапе проектирования или кодирования программы, могут иметь катастрофические последствия для всего проекта. Нисходящее проектирование позволяет существенно повысить производительность труда программиста. Однако, применяя этот метод, не следует доходить до крайностей, чтобы не помешать созданию надежных программ или эффективному использованию возможностей некоторого конкретного процессора.

Техника нисходящего проектирования может быть использована при создании упоминавшихся ранее цифровых весов:

1. Разрабатывается общая блок-схема (рис. 6.4), в которой пока не предусмотрена возможность возникновения ошибок.

Программа в первоначальном варианте просто обращается к подпрограмме ввода информации от АЦП (программная заглушка) и, если введенные данные не равны нулю, обращается к другим подпрограммам (программные заглушки), а затем вновь возвращается к вводу данных через АЦП.

2. Детализируется программная заглушка, которая читает данные с АЦП и выполняет следующие операции (в предположении, что от преобразователя информация поступает в виде трех двоично-десятичных цифр, которые процессор должен выбрать по одной):

а) послать сигнал НАЧАТЬ ПРЕОБРАЗОВАНИЕ в АЦП;

б) проверить линию ПРЕОБРАЗОВАНИЕ ЗАВЕРШЕНО. Ждать пока преобразование не завершится;

в) выбрать одну цифру;

г) проверить, не равна ли цифра нулю;

д) повторить шаги «в» и «г» 3 раза;

е) если все цифры равны нулю, повторить процедуру, начиная с шага «а»;

ж) повторить шаги «а» — «е», чтобы убедиться, что преобразователь получил окончательное значение;

з) если значения входных величин не совпали, повторять шаг «ж» до тех пор, пока значения не станут одинаковыми с учетом точности преобразователя;

и) запомнить окончательное значение введенной величины в памяти.

На рис. 6.5 приведена блок-схема частично расширенной программной заглушки. Шаги «ж» — «и» на блок-схеме не детализированы.

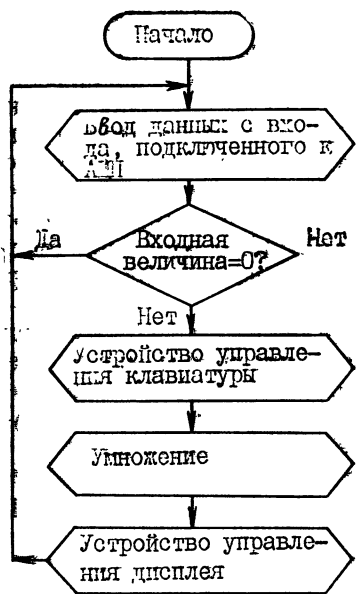


Рис. 6.4. Блок-схема алгоритма работы цифровых весов

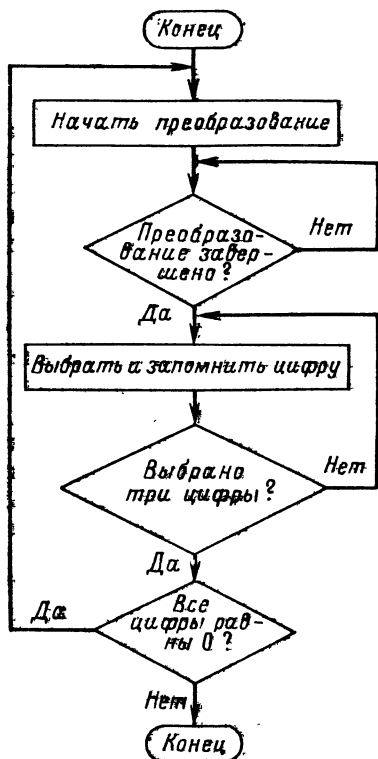


Рис. 6.5. Блок-схема алгоритма ввода данных, поступающих от АЦП

Такой нисходящий процесс детализации будет продолжаться для каждого из блоков обработки, пока степень детализации не станет достаточной для того, чтобы каждую задачу можно было легко закодировать.

Структурное программирование

В структурном программировании используются только простые логические структуры. Якопини¹ доказал, что любая программа может быть написана с использованием только трех структур:

- 1) последовательной, в которой команды или программы выполняются последовательно в том порядке, в котором они записаны;
- 2) условной типа IF—THEN—ELSE, т. е. IF A THEN P₁ ELSE P₂, где A — логическое выражение, а P₁ и P₂ — программы, составленные из трех канонических структур. Если выражение A истинно, ЭВМ выполняет программу P₁. Если A ложно, ЭВМ выполняет программу P₂. Программа P₂ может быть опущена если ЭВМ не должна выполнять никаких действий в том случае, когда выражение A ложно.

C. Bohm and G. Jacopini. Flow Diagrams, Turing Machines and Languages with Only Two-Formation Rules, CACM, vol. 9, № 5, May 1966, p. 366—371.

Пример. Используется полная форма условного оператора с THEN и ELSE.

IF $X \neq 0$ THEN $Y = 1/X$ ELSE $Y = 0$.

Эта конструкция обеспечивает предотвращение деления на нуль и определение значения Y в том случае, когда X равен нулю.

Пример. Сокращенная форма оператора без ELSE. IF CENTS ≥ 50 THEN DOLLARS = DOLLARS + 1. С помощью этого оператора осуществляется округление переменной DOLLARS до ближайшего доллара. Если CENTS < 50, никаких действий выполнять не нужно;

3) циклической типа DO — WHILE, т. е. DO P WHILE A (выполнять P, пока A), где A — логическое выражение, P — программа, состоящая только из допустимых структурных элементов. Электронно-вычислительная машина проверяет справедливость утверждения A и выполняет программу P, если A истинно, а затем снова возвращается к проверке A и т. д. Другими словами, ЭВМ выполняет программу P, пока выражение A остается истинным.

Пример.

```
INDEX = 1
DO WHILE INDEX ≤ MAX
  BLK1 (INDEX) = BLK2 (INDEX)
  INDEX = INDEX + 1
END
```

Приведенный оператор пересылает группу элементов, число которых задано переменной MAX, из массива 1 (BLK1) в массив 2 (BLK2).

Программа называется *структурированной*, если она написана с использованием только рассмотренных трех структур или некоторого другого оговоренного набора структур. На рис. 6.6 приведены блок-схемы условного оператора и оператора цикла.

Каждый такой структурный элемент имеет один вход и один выход. Если при выполнении программы P_1 (рис. 6.6, а) возникнет ошибка, будет точно известно, как ЭВМ попала в эту точку. Если бы программа была неструктурированной (рис. 6.7), то можно было бы предположить, что ошибка возникла в любой из ветвей, ведущих в P_2 . Кроме того, лю-

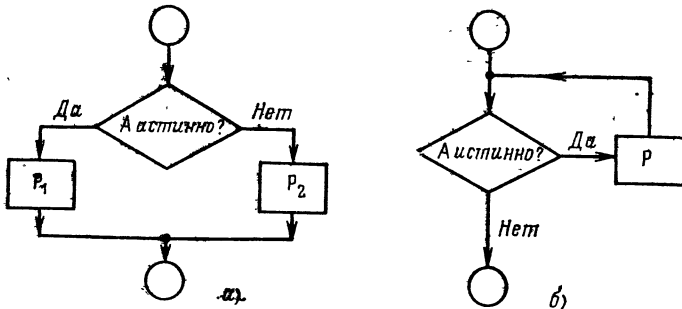


Рис. 6.6. Блок-схемы типовых программных структур:

а — ветвление; б — цикл

бая корректировка может повлиять на выполнение других частей программы. Очевидно, ситуация будет еще более запутанной, если программа имеет к тому же более одного выхода.

Типичный пример тех сложностей, которые возникают при отладке неструктурированных программ, можно проиллюстрировать на следующем фрагменте программы на языке ФОРТРАН

```
40 COUNT = 1
   NUNIT = XLENG/SCALE
```

Предположим, что при вычислении переменной NUNIT ЭВМ пыталась выполнить операцию деления на нуль. Где возникла ошибка? Сначала необходимо обнаружить все операторы, из которых управление передается оператору 40, а это может потребовать просмотра всей программы или таблицы перекрестных ссылок. Затем необходимо выяснить, выполнение какого из операторов привело к ошибке.

Внесение исправлений в фрагмент программы не должно влиять на другие части программы, которые также включают в себя измененный фрагмент. Структурированные программы позволяют упростить работу по тестированию, отладке и сопровождению в процессе разработки ПО. Наибольшие сложности в программах на языке ФОРТРАН возникают в результате использования операторов GO TO и IF, которые настолько усложняют структуру программы, что программисту бывает трудно понять, каким образом программа оказывается в некоторой точке. Многие сторонники структурного программирования твердо убеждены, что для получения структурированных программ надо обязательно отказаться от использования оператора GO TO (безусловный переход). Основной смысл структурного программирования состоит в том, что более простые схемы вычислительного процесса позволяют получить более четкие, более надежные и просто тестируемые программы.

До сих пор результаты использования методов структурного программирования в различных условиях были обнадеживающими. В настоящее время большинство крупных программных проектов реализуется с использованием подобных методов, имеющих иногда некоторые особенности. Сообщалось о том, что производительность труда программистов удавалось повысить от 50 до 100%, хотя в большинстве случаев для повышения производительности труда использовалось сразу несколько методов и отсутствовали измерительные эксперименты с хорошо продуманным контролем.

Очень важным является вопрос о возможности применения подобных методов при

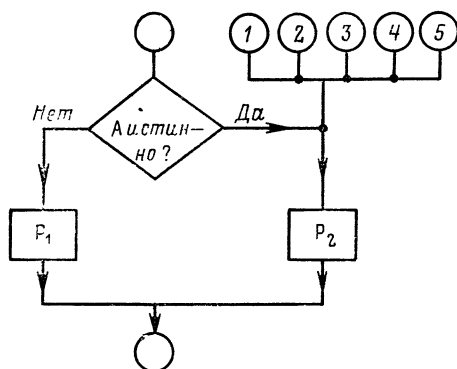


Рис 6.7. Неструктурированная программа

программировании для микропроцессоров. Первоначально структурное программирование использовалось при реализации очень крупных программных проектов, в работе над которыми было занято много групп программистов и которые требовали составления десятков тысяч команд. Микропроцессорные системы редко имеют подобные масштабы. Методы структурного программирования проще всего использовать при составлении программ на языке высокого уровня, в то время как большинство программ для микропроцессоров пишется на языке ассемблера или даже на машинном языке. Дают ли методы структурного программирования какие-то выгоды пользователям микропроцессоров?

По-видимому, на этот вопрос следует ответить утвердительно. Хотя по современным меркам программы для микропроцессоров редко бывают длинными, их трудно отлаживать и тестировать из-за примитивности имеющихся аппаратных и программных средств. Следует тщательно изучить методы, позволяющие упростить процесс отладки и тестирования. Кроме того, применительно к микропроцессорам создается мало программ для одноразового использования. Большая часть этих программ является составной частью систем, и они подвергаются тестированию, документированию и расширению так же, как и аппаратные компоненты системы. Структурное программирование позволяет описывать ПО системы столь же четко, как и ее аппаратные средства.

Сложность состоит в том, что мало кто из проектировщиков систем использует языки, допускающие возможность структурного программирования. Языки высокого уровня, основанные на ПЛ/1 (например, ПЛ/М фирмы Intel и МПЛ фирмы Motorola), имеют необходимые средства структурирования и могут использоваться для составления структурированных программ. В статье Поппера¹ было сообщение о появлении структурного макроассемблера. Однако сейчас большинство программ пишется с помощью обычных ассемблеров, подобных ассемблерам, описанным в гл. 4. Таким образом, структурное программирование следует использовать на этапе проектирования программы, а затем преобразовывать структурированную программу в ассемблерную примерно так же, как это делается при ручном преобразовании ассемблерной программы в машинный код. Поскольку кодирование обычно занимает незначительную долю времени, затрачиваемого на программирование, введение еще одного этапа проектирования может оказаться полезным. Структурированные программы часто оказываются более медленными и требуют больше памяти, чем неструктурированные. Однако уже отмечалось, что время работы программы и затраты памяти не так существенны при разработке микропроцессорных систем, как время, затрачиваемое на создание программ. Использование методов структурного программирования может существенно уменьшить общие сроки разработки программы.

¹C. Popper. SMALL — A Structured Macroassembly Language for a Micro computer, Proceedings of COMPCON, 1974, p. 147.

```

; Установить указатель на первую позицию строки и прочитать
; первый символ с клавиатуры
CHARACTER POINTER = 1
READ INPUT
;
; Анализировать символы до обнаружения символа «возврат ка-
; ретки»
DO WHILE INPUT # CARRIAGE RETURN;
;
; Если входной символ — пробел и не достигнут конец строки,
; увеличить указатель на единицу
IF INPUT = SPACE THEN
    IF CHARACTER POINTER ≠ 80 THEN
        CHARACTER POINTER = CHARACTER POINTER + 1
; Если входной символ — знак возврата и он не находится в начале
; строки, увеличить указатель на единицу
ELSE IF INPUT = BACKSPACE THEN
    IF CHARACTER POINTER ≠ 1 THEN
        CHARACTER POINTER = CHARACTER POINTER + 1
;
; Стереть символ, заменив его пробелом
ELSE IF INPUT = DELETE THEN
    CHARACTER (CHARACTER POINTER) = SPACE
;
; Заменить на введенный символ
ELSE
    CHARACTER (CHARACTER POINTER) = INPUT
    IF CHARACTER POINTER ≠ 80 THEN
        CHARACTER POINTER = CHARACTER POINTER + 1
;
; Прочитать следующий входной символ
; READ INPUT
END

```

Рис. 6.8. Структурированный вариант редактора

Типичным примером применения методов структурного программирования является разработка простой программы-редактора. Программа позволяет осуществлять перемещение влево и вправо по строке (на экране дисплея или телетайпа), стирание или замену символов и завершает процесс редактирования при нажатии на клавишу «возврат каретки» (BK). На рис. 6.8 показана структурированная программа, выполняющая функции редактора (в этом примере комментарии помечаются точкой с запятой, а для выделения начала и конца струк-

турных компонентов используются абзацные отступы). В приведенном примере структурированного редактора отсутствуют операторы безусловного перехода (операторы GOTO). Главный цикл, выполняемый до обнаружения символа ВК (тело цикла не будет выполнено ни разу, если первым встреченным символом будем ВК), реализован с помощью структурного оператора DO-WHILE. Внутри цикла имеются конструкции IF-THEN-ELSE.

Для выделения этих структур на различных уровнях сделаны соответствующие абзацные отступы. Если не предусмотреть такого выделения, возникают большие затруднения при определении того, какие ELSE каким THEN соответствуют и какие ELSE опущены. Некоторые авторы для обозначения конца программного блока, включенного в оператор IF, используют необязательный ELSE, ENDIF или FI (слово IF в обратном порядке). Заметим, что один и тот же структурированный вариант проекта программы может использоваться для того, чтобы переписать программу на другом языке или для другой ЭВМ. Структурное программирование требует соблюдения программистом определенной дисциплины составления программ, но существенно сокращает время, необходимое для разработки.

Модульное программирование

Модульное программирование представляет собой метод разработки программ, при котором большие программы пишутся, тестируются и отлаживаются небольшими частями, которые затем объединяются в единый комплекс. Настоящий нисходящий метод проектирования программ требует использования модульного программирования, но сам метод модульного программирования появился значительно раньше и часто используется независимо от остальных методов. Обычно модули выделяются на функциональной основе. В программировании для микропроцессоров такое выделение модулей особенно конструктивно, так как библиотеку модулей можно использовать в последующей работе. Парнас¹ предложил иной способ деления на модули, основанный на выделении тех компонентов проекта, которые могут изменяться в процессе разработки проекта. Очевидными преимуществами модульного программирования являются уменьшение размеров отлаживаемых и тестируемых программ, создание типовых программ, которые могут повторно использоваться, и возможность деления задач на части или использование ранее созданных программ. К числу недостатков следует отнести сложности стыковки программ, дополнительные затраты времени и памяти на передачу управления между модулями и необходимость проводить тестирование на разных уровнях (на уровне модулей и программы в целом), а также необходимость составления специальных программ для тестирования модулей. Труднее всего воспользоваться модульным программированием тогда, когда наиболее критичной является структура данных, а не структура про-

¹D. L. Parnas. On the Criteria to Be Used in Decomposing Systems into Modules, CACM, vol. 15, № 12, December 1972, p. 1053—1058.

граммы. Однако модульное программирование — это очень эффективный метод разработки ПО для микропроцессоров независимо от использования других методов. В рассмотренной ранее программе для системы взвешивания можно выделить следующие программы-модули:

1) сообщающая АЦП, что он должен начать работать, ожидающая завершения процесса преобразования и затем помещающая результат преобразования в память или регистр. Такую программу (модуль) можно, разумеется, использовать в любой системе с таким же или аналогичным АЦП;

2) считывающая информацию с клавиатуры и регистрирующая факт замыкания контактов. Эту программу можно использовать при работе с аналогичным клавишным устройством;

3) перемножающая два десятичных числа. Эта программа может использоваться в любой системе для выполнения десятичных арифметических операций;

4) выводящая результаты на сегментные дисплеи. Эту программу можно написать таким образом, что можно будет легко изменять постоянные времени дисплеев.

Разумеется, разработчик ПО должен использовать все методы проектирования комплексно. Разработка блок-схем, нисходящее проектирование, структурное программирование и модульное программирование не исключают друг друга. Проектировщик должен помнить, что основная его задача состоит в том, чтобы создать работающую программу, а не слепо следовать предписаниям какого-либо из методов.

6.4. КОДИРОВАНИЕ

Процесс кодирования программ на языке ассемблера подробно рассмотрен в гл. 5. В данном параграфе излагаются некоторые соображения по поводу стиля программирования, которые могут быть полезны независимо от используемого языка или типа микропроцессора. Многие из изложенных рекомендаций заимствованы из работы Кернигана и Плуджера¹.

1. Используйте вместо конкретных значений адресов памяти, констант, масок, номеров устройств ввода-вывода или числовых величин символические имена.

Использование символических имен позволяет не только пояснить смысл и назначение конкретного адреса или элемента данных, но также облегчает внесение в программу изменений. Например, если главным системным устройством печати является выходное устройство 2, в программе будет часто встречаться команда

OUT 2.

Если при изменении конфигурации аппаратурных средств потребуется использовать устройство с другим номером, программист должен найти в программе все ссылающиеся на него команды и изменить

¹B. M. Kernighan and P. J. Plauger. The Element of Programming Style, McGraw-Hill, New York, 1974.

Программа 1 (выбранные обозначения произвольны)			Программа 2 (обозначения несут содержательную нагрузку)		
Z:	LXT	H, X	NEWMX:	LXT	H, BLOCK
W:	MOV	A, M		MOV	A, M
	INX	H	NEXTE:	INX	H
	DCR	B		DCR	B
	JZ	Y		JZ	DONE
	CMP	M		CMP	M
	JC	Z		JC	NEWMX
Y:	JMP	W		JMP	NEXTE
	STA	V	DONE:	STA	MAX
	HLT			HLT	

Рис. 6.9. Поиск максимального значения

их. Однако, если программист использовал символическое имя, достаточно изменить тот оператор, в котором этому имени присваивается конкретное значение. Другими словами, если номер устройства был ранее задан с помощью операторов

```
PRNTR EQU 2
OUT PRNTR
```

то при изменении номера устройства на #3 достаточно изменить один оператор

```
PRNTR EQU 3
```

Использование символических имен позволит избежать путаницы адресов и данных. Например, операторы

```
P1TOP EQU 255
MIN1 EQU 0FFH
```

присваивают различные символические имена адресу 255 (P1TOP) и числу 255 (MIN1) : FF — дополнительный код для числа 1.

2. Используйте содержательные имена и метки. Осмысленные имена и метки помогут документировать и сопровождать программу. Они также удобны при отладке и включении программы в систему. На рис. 6.9 приведены два варианта одной и той же программы для МП Intel 8080: с произвольными и содержательными обозначениями меток и адресов. Дополнительные затраты на подготовку программы окупаются внесением ясности выполняемых операций даже при отсутствии документации. Имена должны быть как можно более простыми и очевидными, например: PRNTR — для системного принтера; TTY — для телетайпа; MAX — для максимального значения; START — для начала программы. Использование при кодировании содержательных символических имен позволит программисту сэкономить время для других этапов разработки ПО.

```

; Ячейки ОЗУ
BUFR EQU 237 ; Входной буфер телетайпа
CLOCK EQU 300 ; Число тактовых сигналов
RFLAG EQU 301 ; Признак готовности телетайпа
; Таблицы
SQR EQU 606 ; Таблица квадратов
SSEG EQU 520 ; Таблица семисегментных кодов
; Числовые величины
MONE EQU -1
; Устройства ввода-вывода
ADCON EQU 4 ; Входное устройство АЦП
KBD EQU 3 ; Входное устройство «клавиатура»
PTAPE EQU 2 ; Входное устройство для ввода с перфо-
; ленты
TTY EQU 1 ; Входное устройство «телетайп»

```

Рис. 6.10. Пример определения символических имен в программе

3. Располагайте все операторы определения данных в начале программы. В этом случае их легко найти, проверить и при необходимости изменить. На рис. 6.10 приведен пример, в котором каждая определяемая величина сопровождается комментарием. Очевидно, что определения, разбросанные по всей программе, существенно затрудняют отладку и документирование программы.

4. Выбирайте метки и имена так, чтобы они легко различались. Это уменьшит вероятность возникновения ошибки. Использование цифр 0, 1 и 2, а также букв O, I и Z должно быть сведено к минимуму. Конечно, не следует использовать в одной программе в качестве идентификаторов имена MINI и MIN1. В этом случае вероятность ошибки очень велика. Те имена, которые постоянно приводят к путанице, должны быть изменены. В программе и без того будет достаточно ошибок, потому не создавайте лишних.

5. Избегайте использования неясных адресных выражений. Примерами могут служить указанные значения смещения по отношению к текущему значению счетчика команд (PC) в ЭВМ с командами переменной длины, использование сложных выражений, значения которых зависят от порядка выполнения операций или использования редких операций, а также средств условного ассемблера. Что именно делает программа, показанная на рис. 6.11? Если даже в момент составления программы программист ясно понимает смысл используемых выражений, то через несколько дней или месяцев они будут ему совершенно непонятны.

```

COND EQU (105 SHR 2) AND 128 | MVI REG - 1, TOP OR 8
IF COND | ADD B
JNC $ + 5 | MOV COND + 3, A
LDA STRT | ENDIF

```

Рис. 6.11. Пример неудачно написанной программы

6. По мере возможности избегайте команд безусловного перехода. Программу с большим числом переходов трудно воспринимать и отлаживать. За счет повторения команды часто можно добиться большей ясности и большего быстрогодействия программы, чем при использовании передачи управления. Программы, в которых имеется много команд перехода, должны быть переписаны перед этапом отладки.

7. Делайте модули короткими. Расчлняйте длинные модули на секции. Короткий модуль не только проще отлаживать и корректировать. Такой модуль с большой вероятностью можно будет использовать при разработке других программ, поскольку выполняемая им операция будет простой и поэтому часто встречающейся. Максимальный размер модуля не должен превышать одной или двух страниц машинного кода (от 50 до 75 строк). Стандартные модули могут быть оформлены как макрокоманды или просто вставляться (в программу) как фрагменты, чтобы избежать большого числа команд вызова подпрограмм и возврата (в микропроцессорах операции вызова подпрограмм не только вызывают дополнительные затраты памяти и времени, но также могут привести к переполнению стека адресов возврата).

8. Делайте модули как можно более обобщенными. Очевидно, что очень специализированный модуль (например, сортирующий 32 элемента или отыскивающий букву R) вряд ли будет часто использоваться. Нередко большая общность выполняемых функций может быть достигнута за счет небольшого увеличения или без увеличения длины машинной программы: программу сортировки для обработки массива произвольной длины написать так же просто, как и программу для сортировки массива фиксированной длины. Универсальность, требующая существенного усложнения программы (например, единая программа преобразования из кода ASCII и EBCDIC), нецелесообразна. Ранее упоминались следующие простые способы достижения универсальности: использование символических имен вместо задания конкретных адресов и числовых значений, группировка всех определений в начале процедуры (возможно, специальное выделение параметров) и использование содержательных имен, которые поясняют смысл и идентифицируют объекты. С помощью подобных средств можно написать программу, которая будет часто использоваться и может быть легко модифицирована.

9. Обращайте внимание на простоту и ясность программы. Целью программирования для микропроцессоров является создание работающей программы. Чаше всего экономия нескольких ячеек памяти или нескольких микросекунд не имеет смысла; оптимизация машинной программы может быть выполнена впоследствии. Первоначальная программа должна быть скорее простой, чем изощренной. Следует избегать инициализации переменных в операторах DATA, использования команд не по их прямому назначению, длинных команд для обработки явно непомеченных переменных, результатов предыдущей операции для инициализации переменных или выполнения вычислений, параметров в качестве постоянных данных. Приведем несколько

примеров (для Intel 8080), иллюстрирующих приемы, которые следует избегать:

а) LXI B,264 ; Установить B = 1, C = 8

В том случае, когда B и C не взаимосвязаны, не следует использовать этот оператор. Вместо него лучше написать

MVI B,1 ; Установить B = 1

MVI C,8 ; Установить C = 8

б) DCR C
JNZ STRT
MOV B,C

Не следует очищать регистр B таким образом. Лучше написать

DCR C
JNZ STRT
MVI B,0

Дополнительный расход памяти (одна ячейка) невелик и компенсируется большей ясностью выполняемых операций.

В данном параграфе основное внимание было уделено *«защитающему» программированию*, т. е. такому стилю программирования, который упрощает внесение исправлений и гарантирует от неправильного понимания выполняемых программой действий и от возникновения ряда ошибок. Разумеется, такой стиль программирования приводит к увеличению затрат времени на кодирование программы; кроме того, программист никогда не сможет предусмотреть всех проблем, которые могут возникнуть впоследствии. Однако более тщательное программирование окупает себя за счет уменьшения числа ошибок и упрощения последующего использования и сопровождения программ. Конечно, ни один программист не будет буквально следовать всем этим советам, однако разумный подход к кодированию программ может существенно облегчить процесс разработки ПО для микропроцессоров.

6.5. ОТЛАДКА

В данном параграфе рассмотрены некоторые средства, используемые при отладке, а также некоторые распространенные ошибки программирования. В теории программирования отладка и тестирование программ часто называются соответственно верификацией и проверкой работоспособности. Верификация обеспечивает проверку, выполняет ли программа именно то, что предусматривал программист. Проверка работоспособности гарантирует, что программа выдает коррективные результаты на всех наборах исходных данных. Невозможно строго провести границу между отладкой и тестированием программ. Отладка программ для микропроцессоров обычно представляет собой сложную проблему, так как невозможно непосредственно посмотреть содержимое регистров. Имеющиеся средства отладки являются примитивными; функции ПО и аппаратуры тесно связаны между собой, часто работа программ зависит от временных соотношений; трудно получить

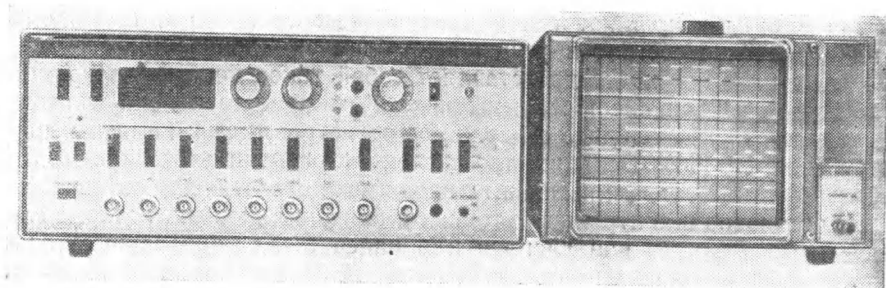


Рис. 6.12. Логический анализатор

требуемые для отладки данные для систем, работающих в реальном масштабе времени.

К числу применяемых средств отладки относятся имитаторы, логические анализаторы, контрольные точки, трассировочные процедуры, дампы памяти, программные прерывания.

Имитатор представляет собой программу, которая имитирует процесс выполнения программ на другой ЭВМ. Она делает то же, что мог бы сделать программист с карандашом и бумагой, регистрируя результат выполнения каждой команды. В отличие от человека имитаторы не устают и имитируют процесс выполнения программ без ошибок. Обычно имитатор работает на большой ЭВМ и моделирует работу малой ЭВМ, у которой не хватает средств для соответствующего тестирования. Как правило, имитаторы, как и кросс-ассемблеры, представляют собой большие программы на языке ФОРТРАН, которыми можно воспользоваться через систему разделения времени. Эти программы эффективны при проверке правильности логики программы, так как программист может изменить исходные данные, посмотреть содержимое регистров и использовать другие средства отладки. Однако имитаторы не могут в полной мере смоделировать ввод-вывод или оказать существенную помощь при решении задач, связанных с временными соотношениями. Иногда имитатор является единственным средством отладки, например в том случае, когда ЭВМ, для которой создается программа, не работает, используется для других целей или еще не создана или не поставлена.

Логический анализатор (рис. 6.12) представляет собой цифровой вариант осциллографа, предназначенного для подключения к общей шине. Анализатор обнаруживает состояния цифровых сигналов во время каждого цикла синхронизации и помещает их в память. Затем он выводит эту информацию на экран электронно-лучевой трубки аналогично тому, как это делает обычный осциллограф. Как и в осциллографе, здесь можно контролировать и выводить одновременно несколько сигналов. Логический анализатор представляет собой удобное средство отображения быстро меняющихся параллельных цифровых сигналов. Некоторые анализаторы имеют возможность запускаться по какой-то команде или группе команд, повторно вызывать данные и фиксировать очень короткие шумовые всплески (glitches). Логические

анализаторы дополняют собой программные имитаторы, и основное их назначение состоит в решении проблем синхронизации.

Контрольная точка представляет собой такое место в программе, в котором выполнение программы может быть приостановлено для того, чтобы проверить текущее содержимое регистров, ячеек памяти и портов ввода-вывода. Большинство систем разработки для микро-ЭВМ и большинство программных имитаторов имеют средства для осуществления остановов в контрольных точках и описания данных, которые следует вывести на экран дисплея или напечатать. Контрольные точки часто можно создать с помощью команды TRAP или команды условного перехода, реагирующей на внешний входной сигнал, который может контролироваться с пульта. Например, команда

ПЕРЕХОД ПО НЕ ТЕСТ \$

будет обеспечивать передачу управления на себя (\$ — текущее значение счетчика команд) до тех пор, пока значение входного сигнала ТЕСТ не станет равным 1. При этом можно будет проверить содержимое регистров и ячеек памяти. В некоторых микропроцессорах (например, в Intel 4040) при останове на шины подается информация о состоянии (эта информация может включать в себя текущее содержимое аккумулятора, счетчика команд или других наиболее важных регистров).

Трассировочные процедуры представляют собой программу, которая через заданные промежутки времени печатает информацию о состоянии процессора. Большая часть программных имитаторов и систем разработки имеет средства трассировки. Иногда трассировщик печатает содержимое всех регистров процессора и значения признаков после выполнения каждой команды. Некоторые программные имитаторы позволяют выдавать содержимое указанных регистров или ячеек памяти каждый раз, когда оно меняется. Если программист хорошо не продумает совокупность выдаваемых переменных и форму выдачи, трассировочная процедура выдаст большой по объему листинг, в котором будет трудно разобраться.

Дамп памяти представляет собой распечатку содержимого некоторой области памяти. Большинство программных имитаторов, систем разработки для микро-ЭВМ и мониторов позволяет выдавать дампы памяти. Разумеется, для расшифровки полного дампа памяти требуется много времени и усилий; особенно не рекомендуется пользоваться средством дампирования, если для печати используется телеайп. Хотя дамп памяти в редких случаях оказывается эффективным средством отладки, нередко он является единственно доступным. По традиции полный дамп памяти используется при отладке программ в тех случаях, когда другие методы оказываются бессильными.

Программные прерывания. Для целей отладки часто используется команда TRAP (ПРОГРАММНОЕ ПРЕРЫВАНИЕ). По этой команде обычно сохраняется текущее значение счетчика команд и осуществляется переход в указанную ячейку памяти. Эта ячейка памяти может являться начальной точкой отладочной программы, которая выдает на печать или высвечивает на экране дисплея информацию о состоянии

(таким образом, команда TRAP позволяет реализовать контрольные точки). Например, когда в МП Motorola 6800 выполняется команда ПРОГРАММНОЕ ПРЕРЫВАНИЕ, в стековой памяти автоматически сохраняется содержимое всех регистров; после этого программист может непосредственно получить доступ к их содержимому. В МП Intel 8080 имеется команда программного прерывания (RST или RESTART), но эта команда обеспечивает сохранение в стеке только значения PC. Программист должен включить в свою программу команду TRAP и написать соответствующую отладочную программу, если ее нет в составе стандартного пакета. При этом желательно воспользоваться услугами монитора, который может поместить команду TRAP по заданному адресу.

Список контрольных значений и ручная проверка

Список контрольных значений представляет собой средство отладки, которое можно использовать в сочетании с методами проектирования, использующими блок-схемы или структурное программирование. Программист должен проверить, что каждой переменной было присвоено начальное значение, что каждый оператор блок-схемы или логической структуры был закодирован, что все определения переменных сделаны корректно и что все ветви программы правильно увязаны. Наличие простого списка контрольных значений может позволить сэкономить очень много времени.

Ручная проверка длинной или сложной программы обычно является пустой тратой времени, так как вероятнее всего, что при проверке программы программист совершит больше ошибок, чем при ее кодировании. Однако циклы и отдельные блоки программы должны быть проверены вручную, чтобы убедиться в правильности общей логики ее работы. Циклы программист должен проверить вручную, чтобы убедиться в том, что первый и последний проходы цикла выполняются правильно; обычно именно на этих проходах возникает большинство ошибок, возникающих при организации циклов (примеры их будут приведены далее). Вручную следует также проверить, что программа верно работает в простейших случаях; тривиальные ситуации типа таблиц, не содержащих ни одного элемента, буферов, не содержащих данных, являются источниками нетривиальных ошибок, которые часто удается обнаружить на последнем этапе отладки программ или при полевых испытаниях. Другими источниками ошибок, которые следует проконтролировать вручную, являются проверки на равенство в циклах или условных переходах. Проблема состоит обычно в том, чтобы определить, какое значение признака использовать в качестве условия перехода и какое действие следует выполнить, если некоторая переменная принимает значение, равное пороговому, а не большее или меньшее его. Ручная отладка таких «тонких» программных ситуаций может ценой небольших затрат избавить от возникновения большого числа ошибок.

Программист должен выполнять операции ручного контроля и другие операции отладки систематически. Не следует думать, что

первая обнаруженная в программе ошибка является последней; чтобы добиться максимального эффекта от каждого отладочного прогона программы, следует перед ее последующим выполнением тщательно проверять всю программу целиком.

Типичные ошибки

Каждая программа имеет свои ошибки. Некоторые ошибки являются общими для всех программ. Одни характерны для программирования на ассемблере, другие не зависят от языка. К таким ошибкам относятся:

1. Отсутствие операций присвоения начальных значений переменным, в частности счетчикам или указателям адреса. Не следует думать, что перед началом выполнения программы значения регистров, признаков и ячеек памяти будут равны нулю.

2. Инкрементирование счетчиков и указателей до того, как они были использованы, или сохранение их значений неизменными. Счетчики и указатели должны быть изменены независимо от того, какая ветвь программы выполнялась в цикле.

3. Отсутствие в программе блоков обработки таких тривиальных ситуаций, как массивы или таблицы, не содержащие ни одного элемента или содержащие единственный элемент.

4. Инвертирование логических условий, например переход по нулю вместо перехода по не нулю. Не забывайте, что признак НУЛЬ равен 1 при нулевом результате и 0 в противном случае.

5. Неверный порядок операндов, например пересылка из А в В, хотя имелась в виду пересылка из В в А. В Intel 8080 по команде `MOV A,B` содержимое В посылается в А.

6. Передача управления по условиям, которые не сохранились неизменными после того, как им были присвоены требуемые значения. При программировании на ассемблере это соответствует ситуации, при которой вы пытаетесь использовать значения признаков для выполнения условного перехода, хотя они изменились в результате выполнения промежуточных команд. Программист должен хорошо представлять себе, как влияют на значения признаков выполняемые команды.

7. Отсутствие в логике программы блоков обработки таких необычных ситуаций, как отсутствие в таблице искомого элемента или проверка логического условия, которое никогда не выполняется. Подобные ошибки обычно приводят к заикливанию.

8. Отсутствие сохранности прежнего содержимого аккумулятора или иного регистра перед его новым использованием.

9. Смещение понятия адреса и его содержимого. Ячейка памяти с адресом 40 не обязательно содержит число 40. Следует особо отметить различие между непосредственной адресацией, при которой данные являются частью команды, и прямой адресацией, при которой в команде указывается адрес данных.

10. Неверный переход в случае «равно». Например, переход по условию «число неотрицательно» вместо условия «число положительно».

11. Обмен содержимого регистров без использования рабочих ячеек. Так, в результате операций

$$\begin{aligned}A &= B \\ B &= A\end{aligned}$$

содержимое A и B станет равным содержимому B, так как первое присвоение стирает прежнее значение A. Правильный результат дает следующая последовательность операций

$$\begin{aligned}T &= A \\ A &= B \\ B &= T\end{aligned}$$

12. Непонимание различия между числовыми и символьными данными. Изображения нуля в кодах ASCII или EBCDIC не совпадают с числом ноль. Большинство периферийных устройств используют для представления данных свои коды.

13. Непонимание различия между числами и кодированными представлениями чисел. Так, двоично-десятичный код 61 не совпадает с двоичным представлением числа 61.

14. Неправильное задание длины таблицы или блока. В ячейках 40—48 содержатся девять информационных слов вместо восьми.

15. Неправильное указание порядка операндов в некоммутативных операциях. Так, ассемблерная команда SUB C вычитает содержимое регистра C из содержимого аккумулятора, а не наоборот. Программист должен быть особенно внимательным при использовании операций вычитания, сравнения и деления.

16. Непонимание особенностей представления чисел в дополнительном коде. Почти во всех ЭВМ для представления чисел используется дополнительный код, у которого старший разряд числа является знаковым, но при этом остальные разряды не несут информации о знаке числа.

17. Игнорирование возможности возникновения переполнения при использовании операций над числами со знаком. Сложение чисел 64 и 64 в 8-битном процессоре дает —128 (проверьте сами!), а сложение —128 и —128 дает 0.

18. Игнорирование влияния подпрограмм и макрокоманд. Выполнение подпрограмм и макрокоманд почти всегда вызывает изменение значений признаков и может также менять содержимое ячеек памяти и регистров. Программист может быть введен в заблуждение тем, что вызов подпрограммы или макрокоманды выглядит как одна команда.

Существует много других ошибок, однако приведенный перечень дает программисту представление о том, на что следует обратить внимание. Типичным примером решения задач отладки является следующая попытка написать на языке ассемблера Intel 8080 программу определения длины строки символов в коде ASCII, которая заканчивается символом «возврат каретки» (BK) (OD₁₆ в коде ASCII). На рис. 6.13 показана первоначальная версия программы.

MBUF	EQU	40H	
CR	EQU	00001101B	
	LXI	H, MBUF	; Адресный указатель = начало
	INX	H	; строки
CHKCR:	INR	B	; Счетчик символов = счетчик символов
			; +1
	CMP	M	; Счетчик символов = 0
	JNZ	CHKCR	; Следующий символ есть ВК?
	HLT		

Рис. 6.13. Первоначальный вариант программы определения длины строки

Пример. Отладка программы, которая определяет длину строки.

Была выполнена простая процедура ручной отладки для случая, когда строка символов состояла из одного символа ВК, расположенного в ячейке MBUF. Прежде чем проанализировать данные, в начальном адресе по команде INX H к указателю была добавлена единица.

В результате символ ВК теперь уже не будет обнаружен. Кроме того, содержимым аккумулятора и регистра В, используемым в цикле, не были заданы начальные значения. После исправления этих ошибок программа стала выглядеть так, как показано на рис. 6.14.

Ручная проверка этой программы показала, что вместо команды JNZ DONE следует поставить команду JZ DONE, если процесс поиска следует закончить в тот момент, когда анализируемый символ есть ВК. В простейшем случае, как и ожидается, результатом будет нулевая длина, указанная в регистре В.

Однако попытка применить программу к анализу строки, состоящей из символа «пробел», за которым следует символ ВК, также даст результат 0; следовательно, программа неверна. Ручная проверка выполнения цикла показала, что команда JNR В увеличивает содержимое счетчика, но затем следует передача управления на команду MVI B,0, которая вновь обнуляет этот счетчик. Метка CHKCR стояла в неверном месте. Исправленная программа показана на рис. 6.15.

MBUF	EQU	40H	
CR	EQU	00001101B	
	LXI	H, MBUF	; Адресный указатель = начало строки
	MVI	A, CR	; Загрузить символ ВК для выполнения сравнений
CHKCR:	MVI	B, 0	; Счетчик символов = 0
	CMP	M	; Следующий символ есть ВК?
	JNZ	DONE	; Да, идти к DONE
	INX	H	
	INR	B	; Нет, увеличить содержимое счетчика
			; символов на 1
	JMP	CHKCR	
DONE:	HLT		

Рис. 6.14. Второй вариант программы определения длины строки

MBUF	EQU	40H	
CR	EQU	00001101B	
	LXI	H, MBUF	; Адресный указатель = начало строки
	MVI	A, CR	; Загрузить символ ВК для выполнения сравнений
	MVI	B, 0	; Счетчик символов = 0
CHKCR:	CMP	M	; Следующий символ есть ВК?
	JZ	DONE	; Да, идти к DONE
	INX	H	
	INR	B	; Нет, увеличить содержимое счетчика символов на 1
	JMP	CHKCR	
DONE:	HLT		

Рис. 6.15. Окончательный вариант программы определения длины строки

Эта программа дала верный результат при анализе нескольких контрольных строк символов. Приведенный пример показывает, что простая процедура ручной отладки при анализе логики программы может оказаться не менее полезной, чем самые изощренные отладочные средства. Однако отладка систем реального времени гораздо сложнее, требует более серьезной подготовки и дополнительного оборудования.

6.6. ТЕСТИРОВАНИЕ

Тестирование и отладка программ тесно взаимосвязаны. Тестирование — это, по существу, следующий этап отладки, на котором правильность работы программы проверяется на подходящих тестовых наборах. Некоторые из тестовых наборов, разумеется, будут совпадать с наборами, использованными в процессе отладки (например, все нули); кроме того, следует осуществить проверку на различных специальных наборах.

Вместе с тем тестирование программы — это не простое повторение выполнения программы на нескольких наборах исходных данных. Исчерпывающая проверка всех возможных ситуаций является наилучшим решением, но такая проверка практически никогда не осуществима. Для простой программы, которая обрабатывает 16-битные наборы исходных данных и дает 16-битный результат, имеется 4 млрд. возможных комбинаций входов и выходов. Существуют некоторые формальные методы верификации, но они применимы только к очень простым программам. Итак, для тестирования программы следует выбрать тестовые наборы. Ситуация осложняется еще и тем, что работа многих программ для микро-ЭВМ зависит от входов реального времени, которые сложно контролировать или имитировать. Нередко микропроцессор должен очень точно взаимодействовать с большой и сложной системой. Каким же образом можно сгенерировать и подать в микро-ЭВМ необходимые данные?

Имеется несколько средств, которые помогают при решении этой задачи. Разумеется, для этой цели могут быть использованы такие уже упоминавшиеся средства отладки, как логические анализаторы и программные имитаторы. В числе других средств отладки можно отметить следующие:

1. *Имитаторы входных и выходных сигналов*, позволяющие с помощью одного входного и одного выходного устройства моделировать работу различных устройств. Подобная имитация позволяет также формировать внешние синхронизирующие сигналы и другие управляющие воздействия. В большинстве систем разработки имеются средства имитации ввода-вывода; большинство программных имитаторов также позволяет моделировать ввод-вывод, но не в реальном времени.

2. *Внутрисхемные эмуляторы* (рис. 6.16), позволяющие подключать для тестирования прототипы системы непосредственно к системе разработки или пульту управления.

3. *Имитаторы ПЗУ*, позволяющие использовать для хранения программ оперативную память, обеспечивая временные характеристики тех ПЗУ, которые будут использоваться в окончательном варианте системы.

4. Операционные системы или мониторы реального времени, которые могут управлять событиями в реальном времени, формировать



Рис. 6.16. Внутрисхемный эмулятор

прерывание и реализовывать в реальном времени трассировку программ и контрольные точки.

5. Эмуляторы на микропрограммных ЭВМ, которые могут выполнять команды из системы команд микро-ЭВМ со скоростями, близкими к реальным, и имеют программируемые средства ввода-вывода.

6. Специальные интерфейсы, позволяющие мини-ЭВМ и программируемому контроллеру протестировать программу для микропроцессора с помощью внешнего блока, управляющего вводом-выводом.

7. Тестирующие программы, автоматически проверяющие работу каждой ветви программы. Подобные программы могут отыскивать логические ошибки, но не помогают в решении проблем, связанных с синхронизацией.

Тестирование микропроцессорного ПО является сложной и слабо исследованной задачей. В настоящее время тестирование чаще всего выполняется с помощью специального оборудования, ориентированного на конкретную задачу; имеется мало соответствующих аппаратных и программных средств общего назначения. Среди правил, помогающих осуществить тестирование программ, можно упомянуть следующие:

1. Включайте план тестирования в проект программы. Тестирование должно быть одним из факторов, учитываемых на этапах постановки задачи, проектирования программы и кодирования.

2. Проверяйте все тривиальные случаи и особые ситуации. Такими ситуациями являются нулевые входы, отсутствие данных на линиях связи, специальные входные сигналы типа предупреждения или тревоги и другие ситуации, которые по каким-либо причинам выпадают из общей схемы. Часто именно простейшие случаи приводят к самым неприятным и непредвиденным ошибкам.

3. Выбирайте тестовые данные случайным образом. Такой способ позволит исключить любую закономерность, которая возникла бы при выборе тестовых данных человеком. Таблицы случайных чисел широко распространены, а в большинстве ЭВМ имеются датчики случайных чисел.

4. Планируйте и документируйте процессы тестирования ПО так же, как и тестирование аппаратуры. Разумеется, с помощью тестирования никогда нельзя окончательно доказать, что ошибки отсутствуют. Поэтому существенной частью процесса тестирования является хорошая проектная проработка как ПО, так и аппаратных средств.

5. Используйте в качестве тестовых значений максимальные и минимальные значения исходных данных. Экстремальные значения нередко являются источниками особых ошибок.

6. Используйте для планирования и оценки сложных тестов статистические методы. Для выбора исходных данных и оценки значимости результатов имеются специальные методы. Методы оптимизации могут обеспечить выбор хороших системных параметров и эффективных наборов тестовых данных.

6.7. ДОКУМЕНТИРОВАНИЕ

Документирование представляет собой этап разработки ПО, которым часто пренебрегают. Вместе с тем правильно составленная документация полезна не только на этапах отладки и тестирования, но исключительно важна на этапах сопровождения и внесения изменения. При необходимости хорошо документированной программой легко воспользоваться снова. Работа с недокументированной программой обычно требует столько дополнительных усилий, что программисту может оказаться легче написать программу заново.

Для документирования широко используются такие средства, как блок-схема, комментарии, карты памяти, списки параметров и спецификаций и библиотеки программ. При структурном программировании используются специальные формы документирования, которые изложены в работах, перечисленных в списке литературы.

Наиболее наглядным средством документирования являются блок-схемы. Обобщенная блок-схема аналогична той, которая была показана ранее, может служить в качестве сжатого графического описания программы. Подобная блок-схема является полезной составной частью документации, но не исчерпывает ее. Более подробная блок-схема программиста (упоминавшаяся ранее) может быть исключительно полезной другому программисту, который должен использовать или сопровождать программу. Обычно такую блок-схему программист создает после завершения разработки программы.

Важной составляющей программной документации являются комментарии. Однако они не исключают необходимости использования правильного стиля программирования. Программа с четкой структурой и удачно выбранными именами частично является самодокументируемой. Комментарии должны пояснять смысл команд, а не просто повторять их значение. На рис. 6.17 и 6.18 приведена одна и та же программа на языке ассемблера Intel 8080, в одном из вариантов которой комментарии являются плохими, а в другом составлены удачно (программа отыскивает максимальный элемент массива, который расположен начиная с ячейки BLKST и длина которого задана в ячейке LENG).

	LDA	LENG	; LENG в A
	MOV	B, A	; A в B
	LXI	H, BLKST	; Поставить BLKST в H
NEWMX:	MOV	A, M	; M в A
NEXTE:	INX	H	; H = H + 1
	DCR	B	; B = B - 1
	JZ	DONE	; Если 0, идти к DONE
	CMP	M	; сравнить с M
	JC	NEWMX	; Если ПЕРЕНОС=1, идти к NEWMX
	JMP	NEXTE	; Идти к NEXTE
DONE:	STA	MAX	; MAX = A
	HLT		

Рис. 6.17. Плохо документированная программа

	LDA	LENG	; Счетчик = длина массива
	MOV	B, A	
	LXI	H, BLKST	; Начало массива данных
NEWMX:	MOV	A, M	; Элемент является новым
			; максимальным
NEXTE:	INX	H	
	DCR	B	
	JZ	DONE	
	CMP	M	; Максимальный элемент больше текущего?
	JC	NEWMX	; Нет, заменить максимальный
	JMP	NEXTE	; Да, перейти к следующему элементу
DONE:	STA	MAX	; Запомнить максимум
	HLT		

Рис. 6.18. Хорошо документированная программа

При составлении комментариев необходимо соблюдать следующие правила:

1. Комментарии должны пояснять действие, выполняемое командой или группой команд, а не просто объяснять смысл кодов операций.

2. Комментарии должны быть возможно более ясными. Следует избегать всяких сокращений и неясных аббревиатур, хотя и не обязательно писать развернутые фразы.

3. Комментировать следует основные действия. Программа с пространными комментариями трудно воспринимается. Стандартные конструкции (управление циклом или операции индексации) нужно пояснить только в тех случаях, когда они выполняются необычным образом.

4. Комментарии следует помещать достаточно близко к соответствующим операторам. Программист должен выработать стандартную форму задания комментариев: повторяемость не является большим недостатком в комментариях. Вместе с тем всякие вариации сбивают с толку.

5. Комментарии должны соответствовать последней версии программы. Наличие комментариев, относящихся к предыдущей версии программы, хуже, чем их отсутствие.

6. Комментарии должны быть четкими. Более подробные объяснения должны быть перенесены в справочное руководство.

Программист должен научиться правильно использовать комментарии. Он должен спросить себя, какие объяснения понадобились бы ему для понимания данной программы, и после этого составить соответствующие комментарии. Подобные систематизированные комментарии могут оказаться полезными на всех этапах разработки ПО.

Карты памяти представляют собой сведения о распределении памяти в каждой программе. Использование таких карт предохраняет от взаимного наложения различных процедур, облегчает передачу параметров и помогает определить требуемый объем памяти и место расположения подпрограмм, таблиц и рабочих областей. Карты памяти осо-

Назначение: программа COMPD вычисляет дополнительный код 16-битного числа

Язык программирования: ассемблер Intel 8080

Исходные данные: 16-битное число в регистровой паре H и L

Результаты: дополнительный код 16-битного числа

Требуемые ресурсы: память емкостью восемь ячеек

время 43 тактовых периода

регистры A, H, L

Типичный пример:

Начало:

(H) = 20 (16-ричное)

(L) = A4 (16-ричное)

Конец:

(H) = DF (16-ричное)

(L) = 5C (16-ричное)

Листинг:

```

;
; Дополнительный код 16-битного числа
;
COMP D: MOV A, L   ; Обратный код младших 8 бит.
        CMA
        MOV L, A
        MOV A, H   ; Обратный код старших 8 бит
        CMA
        MOV H, A
        INX H      ; Прибавить 1 для получе-
                   ; ния дополнительного кода
                   ; из обратного
        RET
```

Рис. 6.19. Библиотечная программа

бенно важны при разработке ПО для микропроцессоров, поскольку в таких разработках память программ и данных (ПЗУ и ОЗУ) используется отдельно, а выделение адресов осуществляется на этапе разработки аппаратной части проекта. Кроме того, необходимо сохранить распределение памяти (особенно более дорогого ПЗУ) и точно знать размещение в памяти некоторых параметров, которые потребуется изменить на месте в соответствии с особенностями конкретного применения.

Список параметров и спецификаций является составной частью любой программы для ЭВМ. Такие списки не только обеспечивают объяснение смысла каждого из параметров и значений различных операций, но и содержат спецификации типов. Наличие подобных списков не исключает необходимости объяснения параметров в комментариях к самим программам.

Карточки библиотечных программ могут содержать описания подпрограмм, которые могут использоваться во вновь разрабатываемых программах. В таких карточках программист должен дать назначе-

ние программы, форматы входных и выходных данных, требования программы к памяти, времени и РОН, описание параметров и контрольный пример, иллюстрирующий работу программы. На рис. 6.19 приведен пример типичной библиотечной программы, которая может быть реализована как подпрограмма или макрокоманда.

Правильно разработанная документация ПО использует все или большую часть изложенных рекомендаций. Полная документация включает в себя: обобщенные блок-схемы, блок-схемы программиста, описание плана тестирования и результатов тестирования программы, текстуальное описание программы, документированный листинг каждого программного модуля, список параметров и спецификаций, карту памяти.

Разработка документации требует много времени, поэтому программист должен выполнять эту работу параллельно с проектированием, кодированием, отладкой и тестированием ПО. Рассмотренные в настоящей главе методы проектирования программы позволяют облегчить их документирование. В свою очередь, хорошая документация упрощает сопровождение и перепроектирование и существенно облегчает разработку программ, которые придется составлять в будущем.

6.8. ПЕРЕПРОЕКТИРОВАНИЕ

Перепроектирование ПО может заключаться в добавлении новых возможностей или удовлетворении новых требований. Перепроектирование должно выполняться поэтапно в соответствии с последовательностью, изложенной применительно к разработке нового ПО. Процесс перепроектирования может заключаться в таком изменении программы, чтобы она удовлетворяла заданным временным требованиям и ограничениям по памяти. В данном параграфе кратко описаны методы, позволяющие сделать программу быстрее и короче.

В тех случаях, когда следует добиться сравнительно небольшого (25% или менее) повышения быстродействия или небольшой экономии памяти, этого можно достичь за счет изменения структуры программы. Возможно некоторое отступление от структурности программы. При этом новая программа должна быть тщательно документирована, так как ее назначение становится менее очевидным. Подобная реорганизация может потребовать больших затрат времени программиста, и ее следует по возможности избегать.

Ниже перечислены некоторые рекомендации, позволяющие повысить скорость работы программы.

1. Обратите внимание на часто выполняемые циклы. Эти циклы можно выявить методами ручного анализа или программного тестирования. Команды, выполняемые однократно или малое число раз, не оказывают существенного влияния на время работы программы. Замена подобных команд может привести к неоправданным ошибкам.

2. Где это возможно, используйте регистровые операции. Подобные операции будут выполняться быстрее других, но при этом возникнут дополнительные операции инициализации и манипулирования. Рекомендуется использовать команды перехода с косвенной адресацией,

которые выполняются путем отправки содержимого регистра в счетчик команд (в Intel 8080 такой командой является команда RCHL).

3. Попробуйте вынести повторно выполняемые операции за пределы цикла. Для этого выявите те операции, которые одинаково выполняются при каждом проходе цикла. Подобные операции можно выполнить за пределами цикла, а результат поместить в регистр или ячейку памяти.

4. По возможности используйте форматы команд с короткими адресами. Использование форматов команд с короткими адресами может потребовать нетривиальных решений в организации данных.

5. Попытайтесь исключить операторы перехода. Для их исполнения почти всегда требуется много времени и памяти.

6. Вместо использования подпрограмм и циклов многократно повторите соответствующие последовательности команд.

7. Используйте преимущества тех адресов, которые могут обрабатываться как 8-битные числа. К подобным адресам относятся адреса на нулевой странице памяти и адреса, кратные числу 100_{16} .

8. Для пересылки данных между памятью и регистрами используйте вместо прямой адресации стековую.

Многие из указанных рекомендаций позволяют одновременно уменьшить время выполнения программы и расход памяти, так как (в большинстве ситуаций) более длинные программы приводят к большему числу обращений к памяти и к большему времени выполнения. Вместе с тем использование подпрограмм представляет пример экономии памяти ценой дополнительных затрат времени на вызов подпрограммы и возврат из нее. Использование циклов является примером аналогичной взаимозависимости между временем и памятью. Если требуется минимизировать время выполнения программы, то обращения к подпрограммам и циклы следует заменить повторным копированием соответствующих участков программ. Для уменьшения затрат памяти можно использовать противоположные приемы.

Не следует ожидать, что процесс оптимизации приведет к очень большой экономии. Если исключить случаи очень плохо написанной программы, обычно достигается увеличение скорости работы или экономия памяти не более чем на 25 %. Если требуется еще больше повысить быстродействие или сэкономить память, можно применить методы, рассмотренные ниже:

1. Разработка нового алгоритма. Другой метод решения может обеспечить большее увеличение быстродействия и более существенное уменьшение требуемой памяти. Модификация использованного ранее метода редко позволяет достигнуть существенного его улучшения.

2. Использование микропрограммирования. Процессор с микропрограммным управлением может оказаться способным выполнить данную программу значительно быстрее.

3. Повышение тактовой частоты. Если имеется более быстрая память, нужно попробовать заставить процессор работать при более высокой тактовой частоте.

4. Использование подключаемых (внешних) аппаратных средств. Внешние аппаратные средства типа устройств умножения,

счетчиков, УАПП и другие могут повысить производительность ЦП, избавив его от выполнения некоторых операций обработки данных.

5. Параллельная обработка. Два или большее число процессоров могут обеспечить выполнение работы с более высокой скоростью, при этом стоимость системы существенно не повышается.

6. Распределенная обработка. Работа может быть выполнена быстрее за счет того, что отдельные функции будут разделены между двумя или более процессорами.

7. Замена программной логики путем использования больших быстродействующих ПЗУ. С помощью ПЗУ большого объема можно реализовать часть программно выполняемых функций. Очевидно, что подобное решение представляет собой компромисс между быстродействием и расходом памяти, типа того, что был отмечен в связи с табличной реализацией вычислений.

В настоящее время микропроцессоры сравнительно редко используются на пределе своих технических возможностей. В тех случаях, когда процессоры работают в напряженном режиме, по-видимому, более эффективно применять последние методы, а не пытаться достигнуть существенного повышения производительности средствами оптимизации программ.

6.9. СИСТЕМЫ РАЗРАБОТКИ ПО

В данном параграфе кратко рассмотрены системы, которые используются при разработке программ для микропроцессоров. Поскольку для разработки ПО трудно использовать сами микропроцессоры, было создано множество систем, которые позволяют разрабатывать программы с помощью специального ПО и периферийных устройств, а затем переносить их на ту ЭВМ, где они будут выполняться. Имеются три типа систем разработки ПО: 1) микромашинные, 2) использующие системы разделения времени и 3) системы, основанные на других ЭВМ. Некоторые особенности этих систем были рассмотрены в предыдущих параграфах.

Микромашинная система разработки состоит из соответствующей микро-ЭВМ (или ее эмулятора), дополненной аппаратурой и ПО, облегчающими процесс разработки. В общем случае системы разработки имеют следующие средства:

1) индикации на лицевой панели, позволяющие программисту отслеживать содержимое шин и ячеек памяти;

2) для изменения содержимого ячеек памяти;

3) позволяющие осуществить запуск процессора в определенном состоянии;

4) одношаговый режим, позволяющий в процессе отладки осуществлять покомандное выполнение программы;

5) позволяющие выполнить программу, начиная с заданной ячейки памяти;

6) ПЗУ и ППЗУ, которые могут использоваться в качестве памяти программ;

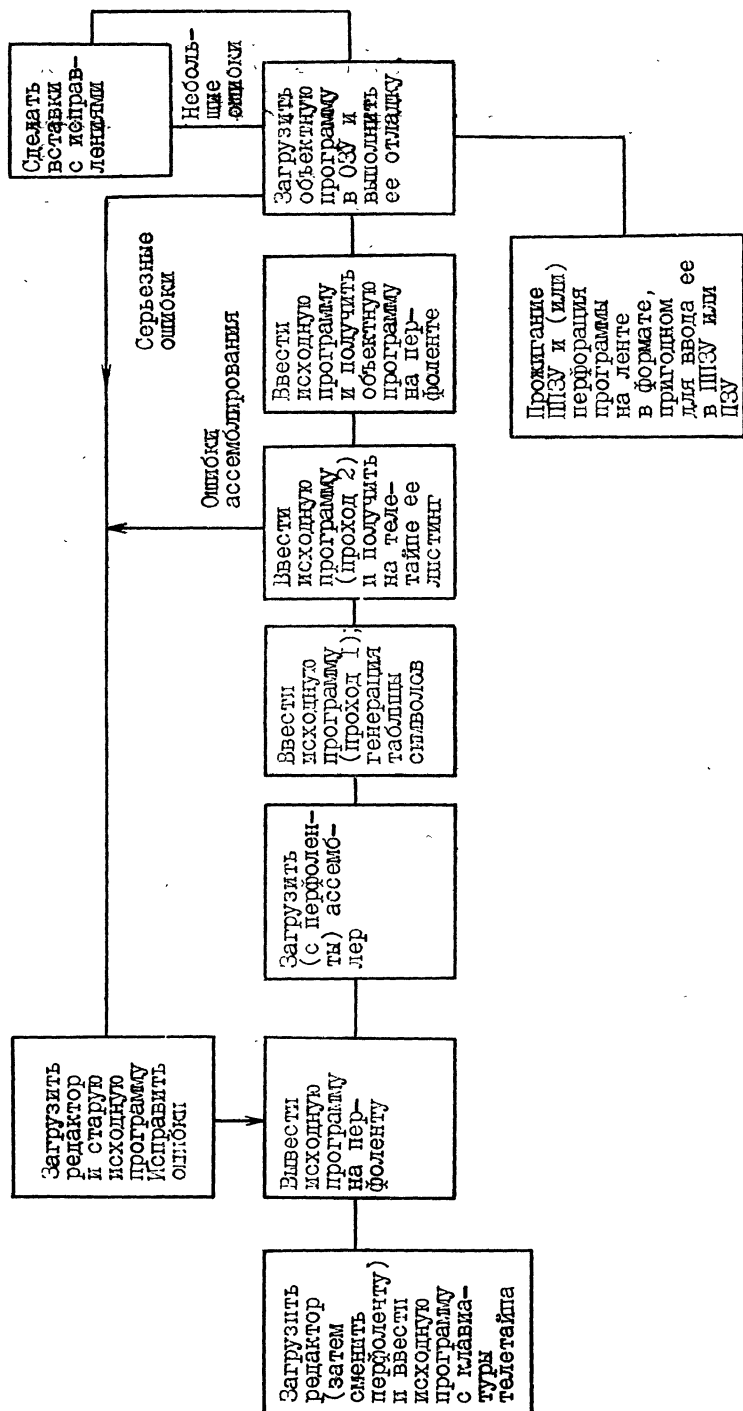


Рис. 6.20. Разработка с использованием микромашиной системы

Рис. 6.21. Стандартный телетайп

7) интерфейсы для подключения телетайпов и других стандартных устройств ввода-вывода, типа устройств ввода с перфоленты, клавиатур, дисплеев на светодиодах, построчно или посимвольно печатающих устройств, кассетных накопителей на магнитных лентах и накопителей на гибких дисках;

8) начальный загрузчик, осуществляющий ввод в память микро-ЭВМ таких программ, как стандартный загрузчик;

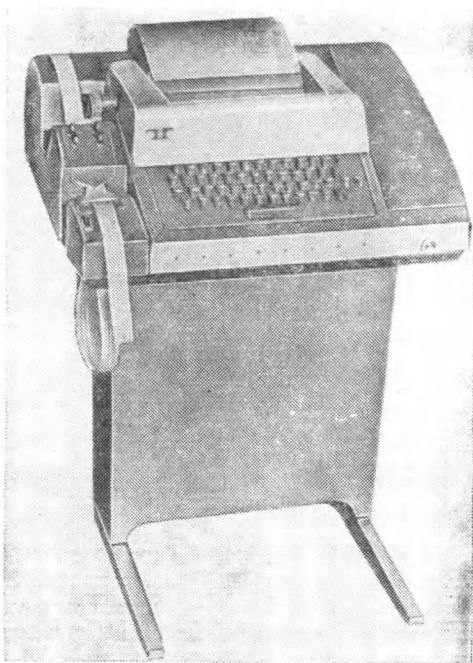
9) служебные программы (утилиты), загружающие программы в память ЭВМ с клавиатуры или с перфолент.

Даже простая система разработки позволяет программисту просматривать и менять содержимое ячеек памяти, а также вводить программы на машинном языке с помощью устройства ввода с перфоленты или клавиатуры телетайпа. Пользователь должен подключить к системе телетайп. Обычно система разработки имеет для этого соответствующий интерфейс и необходимое ПО.

К числу других средств, предоставляемых большинством систем разработки, относятся:

- 1) собственно ассемблер;
- 2) системный монитор;
- 3) средства задания контрольных точек или инициирования трассировок;
- 4) коннекторы для подключения внешних устройств к шинам процессоров;
- 5) редактор, который можно использовать для внесения небольших изменений в программах пользователя.

На рис. 6.20 приведена структурная схема процесса разработки программы с помощью простой системы разработки. Такой процесс занимает много времени, поскольку телетайп (рис. 6.21) работает медленно. Он печатает информацию со скоростью 10 символов/с. Кроме того, он предназначен для выдачи небольших объемов и не может систематически использоваться для выдачи длинных листингов. Устройство ввода с перфоленты вводит 10 символов/с. В результате для ввода в ЭВМ ассемблера или длинной программы пользователя может потребоваться около 30 мин.



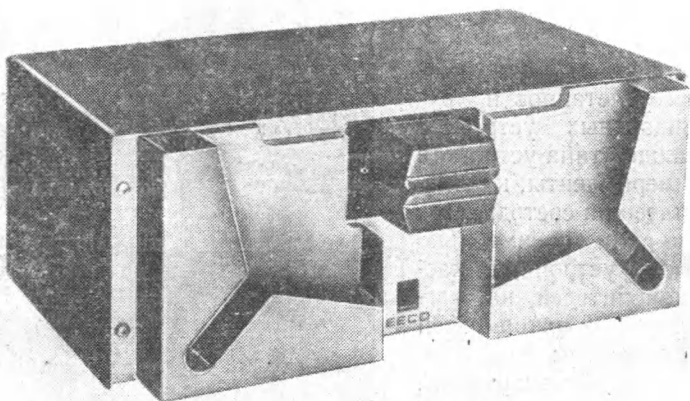


Рис. 6.22. Быстродействующее устройство ввода с перфоленты

Скорость ввода-вывода информации можно увеличить за счет использования дополнительных средств. Следует, однако, иметь в виду, что каждое из таких средств стоит значительно дороже, чем простая система, основанная на стандартном телетайпе. К числу дополнительных средств относятся:

1. Быстродействующее устройство ввода с перфоленты и вывода на перфоленту (рис. 6.22). Быстродействующее устройство ввода с перфоленты вводит информацию со скоростью 300 символов/с. Такое устройство позволяет уменьшить время ввода программы до нескольких минут.

2. Кассетный накопитель на магнитной ленте (рис. 6.23). Это устройство имеет такое же быстродействие, как и быстродействующее устройство ввода с перфоленты, и представляет собой более удобное (хотя и более дорогое) средство хранения информации. Для подклю-

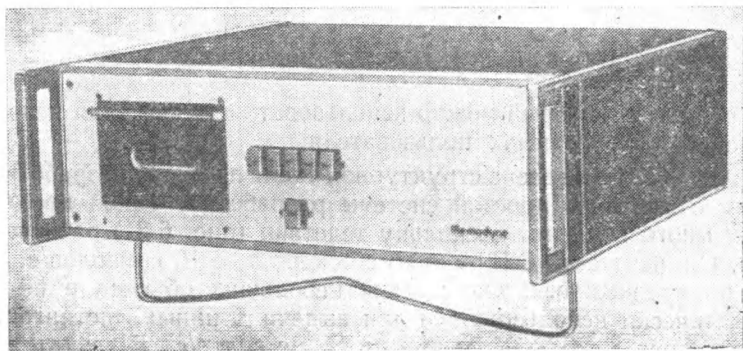


Рис. 6.23. Кассетный накопитель на магнитной ленте

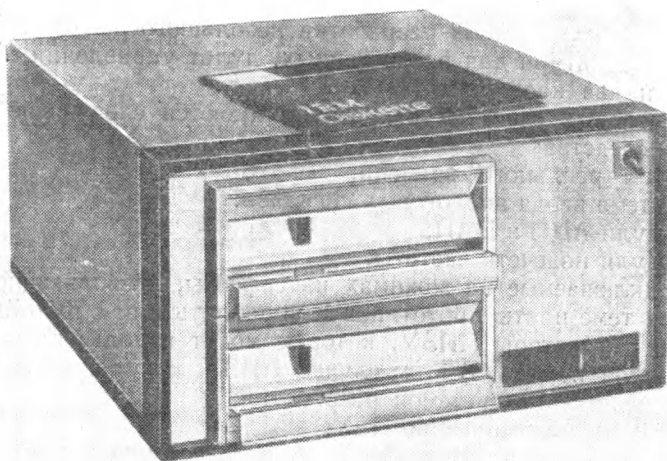


Рис. 6.24. Накопитель на гибком диске (ICOM 360)

ния кассетного накопителя также потребуется свой интерфейс, так как в большинстве систем этот интерфейс отличается от стандартного, используемого для подключения телетайпа. Производительность кассетных накопителей изменяется в широких пределах, некоторые из них оказываются недостаточно помехоустойчивыми для использования в ЭВМ.

3. Накопитель на гибком диске (рис. 6.24). На таком накопителе легко хранить и выбирать программы, так как среднее время доступа этого устройства существенно меньше, чем у кассетных накопителей и устройств ввода с перфоленты. Однако накопитель на гибком диске очень дорог и требует развитого программного обеспечения.

4. Построчно или посимвольно печатающее устройство. Подобные печатающие устройства обычно имеют в несколько раз большее быстродействие, чем стандартный телетайп, и могут использоваться для выдачи длинных листингов. Пользователь должен обеспечить интерфейс между печатающим устройством и системой разработки.

5. Дисплей на базе ЭЛТ. Такой дисплей более удобен при отладке и редактировании программ, чем печатающее устройство. Однако могут возникнуть сложности обеспечения интерфейса с дисплеем; кроме того, дисплей не обеспечивает твердой копии данных.

6. Интерактивный редактор. Программа-редактор удобна для внесения небольших изменений в программы, но требует большого объема памяти.

Удобная микромашинальная система разработки должна иметь в своем составе механизм быстрого получения твердой копии и достаточный объем памяти. Очень полезным может быть также дисплей на базе ЭЛТ. Очевидно, что чем больше возможностей имеет система разработки, тем более удобной (и дорогой) она становится. На рис. 6.25 по-

казана типичная система разработки небольшого размера, в которой имеется клавиатура для ввода данных, пульт управления оператора, дисплей и два кассетных НМЛ.

В некоторых системах разработки имеются следующие дополнительные средства:

- 1) часы реального времени;
- 2) система ввода аналоговых сигналов;
- 3) модули АЦП и ЦАП;
- 4) модули подсчета частоты;
- 5) подключаемые на зажимах интерфейсы, позволяющие подключать к системе платы прототипов и управлять ими с пульта;
- 6) программаторы ППЗУ, которые могут использовать внешнюю оперативную память. В этом случае ППЗУ могут программироваться непосредственно системой разработки;
- 7) модули, имитирующие ПЗУ;
- 8) контроллеры прерывания и прямого доступа к памяти (ПДП);
- 9) интерфейсы для УАПП;
- 10) сенсорные переключатели, значения которых могут считываться из программы;
- 11) интерфейсы стандартной шины;
- 12) модули контроллеров двигателей.

Достоинства микромашинных систем разработки является их небольшая стоимость и тесная связь с микро-ЭВМ, для которой ведется разработка. Такие системы являются замкнутыми и обладают теми же временными характеристиками и интерфейсами, что и микро-ЭВМ. Такие системы особенно удобны, если прототип может быть подключен к системе и может контролироваться с пульта или других системных устройств. Недостатками подобных систем являются низкое быстродействие, ограниченный набор периферийных устройств и средств ПО, а



Рис 6.25. Микромашинная система разработки

также их привязка к конкретному типу процессора. Большую пользу могут принести внешние интерфейсы, но их использование может осложнить разработку системы, потому что они зависят от возможностей, имеющихся в системе разработки, а не в конечном продукте. Сейчас получают распространение более совершенные и гибкие микромашинные системы разработки, которые решают многие из указанных проблем за счет несколько больших первоначальных затрат.

Во втором типе систем разработки ПО используются с и с т е м ы раз д е л е н и я в р е м е н и. В большинстве систем разделения времени имеются имитаторы и кросс-ассемблеры для распространенных микропроцессоров. В системах разделения времени предоставляются большие объемы легко доступной памяти, развитые интерактивные средства и быстродействующие периферийные устройства.

На рис. 6.26 показан процесс разработки ПО для микропроцессоров с использованием системы разделения времени. Достоинствами использования системы разделения времени являются развитые программные средства, невысокие первоначальные затраты, независимость от конкретного типа микропроцессора и доступ к быстродействующим периферийным устройствам. К числу недостатков относятся высокий уровень постоянных затрат, невозможность тестировать программы с использованием реальной аппаратуры и необходимость использования средств (например, быстродействующего печатного устройства), которые находятся на значительном удалении, что увеличивает время программного цикла. Средства системы разделения времени часто удобно использовать, если разработку ПО желательно начать еще до приобретения своего оборудования.

Третьим типом систем разработки ПО для микропроцессора являются с и с т е м ы, о с н о в а н н ы е н а д р у г и х Э В М. Для этого необходимо приобрести кросс-ассемблер и программный имитатор, которые будут работать на большой или малой ЭВМ. Этот вариант разработки ПО дешевле, чем длительное использование услуг системы разделения времени. Однако для его реализации часто требуется модифицировать поставляемое ПО, чтобы приспособить его к особенностям определенной ЭВМ. Разумеется, использование этого типа систем зависит от возможностей используемой ЭВМ, которые обычно существенно меньше возможностей системы разделения времени. Работа в режиме пакетной обработки без программы-редактора и с ограниченной памятью существенно менее удобна, чем использование мощной интерактивной системы. На рис. 6.27 показан процесс разработки ПО в режиме пакетной обработки.

Некоторые системы разработки основаны на мини-ЭВМ. Большие мини-ЭВМ обычно имеют в своем составе быстродействующее печатающее устройство, развитый редактор, интерактивный дисплейный терминал и большой накопитель на дисках. Такая мини-ЭВМ также удобна для разработки ПО, как и система разделения времени. Кроме того, можно организовать интерфейс с шиной ввода-вывода мини-ЭВМ, который позволит передавать объектный код, получаемый в мини-ЭВМ, непосредственно в ОЗУ микро-ЭВМ. На рис. 6.28 представлена структурная схема подобной системы.

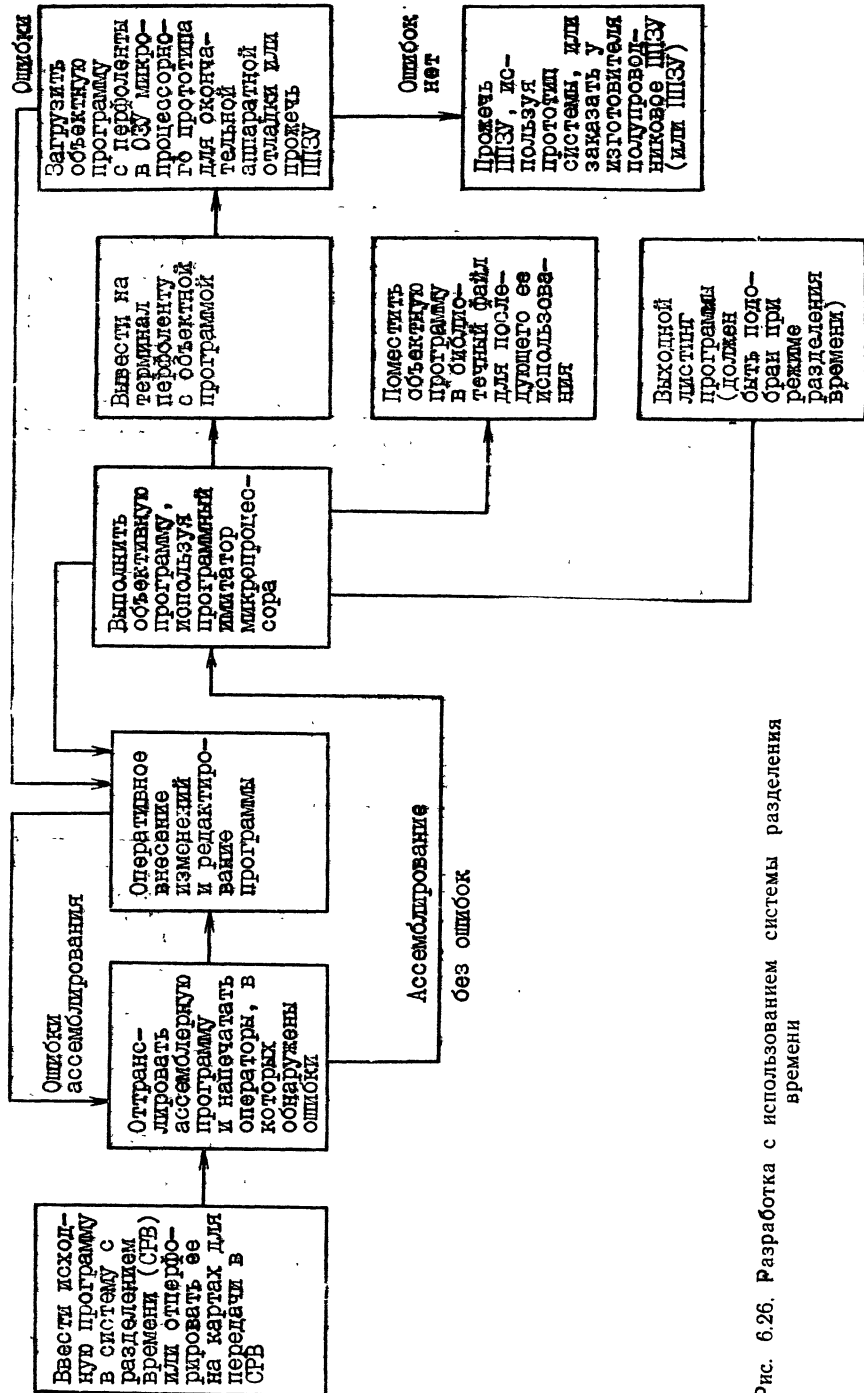


Рис. 6.26. Разработка с использованием системы разделения времени

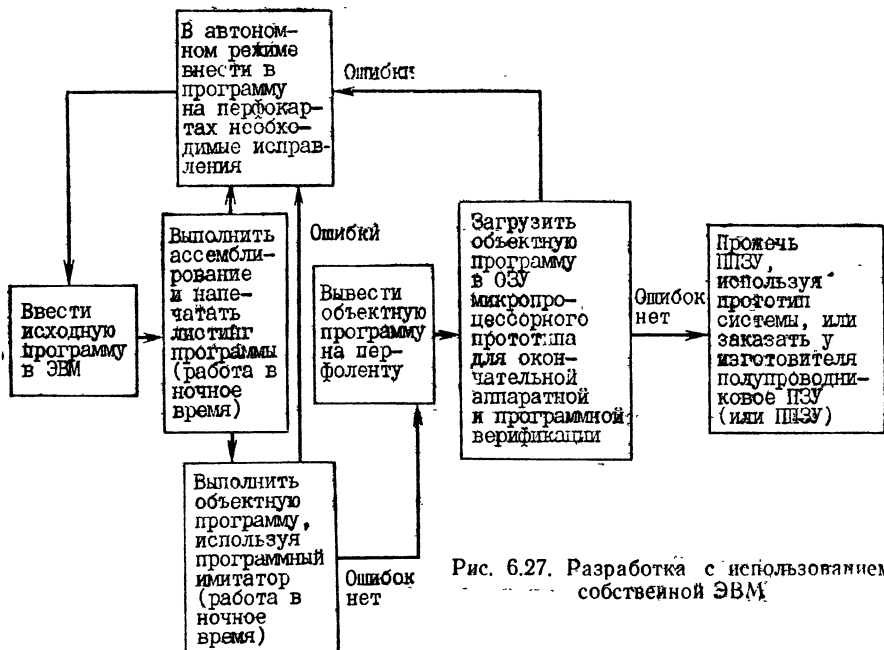


Рис. 6.27. Разработка с использованием собственной ЭВМ

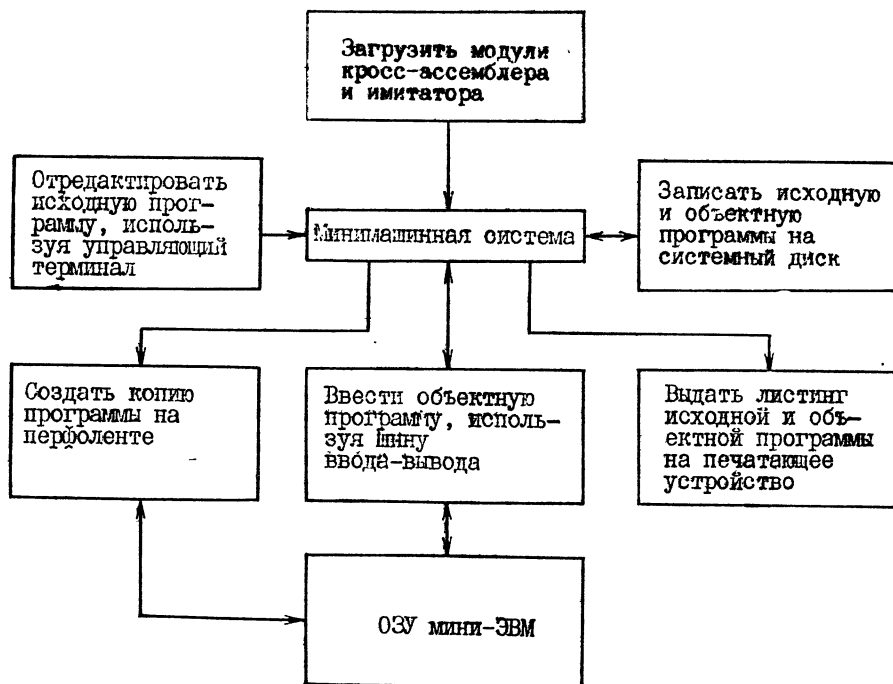


Рис. 6.28. Структурная схема минимашинной системы разработки

Существует несколько типов систем, объединяющих мини-ЭВМ и микро-ЭВМ. В одних системах просто обеспечивается пересылка программ в память микро-ЭВМ, а все остальные функции выполняются под управлением микро-ЭВМ. В других пользователю предоставляется возможность выполнять и модифицировать программы, управлять портами ввода-вывода и выполнять другие функции с помощью микро-ЭВМ и ее системы управления. Системы последнего типа более удобны, но требуют более сложного интерфейса и более развитого ПО.

Минимашинная система разработки является дорогостоящей, ее следует использовать только в случае, если уже имеется мини-ЭВМ или если система будет использована при реализации нескольких проектов. Эти системы, как и системы разработки, основанные на микро-ЭВМ, позволяют выполнять отладку и тестирование программ на реальном оборудовании, предоставляя вместе с тем средства отладки, характерные для систем разделения времени. Минимашинные системы разработки могут использоваться для большого числа процессоров, хотя каждый из них требует своего интерфейса. Подобные системы, по-видимому, обладают большим потенциалом, особенно если они располагают средствами непосредственной отладки и тестирования.

6.10. ВЫВОДЫ

Процесс разработки ПО распадается на несколько этапов. Кодирование программ (составление команд на языке, понятном ЭВМ) является только одним из этапов разработки и при этом не самым важным. Сначала проектировщик должен поставить задачу и спроектировать программу, затем написать ее, отладить, протестировать и документировать, а позднее расширить и заново спроектировать. Все этапы взаимосвязаны. Проектировщик часто одновременно выполняет работы по нескольким этапам, учитывая их взаимное влияние.

Этапы постановки и проектирования позволяют перейти от исходной формулировки задачи к такому ее представлению, которое удобно для программирования. На этапе постановки задачи следует определить входы и выходы программ, требования к процессу обработки, ограничения по времени и памяти и способы обработки ошибок. Проектирование программы требует точного описания ее логики и всех временных соотношений. Чтобы упростить кодирование, отладку и тестирование программ, следует использовать методы блок-схем, модульного и структурного программирования и нисходящего проектирования.

Хорошо написанная (закодированная) программа упрощает работу на последующих этапах. Четкость и удобочитаемость программы, простота ее структуры и тщательное документирование упрощают процесс отладки и тестирования. Если это потребует впоследствии, проектировщик может сделать программу более эффективной.

Отладка, тестирование и документирование представляют собой этапы разработки ПО, которые следует выполнять тщательно и систематически. Контрольные точки, трассы, дампы памяти и пошаговый режим выполнения программы полезны для отладки и тестирования программ. С помощью программных имитаторов может быть проверена логика программы, а логические или микропроцессорные анализаторы могут помочь в решении вопросов синхронизации и других аппаратных проблем. Документирование требует составления блок-схем, хорошей организации программы и наличия комментариев в программе, карт памяти и других справочных материалов.

Целью разработки ПО для микропроцессора является создание надежного и хорошо документированного ПО при умеренной стоимости разработки. Особое **внимание при достижении этой цели требуется уделить этапам проектирования, отладки и тестирования.** В редких случаях при разработке ПО для больших систем **следует уделить внимание стоимости аппаратных средств и памяти,** хотя эти факторы проектировщик должен учитывать всегда.

ГЛАВА СЕДЬМАЯ

МОДУЛИ ПАМЯТИ МИКРОПРОЦЕССОРОВ

До сих пор в данной книге микропроцессоры рассматривались как самостоятельные устройства, которые каким-то образом выбирают данные и команды из памяти, посылают их в память и с помощью модулей ввода-вывода осуществляют обмен информацией с внешней средой. В следующих трех главах будет рассматриваться вопрос о том, каким образом функционирует микропроцессор, выполняющий в микро-ЭВМ роль ЦП. В настоящей главе описаны принципы взаимодействия микропроцессора с памятью. В первую очередь рассмотрены общие характеристики этого взаимодействия, принципы построения простых модулей памяти, состоящих из простейшего ПЗУ или простейшего ОЗУ, а также конструирование и реализация интерфейса для более сложных модулей памяти. В заключительной части главы рассмотрены циклы команд и принципы конструирования модулей памяти для МП Intel 8080 и Motorola 6800.

В данной книге рассматриваются модули памяти, состоящие из корпусов памяти, выполненных на полупроводниках. В большинстве микро-ЭВМ используются корпуса именно такого типа. Память на магнитных сердечниках, которая используется в больших ЭВМ, в микро-ЭВМ используется редко.

Полупроводниковые ЗУ имеют следующие характерные особенности:

1. *Неразрушающее считывание.* При чтении данных из полупроводникового ЗУ содержимое его ячеек (в отличие от ЗУ на магнитных сердечниках) не меняется, что позволяет не выполнять регенерацию путем записи данных в память.

2. *Оперативные ЗУ.* При отключении питающих напряжений содержимое ячеек полупроводниковых ОЗУ не сохраняется. По этой причине во многих микро-ЭВМ для хранения программ используют ПЗУ, что дает возможность не выполнять повторную загрузку при каждом включении питающих напряжений.

3. *Одиночные модули.* Полупроводниковые ЗУ выполняются в виде отдельных корпусов различного размера, с различной длиной слова и различной организацией. Средства дешифрирования и интерфейс являются частью корпуса. Полупроводниковые ЗУ обладают такими же свойствами, как и другие ИС, изготовленные по той же технологии.

7.1. ОБЩИЕ ХАРАКТЕРИСТИКИ ИНТЕРФЕЙСА ПАМЯТИ

На рис. 7.1 показан интерфейс между микропроцессором и памятью. В качестве интерфейса используются:

- 1) шина адреса. Центральный процессор использует эту шину для выбора некоторой ячейки памяти при выборке или записи данных;
- 2) входная шина данных. Блок памяти использует эту шину для передачи содержимого ячейки памяти в ЦП;

3) выходная шина данных. Центральный процессор использует эту шину для передачи данных в память;

4) шина управления. По этой шине передаются управляющие и синхронизирующие сигналы. К сигналам, передаваемым в память, относятся сигналы, синхронизирующие цикл памяти и процессора, сигналы записи и адресные или информационные стробирующие сигналы. Сигналы, посылаемые в ЦП, обычно сообщают об успешном доступе к памяти.

Основным процессом взаимодействия микропроцессора и модуля памяти является процесс выборки команды. Прежде чем выполнить некоторое действие, ЦП должен выбрать команду из памяти. Выбранная команда определяет действие ЦП в оставшейся части цикла команды. Выборка команды (рис. 7.2) осуществляется следующим образом.

Шаг 1. Центральный процессор посылает в адресную шину содержимое счетчика команд (СК).

Шаг 2. Центральный процессор ждет получения данных из памяти.

Шаг 3. Центральный процессор помещает данные с входной шины данных в регистр команд (РК).

Шаг 1 определяет, откуда берется команда. На шаге 2 для обеспечения корректности передачи данных осуществляется синхронизация работы процессора и памяти. На шаге 3 информация, полученная из памяти, интерпретируется как команда, а не как адрес или данные.

Для выполнения шага 1 требуется синхронизирующий сигнал, который помещает содержимое СК на адресную шину. Синхронизирующий сигнал поступает от процессора и представляет собой регулярную последовательность сигналов, вырабатываемых тактовым генератором. Особенности внутренней организации процессора определяют частоту и ширину тактовых сигналов.

На шаге 2 ЦП должен суметь определить тот момент, когда данные из памяти становятся доступными. Разумеется, на протяжении всего этого промежутка времени содержимое адресной шины должно оставаться постоянным. С точки зрения правильности работы ЭВМ время задержки является критичным. Если ЦП не будет ожидать в течение

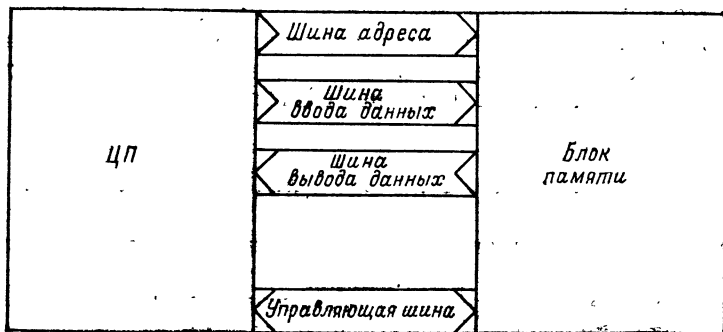
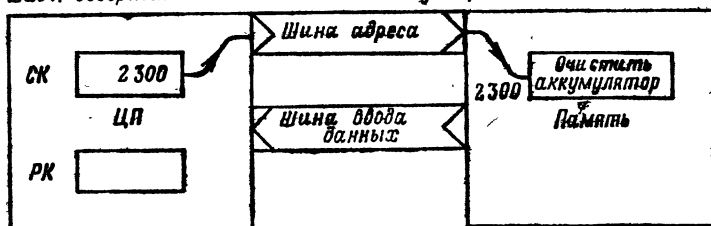
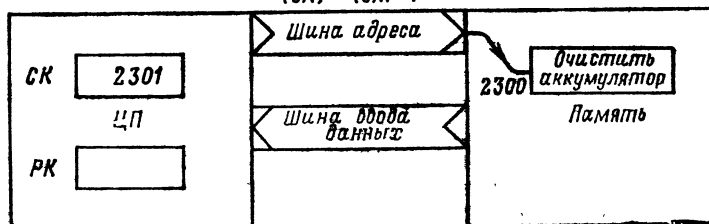


Рис. 7.1. Интерфейс между ЦП и памятью

Шаг 1. Содержимое СК подается на шину адреса



Шаг 2. Ожидание доступности данных
 $(СК) = (СК) + 1$



Шаг 3. Передача команды из шины данных в РК

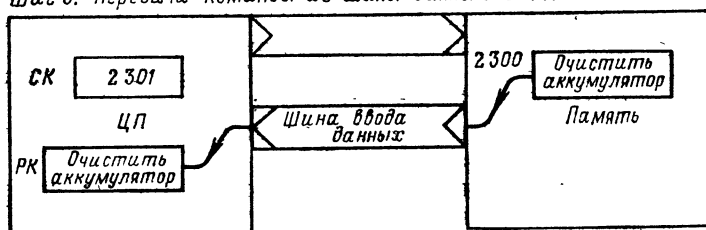


Рис. 7.2. Выборка команды

достаточного времени, прием команды может завершиться некорректно. Такую ошибку трудно обнаружить, а ее возникновение может привести к непредсказуемому результату. Задача состоит в том, чтобы определить длительность задержки. Простейшим вариантом решения этой проблемы является выбор такого типа памяти, время доступа к которой меньше периода тактового генератора. В этом случае время ожидания определяется периодом тактового генератора. Более сложные варианты определения временной задержки будут рассмотрены далее.

Для выполнения шага 3 требуется синхронизирующий сигнал, по которому содержимое входной шины данных помещается в регистр команд. Если время доступа к памяти меньше периода тактового генератора, то этим синхронизирующим сигналом может быть следующий тактовый сигнал генератора тактовых импульсов процессора. С помощью триггера можно определить, является ли данный цикл циклом вывода адреса или циклом ввода данных.

Проектировщик должен обеспечить согласованность временных характеристик блока памяти и процессора. Синхронизирующие сиг-

налы, по которым содержимое СК помещается на шину адреса, а содержимое шины ввода данных — в регистр команд, определяются архитектурной ЦП. Блок памяти должен предоставить данные в течение соответствующего временного интервала.

Кроме выборки команд, ЦП должен выбрать из памяти данные и адреса. Разница между этими двумя операциями выборки и выборкой команды состоит в том, что адрес берется из одного места памяти, а данные посылаются в другое. Синхронизация циклов осуществится аналогично. Если время доступа ко всем частям модуля памяти останется неизменным, то синхронизацию циклов можно осуществить так же, как и при выборке команд. Другие циклы чтения включают в себя операции выборки:

1) адресной части команды. В данном случае адрес находится в СК, а ЦП помещает данные в адресный регистр;

2) данных из ОЗУ. В данном случае адрес содержится в адресном регистре и ЦП помещает данные в регистр данных.

3) непосредственных данных. В данном случае адрес содержится в СК, а ЦП помещает данные в регистр данных;

4) косвенного адреса. В данном случае адресом является содержимое адресного регистра, и ЦП помещает данные в адресный регистр.

Для примера предположим, что выполняется команда ADD 150, занимающая две ячейки памяти. В первой ячейке содержится код операции ADD, во второй — непосредственный адрес 150. Для выполнения данной команды требуется три цикла:

1) выборки команды. Центральный процессор выбирает из памяти операции и помещает его в регистр памяти;

2) выборки адреса. Центральный процессор выбирает из памяти адрес и помещает его в регистр адреса. При выполнении обоих циклов счетчик команд увеличивается на единицу;

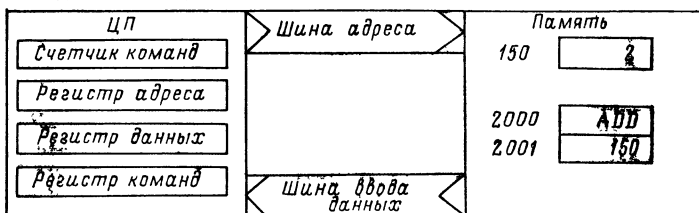
3) выборки данных. Используя регистр адреса, ЦП выбирает данные из памяти и помещает их в регистр данных. В этом цикле содержимое счетчика команд не меняется.

На рис. 7.3 показано, как в процессе выполнения команды осуществляется передача информации. Синхронизация осуществляется одинаково для всех циклов.

Очевидно, ЦП должен иметь возможность указывать адрес источника для адресной шины и адрес получателя для входной шины данных. Это обеспечивается с помощью управляющих сигналов, формируемых путем дешифрирования команды, формирования внутреннего сигнала и с помощью тактовых сигналов процессора. Источник для адресной шины может определить мультиплексор или селектор, как это показано на рис. 7.4. Демультимплексор может определить приемник для шины входных данных, как это показано на рис. 7.5. Чем больше источников и приемников, тем сложнее селекторы и демультимплексоры и тем больше управляющих сигналов требуется для управления ими. Эти внутренние устройства также создают задержки (от ввода до вывода). Селекторы и демультимплексоры являются базовыми элементами в схемах ЭВМ, так как они обеспечивают ту степень гибкости ЭВМ, которая позволяет широко их использовать.

Чтобы выполнить некоторые команды, процессор должен пересылать данные в память, т. е. выполнять операции записи в память. Циклы записи в память не обладают такой же степенью общности, как циклы чтения содержимого памяти. Многие микро-ЭВМ не имеют

1. Исходное состояние



2. Цикл выборки команд

$$(\text{Регистр команд}) = (2000)$$

$$(\text{счетчик команд}) = (\text{счетчик команд}) + 1$$

$$= 2001$$



3. Цикл выборки адреса

$$(\text{Регистр адреса}) = (2001)$$

$$(\text{счетчик команд}) = (\text{счетчик команд}) + 1$$

$$= 2002$$



4. Цикл выборки данных

$$(\text{регистр данных}) = ((\text{регистр адреса}) = (150)) = 2$$



Рис. 7.3. Команда с тремя циклами чтения (ADD 150)

оперативной памяти и не выполняют операций записи в память. В таких системах для временного хранения данных используются регистры микропроцессора.

Цикл записи в память подобен циклу чтения и выполняется следующим образом.

Шаг 1. Центральный процессор помещает содержимое регистра адреса в адресную шину.

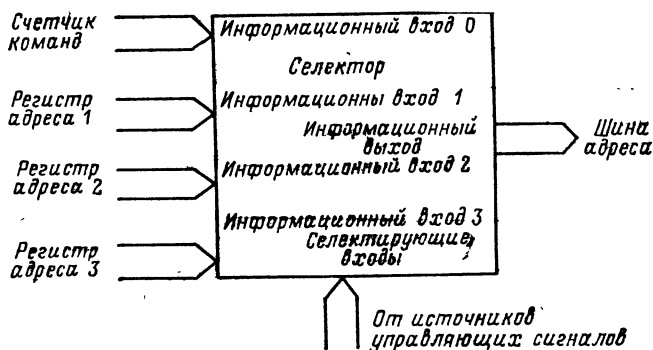
Шаг 2. Центральный процессор помещает содержимое регистра данных в шину вывода данных.

Шаг 3. Центральный процессор выдает в блок памяти сигнал записи.

Шаг 4. Центральный процессор ожидает успешного завершения операции записи.

Шаг 1 цикла записи совпадает с соответствующим шагом при выборке данных. Для выполнения шага 2 требуется синхронизирующий сигнал, по которому данные помещаются на выходную шину данных. Получение данных из различных регистров данных можно обеспечить с помощью селектора.

Для выполнения шага 3 требуются средства генерации, которые формируют сигнал достаточной длительности для отправки в память.

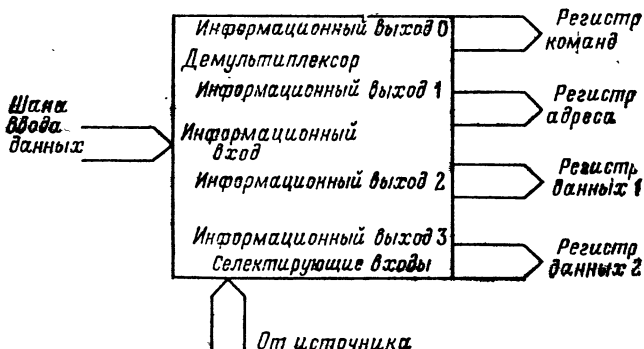


В соответствии с приведенной таблицей селектирующие входы определяют, какой информационный вход будет подключен к шине адреса:

Селектирующие входы	Содержимое шины адреса
0	Счетчик команд
1	Регистр адреса 1
2	Регистр адреса 2
3	Регистр адреса 3

Дополнительные селектирующие входы позволяют увеличить число источников для шины адреса

Рис. 7.4. Селекция адреса источника с помощью селектора



Селектирующие входы определяют назначение входных данных в соответствии с приведенной таблицей

Селектирующие входы	Назначение	Функция входных данных
0	Регистр команд	Команда
1	Регистр адреса	Адрес
2	Регистр данных 1	Данные
3	Регистр данных 2	Данные

Увеличение числа селектирующих входов позволяет увеличить число адресатов. В процессе демультиплексирования содержание памяти может рассматриваться как команда, адрес или данные.

Рис. 7.5. Определение адреса назначения данных с помощью демультиплексора

В цикле записи очень важно выдержать порядок передачи сигналов в память и их взаимную синхронизацию. Поскольку операция чтения является неразрушающей, порядок сигналов в цикле чтения не очень важен. Так, например, адрес должен достигнуть памяти до начала сигнала записи, а затем до окончания этого сигнала данные должны оставаться неизменными.

Шаг 4 синхронизирует операцию записи. Центральный процессор должен убедиться в том, что операция записи завершена. Как и в цикле чтения, для этого достаточно выбрать такой тип памяти, чтобы цикл записи завершался за один период тактового генератора.

Цикл записи более сложен, чем цикл чтения. В цикле чтения необходимо только сделать данные доступными ЦП; что происходит до и после обмена, значения не имеет. Цикл записи должен не только корректно пересылать данные, но и гарантировать, что не произойдут никакие предположки, которые меняют содержимое ячеек памяти.

Цикл чтения и записи памяти не являются единственными операциями процессора. Процессор должен также декодировать и выполнять команды. Следует обратить внимание на то, что ЦП не использует память при выполнении внутренних циклов, дешифрировании и вы-

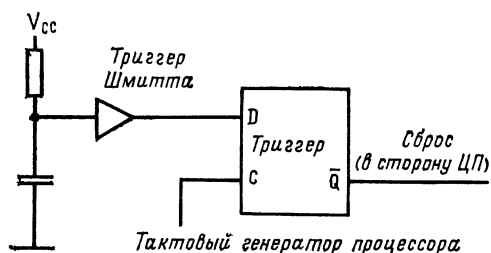


Рис. 7.6. Схема, реализующая операцию СБРФС

(триггер осуществляет задержку сигнала СБРФС таким образом, что этот сигнал возникает после сигнала тактового генератора процессора. Схема построена таким образом, что действующим значением сигнала СБРФС является 1)

полнении команды. Поэтому другие устройства, не создавая помех для ЦП во время выполнения этих циклов, могут использовать память (подобная процедура называется занятием цикла).

Циклы чтения и записи, а также декодирования и выполнения команды представляют собой нормальные операции ЦП. Под управлением своего тактового генератора ЦП непрерывно выбирает, декодирует и выполняет команды. Выполнение этих операций начинается в момент включения питания и запуска тактового генератора и завершается при отключении того или другого. Возникает вопрос, каким образом необходимо запустить ЦП, чтобы начать выполнение программы. Если это сделано, то ЦП будет выполнять команды программы до тех пор, пока его не отключат.

Управляющий сигнал, который переводит ЦП в определенное исходное состояние, называется сигналом СБРФС. Требования к этому сигналу и конкретные действия, которые он выполняет, различны для разных процессоров. Основное назначение сигнала СБРФС состоит в том, чтобы поместить в СК определенный адрес. Разработчик может создать программу, которая располагается в определенном месте памяти (обычно в ПЗУ) и передает управление основной программе. Часто для этого бывает достаточно одной команды передачи управления. Сигнал СБРФС может быть подан в любой момент, поступление этого сигнала вернет процессор в первоначальное состояние.

Сигнал СБРФС при включении напряжения питания можно реализовать с помощью резисторно-емкостной схемы, подключенной к источнику питания (рис. 7.6). Триггер Шмитта преобразует небольшое увеличение питающего напряжения в сигнал строго определенной формы. Многие процессоры (например, Zilog Z-80 и MOS Technology 6502) имеют в своем корпусе эту схему. В других процессорах (например, Intel 8080) предполагается, что схема находится в одном корпусе с тактовым генератором. Такой способ реализации сигнала СБРФС обеспечивает запуск процессора с определенного состояния и исключает все проблемы, связанные с переходными процессами при инициализации.

7.2. ПРОСТЫЕ МОДУЛИ ПАМЯТИ

Простые модули памяти состоят из одного корпуса памяти или простого набора корпусов с одинаковыми адресными соединениями. Для более сложных модулей памяти требуются схемы дешифрирования, которые активируют некоторую часть модуля для выполнения операций обмена.

Простейшие ЗУ

Простейший модуль памяти состоит из одного модуля ПЗУ. Простые контроллеры представляют собой двухкорпусные микро-ЭВМ, которые состоят из одного корпуса процессора и одного корпуса ПЗУ. На рис. 7.7 показано назначение выводов типичного ПЗУ. Адресные входы определяют, какое слово появится на информационных выходах. Схемы дешифратора являются частью такого ПЗУ. Каждая адресная линия увеличивает вдвое число адресуемых ячеек, в то же время число информационных линий равно длине ячейки. Большинство ПЗУ имеют длину ячейки 4 или 8 бит.

Постоянные ЗУ имеют небольшие габариты, так как в них отсутствуют информационные или управляющие входы. Для ПЗУ типа $A256 \times 8$ требуется от двух до четырех выводов для подключения питания и по восемь выводов для информационных и адресных шин. Таким образом, вся память системы может быть заключена в одном корпусе, имеющем от 18 до 24 выводов.

Синхронизация ПЗУ осуществляется просто. Максимальный промежуток времени от момента подачи адреса до момента выдачи данных называется *максимальным временем доступа*. Изготовитель гарантирует определенное значение этой величины в определенном диапазоне температур и в определенных операционных условиях. Данный промежуток времени является значением, которое необходимо учитывать при организации работы процессора. Предполагается, что на протяжении всего процесса доступа значение адреса остается постоянным.

На рис. 7.8 показаны входы и выходы типичного микропроцессора. Адресные выходы содержат адрес ячейки, к которой обращается ЦП. На информационные входы поступает содержимое этой ячейки. Поступление сигнала на вход СБРОС приводит к посылке в СК заранее известного адреса. Адресные шины микропроцессоров обычно имеют от 8 до 16 линий, а шины данных — от 4 до 8 линий.

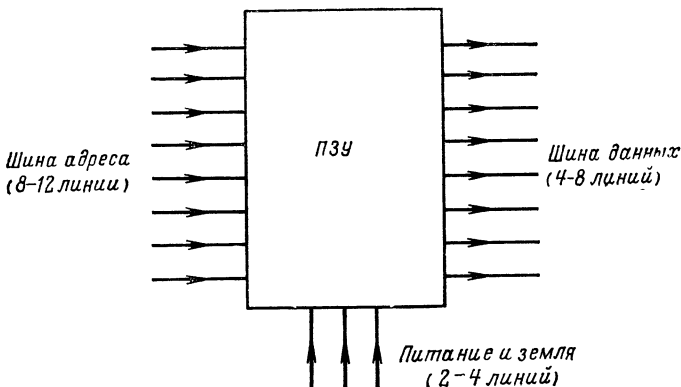


Рис. 7.7. Типичный модуль ПЗУ (не требуются специальные управляющие сигналы)

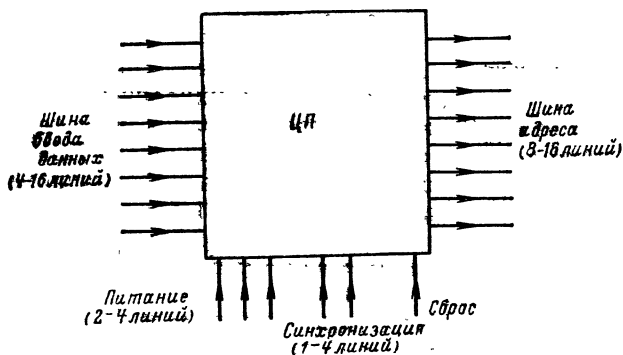


Рис. 7.8. Типичный модуль микропроцессора

Чтобы упростить взаимное соединение ЦП и ПЗУ, будем считать, что имеется:

- а) ПЗУ на 256 8-битных ячеек (как в типовом устройстве 1702);
- б) 8-битный процессор с 8-битной информационной и 8-битной адресной шинами;
- в) сигнал СБРОС, который обнуляет СК.

Несложно выполнить подключение при других допущениях.

На рис. 7.9 показано взаимное подключение ЦП и ПЗУ для оговоренной выше конфигурации. Адресные выходы ЦП подключены непосредственно к адресным входам ПЗУ. Информационные входы ЦП подключены непосредственно к информационным выходам ПЗУ. Система работает следующим образом:

- а) сигнал СБРОС запускает ЦП, заслав в СК нулевой адрес;
- б) ЦП выполняет программу, хранимую в ПЗУ, начиная с нулевого адреса.

Разумеется, ЦП не может осуществлять запись в ПЗУ.

Поскольку число адресных линий равно восьми, общая емкость памяти равна 256 ячейкам. Если фактически в ЦП имеются лишние адресные линии, то их можно просто оставить незадействованными, как в системе, структура которой показана на рис. 7.9. Значения сигналов на незадействованных линиях не участвуют в дешифрировании адреса. Таким образом, каждой ячейке ПЗУ будет соответствовать несколько адресов. Пусть подобная избыточность не смущает читателя, так как она не отражается на работе ЭВМ. Сложности возникают тогда, когда одному адресу соответствует несколько различных ячеек.

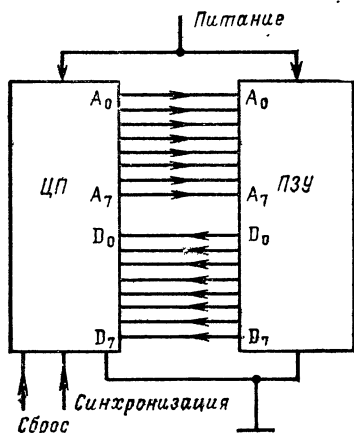


Рис. 7.9. Интерфейс между ЦП и ПЗУ

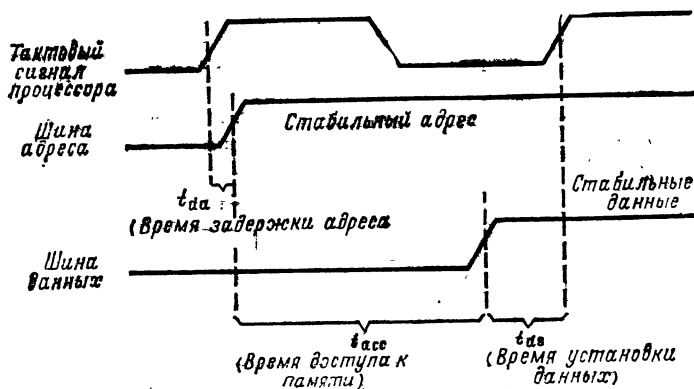


Рис. 7.10. Времена задержки и установки данных для цикла чтения в ЦП

Синхронизация с циклом памяти осуществляется следующим образом:

1. По тактовому сигналу процессор выставляет содержимое СК или регистра адреса на адресную шину. Максимальная задержка между фронтом тактового сигнала и появлением адреса на шине равна t_{da} (рис. 7.10).

2. Процессор ждет время, равное максимальному времени доступа к ПЗУ t_{acc} .

3. По тактовому сигналу процессор считывает содержимое информационной входной шины в регистр команд или регистр данных. Чтобы эта передача произошла корректно, данные должны стать доступными не менее чем за время t_{ds} до появления тактового сигнала. Величина t_{ds} — это время установки данных (рис. 7.10).

Чтобы ЦП и ПЗУ взаимодействовали правильно, необходимо, чтобы время чтения данных из ПЗУ было меньше времени, определенного допустимым числом тактовых циклов процессора. Это число фиксируется при проектировании ЦП и обычно равно 1 или 2, но в отдельных случаях может достигать значения 10 или 20. Если через k обозначить допустимое число циклов, то система должна удовлетворять условию

$$t_{da} + t_{acc} + t_{ds} < kt_p, \quad (7.1)$$

где t_p — период тактового генератора процессора. Значения t_{da} и t_{ds} определяются характеристиками ЦП.

Пример 1. Микропроцессор типа Intel 8080:

$$t_{da} = 200 \text{ нс}; t_{ds} = 130 \text{ нс}; k = 2; t_p = 500 \text{ нс}$$

(стандартное значение).

Чтобы обеспечить максимальное быстродействие ЦП, время доступа к ПЗУ должно удовлетворять условию

$$t_{acc} < kt_p - (t_{da} + t_{ds});$$

$$t_{acc} < 1000 - 330, \text{ т. е. } t_{acc} < 670 \text{ нс.}$$

Пример 2. Микропроцессор типа Motorola 6800:

$t_{da} = 300$ нс; $t_{ds} = 100$ нс; $k = 1$; $t_p = 1000$ нс (стандартное значение).

Чтобы обеспечить максимальное быстродействие ЦП, время доступа к ПЗУ должно удовлетворять условию

$$t_{acc} < kt_p - (t_{da} + t_{ds});$$
$$t_{acc} < 1000 - 400, \text{ т. е. } t_{acc} < 600 \text{ нс.}$$

На практике МП Intel 8080 и Motorola 6800 предъявляют еще более жесткие требования к времени доступа (см. 7.5), так как каждый из них имеет две фазы тактового сигнала и более сложную систему синхронизации, чем та, которая была описана.

Кроме того, некоторые процессоры и ЗУ не могут взаимодействовать непосредственно, поскольку они работают с различными уровнями напряжения. В этом случае микро-ЭВМ потребуются буферные устройства. Буферные устройства могут изменять уровень напряжения, вырабатывать дополнительный управляющий токовый сигнал или преобразовывать уровни сигналов, соответствующие МОП- и ТТЛ-технологии¹. В этой главе будут еще рассмотрены типичные буферные устройства и драйверы.

Для системы с буферами соотношение (7.1) должно быть изменено с учетом максимального времени задержки в буферах. Если t_{ab} — максимальное время задержки в адресном буфере, а t_{db} — максимальное время задержки в информационном буфере, то соотношение, определяющее согласованность ЦП и ПЗУ, примет вид

$$t_{da} + t_{acc} + t_{ds} + t_{ab} + t_{db} < kt_p. \quad (7.2)$$

Разумеется, желательно иметь как можно меньшие задержки в буферах. По этой причине в большинстве микро-ЭВМ используются буфера, выполненные по ТТЛ-технологии с временами задержки от 20 до 50 нс.

В простой системе, анализируя уровни потенциальных и токовых сигналов, требуемых и выдаваемых ЦП и ПЗУ, можно легко определить, требуются буфера или нет. Поскольку модули, выполненные по МОП-технологии, вырабатывают малые управляющие токовые сигналы и редко могут непосредственно управлять более чем одним элементом, выполненным по ТТЛ-технологии, одновременно, то буферы в микро-ЭВМ обычно используются. Кроме того, даже для устройств, выполненных по МОП-технологии, которые совместимы с устройствами, выполненными по ТТЛ-технологии, может потребоваться напряжение, более близкое к номинальному, чем для стандартных устройств ТТЛ.

Спроектировать базовый интерфейс между микропроцессором и ПЗУ несложно. Для обеспечения его правильной работы должно выполняться только одно неравенство [см. (7.1) и (7.2)].

¹Номинальные уровни сигналов в схемах ТТЛ-технологии составляют 0 и +5В, а в схемах МОП-технологии 0 и —15 В для логических 0 и 1 соответственно.

Простейшая оперативная память

Рассмотрим интерфейс между микропроцессором и простейшим ОЗУ. На рис. 7.11 показан типичный модуль ОЗУ. Будем считать, что ОЗУ состоит из 256 8-битных ячеек и подключено к 8-битному процессору. В данном случае операции обмена включают в себя как ввод, так и вывод данных, а также сигнал записи.

Цикл чтения для ОЗУ такой же, как для ПЗУ. Однако программу перед запуском следует загрузить в ОЗУ, так как при отключении питания содержимое ОЗУ не сохраняется. Процесс может начать работу только после загрузки программы.

Цикл записи для ОЗУ значительно сложнее цикла чтения. Характеристики ОЗУ должны отвечать следующим требованиям:

- 1) сигнал записи должен иметь минимальную длину;
- 2) адрес должен быть стабильным в течение минимального промежутка времени до начала сигнала записи;
- 3) данные должны быть стабильными в течение минимального промежутка времени до конца сигнала записи;
- 4) данные должны быть стабильными в течение минимального промежутка времени по окончании сигнала записи.

Для операции записи очень важен порядок сигналов. Прежде чем будет получен сигнал записи, должен быть установлен адрес (t_{as} — требуемое время установки адреса), а данные должны быть установлены до (t_{ds} — требуемое время установки данных) и задержаны после (t_{dh} — требуемое время задержки данных) окончания сигнала записи. Совмещение во времени установившегося состояния адреса, данных и сигнала записи должно быть минимальным (t_w). На рис. 7.12 показана временная диаграмма цикла записи для ОЗУ. В данном случае требуются три входных сигнала вместо одного при использовании ПЗУ.

Временная диаграмма цикла записи для ЦП значительно сложнее (рис. 7.13). Время задержки адреса такое же, как в цикле чтения. Вместе с тем появляются времена задержки данных (t_{dd}) и сигнала

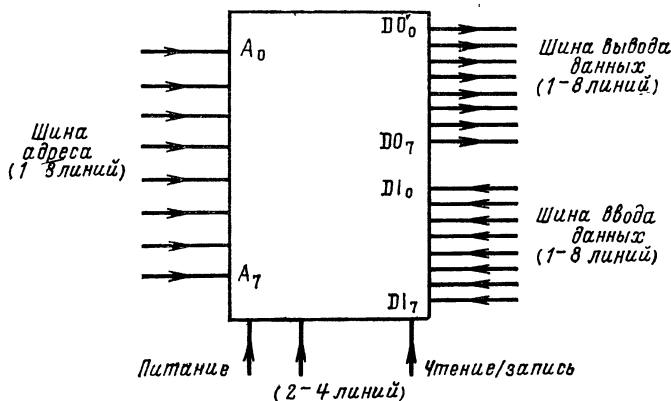


Рис. 7.11. Типичный модуль ОЗУ

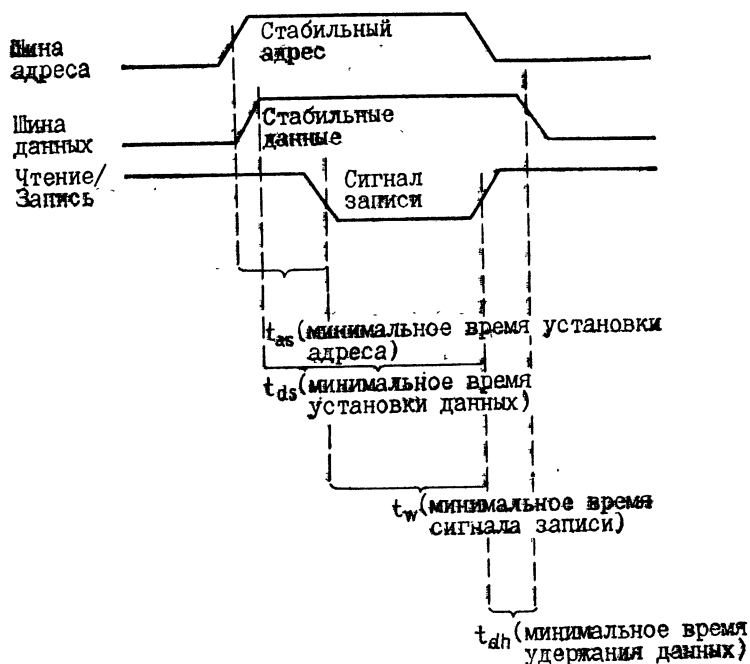


Рис. 7.12. Цикл записи для ОЗУ

записи (t_{dw}). Процессор должен выдать сигнал записи после цикла адреса (через t_{ew}^+) и закончить его до следующего цикла адреса и данных (за t_{ew}^-).

Схема интерфейса между процессором и ОЗУ приведена на рис. 7.14. Для цикла записи должны выполняться следующие временные соотношения:

1. Длительность сигнала записи от процессора должна быть больше минимального времени, определяемого из неравенства

$$t_{wp} > t_w. \quad (7.3)$$

2. Сигнал записи должен быть задержан на необходимый промежуток времени от момента стабилизации адреса. Другими словами, сигнал записи должен быть задержан настолько, чтобы обеспечивался прием корректного адреса до поступления сигнала записи:

$$t_{ew}^+ > t_{da} + t_{as}. \quad (7.4)$$

3. Данные должны оставаться стабильными в течение минимального промежутка времени после окончания сигнала записи. Другими словами, ЦП должен завершить сигнал записи до смены данных:

$$t_{ew}^- > t_{dh}. \quad (7.5)$$

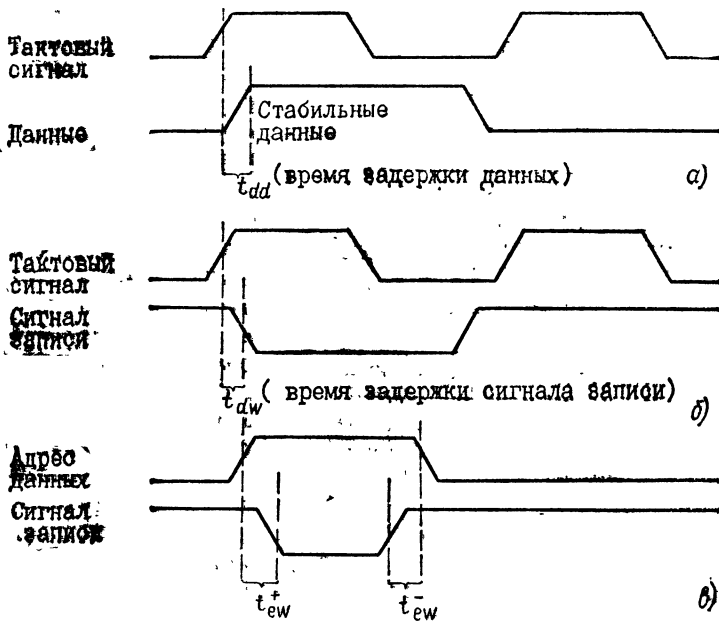


Рис. 7.13. Временная диаграмма цикла записи для ЦП:

а — задержка данных; б — задержка сигнала записи по отношению к тактовому сигналу; в — задержка сигнала записи по отношению к данным и адресу; t_{dd} — максимальная задержка между фронтом тактового сигнала и появлением стабильных данных на шине данных; t_{dw} — максимальная задержка между фронтом тактового сигнала и сигналом записи; t_{ew}^+ — минимальная задержка между стабильным адресом и сигналом записи; t_{ew}^- — минимальная задержка между концом сигнала записи и сменой данных (подготовка к следующему циклу)

4. Вся операция пересылки должна завершиться в течение k периодов тактового генератора:

$$t_{da} + t_{as} + t_w + t_{ew}^- < kt_p; \quad (7.6a)$$

$$t_{dd} + t_{ds} + t_{dh} > kt_p. \quad (7.6б)$$

Все неравенства являются линейными, но найти их совместное решение нелегко. Обычно для обеспечения правильной синхронизации оставляют запас времени. Такой запас всегда необходим для компенсации колебания напряжения в сети, температуры окружающего воздуха и других внешних и электрических воздействий.

Если в системе есть буфера, они будут вызывать задержку данных, адресов и сигналов записи. Если предположить, что время задержки в буфере равно t_b , то модифицированные соотношения принимают следующий вид:

$$t_{wp} - t_b > t_w; \quad (7.7)$$

$$t_{ew}^+ > t_{da} + t_{as} + t_b; \quad (7.8)$$

$$t_{ew}^- > t_{dh} + t_b; \quad (7.9)$$

$$t_{da} + t_{as} + t_w + t_{ew}^- + t_b < kt_p; \quad (7.10a)$$

$$t_{dd} + t_{ds} + t_{dh} + t_b < kt_p. \quad (7.10б)$$

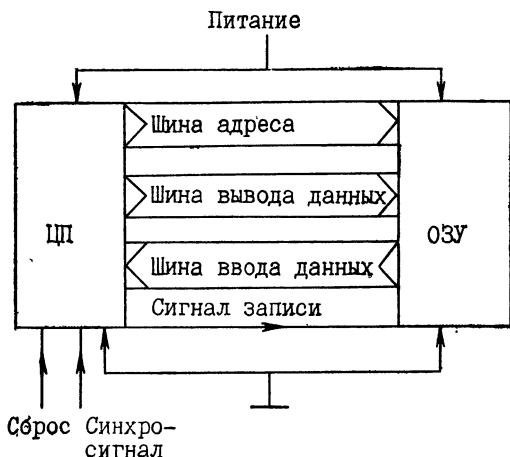
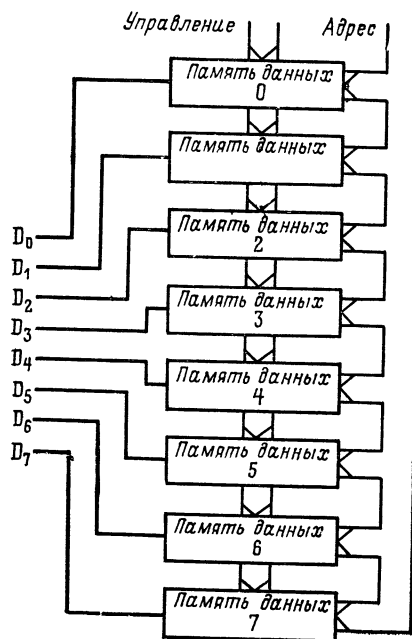


Рис. 7.14. Интерфейс между ЦП и ОЗУ

сто в микропроцессорах имеется одна двунаправленная шина данных. В некоторых микропроцессорах для обработки данных и адресов имеется только одна шина. В ОЗУ также может иметься единственный набор выводов для ввода и вывода данных (общий ввод-вывод в отличие от раздельного ввода-вывода).

В ОЗУ с малой длиной слова интерфейс обеспечивается следующим образом (рис. 7.15):



1. Адресные и управляющие соединения являются одними и теми же (общими) для всех корпусов памяти.

2. Каждый корпус памяти имеет индивидуальное подключение к шине данных. Эти соединения определяют, какой бит или биты слова ЭВМ содержит данный корпус.

Работа ЦП не будет зависеть от того, что фактически каждое слово памяти разбито на части. Для проектировщика такое разбиение выразится в увеличение содержимого счетчика числа элементов и степени загрузки адресных и уп-

Рис. 7.15. Взаимное соединение 1-битных ЗУ, обеспечивающее образование 8-битных ячеек (соединения АДРЕС И УПРАВЛЕНИЕ одинаковы для всех элементов памяти)

Таблица 7.1. Выводы модулей памяти емкостью 1 Кбит различных вариантов исполнения

Организация	Адресные выводы	Информационные выводы	Всего	
			Раздельный ввод-вывод	Общий ввод-вывод
1K×1	10	1	12	11
512×2	9	2	13	11
256×4	8	4	16	12
128×8	7	8	23	15
64×16	6	16	38	22

управляющих шин, так как по ним необходимо будет передать большее число входных сигналов.

Запоминающие устройства с малой длиной слова имеют преимущества с точки зрения простоты модулей. В табл. 7.1 показано, какое число информационных и адресных выводов имеют модели памяти емкостью 1 Кбит различных вариантов исполнения. Для ЗУ с коротким словом требуется меньше внешних соединений, они имеют меньшие габариты и меньшая часть объема их корпуса используется для внешних соединений. В результате самые большие модули памяти строятся на ОЗУ, имеющих организацию типа 1K×1, 4K×1 или 16K×1.

Только в немногих микропроцессорах имеется достаточное число выводов, необходимых для того, чтобы обеспечить раздельные шины для адресов, входных и выходных данных. В 8-битном процессоре с 16-битной шиной для двух шин данных и адресной шины потребовалось бы 32 вывода. Таким образом, в стандартном корпусе с 40 выводами осталось бы только восемь выводов для подключения питания, источников тактирующих и управляющих сигналов. Обычно для этих целей необходимо как минимум вдвое больше выводов. Корпуса большего объема дороги и занимают больше места на плате.

Обычным методом решения этой проблемы является использование двусторонней шины. При этом работа процессора заметно не усложняется, так как ЦП во время цикла памяти не выполняет одновременно операции чтения и записи. В некоторых ЦП вырабатывается сигнал (УПРАВЛЕНИЕ ШИНОЙ ДАННЫХ), который указывает, в каком состоянии (ввода или вывода) находится шина данных. Этот сигнал можно использовать для управления двунаправленными буферами или драйверами и приемниками, как это показано на рис. 7.16. Чтобы защитить выходы ПЗУ от случайных циклов записи, может понадобиться односторонний буфер (рис. 7.17).

Комбинированные шины данных и адресов усложняют интерфейс между ЦП и памятью. Адрес должен быть зафиксирован внешним сигналом так, чтобы он оставался постоянно доступным, пока шина используется для передачи данных. Комбинированные шины широко применяются в простых процессорах типа Intel 4040, в которых используются очень маленькие модули памяти, и в 16-битных процессорах типа National PACE, в которых нелегко встроить в стандартный корпус с 40 выводами раздельные 16-битные шины.

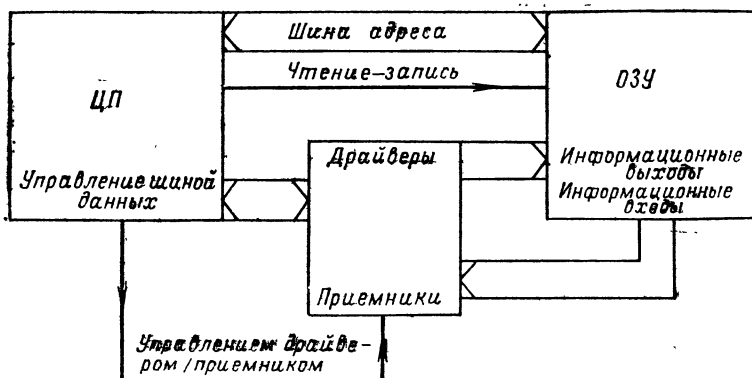


Рис. 7.16. Двусторонний интерфейс между ЦП и ОЗУ при использовании ОЗУ с отдельными входами и выходами (сигнал управления шиной данных передает драйвер-приемник в состояние драйвера во время циклов записи и в состояние приемника во время циклов чтения)

Система должна иметь синхронизирующие сигналы или стробы адресов и данных, которые могут использоваться для записывания адресов и помещения данных на шину. На рис. 7.18 показан интерфейс, использующий стробы адресов и данных. Работа ЭВМ при этом становится очень сложной. Например, в Intel 4040 имеется единственная 4-битная шина, которая используется для передачи данных (4 бит), адресов (12 бит) и команд (8 бит). Цикл команды разделен на восемь тактовых циклов.

Цикл I. Центральный процессор помещает младшие 4 бита СК на шину данных.

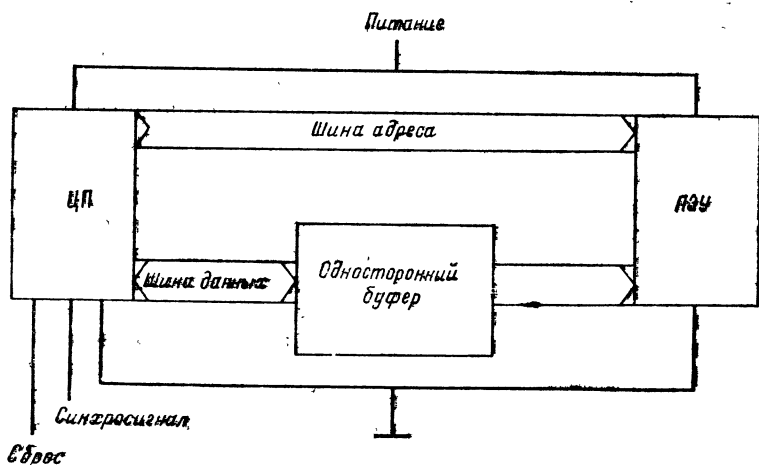


Рис. 7.17. Двусторонний интерфейс между ЦП и ПЗУ с односторонним буфером (односторонний буфер защищает информационные выходы ПЗУ от двусторонней шины)

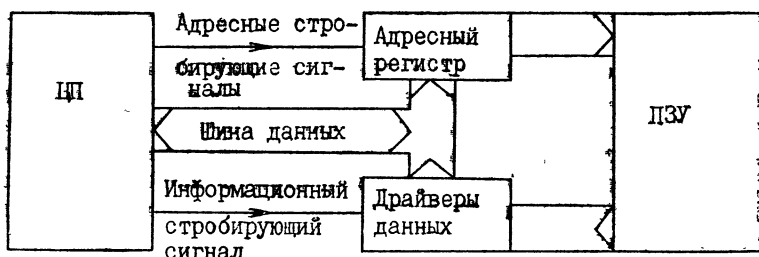


Рис. 7.18. Интерфейс между ЦП и ПЗУ с использованием единственной шины и стробирующих сигналов (адресный стробирующий сигнал помещает адрес в адресный регистр. Информационный стробирующий сигнал разрешает работу драйверов данных и посылает команду в ЦП. Для обоих сигналов могут потребоваться и дополнительные внешние схемы)

Цикл 2. Центральный процессор помещает средние 4 бит СК на шину данных.

Цикл 3. Центральный процессор помещает старшие 4 бит СК на шину данных.

Цикл 4. Постоянное ПЗУ посылает старшие 4 бит команды в ЦП.

Цикл 5. Постоянное ПЗУ посылает младшие 4 бит команды в ЦП.

Циклы 6, 7 и 8 используются для дешифрирования и исполнения команды.

В микро-ЭВМ на базе Intel 4040 требуется специальная память и интерфейсы, которые генерируют синхронизирующие сигналы, запоминают адрес и помещают данные на шину. В большинстве процессоров за счет более широкой шины циклы памяти оказываются проще.

Подключение медленных ЗУ

Во всех ранее рассмотренных в данной главе случаях предполагалось, что ЗУ имеют достаточно малое время доступа и поэтому могут работать с процессором в режиме максимального быстродействия. Поскольку довольно часто встречаются менее быстродействующие ЗУ, в большинстве микропроцессоров предусмотрена возможность их использования.

Медленные ЗУ иногда бывают более дешевыми и более доступными, чем быстродействующие. Вместе с тем в настоящее время разница в стоимости уменьшается и в будущем может совсем исчезнуть. Некоторые типы ЗУ, в частности пользующиеся популярностью ППЗУ, изготовленные по МОП-технологии со стиранием, обычно выпускались в вариантах с малым быстродействием. Теперь положение изменилось в связи с появлением более быстродействующих модификаций этих устройств. Поскольку медленные ЗУ используются все реже, рассмотрим кратко методы реализации интерфейса для них.

Существует много методов замедления работы процессора. При этом должны быть решены следующие проблемы.

1. Необходимо обеспечить согласование требуемого замедления с частотой тактового генератора. Замедление работы процессора может повлечь за собой замену тактового генератора или создание специаль-

ной синхронизирующей схемы, управляемой этим тактовым генератором.

2. Необходимо обеспечить управление ЗУ, работающими с различными скоростями, а также управление циклами, в которых совсем не участвует память. Максимальная пропускная способность достигается, когда работа процессора замедляется только во время тех циклов, в которых фактически осуществляется доступ к медленной памяти.

3. Должны быть минимизированы временные задержки и счетчики числа элементов.

4. При увеличении длительности циклов может потребоваться увеличение длительности других сигналов. К ним относятся сигналы записи и стробирующие сигналы для данных.

Замедление работы процессора может быть осуществлено следующими способами:

1) замедлением тактового генератора процессора. Это самый простой метод, так как для его осуществления достаточно заменить схему тактового генератора. Большинство микропроцессоров может работать с достаточно низкой тактовой частотой, что дает возможность подключать медленные ЗУ. Однако этот метод приводит к нежелательным последствиям, так как замедляется выполнение всех циклов;

2) изменением тактовой частоты только в момент доступа к медленной памяти. Этот метод повышает пропускную способность ЦП за счет усложнения схем. Управляющие сигналы, которые активируют медленные ЗУ, должны запускать схемы, которые понижают тактовую частоту;

3) добавлением дополнительных тактовых периодов в цикл команды при обращении к медленным ЗУ. Во многих процессорах для этой цели предусмотрен особый вход READY (ГОТОВ). Сигнал ГОТОВ должен синхронизироваться с тактовым генератором процессора и может быть задержан в течение нескольких циклов.

Все эти способы очевидны для пользователя. В результате их применения осуществляется необходимая синхронизация с тактовым генератором процессора и обеспечивается автоматическое удлинение требуемых управляющих сигналов.

7.3. СТРУКТУРЫ ШИН

Только незначительное число модулей памяти состоит из одного корпуса памяти или простого набора корпусов памяти, которые имеют одни и те же адресные соединения. Для более сложных модулей памяти требуется структура шин, которая позволяет ЦП пересылать данные в различные части (элементы) памяти или из различных частей. С практической точки зрения нерациональными являются следующие две простые структуры шин:

1) использующая для каждого элемента памяти отдельную шину. Это неэффективно, поскольку в большинстве ЭВМ в каждый момент времени используется одна часть памяти. При раздельных шинах внутри ЦП потребовалось бы определенное их комбинирование, а также огромное число входных и выходных выводов;

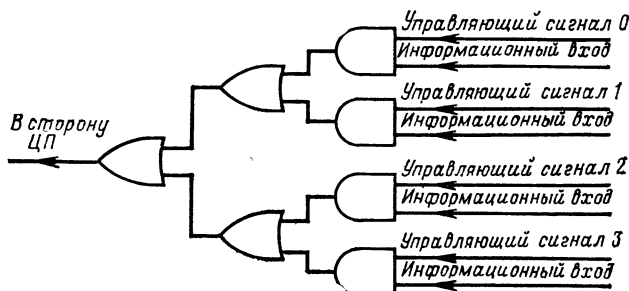


Рис. 7.19. Структура шины при использовании простых клапанов (последовательное включение клапанов с двумя входами дает только 1-битную шину; для реализации параллельной шины потребовались бы параллельно включенные клапаны с общим управляющим сигналом)

2) простое монтажное объединение выходов стандартных схемных элементов. Эта структура нежелательна, так как ее невозможно использовать совместно со стандартными элементами схем. Если с выходов двух элементов на линию подаются противоположные логические уровни (0 и 1), то результат оказывается неопределенным. Если сигналы с выходов двух элементов одновременно попытаются возбудить одну линию, то входы адреса могут быть повреждены в результате превышения максимально допустимого тока. Ситуация, при которой сигналы с выходов двух элементов одновременно пытаются занять шину, называется *конкуренцией при доступе к шине*.

Таким образом, обеспечение интерфейса между ЦП и модулем памяти требует раздельного использования шин. Использование шин может контролироваться с помощью логических схем И и ИЛИ, как это показано на рис. 7.19. Схемы И позволяют с помощью управляющих сигналов заблокировать все входы, кроме того, который нужен. Схемы ИЛИ позволяют управлять шиной любому входу. Данные можно записать в нужный элемент памяти путем подачи на схему совпадения сигнала записи вместе с теми же управляющими сигналами, которые используются в схемах И. Разумеется, циклы вывода данных не могут приводить к конкурентному использованию шин.

Сами управляющие сигналы поступают из последовательно соединенных схем И, как это показано на рис. 7.20. В процессе выполнения цикла действующим может быть только один управляющий сигнал.

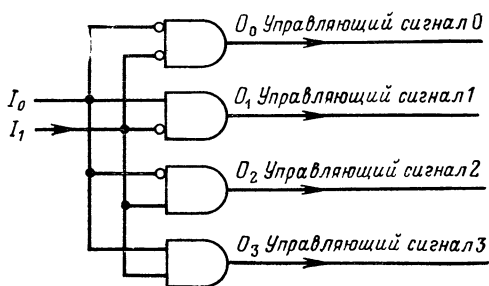


Рис. 7.20. Схема для выработки управляющих сигналов

[четыре схемы И (вместе с несколькими инверторами) принимают входные адресные сигналы и вырабатывают четыре взаимно исключающих управляющих сигнала, которые затем можно использовать при реализации структуры подключения к шине]

В цикле чтения наличие действующих управляющих сигналов приведет к тому, что два различных элемента памяти будут претендовать на шину данных. В цикле записи появление двух действующих управляющих сигналов приведет к тому, что один и тот же элемент данных будет записан в два различных элемента памяти.

Прежде чем рассмотреть методы, позволяющие упростить использование шины, кратко проанализируем критерии оценки качества структур шин.

1. Структура должна обеспечивать обслуживание большого числа (например, от 50 до 200) входов и выходов с использованием небольшого числа буферов и драйверов.

2. Для реализации структуры должно использоваться как можно меньше элементов.

3. Структура должна быть проста с точки зрения расширения или модификации.

4. Память должна иметь правильные фиксированные адреса. Такие адреса могут включать в себя адрес процедуры СБРОС и адреса процедуры обработки прерываний.

5. Структура не должна допускать конкурентного доступа к памяти и обеспечивать корректную синхронизацию памяти.

Для построения структуры шин можно использовать следующие два типа схем средней степени интеграции:

а) дешифраторы, которые активируют необходимые управляющие сигналы в зависимости от входных кодов;

б) селекторы, которые выбирают один из возможных входов для подачи его на выход.

Дешифратор выполняет те же функции, что и схемы И, показанные на рис. 7.20. В табл. 7.2 приведена таблица истинности для дешифратора типа 2-в-4, у которого действующим значением сигналов на выходах является 1. Дешифраторы поставляются в стандартном исполнении (серия 7400) в одном корпусе в следующих вариантах: 2-в-4 (74139, 74155 и 74156), 3-в-8 (74138), 4-в-10 (7442) и 4-в-16 (74154 и 74159). В некоторых из этих устройств действующим значением сигналов на выходах является нуль (недействующие значения всех управляющих сигналов — 1) или имеются специальные входы и выходы, которые упрощают расширение системы. Наиболее общим типом входов являются разрешающие входы. При недействующих значениях сигналов на этих входах значения сигналов на всех выходах также недействующие; при этом дешифраторы могут соединяться последовательно, как это показано на рис. 7.21. Дешифраторы недороги и просты в использовании, имеют различные размеры. Дешифратор вносит дополнительную временную задержку t_{DEC} , которая представляет собой максимальную задержку появления выходного управляющего сигнала после подачи входного сигнала. При последовательном соединении дешифраторов увеличивается время задержки. Кроме того, при наличии двух действующих управляющих сигналов за изменением входов также последует небольшая задержка. Обычно длительность этого промежутка t_{REC} представляет собой максимальное время восстановления, которое затрачивается на переход сигнала из дейст-

Таблица 7.2. Таблицы истинности для дешифратора 2-в-4

Входы		Выходы			
I_0	I_1	O_0	O_1	O_2	O_3
0	0	1	0	0	0
1	0	0	1	0	0
0	1	0	0	1	0
1	1	0	0	0	1

Таблица 7.2а. Таблица истинности для дешифратора 2-в-4, имеющего разрешающий вход

Входы			Выходы			
F	I_0	I_1	O_0	O_1	O_2	O_3
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	1	0	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	1

вующего состояния в недействующее. Для предотвращения конкуренции во время таких периодов используются дополнительные синхронизирующие сигналы.

Если дешифратор имеет также разрешающий вход E , действующим значением сигнала которого является 1, то таблица истинности примет вид, приведенный в табл. 7.2а (здесь X означает, что значение переменной не играет роли и может быть равно 0 или 1).

В модулях памяти дешифраторы генерируют управляющие сигналы в зависимости от содержимого старших линий адресной шины, которые непосредственно не подключены к элементам памяти. Если проектировщик желает подключить максимальное число элементов памяти, то следует дешифровать все адресные линии. Если адресная

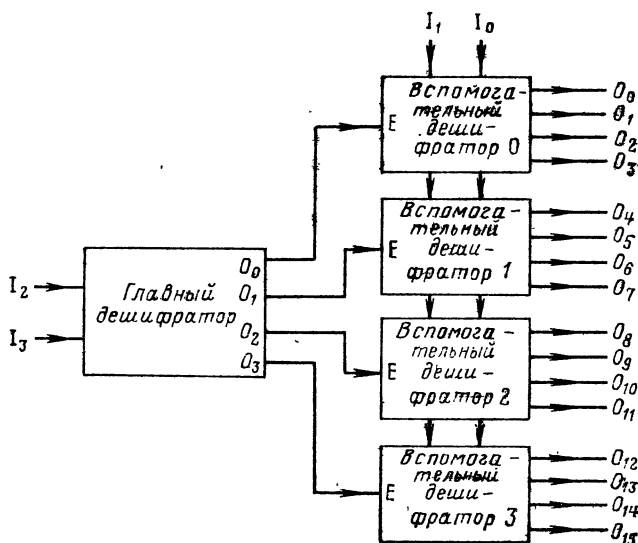


Рис. 7.21. Формирование управляющих сигналов с помощью последовательно включенных дешифраторов

(все вспомогательные дешифраторы имеют одинаковые информационные входы, но в каждый момент времени главный дешифратор разрешает работу только одного из них)

Таблица 7.3. Таблица истинности для селектора «1-из-4»

Селектирующие входы		Информационные входы				Выход	Селектирующие входы		Информационные входы				Выход
S_0	S_1	D_0	D_1	D_2	D_3	O	S_0	S_1	D_0	D_1	D_2	D_3	O
0	0	0	X	X	X	0	0	1	X	X	0	X	0
0	0	1	X	X	X	1	0	1	X	X	1	X	1
1	0	X	0	X	X	0	1	1	X	X	X	0	0
1	0	X	1	X	X	1	1	1	X	X	X	1	1

линия не участвует в дешифрировании, ее значение не будет влиять на процесс выборки элемента памяти. Адреса, которые отличаются только значением сигнала в упомянутой линии, будут порождать одни и те же управляющие сигналы. Чтобы избежать конкуренции при доступе к шине, все такие адреса должны указывать на одну и ту же ячейку памяти. Использование подобного метода затрудняет расширение памяти.

Селектор сочетает в себе свойства дешифратора и устройства подключения к шине. Сигналы, поступающие на входы селекции, осуществляют выбор того информационного сигнала, который должен появиться на выходе, как это показано на рис. 7.4. Входы селекции могут быть подключены непосредственно к адресным линиям. В табл. 7.3 приведена таблица истинности для селектора типа 1-из-4. Селекторы поставляются в стандартном исполнении (серия 7400) в следующих вариантах: 1-из-2 (74157, 74158, 74257 и 74298), 1-из-4 (74153 и 74253), 1-из-8 (74151, 74152 или 74251) и 1-из-16 (74150).

Селекторы могут иметь выходы с действующим значением 0, а также специальные входы или выходы, которые облегчают расширение системы. Наличие единственного разрешающего входа и схемы ИЛИ позволяет объединить селекторы, например так, как это показано на рис. 7.22. В приведенной схеме адресная линия A_2 определяет, какой селектор будет действующим. Если $A_2 = 0$, действует селектор 0, если $A_2 = 1$, действует селектор 1. Значение недействующего выходного сигнала селектора всегда будет равно 0, и он не будет влиять на выходной сигнал, схемы ИЛИ.

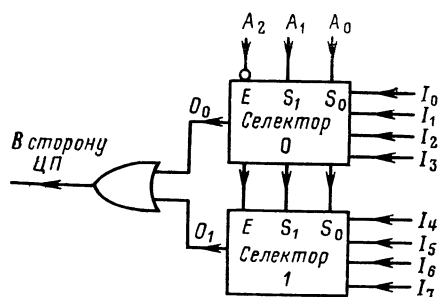


Рис. 7.22. Использование селекторов для образования шины

Действующим значением разрешающего сигнала считается 1. Более широкая шина требует параллельного соединения селекторов с общими входами селекции и разрешения. Использование селекторов, так же как и дешифраторов, приводит к появлению задержек передачи и времен восстановления, которые влияют на временные характеристики системы. Однако появление дополнительных селекто-

ров не увеличивает задержку передачи, так как данные проходят только через один селектор.

При использовании дешифраторов и селекторов, выполненных по ТТЛ-технологии, требуется решить следующие проблемы.

1. Дополнительные временные задержки могут значительно замедлить работу системы. Времена задержек можно уменьшить, используя схемы Шоттки, выполненные по ТТЛ-технологии. Такие схемы стоят дороже и потребляют больше энергии, но работают быстрее стандартных устройств, выполненных по ТТЛ-технологии.

2. Нецелесообразно одновременно подключать к одной шине устройства, выполненные по МОП- и ТТЛ-технологии. Выходы устройств на МОП-структурах, даже если они имеют такой же уровень напряжения, не в состоянии управлять резистивной нагрузкой ТТЛ-схем. Интерфейс между элементами, выполненными по МОП- и ТТЛ-технологии, может потребовать использования преобразователей уровня, буферов и драйверов. Кроме того, для элементов, выполненных по МОП- и ТТЛ-технологии, потребуются отдельные источники питания и тактовые генераторы.

3. ТТЛ-схемы рассеивают гораздо большую мощность, чем МОП-схемы, а это может привести к необходимости обеспечить средства теплоотвода и более мощные источники питания. Потребление энергии может быть снижено благодаря использованию маломощных приборов Шоттки, выполненных по ТТЛ-технологии. Они дороже стандартных ТТЛ-схем, но имеют такое же быстродействие и потребляют меньшую мощность.

При разрешении первой и третьей проблем перед проектировщиком встают противоречивые задачи. Решением этих проблем является использование буферов и драйверов Шоттки, так как работа системы критична по отношению к их временам задержек, а также маломощных дешифраторов и селекторов Шоттки, поскольку времена их задержек не являются критичными. При необходимости для снижения стоимости устройства можно использовать стандартные ТТЛ-схемы.

Выходы с открытым коллектором

Структуры шин можно упростить, если использовать специальные выходы, которые можно объединять без помощи логических схем. О таких выходах говорят, что они допускают «монтажное ИЛИ». Этот термин является не совсем удачным, поскольку фактически подобное соединение не выполняет логической функции ИЛИ, и предполагают, что в каждый момент времени действующим будет только один выход. Смысл сказанного станет яснее, когда перейдем к рассмотрению выходов устройств с открытым коллектором и тремя устойчивыми состояниями, которые обычно используются для реализации шин в ТТЛ- и МОП-системах.

В схемах с открытым коллектором отсутствует окончательный нагрузочный резистор, который имеется в обычной схеме, выполненной по ТТЛ-технологии, и поэтому действующее значение сигналов на их выходах равно 0, а не 1. Схемы с открытым коллектором могут быть объе-

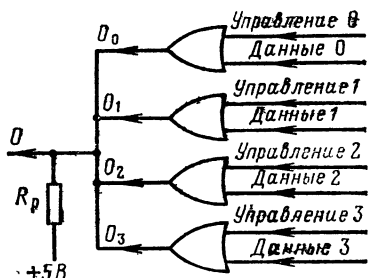


Рис. 7.23. Объединение схем с открытым коллектором (сигнал на общем выходе имеет значение 1 тогда и только тогда, когда выходной сигнал каждой схемы ИЛИ равен 1. При сигнале на выходе любой из схем ИЛИ, равном 0, сигнал на выходе равен 0 R_p — нагрузочный резистор)

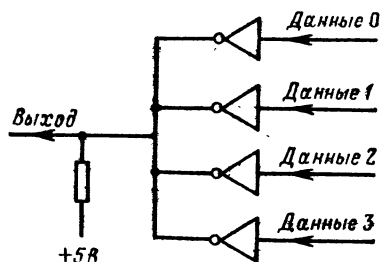


Рис. 7.24. Индикатор пуска, реализованный на инверторах с открытым коллектором

динены по схеме монтажного ИЛИ с использованием нагрузочного резистора, как показано на рис. 7.23. Выходной сигнал (табл. 7.4) равен 0, если сигнал на любом из объединенных выходов равен 0, и равен 1, если сигналы на всех объединенных выходах равны 1.

Схема на рис. 7.24 иллюстрирует, как использовать элементы с открытым коллектором для реализации индикатора нуля. Выходной сигнал равен 1, когда сигналы на всех линиях данных равны 0. Если хотя бы на одной из линий данных содержится сигнал, равный 1, то появление сигнала, равного 0, на выходе соответствующего инвертора с открытым коллектором приводит к появлению 0 на общем выходе. С помощью данной схемы можно формировать значение признака НУЛЬ, описанного в гл. 3.

В основном варианте структуры шин на схемах с открытым коллектором над управляющими и информационными сигналами выполняется логическая операция ИЛИ (см. рис. 7.23). Таблица истинности для этой структуры приведена в табл. 7.5. Необходимые управляющие сигналы можно сформировать с помощью дешифратора, действующее значение выходного сигнала которого равно 0.

Поступление на вход управляющего сигнала, равного 1, приводит к тому, что на выходе соответствующей схемы ИЛИ появляется 1 (независимо от значения информационного сигнала) и в результате общий выходной сигнал не меняется. При низком уровне управляющего сиг-

Таблица 7.4. Таблица истинности для выхода 0

Входы				Выход	Входы				Выход
O_0	O_1	O_2	O_3	0	O_0	O_1	O_2	O_3	0
0	X	X	X	0	X	X	X	0	0
X	0	X	X	0	1	1	1	1	1
X	X	0	X	0					

Таблица 7.5. Таблица истинности для схемы, изображенной на рис. 7.23 *

Управление				Данные				Выход
C ₀	C ₁	C ₂	C ₃	D ₀	D ₁	D ₂	D ₃	O
1	1	1	1	X	X	X	X	1
0	1	1	1	0	X	X	X	0
0	1	1	1	1	X	X	X	1
1	0	1	1	X	0	X	X	0
1	0	1	1	X	1	X	X	1
1	1	0	1	X	X	0	X	0
1	1	0	1	X	X	1	X	1
1	1	1	0	X	X	X	0	0
1	1	1	0	X	X	X	1	1

* Выходной сигнал равен 1, если все сигналы управления равны 0; если сигнал управления равен 1, то значение выходного сигнала равно значению соответствующего выходного сигнала.

нала сигнал на выходе схемы ИЛИ равен информационному. Таким образом, значение общего выходного сигнала зависит от значения сигнала на информационном входе.

Шины на схемах с открытым коллектором могут использоваться с обычными дешифраторами, выполненными по ТТЛ-технологии. Эти шины имеют очень простую структуру и могут быть легко расширены или модифицированы. Следует обратить внимание на то, что шины на схемах с открытым коллектором активны при нулевом значении сигнала (используют отрицательную логику). Поэтому в качестве управляющих элементов используются схемы ИЛИ, а не схемы И, показанные на рис. 7.19.

Тем не менее шины на схемах с открытым коллектором не нашли широкого использования в микро-ЭВМ по следующим причинам.

1. Без применения буферизации оказывается возможным объединить сравнительно небольшое число схем с открытым коллектором. Каждая схема с открытым коллектором, имеющая на выходе сигнал, равный 1, потребляет некоторый ток (обычно не более 0,25 мА). Если не используется буферизация, возникающие при этом процессы быстро приводят к снижению качества шины. Поскольку стандартная схема, выполненная по ТТЛ-технологии, обычно обеспечивает управляющий ток около 20 мА, с помощью монтажного ИЛИ можно объединить сравнительно немного (не более 10—20) схем с открытым коллектором.

2. Для управления шиной требуются логические схемы. Наличие этих схем приводит к увеличению числа элементов в системе; кроме того, эти схемы занимают место на плате.

3. Нагрузочный резистор занимает место на плате и потребляет ток, когда шина находится в рабочем состоянии. Появление этого тока (обычно около 1 мА) приводит к еще большему уменьшению уровня сигнала в шине.

Выходы с тремя устойчивыми состояниями

В ЭВМ с большим числом входов схемы с тремя устойчивыми состояниями [высокий уровень выходного сигнала, низкий уровень выходного сигнала или «цепь разомкнута» (большой импеданс)] вытеснили схемы с открытым коллектором. В состоянии «цепь разомкнута» через схему протекает очень маленький ток; при этом схемы могут быть объединены друг с другом без существенного влияния на общий выходной сигнал. Из буферов с тремя состояниями и нулевыми действующими значениями разрешающих сигналов можно образовать шину, как это показано на рис. 7.25. Недействующие (единичные) значения управляющих сигналов переводят выходы в состояние «цепь разомкнута» (табл. 7.6) и при этом значение сигнала на общем выходе не изменяется.

Шины на схемах с тремя состояниями просты с точки зрения расширения или модификации и имеют простую структуру. Кроме того, они имеют следующие достоинства.

1. Не прибегая к буферизации, можно объединять большое число выходов схем с тремя состояниями. Схема с тремя состояниями в состоянии «цепь разомкнута» потребляет ток значением не более 0,4 мА. Это значение существенно меньше того, которое соответствует недействующему (единичному) состоянию схемы с открытым коллектором. В результате в схемах с тремя состояниями можно объединять гораздо больше выходов, чем в схемах с открытым коллектором.

2. Устройства на СИС и БИС могут иметь в своем составе буфера с тремя состояниями. Для «трестабильного разрешения» необходимо иметь один дополнительный вывод. Если имеются дополнительные выводы, можно использовать несколько разрешающих входов (которые объединяются в модуле с помощью схемы совпадения). Чтобы осуществить правильное дешифрирование, часто необходимо использовать всего лишь несколько инверторов.

3. Нет необходимости использовать нагрузочный резистор, за исключением того случая, когда может возникнуть ситуация, при которой все выходы имеют состояние «цепь разомкнута».

4. Шины на схемах с тремя состояниями могут управляться различными устройствами. Например, шины на элементах с тремя состояниями, соединяющие процессор и память, могут быть выведены из

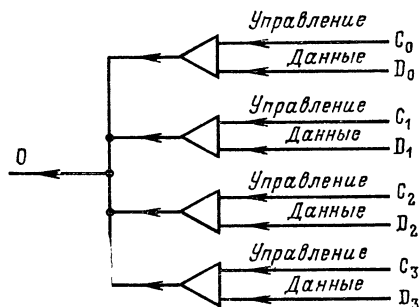


Рис. 7.25. Объединение схем, имеющих три состояния

[все схемы с недействующими значениями управляющих сигналов находятся в третьем состоянии («цепь разомкнута») и не влияют на общий выходной сигнал]

Таблица 7.6. Таблица истинности для шины с тремя состояниями, показанной на рис. 7.25

Управление				Данные				Выход
C ₀	C ₁	C ₂	C ₃	D ₀	D ₁	D ₂	D ₃	0
1	1	1	1	X	X	X	X	?
0	1	1	1	0	X	X	X	0
0	1	1	1	1	X	X	X	1
1	0	1	1	X	0	X	X	0
1	0	1	1	X	1	X	X	1
1	1	0	1	X	X	0	X	0
1	1	0	1	X	X	1	X	1
1	1	1	0	X	X	X	0	0
1	1	1	0	X	X	X	1	1

Примечание. Если сигналы управления являются недействующими, то результирующий выходной сигнал имеет состояние «цепь разомкнута». При определенном значении нагрузочного резистора (1 кОм) выходной сигнал, соответствующий состоянию «цепь разомкнута», принимает значение логической 1.

рабочего состояния путем переключения выходов процессора в состояние «цепь разомкнута». В этом случае внешний контроллер может использовать эти же шины для прямого доступа к памяти.

Разумеется, шины на схемах с тремя состояниями имеют и недостатки.

1. Устройства, имеющие обычные выходы, должны подключаться к таким шинам через буфера с тремя состояниями. Это увеличивает содержимое счетчика числа элементов. Вместе с тем при использовании выходов с тремя состояниями становится доступным большее число устройств.

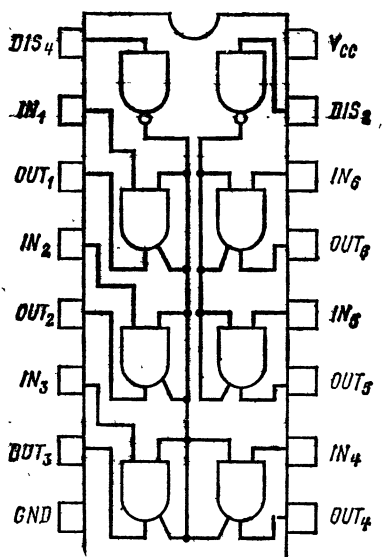
2. Устройства с тремя состояниями вносят в систему новые временные задержки: t_{eo} — максимальное время задержки от момента появления сигнала разрешения до момента появления сигнала на выходе; t_{dis} — максимальное время от момента снятия сигнала разрешения до момента переключения на выходе в состояние «цепь разомкнута».

Появление этих задержек усложняет процесс проектирования модулей памяти.

Буфера и драйверы с тремя состояниями

Часто в микро-ЭВМ необходимо использовать буфера, драйверы и приемники. К числу распространенных устройств такого рода относятся буфер с тремя состояниями типа 8T97 и приемопередатчик шины с тремя состояниями типа 8T28 фирмы Signetics на элементах Шоттки, выполненных по ТТЛ-технологии. Имеются и другие изготовители, поставляющие аналогичные устройства: National — DM 8097, Intel — 8216 и Advanced Micro Devices — AM2915.

На рис. 7.26 приведена структурная схема и таблица истинности для буфера типа 8T97. Каждое устройство типа 8T97 имеет шесть информационных линий, четыре из которых контролируются входом



8T97			
Отмена DIS_4	Вход DIS_2	Вход	Выход
0	0	0	0
0	0	1	1
X	1	X	H-z*
1	X	X	H-z*

*H-z означает большое сопротивление для разомкнутой цепи

Рис. 7.26. Структурная схема и таблица истинности для буфера типа 8T97, имеющего три состояния

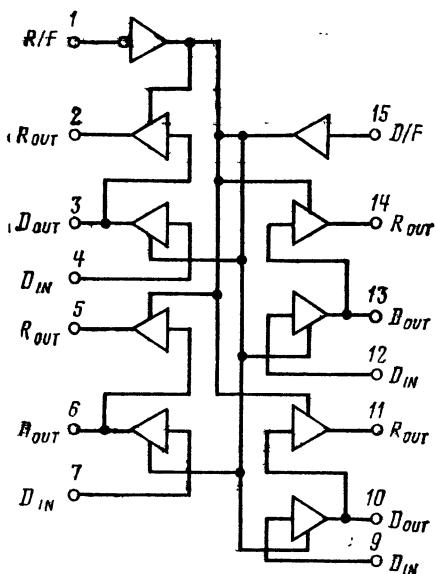


Рис. 7.27. Схема приемопередающего элемента шины типа 8T28 (8T28 — неинвертирующий выход с тремя состояниями)

DIS_4 (с действующим значением сигнала на входе, равным 0) и два — входом DIS_2 (с действующими значениями сигналов, равными 0). Комбинируя различные наборы линий с одним и тем же управляющим сигналом, можно создавать шины различной ширины. Устройство потребляет ток не более 0,4 мА и вырабатывает управляющий ток значением не менее 40 мА. Таким образом, устройство потребляет существенно меньший ток,

чем стандартная нагрузка устройств ТТЛ (1,6 мА), и может управлять работой не менее 25 элементов, выполненных по ТТЛ-технологии. Оно вносит в схемы очень небольшую дополнительную задержку: максимальное время задержки данных составляет 13 нс, максимальное время выдачи выходного сигнала 16 нс и максимальная задержка от момента появления сигнала разрешения до момента появления сигнала по выходе составляет 25 нс.

На рис. 7.27 приведена структурная схема устройства с тремя состояниями типа 8T28, представляющего собой счетверенный приемопередатчик шины. Каждое устройство 8T28 имеет четыре информационные входные линии (DIN), четыре выходные линии приемника ($R_{\text{вх}}$)

и четыре двусторонние драйверные линии ($D_{\text{вых}}$). Сигнал «разрешение работы драйвера» D/E (с действующим значением 1) переключает драйверные линии в состояние «цепь разомкнута». Под действием сигнала «разрешение работы приемника» R/E (с действующим значением 1) линии приемника переходят в состояние «цепь разомкнута».

Устройство 8T28 допускает подключение двусторонней шины к двум односторонним (например, информационную шину ЦП к информационной входной и информационной выходной шинам) путем установки такого значения сигнала «разрешение работы приемника», которое является инверсным значению сигнала «разрешение работы драйвера». В результате двусторонняя шина не будет использоваться одновременно в двух направлениях.

Устройство 8T28 потребляет ток не более 0,2 мА и выдает управляющий ток не менее 30 мА. Оно может работать при емкости нагрузки шины до 300 пФ, что очень важно, так как МОП-схемы создают емкостную нагрузку на линию (обычно 10 пФ на одну схему). Времена переключения устройств типа 8T28 очень малы. Максимальное время задержки данных составляет 17 нс, максимальное время выдачи выходного сигнала 23 нс и максимальное время задержки от момента появления сигнала разрешения до момента появления сигнала на выходе равно 28 нс.

7.4. ПРОЕКТИРОВАНИЕ МОДУЛЕЙ ПАМЯТИ С ТРЕМЯ СОСТОЯНИЯМИ

Простейший модуль памяти с тремя состояниями показан на рис. 7.28. В приведенной схеме выходы дешифратора соединены с разрешающими входами модулей памяти. С помощью неиспользован-

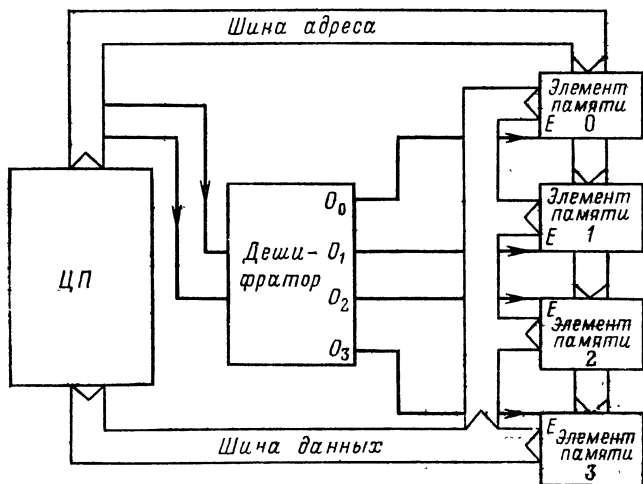


Рис. 7.28. Простой модуль памяти с тремя состояниями (дешифратор использует две старшие адресные линии для разрешения работы с одним из четырех элементов памяти. Младшие адресные линии подключены непосредственно ко всем элементам памяти)

ных адресных линий можно отключить дешифраторы или подключить их последовательно. Такую систему легко расширить. Однако в больших системах может потребоваться много дешифраторов, а также буферов и драйверов. Тремя состояниями должна обладать только информационная шина, поскольку в простейшей системе это единственная шина, имеющая более одного входа. За исключением случаев, когда к памяти также должен иметь доступ некоторый внешний контроллер (см. гл. 9), адресная и управляющая шины не должны иметь три состояния.

Дешифратор можно исключить, если ограничить значения адресов или использовать модули памяти с несколькими разрешающими входами. Простым способом ограничения адресов является использование каждой доступной адресной линии в качестве разрешающего входа для другого элемента памяти. В результате число элементов памяти, которые могут быть подключены с помощью k адресных линий, уменьшается с 2^k до k . На рис. 7.29 показана структура модуля памяти, реализованного этим методом, получившим название линейной выборки. Метод линейной выборки позволяет обойтись без дешифраторов, но приводит к тому, что адресное пространство перестает быть непрерывным. При этом методе недопустимыми являются адреса, у которых име-

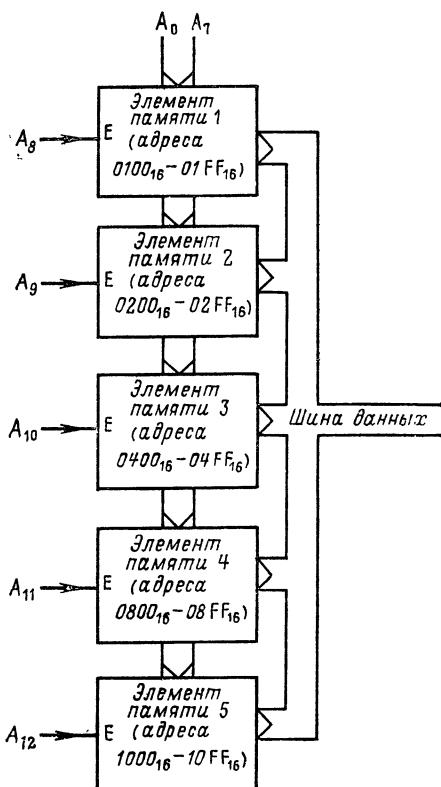


Рис. 7.29. Блок памяти с линейной выборкой (на элементах памяти емкостью 256 слов)

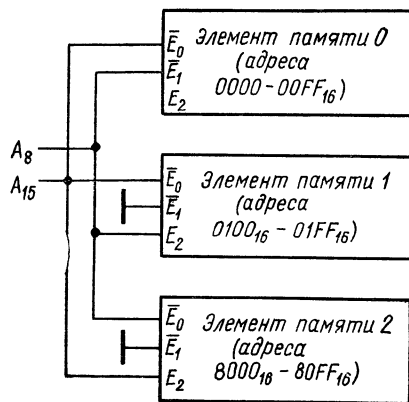


Рис. 7.30. Блок памяти, использующий разрешающие входы (элементы памяти емкостью 256 слов)

ется более одного единичного бита в линиях выборки, так как каждый единичный бит активирует один элемент памяти. Наличие всех нулей на линиях выборки также недопустимо, так как подобные адреса не будут активировать никаких элементов памяти.

Корпуса памяти с несколькими разрешающими входами особенно удобны, если сигналы на некоторых разрешающих входах имеют действующее значение, равное 1, а на других — 0. На рис. 7.30 показан модуль памяти без дешифраторов, в котором для выборки одного из трех корпусов используются один разрешающий вход с действующим значением сигнала, равным 1, и два разрешающих входа с действующим значением сигнала, равным 0. В данном случае емкость памяти не так ограничена, как при линейной выборке. Кроме того, адресное пространство может быть непрерывным. Этот метод имеет и свои недостатки; трудно расширять модуль памяти, так как адресные линии дешифрируются не полностью; необходимо иметь специальные корпуса памяти большего размера (из-за дополнительных разрешающих входов), кроме того, во избежание конфликтов проектировщик должен очень тщательно продумать соединения.

Этот метод особенно удобен, если для размещения программы СБРОС или программ обработки прерываний ЦП использует фиксированные адреса памяти, отличные от нуля. Заметим, что модуль памяти, показанный на рис. 7.30, содержит как старшие, так и младшие адреса памяти. Никаких логических схем или дешифраторов не требуется. Адреса, у которых $A_{15} = A_8 = 0$, находятся в элементе памяти 0, адреса, у которых $A_{15} = 0$ и $A_8 = 1$, находятся в элементе памяти 1, а те, у которых $A_{15} = 1$ и $A_8 = 0$, — в элементе памяти 2. Чтобы допустить использование адресов с $A_{15} = A_8 = 1$, потребуется еще один разрешающий вход с действующим значением сигнала, равным 1, или один инвертор.

Временные характеристики ЗУ с тремя состояниями

Запоминающие устройства с тремя состояниями имеют такие же временные характеристики, как и другие ЗУ, за исключением задержек для разрешения и снятия разрешения. Разрешение не оказывает влияния на цикл чтения, если система дешифрирования не задерживает сигнал разрешения слишком долго. Максимальное время доступа определяется либо максимальным временем доступа от момента появления адреса, либо максимальным временем доступа от появления разрешения. Определяющим является то, какое из двух времен наступит в цикле позднее. До тех пор, пока время задержки разрешения не превышает значения разности времен доступа, она не оказывает влияния на максимальное время доступа. Проектировщик должен следить за тем, чтобы максимальная задержка дешифрирования незначительно превысила упомянутую разность. Благодаря использованию маломощных дешифраторов Шоттки обеспечивается достаточно высокое быстродействие и уменьшается мощность рассеяния.

Задержка отмены разрешения приводит к появлению двух управляющих сигналов, которые являются одновременно действующими в

течение некоторого времени после каждой смены адреса. Старый и новый адреса будут конкурировать, если в это время не отменено появление сигнала на всех выходах. В большинстве микро-ЭВМ предусматривается неактивный период, в течение которого адреса могут смениться без конкуренции. Сигнал активной фазы может использоваться для задействования всех элементов памяти; он является действующим только в течение некоторой части цикла, во время которого может происходить обмен данными.

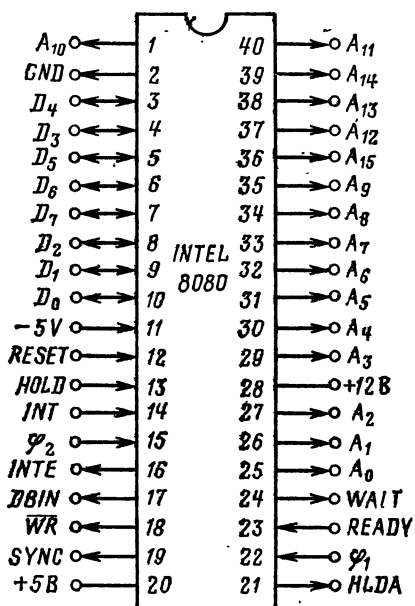
7.5. МОДУЛИ ПАМЯТИ ДЛЯ КОНКРЕТНЫХ МИКРОПРОЦЕССОРОВ

В этом параграфе описываются блоки памяти и циклы чтения и записи в микропроцессорах Intel 8080 и Motorola 6800, рассматриваются управляющие сигналы, временные характеристики, адресация и синхронизация этих процессоров.

Структура управления МП Intel 8080

На рис. 7.31 показано расположение выводов микропроцессора Intel 8080. Интерес представляют следующие сигналы:

- 1) двухфазного тактового генератора, выполненного по МОП-технологии (входы Φ_1 и Φ_2);
- 2) 16-битной адресной шины с тремя состояниями (A_0 — A_{15});
- 3) 8-битной двусторонней информационной шины с тремя состояниями (D_0 — D_7);
- 4) сигнал RESET (сброс) (вывод 12). Этот сигнал длительностью три периода обнуляет счетчик адреса;



- 5) синхронизирующий сигнал SYNC (вывод 19), идентифицирующий начало каждого цикла доступа к памяти;

- 6) сигнал записи \overline{WR} (вывод 18), действующий (0), когда процессор посылает данные на информационную шину;

- 7) сигнал DBIN, определяющий направление передачи данных по информационной шине (вывод 17). Это тот сигнал активной фазы, который используется для исключения конкуренции при доступе к шине во время смены адреса;

- 8) выходной сигнал WAIT (ожидание) (вывод 24) и входной сигнал READY (готов) (вывод 23), используемые для того, чтобы доба-

Рис. 7.31. Расположение выводов микропроцессора Intel 8080

Таблица 7.7. Слово состояния МП Intel 8080

Символическое обозначение	Бит шины данных	Смысл сообщаемой информации
INTA *	D ₀	Сигнал подтверждения запроса на прерывание. Сигнал должен быть использован для подачи команды РЕСТАРТ, когда сигнал DBIN является действующим
\overline{WO}	D ₁	Указывает, что в текущем машинном цикле будет выполняться операция записи в память или вывод ($\overline{WO}=0$). В противном случае будет выполняться операция чтения из памяти или ввод
STACK	D ₂	Указывает, что в адресной шине находится адрес, взятый из указателя стека
HLTA	D ₃	Сигнал подтверждения команды HALT
OUT	D ₄	Указывает, что в шине адреса находится адрес выходного устройства и что в шине данных при действующем значении сигнала \overline{WR} будут находиться выводимые данные
M _I	D ₅	Сообщает, что ЦП находится в цикле выборки первого байта команды
INP *	D ₆	Указывает, что в шине адреса содержится адрес устройства ввода и при действующем значении сигнала DBIN входные данные должны быть помещены на шину данных
MEMR *	D ₇	Указывает, что шина данных будет использоваться для чтения данных из памяти

* Эти 3 бит слова состояния могут быть использованы для управления потоком данных, поступающих на шину данных МП Intel 8080.

Примечание. В МП Intel 8080 для выполнения команд требуется от одного до пяти машинных циклов. В начале каждого машинного цикла (во время действия сигнала SYNC) МП Intel 8080 посылает на шину данных 8-битное слово состояния. Данная таблица поясняет значение бит слова состояния.

Таблица 7.8. Структура слова состояния МП Intel 8080

Бит шины данных	Информация о состоянии	Выборка команды	Чтение из памяти	Запись в память	Чтение из стека	Запись в стек	Ввод	Вывод	Подтверждение прерывания	Подтверждение останова	Подтверждение прерывания во время останова
		1	2	3	4	5	6	7	8	9	10
D ₀	INTA	0	0	0	0	0	0	0	1	0	1
D ₁	\overline{WO}	1	1	0	1	0	1	0	1	1	1
D ₂	STACK	0	0	0	1	1	0	0	0	0	0
D ₃	HLTA	0	0	0	0	0	0	0	0	1	1
D ₄	OUT	0	0	0	0	0	0	1	0	0	0
D ₅	M _I	1	0	0	0	0	0	0	1	0	1
D ₆	INP	0	0	0	0	0	1	0	0	0	0
D ₇	MEMR	1	1	0	1	0	0	0	0	1	0

Примечание. Цифры 1—10 — номера слова состояния.

вить к основному машинному циклу дополнительные тактовые периоды при обращении к медленным ЗУ.

В МП Intel 8080 имеются и другие сигналы состояния, которые на рис. 7.31 не показаны, так как для них не зарезервированы специальные выводы. Центральный процессор посылает эти сигналы состояния в начале каждого машинного цикла на информационную шину. Они должны быть зафиксированы внутри, если необходимо, чтобы они были доступны в то время, пока информационная шина используется для передачи данных или команд. Эти сигналы приведены в табл. 7.7, а в табл. 7.8 показана их связь с типом выполняемого машинного цикла.

Временные характеристики МП Intel 8080

Синхронизирующая серия в МП Intel 8080 состоит из двух сигналов: короткого ϕ_1 (стандартной длительностью 100 нс) и следующего за ним длинного ϕ_2 (стандартной длительностью 250 нс). В стандартном варианте МП Intel 8080 тактовая частота равна 2 МГц, хотя в некоторых версиях он может работать вдвое быстрее. Обычно для генерации тактовых сигналов, управляющих работой МОП- и ТТЛ-схем, для синхронизации сигналов ГОТОВ и СБРОС, а также для выработки стробирующего сигнала STSTB (СТРОБ СЛОВА СОСТОЯНИЯ), необходимого для запоминания данных о состоянии, полученных из шины данных, используют специальный модуль Intel 8224 (рис. 7.32).

Работа процессора может рассматриваться в терминах следующих временных интервалов:

командный цикл, представляющий собой время, необходимое для выборки, дешифрирования и выполнения команды;

машинный цикл — это время, необходимое для передачи данных в память или в порты ввода-вывода или обратно;

такт, представляющий собой промежуток времени между двумя соседними переходами тактового сигнала ϕ_1 из отрицательного состояния в положительное. Другими словами, такт совпадает с одним периодом тактового генератора.

Машинный цикл длится от трех до пяти тактов. Три такта используются для осуществления доступа к модулю памяти или ввода-вывода. Остальные два такта (если они присутствуют) используются для декодирования и выполнения команды.

Если через M и T обозначить соответственно машинный цикл и такт, то работа процессора может быть описана следующим образом.

1. Во время такта T_1 цикла M_1 ЦП помещает содержимое СК на адресную шину, а информацию о состоянии (нужную в цикле выборки команды) на шину данных. Все эти операции выполняются непосредственно за фронтом тактового сигнала ϕ_2 . Первая часть цикла используется для завершения предыдущего цикла с целью предотвращения возможного наложения. Максимальное время задержки адреса составляет 200 нс, а максимальное время задержки данных — 220 нс.

2. Во время такта T_2 цикла M_1 информация о состоянии фиксируется внутри процессора и информационная шина либо используется для

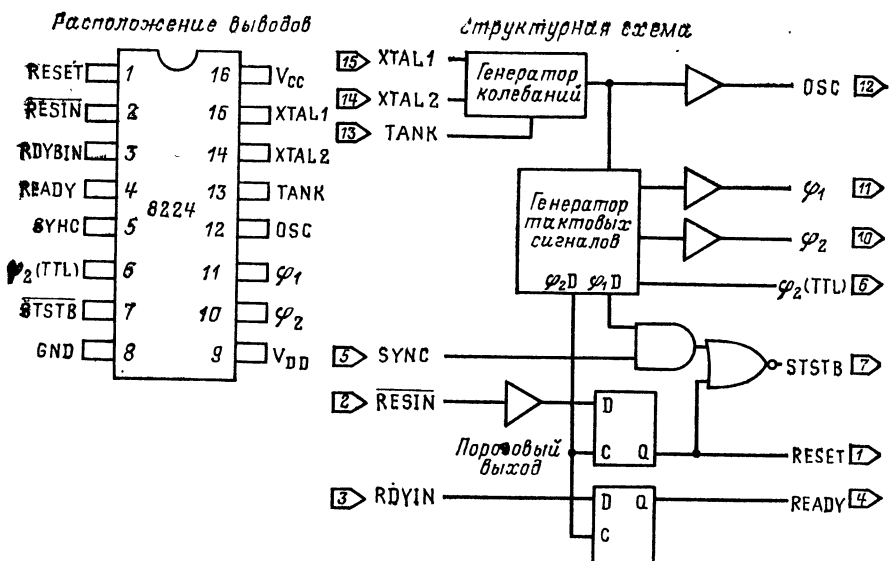


Рис. 7.32. Описание тактового генератора Intel 8224 RESIN — сигнал опроса (вход); RESET — сигнал сброса (выход); RDYIN — сигнал готовности (вход); READY — сигнал готовности (выход); SYNC — сигнал синхронизации (выход); STSTB — строб состояния (действующее значение 0); φ_1 , φ_2 — тактовые сигналы МП 8080; XTAL1, XTAL2 — кварцевый генератор; TANK — дополнительный генератор; OSC — выход генератора колебаний; φ_2 (TTL) — тактовый сигнал φ_2 для TTL-схем; $V_{CC}+5$ В; $V_{DD}+12$ В; GND — 0

вывода данных, либо переводится в состояние ожидания ввода данных (DBIN=1). Все эти действия выполняются непосредственно за фронтом тактового сигнала φ_2 , за исключением фиксации состояния, которая происходит по срезу сигнала φ_1 .

3. Во время такта T_3 цикла M_1 осуществляется либо передача данных в процессор (цикл чтения), либо формирование сигнала записи \overline{WR} . Чтобы данные были приняты корректно, этот сигнал должен удовлетворять временам установки как во время сигнала φ_1 , так и во время сигнала φ_2 . Минимальные времена установки составляют 30 нс до среза φ_1 и 130 нс до среза φ_2 . Сигнал записи прекращается примерно через 130 нс после смены данных и адреса относительно фронта φ_2 в следующем цикле.

4. Такты T_4 и T_5 используются для того, чтобы дешифровать и выполнить команды. Эти такты обычно присутствуют во время цикла выборки команды. Различие между командами, требующими четыре или пять тактов, определяется при внутреннем дешифрировании.

Выполнение команды в МП Intel 8080

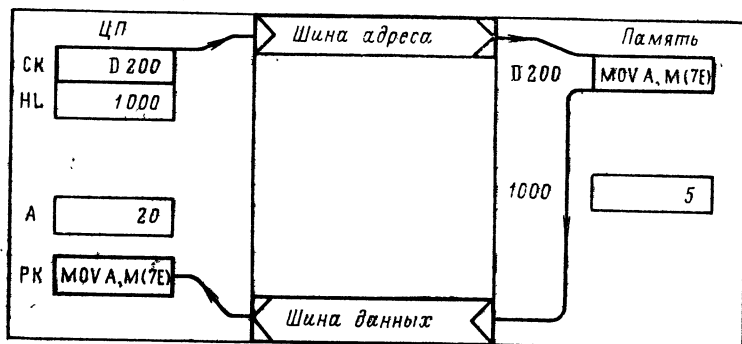
Цикл выборки и дешифрирования каждой команды занимает четыре или пять тактов генератора. Команды, для выполнения которых не требуется дополнительных обращений к памяти или внутренних

операций, выполняются в течение этого промежутка времени. Другие команды требуют дополнительных обращений к памяти (на каждое из которых затрачивается три тактовых периода) для выборки адресов или данных из памяти и портов ввода или для передачи данных в память или порты вывода.

Пример 1. MOV A, M (пересылка данных из ячейки памяти, адресуемой регистрами H и L, в аккумулятор).

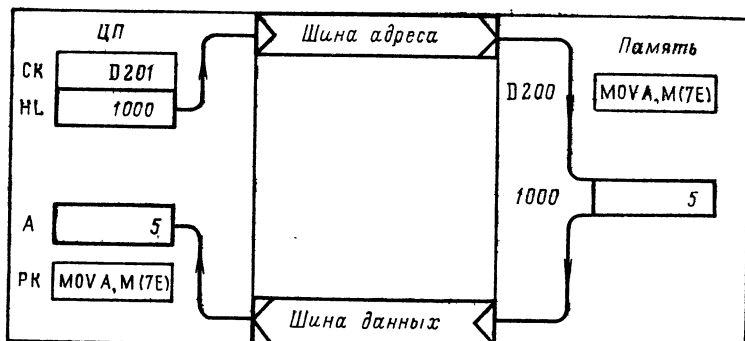
Процесс выполнения команды MOV A, M изображен на рис. 7.33. Для выполнения команды требуется два машинных цикла. Во время первого цикла, состоящего из четырех тактов, команда выбирается из памяти, посылается в регистр команд и дешифрируется. Во время второго цикла, состоящего из трех тактов, данные с помощью адресных регистров H и L выбираются из памяти и посылаются в аккумулятор. Выполнение всей команды происходит за семь тактовых циклов. В

(шина адреса) = (счетчик команд)
(шина данных) = MOV A, M (7E₁₆)
(счетчик команд) = (счетчик команд) + 1
Команда декодирована



(шина адреса) = (регистры H и L)
(шина данных) = (регистры H и L)
Команда выполнена
(аккумулятор) = (регистры H и L)

а)



б)

Рис. 7.33. Выполнение команды MOV A, M:

а — цикл 1. Выборка команды; б — цикл 2. Чтение из памяти

первом цикле выбирается команда ($M_1 = 1$, $MEMR = 1$), а во втором читается содержимое памяти ($M = 0$, $MEMR = 1$).

Пример 2. Команда CALL 3050H помещает шестнадцатиричное число 3050 в счетчики команд и сохраняет прежнее содержимое СК в стеке, расположенном в ОЗУ.

Процесс выполнения команды CALL 3050H изображен на рис. 7.34. Для выполнения этой команды требуется пять машинных циклов, так как она включает в себя операции чтения и записи. В первых трех циклах из памяти программ выбирается команда и адрес. Центральный процессор временно запоминает адрес во внутренних регистрах W и Z. В следующих двух циклах содержимое счетчика команд посылается в стек. Указатель стека декрементируется и во время каждого цикла 8 бит СК помещаются в стек (первыми фиксируются старшие 8 бит). Затем ЦП пересылает содержимое регистров W и Z в счетчик команд. После выборки всей команды CALL содержимое СК помещается в стек. В результате в стеке запоминается адрес команды, идущей непосредственно за командой CALL. Вся команда выполняется из 17 тактов, так как выборка команды выполняется за 5 тактов, а следующие затем два цикла чтения памяти и два цикла записи в стек требуют для своего исполнения по три такта каждый.

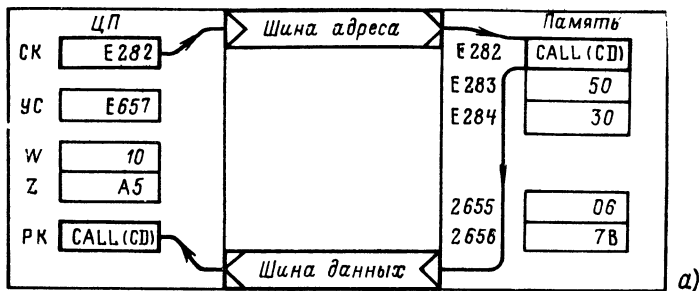
Управление памятью МП Intel 8080

Для управления операциями обмена с памятью в МП Intel 8080 используются сигналы, описанные ранее, а также два производных сигнала. Производными являются сигналы ЧТЕНИЕ ПАМЯТИ (MEMR) и ЗАПИСЬ В ПАМЯТЬ (MEMW). Сигнал ЧТЕНИЕ ПАМЯТИ принимает действующее значение, когда данные читаются из памяти и информационная шина находится в состоянии ВВОД ($DBIN = 1$). Сигнал ЗАПИСЬ В ПАМЯТЬ принимает действующее значение, когда данные записываются в память. Эти сигналы могут быть получены либо от логических схем, либо от контроллера системы 8228 и драйверов шины, показанных на рис. 7.85. Устройство 8228 состоит из регистра состояния, драйвера двусторонней информационной шины и матрицы логических схем.

Сигнал MEMR может выполнять несколько функций.

1. Разрешать открывать тристабильные выходы ПЗУ таким образом, что они будут иметь действующее значение только во время цикла чтения.
2. Разрешать открывать тристабильные выходы из ОЗУ таким образом, что сигналы на этих выходах будут формироваться только во время циклов чтения.
3. Разрешать работу буферов с тремя состояниями, которые соединяют ЦП и блоки памяти.
4. Исключить возникновение конкурирующих запросов на шину во время смены адресов. Работа каждого элемента памяти разрешается отдельно с помощью сигнала MEMR. При смене адресов все эле-

$(\text{шина адреса}) = (\text{счетчик команд})$
 $(\text{шина данных}) = \text{CALL}(\text{CD}_{16})$
 $(\text{счетчик команд}) = (\text{счетчик команд}) + 1$
 Команда декодирована



$(\text{шина адреса}) = (\text{счетчик команд})$
 $(\text{шина данных}) = ((\text{счетчик команд}))$
 $(\text{счетчик команд}) = (\text{счетчик команд}) + 1$
 Младшие 8 бит адреса помещаются в рабочий регистр



$(\text{шина адреса}) = (\text{счетчик команд})$
 $(\text{шина данных}) = (\text{счетчик команд})$
 $(\text{счетчик команд}) = (\text{счетчик команд}) + 1$
 Старшие 8 бит помещены в рабочий регистр W

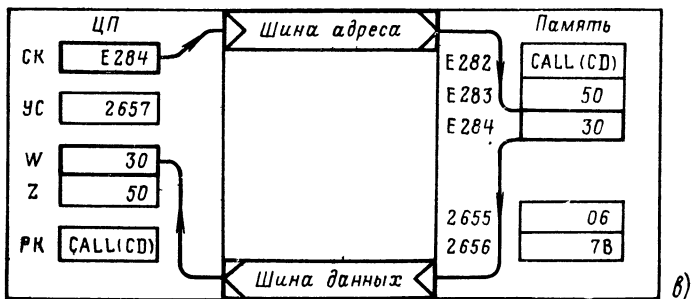


Рис. 7.34. Выполнение

а — цикл 1, Выборка команды; б — цикл 2, Чтение из памяти (регистры W и Z являются Запись в стек;

менты памяти будут отключены (так как $\overline{\text{MEMR}}$ включает в себя $\overline{\text{DBIN}}$).

Чтобы сформировать общие разрешающие сигналы для памяти или буферов, действующим значением которых является 0, может потребоваться подача сигнала $\overline{\text{MEMR}}$ и разрешающих сигналов от дешифраторов на схемы ИЛИ (при условии, что действующими значениями выходных сигналов дешифраторов являются 0).

Зная времена задержки и установки, легко получить временные характеристики для наихудшего случая. Время доступа для совместимой памяти t_{acc} определяется минимальным временем, которое обеспечит требуемые времена установки t_{ds1} (в течение Φ_1) и t_{ds2} (в течение Φ_2):

$$t_{da} + t_{acc} < 2t_p - t_{ds2}; \quad (7.11a)$$

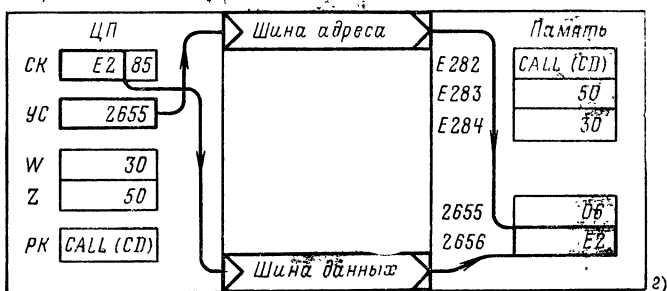
$$t_{da} + t_{acc} < 2t_p - t_{\Phi 2} - t_{d1} - t_{ds1}, \quad (7.11b)$$

(указатель стека) = (указатель стека) - 1

(шина адреса) = (указатель стека)

(шина данных) = старшие 8 бит счетчика команд

Старшие 8 бит счетчика команд помещаются в стек



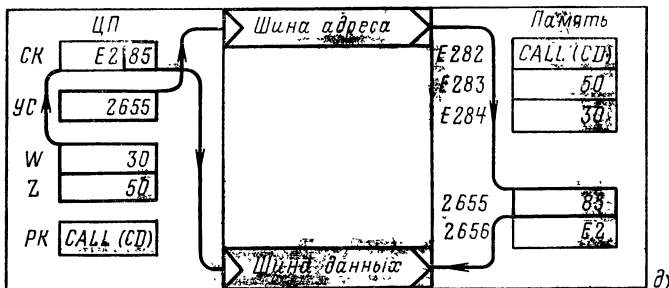
(указатель стека) = (указатель стека) - 1

(шина адреса) = (указатель стека)

(шина данных) = младшие 8 бит счетчика команд

Младшие 8 бит счетчика команд помещаются в стек

(счетчик команд = (W и Z))



команды CALL 3050H:

внутренними и программисту не доступны); в — цикл 3. Чтение из памяти; г — цикл 4. д — цикл 5. Запись в стек

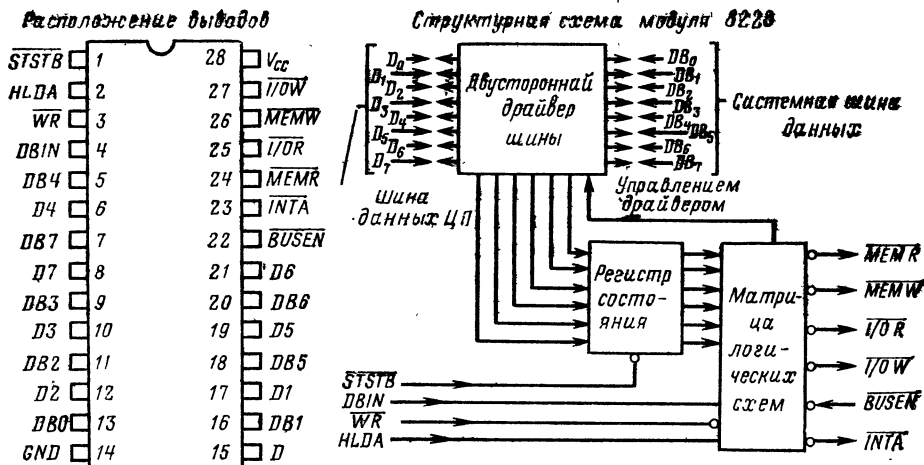


Рис. 7.35. Описание контроллера системы типа Intel 8228

D7=D0 — шина данных (со стороны 8080); DB7=DB0 — шина данных (со стороны системы); I/OR — чтение ввода-вывода; I/OW — запись ввода-вывода; MEMR — чтение из памяти; MEMW — запись в память; DBIN — DBIN (от 8080); INTA — подтверждение прерывания; HLDA — HLDA (от 8080); WR — WR (от 8080); BUSEN — разрешающий вход шины; STSTB — стробирующий сигнал состояния (от 8224); Vcc — +5 В; GND — 0

где t_{da} — максимальное время задержки адреса, равное 200 нс; t_p — период тактового генератора, равный 500 нс (стандартное значение); t_{ds2} — время установки данных во время ϕ_2 , равное 130 нс; t_{ϕ_2} — длительность ϕ_2 , равная 250 нс (стандартное значение); t_{d1} — время задержки от ϕ_1 до ϕ_2 , равное 50 нс (стандартное значение); t_{ds1} — время установки данных в течение ϕ_1 , равное 30 нс. Итак, требуемое время доступа — это минимальное время из двух времен:

$$t_{acc} < 670 \text{ нс}; t_{acc} < 470 \text{ нс}.$$

Таким образом, для работы с максимальной скоростью для МП Intel 8080 требуется память с временем доступа, не превышающим 470 нс.

Интерфейс с более медленными ЗУ может быть обеспечен с помощью линии READY (ГОТОВ). Если линия READY не находится в состоянии 1 хотя бы за 120 нс до среза сигнала ϕ_2 в такте T_2 , ЦП на один цикл тактового генератора перейдет в состояние WAIT (ОЖИДАНИЕ) и автоматически продлит действие всех прочих управляющих сигналов. Линия ГОТОВ может быть синхронизирована с тактовым генератором с помощью триггера, который является частью модуля 8224, представляющего собой тактовый генератор.

Проектирование модуля памяти МП Intel 8080 не вызывает особых сложностей. Так как по сигналу СБРОС управление передается ячейке с нулевым адресом то младшие адреса памяти отводятся под ПЗУ, и систему можно запускать без необходимости повторной за-

грузки программы. Дешифраторы могут вырабатывать разрешающие сигналы для различных элементов памяти; модули памяти в зависимости от новых требований могут быть легко реорганизованы или расширены.

Структура управления МП Motorola 6800

На рис. 7.36 показано расположение выводов микропроцессора Motorola 6800. Интерес представляют следующие сигналы:

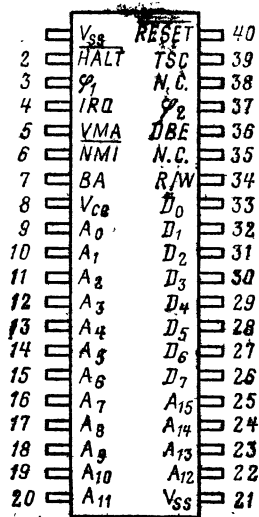
- 1) двухфазного тактового генератора (входы Φ_1 и Φ_2);
- 2) 16-битной адресной шины с тремя состояниями (A_0-A_{15});
- 3) 8-битной двунаправленной информационной шины (D_0-D_7);
- 4) сигнал $\overline{\text{RESET}}$ (вывод 40). По положительному перепаду сигнала в этой линии в СК загружается содержимое двух старших ячеек памяти (имеющих адреса FFFE и FFFF);
- 5) сигнал VMA (вывод 5), сообщающий о том, что на адресной шине присутствует стабилизированный адрес. Этот сигнал существует один в течение всех циклов, в которых используется память;
- 6) сигнал (R/W) (ЧТЕНИЕ/ЗАПИСЬ) (вывод 34), сигнализирующий о направлении передачи данных между процессором и памятью (при записи сигнал равен 0);
- 7) сигнал разрешения для информационной шины DBE (вывод 35), при нулевом значении переводящей шину данных в состояние «цепь разомкнута».

Непосредственно в МП Motorola 6800 вырабатывается очень мало сигналов состояния. Недостающие сигналы должны быть сгенерированы вне микропроцессора. Следует обратить внимание на то, что фиксированная ячейка для обработки сигнала СБРОС имеет ненулевой адрес; ее адрес хранится в ячейках с адресами FFFE и FFFF. Поскольку Motorola имеет прямую адресацию нулевой страницы памяти, ячейки с младшими адресами в системе 6800 обычно относятся к ОЗУ; в результате фиксированные ячейки памяти, резервируемые для передачи управления по сигналу СБРОС и обработки прерываний, должны располагаться в другом месте. Такое их расположение затрудняет процесс проектирования блоков памяти.

Временные характеристики МП Motorola 6800

В МП Motorola 6800 используется двухфазный неперекрывающийся тактовый генератор с максимальной частотой 1 МГц (т. е. с длительностью периода 1 мкс). Фазы Φ_1 и Φ_2 обычно имеют длительность около 450 нс.

Рис. 7.36. Расположение выводов микропроцессора Motorola 6800



Процессор работает по простой схеме. На выполнение команды затрачивается от 2 до 12 циклов тактового генератора. Каждый цикл состоит из фазы ВЫБОРКИ (φ_1), во время которой ЦП помещает на адресную шину адрес, и фазы ВЫПОЛНЕНИЯ (φ_2), во время которой ЦП выполняет операцию обмена данными с памятью. Обычно сигнал φ_2 привязан к входу РАЗРЕШЕНИЕ РАБОТЫ ИНФОРМАЦИОННОЙ ШИНЫ так, что информационная шина находится в состоянии «цепь разомкнута» при отсутствии сигнала φ_2 .

Циклам, в которых осуществляется доступ к памяти, соответствует сигнал $VMA=1$. Во время циклов, используемых ЦП для внутренних операций, $VMA=0$.

Процессор работает следующим образом:

1. По фронту φ_1 ЦП подает адрес и сигналы R/W и VMA на соответствующие линии. Максимальное время задержки для всех сигналов равно 300 нс.

2. В течение фазы φ_2 ЦП пересылает данные в память или из памяти. Максимальная задержка данных относительно фронта φ_2 составляет 225 нс, а минимальное время установки данных относительно среза φ_2 равно 100 нс.

Следует обратить внимание на то, что специальный сигнал записи не предусмотрен. При нормальных условиях тактовый сигнал φ_2 завершает операцию чтения или записи, если только некоторое внешнее устройство не обеспечивает подачу сигнала записи. Запоминающее устройство, используемое совместно с Motorola 6800, должно правильно работать под управлением тактового генератора. Это значит, что время захвата данных должно быть равно нулю, так как сигнал ЧТЕНИЕ/ЗАПИСЬ действует в течение всей операции.

Выполнение команды в МП Motorola 6800

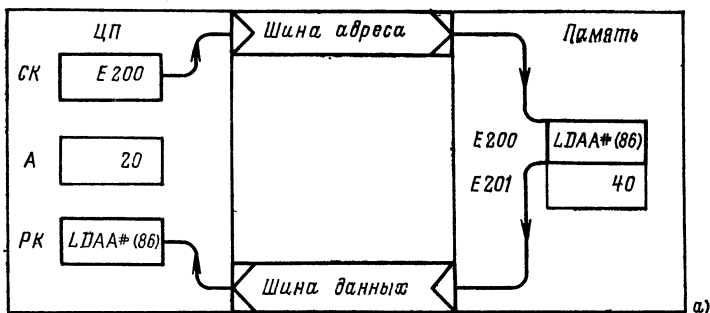
Каждая операция выборки команды или данных выполняется за один период тактового генератора. Все команды требуют по крайней мере еще одного периода после первоначальной выборки. Команды длиной в одну и две ячейки не требуют дополнительных обращений к памяти или внутренних операций, а следовательно, выполняются за два такта генератора. При выполнении других команд требуются дополнительные циклы для выборки данных и адресов и для выполнения таких внутренних операций, как сложение при индексной или относительной адресации (и тот и другой методы адресации требуют выполнить сложение 16-битных чисел, на что уходит два цикла).

Пример 1. LDAA # \$ 40 [послать данные, указанные непосредственно в команде (40_{16}), из памяти в аккумулятор A].

На рис. 7.37 изображен процесс выполнения команды LDAA # \$ 40. Выполнение команды осуществляется за два тактовых цикла. Во время первого цикла ЦП выбирает и дешифрирует команду. Во время второго цикла ЦП выбирает данные и помещает их в аккумулятор A.

В обоих циклах осуществляется доступ к памяти и сигнал VMA остается равным 1.

$\{шина\ адреса\} = \{счетчик\ команд\}$
 $\{шина\ данных\} = LDA\#(86)_{16}$
 $\{счетчик\ команд\} = \{счетчик\ команд\} + 1$
 Команда декодирована



$\{шина\ адреса\} = \{счетчик\ команд\}$
 $\{шина\ данных\} = \{счетчик\ команд\} = 40$
 $\{счетчик\ команд\} = \{счетчик\ команд\} + 1$
 $\{A\} = \{шина\ данных\} = 40$
 Команда выполнена

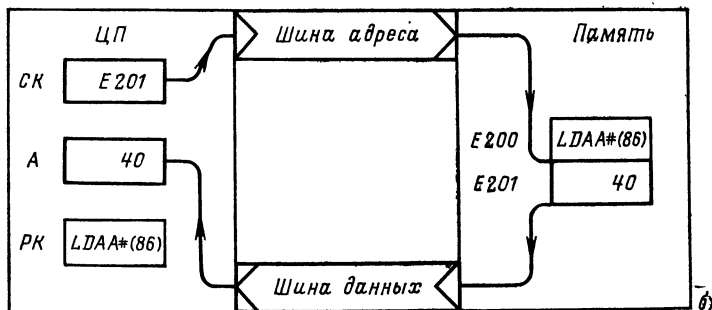


Рис. 7.37. Выполнение команды $LDA\# \$40$

а — цикл 1. Выборка команды; б — цикл 2. Чтение из памяти

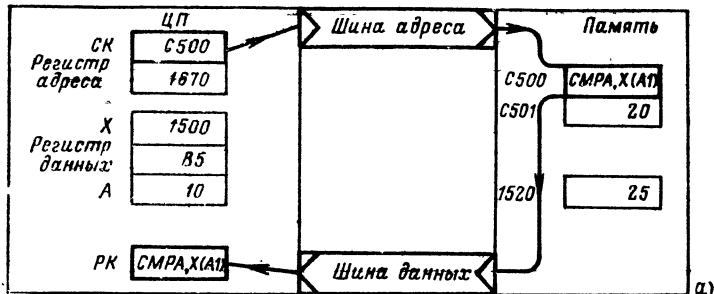
Пример 2. $CMRA\ \$20, X$ (сравнить данные из ячейки, определяемой индексным регистром, с содержимым аккумулятора А).

На рис. 7.38 показано выполнение команды $CMRA\ \$20, X$. На выполнение команды затрачивается пять тактовых циклов. В первом цикле выбирается команда, во втором — смещение, в третьем и четвертом выполняется индексирование (число 20_{16} прибавляется к содержимому индексного регистра) и в пятом цикле осуществляется выборка данных из индексируемой ячейки памяти и выполняется команда. В первом, втором и пятом циклах осуществляется доступ к памяти и поэтому $VMA=1$. В третьем и четвертом циклах выполняются внутренние операции и поэтому $VMA=0$.

Управление памятью МП Motorola 6800

Управление памятью в МП Motorola 6800 несколько сложнее, чем в МП Intel 8080, поскольку память не участвует в большинстве циклов, отсутствует сигнал записи и используется активная фаза тактово-

(шина адреса) = (счетчик команд)
 (шина данных) = $CMRA, X(A1_{16})$
 (счетчик команд) = (счетчик команд) + 1
 Команда декодирована



(шина адреса) = (счетчик адреса)
 (шина данных) = ((счетчик команд) = 20
 (счетчик команд) = (счетчик команд) + 1
 Смещение помещено в регистр данных

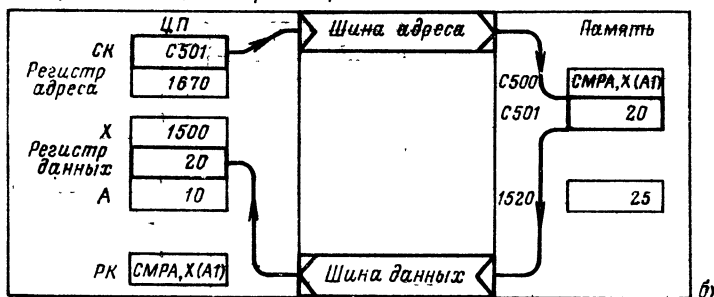


Рис. 7.38. Выполнение

а — цикл 1. Выборка команды (регистры адреса и данных являются рабочими регистрами
 б — циклы 3 и 4. Вычисление исполнительного адреса (во время выполнения этих циклов
 жимое шины адреса. Следует обратить внимание на то, что выполняются оба цикла, даже
 память используется,

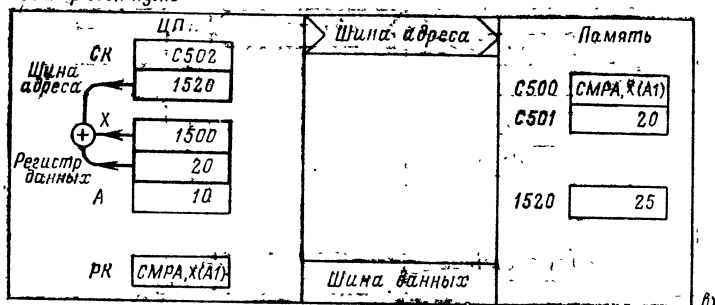
го генератора. Управление модулем памяти проектировщик может организовать следующим образом:

1. Разрешить работу всех элементов памяти с помощью ϕ_2 и VMA так, чтобы не возникло конкуренции при доступе к памяти и чтобы внешние устройства могли использовать память во время циклов, в которых ЦП не использует ее. Однако следует иметь в виду, что сигнал записи отсутствует и потому нужны ЗУ с нулевым временем захвата данных.

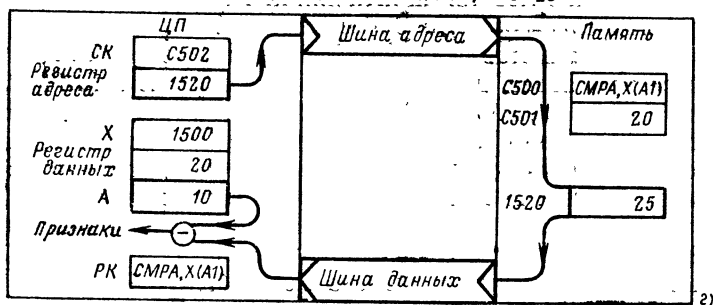
2. Разрешить работу ПЗУ, используя сигнал ЧТЕНИЕ/ЗАПИСЬ так, чтобы они не работали во время циклов записи.

3. Управлять буферами с помощью фазы ϕ_2 тактового генератора таким образом, чтобы избежать конкуренции при доступе к шине и чтобы разрешить использование информационной шины другими устройствами в течение ϕ_1 .

(шина адреса) = 8 циклов 3 (индексный регистр), 8 циклов 4
(индексный регистр) + смещение
Шина данных не используется
(регистр адреса) = индексный регистр + смещение
VMA равен нулю



(шина адреса) = (регистр адреса)
(шина данных) = ((регистр адреса))
Признаки формируются в соответствии с результатом операции
(A) - (шина данных) = 10 - 25



команды C501 \$ 20, X:

процессора, используемыми для хранения адресов и данных); б — цикл 2. Чтение из памяти; ЦП не обращается к памяти и поэтому VMA=0. Для целей отладки можно получить содержание смещения равно 0); г — цикл 5. Чтение из памяти и выполнение команды (поскольку VMA=1)

Время доступа к совместимой памяти зависит от характеристик тактового генератора, времени задержки адреса и времени установки данных. Ограничение имеет вид:

$$t_{da} + t_{acc} < t_{\phi_1 \phi_2} - t_{dsr}, \quad (7.12)$$

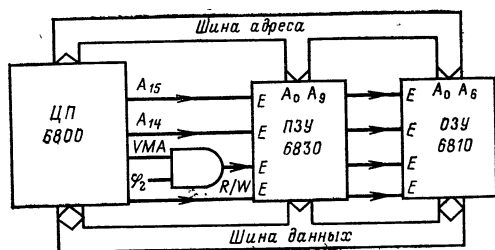
где t_{da} — время задержки адреса; $t_{\phi_1 \phi_2}$ — общая минимальная длительность ϕ_1 и ϕ_2 ; t_{dsr} — время установки данных.

Минимальное время доступа

$$t_{acc} < t_{\phi_1 \phi_2} - t_{dsr} - t_{da};$$

$$t_{acc} < 965 - 300 - 100 < 565 \text{ нс.}$$

Для подключения ЗУ с меньшим временем доступа необходимо замедлить фазу ϕ_2 тактового генератора. В МП Motorola 6800 это замедление может быть произвольным. Значение задержки не обязательно



Работа ПЗУ разрешена,
если

$$A_{15} = 1$$

$$A_{14} = 1$$

$$\text{VMA } \varphi_2 = 1$$

$$R/W = 1 \text{ (чтение)}$$

Работа ОЗУ разрешена,
если

$$A_{15} = 0$$

$$A_{14} = 0$$

$$\text{VMA } \varphi_2 = 1$$

Рис. 7.39. Простейший блок памяти для МП Motorola 6800

должно выражаться целым числом тактовых циклов, как это имеет место в МП Intel 8080.

Проектирование модуля памяти для МП Motorola 6800 осложняется тем обстоятельством, что адрес передачи управления по сигналу СБРОС находится в старших адресах памяти. Ячейки ПЗУ, в которых содержится этот адрес, должны иметь адреса FFFE и FFFF. Следовательно, это ПЗУ должно иметь самые старшие адреса и должно быть размещено так, чтобы не мешать расширению памяти. Кроме того, ячейки с 0000 по 00FF должны находиться в ОЗУ, чтобы их можно было использовать для временного хранения данных, самый быстрый доступ к которым обеспечен тем, что им соответствуют прямые адреса нулевой страницы.

Блоки памяти большего размера могут быть образованы с помощью дешифраторов по схеме, описанной ранее. При проектировании небольших блоков памяти целесообразно придерживаться следующих рекомендаций:

1. Использовать корпуса памяти с несколькими разрешающими входами. Эти разрешающие входы могут быть связаны с адресными линиями, с входами VMA, φ_2 и ЧТЕНИЕ/ЗАПИСЬ (от ПЗУ). Для реализации небольших блоков памяти целесообразно использовать модуль ОЗУ типа Motorola 6810 на 128 8-битных ячейках с двумя разрешающими входами с действующими значениями 0 и четырьмя разрешающими входами с действующими значениями 1, а также модуль ПЗУ

Таблица 7.9. Распределение памяти в простейшем варианте системы на базе МП Motorola 6800

A_{15}	A_{14}	Значение
0	0	ОЗУ, включая страницу 0
0	1	Ввод-вывод
1	0	Ввод-вывод
1	1	ПЗУ, включая программу обработки сигнала СБРОС и адреса программ обработки прерываний

типа Motorola 6830 на 1К 8-битных ячеек с четырьмя программируемыми разрешающими входами.

2. Использовать для выборки ПЗУ, ОЗУ и устройств ввода-вывода одну или две старшие адресные линии. Использование двух старших адресных линий, как это показано на рис. 7.39, позволяет иметь по 16 К ячеек памяти каждого типа с адресами, распределенными в соответствии с табл. 7.9.

3. Выполнять функции управления шиной путем разрешения ее работы только в течение фазы Φ_2 тактового периода. Модули ОЗУ типа Motorola 6810 и ПЗУ типа Motorola 6830 обеспечивают совместимые с требованиями системы времени доступа.

7.6. ВЫВОДЫ

Основной операцией, выполняемой микропроцессором, является пересылка данных и команд в модуль памяти или из него. Адресная шина содержит сигналы, которые указывают нужную ячейку памяти. Обычно адресные сигналы частично дешифрируются в блоках памяти и частично во внешнем дешифраторе. Шина данных содержит данные или команды. Поскольку блок памяти обычно состоит из двух раздельно адресуемых частей, необходимо реализовать такую структуру шины, которая обеспечивает использование информационной шины в каждый данный момент времени только одним типом памяти. Требуемая структура может быть реализована путем использования выходов и буферов с тремя состояниями, взаимно исключающих сигналов дешифрирования и синхронизирующего сигнала активной фазы. Блок памяти должен удовлетворять таким временным ограничениям, которые обеспечивают предоставление входных данных процессору в нужное время и корректируют запись выходных данных в память без возникновения ошибочных операций, разрушающих информацию.

ГЛАВА ВОСЬМАЯ

ОРГАНИЗАЦИЯ ВВОДА-ВЫВОДА В МИКРОПРОЦЕССОРНЫХ СИСТЕМАХ

В этой главе рассматривается подсистема ввода-вывода микро-ЭВМ. Глава начинается с описания общих принципов организации ввода-вывода и интерфейса между УВВ и ЦП для простых устройств ввода-вывода, состоящих из одного порта. Далее рассматриваются более сложные подсистемы ввода-вывода, аппаратные средства ввода-вывода, простые периферийные устройства и реализация подсистем ввода-вывода для микропроцессоров Intel 8080 и Motorola 6800.

8.1. ОСНОВНЫЕ ПРОБЛЕМЫ, ВОЗНИКАЮЩИЕ ПРИ ОРГАНИЗАЦИИ ВВОДА-ВЫВОДА

Ввод и вывод информации подобен обращению к памяти. Процессор может передавать данные УВВ и получать их от УВВ аналогично тому, как он обменивается данными с памятью. Действительно, память — всего лишь разновидность периферийного устройства. Почему же организация подсистемы ввода-вывода оказывается настолько сложной,

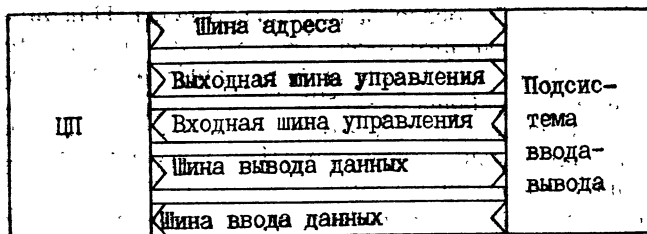


Рис. 8.1. Шины, связывающие ЦП и подсистему ввода-вывода.

По шинам управления могут передаваться сигналы прерывания, ПДП, синхронизации и строб-сигналы

что эту тему часто не затрагивают во вводных курсах? Почему так мало стандартов, относящихся к устройствам ввода-вывода?

Основные проблемы возникают из-за:

- 1) большого разнообразия типов периферийных устройств;
- 2) огромного диапазона скоростей;
- 3) разнообразия типов и уровней сигналов;
- 4) сложности структуры сигналов.

Основная проблема при организации ввода-вывода — разнообразие типов периферийных устройств. Модули памяти выпускаются в виде нескольких основных типов, они обладают близкими скоростями и требуют для своей работы простых сигналов управления. В большинстве микро-ЭВМ модуль памяти представляет собой полупроводниковую БИС, работа которой во многом аналогична работе центрального процессора (ЦП). Кроме того, модуль памяти сохраняет свое содержимое — поэтому у ЦП нет необходимости выбирать данные из памяти в точное заданное время.

Имеется большое разнообразие периферийных устройств. Они могут быть механическими, электромеханическими, электронными и т. д. Для организации их работы могут использоваться дискретные или непрерывные (аналоговые) сигналы. Простой модуль ввода-вывода может содержать температурный первичный преобразователь, данные от которого поступают каждые 5 мин; он может содержать телетайп, передающий 100 бит/с, или гибкий диск, скорость обмена с которым составляет 250 000 бит/с. Разумеется, данные на вводных линиях ЭВМ могут меняться независимо от ее работы. Могут потребоваться сигналы для фиксации или преобразования данных, а также сигналы управления режимом работы УВВ.

Существует мало стандартов на организацию работы подсистем ввода-вывода. Большинство наиболее известных стандартов рассматривается далее. Организация работы каждого периферийного устройства порождает уникальную проблему. Для преобразования сигналов из формата, используемого ЭВМ, в формат УВВ и обратно требуется специальный интерфейс. Учитывая доступность однокристальных процессоров и модулей памяти большой емкости, легко понять, что подсистема ввода-вывода — наиболее дорогостоящая часть многих микро-ЭВМ. Многие разработанные в последнее время сложные БИС предназна-

ны исключительно для организации ввода-вывода. Например, универсальный периферийный интерфейс (Universal Peripheral Interface) Intel 8041 — микро-ЭВМ, используемая как периферийный контроллер в системах на основе процессора 8080¹.

Циклы ввода-вывода выполняются аналогично циклам памяти. На рис. 8.1 показаны соединения между ЦП и модулем ввода-вывода. Все шины предназначены для тех же целей, что и шины, обслуживающие модули памяти. Однако управляющие сигналы в подсистеме ввода-вывода часто более многочисленны. Это связано с большим разнообразием функций управления, необходимых для организации ввода-вывода.

Описание процедуры ввода

Операция ввода информации аналогична циклу чтения из памяти. Для ее реализации необходимы три шага:

1) центральный процессор выставляет адрес на шине адреса. На этом шаге выбирается конкретная схема ввода, или *порт*. Порт может иметь любую разрядность, однако наиболее удобны порты с разрядностью, равной длине слова, используемого ЦП;

2) центральный процессор ждет, когда данные станут доступными;

3) центральный процессор считывает данные с шины данных и помещает их в один из регистров.

Проблема состоит в том, как определить, когда данные доступны для чтения. Цикл памяти происходит в течение фиксированного промежутка времени, а так как модули памяти однотипны по своей структуре с процессором, то легко можно сконструировать совместимый с ЦП модуль памяти либо добавить простую схему задержки. Но как справиться с огромным диапазоном скоростей, с которыми работают периферийные устройства? Простые схемы задержки здесь редко могут принести пользу. Организация ввода-вывода требует более сложного управления временными интервалами.

Цикл ввода имеет такую же временную диаграмму, как и цикл чтения из памяти. Эту временную диаграмму весьма просто реализовать, если анализ наихудшего случая для модуля памяти уже был проведен. Естественно, построить временную диаграмму цикла — не означает решить все проблемы согласования во времени, так как лишь немногие быстродействующие УВВ могут работать подобно памяти. Для таких УВВ времена задержки, установления и захвата (*hold times*) совпадают с временами, описанными в гл. 7.

Согласовать во времени работу ЦП и периферийных устройств ввода можно различными методами:

1. Предполагается, что данные, поступающие с устройства ввода, всегда доступны, так же, как данные в памяти. Такой подход приемлем для организации ввода данных, поступающих от низкоскоростных устройств: механических переключателей или первичных преобразова-

¹D. Phillips and A. Goodman. Slave Microcomputer Lightens Main Microprocessor Load. Electronics, vol. 50, № 14, July 1977, p. 109—112.

телей, измеряющих такие физические величины, как температура или давление. Требуется только, чтобы ЦП считывал данные достаточно часто, чтобы он успевал реагировать на поступление новых данных.

2. Вырабатывается специальный сигнал (например, DATA READY—«данные готовы») для указания, что данные доступны. Этот сигнал может быть дополнительным битом, или стробом, или конкретным кодом, не имеющим другого значения. Такой метод приемлем для скоростей обмена данными в диапазоне от малых до умеренных и пригоден для УВВ, которые подготавливают данные через нерегулярные промежутки времени, или *асинхронно*. Центральный процессор может проверять наличие этого специального сигнала или предупреждаться прерыванием о его появлении.

3. Данные могут приниматься со скоростью, определяемой внешним блоком синхронизации. В этом случае ЦП осуществляет операции ввода с определенной частотой. С помощью этого метода можно управлять вводом данных (с умеренной скоростью) от периферийных устройств, которые подготавливают данные через регулярные промежутки времени, или *синхронно*. Для передачи данных с более высокой скоростью можно использовать метод прямого доступа к памяти (ПДП). В данном случае проблема состоит в первоначальной синхронизации процессора с внешним блоком синхронизации, для чего может потребоваться особая линия управления или специальное синхронизирующее сообщение.

При использовании первого метода ЦП просто выполняет операции ввода с подходящей частотой. Часто даже нет необходимости в периодичности операций, если интервалы между операциями не превосходят максимального времени реакции. Единственная проблема в данном случае — переходные процессы, возникающие при изменении значений вводимых данных. При изменении значений, измеряемых первичными преобразователями, при вращении наборных дисков или при установке переключателей в новое положение могут произойти ошибки ввода. Основные способы устранения указанных ошибок заключаются в следующем:

1) преобразовать медленные или нерегулярные изменения значений данных в дискретные сигналы с помощью одновибраторов, триггеров Шмитта, обычных триггеров или логических схем;

2) использовать другой вход (например, переключатель LOAD), чтобы информировать ЦП о том, что данные готовы;

3) программно организовать проверку данных. Центральный процессор убеждается в правильности первого чтения путем повторного чтения после задержки.

Второй метод, требующий специального сигнала «данные готовы», иногда называют «рукопожатием» (handshake)¹. Он состоит из следующих шагов (рис. 8.2):

1) периферийное устройство посылает данные и сигнал «данные готовы» устройству ввода-вывода;

¹В последнее время в отечественной литературе употребляется термин «передача с квитированием». (Прим. пер.)

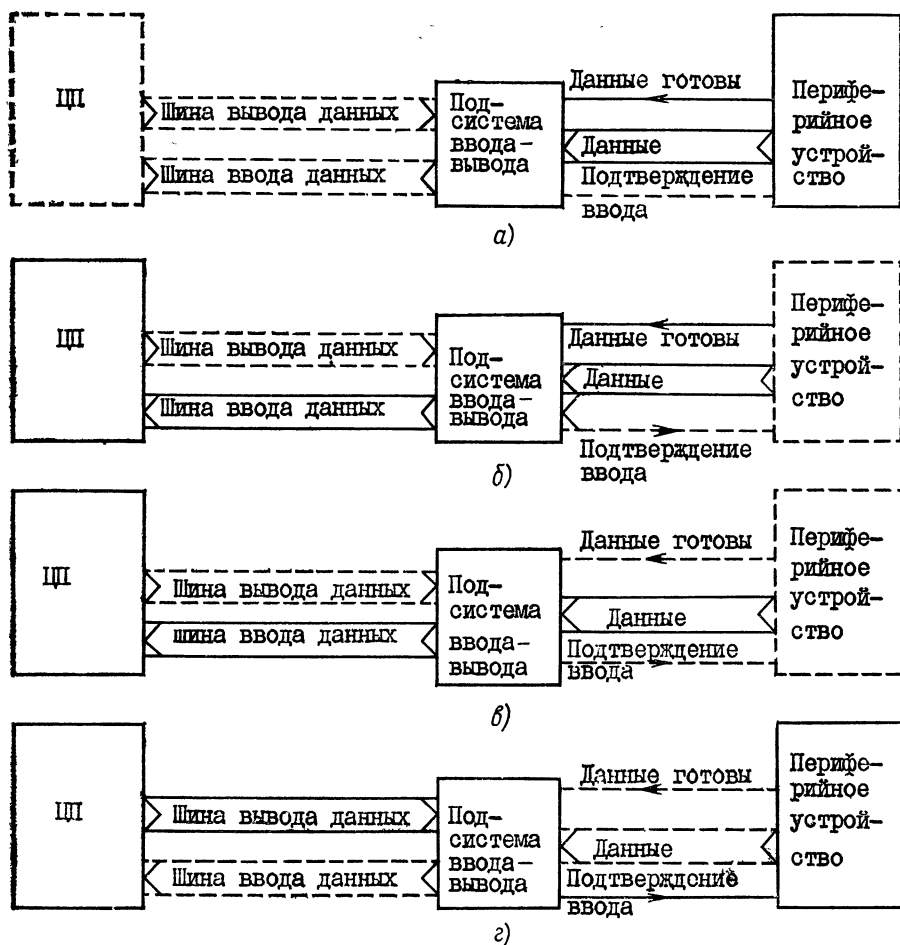


Рис. 8.2. Операция ввода, выполняемая по методу «рукопожатия» (передача с квитированием):

а — шаг 1. Периферийное устройство посылает данные и сигнал «данные готовы» процессору; б — шаг 2. Центральный процессор считывает сигнал «данные готовы»; в — шаг 3. Центральный процессор читает данные; г — шаг 4. Центральный процессор посылает сигнал «подтверждение ввода» периферийному устройству

2) ЦП определяет, что возбужден сигнал «данные готовы». Триггер-защелка (фиксатор) может хранить этот сигнал до момента, когда ЦП прочтает его;

3) ЦП читает данные;

4) ЦП посылает сигнал «подтверждение ввода» (INPUT ACKNOWLEDGE) периферийному устройству. Сигнал указывает, что передача завершилась и периферийное устройство может послать новые данные.

Фактическая передача данных (на шаге 3) — лишь малая часть процесса ввода-вывода. Дополнительная операция ввода должна про-

верить сигнал «данные готовы», а дополнительная операция вывода — выдать сигнал «подтверждение ввода». Модуль ввода-вывода должен фиксировать данные и сигналы управления в течение времени, гарантирующего их прием.

Третий метод, т. е. синхронная передача данных, — самый быстрый метод ввода-вывода.

Если однажды синхронизировать должным образом процессор с периферийным устройством, то передача данных становится регулярной. Например, если УВВ — линия связи, передающая данные со скоростью 2400 бит/с, то процессор должен выполнять операцию ввода каждую $1/2400$ долю секунды. В данном случае проблема заключается в том, как начать и остановить передачу данных.

Обычный метод решения этой проблемы — использовать специальное сообщение, которое не предназначено ни для какой другой цели, кроме синхронизации процессора и периферийного устройства. Конец передачи данных также может быть обозначен специальным кодом. Указанная проблема аналогична проблеме настройки приемника на передатчик в сетях связи. Синхронная передача данных осуществляется быстро, но требует дополнительного оборудования и программного обеспечения.

Описание процедуры вывода

Операция вывода информации аналогична циклу записи в память. Для ее реализации необходимы три шага:

- 1) ЦП выставляет адрес на шине адреса;
- 2) ЦП выставляет данные на шине данных;
- 3) ЦП ждет завершения передачи данных.

Для операции вывода также необходим сигнал записи. Как и при операции записи в память, решающую роль играет порядок выполнения указанных шагов. Ведь требуется не только правильная пересылка данных, но и своевременное окончание их передачи. Выводимые данные нельзя временно разместить в какой-либо вспомогательной области. Поэтому операция вывода должна своевременно завершаться. Временные ограничения здесь такие же, как и для цикла записи в память.

При организации вывода информации трудность в том, чтобы определить, когда передача данных начинается и заканчивается. Сам цикл протекает в соответствии с такой же временной диаграммой, что и цикл записи в память. Согласно этой временной диаграмме сигналы формируются в требуемом порядке, а операция вывода завершается в соответствующий момент времени. Однако реализации временной диаграммы еще недостаточно для решения проблемы, связанной с большим диапазоном скоростей УВВ; центральный процессор должен определить, готово периферийное устройство к приему данных или нет. Кроме того, порт вывода должен сохранять данные в течение времени, достаточного для их приема периферийным устройством.

Для согласования работы ЦП и устройств вывода могут использоваться те же методы, что и для устройств ввода:

1. Вывод осуществляется в предположении, что периферийное устройство всегда готово. Такой метод приемлем для низкоскоростных передач данных на терминалы, содержащие дисплеи, реле, ручные переключатели, т. е. такие, которые работают со скоростью механизма или человека. Скорость передачи данных ЦП не должна превышать скорости реакции этих УВВ.

2. Вырабатывается специальный сигнал, сопровождающий данные. Он может быть реализован как дополнительный бит или как особый код. Такой метод удовлетворителен для скоростей, находящихся в диапазоне от малых до умеренных, и для асинхронных УВВ.

3. Данные передаются со скоростью, определяемой внешним блоком синхронизации. Метод прямого доступа к памяти (ПДП) может обеспечить более высокую скорость передачи данных.

Роли передатчика и приемника данных одинаковы при вводе и выводе. Приемник должен определить, когда данные готовы, и завершить передачу. При вводе ЦП выполняет функции приемника и должен найти данные, при выводе он выполняет функции передатчика и должен гарантировать, что периферийное устройство найдет данные.

Первый из указанных методов пригоден для большинства медленных УВВ. Модуль ввода-вывода должен зафиксировать данные, так как ЦП только на короткое время выставляет их на шине. Данные должны удерживаться на шине достаточное время для считывания их периферийным устройством. Так как низкоскоростные УВВ реагируют медленно, то быстрая смена значений выходных данных бесполезна.

Второй метод — метод асинхронной передачи, или передачи с квитированием — при выводе данных включает в себя следующие шаги:

1) периферийное устройство посылает сигнал «запрос вывода» (OUTPUT REQUEST) или «УВВ готово» (PERIPHERAL READY) в подсистему ввода-вывода;

2) ЦП определяет, что сигнал «УВВ готово» возбужден. Триггер-зашелка может фиксировать этот сигнал;

3) ЦП посылает сигнал «вывод готов» (OUTPUT READY) периферийному устройству;

4) ЦП передает данные периферийному устройству. Следующий сигнал «УВВ готово» может указать ЦП, что передача данных завершена.

Здесь также собственно передача данных — только один из этапов операции вывода. Чтобы гарантировать правильное выполнение передачи данных, необходимы дополнительные устройства, программное обеспечение и время. Модуль ввода-вывода должен содержать выходной порт, а также фиксаторы и схемы управления.

Синхронный вывод обеспечивает наибольшую скорость передачи данных. Как и при вводе данных, проблема заключается в том, как начать и остановить процесс передачи. Обычный способ решения этой проблемы — использование специальных сообщений для синхронизации.

8.2. ПРОСТЫЕ ПОДСИСТЕМЫ ВВОДА-ВЫВОДА

Подсистема ввода-вывода, содержащая один порт ввода

Простая подсистема ввода-вывода может иметь один порт ввода. Если данные поступают от низкоскоростных УВВ (например, от переключателей), то единственными нужными соединениями ЦП с УВВ являются линии шины данных (рис. 8.3). Шина адреса не нужна, так как имеется только один порт.

Центральный процессор оперирует, естественно, со словами только определенной длины. Если у периферийного устройства слово короче, чем у процессора, то оставшиеся линии шины данных можно не подсоединять к УВВ. Неиспользуемые разряды можно очистить с помощью операции маскирования. Если, например, УВВ передает 4-разрядные данные 8-разрядному процессору, то операция маскирования будет иметь вид:

AND \neq 00001111B,

если используются четыре линии младших разрядов шины данных. Периферийное устройство с длиной слова, большей чем у ЦП, требует для передачи данных несколько портов и несколько операций ввода.

Одиночный порт ввода можно использовать как для асинхронной, так и для синхронной передачи данных. При асинхронной передаче данных ЦП должен определить, когда данные доступны для ввода, а при асинхронной — синхронизироваться по внешнему блоку синхронизации. Для этого в случае УВВ, имеющего один порт, нужен специальный код. Центральный процессор должен вести отсчет времени по собственному генератору тактовых импульсов.

При асинхронной передаче данных каждому передаваемому символу должен предшествовать специальный код. Предположим, например, что специальный код — слово, состоящее из всех единиц, т. е. слово 11111111 для 8-разрядных двоичных слов. Передача данных будет происходить следующим образом:

Шаг 1. Прочитать входные данные.

Шаг 2. Если входные данные отличны от специального кода, вернуться к шагу 1.

Шаг 3. Прочитать собственно данные.

Характеристики периферийного устройства определяют временную последовательность операции ввода. Если УВВ передает данные регулярно, то программа может «центрировать» момент приема данных, ожидая дополнительное время, равное половине длительности передачи данных. В этом случае ЦП получит данные в середине последова-

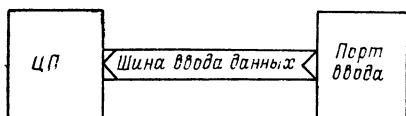
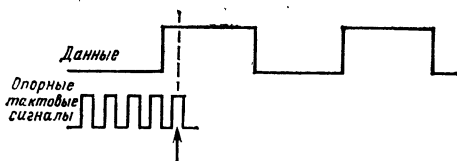


Рис. 8.3. Одиночный порт ввода (предполагается, что УВВ медленное, поэтому нет необходимости в сигналах синхронизации и управления)

Рис. 8.4. Временная диаграмма ввода данных (в момент, отмеченный стрелкой, процессор обнаруживает код. Ожидание в течение половины длительности передачи данных позволяет процессору надежно считать данные)



тельности тактовых сигналов, а не в начале, когда данные могут еще изменяться и вероятность ошибки больше (рис. 8.4).

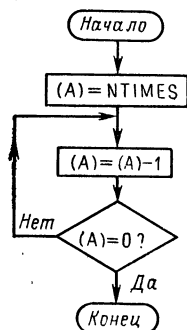
Пример. Предположим, что периферийное устройство передает данные асинхронно с максимальной скоростью 100 8-разрядных слов в секунду. Чтобы оповестить ЦП о доступности данных, используется слово, состоящее из восьми единиц. Процедура ввода данных такова:

Шаг 1. Центральный процессор проверяет входные данные, пока не обнаружит слово 11111111.

Шаг 2. Центральный процессор ждет 15 мс, чтобы выйти на центр сигнала передаваемых данных.

Шаг 3. Центральный процессор читает собственно данные.

1. Блок-схема



2. Расчет времен

Команда	Время выполнения команды	Суммарное время
ЗАГРУЗИТЬ АККУМУЛЯТОР	t_L	t_L
ДЕКРЕМЕНТИРОВАТЬ АККУМУЛЯТОР	t_D	$NTIMES \times t_D$
ПЕРЕЙТИ, ЕСЛИ НЕ НУЛЬ	t_J	$NTIMES \times t_J$

Общее затраченное время равно $NTIMES \times (t_D + t_J) + t_L$.

3. Примеры:

а) Intel 8080 (частота генератора 2 МГц)

$t_D = 2,5$ мкс (DCR A)	MVI A, NTIMES
$t_J = 5$ мкс (JNZ LOOP) LOOP:	DCR A
$t_L = 3,5$ мкс (MVI A, NTIMES)	JNZ LOOP

Общее затраченное время равно $NTIMES \times 7,5 + 3,5$ мкс.

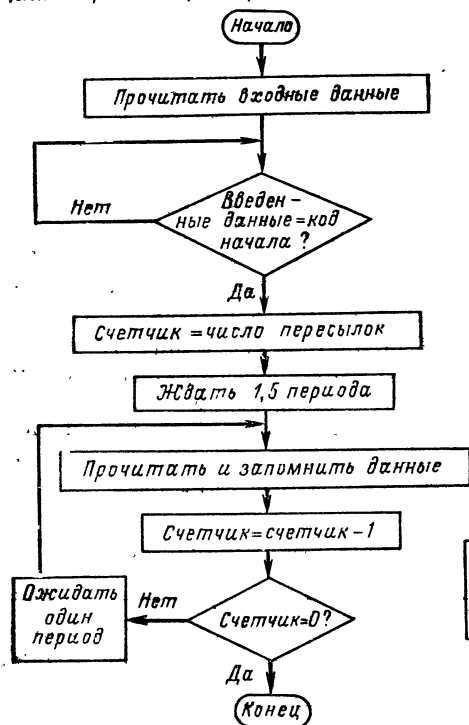
б) Motorola 6800 (частота генератора 1 МГц)

$t_D = 2$ мкс (DECA)	LDA A # NTIMES
$t_J = 4$ мкс (BNE LOOP) LOOP:	DEC A
$t_L = 2$ мкс (LDAA # NTIMES)	BNE LOOP

Общее затраченное время равно $NTIMES \times 6 + 2$ мкс

Рис. 8.5. Простая программно-организованная временная задержка

Число пересылок фиксировано



Резервирован специальный код окончания

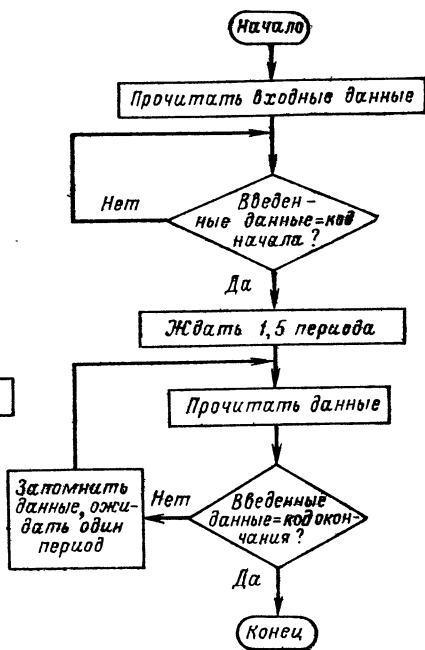


Рис. 8.6. Блок-схемы синхронного обмена

В этом простом случае ЦП может ввести отсчет времени по собственному генератору тактовых импульсов. Приведем пример типичной последовательности команд, реализующих программно-организованную временную задержку:

DELAY: Загрузить аккумулятор # NTIMES
 Декрементировать аккумулятор
 Если не ноль, переход на DELAY

Центральный процессор выполняет команду ЗАГРУЗИТЬ АККУМУЛЯТОР 1 раз, а команды ДЕКРЕМЕНТИРОВАТЬ АККУМУЛЯТОР и ЕСЛИ НЕ НУЛЬ, ТО ПЕРЕХОД столько раз, каково значение константы NTIMES. На рис. 8.5 изображена блок-схема и приведен расчет времени цикла-задержки для МП Intel 8080 и Motorola 6800. Во время задержки процессор полностью загружен.

Центральный процессор может осуществлять синхронную передачу данных таким же способом, как и асинхронную. Единственное различие состоит в том, что процессор продолжает выбирать данные до тех пор, пока процесс передачи не будет остановлен. Если УВВ из предыдущего примера были бы синхронными, то передача данных происходила бы следующим образом:

Шаг 1. Центральный процессор считывает входные данные до тех пор, пока не обнаружит слово, состоящее из всех единиц.

Шаг 2. Центральный процессор ожидает 15 мс, чтобы синхронизироваться со входом.

Шаг 3. Центральный процессор читает слово данных.

Шаг 4. Центральный процессор проверяет, завершена ли передача. Если нет, то ЦП ожидает 10 мс и возвращается к выполнению шага 3.

Остается вопрос, как определить момент окончания передачи. Первая возможность: общая длина всего сообщения известна заранее или ее значение содержится в передаваемых данных. Например, два первых символа потока данных могут указывать его длину. Другая возможность определить момент окончания передачи — резервировать специальный символ, указывающий конец данных. Блок-схемы программной реализации этих двух возможностей показаны на рис. 8.6.

Подсистема ввода-вывода, имеющая один порт ввода, требует небольшого объема аппаратуры. Однако ЦП должен сам искать данные и обеспечивать необходимую синхронизацию. Кроме того, использование дополнительных символов, отмечающих начало и конец потока данных, уменьшает скорость передачи собственно данных.

Устройства ввода-вывода, имеющие один порт вывода

Порт вывода работает по-иному, чем порт ввода, так как необходимо сохранять выходные данные для УВВ. Требуемые соединения ЦП с УВВ (рис. 8.7) — линии шины данных и сигнал «чтение» (точнее, сигнал «чтение/запись»), по которому фиксируются данные. Как и при операции ввода, все операции вывода передают данные в один-единственный порт, а поэтому шина адреса не нужна.

Отличительной чертой устройства вывода является регистр-защелка, или регистр-фиксатор, работающий следующим образом:

1. Когда сигнал синхронизации принимает действующее значение, сигналы данных на выходе регистра совпадают с сигналами данных на входе регистра.

2. Когда сигнал синхронизации принимает недействующее значение, сигналы данных на выходе сохраняют значения, которые они имели перед последней подачей сигнала синхронизации.

Данные должны быть стабильны в течение некоторых минимальных интервалов времени, предшествующих и следующих за подачей сиг-

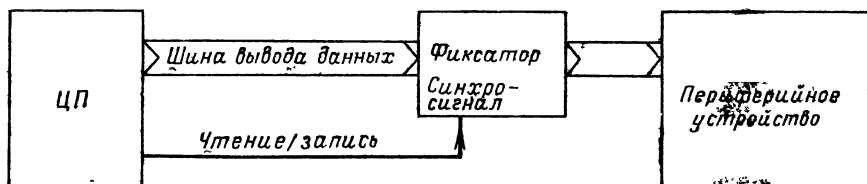


Рис. 8.7. Одиночный порт вывода (данные фиксируются по сигналу WRITE и хранятся в фиксаторе достаточно длительное время для того, чтобы УВВ успело прочитать данные)

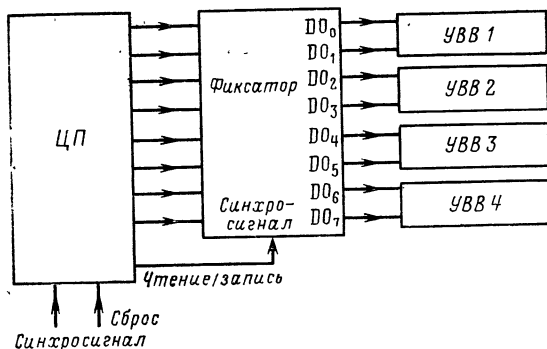


Рис. 8.8. Вывод данных на несколько периферийных устройств через один порт вывода (при выполнении операции вывода на каждое УВВ посылается по 2 бита данных. Разряды 0, 1 данных передаются первому УВВ и т. д. Программа должна подготовить данные для вывода с помощью команд сдвига)

нала синхронизации. Регистр-защелка должен активироваться сигналом «чтение/запись».

Разумеется, порт вывода может иметь любую разрядность. Линии шины данных, не связанные с портом или УВВ, игнорируются; в результате этого каждая операция вывода выполняется частично, а максимальная скорость передачи данных уменьшается. Один порт с длиной слова, равной длине слова ЦП, может управлять несколькими УВВ (рис. 8.8). Данные, предназначенные для УВВ1, помещаются в два младших разряда порта вывода и т. д. Все данные фиксируются в регистре по одному сигналу.

Как синхронная, так и асинхронная передача данных требует дополнительной аппаратуры помимо порта вывода. Так, у ЦП должен быть вход, который ЦП может использовать для синхронизации или для выяснения, готово ли устройство вывода. В данном случае, так же как и для порта ввода, сокращение аппаратных средств ведет к усложнению процесса передачи сигналов, снижению скорости передачи данных и увеличению объема программного обеспечения.

При асинхронной передаче данных каждому передаваемому символу должен предшествовать специальный код, с помощью которого ЦП указывает на присутствие данных. Передача данных происходит следующим образом:

Шаг 1. Центральный процессор посылает специальный сигнал «звывод готов» (OUTPUT READY).

Шаг 2. Центральный процессор ожидает в течение времени, равного одному периоду передачи.

Шаг 3. Центральный процессор посылает собственно данные.

Центральный процессор сам управляет операциями вывода, поэтому здесь нет проблемы, связанной с выходом на центр сигнала. Предполагается, что периферийное устройство подготовится к приему данных за время, пока ЦП ожидает в течение одного периода передачи данных. Если потребуется, ЦП может посылать фиктивные символы периферийному устройству, чтобы дать последнему дополнительное время для завершения подготовки к приему данных. Фиктивные символы — это коды, которые УВВ игнорирует. Ими часто служат символы SYN (пустой символ синхронизации) или NUL (нуль) кода ASCII.

Синхронный вывод данных аналогичен асинхронному, за исключением того, что после первоначальной синхронизации передача данных происходит до тех пор, пока она не будет остановлена специальным способом. Число передаваемых слов может быть фиксировано или содержаться в данных. Для указания конца передачи данных может служить специальный символ.

Операции ввода и вывода несколько отличаются. При вводе ЦП должен определить, когда передача данных начинается и заканчивается. Это означает, что ЦП должен согласовать свои средства отсчета времени с синхросигналами периферийного тактового генератора, которые недоступны процессору. При выводе ЦП определяет начало и конец передачи данных и делает это так, чтобы в момент передачи данных УВВ было готово к приему данных. Порт вывода должен фиксировать передаваемые данные.

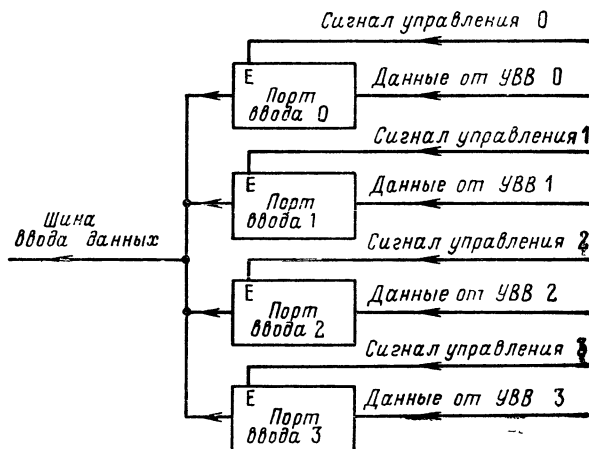
8.3. БОЛЕЕ СЛОЖНЫЕ ПОДСИСТЕМЫ ВВОДА-ВЫВОДА

Большинство подсистем ввода-вывода имеет более одного порта. Если разрядность данных превосходит длину слова процессора или если управляющие сигналы и информация о состоянии должны передаваться раздельно, то для обмена данными ЦП даже с одним УВВ может потребоваться несколько портов. Для подобных подсистем ввода-вывода необходимы шинные структуры, совместимые с шинной структурой подсистемы памяти.

Шинные структуры ввода-вывода

Шинная структура ввода должна позволять адресуемому порту ввода без помех управлять шиной данных. Шинная структура вывода должна обеспечивать фиксацию значений данных, находящихся на шине данных, в адресуемом порте вывода. На рис. 8.9 изображена шинная структура с тремя состояниями. Сигналы управления используются

Рис. 8.9. Шинная структура с тремя состояниями (по сигналам управления, однозначно определяющим порт, данные, находящиеся в одном из портов, помещаются на шину данных ввода, а выходы остальных портов переводятся в высокоимпеданное состояние)



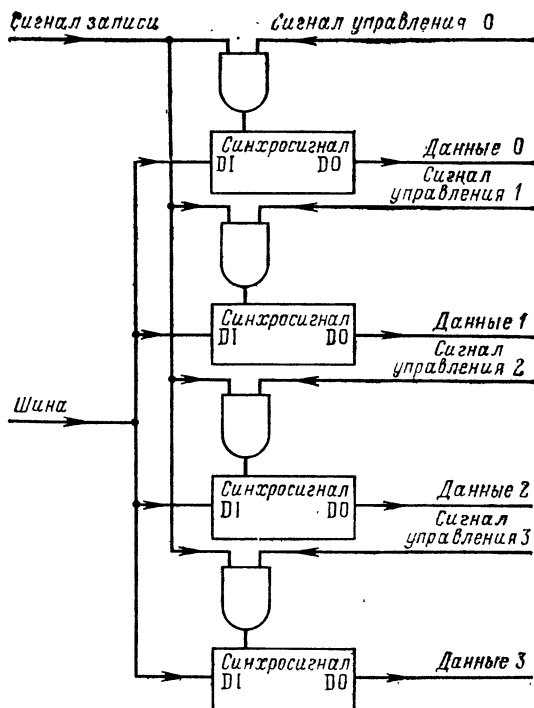


Рис. 8.10. Шинная структура вывода (взаимно исключающие друг друга сигналы управления определяют, какой из фиксаторов будет синхронизироваться сигналом записи и будет, таким образом, повторять на выходе входные для этого фиксатора данные; *DI* — вход фиксатора; *DO* — выход фиксатора. Предполагается, что действующее значение сигнала записи высокое)

для того, чтобы открыть порт. В шинной структуре вывода, изображенной на рис. 8.10, управляющие сигналы используются для синхронизации фиксаторов. Чтобы управляющие сигналы были взаимно исключающими, используются декодеры, как это было описано в гл. 7.

Способы сочетания адресации памяти с адресацией УВВ

Почти все микропроцессоры используют одни и те же шины для обмена данными как с памятью, так и с УВВ. Следовательно, возникает вопрос, как различить эти два вида обмена. Существуют три стандартных метода:

1. *Изолированных линий.* При применении этого метода адреса, используемые для обращения к памяти и УВВ, декодируются отдельно.

2. *Ввода-вывода, адресуемого как память (memory-mapped input/output).* В данном методе порты ввода-вывода рассматриваются как ячейки памяти.

3. *Встроенного ввода-вывода (attached input/output)*. Порты ввода-вывода являются частью модуля ЦП или модулей памяти. Они активируются специальными командами.

На рис. 8.11 изображена структура микро-ЭВМ с изолированными линиями ввода-вывода. С помощью сигнала «выбор подсистемы» (SECTION SELECT) система различает операции ввода-вывода данных и обмена данными с памятью и открывает в зависимости от этого ту или иную группу шинных драйверов. Этот сигнал может также активировать любой модуль памяти или порт ввода-вывода. Для организации ввода-вывода необходимы особые команды. Выполнение этих команд устанавливает сигнал «выбор подсистемы» в состояние «ввод-вывод». В качестве примеров микропроцессоров, которые могут использовать метод изолированных линий, можно привести микропроцессоры Intel 8080, Zilog Z-80, Signetics 2650.

Преимущества метода изолированных линий заключаются в следующем:

1) адрес порта ввода-вывода может быть коротким. В большинстве систем число портов намного меньше числа ячеек памяти. Для адреса порта ввода-вывода обычно достаточно восьми разрядов, для адреса ячейки памяти требуется обычно больше разрядов. Короткий адрес порта ввода-вывода означает возможность упрощения декодирующих систем и использования более коротких команд;

2) могут быть легко разработаны дополнительные сигналы управления передачей информации при вводе-выводе. Для модулей памяти редко требуются сигналы стробирования и квитирования, часто необходимые для операций ввода-вывода;

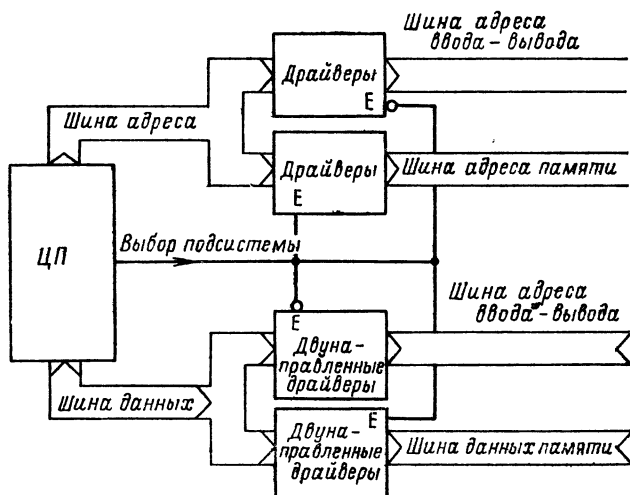


Рис. 8.11. Микро-ЭВМ с изолированными линиями ввода-вывода (сигнал «выбор подсистемы» (SECTION SELECT) указывает, что выполняется цикл памяти, если уровень сигнала высокий, и цикл ввода-вывода, если уровень — низкий. Высокое значение сигнала отпирает шины памяти, низкое — шины ввода-вывода)

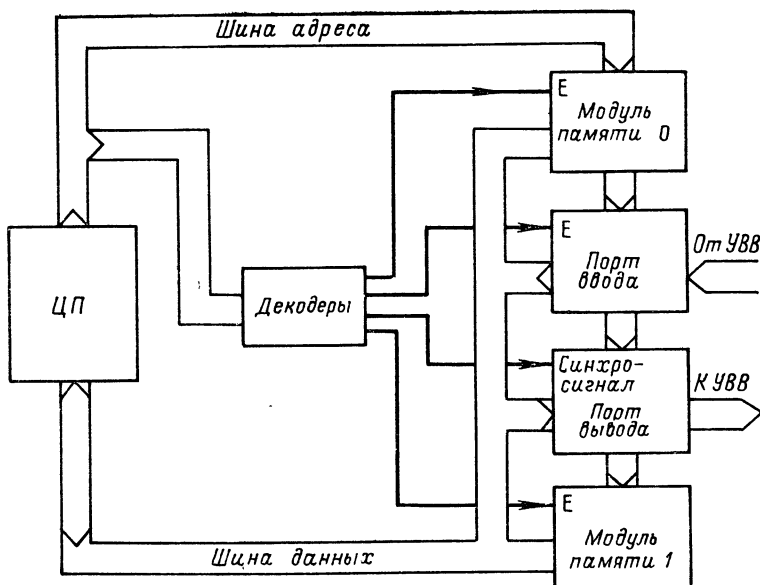


Рис. 8.12. Ввод-вывод, адресуемый как память (единственное отличие модулей ввода-вывода и памяти состоит в адресах, по которым производится обращение к модулям)

3) программы становятся более наглядными, так как операции ввода-вывода выполняются с помощью команд, отличных от команд других операций;

4) разработка подсистем ввода-вывода может производиться отдельно от разработки модулей памяти, так как управляющие структуры могут быть независимыми.

Изолированный ввод-вывод требует дополнительного декодирования и дополнительных команд; теряется также гибкость, но зато память и порты ввода-вывода здесь действительно различимы естественным образом. Если порты ввода-вывода — простые устройства ТТЛ, а периферийные устройства однонаправленные, то недостаточно четкое различие между памятью и портами ввода-вывода может привести к ошибкам.

На рис. 8.12 показана структура микро-ЭВМ с вводом-выводом, адресуемым как память. Процессор использует одни и те же команды для обмена как с памятью, так и с УВВ. Среди процессоров, разработанных специально для такого ввода-вывода, можно отметить такие, как Motorola 6800, MOS Technology 6502 и PACE фирмы National. Порт ввода-вывода определяется только своим адресом. Естественно, процессор с обособленными командами ввода-вывода может использовать метод адресации портов ввода-вывода как ячеек памяти, если модули памяти и ввода-вывода можно сочетать. Преимущества ввода-вывода, адресуемого как память, заключаются в следующем:

1) любая команда, работающая с данными, находящимися в памяти, может работать и с данными, находящимися в порте ввода или вывода. Никаких особых команд ввода-вывода не требуется, и программирование многих задач может существенно упроститься;

2) не требуется отдельной системы для декодирования и управления вводом-выводом. Благодаря этому число модулей в системе можно сократить;

3) в систему легко включить интерфейсные БИС и специальные контроллеры. Эти устройства часто содержат запоминающие элементы или регистры, настраиваемые процессором. Это обстоятельство является существенным, так как управление столь сложными устройствами, как наборы портов ввода-вывода, затруднительно.

У метода ввода-вывода, адресуемого как память, имеется также и ряд недостатков:

1) трудно отличить операции передачи информации при вводе-выводе от других операций: Трудно читать и отлаживать программу, в которой простые команды вызывают выполнение сложных операций ввода-вывода;

2) порты ввода-вывода требуют выделения им некоторого адресного пространства;

3) результат выполнения многих команд, возможно, будет трудно понять из-за ограничений, связанных с физической природой УВВ. Что произойдет, если, например, ЦП попытается записать данные в УВВ «переключатель» или считать данные с выходных линий?

4) система декодирования может оказаться сложной, так как порты ввода-вывода занимают значительно меньшие участки адресного пространства, чем память. Поэтому придется либо усложнять декодирующую систему, либо примириться с потерей части адресного пространства. Для порта ввода-вывода редко требуется много адресов памяти;

5) использование этого метода затруднительно, если применять только стандартные схемы: сигналы управления не смогут управлять сложным вводом-выводом. Поэтому интерфейсные БИС часто должны генерировать дополнительные сигналы под управлением программы.

Ввод-вывод, адресуемый как память, более всего подходит для систем, использующих сложные интерфейсные БИС, в то время как изолированный ввод-вывод — для систем, в которых используются схемы малой и средней степеней интеграции.

Встроенный ввод-вывод полезен как альтернативный вариант для малых систем, для которых число корпусов играет важную роль. При использовании данного метода порты ввода-вывода и цепи управления являются частью модулей памяти или процессора (рис. 8.13). Встроенный ввод-вывод позволяют применять процессоры Intel 4040, Fairchild F-8, National SC/MP, Texas Instruments TMS 1000 NC и Intel 8048.

Разумеется у подобных систем есть много недостатков. Отметим следующие:

1) требования к подсистеме ввода-вывода не должны быть слишком высокими. Корпуса ЦП и памяти могут предоставить системе небольшое число портов и обеспечить незначительное число сигналов управления;

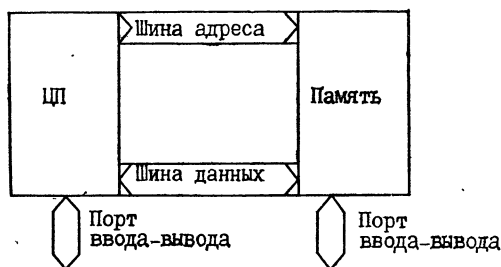


Рис. 8.13. Встроенная подсистема ввода-вывода (здесь порты ввода-вывода интегрированы с модулями ЦП и памяти и находятся на тех же кристаллах)

2) специальные кристаллы памяти более дороги, чем стандартные, а недостаточное число сигналов управления может сильно затруднить использование стандартных микросхем;

3) расширение системы становится трудным и дорогостоящим;

4) необходимы специальные команды или адреса памяти для передачи данных в порты и из них.

Встроенный ввод-вывод полезен в том случае, когда цена и размеры одной платы имеют решающее значение, а в расширении системы нет необходимости. Контроллеры игровых автоматов, электрооборудования и других простых медленных устройств — вот типичные области применения одно- или двухкристальных микро-ЭВМ со встроенным вводом-выводом.

Передача данных с использованием нескольких портов

Для работы многих периферийных устройств необходимо несколько портов. В простейшем случае наличие нескольких портов необходимо, когда слово УВВ длиннее слова ЦП. В такой ситуации процессор должен передавать данные посегментно. Операция вывода 16-разрядного слова, выполняемая 8-разрядным процессором, происходит следующим образом (рис. 8.14):

Шаг 1. Центральный процессор готовит восемь старших разрядов данных.

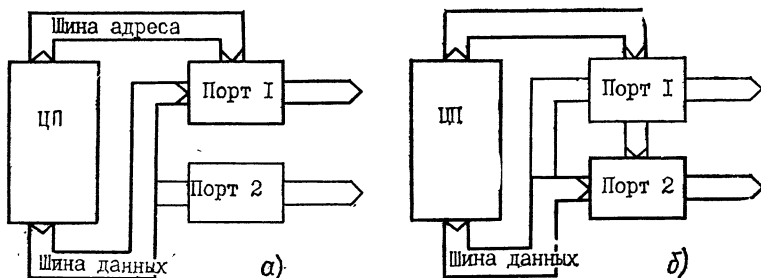


Рис. 8.14. Операция вывода блока, состоящего из нескольких слов:

а — операция вывода 1. Старшие разряды — в порт 1 (центральный процессор готовит старшие восемь разрядов данных; они доступны устройству первыми); **б** — операция вывода 2. Младшие разряды — в порт 2 (центральный процессор готовит младшие восемь разрядов данных и посылает их в порт 2). После некоторой задержки все 16 бит данных становятся доступными для УВВ

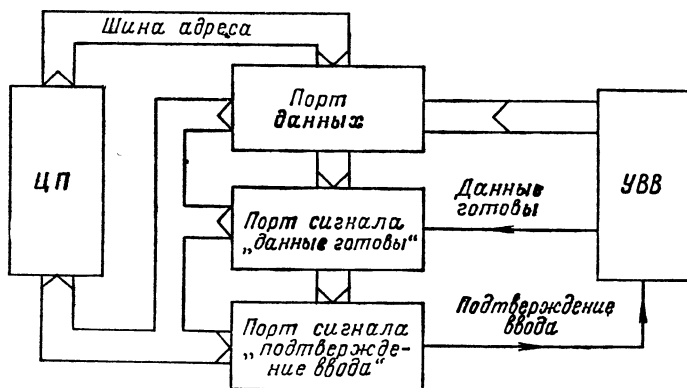


Рис. 8.15. Подсистема ввода-вывода, использующая отдельные порты для обмена сигналами управления и состояния при квитировании (адреса различных портов должны быть различными)

Шаг 2. Центральный процессор посылает восемь старших разрядов данных в первый порт.

Шаг 3. Центральный процессор готовит восемь младших разрядов данных.

Шаг 4. Центральный процессор посылает восемь младших разрядов данных во второй порт.

В приведенном примере две передачи данных выполняются не одновременно. Если необходимо обеспечить одновременность передачи данных, то можно управлять выходным буфером с тремя состояниями с помощью третьего порта. Программа закрывает буфер перед передачей данных и открывает его после передачи. Тогда все 16 бит данных выставляются одновременно, когда ЦП открывает буфер.

Передача блоками, состоящими из нескольких слов, требует выполнения внутренних действий, так как ЦП должен выбрать или поместить каждое слово по нужному адресу памяти или в нужный регистр.

Дополнительные порты можно также использовать для сигналов управления и сигналов состояния. На рис. 8.15 для сигнала «данные готовы» предназначен специальный порт ввода, а специальный порт вывода предназначен для сигнала «подтверждение ввода». Специальные коды для передачи данных не требуются, но используемая аппаратура в данном случае сложнее, чем в описанном выше случае одиночного порта. Либо чтение данных, либо посылка сигнала «подтверждение ввода» должны сбросить сигнал «данные готовы» перед следующим циклом. При выводе данных можно аналогичным образом использовать три порта: данных, ввода (для сигнала «УВВ готово») и вывода (для сигнала «вывод готов»).

В некоторых микропроцессорах имеются специальные последовательные линии ввода-вывода для передачи данных или управляющих сигналов. Определенными командами можно передавать данные на эти линии (или считывать с них) или использовать сигналы на этих линиях в качестве условий переходов. Процессоры Intel 8085, PACE

фирмы National, CP1600 фирмы General Instrument и 9900 фирмы Texas Instruments — примеры процессоров с последовательными линиями ввода-вывода.

Совмещение порта для сигналов состояния с портом для управляющих сигналов позволяет сэкономить оборудование. Такие сигналы, как «данные готовы», «подтверждение ввода», «УВВ готово» и «вывод готов» — однобитные, а поэтому один порт с длиной слова, такой же, как у ЦП, может обеспечить работу с сигналами состояния и управления сразу для нескольких периферийных устройств. Чтобы получить значение сигнала «готово», программа должна проверить (а чтобы выдать сигнал «подтверждение» — изменить) соответствующий разряд. Порты управления могут быть частью порта данных. В таком случае порт данных фиксирует сигнал «готово» и обеспечивает сигнал «подтверждение».

8.4. АППАРАТНЫЕ СРЕДСТВА, ПРИМЕНЯЕМЫЕ ПРИ ВВОДЕ-ВЫВОДЕ

Функции подсистемы ввода-вывода могут выполняться с помощью разнообразной аппаратуры. К числу простых устройств относятся триггеры и одновибраторы. К числу устройств средней степени интеграции относятся декодеры, селекторы, счетчики, сдвиговые регистры и порты ввода-вывода. Большие интегральные микросхемы ввода-вывода включают в себя универсальные асинхронные приемопередатчики (УАПП), универсальные синхронные приемопередатчики (УСПП) и параллельные интерфейсы.

Триггеры

К числу распространенных типов триггеров относятся D-триггеры (от delay — задержка) и RS-триггеры (от reset-set — сбросить-установить). Триггеры этих типов широко используются в качестве (или в составе) счетчиков, переключателей, фиксаторов, регистров и ячеек памяти.

D-триггер — синхронный фиксатор, выходной сигнал которого при действующем значении синхросигнала равен входному, а смена значения синхросигнала фиксирует данные. RS-триггер — несинхронный фиксатор. По сигналу, действующему на входе R, триггер сбрасывается; сигнал на входе S устанавливает триггер. RS-триггер может сохранять сигнал «данные готовы» или «УВВ готово» до тех пор, пока ЦП не проверит этот сигнал. Сигнал «подтверждение» может сбросить триггер после завершения передачи данных. D-триггер, на вход данных которого постоянно подается сигнал высокого уровня, а на вход, предназначенный для синхросигнала, произвольный сигнал, ведет себя (по отношению к последнему сигналу) как RS-триггер.

Одновибраторы

Одновибратор, или *моностабильный мультивибратор*, генерирует одиночный импульс постоянной длительности при поступлении на его вход сигнала. Одновибратор может генерировать импульсы раз-

личной длительности для управления теми УВВ, которые не находятся под непосредственным управлением процессора. Такими устройствами могут быть периферийные устройства, представляющие собой механические или электромеханические системы, действующие гораздо медленнее процессора, генераторы сигналов или аппаратура связи, рабочая частота которых превышает частоту ЦП. Одновибратор может также преобразовывать последовательность коротких импульсов в одиночный импульс большой длительности.

Декодеры

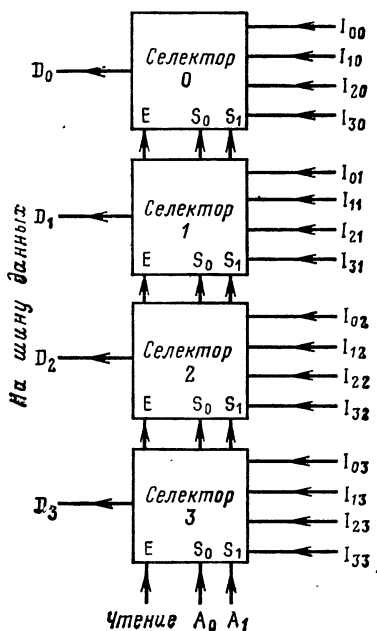
Декодеры, рассмотренные в § 7.3, при поступлении сигнала на вход вырабатывают на выходе сигнал с действующим значением. Выходной сигнал с действующим значением может открывать или тактировать модули памяти, порты ввода-вывода, регистры, буфера или другие элементы системы. Некоторые декодеры фактически являются преобразователями кодов. Наиболее распространенные устройства считывают входные сигналы в кодах BCD или ASCII и преобразовывают их в форму, которая требуется для таких УВВ, как, например, широко распространенный семисегментный цифровой индикатор.

Селекторы

Функции селекторов были описаны в § 7.3. Из этих устройств можно также построить порты ввода; при этом нет необходимости в декодере, так как селектор может декодировать адреса, подаваемые на вход. На рис. 8.16 изображены четыре селектора 4-в-1, работающие как четыре 4-разрядных входных порта. Все устройства имеют одинаковые входы «открыть» и «выбрать», но каждое устройство связано со своим разрядом шины данных процессора; оно, таким образом, работает как составная часть

Выходы селектора					
Адреса		Данные			
A ₁	A ₀	D ₃	D ₂	D ₁	D ₀
0	0	I ₀₃	I ₀₂	I ₀₁	I ₀₀
0	1	I ₁₃	I ₁₂	I ₁₁	I ₁₀
1	0	I ₂₃	I ₂₂	I ₂₁	I ₂₀
1	1	I ₃₃	I ₃₂	I ₃₁	I ₃₀

Рис. 8.16. Использование селекторов в качестве портов ввода (в селекторах входные данные не фиксируются)



системы, состоящей из четырех портов, а не как отдельный порт. Например, если $S_0 = S_1 = 0$, то на шину данных будут поданы значения: I_{00} от селектора 0, I_{01} от селектора 1, I_{02} от селектора 2 и I_{03} от селектора 3. Параллельный 4-разрядный вход должен использовать по одному разряду от каждого селектора. Селектор должен обладать выходами с тремя состояниями для подключения к шине с тремя состояниями.

Счетчики

Счетчик — устройство, которое переходит в отличное от всех предыдущих состояние с поступлением каждого синхросигнала — до исчерпания емкости счетчика. С помощью счетчиков можно менять единицу отсчета времени, организовывать временные задержки, вести отсчет времени, выдавать сигналы управления для мультиплексоров и демultipлексоров. Обычно счетчики бывают десятичными, двоичными, 12-ричными. Возможные режимы работы счетчика: отсчет в прямом и обратном направлениях, режим синхронного выхода (состояние всех

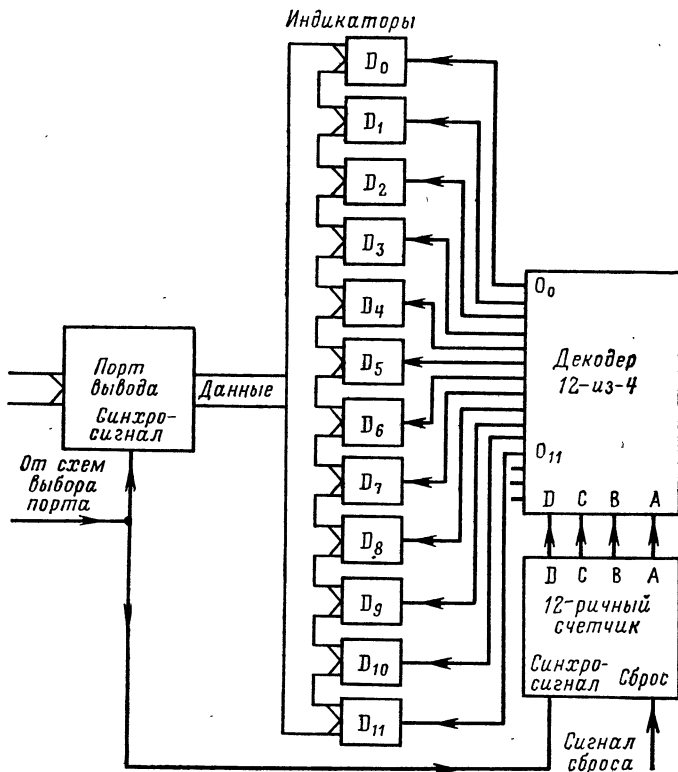


Рис. 8.17. Мультиплексное управление индикаторами с помощью счетчика (декодер определяет, какой индикатор должен быть включен. Счетчик синхронизируется теми же схемами, которые осуществляют выбор порта вывода)

разрядов меняется одновременно). Имеется возможность очистить или предварительно установить счетчик, а также получить выходные сигналы с тремя стабильными состояниями.

На рис. 8.17 показано стандартное использование счетчика в микрокомпьютере. Двенадцатиричный счетчик и декодер 4-в-12 управляют 12-цифровой индикаторной панелью. Сигнал «сброс» очищает счетчик в начале работы. Каждая операция изменяет состояние счетчика и направляет выходной сигнал к другому индикатору. Необходим только один порт вывода. Процессор передает цифры (по одной), а счетчик совместно с декодером автоматически обеспечивают правильное направление передачи. Так как каждая цифра фиксируется достаточно продолжительное время, а сигналы подаются на индикаторы достаточно часто, то наблюдателю будет казаться, что все 12 индикаторов горят непрерывно. Подобные индикаторы обычно применяются в калькуляторах, различных измерительных приборах, контрольных системах и терминалах.

Сдвиговые регистры

Сдвиговой регистр — устройство, которое с поступлением каждого синхросигнала сдвигает свое содержимое на одну позицию влево или вправо. С помощью сдвиговых регистров можно расширить возможности ввода-вывода, тактирования и организации задержек; с их помощью можно также преобразовывать данные из параллельного формата в последовательный и наоборот. Обычно сдвиговые регистры сдвигают влево; они имеют последовательные входы и выходы. Бывают регистры с параллельными входами и выходами, с синхронными выходами, с последовательным и параллельным входом, с возможностью сдвига в обе стороны, с очисткой и предварительной установкой, а также с тристабильными выходами.

На рис. 8.18 показано, каким образом сдвиговой регистр может расширить возможности ввода-вывода процессора. Сначала ЦП очищает сдвиговые регистры по сигналу, поступившему из порта управления; затем он засылает по 7 бит в каждый сдвиговой регистр и, наконец, активирует индикаторы через порт управления. Таким образом, 4-разрядный процессор может обеспечить параллельный вывод 28 бит через единственный порт. Не требуется ни декодирования, ни мультиплексирования.

Порты ввода-вывода

В портах ввода-вывода средней степени интеграции скомбинированы буфера, триггеры, фиксаторы, драйверы и другие схемы. Этими портами можно заменить несколько стандартных устройств. Портam при-сущи некоторые (или все) из следующих признаков:

возможность варьировать соединения между входом и выходом; пользователь может выбирать нужное соединение;

наличие синхронного фиксатора для хранения данных;

наличие буфера, выходы которого имеют три состояния;

наличие шинных драйверов;
 наличие триггеров для фиксации сигнала «готово»;
 возможность генерировать строб-сигналы ввода и вывода;
 наличие нескольких сигналов «открыть» (enable) для управления выбором порта;
 прямая связь с двунаправленными шинами;
 наличие сигналов «открыть» и «очистка» (master enable and master clear);

первоначально определенное состояние запуска (start-up-state).

Модуль Intel 8212, изображенный на рис. 8.19,— типичный порт ввода-вывода. У него имеется два рабочих режима, один из которых выбирается по сигналу MD (от mode — режим; MD=1 для режима вывода; MD=0 для режима ввода).

В режиме ввода периферийное устройство требует, чтобы порт зафиксировал данные, засылая для этого сигнал «данные готовы» (действующее значение — высокий уровень) на вход «строб» (STB). Процессор предоставляет шину для данных в соответствии с логикой работы схем выбора устройства. При такой структуре управления периферийное устройство помещает данные в порт, а ЦП считывает данные.

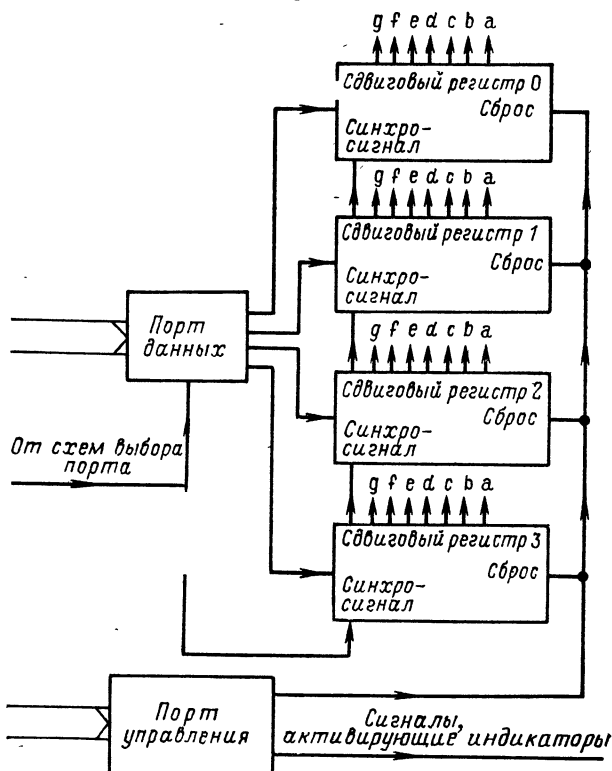


Рис. 8.18. Увеличение разрядности при выводе данных с помощью сдвиговых регистров

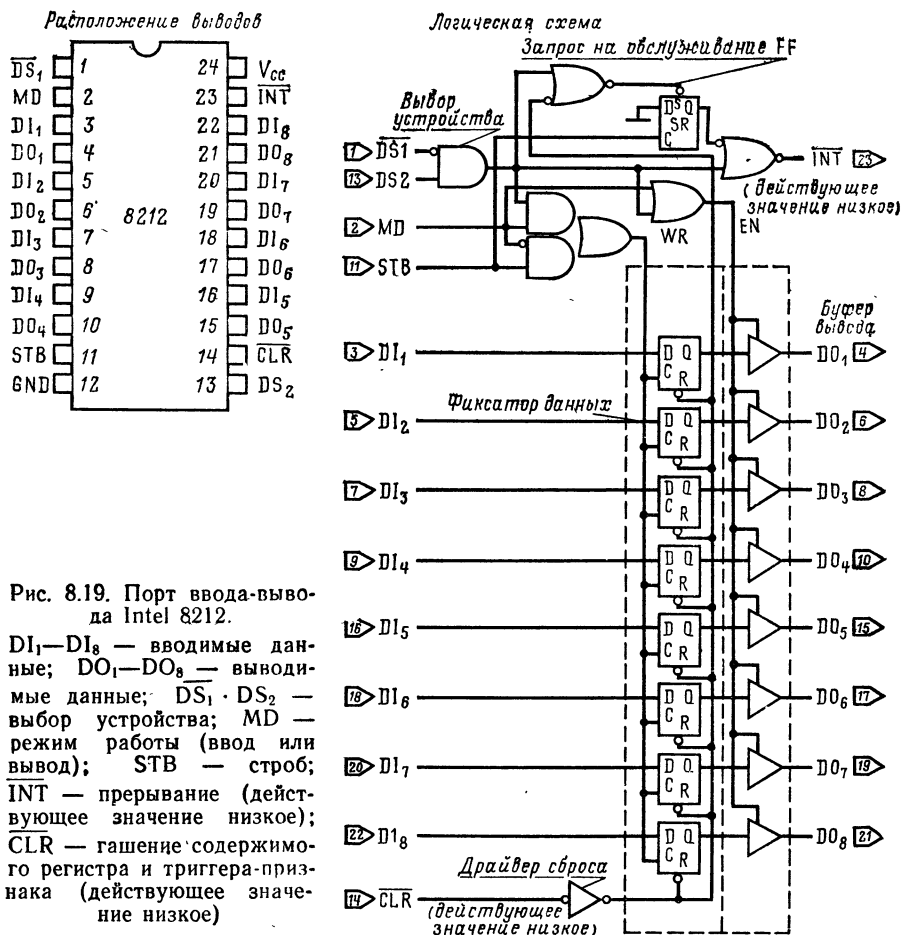


Рис. 8.19. Порт ввода-вывода Intel 8212.

DI_1 — DI_8 — вводимые данные; DO_1 — DO_8 — выводимые данные; DS_1 · DS_2 — выбор устройства; MD — режим работы (ввод или вывод); STB — строб; \overline{INT} — прерывание (действующее значение низкое); CLR — гашение содержимого регистра и триггера-признака (действующее значение низкое)

В режиме вывода ЦП требует (с помощью схемы выбора устройства), чтобы конкретный порт зафиксировал данные. Буфер данных всегда открыт, так что данные сразу становятся доступными для УВВ. При такой структуре управления ЦП помещает данные в порт; процедура выборки не нужна, так как обычно периферийное устройство владеет внешней шиной монопольно, а не в режиме разделения времени.

Устройство Intel 8212 имеет также RS-триггер для фиксации запроса на обслуживание. Этот триггер изображен в верхней части рис. 8.19; его действующее значение соответствует низкому уровню. Триггер очищается строб-сигналом от периферийного устройства. Триггер устанавливается или сигналом «очистить» (CLEAR) или схемой выбора устройства. Типичная последовательность операций, связанная с запросом на обслуживание, следующая:

1. Сигнал системного сброса очищает порт 8212 и устанавливает триггер запроса на обслуживание. Уровень выхода $\overline{\text{INT}}$ (interrupt — прерывание) — высокий.

2. Строб-сигнал (STB) сбрасывает триггер запроса на обслуживание и посылает сигнал $\overline{\text{INT}}$ низкого уровня (действующее значение).

3. Центральный процессор выбирает порт, активируя схему выбора устройства. Эта схема устанавливает триггер запроса на обслуживание, но сохраняет низкий уровень сигнала $\overline{\text{INT}}$ до тех пор, пока порт не будет дезактивирован. После выбора определенного порта уровень сигнала $\overline{\text{INT}}$ становится низким, даже если строб-вход отсоединен; в этом случае $\overline{\text{INT}}$ указывает, что происходит операция ввода-вывода, для которой нужен данный порт. С помощью такого сигнала можно мультиплексировать индикаторы, связанные с этим портом, как показано на рис. 8.17.

Универсальные асинхронный и синхронный приемопередатчики

Для последовательных УВВ нужны схемы, которые генерируют бит четности, осуществляют проверку четности, добавляют биты «старт» и «стоп» для маркировки начала и конца асинхронных передач, а также вырабатывают сигналы управления для стандартных интерфейсов. Скорость передачи данных, способы кодирования и форматы данных столь разнообразны, что обеспечить связь с помощью простых логических схем затруднительно.

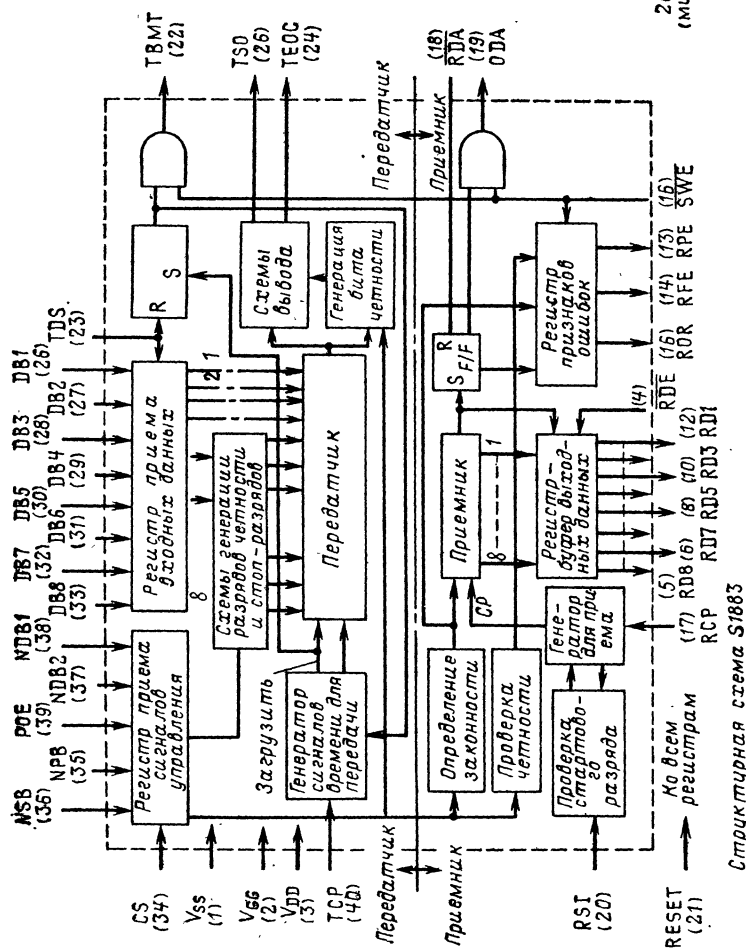
Указанные операции являются стандартными, они строго определены и могут выполняться медленно. Это оправдывает производство специальных МОП-БИС. Для асинхронных УВВ используют универсальный асинхронный приемопередатчик (УАПП), который выполняет следующие функции:

1) преобразует выходные данные из параллельного формата в последовательной, а входные данные — из последовательного в параллельный;

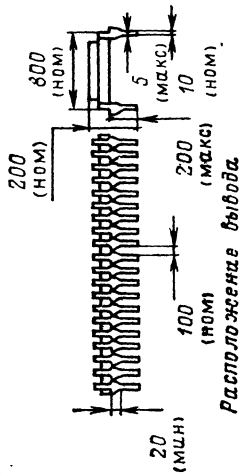
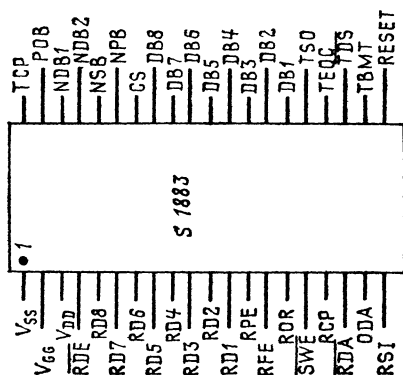
2) работая в качестве передатчика, добавляет биты, маркирующие начало и конец, вырабатывает биты четности и обеспечивает требуемую скорость передачи данных. Работая как приемник, распознает и исключает комбинации битов «старт» и «стоп», производит проверку четности и обеспечивает требуемую скорость приема данных;

3) выдает признаки-индикаторы, показывающие, получил ли он данные или готов ли принять данные для передачи. Другие индикаторы могут указывать на наличие ошибок в полученных данных.

На рис. 8.20 изображен УАПП S1883 фирмы American Microsystems. Это устройство выполняет перечисленные выше функции. Кроме того, скорости приема и передачи у него независимы, выходы имеют три стабильных состояния, осуществляется двойная буферизация. Устройство можно прямо связать с шиной данных, имеющей три состояния. Устройство имеет вход «сброс», с помощью которого производится начальная очистка всех внутренних регистров и счетчиков, регистр хранения управляющей информации, где фиксируется выбран-



Структурная схема S1883



Расположение выводов

Рис. 8.20. Универсальный асинхронный приемопередатчик S1883 фирмы American Microsystems. Характеристики: скорость передачи данных 12,5 Кбайт; длина слова от пяти до восьми разрядов; генерация бита четности и проверка четности (на «четно»; на «нечетно»; «безразлично»); обнаружение ошибок формата и настройки (1; 1,5 или 2 столбцов бита); ввод-вывод с двойной буферизацией; независимость скорости передачи и приема; старт- и стоп-разряды вырабатываются и проверяются; может обмениваться информацией с устройствами TMS6011, COM2017, TR1602, AY-5-1013; выходы с тремя состояниями

ный режим управления, слово состояния, содержащее различные признаки, входы синхронизации, влияющие на скорость передачи. Универсальный асинхронный приемопередатчик имеет два порта ввода и два порта вывода: один порт ввода для принимаемых данных, другой — для слова состояния; один порт вывода для передаваемых данных, другой — для сигналов управления.

Порт управления УАПП определяет следующие характеристики режимов работы устройства:

- число разрядов на символ (обычно от 5 до 9);

- число стоп-разрядов (обычно 1; 1,5 или 2);

- четность (включены ли генерация и проверка четности);

- тип четности (четный или нечетный).

Режимы, требуемые для конкретного применения, могут выбираться при фиксации соответствующих данных в порте управления при инициализации системы.

Порт состояния УАПП сохраняет сигналы «данные готовы» и «вывод готов», а также различные признаки ошибок. Обычно в число сигналов состояния входят следующие:

- буфер передатчика пуст, т. е. УАПП готов для передачи нового символа;

- выходные данные доступны или приемный буфер заполнен, т. е. УАПП получил символ;

- ошибка четности;

- ошибка формата, т. е. УАПП получил символ без нужного числа стоп-разрядов;

- ошибка настройки, т. е. УАПП получил новый символ, прежде чем предыдущий символ был прочитан.

Для работы УАПП необходимо поступление сигналов синхронизации. Такие сигналы могут обеспечить процессор или генератор тактовых импульсов.

Для связи с синхронными периферийными устройствами применяется УСПП. Его работа аналогична работе УАПП, за исключением того, что данные передаются синхронно по отношению к символу синхронизации `sync`. Приемник УСПП сравнивает полученные данные с содержимым внутреннего регистра до тех пор, пока не обнаружит соответствия. После этого прием выполняется синхронно. Передатчик посылает символ `sync`, за которым следует передача в синхронном режиме. Для работы УСПП необходимо поступление сигналов синхронизации. Большинство режимов управления и признаков состояния УАПП имеется и в УСПП. В состав УСПП могут также входить регистры для хранения символов синхронизации и заполнения (`fill character`). Символ заполнения передается в тех случаях, когда никакие другие данные не доступны.

В отличие от указанных выше простых схем, УАПП и УСПП являются БИС, выполненными по МОП-технологии. Эти устройства работают с меньшей скоростью, чем схемы ТТЛ, имеют малые значения выходного тока и требуют иных способов тактирования и подачи питающего напряжения, чем устройства ТТЛ. Изготовители микропроцессоров часто поставляют УАПП и УСПП, которые совместимы с конкрет-

ными процессорами. Тем не менее разработчик должен осторожно использовать УАПП и УСПП в микропроцессорной системе и учитывать их временные характеристики.

Параллельные интерфейсы

Периферийные устройства, работающие с данными в параллельном формате, могут быть связаны с группами портов данных и управления так, как показано выше. Однако реализация передачи данных в параллельном формате представляет собой проблему, которую можно решить и с помощью программируемых устройств большой степени интеграции.

Таким устройствам присущи следующие признаки:

наличие буферов и фиксаторов для входных и выходных данных; возможность вырабатывать сигналы состояния и управления, необходимые для квитирования;

прочие сигналы управления и синхронизации для УВВ;

прямая связь с шинами (адреса данных и управления) процессора;

наличие регистров управления, содержимое которых может изменяться процессором, что позволяет устройствам выполнять различные функции.

Некоторые параллельные интерфейсы имеют средства для управления прерываниями и ПДП, двунаправленные линии, таймеры и даже дополнительную память для программ и данных. Эти устройства гораздо сложнее портов ввода-вывода или УАПП. По сложности логических схем они могут приближаться к процессору. Как было отмечено ранее, сложность параллельных интерфейсных БИС означает, что их легче всего компоновать и использовать, если их можно адресовать как набор ячеек памяти, а не как набор портов ввода-вывода. Широко известен параллельный интерфейс «адаптер интерфейса периферийных устройств» (PIA — Peripheral Interface Adapter) 6820 фирмы Motorola. На рис. 8.21 представлена структурная схема устройства. Адаптер PIA имеет два порта, обозначаемые A и B; порт A предназначен главным образом для ввода, а порт B — для вывода. В каждом порте имеются:

1) регистр данных. Этот регистр работает с фиксацией, если он используется для вывода данных, и без фиксации, если используется для ввода;

2) регистр направлений передачи, содержимое которого определяет, выходными или входными являются линии ввода-вывода;

3) регистр управления, определяющий действующие логические связи в устройстве и хранящий также сигналы «данные готовы» или «УВВ готово»;

4) две линии управления: CA(CB)1 и CA(CB)2, функции которых определяются содержимым регистра управления.

Адаптер PIA занимает четыре адреса в адресном пространстве памяти. Устройство может непосредственно присоединяться к шинам микропроцессора 6800 фирмы Motorola (рис. 8.22). Линии выбора ре-

гистра RS (register select) адресуют внутренние регистры и обычно подсоединены к двум младшим разрядам шины адреса. Линии выбора кристалла CS (chip select) обеспечивают адресацию без использования декодеров (в малых системах). На рис. 8.22 линии A_{15} и A_{14} соединены со входами выбора кристалла, причем действующие значения — низкое для первого входа и высокое для второго; поэтому адресация PIA не мешает ни адресации ОЗУ в пределах нулевой страницы памяти, ни адресации ПЗУ для самых старших адресов (см. обычную структуру памяти для МП Motorola 6800, описанную в гл. 7). На один из входов «выбор кристалла» подается конъюнкция сигналов VMA и A_{14} , а поэтому PIA не может непредвиденным образом изменять свое состояние во время циклов процессора, не использующих память.

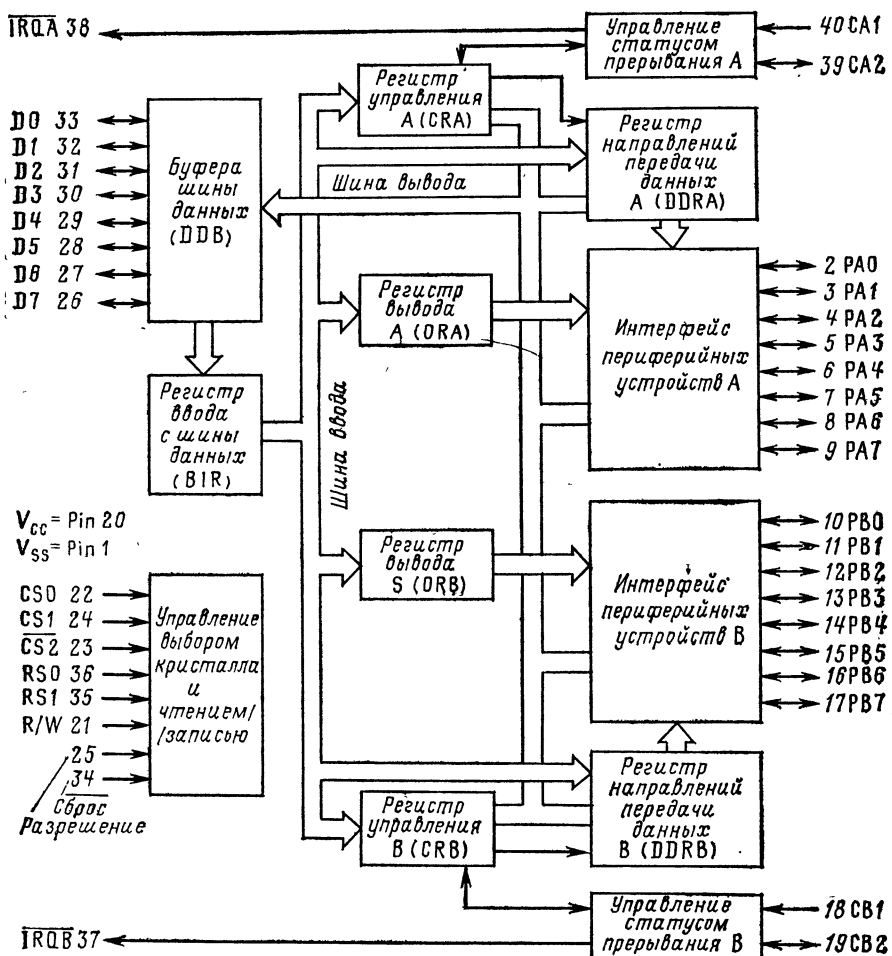
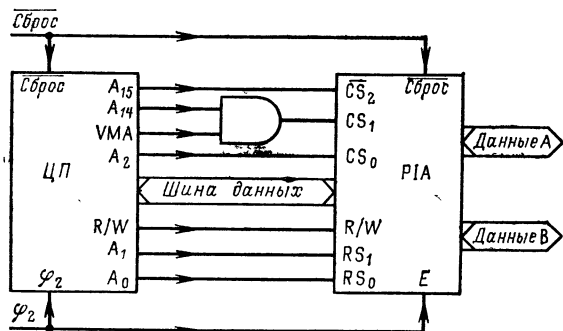


Рис. 8.21. Расширенная структурная схема и расположение выводов адаптера интерфейса периферийных устройств (PIA) Motorola 6820

Рис. 8.22. Интерфейс между процессором Motorola 6800 и адаптером интерфейса периферийных устройств (PIA) Motorola 6820



Адаптер PIA имеет шесть регистров (три с каждой стороны), но только четыре адреса, используемые в соответствии с табл. 8.1. Регистры данных и направлений имеют общий адрес памяти. Выбор между ними осуществляется по значению бита 2 соответствующего регистра управления: этот бит равен 0 для регистра направлений и 1 — для регистра данных. Необходимость такого выбора редко вызывает затруднения, как как в большинстве систем регистр направлений загружается при инициализации системы нужными значениями, которые в дальнейшем редко меняются. Если какой-либо бит в регистре направлений равен 1, то линия ввода-вывода с соответствующим номером работает как выводная, а если 0 — то как вводная.

На рис. 8.23 показано, как организованы регистры управления PIA. Разряды IRQ (interrupt request — запрос на прерывание) — это разряды «готово», устанавливаемые при изменении значений сигналов на линиях управления.

Эти разряды не могут быть изменены путем записи данных в регистр управления. Они автоматически сбрасываются любой операцией, считывающей данные из регистра данных PIA. Дизъюнкция значений этих разрядов может быть использована как сигнал «прерывание».

Таблица 8.1. Внутренняя адресация адаптера интерфейса периферийных устройств

RS1	RS0	Биты регистра управления		Определяемый регистр
		CRA-2	CRB-2	
0	0	1	X	Регистр периферийного устройства A
0	0	0	X	Регистр направлений передачи A
0	1	X	X	Регистр управления A
1	0	X	1	Регистр периферийного устройства B
1	0	X	0	Регистр направлений передачи B
1	1	X	X	Регистр управления B

Примечание. X — любое значение.

	7	6	5	4	3	2	1	0
CRA	IRQA1	IRQA2	Управление CA2			Обращение к регистру управления пересылкой данных A	Управление CA1	
	7	6	5	4	3	2	1	0
CRB	IRQB1	IRQA2	Управление CB2			Обращение к регистру управления пересылкой данных B	Управление CA2	

Рис. 8.23 Формат управляющего слова регистра управления PIA

Бит 0 регистра управления PIA определяет, разрешена ли работа линий прерывания (если его значение 1, то прерывание разрешено). Бит 1 регистра управления определяет, по фронту или по срезу сигнала на линии управления 1 должен быть установлен признак прерывания (CRA-7). Использование битов 0 и 1 описывается в табл. 8.2. Бит 1 допускает строб-сигналы любой полярности и с переключением по любому фронту.

Таблица 8.2. Управление входами запросов на прерывание CA1 и CB1

CRA-1 (CRB-1)	CRA-0 (CRB-0)	Вход прерывания CA1 (CB1)	Признак прерывания CRA-7 (CRB-7)	Запрос на прерывание IRQA (IRQB)
0	0	↓ Действующее значение	Устанавливается высоким по ↓ на CA1 (CB1)	Запрещено — потенциал IRQ остается высоким
0	1	↓ То же	То же по ↓	Становится низким, когда признак прерывания CRA-7 (CRB-7) становится высоким
1	0	↑ » »	по ↑ на CA1 (CB1)	Запрещено — потенциал IRQ остается высоким
1	1	↑ » »	по ↑ на CA1 (CB1)	Становится низким, когда признак прерывания CRA-7 (CRB-7) становится высоким

Примечания. 1. ↑ — переход в положительном направлении (от низкого значения к высокому). 2. ↓ — переход в отрицательном направлении (от высокого значения к низкому). 3. Признак прерывания (CRA-7) сбрасывается при чтении данных микропроцессором из регистра данных A; CRB-7 сбрасывается при чтении данных микропроцессором из регистра данных B. 4. Если потенциал на CRA-0 (CRB-0) в момент возникновения запроса на прерывание (при запрещенном прерывании) низкий, а затем становится высоким, то сигнал IRQA (IRQB) выдается при положительном переходе на линии CRA-0 (CRB-0).

Таблица 8.3. Управление входами запросов на прерывание CA2 и CB2 (потенциал линий на CRA-5 (CRB-5) низкий)

CRA-5 (CRB-5)	CRA-4 (CRB-4)	CRA-3 (CRB-3)	Вход прерывания CA2 (CB2)	Признак прерывания CRA-6 (CRB-6)	Запрос на прерывание IRQA (IRQB)
0	0	0	↓ Действующее значение	Устанавливается высоким по ↓ на CA2 (CB2)	Запрещено — IRQ остается высоким
0	0	1	↓ То же	То же по ↓ на CA2 (CB2)	Становится низким, когда признак прерывания CRA-6 (CRB-6) становится высоким
0	1	0	↑ То же	То же по ↑ на CA2 (CB2)	Запрещено — IRQ остается высоким
0	1	1	↑ То же	То же по ↑ на CA2 (CB2)	Становится низким, когда признак прерывания CRA-6 (CRB-6) становится высоким

Примечания: 1. ↑ — переход в положительном направлении (от низкого значения к высокому). 2. ↓ — переход в отрицательном направлении (от высокого значения к низкому). 3. Признак прерывания (CRA-6) сбрасывается при чтении данных микропроцессором из регистра данных A, CRB-6 сбрасывается при чтении данных процессором из регистра данных B. 4. Если потенциал на CRA-3 (CRB-3) в момент возникновения прерывания (при запрещенном прерывании) низкий, а позже становится высоким, то сигнал IRQA (IRQB) выдается при положительном переходе на линии CRA-3 (CRB 3).

С помощью разрядов 3—5 регистра управления можно изменять выполняемые PIA функции. Разряд 5 определяет, входной или выходной является линия управления 2 (CA2 или CB2): если его содержимое равно 0, то линия 2 входная. В этом случае стороны A и B PIA действуют одинаково. Управляющий разряд 6 устанавливается при переходе на линии управления 2. Управляющие разряды 3 и 4 определяют, принимает ли выходной сигнал действующее значение, а также тип перехода, по которому устанавливается разряд 6. Линия управления 2 может передавать дополнительный строб-сигнал от одиночного УВВ или строб-сигнал от группы мультиплексированных УВВ, использующих один и тот же порт. В табл. 8.3 описывается использование линий CA2 и CB2 в качестве входных.

Если бит 5 принимает значение 1, то линия управления 2 — выходная. В этом случае стороны A и B отличаются друг от друга. Сигнал по линии CA2 выполняет функции строба чтения, а на линии CB2 — строба записи. Использование линии CB2 описано в табл. 8.4. Если бит 4 равен 0, то потенциал на CB2 становится низким при положительном переходе первого сигнала «разрешение» (ENABLE), следующего за тем циклом, в котором ЦП записывает данные в регистр данных B. Управляющий бит 3 определяет условие окончания строб-сигнала. Если бит 3 принимает значение 1, то строб-сигнал длится

только до следующего сигнала «разрешение», если — значение 0, то строб-сигнал продолжается до тех пор, пока его не прервет действующее значение перехода на *CB1*. Возможные виды сигнала: короткий строб, по которому УВВ зафиксирует данные, или более длительный сигнал подтверждения (при обмене с квитирующим), который длится до начала новой передачи данных периферийным устройством. Линия «разрешение» в большинстве случаев непосредственно соединена с системным генератором тактовых сигналов.

Если бит 4 принимает значение 1, то *CB2* — выходной сигнал с фиксированным уровнем, равным значению бита 3. Он может использоваться для выбора режима работы со словом состояния или для включения и выключения устройства. Нет необходимости в отдельном последовательном порте управления.

В табл. 8.5 приведено использование линии *CA2* для строб-сигнала чтения, уровень которого становится низким после чтения процессором данных из регистра данных *A*. Зафиксированный в управляющих разрядах режим остается тем же, что и для линии *CB2*. Можно программным способом создать строб чтения на стороне *B* (или строб за-

Таблица 8.4. Управление линией *CB2*, используемой в качестве выхода (линия *CRB-5* имеет высокий потенциал)

CRB-5	CRB-4	CRB-3	Сигнал на <i>CB2</i>	
			сброшен	установлен
1	0	0	Низкий потенциал по положительному переходу первого сигнала <i>E</i> («разрешение»), следующего за операцией процессора «запись в регистр данных <i>B</i> »	Высокий потенциал, когда признак прерывания <i>CRB-7</i> установлен действующим значением перехода на <i>CB1</i>
1	0	1	То же	Высокий потенциал по положительному переходу следующего сигнала <i>E</i> («разрешение»)
1	1	0	Низкий потенциал, когда уровень <i>CRB-3</i> становится низким в результате операции процессора «запись в регистр управления <i>B</i> »	Потенциал всегда низкий, пока <i>CRB-3</i> имеет низкий уровень. Станет высоким в результате операции процессора «запись в регистр управления <i>B</i> », если эта операция изменит значение <i>CRB-3</i> на 1
1	1	1	Потенциал всегда высокий, пока <i>CRB-3</i> имеет высокий уровень. Будет сброшен, если операция процессора «Запись в регистр управления <i>B</i> » приведет к сбросу <i>CRB-3</i> в 0	Высокий потенциал, когда уровень <i>CRB-3</i> становится высоким в результате операции процессора «запись в регистр управления <i>B</i> »

Таблица 8.5. Управление линией CA2, используемой в качестве выхода (линия CRA-5 имеет высокий потенциал)

CRA-5	CRA-4	CRA-3	Сигнал на CA2	
			сброшен	установлен
1	0	0	Низкий потенциал по отрицательному переходу сигнала Е («разрешение») после операции процессора «чтение данных А»	Высокий потенциал по действующему значению перехода на CA1
1	0	1	Низкий потенциал после операции процессора «чтение данных А»	Высокий потенциал по срезу следующего сигнала Е («разрешение»)
1	1	0	Низкий потенциал, когда уровень CRA-3 становится низким в результате операции процессора «запись в регистр управления А»	Потенциал всегда низкий, пока CRA-3 имеет низкий уровень
1	1	1	Потенциал всегда высокий, пока CRA-3 имеет высокий уровень	Высокий потенциал, когда уровень CRA-3 становится высоким в результате операции процессора «запись в регистр управления А»

писи со стороны А) путем выполнения фиктивной операции записи после операции чтения на стороне В (соответственно фиктивной операции чтения после операции записи на стороне А). Фиктивные операции не должны менять содержимого ни одного из регистров. Операции чтения данных со стороны В или записи их со стороны А не вырабатывают строб-сигнал.

При поступлении сигнала на вход «сброс» очищаются все регистры PIA. Все линии данных и управления первоначально являются входными, все прерывания запрещены и выбран регистр направлений передачи. Программа должна настроить PIA, исходя из этого начального состояния (см. примеры в § 8.6).

Линии данных у разных сторон отличаются. На стороне В имеется буфер с тремя состояниями, обеспечивающий высокую нагрузочную мощность и позволяющий правильно считывать данные с выходных линий. На стороне А нет буфера и правильное чтение с выходных линий невозможно, если эти линии не очень слабо нагружены или не буферизованы внешним образом.

Хотя PIA состоит из одного корпуса, применение этого устройства может быть весьма разнообразным. Адаптер PIA может обеспечить обмен с квити́рованием, управление прерываниями и выполнение других функций, что при использовании других устройств потребовало бы нескольких портов и других схем.

8.5. УСТРОЙСТВА ВВОДА-ВЫВОДА

В этом параграфе описываются простые устройства ввода-вывода и интерфейсы между ними и микропроцессором: переключатели, клавишные пульты, индикаторы, аналого-цифровые и цифро-аналоговые преобразователи и т.д. Более сложные устройства требуют и более сложного интерфейса, но многие положения, справедливые для простых устройств, сохраняются и для более сложных.

В данном параграфе также обсуждаются распространенные интерфейсы EIA RS-232 и IEEE-488.

Простые переключатели

Простейшим из возможных устройств ввода является кнопка, при нажатии которой включается короткая логическая цепь или обнуляется один разряд. Соединение кнопки с микропроцессором показано на рис. 8.24. Если кнопка не нажата, то потенциал на выходе этого простейшего УВВ высокий благодаря наличию резистора, находящегося под высоким потенциалом. Кнопка — это однополюсный однопозиционный переключатель.

Кнопочный переключатель легко подсоединить к микро-ЭВМ. В качестве интерфейса нужен только адресуемый буфер с тремя состояниями (рис. 8.24). Фиксатор сигнала не нужен, так как изменение положения кнопки происходит в точки зрения быстрого действия ЭВМ) очень медленно.

Можно легко определить, в каком положении находится кнопка. Центральный процессор читает содержимое буфера, т. е. передает это содержимое через шину данных в процессор. Затем выполняется команда логического умножения (AND) над данными и маской, содержащей 1 в разряде, соответствующем линии шины данных, с которой соединена кнопка. Если этот разряд содержит 0, то кнопка нажата. Если кнопка соединена с младшим или старшим разрядом шины данных, то с помощью команды сдвига можно содержимое этого разряда поместить в триггер «перенос» (CARRY) и команда AND не понадобится. Если доступно значение знакового бита (SIGN), то его также можно использовать для определения положения кнопки.

Единственная проблема при использовании кнопочного переключателя состоит в том, что замыкание с помощью механического пере-

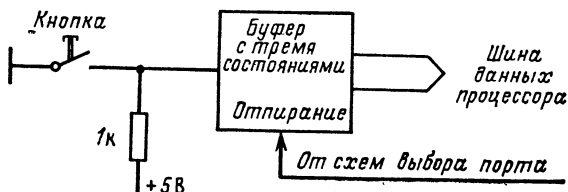


Рис. 8.24. Соединение простейшего УВВ (кнопка) с микропроцессором (система декодирования открывает тристабильный буфер только на время операции ввода, запрашивающей данный порт)

ключателя не обеспечивает четкого фронта сигнала. Напряжения, т. е. цепь нерегулярно замыкается и размыкается до тех пор, пока не установится надежный контакт. Длительность этого периода зависит от конкретного переключателя, но обычно не превышает 2 мс. Нестабильность сигнала можно устранять («демпфировать») двумя способами:

1) программным. Когда ЦП обнаружит контакт, то программа выполняет специальный цикл задержки в течение времени, достаточного для прекращения скачков напряжения. Задержку можно осуществить с помощью программы, изображенной на рис. 8.5, или предоставляя процессору возможность выполнять другую работу в течение требуемого времени;

2) аппаратным (с использованием одновибратора). Длительность импульса одновибратора должна превосходить период неустойчивости, чтобы выходной сигнал представлял собой одиночный импульс.

Определить, изменилось ли положение кнопки, просто, если воспользоваться командой ИСКЛЮЧАЮЩЕЕ ИЛИ. Это выполняется следующим образом:

Шаг 1. Запомнить старое состояние кнопки.

Шаг 2. Прочитать новое состояние кнопки и выполнить команду ИСКЛЮЧАЮЩЕЕ ИЛИ с аргументами «новое состояние» и «старое состояние».

Шаг 3. Если результат не равен 0, то положение кнопки изменилось.

С помощью команды ИСКЛЮЧАЮЩЕЕ ИЛИ можно также идентифицировать событие, состоящее в том, что нажата хотя бы одна из нескольких кнопок, подсоединенных к одному порту ввода. Результат операции ИСКЛЮЧАЮЩЕЕ ИЛИ, вторым аргументом которой является слово, состоящее из одних единиц, равен нулю, если не была нажата ни одна кнопка.

Обычный однополюсный двухпозиционный переключатель устроен немного сложнее. Он имеет общий вывод (common lead) и выводы NO (нормально открыт — normal open), NC (нормально закрыт — normally closed). Этот переключатель может быть присоединен к процессору через буфер с тремя состояниями. Механические переключатели такого типа можно демпфировать с помощью одновибратора, двух схем И—НЕ (рис. 8.25) или программным способом. Перекрестно соединенные схемы И—НЕ работают следующим образом. Если переключатель замкнут, то заземлен вывод NC, а поэтому на выходе первой схемы

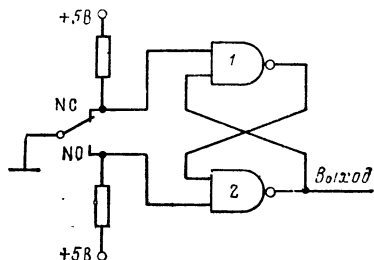


Рис. 8.25. Устранение неустойчивости сигнала от однополюсного двухпозиционного переключателя с помощью перекрестно соединенных схем И—НЕ

И—НЕ — высокий потенциал (так как на одном из ее входов имеется низкий потенциал). Вывод NO имеет высокий потенциал, поэтому на выходе второй схемы И—НЕ — низкий потенциал (оба ее входа имеют высокий потенциал). Если переключатель разомкнут, то NO заземлен и на выходе второй схемы устанавливается высокий потенциал, на выходе первой — низкий. В промежуточных (нестабильных) состояниях, когда как NC, так и NO имеют высокий потенциал, потенциал на выходе схемы не меняется.

Предположим, что программа должна обнаружить изменение положения переключателя. В данном случае снова можно воспользоваться командой ИСКЛЮЧАЮЩЕЕ ИЛИ. В результате ее применения к аргументам «старое состояние» и «новое состояние» изменится содержимое битов, соответствующих переключателям, изменившим свое положение.

Многопозиционные переключатели

Поворотные переключатели, наборные диски и селекторные переключатели имеют много возможных положений. Когда переключатель находится в какой-либо определенной позиции, соответствующий вывод заземляется через общую линию. Такие переключатели часто входят в состав электронного оборудования.

Если переключатель не имеет собственных средств для кодирования информации о своем положении, то каждый его вывод должен соединяться с шиной данных через буфер с тремя состояниями. Программа может определить, в каком положении находится переключатель, путем сдвига слова данных до тех пор, пока разряд, равный нулю, не окажется в триггере переноса. Число потребовавшихся сдвигов идентифицирует положение переключателя. На рис. 8.26 приведена блок-схема указанного метода. Если разряд с нулевым значением не найден, то переключатель находится в промежуточном положении и должен быть опрошен позже. Следует обратить внимание на то, что при использовании 8-разрядного процессора для 10-позиционного переключателя необходимы два порта данных и некоторые операции над словами двойной длины.

Очевидно, что такой интерфейс неэкономичен, так как с помощью 10 бит описывается только десять возможных положений. Для определения положения переключателя, кодирующего информацию о своем положении в BCD-коде или в шестнадцатиричном коде, требуется только четыре

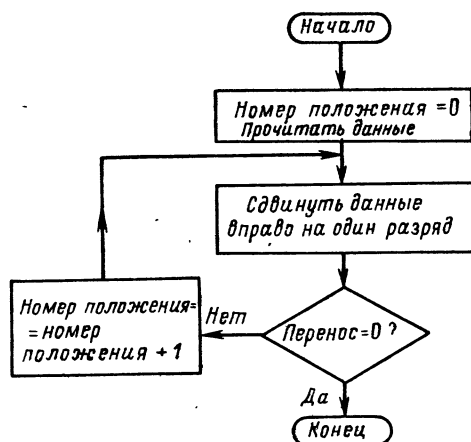


Рис. 8.26. Блок-схема программы, определяющей положение селекторного переключателя

линии ввода. При этом входное слово идентифицирует положение переключателя без дополнительной обработки. Для связи процессора с таким переключателем требуется только буфер с тремя состояниями.

Клавиатура

Клавиатура — набор переключателей, каждый из которых может соединяться с портом ввода независимо. Тогда интерфейс с ЦП и программирование ввода данных осуществляются таким же образом, как и для любого набора переключателей. Программные средства должны выполнять следующие функции:

1) определять, была ли нажата хотя бы одна клавиша. Если клавиши — однополюсные однопозиционные переключатели, то это означает проверку на наличие хотя бы одного нуля на входах. Для этой проверки можно воспользоваться сравнением со словом, состоящим из одних единиц;

2) если одна из клавиш нажата, определять, какая именно. Для реализации этой функции выполнить последовательность сдвигов такую, как на рис. 8.26;

3) выбрать действие, соответствующее нажатой клавише. Этот выбор зависит от назначения клавиатуры. С помощью клавиш могут определяться входные данные, режимы, операции управления, команды, масштабные множители и т. д. Один из методов выбора таков: использовать таблицу (называемую таблицей переходов), в которой содержатся адреса подпрограмм, выполняющих действия, соответствующие различным клавишам. Адрес, выбираемый из таблицы, определяется положением клавиши. На рис. 8.27 приведена блок-схема метода таблицы переходов.

При использовании метода таблицы переходов для независимо подключаемых (без кодирования) клавиш необходимо большое число входных линий и выполнение многих операций с блоками, состоящими из более чем одного слова. Например, для такой клавиатуры, как у телетайпа с 64 клавишами, требуется 64 линии и выполнение последовательности из восьми операций над 8 разрядными словами. На поиск нажатой клавиши расходуется значительное время. Независимое подсоединение переключателей обычно применяется только для малых клавиатур (до 16 клавиш).

Указанная проблема решается с помощью кодирования. Для больших клавиатур можно использовать кодирующие устройства ТТЛ. Для клавиатур большего объема обычно используют МОП-схемы, так как высокая скорость кодирования не требуется. МОП-кодеры имеют ПЗУ. Для обычных форматов (таких, как у телетайпа) выпускаются готовые кодеры, а для форматов пользователя нужна специальная маска или ППЗУ, т. е. МОП-кодеры не только реагируют на нажатие клавиши, но и кодируют сигнал от нее. Большинство из них управляют перекрытием (rollover) (разделяют одновременное нажатие клавиш и интерпретируют нажатия последовательно), демпфируют сигналы от клавиш, осуществляют выбор режима (например: «верхний регистр» или «нижний регистр»), вырабатывают строб-сигнал, который

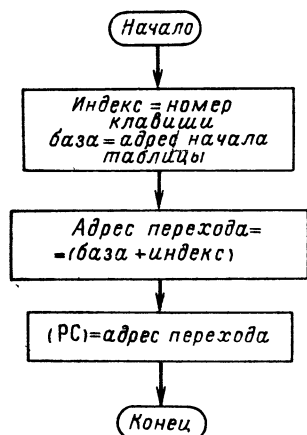


Таблица переходов, начинающаяся с адреса JТАВ, организована следующим образом:

Адрес	Содержимое
JТАВ	Адрес программы обработки сигнала от клавиши 0
JТАВ+1	Адрес программы обработки сигнала от клавиши 1
⋮	⋮
JТАВ+n	Адрес программы обработки сигнала от клавиши n

Для 8-разрядного микропроцессора с 16-разрядной адресацией каждая строка таблицы занимает два слова.

Рис. 8.27. Блок-схема программы, использующей таблицу переходов

может действовать как сигнал «данные готовы» и фиксировать данные в порте ввода.

Кодирующая клавиатура распознает и идентифицирует нажатие клавиш, но выбор действия остается за программой. Метод таблицы переходов можно использовать, если число различных функций велико. Многие программы могут использовать одну клавиатуру в различных режимах; с клавиатуры можно вводить как данные, так и команды. В этом случае вводимые данные следует сохранять, а команды — интерпретировать.

Декодирование можно полностью или частично выполнять программными средствами. Для организации ввода-вывода требуется меньше средств, если клавиатура организована в виде матрицы, представленной на рис. 8.28. Поиск нажатой клавиши производится с помощью заземления линий (строк или столбцов) через порт вывода. Процедура поиска происходит следующим образом:

Шаг 1. Определить, нажата ли хотя бы одна клавиша. Заземлить все линии столбцов, фиксируя в порте вывода нулевые значения разрядов. Если среди линий строк окажутся заземленные, то существует контакт между строками и столбцами, т. е. некоторые клавиши нажаты. Подавая сигналы с линий строк на входы схем И—НЕ, можно выработать строб-сигнал с высоким действующим значением.

Шаг 2. Определить, какая клавиша нажата.

Если программа обнаружила контакт, она может идентифицировать нажатую клавишу, заземляя столбцы по одному. Если нажатая клавиша не находится в данном столбце, то все входы с линий строк будут единичными. Если нажатая клавиша находится в данном столбце, то среди входов будет нуль в позиции, соответствующей нужной строке. В табл. 8.6 показаны сочетания состояний строк и столбцов, определяющие одну из клавиш, изображенных на рис. 8.28.

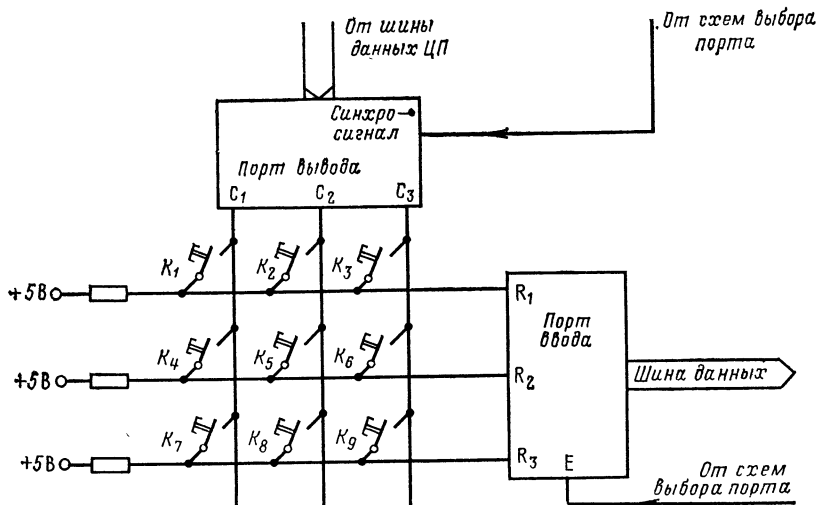


Рис. 8.28. Интерфейс между клавиатурой, организованной по матричному принципу, и микропроцессором (порт вывода, к которому присоединяются линии столбцов, должен иметь фиксатор, в то время как для порта вывода необходим только буфер с тремя состояниями)

Блок-схема на рис. 8.29 описывает процедуру поиска нажатой клавиши. Для поиска клавиши требуется дополнительное время, но если в матрице m строк и n столбцов, то для всей клавиатуры достаточно иметь m входных и n выходных линий, а не $m \times n$ линий, используемых при независимом подсоединении клавиш. Однако при использовании описанного метода имеются свои проблемы. Перечислим некоторые из них:

- 1) среднее число просмотров столбцов растет линейно с ростом числа столбцов;
- 2) поиск и проверка становятся длительными и громоздкими, если число строк или столбцов превышает длину слова ЦП;
- 3) необходимы специальные процедуры для обработки ситуации, когда контакт исчезает до его обнаружения процедурой поиска.

Таблица 8.6. Идентификация клавиши с помощью стробирования столбцов

Клавиша	Заземленный столбец (входной сигнал для клавиатуры)	Заземленная строка (выходной сигнал клавиатуры)	Клавиша	Заземленный столбец (входной сигнал для клавиатуры)	Заземленная строка (выходной сигнал клавиатуры)
K ₁	C ₁	R ₁	K ₆	C ₃	R ₂
K ₂	C ₂	R ₁	K ₇	C ₁	R ₃
K ₃	C ₃	R ₁	K ₈	C ₂	R ₃
K ₄	C ₁	R ₂	K ₉	C ₃	R ₃
K ₅	C ₂	R ₂			

Во многих простых устройствах (например, в карманных калькуляторах) производится постоянное сканирование клавиатуры в ожидании нажатия клавиши.

Существует и другая процедура, в которой используется устройство, подобное адаптеру интерфейса периферийных устройств (PIA), т. е. устройство, линии ввода-вывода которого можно программно настроить на ввод или вывод. В этом случае для определения нажатой клавиши необходимы следующие шаги: заземлить все столбцы и сохранить входы с линий строк; заземлить все строки и сохранить входы с линий столбцов. Сочетание данных, полученных от линий строк и столбцов, идентифицирует положение клавиши в соответствии с табл. 8.7. На рис. 8.30 приведена блок-схема этого метода. Нет необходимости в сканировании столбцов и в поиске строки, но зато, чтобы идентифицировать клавишу, ЦП должен просмотреть таблицу возможных входов.

Имеется еще ряд проблем, связанных с интерфейсом между клавиатурой и процессором. Программными или аппаратными средствами не-

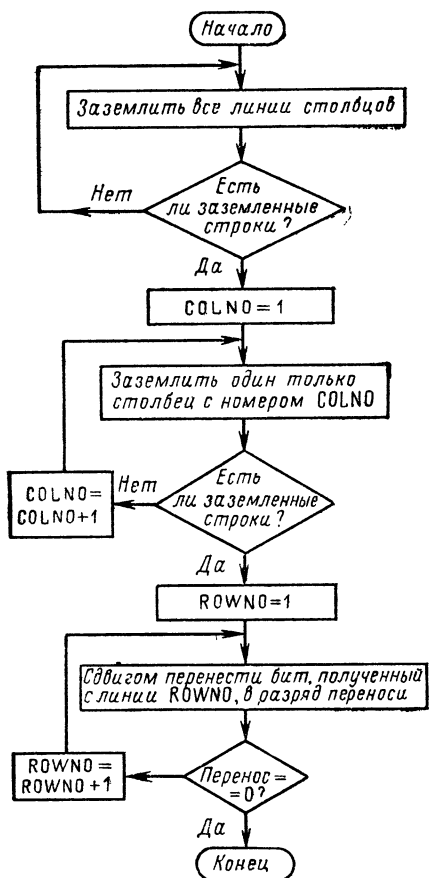


Рис. 8.29. Блок-схема процедуры поиска нажатой клавиши

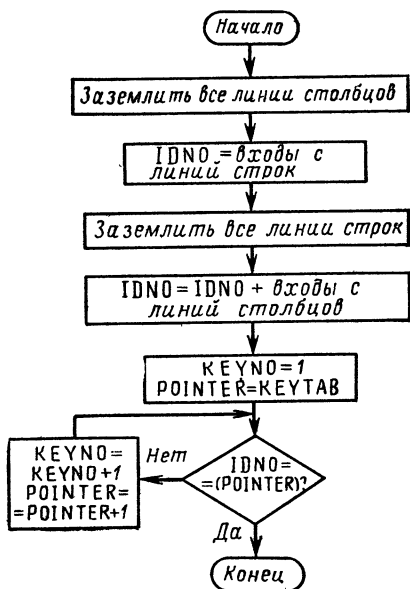


Рис. 8.30. Блок-схема процедуры идентификации нажатой клавиши с использованием чередования режимов ввода и вывода

Таблица 8.7. Идентификация клавиши при чередовании режимов ввода и вывода

Клавиша	Выходы с линий столбцов (линии строк заземлены)			Выходы с линий строк (линии столбцов заземлены)			Шестнадцатирочный код (ООС, С ₂ , С ₄ , R ₂ , R ₁ , R ₁)
	C ₂	C ₃	C ₁	R ₂	R ₁	R ₁	
K ₁	1	1	0	1	1	0	38
K ₂	1	0	1	1	1	0	2B
K ₃	0	1	1	1	1	0	1E
K ₄	1	1	0	1	0	1	35
K ₅	1	0	1	1	0	1	2D
K ₆	0	1	1	1	0	1	1D
K ₇	1	1	0	0	1	1	33
K ₈	1	0	1	0	1	1	2B
K ₉	0	1	1	0	1	1	1B

обходимо демпфировать сигналы от механических переключателей, обрабатывать одновременные нажатия клавиш. Процессор должен отличать ранее возникающий контакт от более позднего. Строб-сигнал может идентифицировать новые данные только тогда, когда строб будет сбрасываться при чтении данных процессором. Этот сброс выполняется автоматически портом ввода-вывода 8212 и адаптером интерфейса периферийных устройств (PIA). Если строга вообще нет, то процессор может просто ожидать окончания контакта. Процедура сканирования клавиатуры совпадает с первым шагом блок-схемы на рис. 8.29, за исключением того, что условия перехода инвертируются. В этой ситуации пользователь должен делать паузу между нажатиями клавиш, так как в противном случае система не будет реагировать на его запросы.

Простые индикаторные устройства

Отдельный светоизлучающий элемент — простейшее устройство вывода. В настоящее время наиболее широко распространены так называемые светоизлучающие диоды (светодиоды). Эти приборы испускают свет, когда они включены в положительном направлении, т. е. когда их анод имеет положительный потенциал по отношению к катоду. Распространенность этих приборов объясняется их малыми размерами, низкими значениями тока и напряжения питания, длительным сроком безотказной работы, малым значением мощности рассеяния и низкой стоимостью. Светодиоды легко мультиплексировать, т. е. один порт может управлять многими индикаторами. К недостаткам светодиодов можно отнести их малую яркость, чувствительность к изменениям температуры и малое разнообразие цветов. Другие индикаторы: на жидких кристаллах, газоразрядные, с нитью накаливания, флуоресцентные также являются полезными и распространенными приборами, но в данной книге они не рассматриваются.

На рис. 8.31 изображена цепь светодиода. Резистор ограничивает ток, проходящий через диод, значением около 10 мА. Разумеется,

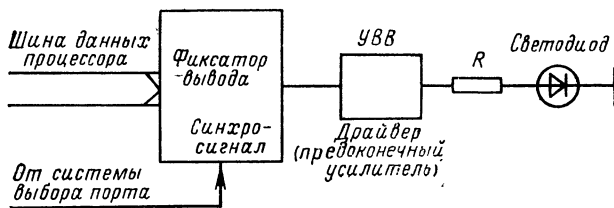


Рис. 8.31. Интерфейс между микропроцессором и светоизлучающим диодом (управляющее напряжение может подаваться также и на катод. Для этого нужно либо инвертировать логические уровни выхода, либо использовать инвертирующий предоконечный усилитель)

ЭВМ может подать сигнал на любой из выводов диода. Если к аноду подводится постоянный потенциал $+5\text{ В}$, то диод светится, когда ЭВМ подает логический ноль на катод; если катод всегда заземлен, то диод светится, когда ЭВМ подает логическую единицу на анод.

Светодиод наилучшим образом работает при токе, поступающем импульсами, так как излучение индикатора сильно растет с ростом тока, а время нарастания яркости мало. В стандартных системах на светодиод подаются импульсы длительностью несколько миллисекунд напряжением $10\text{—}15\text{ В}$, а стандартные частоты — от 100 до 500 Гц . Большинство портов ввода-вывода не может непосредственно нагружаться на светодиод, поэтому необходимо использовать периферийные предоконечные усилители или транзисторы.

Длительностью и периодом импульсов (и, следовательно, яркостью свечения) можно управлять как аппаратным, так и программным способом. Программное управление предусматривает процедуры задержки. При аппаратном способе управления для генерации импульсов требуется таймер.

Если частота поступления импульсов мала, то индикатор не будет казаться светящимся постоянно. Если частота сигналов лежит в пределах от 10 до 50 Гц , то будут заметны периодические вспышки индикатора. С помощью мигающего индикатора можно указывать неисправность, ошибочный вход, переполнение, истощение источника питания и другие особые состояния.

Управлять индикатором можно простыми программными средствами. От полярности включения индикатора зависит, по какому логическому значению возбуждается свечение. Программа может управлять состоянием выходного разряда следующим образом:

1) с помощью команды логического сложения (ИЛИ) с маской, имеющей 1 в нужной позиции.

Например, пятый бит 8-разрядного слова можно установить, применяя дизъюнкции с двоичным словом 00100000 ;

2) с помощью команды логического умножения (И) с маской, имеющей 0 в нужной позиции.

Например, третий бит 8-разрядного слова можно сбросить, применяя конъюнкции с двоичным словом 11110111 ;

3) инвертировать значения разряда с помощью команды ИСКЛЮЧАЮЩЕЕ ИЛИ и маски, имеющей 1 в нужной позиции.

Например, инвертированное значение второго бита 8-разрядного слова можно получить путем сложения (по модулю два) с двоичным словом 00000100.

Для иллюстрации сказанного воспользуемся интерфейсом, приведенным на рис. 8.31 (при появлении на выходе 1 индикатор загорается). По команде ИЛИ индикатор загорается, по команде И — гасится, а по команде ИСКЛЮЧАЮЩЕЕ ИЛИ предыдущее состояние меняется на противоположное.

Многосегментные индикаторы

С помощью многосегментных индикаторов можно изображать цифры, буквы алфавита и другие символы. На рис. 8.32 показаны обычные индикаторы, находящие применение в приборах, терминалах, калькуляторах и измерительной аппаратуре. Семисегментные индикаторы особенно широко распространены, так как они имеют наименьшее число независимо управляемых элементов, с помощью которых все еще можно с некоторым разумным упрощением воспроизводить десятичные цифры и некоторые другие символы.

Индикатор можно собрать из сегментов двумя способами. При первом способе (индикатор с общим катодом) все катоды заземляются; при сигнале 1 на аноде индикатор загорается. При втором способе (с общим анодом) все аноды связаны с источником питания; при сигнале 0 на катоде индикатор загорается. Основная проблема состоит в том, как преобразовать данные в требуемую форму. Это преобразование можно осуществить как программным, так и аппаратным способом. Стандартные аппаратные средства — специальные декодеры, хранимые в ППЗУ таблицы и стандартные ПЗУ, такие как генератор символов ASCII. Аппаратный способ требует применения более сложных схем, но меньше программного обеспечения. Программный способ декодирования ЦП может реализовать путем поиска в таблице. Вось-

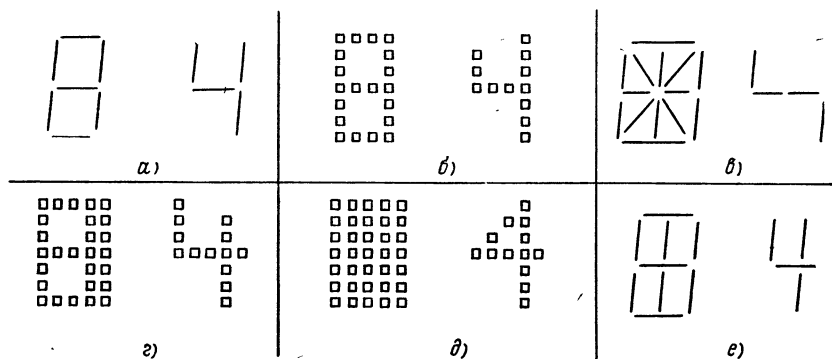


Рис. 8.32. Типы многосегментных индикаторов:

а — 7-сегментный; б — сокращенная точечная матрица; в — 14-сегментный; г — сокращенная точечная матрица; д — точечная матрица 5×7; е — 9-сегментный

миразрядный процессор легко может выполнять преобразование данных для семисегментного индикатора, но при 9 или 14 сегментах для осуществления преобразования необходимо выполнять операции над **двойными словами**¹. Аппаратура или программа должны определять длительность и частоту сигналов, как и для односегментных индикаторов. Если индикаторы не требуют для своей работы очень длинных импульсов, то индикаторами можно управлять мультиплексно с помощью одного порта, как показано на рис. 8.17.

Цифро-аналоговые преобразователи

Цифро-аналоговые преобразователи (ЦАП) преобразуют цифровые выходные данные в непрерывный, или аналоговый, сигнал. Такой сигнал необходим для управления двигателями, соленоидами, реле, графопостроителями и другими УВВ. Наиболее распространенные ЦАП имеют схемы со взвешенными сопротивлениями резисторов и резисторно-цепные схемы². У большинства преобразователей соответствие входного кода выходному напряжению фиксировано, но в ЦАП с умножением допускается изменение этого соответствия. Выходной сигнал ЦАП с умножением равен произведению значения цифрового сигнала на входе на эталонное напряжение.

Выпускаются ЦАП, разработанные специально для микро-ЭВМ. Обычно эти устройства имеют адресуемые входные буфера, через которые устройство можно загрузить 8-разрядным словом. Примером может служить 10-разрядный ЦАП AD7522 фирмы Analog Devices, изображенный на рис. 8.33.

Микропроцессор может либо загрузить ЦАП за одну операцию, либо загрузить 2 байта входного кода для ЦАП раздельно. Для выполнения загрузки за одну операцию необходимо, чтобы один и тот же управляющий сигнал отпирали оба буфера. Раздельная загрузка 2 байт требует, чтобы каждый из двух буферов отпирался по своему сигналу управления, т. е. старший и младший байты кода передаются через различные выходные порты. По сигналу LDAC (LOAD DAC — «загрузить ЦАП») данные передаются из буферов в регистр преобразователя. Сигнал LDAC — сигнал по уровню напряжения; он должен иметь действующее значение (высокий уровень) по крайней мере в течение 0,5 мкс.

На рис. 8.34 приведен стандартный интерфейс между 8-разрядным процессором и преобразователем. Передача данных происходит следующим образом:

Шаг 1. Центральный процессор посылает восемь младших разрядов данных в порт 0 (младший байт регистра ЦАП).

Шаг 2. Центральный процессор посылает два старших бита данных в порт 1 (старший байт регистра ЦАП).

¹L. A. Leventhal. Cut Your Processors Computation Time. Electronic Design, vol. 25, № 17, August 16, 1977, p. 82—89.

²E. R. Hnatek. A User's Handbook of D/A and A/D Converters. — New York, Wiley, 1976.

функциональная схема

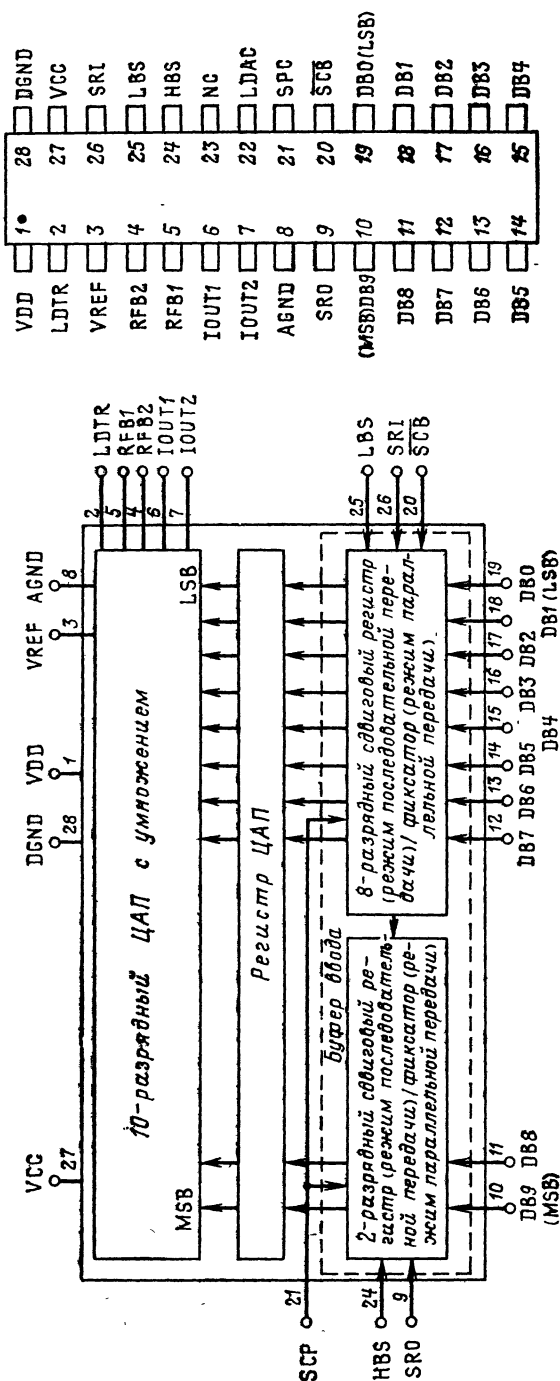


Рис. 8.33. Цифро-аналоговый преобразователь AD7522 фирмы Analog Devices

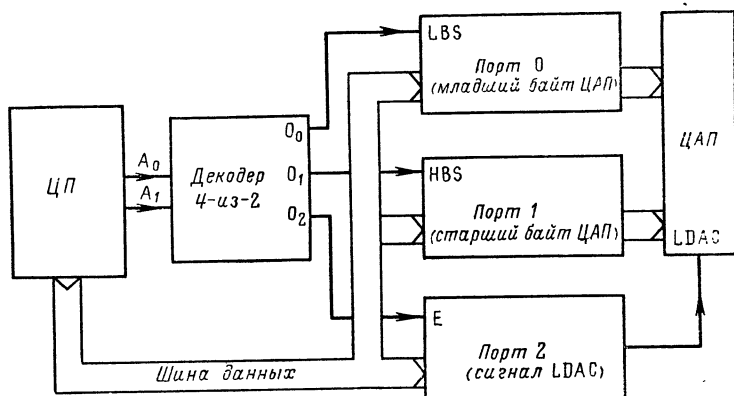


Рис. 8.34. Интерфейс между цифро-аналоговым преобразователем и 8-разрядным процессором. В данной конфигурации ЦАП использует три порта вывода: порт 0 — для младшего байта; порт 1 — для старшего байта и порт 2 — для управляющего сигнала «загрузить ЦАП» (LOAD DAC). На вход LBS подается строб-сигнал для младшего байта, а на вход HBS — для старшего

Шаг 3. Центральный процессор посылает единицу в порт 2, что делает уровень сигнала LDAC высоким.

Шаг 4. Центральный процессор посылает нуль в порт 2, завершая выдачу сигнала LDAC.

Аналого-цифровые преобразователи

Аналого-цифровые преобразователи (АЦП) преобразуют аналоговые входные сигналы в цифровые. Процессор может получать аналоговые сигналы от первичных преобразователей, датчиков и других источников непрерывных сигналов. Такие входные сигналы играют большую роль в системах контроля и управления процессами, шкальных приборах, системах управления механизмами и т. д. Существует несколько различных типов АЦП: счетчик-компаратор, двойственный интегратор (dual-ramp) и АЦП с последовательным взвешиванием. В АЦП с последовательным взвешиванием используется аналоговый компаратор и цифро-аналоговый преобразователь, чтобы последовательно установить значение («вес») каждого входного бита путем сравнения входного сигнала с выходным сигналом ЦАП. Этот метод отличается быстротой выполнения преобразований и простотой реализации, однако он чувствителен к шуму на аналоговом входе, вызывающему ошибки при преобразованиях.

Выпускаются АЦП, разработанные специально для работы в микропроцессорных системах. Эти АЦП имеют выходы с тремя состояниями и раздельные сигналы управления для каждого байта, что позволяет легко связать АЦП с 8-разрядным процессором. На рис. 8.35 изображен 10-разрядный монолитный КМОП-преобразователь AD7570 фирмы Analog Devices, в котором преобразование методом последовательного взвешивания выполняется не более чем за 120 нс.

Микропроцессор может считывать данные из ЦАП за одну или две операции ввода. Имеются отдельные входы «разрешено» (enable) с тремя состояниями: для старшего байта (два старших разряда), для младшего байта (восемь младших разрядов) и для сигнала BUSY («занято»). На рис. 8.36 приведен интерфейс между 8-разрядным микропроцессором и преобразователем. Процессор выбирает данные следующим образом:

1) посылает 1 в порт вывода 0, в результате чего на линию STRT (CONVERT START — «начать преобразование») подается высокий потенциал. На этом шаге сбрасываются все фиксаторы данных, за исключением старшего разряда, который устанавливается единичным;

2) посылает 0 в порт вывода 0, в результате чего потенциал на линии CONVERT START становится низким. По этому сигналу начинается процесс преобразования. Потенциал на линии CONVERT START должен оставаться высоким не менее чем 50 нс;

3) считывает сигнал BUSY из порта ввода 2. Если этот сигнал имеет значение 0, то преобразование все еще продолжается;

4) если в какой-то момент преобразование завершится (BUSY = 1), то процессор считывает восемь младших разрядов данных из порта ввода 0;

5) считывает два старших разряда данных из порта 1.

Остается вопрос, как интерпретировать цифровой вход. Программа может использовать: уравнение, описывающее соответствие аналогового входного сигнала измеряемой величине, или таблицу градуировки. Последний способ интерпретации оказывается полезным; если диапа-

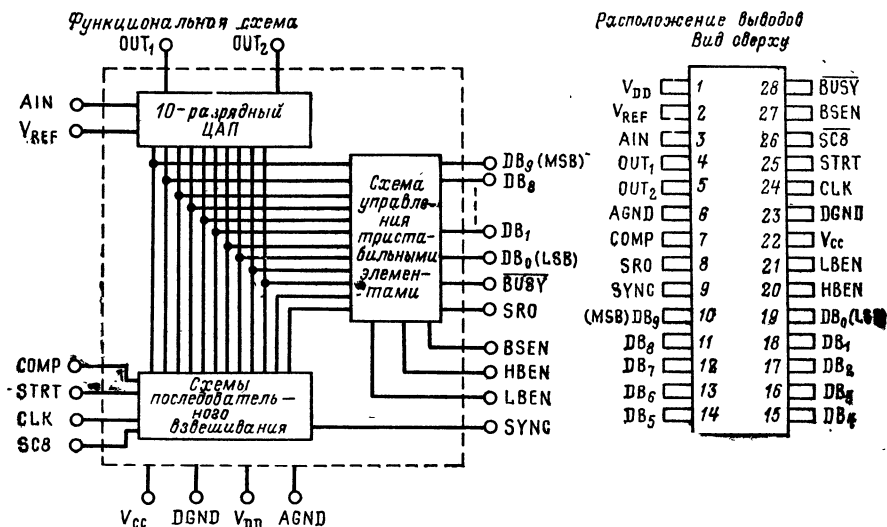


Рис. 8.35. Десятиразрядный монолитный аналого-цифровой преобразователь AD7570 фирмы Analog Devices [для работы необходимо внешнее устройство — аналоговый компаратор (311)]

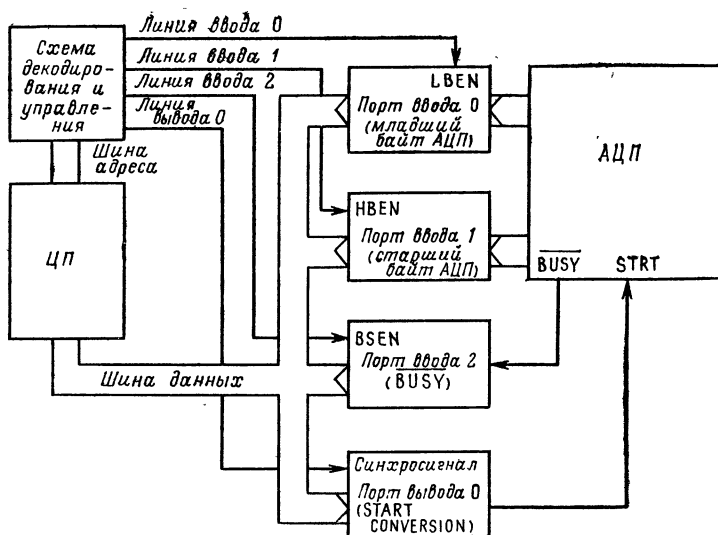


Рис. 8.36. Интерфейс между аналого-цифровым преобразователем и 8-разрядным процессором:

LBEN — сигнал разрешения для младшего байта; HBEN — сигнал разрешения для старшего байта; BSEN — разрешение для сигнала BUSY

зон значений не слишком велик; этот метод не требует решения уравнений, но для размещения таблицы градуировки требуется дополнительная память.

Телетайпы

Несмотря на процесс усовершенствования периферийных устройств, телетайп остается наиболее распространенным УВВ малых систем. Стандартный телетайп состоит из:

- 1) клавишного пульта, передающего символы (в коде ASCII) асинхронно с максимальной скоростью 10 символов/с;
- 2) печатающего устройства (символы кода ASCII), принимающего символы асинхронно с максимальной скоростью 10 символов/с;
- 3) устройства считывания с перфоленты (10 символов/с);
- 4) ленточного перфоратора (10 символов/с);
- 5) интерфейса по току (current loop interface); ток 20 мА в контуре соответствует логической 1, а отсутствие тока — 0.

Каждый передаваемый символ имеет 11 разрядов (рис. 8.37): стартовый бит (всегда 0), семь битов данных, бит четности (дополняет число единиц в двоичном представлении символа до четного или нечетного числа или не используется) и два стоповых бита (всегда 1). Обычно линия находится в состоянии 1, или «маркер». Так как устройство может передавать десять 11-разрядных символов в секунду, передача каждого бита длится 1/110, с т. е. 9,1 мс.

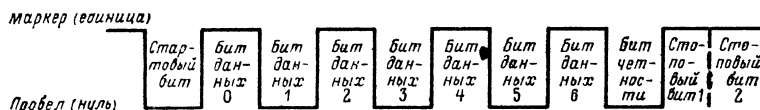


Рис. 8.37. Стандартная форма представления данных при обмене с телематомом. Для кодирования каждого символа используется 11 разрядов. Длительность сигнала для одного разряда 9,1 мс)

Интерфейс телетайпа должен обеспечивать выполнение следующих функций:

1) преобразование сигналов контурного тока в сигналы для ТТЛ-устройств (по напряжению) и обратно. Сигналы должны быть электрически разделены (трансформатором, реле или устройством оптронной развязки). На рис. 8.38 в качестве разделителя использована оптронная развязка;

2) последовательный ввод-вывод телетайпа;

3) управление устройством считывания с ленты;

4) управление возвратной, или «эхо»-линией, для печати символов, вводимых с клавиатуры;

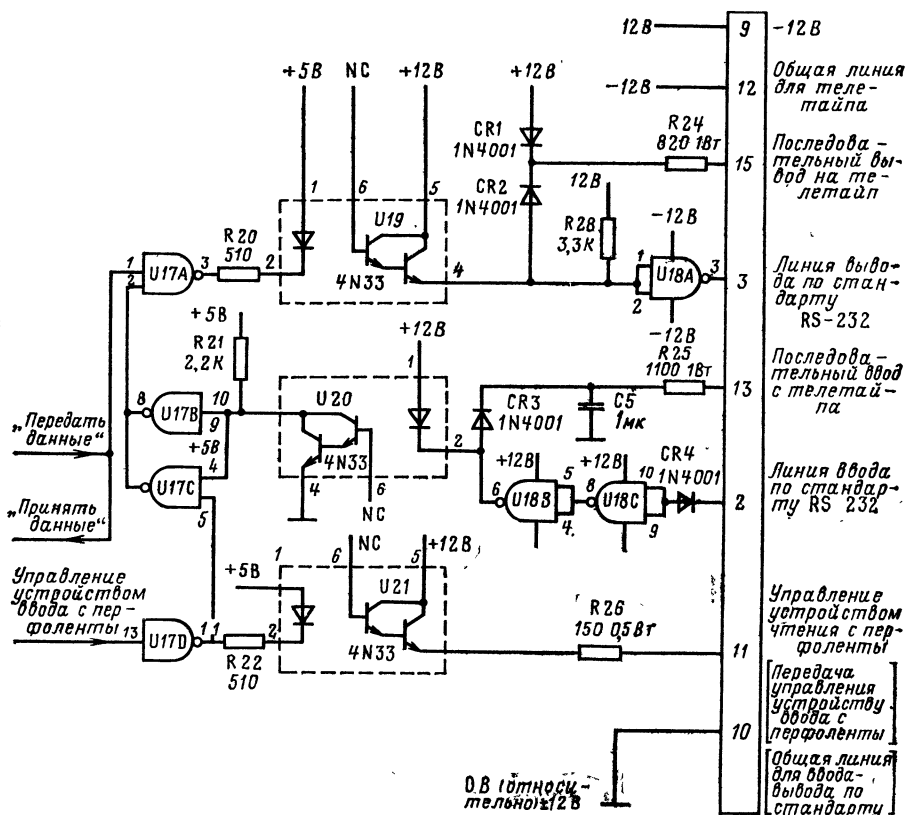


Рис. 8.38. Интерфейс телетайпа

- 5) распознавание (при приеме) и добавление (при передаче) стартового бита;
- 6) генерация сигналов синхронизации для обеспечения нужных интервалов времени между битами;
- 7) выработка бита четности; контроль четности;
- 8) распознавание и добавление стоп-разрядов.

Большинство этих функций может выполнить УАПП. Для выполнения функций преобразования сигналов, правления устройством считывания с ленты и «эхо»-линией требуется дополнительная аппаратура.

Интерфейс, представленный на рис. 8.38 может вырабатывать как сигналы для управления телетайпом, так и, сигналы стандарта RS-232, а также сигналы управления чтением с перфоленты. На рисунке изображены следующие активные устройства:

ИС MC1489 — счетверенный приемник последовательного интерфейса типа RS-232 (U17);

ИС MC1488 — счетверенный последовательный драйвер типа RS-232 (U18);

три оптронные развязки Motorola 4N33 (U19 — U21);

четыре диода Motorola 1N4001.

Часть функций интерфейса можно реализовать программно. Процессор может выполнять некоторые или даже все функции УАПП. На

рис. 8.39 приведена блок-схема процедуры передачи данных, а на рис. 8.40 — блок-схема процедуры приема данных. Программа может выполнять параллельно-последовательные и последовательно-параллельные преобразования, управлять старт- и стоп-разрядами, генерировать бит четности и проверять четность, обеспечивать синхронизацию. Устройство, аналогичное адаптеру интерфейса периферийных устройств (PIA) фирмы Motorola, может обеспечить работу с последовательными линиями ввода-вывода, а также с сигналами управления (как, например, для устройства считывания с перфоленты). Входы для данных могут быть

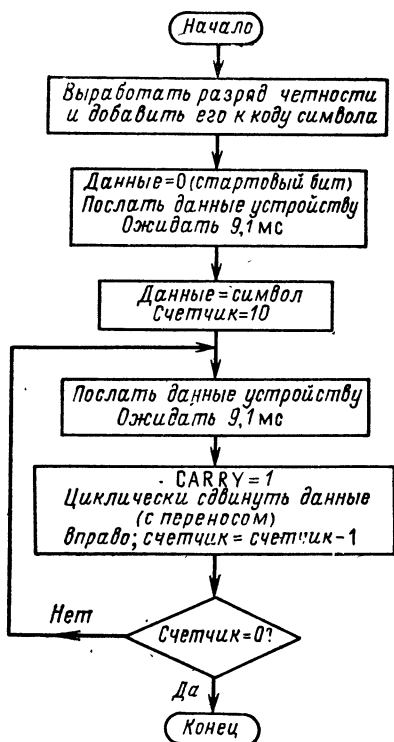
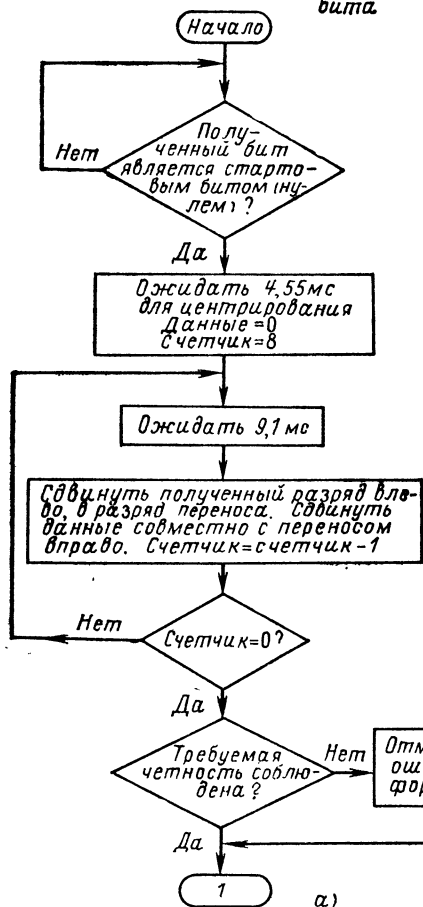


Рис. 8.39. Блок-схема процедуры передачи данных на телетайп (значение разряда 0 выхода поступает к периферийному устройству. Установка в единицу разряда переноса при каждом прохождении цикла формирует требуемые стоп-разряды)

Ожидание появления стартового бита



Ожидание стоп-разрядов

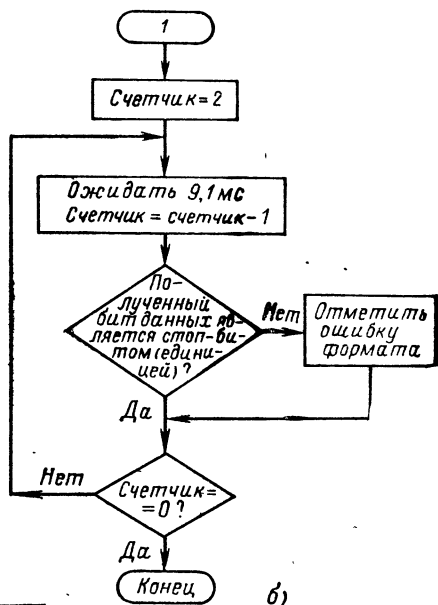


Рис. 8.40. Блок-схема процедуры приема данных с телетайпа (значение разряда 7 последовательного входа поступает от периферийного устройства. Ошибка формата состоит в том, что отсутствуют один или оба стоп-разряда)

соединены (или программно настроены) в соответствии с конкретным периферийным устройством.

Аналогичным образом, изменяя некоторые параметры, программа может управлять другими терминалами. Например, широко распространенные терминалы, обладающие скоростью 30 символов/с, требуют для своей работы один стоп-бит, кроме того, их временная диаграмма отличается от временной диаграммы телетайпа. Использование одной из линий ввода в качестве индикатора позволяет программе управлять работой терминалов обоих типов. Например, на эту линию можно выставлять высокий потенциал при работе с телетайпом и низкий — при работе с терминалом, имеющим скорость 30 символов/с. В этом случае программа устанавливает нужные значения параметров в соответствии с состоянием на входе.

Длительное время организация ввода-вывода данных усложнялась тем, что применение каждого нового периферийного устройства порождало уникальную проблему связи УВВ с ЦП. Однако сейчас для многих УВВ пользуются одним из следующих трех интерфейсов:

- 1) интерфейс для телетайпа (по току);
- 2) последовательный интерфейс RS-232 для периферийного информационного оборудования (DTE — data terminal equipment) и для информационного оборудования связи (DCE — data communications equipment). Максимальная скорость передачи данных — 20 Кбит/с;
- 3) параллельный интерфейс IEEE STD 488-1975 — так называемый стандартный цифровой интерфейс программируемых приборов (Standard Digital Interface for Programmable Instrumentation). Этот интерфейс называют также интерфейсом Hewlett-Packard или шиной связи общего назначения (GPIB — General Purpose Interface Bus). Максимальная скорость передачи данных — 10^6 бит/с.

Интерфейс RS-232. В табл. 8.8 приведены сигналы устройства RS-232, которые разбиты на следующие группы: сигналы данных, сигналы управления, сигналы времени; отмечено, передаются ли эти сигналы от терминалов или к терминалам информационного оборудования связи. Большинство реальных интерфейсов использует только часть этих сигналов.

Линии данных работают последовательно. Эти линии не обязательно должны быть физически различными. Система с физически различными линиями, которая может передавать и принимать данные одновременно, называется *дуплексной* (full-duplex), а система, которая в любой момент может либо передавать, либо принимать данные, — *полудуплексной*. (half-duplex).

Линии состояния оборудования указывают, готово ли оборудование послать или принять данные. Сигналы на этих линиях следующие: «пакет данных готов» (DATA SET READY) — для оборудования связи, или «терминал данных готов» — (DATA TERMINAL READY) — для терминального оборудования. Многие интерфейсы типа RS-232 не используют эти сигналы и работают в предположении, что оборудование всегда готово. Эти сигналы могут понадобиться при тестировании оборудования.

Имеются два сигнала управления передачей данных: «запрос послышки» (REQUEST TO SEND) и «очистить для послышки» (CLEAR TO SEND). Терминал возбуждает сигнал «запрос послышки», когда имеет данные для передачи. Оборудование связи выдает сигнал «очистить для послышки», когда становится готовым к передаче данных. Эти сигналы применяются в таких устройствах, как асинхронный адаптер интерфейса оборудования связи 6850 фирмы Motorola (Asynchronous Communications Interface Adapter), низкоскоростной модем 6860 той же фирмы и программируемый интерфейс связи 8251 фирмы Intel. Адаптер 6850 фирмы Motorola будет рассмотрен в следующем параграфе. Сигналы времени интерфейса RS-232 отмечают центры сигналов (битов) данных; часто эти сигналы не используются.

Таблица 88 Сигналы последовательного интерфейса EIA RS-232

Цепь обмена	Эквивалент С.С.Т.Т.	Описание	Земля	Данные от DCE	Данные к DCE	Сигналы управ- ления от DCE	Сигналы управ- ления к DCE	Сигналы син- хронизации от DCE	Сигналы син- хронизации к DCE
AA	101	Защитное за- земление	X						
AB	102	Обратная линия (земля/общая шина) сигнала	X						
BA	103	Передаваемые данные			X				
BB	104	Получаемые данные		X					
CA	105	Запрос посылки					X		
CB	106	Очистить для посылки				X			
CC	107	Пакет данных готов				X			
CD	108,2	Терминал дан- ных готов					X		
CE	125	Звонок (инди- катор звонка)				X			
CF	109	Получен сигнал с линии				X			
CG	110	Качество сиг- нала				X			
CH	111	Селектор ско- рости передачи сигналов дан- ных (DTE)					X		
CI	112	Селектор ско- рости передачи сигналов дан- ных (DCE)				X			
DA	113	Синхрониза- ция элементар- ных сигналов передатчика (DTE)							X
DB	114	Синхрониза- ция элементар- ных сигналов передатчика (DCE)						X	
DD	115	Синхрониза- ция элементар- ных сигналов приемника (DCE)						X	

Имеются три сигнала управления приемом данных: «звонок» (RING INDICATOR), «получен сигнал с линии» (RECEIVED LINE SIGNAL DETECTOR) и «качество сигнала» (SIGNAL QUALITY DETECTOR). Сигнал «звонок» означает, что в канале связи принят сигнал звонка (как, например, при телефонной связи). Сигнал «получен сигнал с линии» указывает, что оборудование связи получает некоторый сигнал (интерпретация которого зависит от оборудования связи). Сигнал «качество сигнала» дает информацию о том, какова вероятность появления ошибок в принимаемых данных.

Уровни напряжения для указанных сигналов несколько отличаются от обычных уровней для ТТЛ-устройств. Для преобразования напряжения требуются специальные линейные драйверы или приемные устройства. Широко используются следующие устройства: преобразователь уровня напряжения 1488 (от RS-232-уровней к ТТЛ-уровням), преобразователь уровня напряжения 1489 (от ТТЛ-уровней к RS-232-уровням), линейные драйверы и приемники 8T15 и 8T16 EIA фирмы Signetics. Эти устройства преобразуют уровни напряжения, обеспечивают необходимый нагрузочный ток и выполняют другие преобразования для обеспечения требуемых физических свойств сигналов (см. рис. 8.38, где приведен пример простого RS-интерфейса). Линии данных RS-232 используют отрицательную логику; линии управления — положительную.

Интерфейс RS-232 широко используется при скоростях передачи данных от низких до средних. Электронные устройства, согласующиеся с характеристиками этого интерфейса, недорогие и широко распространены. При более высоких скоростях обмена стандарт RS-232 заменяется стандартами RS-423 (скорость обмена данными до 100 Кбит/с) и RS-422 (скорость обмена данными до 10 Мбит/с)¹.

Интерфейс IEEE-488. Интерфейс IEEE-488 разработан для сетей приборов. На рис. 8.41 описаны линии сигналов. 24-разрядная шина содержит восемь линий заземления, восемь линий данных и восемь линий управления.

Сеть может содержать до 15 устройств трех следующих типов:

- 1) «говорящие» устройства, которые могут выставлять данные на линии данных;
- 2) «слушающие» устройства, которые могут считывать данные с линий данных;
- 3) контроллеры, назначающие устройствам тип «говорящего» или «слушающего».

Контроллер назначает устройствам указанные типы в режиме управления. Линия ATN имеет низкий уровень, поэтому все устройства могут следить за линиями. Команды НЕ РАЗГОВАРИВАТЬ (UNTALK) и НЕ СЛУШАТЬ (UNLISTEN) уничтожают все ранее установленные связи; команды СЛУШАТЬ АДРЕС (LISTEN ADDRESS) и СКАЗАТЬ АДРЕС (TALK ADDRESS) формируют конфигурацию шины. Адреса приборов назначаются производителем аппаратным образом. Разуме-

¹D. Morris. Revised Data-Interface Standards, Electronic Design, vol. 25, № 18, September 1, 1977, p. 138—141.

Восемь линий данных:

линии данных — линии от DIO-1 до DIO-8. По этим линиям происходит обмен данными между всеми приборами, связанными с шиной (биты, составляющие байт данных, передаются параллельно; байты — последовательно).

Три линии управления передачей данных:

линии передачи содержат линии DAV (data valid) — правильные данные, NRFD (not ready for data — не готово для данных) и NDAC (not data accepted — данные не приняты). Эти линии обеспечивают связь между «говорящим» и «слушающим» приборами для синхронизации потока информации по восьми линиям данных:

а) DAV. Указывает, что информация, доступная на линиях данных, является корректной;

б) NDAC. Указывает, что прибор готов воспринять информацию;

в) NRFD. Указывает, что информация воспринята «слушающим» устройством.

Пять линий управления шиной:

линии интерфейса, координирующие поток информации на шине.

а) IFC. Приводит систему в известное состояние;

б) ATN. Указывает характер информации на линиях данных;

в) REN. Приказывает приборам выполнить дистанционные операции;

г) SRQ. Запрос на обслуживание;

д) EOI. Указывает на конец многобайтной последовательности обмена или в комбинации с ATN приказывает выполнить процедуру полинга.

Рис. 8.41. Действующие линии сигналов интерфейса IEEE-488

ется, только одно устройство может «говорить» в любой данный момент, но несколько устройств могут одновременно «слушать». Можно адресовать устройство как «слушающее» и как «говорящее», но не в один и тот же момент.

В режиме «данные» «говорящее» устройство передает данные «слушающим». Линия ATN имеет высокий уровень, поэтому только те устройства, которые были адресованы ранее, примут участие в процессе обмена данными. Линии управления передачей данных могут использоваться «слушателями» для проверки шины данных. Когда передача данных завершилась, «говорящее» устройство возвращает управление сетью контроллеру для реконфигурации. Устройства, связанные с шиной, могут запросить внимания контроллера, даже если они не входят в состав данной конфигурации.

На рис. 8.42 приведены ограничения, обусловленные стандартом IEEE-488. С помощью специальных схем, таких как MC3400 фирмы Motorola, можно обеспечить необходимые электрические характеристики.

Максимальное число устройств, которые могут быть подсоединены к GPIB:

15

Максимальная длина кабеля:

2 м \times (число устройств) или

20 м (меньшее из двух этих чисел)

Максимальная скорость передачи данных:

1 Мбит/с (по любой линии)

З а м е ч а н и я:

а) максимум 250 000 байт/с на 20 м с усилением (load) через каждые 2 м с помощью драйверов с открытым коллектором (48 мА);

б) максимум 500 000 байт/с на 20 м с усилением через каждые 2 м с помощью тристабильных драйверов (48 мА)

в) 1 Мбайт/с на максимальном расстоянии между двумя устройствами 1 м при использовании тристабильных драйверов (48 мА)

Максимальное число доступных адресов:

31 для «говорящих»

31 для «слушающих»

Рис. 8.42. Ограничения, обусловленные стандартом IEEE-488, для шины

8.6. РАЗРАБОТКА ПОДСИСТЕМ ВВОДА-ВЫВОДА ДЛЯ КОНКРЕТНЫХ МИКРОПРОЦЕССОРОВ

Микропроцессор Intel 8080

Подсистема ввода-вывода для Intel 8080 обычно строится на ТТЛ-устройствах методом изолированных линий ввода-вывода. Пользователю доступны программируемые интерфейсы однако в большинстве систем используется порт ввода-вывода 8212

Микропроцессор 8080 имеет две команды ввода-вывода:

команда IN передает 8-битные данные из адресуемого порта ввода в аккумулятор;

команда OUT передает 8-битные данные из аккумулятора в адресуемый порт вывода.

Обе команды выполняются за 10 периодов тактовых импульсов и занимают в памяти 2 байта: один байт — для кода операции, другой — для 8-разрядного адреса устройства (или порта). Таким образом, процессор может управлять 256 портами ввода и 256 портами вывода. Центральный процессор выставляет 8-разрядный адрес устройства как в старшие, так и в младшие разряды шины адреса, поэтому любую из групп разрядов можно декодировать как адрес.

Два сигнала состояния, выдаваемые процессором в первом такте каждого машинного цикла, указывают, что данный цикл является циклом ввода-вывода. Бит INP (шестой бит состояния) идентифицирует цикл ввода, а бит OUT (четвертый бит состояния) — цикл вывода. Для формирования сигналов управления $\overline{I/O}$ и $\overline{I/O}$ (действующее значение — низкий уровень), совместимых с сигналами управления памятью (рассмотренными в предыдущей главе), можно скомбинировать сигналы INP и OUT с сигналом записи (\overline{WR}) и сигналом направления работы шины данных (\overline{DBIN}). Поступление этих сигналов обеспечивается системным контроллером 8228. Отметим следующие основные положения, которые следует иметь в виду при организации ввода данных.

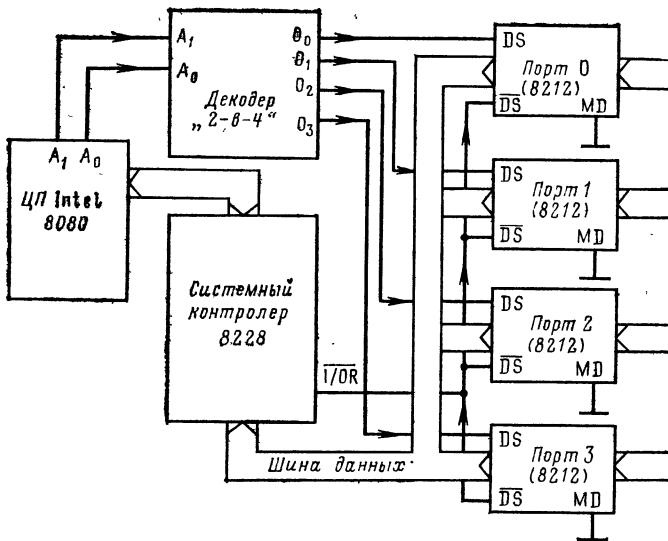


Рис. 8.43. Подсистема ввода Intel 8080 (конкретный порт выставляет данные на шину данных процессора только тогда, когда $\overline{I/O\overline{R}}$ принимает действующее значение, а порт выбран декодером)

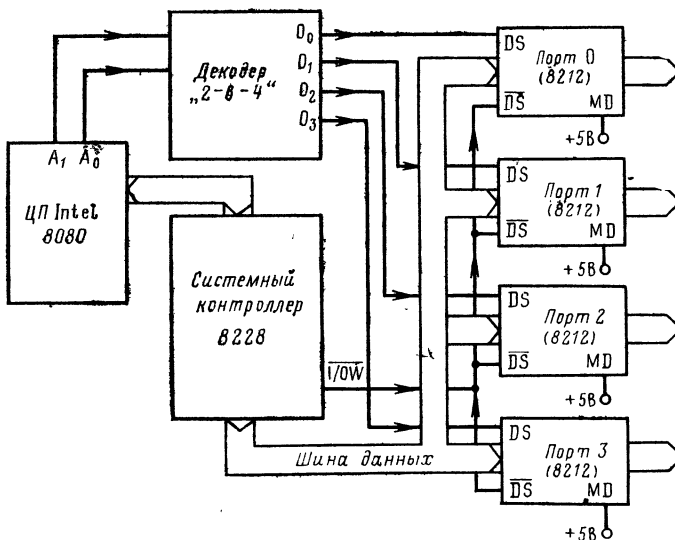


Рис. 8.44 Подсистема вывода Intel 8080 (данные по сигналу синхронизации помещаются в конкретный порт вывода, когда $\overline{I/O\overline{W}}$ принимает действующее значение, а декодер выбрал этот порт)

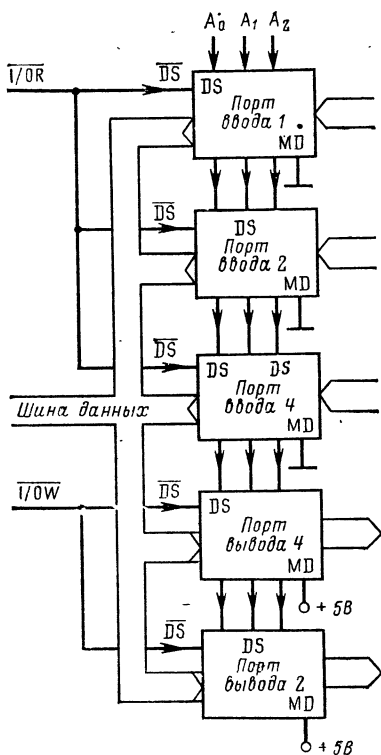


Рис. 8.45. Подсистема ввода-вывода Intel 8080, в которой используется выбор с помощью линий адреса (все порты ввода открываются сигналом $\overline{I/OR}$, все порты вывода — сигналом $\overline{I/OW}$. Линия адреса A_0 выбирает порт 1, линия A_2 — порт 4 и т. д. Недопустимы адреса портов с более чем одной линией адреса, находящейся под высоким потенциалом)

1. Все порты ввода должны иметь выходы или буфера с тремя состояниями; можно применить, например, работающее в режиме ввода ($MD = 0$) устройство 8212 фирмы Intel.

2. Декодирующая система должна отличать адреса ввода-вывода от адресов памяти. Для этого требуется, чтобы либо каждый порт ввода открывался сигналом \overline{INP} (или $\overline{I/OR}$), либо порты ввода были изолированы от общей шины данных буферами с тремя состояниями, открываемыми по одному из упомянутых сигналов.

3. Декодирование 8-разрядного адреса устройства не должно противоречить действию сигналов открытия тристабильных устройств. Если число портов ввода не велико, то полное декодирование не требуется.

4. Синхронизация редко вызывает трудности, так как ТТЛ-порты работают с гораздо меньшей временной задержкой, чем модули памяти на МОП-схемах.

На рис. 8.43 изображена подсистема ввода, содержащая четыре порта. Каждый порт открывается сигналом от декодера и сигналом $\overline{I/OR}$ от системного контроллера. Эта подсистема ввода может пользоваться шиной данных совместно с памятью.

Отметим следующие основные положения, которые следует иметь в виду при организации вывода данных.

1. Все порты вывода должны иметь адресуемый фиксатор. В качестве такого фиксатора можно использовать модуль 8212 фирмы Intel в режиме вывода ($MD = 1$). Порт вывода не обязательно должен иметь три состояния, так как его выход не выставляется на совместно используемую шину.

2. Система должна отличать адреса ввода-вывода от адресов памяти. Обычно каждый порт вывода синхронизируется сигналом \overline{OUT} или $\overline{I/OW}$.

3. Одна и та же система декодирования может вырабатывать адреса как входных, так и выходных портов.

4. Синхронизация редко вызывает трудности, за исключением тех случаев, когда данные не фиксируются и доступны в течение слишком короткого промежутка времени.

На рис. 8.44 показана подсистема вывода, содержащая четыре порта. Каждый порт синхронизируется сигналом от декодера и сигналом $\overline{I/OW}$ от контроллера 8228. Эта подсистема вывода может использовать шину данных совместно с памятью и с портами ввода.

Малые подсистемы ввода-вывода могут использовать линию выбора порта так, как показано на рис. 8.45. Каждая линия адреса выбирает один из портов. Число портов здесь ограничено числом линий адреса (восьмью), а номера портов — степенью двойки (табл. 8.9). В декодерах нет необходимости.

Таблица 8.9. Адресация портов при использовании линии шины адреса для выбора порта (8-разрядный адрес)

Используемая линия шины адреса	Номер порта		Используемая линия шины адреса	Номер порта	
	десятичный	шестнадцатеричный		десятичный	шестнадцатеричный
A ₀	1	01	A ₄	16	10
A ₁	2	02	A ₅	32	20
A ₂	4	04	A ₆	64	40
A ₃	8	08	A ₇	128	80

Процессор 8080 может использовать и метод ввода-вывода, адресуемого как память. В этом случае управляющие сигналы и команды ввода-вывода не используются; вместо этого для адресации устройств ввода-вывода резервируются определенные адреса памяти. Например, линия A₁₅ адресной шины может использоваться для того, чтобы отличить ВВВ от памяти. При этом устройствам ввода-вывода в памяти предоставляется пространство по 32 К, из которых память занимает младшие 32 К адресов, среди которых находится адрес перехода по сигналу «оброс» (RST) и адреса подпрограмм обработки прерываний. Организованная таким способом подсистема ввода-вывода может адресовать до 15 портов, причем декодеры не нужны. Ввод-вывод, адресуемый как память, удобен для микро-ЭВМ в состав которых входят такие устройства, как программируемый интерфейс связи 8251, программируемый интерфейс периферийных устройств 8255 и многофункциональный контроллер ввода-вывода TMS 5501

Интерфейс между простыми периферийными устройствами и микропроцессором Intel 8080

Пример 1. Двупозиционный переключатель. Для однополюсного переключателя с одним или двумя положениями требуется только 1 бит порта ввода 8212. Определение того, находится ли переключатель во включенном положении, выполняется тремя командами:

IN PORT ;Считать состояние переключателя
ANI MASK ;Маскировать данные от переключателя
JZ CLSD ;Переход, если переключатель выключен

Параметр MASK содержит 1 в разряде, соответствующем данному переключателю, и 0 — в остальных разрядах. Если переключателю соответствует нулевой, шестой или седьмой разряд, то в программе можно воспользоваться командой сдвига и флагами «перенос» или «знак». Например, если переключатель присоединен к шестому разряду, то его положение можно проверить с помощью следующей последовательности команд:

IN PORT ;Считать состояние переключателя
ADD A ;Знаковый бит регистра признаков = состояние переключателя
JP CLSD ;Перейти, если переключатель выключен

С помощью команды ИСКЛЮЧАЮЩИЕ ИЛИ можно обнаружить изменение положения переключателя:

IN PORT ;Считать новое состояние переключателя
MOV B,A
LXI H,OLD
XRA M ;Проверить, есть ли изменения по сравнению со старым
JZ NOCH ;состоянием
MOV M,B ;Заменить старое состояние на новое, если произошло изменение

После выполнения приведенных команд единицы находятся в тех разрядах аккумулятора, которые соответствуют переключателям, изменившим свои состояния.

Пример 2. Многопозиционный переключатель без кодирования. Для переключателя на восемь позиций необходим 8-разрядный порт ввода. Положение

переключателя можно определить с помощью следующей последовательности команд:

```
GETSW: IN    PORT    ;Считать состояние переключателя
        CPI    OFFH   ;Находится ли переключатель в какой-нибудь из
                        ;позиций?
        JZ     GETSW   ;Нет, ждать, пока позиция переключателя не
                        ;станет определенной
        MVI    B,0     ;Позиция переключателя = 0
SRPOS:  RAR     ;Следующий разряд = 0?
        JNC    FOUND; ;Да, позиция найдена
        INR    B       ;Нет, позиция = позиция + 1
        JMP    SRPOS
FOUND:  HLT
```

После выполнения программы информация о положении переключателя окажется в регистре В.

Пример 3. Клавиатура 3×3 без кодирования. Если клавиатура подсоединена так, как показано на рис. 8.28, то нажатая клавиша определяется с помощью следующей программы:

```
;
; Определить, нажата ли хотя бы одна клавиша
;
CHKBD:  SUB     A       ;Обнулить (заземлить) все столбцы
        OUT    KOUT
        IN     KIN
        ANI    00000111B ;Выделить маскированием разряды линий строк
        CPI    00000111B ;Равны ли нулю какие-нибудь строки?
        JZ     CHKBD    ;Нет, продолжить просмотр
;
;Идентифицировать столбец, обнуляя по одному столбцу
;
        MVI    B,3      ;Число столбцов = 3
        MVI    C,0      ;Номер клавиши = 0
        MVI    D,11111111 0B ;Маска для обнуления первого столбца
FCOL:   MOV     A,D
        OUT    KOUT
        RLC          ;Сформировать сдвигом маску для следующего столбца
        MOV     D,A
        IN     KIN    ;Считать входы с линий строк
        ANI    00000111 B ;Маскированием выделить разряды строк
        CPI    00000111B ;Есть нулевые строки?
        JNZ    FROW    ;Да, найти нулевую строку
        MOV     A,C     ;Нет, подготовить номер клавиши для следующего столбца
        ADI     3       ;Номер клавиши = номер клавиши + число строк
        MOV     C,A
        DCR     B       ;Все столбцы проверены?
        JNZ    FCOL    ;Нет, обнулить следующий столбец
        JMP    ERROR    ;Да, ошибка, клавиша не найдена
;
;Идентифицировать строку, проверяя входы
;
FROW:   INCR    C       ;Номер клавиши-номер клавиши + 1
        RAR     ;Бит с линии строки = 0?
        JC     FROW     ;Нет, продолжать поиск
```

Номера клавиш могут соответствовать цифрам, символам, командам или функциям. Программа должна определить функциональный смысл номера клавиш (возможно, с помощью таблицы преобразования или таблицы переходов).

Пример 4. Клавиатура с кодированным выходом и строб-сигналом. На рис. 8.46 показано использование двух портов 8212 фирмы Intel для связи с кодирующим клавишным пультом, выдающим 8-разрядные данные и строб-сигнал. По стробу данные фиксируются в порте 1 и устанавливается сигнал состояния (с низким действующим значением) в порте 2. Следует обратить внимание на то, что ЦП не может непосредственно определить состояние триггера запроса на обслуживание по данным из порта, так как этот триггер не адресуем. Устройство 8212 разработано для использования в системах, управляемых по сигналам прерывания (см. гл. 9), в которых сигнал \overline{INT} непосредственно воздействует на последовательность команд процессора, вызывая прерывание.

Программа должна определить, имеет ли сигнал состояния действующее значение, и, если имеет, считать данные. Чтение данных автоматически устанавливает недействующее значение сигнала состояния, так что процессор не будет второй раз читать те же самые данные. Программа выглядит следующим образом:

```
SPORT EQU 1           ;Порт состояния клавиатуры
DPORT EQU 2           ;Порт данных клавиатуры
;
; Проверка: принимает ли зафиксированный строб-сигнал
; действующее (низкое) значение
CSTB: IN SPORT         ;Считать зафиксированное состояние
      ANI SMASK        ;Состояние имеет действующее значение (0)?
      JNZ CHSTB        ; Нет; продолжить проверку
;
; Считать данные
; IN DPORT             ;Считать данные с клавиатуры
```

Данные с клавиатуры могут поступать в различных формах. Если это УВВ использует отрицательную логику, то необходимо использовать команду *СМА* (дополнение). Программа может также добавлять и исключать разряды четности, преобразовывать коды, выдавать «эхо»-информацию печатающему устройству и интерпретировать буквы и цифры.

Пример 5. Один индикатор. Для организации работы одиночного индикатора нужен только один разряд порта вывода 8212. От полярности подключения индикатора зависит, какие команды необходимы для включения и выключения индикатора. Простая последовательность команд управления состоянием индикатора может быть следующей:

```
MVI A, MASK           ;Подготовить данные для индикатора
OUT PORT              ;Послать данные индикатору
```

Здесь параметр *MASK* содержит нужное значение (0 или 1) в разряде, соответствующем данному индикатору. С помощью приведенной ниже последовательности команд можно подать одно и то же значение сразу на группу индикаторов:

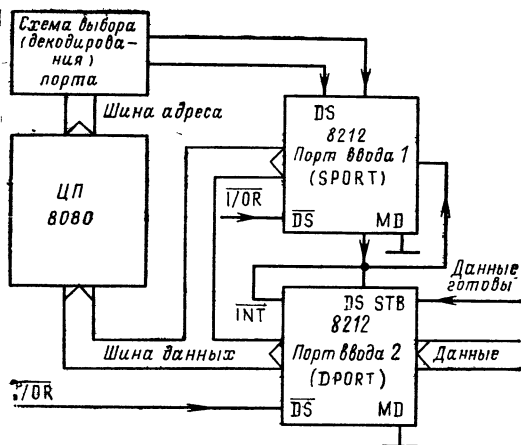


Рис. 8.46. Интерфейс между Intel 8080 и клавишным пультом с кодированным выходом

Для управления состоянием отдельного индикатора можно воспользоваться одной из следующих последовательностей команд:

LDA	DSPLY	;Подготовить данные для индикатора
ORI	MASK	;Установить бит, соответствующий индикатору, в единицу
OUT	PORT	;Послать данные на индикатор

LDA	DSPLY;	;Подготовить данные для индикатора
ANI	MASK	;Очистить бит, соответствующий индикатору
OUT	PORT	;Послать данные на индикатор

LDA	DSPLY	;Подготовить данные для индикатора
XRI	MASK	;Взять дополнение к разряду, соответствующему индикатору
OUT	PORT	;Послать данные на индикатор

Пример 6. Семисегментный индикатор. Семисегментный индикатор (без декодирования) можно присоединить к порту вывода Intel 8212 способом, показанным на рис. 8.47. Программа должна преобразовывать выходные данные в нужную форму и посылать их в порт. С помощью команды дополнения можно изменять логические уровни сигналов на противоположные.

[illegible]

368

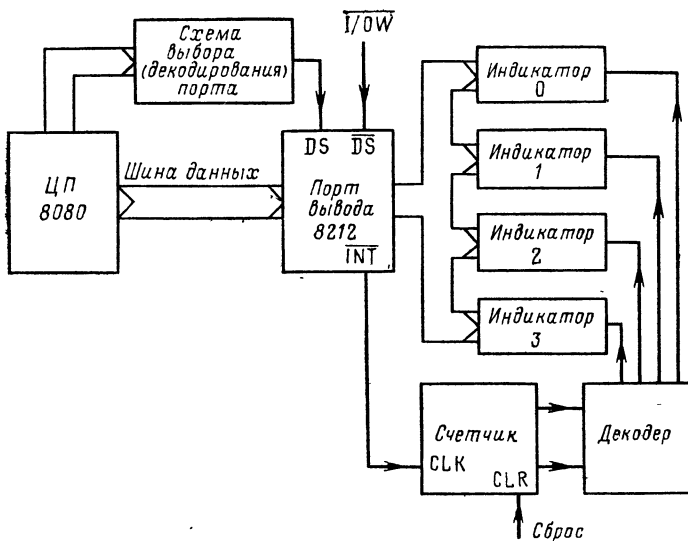


Рис. 8.48. Интерфейс между Intel 8080 и набором семисегментных индикаторов с использованием счетчика и декодера (счетчик и декодер определяют, какой индикатор должен гореть; выход \overline{INT} генерирует сигнал синхронизации после выполнения каждой операции вывода данных)

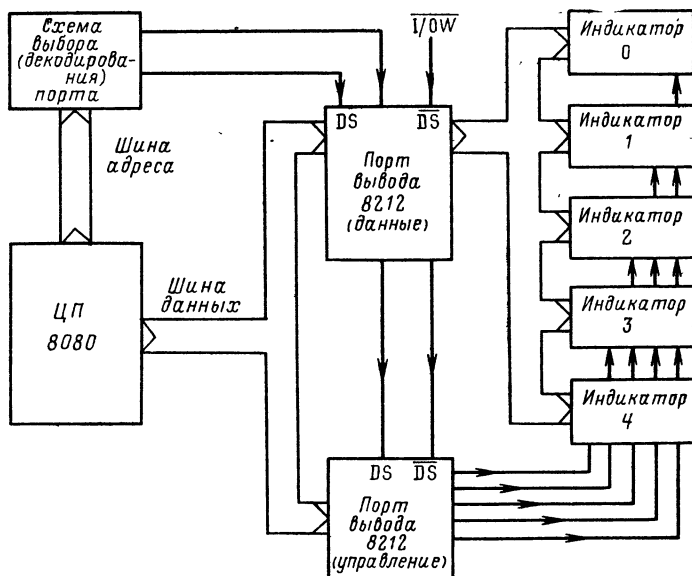


Рис. 8.49. Интерфейс между Intel 8080 и набором семисегментных индикаторов с использованием порта для управления индикаторами (данные в порте управления определяют, какой индикатор должен гореть)

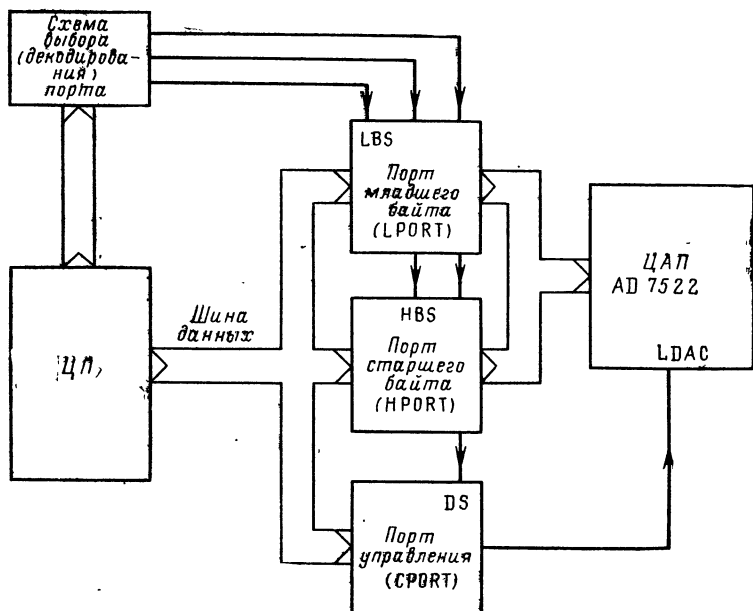


Рис. 8.50. Интерфейс между Intel 8080 и цифро-аналоговым преобразователем AD7522 фирмы Analog Devices (порты данных для младшего и старшего байтов фактически являются составной частью преобразователя; порт управления — отдельное устройство)

порт управления. С помощью этого метода можно подать информацию на любой индикатор или группу индикаторов в любой момент времени.

Пример 7. Цифро-аналоговый преобразователь. Цифро-аналоговый преобразователь, подобный AD7522 фирмы Analog Devices, можно подключить к процессору Intel 8080 так, как показано на рис. 8.50. Преобразователь использует три порта: первый — для восьми младших разрядов данных, второй — для двух старших, а третий — для сигнала загрузки (LDAC). Вывод информации в аналоговой форме обеспечивает следующая программа:

;Данные передать в ЦАП:

LDA LBYTE ;Подготовить восемь младших разрядов

OUT LPORT ;Переслать в ЦАП

LDA HBYTE ;Подготовить два старших разряда

OUT HPORT ;Переслать в ЦАП

;Загрузить ЦАП и начать преобразование:

MVI A, LDAC1 ;Подготовить сигнал загрузки ЦАП

OUT CPORT ;LDAC = 1

SUB A

OUT CPORT ;LDAC = 0

Маска LDAC 1 имеет 1 в разряде, соответствующем данному преобразователю.

Пример 8. Аналого-цифровой преобразователь.

Аналого-цифровой преобразователь, подобный AD7570 фирмы Analog Devices, можно подключить к процессору Intel 8080 так, как показано на рис. 8.51. Аналого-цифровой преобразователь использует три порта ввода: один для восьми младших разрядов данных, второй — для двух старших, а третий — для сигнала «занято» (BUSY). Необходим также порт вывода для сигнала «начать преобразование». Три порта ввода с тремя состояниями фактически являются частью

преобразователя. Данные от преобразователя можно получить с помощью следующей программы:

```

;
; Послать сигнал «начать преобразование»
;
    MVI    A, STCON      ;Подготовить сигнал «начать преобразование»
    OUT    CPORT         ;STRT = 1
    SUB    A
    OUT    CPORT         ;STRT = 0
; Ожидать окончания преобразования:
;
    CHBSY: IN    BPORT    ;Считать сигнал «занято»
    ANI     MASK         ;Закончено ли преобразование
                        ;(BUSY = 1)?
    JZ      CHBSY        ;Нет; продолжить проверку
;
; Прочитать данные и сохранить их в памяти:
;
    IN      LPORT         ;Прочитать восемь младших разрядов
    STA     LBYTE
    IN      HPORT         ;Прочитать два старших разряда
    STA     HBYTE

```

Маска STCON должна иметь единицу в нужной позиции, чтобы послать сигнал «начать преобразование» по линии STRT. Маска MASK должна иметь единицу в разряде, соответствующем сигналу «занято». Если программа организует достаточно длинную задержку (для гарантии завершения передачи), то проверять сигнал «занято» не требуется.

Пример 9. Телетайп. В системах на основе микропроцессора Intel 8080 можно использовать как программный, так и аппаратно реализованный интерфейс, выполняющий необходимые для телетайпа преобразования данных и форма-

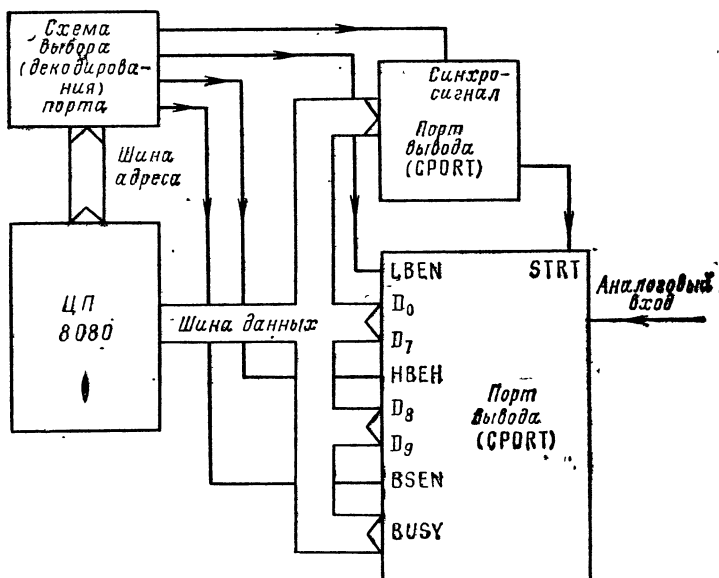


Рис. 8.51. Интерфейс между ЦП Intel 8080 и аналого-цифровым преобразователем AD7570 фирмы Analog Devices

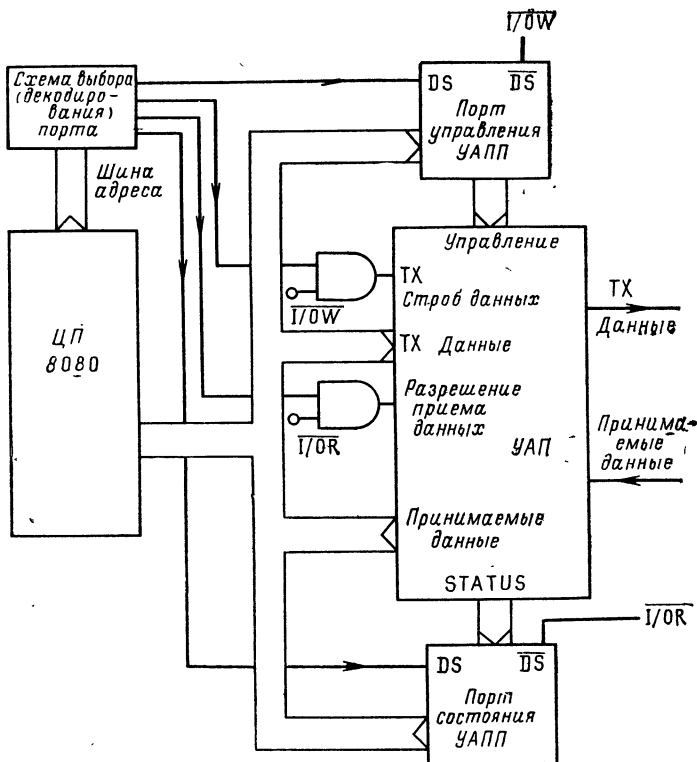


Рис. 8.52. Интерфейс между Intel 8080 и УАПП с гристабильными выходами (УАПП содержит порты ввода и порты вывода данных: порт ввода для слова состояния и порт вывода для управления)

тизацию сигналов. Аппаратный интерфейс обычно состоит из УАПП, имеющего выходы с тремя состояниями, порта ввода, необходимого для проверки сигналов состояния, и порта вывода для выбора одного из режимов работы УАПП. Для программного интерфейса требуется только один входной и один выходной разряды, но при этом программирование ввода-вывода становится сложнее. Низкая стоимость и доступность совместимых с микропроцессорами УАПП привели к широкому распространению аппаратного интерфейса.

На рис. 8.52 показан интерфейс между микропроцессором Intel 8080 и УАПП, имеющим выходы с тремя состояниями. Устройство асинхронного приемопередатчика содержит все порты ввода-вывода, а также входы для адресации ввода-вывода, которые можно непосредственно присоединить к схеме выбора порта.

В приведенной ниже программе передачи данных процессор ждет сигнала «вывод готов» (буфер передатчика пуст), а затем посылает данные.

```

;
; Передать данные на гелетайп через УАПП
;
; Ожидать сигнала «буфер передатчика пуст»
STRBE:  IN    SPORT      ;Считать состояние
        ANI    TMASK     ;Буфер передатчика пуст?
        JZ     STRBE     ;Нет, ожидать
;
; Послать данные в регистр передатчика
        LDA    TDATA     ;Подготовить данные
        OUT    TPORT     ;Послать данные в регистр передатчика

```

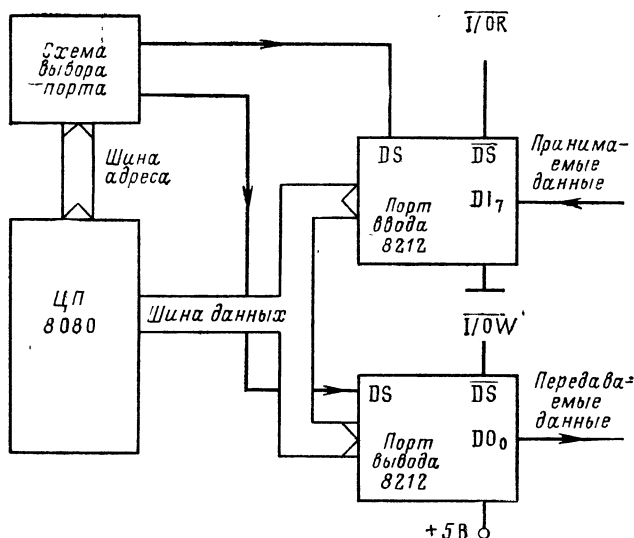


Рис. 8.53. Интерфейс между Intel 8080 и телетайпом (для принимаемых данных нужен один разряд порта ввода, а для передаваемых данных — один разряд порта вывода)

Программа, получающая данные, прежде чем запоминать их, должна проверить, есть ли в данных ошибки. Типичная программа ввода данных выглядит следующим образом:

;Получить данные от телетайпа через УАПП

;Ожидать сигнала «буфер приемника заполнен»

```

SRRDF: IN      SPORT      ;Считать состояние
        ANI      RMASK     ;Буфер приемника заполнен?
        JZ       SRRDF     ;Нет, ожидать
    
```

Проверить, есть ли ошибки

```

IN      SPORT      ;Считать состояние
ANI     EMASK      ;Маскированием выделить
                        разряды ошибок
JNZ     RERR       ;Ошибка, если хотя бы один бит = 1
IN      RPORT      ;Считать данные
STA     RDATA
    
```

Подпрограмма по адресу RERR должна определить характер ошибок и выполнить соответствующие действия.

На рис. 8.53 изображен прямой интерфейс между процессором Intel 8080 и телетайпом. На рис. 8.39 приведена блок-схема программы передачи данных. В указанной блок-схеме стартовый бит — логический 0, а стоповые — логические 1. Стоп-биты получают путем установки в единицу триггера переноса при каждом проходе цикла передачи данных. Приведем программу передачи 8-разрядных символов ASCII без разряда четности; предполагается, что последовательная линия вывода соответствует нулевому разряду, а подпрограмма DELAY обеспечивает задержку длительностью 9,1 мс между процедурами передачи битов, не меняя содержимого регистров,

```

;
; Последовательный вывод на телетайп с одним стартовым и двумя
; стоповыми битами с задержкой 9,1 мс между процедурами передачи битов (ли-
; ния вывода (0))
      ANA      A          ;Перенос = стартовый бит = 0
      LDA      TDATA      ;Считать символ из памяти
      RAL      ;Стартовый бит поместить в позицию линии
                        вывода
TRBIT:  MVI      B, 11      ;Счетчик = 11 (число разрядов)
      OUT      TPORT      ;Передать бит
      CALL     DELAY      ;Ожидать в течение 9,1 мс
      RAR      ;Поместить следующий бит в позицию линии
                        вывода
      STC      ;Перенос = 1 (для генерации стоповых битов)
      DCR      B
      JNZ      TRBIT

```

На рис. 8.40 приведена блок-схема программы приема данных. Программа должна обнаружить стартовый бит, ожидать в течение времени, равного половине длительности передачи бита (для выхода на центр импульса), преобразовать данные в параллельный формат и найти стоповые биты. Программа, приведенная ниже, не выполняет проверку на четность. Программа использует подпрограмму DLY2, которая обеспечивает задержку центрирования. Линии последовательного ввода соответствует бит 7.

```

;
; Последовательный ввод с телетайпа с одним стартовым и двумя
; стоповыми битами с задержкой 9,1 мс между процедурами приема
; битов (линия ввода 7)
; Обнаружение стартового бита (нуля)
;
;
; SRSTB  IN  RPORT      ;Считать последовательные данные с линии 7
;        ANA  A          ;Есть ли стартовый бит (0)?
;        JM  SRSTB      ;Нет, продолжить проверку
;
; Преобразование данных в параллельный формат
;
      CALL     DLY2      ;Ожидать в течение половины времени пере-
                        дачи сигнала (центрирование)
RCVBT:  MVI      B, 1000000B ;Поместить бит отсчета в старший разряд
      CALL     DELAY      ;Ожидать в течение 9,1 мс
      IN       RPORT      ;Считать бит данных
      RAL      ;Сохранить полученный бит в разряде переноса
      MOV      A, B
      RAR      ;Присоединить полученный бит к данным,
                        принятым ранее
      MOV      B, A
      JNC      RCVBT      ;Продолжать до тех пор, пока бит отсчета не
                        пройдет через все слово
      STA      RDATA
;
; Проверка правильности стоповых битов
;
      MVI      B, 2      ; Число стоповых битов = 2
STOPS:  CALL     DELAY      ; Ожидать в течение 9,1 мс
      IN       RPORT      ; Считать бит данных
      ANA      A          ; Является ли он стоповым битом (1)?
      JP       FRERR      ; Нет, ошибка формата
      DCR      B
      JNZ      STOPS

```

Приведенная программа считывает каждый бит данных только 1 раз. Если уровень шума на линии высокий, то для ввода данных можно применить программу, считывающую данные несколько раз за время передачи битов. Один из способов: читать данные 3 раза (например, в моменты, отстоящие на $1/4$, $1/2$ и $3/4$ длительности сигнала от его начала) и использовать метод голосования для определения истинного значения. Этот способ позволяет также сократить число ошибочных обнаружений стартового бита.

Генерация разрядов четности и проверка четности весьма просто выполняется процессором 8080 благодаря наличию специального признака четности. Например, для проверки на «четно» можно использовать следующие процедуры:

1. Генерация разряда четности

LDA TDATA

;Считать из памяти символ, седьмой разряд которого равен 0

ANA A

;Число единиц уже является четным?

JPE TRANS

ORI 10000000 B

;Нет, сделать четным установкой седьмого разряда в 1

TRANS: RAL

;Стартовый бит поместить в позицию, соответствующую линии передачи

2. Проверка на четность

ANA A

;Число единиц четно?

JPO PRERR

;Нет, обнаружена ошибка четности

STA RDATA

Для проверки на «нечетно» нужно изменить условия в командах перехода на противоположные.

Микропроцессор Motorola 6800

Микропроцессор Motorola 6800 ориентирован на использование в подсистеме ввода-вывода БИС последовательного или параллельного интерфейса. У процессора нет специальных команд ввода-вывода или сигналов управления им. Хотя и возможно использование ТТЛ-устройств, в большинстве случаев подсистемы ввода-вывода для этого процессора строятся на основе адаптера интерфейса периферийных устройств 6820.

Любая команда, связанная с обращением к памяти, может выполнить операцию ввода-вывода. Так, один из операндов команд СЛОЖЕНИЕ, ВЫЧИТАНИЕ, И, ИЛИ, ИСКЛЮЧАЮЩЕЕ ИЛИ и др. находится в аккумуляторе, а другой может находиться в порте ввода. Для организаций ввода-вывода используются следующие команды:

ЗАПОМНИТЬ (STORE) — передает восемь разрядов данных из аккумулятора в порт вывода.

ЗАГРУЗИТЬ (LOAD) — передает восемь разрядов данных из порта ввода в аккумулятор.

СБРОС (CLEAR) — очищает порт вывода.

ПРОВЕРКА (TEST) — устанавливает значения признаков в соответствии с данными, находящимися в порте ввода.

СРАВНИТЬ (COMPARE) — устанавливаемые признаки совпадают с признаками, устанавливаемыми при вычитании данных, находящихся в порте ввода, из содержимого аккумулятора.

ПРОВЕРКА РАЗРЯДА (BIT TEST) — устанавливаемые признаки совпадают с признаками, устанавливаемыми при команде И, примененной к данным в порте ввода и содержимому аккумулятора.

Разработчику следует обращать особое внимание на такие ситуации, когда функционирование портов ввода-вывода отличается от поведения ячеек памяти. Так, возможно, что входные данные не зафиксированы, а выходные — не буферизуются. Вообще, следует избегать записи в порт и чтения из порта, пока не произошла фиксация или буферизация. Заметим, что такие операции, как «проверка», «сдвиг» и «дополнение», включают как цикл чтения, так и цикл записи. При использовании этих операций для управления портами необходима осторожность.

При разработке подсистемы ввода-вывода на основе адаптера интерфейса периферийных устройств необходимо учитывать следующее:

1. Система декодирования должна отличать адреса адаптера от адресов памяти. Как было отмечено ранее, во многих микро-ЭВМ для этой цели используются адресные линии A_{15} и A_{14} . Этот способ удобен для применения, так как у каждого PIA имеется одна линия выбора кристалла с низким действующим значением напряжения и две — с высоким. Чтобы отличить ввод-вывод от обращения к памяти, требуется одна линия выбора кристалла с высоким действующим значением и одна — с низким; еще одна линия выбора (с высоким действующим значением) остается для адресации различных адаптеров.

2. Линии RS (выбора регистра) определяют внутренние адреса PIA так, как описано в табл. 8.1. Если, как обычно, линии RS0 и RS1 соединены с адресными линиями A_0 и A_1 соответственно, то адрес регистра данных A можно поместить в индексном регистре и все регистры адаптера адресовать с помощью индексированных смещений:

- 0, X — регистр данных или регистр направления передачи A;
- 1, X — регистр управления A;
- 2, X — регистр данных или регистр направления передачи B;
- 3, X — регистр управления B.

Такой метод удобен при настройке (определении конфигурации) адаптера.

3. Адресация PIA должна быть непротиворечивой. В полном декодировании нет необходимости, если только адреса ввода-вывода не расположены один за другим в небольшом пространстве адресов. Во многих подсистемах ввода для адресации PIA используется выбор с помощью линий. В этом случае каждый адаптер соединен с отдельной линией адреса (рис. 8.54). Так как PIA использует разряды A_0 и A_1 внутренним образом, то разряды от второго до тринадцатого можно использовать для выбора одного из 24 PIA, как показано в табл. 8.10.

Если для адресации памяти используется больше разрядов, то лучше использовать метод простого декодирования адресов PIA.

4. Максимальное время задержки PIA меньше соответствующего времени для стандартных ОЗУ 6810 и ПЗУ 6830; но, возможно, при появлении более быстрых модулей памяти время задержки PIA необходимо будет принимать во внимание.

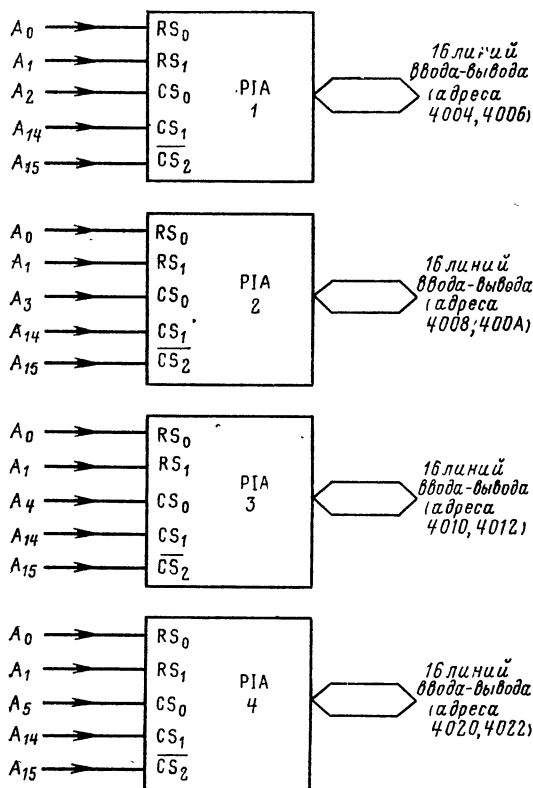


Рис. 8.54. Использование выбора с помощью линии адреса для адресации адаптеров интерфейса периферийных устройств (PIA) (здесь входы CS_0 различных PIA соединены с различными линиями адреса. Память и ввод-вывод различаются с помощью сигналов CS_1 и CS_2 . Входы RS_0 и RS_1 используются для адресации регистров PIA)

Таблица 8.10. Адреса PIA в системе, в которой используется выбор с помощью линий шины адреса

Линия адреса ¹ , соединенная с CS	Адреса PIA (шестнадцатичные)		Линия адреса ¹ , соединенная с CS	Адреса PIA (шестнадцатичные)	
	A ₁₅ =0, A ₁₄ =1	A ₁₅ =1, A ₁₄ =0		A ₁₅ =0, A ₁₄ =1	A ₁₅ =1, A ₁₄ =0
A ₂	4004—4007	8004—8007	A ₈	4100—4103	8100—8103
A ₃	4008—400B	8008—800B	A ₉	4200—4203	8200—8203
A ₄	4010—4013	8010—8013	A ₁₀	4400—4403	8400—8403
A ₅	4020—4023	8020—8023	A ₁₁	4800—4803	8800—8803
A ₆	4040—4043	8040—8043	A ₁₂	5000—5003	9000—9003
A ₇	4080—4083	8080—8083	A ₁₃	6000—6003	A000— A003

¹ Память и ввод-вывод различаются с помощью сигналов на линиях A₁₄ и A₁₅; линии A₀ и A₁ адресуют внутренние регистры PIA.

5. Так как PIA не имеет фиксатора входных данных, то для их фиксации может потребоваться ТТЛ-фиксатор; он используется в том случае, когда данные доступны только в течение короткого времени.

6. PIA не может нагружать линии вывода непосредственно. Буфер 8T97 обеспечивает более высокий нагрузочный ток.

7. Ввод «разрешение» (ENABLE) адаптера обычно соединен с линией фазы Ф₂ генератора тактовых сигналов, так как сигнал «разрешение» используется для распознавания сигналов прерывания и определения длительности строб-сигналов. Следовательно, сигнал «разрешение» должен подаваться регулярно, если используются строб-сигналы, и не зависеть от состояния процессора. Ввод «разрешение» не должен соединяться с линией VMA, так как при таком соединении адаптер не синхронизирован во время останова (ожидания) процессора.

Разнообразные команды процессора Motorola 6800 работают с данными, находящимися в портах ввода-вывода, точно так же, как с данными в памяти. Циклы команд такие же, как циклы, описанные в гл. 7.

Интерфейс между простыми периферийными устройствами и процессором Motorola 6800

Пример 1. Двухпозиционный переключатель.

Для однополюсных переключателей (одно- и двухпозиционных) требуется только один разряд ввода PIA. Фиксатор данных не нужен. Приведенная ниже программа настраивает PIA, используя для ввода сторону A адаптера, не содержащую буфера.

LDX #PIADRA	Считать из памяти базовый адрес PIA
CLR 1,X	Получить доступ к регистру направлений передачи стороны A
LDAA # DIRS	Установить направления
STAA X	
LDAA #%00000100	Получить доступ к регистру данных
STAA 1,X	Настроить регистр управления PIA

Первая часть приведенной программы требуется для получения доступа к регистру направлений пересылки. Сброс регистра управления A (разряд 2 = 0) приводит к тому, что адрес 0 в PIA указывает на регистр направлений A. Остальная часть программы требуется для загрузки этого регистра (1 соответствует выводу, 0 — вводу) и настройке регистра управления так, чтобы все прерывания были запрещены, а адрес 0 указывал на регистр данных (разряд 2 регистра управления находится в состоянии 0). Линии управления здесь не используются.

Чтобы определить, замкнут переключатель или нет, требуются три команды:

LDAA	PIADRA	Получить данные от переключателя
ANDA	#MASK	Маскированием выделить разряд, соответствующий переключателю
BEQ	CLSD	Переход, если переключатель выключен

Маска **MASK** имеет единицу в разряде, соответствующем переключателю, и нули — в остальных разрядах. Если переключатель соединен с разрядом 7, то команда **И** не нужна, так как при загрузке аккумулятора будет установлен признак отрицательного результата:

LDAA	PIADRA	Получить данные от переключателя
BPL	CLSD	Переход, если переключатель выключен

С помощью команды **ИСКЛЮЧАЮЩЕЕ ИЛИ** можно обнаружить изменение в положении хотя бы одного из группы переключателей, например, таким образом:

LDAA ,	OLD	Считать из памяти данные о старом положении переключателей
EORA ,	PIADRA	Есть ли изменения в новых данных о положении переключателя?
JZ	NOCH	
LDAB	PIADRA	Заменить старые данные на новые, если есть изменения
STAB	OLD	

Теперь аккумулятор **A** содержит единицу в каждом из разрядов, соответствующих переключателям, положение которых изменилось.

Пример 2. Восьмипозиционный переключатель. Для 8-разрядного селекционного переключателя необходимо полностью использовать сторону **A** адаптера, Команда очистки регистра направлений делает все линии данных входными:

CLR X Все линии настроить на ввод

Приведенная ниже программа ожидает, когда переключатель окажется в одном из положений, и определяет это положение (предполагается, что разряд 0 входа соответствует положению 0 переключателя и т. д.).

	LDAA	#% FF	
CHSW	CMPA	PIADRA	Находится ли переключатель в каком-нибудь из положений?
	BNE	CHSW	Нет, ожидать
	CLRB		Положение переключателя = 0
	LDAA		Считать данные от переключателя
SRPOS	RORA		Следующий разряд = 0?
	BCC	FOUND	Да, положение установлено
	INCB		Нет, положение переключателя увеличить на единицу
	BRA	SRPOS	
FOUND	SWI		

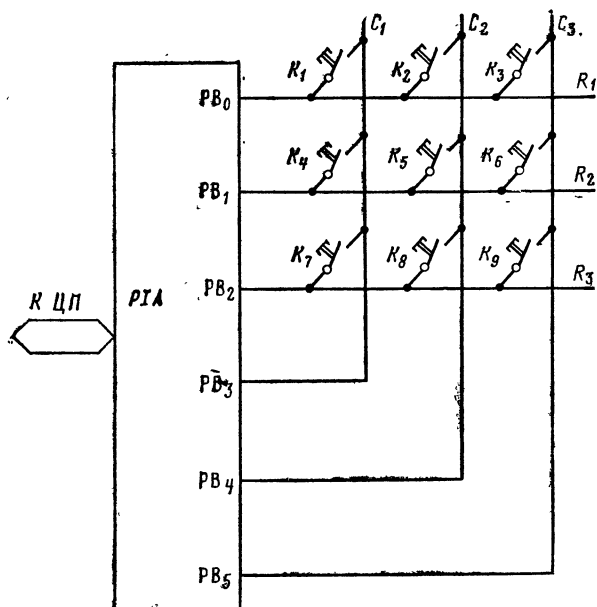
В конце работы этой программы положение переключателя (от 0 до 7) будет храниться в аккумуляторе **B**.

Пример 3. Клавиатура 3×3 без кодирования. На рис. 8.55 показана простая клавиатура 3×3 , соединенная со стороной **B** **PIA**. Приведенная ниже программа идентифицирует нажатую клавишу (см. рис. 8.30), изменяя режим работы линий (ввод на вывод и наоборот) с тем, чтобы получить уникальный 6-разрядный код (см. табл. 8.7). Алгоритм идентификации следующий:

1) заземлить (обнулить) столбцы и сохранить данные с линией строк в качестве трех младших разрядов кода;

2) заземлить строки и сохранить данные с линий столбцов в качестве трех старших разрядов кода;

Рис. 8.55. Интерфейс между Motorola 6800 и клавиатурой 3×3 без кодирования. (как линии строк, так и линии столбцов соединены со стороной В PIA: линии строк — с двумя младшими разрядами, а линии столбцов — с тремя следующими разрядами)



3) найти в таблице 6-разрядный код и соответствующий ему номер клавиши

Программа ввода с клавишного пульта

Настроить PIA так: линии столбцов — на вывод; линии строк — на ввод

CLR PIACRB

Получить доступ к регистру направлений передачи

LDA # % 00111000

Линии строк — на ввод; линии столбцов — на вывод

STAA PIADRB

LDA # % 00000100

Получить доступ к регистру данных

STAA PIACRB

Заземлить (обнулить) линии столбцов и сохранить данные с линий строк

LDA # % 11000111

SRKEY STAA PIADRB

Обнулить линии столбцов

LDAB PIADRB

Считать данные о контактах

ANDB # % 00000111

Маскированием выделить разряды строк

CMPB # % 00000111

Клавиша нажата?

BEQ SRKEY

Нет, продолжать проверку

Настроить PIA так: линии столбцов — на ввод, линии строк — на вывод

CLR PIACRB

Получить доступ к регистру направлений

LDA # % 00000111

Линии столбцов — на ввод, линии строк — на вывод

STAA PIADRB

LDA # % 00000100

Получить доступ к регистру данных

STAA PIACRB

*
 * Заземлить (обнулить) линии строк и сохранить данные с линий столбцов
 *
 LDAA $\# \% 11111000$
 STAA PIADRB Обнулить линии строк
 LDAA PIADRB Считать данные о контактах
 ANDA $\# \% 00111000$ Маскированием выделить разряды столбцов
 CMPA $\# \% 00111000$ Клавиша нажата?
 BEQ SRKEY Нет, начать просмотр с начала
 ABA Объединить данные, полученные в результате
 анализа строк и столбцов, в 6-разрядный код

*
 *
 * Полученный код использовать для поиска по таблице
 *
 LDX $\#$ KEYTAB Считать из памяти адрес таблицы
 LDAB $\#$ Номер клавиши, $\equiv 1$
 SRTAB CMPA X Содержимое элемента таблицы совпадает с
 кодом?

*
 BEQ FOUND Да, клавиша найдена
 INCB Нет, номер клавиши увеличить
 на единицу
 *
 INX
 CPX $\#$ KEYTAB + 9 Таблица просмотрена полностью?
 BNE SRTAB Нет, продолжить просмотр
 BRA ERROR Да, ошибка
 FOUND SWI
 KEYTAB FCB $\$36, \$2E, \$1E, \$35, \$2D, \$1D$
 FCB $\$33, \$2B, \$1B$

Следует помнить, что сторона В адаптера имеет буфер. Может оказаться, что необходимо программно повысить потенциал линий ввода, объявляя для этого все линии данных выводными и помещая единицы в нужные разряды. Можно использовать и сканирующую программу, аналогичную программе для МП 8080 фирмы Intel.

Строб-сигнал можно получать, соединяя все линии столбцов схемой И-НЕ и подавая результат на линию управления CB1. После идентификации нажатой клавиши программа должна заземлить линии строк для того, чтобы по следующему нажатию выдавался строб-сигнал. Так как нажатие клавиши вызывает переход от низкого уровня напряжения к высокому на линии CB1, то при настройке PIA бит 1 регистра управления должен устанавливаться в единицу. Процедура выглядит следующим образом:

*
 * Настроить PIA на работу с клавиатурой при использовании строб-сигнала
 *
 CLR PIADRB Получить доступ к регистру
 направлений
 LDAA $\# \% 00000111$ Линии строк — на вывод, линии столбцов — на
 ввод

*
 STAA PIADRB
 LDAA $\# \% 00000110$ Получить доступ к регистру данных, устано-
 вить режим возникновения строга по перед-
 нему фронту

*
 STAA PIACRB
 Строб-сигнал установит бит 7 регистра управления В. Для проверки этого
 разряда можно использовать следующую программу:
 LDAA PIACRB Есть ли строб-сигнал?
 BMI KEYCL Если есть, то идентифицировать клавишу
 Этот разряд будет сброшен при чтении данных из регистра данных,
 380

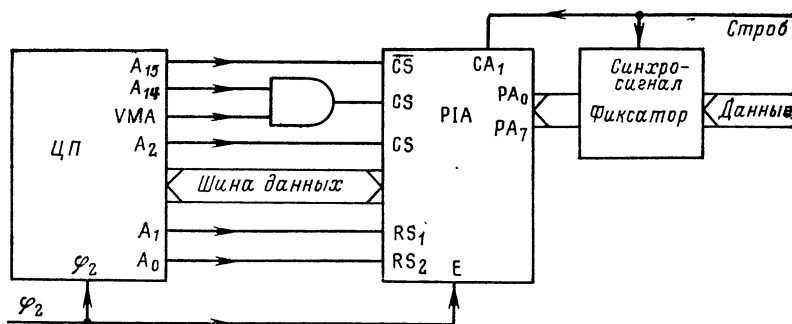


Рис. 8.56. Интерфейс между Motorola 6800 и кодирующей клавиатурой (PIA за-
нимает адреса с 4004 по 4007. Данные от клавиатуры фиксируются в восьмираз-
рядном синхронизируемом фиксаторе)

Пример 4. Клавиатура с кодированным выходом и строб-сигналом. На рис. 8.56 показано, как используются фиксатор и PIA для создания интерфейса с кодирующей клавишным пультом, имеющим 8-разрядный выход и строб-сигнал (действующее значение — низкий уровень). По строб-сигналу фиксируются данные (PIA не имеет фиксатора на вводе), и происходит переход на линии управления CA1.

В данном случае устанавливать разряд 1 регистра управления не нужно, так как возникновение строб-сигнала выражается переходом от высокого уровня напряжения к низкому. В следующей программе используется сторона A PIA.

*	CLR	PIACRA	Получить доступ к регистру направлений
	CLR	PIADRA	Все линии настроить на ввод
	LDAA	# % 00000100	Получить доступ к регистру данных
	STAA	PIACRA	

Программа ввода выглядит так:

*	Проверка строб-сигнала		
CHSTB	LDAA	PIACRA	Строб-сигнал имеет действующее значение (0)?
	BPL	CHSTB	Нет, продолжать проверку
*	Считать данные		
	LDAA	PIADRA	Считать данные с клавиатуры

Если клавиатура использует отрицательную логику, то в программе необходимо использовать команду ДОПОЛНЕНИЕ.

Пример 5. Один индикатор. Для управления одиночным индикатором достаточно одного выходного разряда PIA. Следует обратить внимание на то, что PIA не имеет выходного фиксатора. От полярности включения индикатора зависит, какие команды необходимы для зажигания и выключения индикатора. С помощью следующей программы можно управлять состояниями сразу нескольких индикаторов:

LDAA	# MASK	Подготовить данные для индикатора
STAA	PIADRA	Послать данные на индикатор

Маска MASK содержит в разряде, соответствующем индикатору, нуль или единицу в зависимости от полярности включения индикатора. Если линии вывода буферизованы, то очистить регистр данных или взять дополнение к его содержанию можно с помощью одной команды:

CLR	PIADRA	На все индикаторы подать нули
COM	PIADRA	Изменить состояние каждого индикатора на противоположное

Для настройки PIA необходимо, чтобы все линии были выходными.

CLR	PIACRA	Получить доступ к регистру направлений
LDA	# \$FF	Все линии — на вывод
STAA	PIADRA	
LDA	# % 00000100	Получить доступ к регистру данных
STAA	PIACRA	Настроить PIA

С помощью логических операций можно изменять отдельные разряды маски, например, следующим образом:

ORA #MASK устанавливает в единицу разряд, соответствующий единичному разряду маски;

ANDA #MASK сбрасывает разряд, соответствующий нулевому разряду маски;

EORA #MASK изменяет на противоположное значение разряд, соответствующий единичному разряду маски.

С помощью этих команд можно включать или выключать отдельные индикаторы или изменять их состояние на противоположное.

Пример 6. Семисегментный индикатор. Семисегментный индикатор (без декодирования) можно подключить к PIA фирмы Motorola так, как показано на рис. 8.57. Программа должна преобразовывать выходные данные в требуемую форму и записывать их в регистр данных PIA. Логические уровни можно изменять на противоположные с помощью команды дополнения.

Для управления несколькими индикаторами (панелями) можно воспользоваться счетчиком и декодером (рис. 8.58). На этом рисунке линия управления CB2 играет роли линии сигнала BYTE OUT, синхронизирующего счетчик. Требуется, чтобы разряды регистра управления PIA принимали следующие значения:

разряд 5 = 1, т. е. CB2 — выход;

разряд 4 = 0, т. е. CB2 — строб;

разряд 3 = 1, т. е. CB2 приобретает прежнее значение по сигналу «разрешение» (ENABLE), поступившему в момент, когда CB2 имеет низкий уровень; это означает, что если линия «разрешение» соединена с выходом Φ_2 генератора тактовых импульсов, то строб имеет длительность, равную периоду тактирования.

Приведем пример программы, настраивающей PIA.

CLB	PIACRB	Получить доступ к регистру направлений
LDA	# \$FF	Все линии — на вывод
STAA	PIADRB	
LDA	# % 0010110	Определить CB2 как строб вывода
STAA	PIACRB	

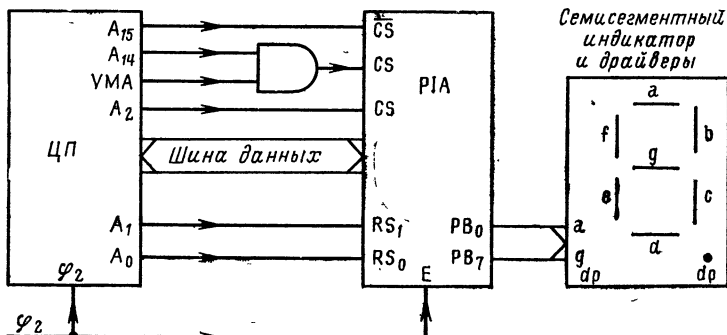


Рис. 8.57. Интерфейс между Motorola 6800 и семисегментным индикатором без декодирования (PIA занимает адреса с 4004 по 4007. Семисегментный индикатор связан со стороной В. Старший бит управляет сегментом десятичной точки, а остальные — прочими сегментами)

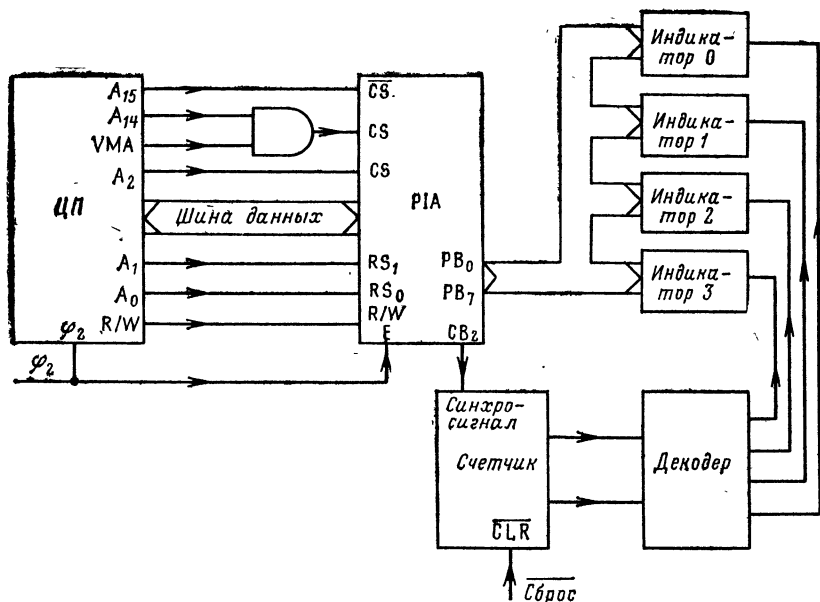


Рис. 8.58. Интерфейс между Motorola 6800 и набором семисегментных индикаторов с использованием счетчика и декодера (по линии управления CB_2 на счетчик посылается синхронизирующий сигнал после каждой операции вывода. Счетчик и декодер определяют, какой индикатор должен светиться)

Бит 2 регистра управления устанавливается равным единице для обеспечения доступа к регистру данных.

Другой способ управления индикаторами состоит в использовании второй стороны PIA, как показано на рис. 8.59. Если, например, для управления используется сторона *B* и индикатор возбуждается с помощью логического нуля,

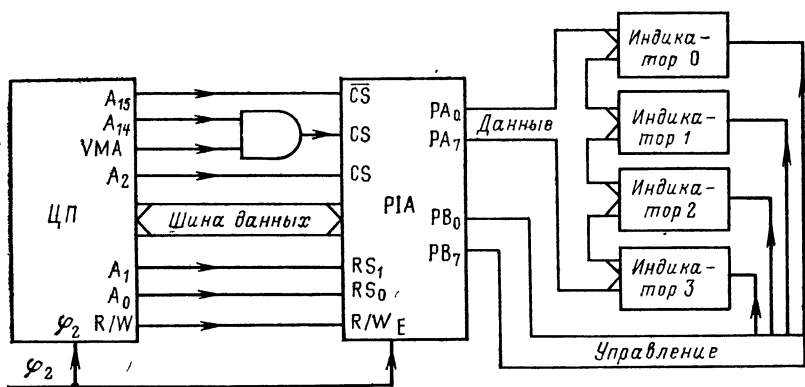


Рис. 8.59. Интерфейс между Motorola 6800 и набором семисегментных индикаторов с использованием порта для управления индикаторами (данные передаются на индикаторы со стороны *A* PIA; сторона *B* определяет, какой индикатор должен светиться)

то приводимая ниже программа выдаст на восемь цифровых индикаторов во семь слов данных, расположенных в памяти начиная с ячейки DSPLY.

* Вывод данных на панель с восьмью цифровыми индикаторами

* Шаг 1. Настроить PIA

LDX #PIADRA

CLR 1,X

Получить доступ к регистру направлений сторо-
ны А

CLR 3,X

Получить доступ к регистру направлений сто-
роны В

LDAA # \$FF

Все линии — на вывод

STAA X

STAA 2,X

LDAA # % 00000100

Получить доступ к регистру данных

STAA 1,X

Настроить обе стороны PIA

STAA 3,X

* Шаг 2. Вывод данных на индикаторы

LDX #DSPLY

Указатель = DSPLY

LDAB # \$7F

Установка на старший индикатор

SEC

Перенос = 1 (для последующего циклического
сдвига)

STAB PIADRB

DSPLY1 LDAA X

Подготовить данные для индикатора

STAA PIADRA

Послать данные на индикатор

JSR DELAY

Обеспечить достаточную длительность сигнала

INX

ROR PIADRB

Установка на следующий индикатор

BCS DSPLY1

Продолжать, если выведены не все данные

В табл. 8.11 указано, каким должно быть содержимое регистра для включе-
ния различных цифровых индикаторов. Подпрограмма DELAY не должна изме-
нять содержимое ни одного из регистров.

Пример 7. Цифро-аналоговый преобразователь. К МП Motorola 6800 мож-
но подключить ЦАП, аналогичный преобразователю AD7522 фирмы Analog
Devices так, как показано на рис. 8.60. Преобразователь использует десять ли-
ний вывода данных и две линии управления: одну для строб-сигнала, по которо-
му устройство получает данные (необходим переход от низкого уровня к высоко-
му), и другую — для загрузки ЦАП (продолжительность сигнала высокого уров-
ня должна быть не менее 0,5 мкс).

**Таблица 8.11. Содержимое регистра, необходимое для зажигания мульти-
плексно подключенных индикаторов**

Номер инди- катора (управляю- щий бит)	Содержимое регистра (шестнадцатирчное)		Номер инди- катора (управляю- щий бит)	Содержимое регистра (шестнадцатирчное)	
	Высокое действующее значение	Низкое действующее значение		Высокое действующее значение	Низкое действующее значение
0	01	FE	4	10	EF
1	02	FD	5	20	DF
2	04	FB	6	40	BF
3	08	F7	7	80	7F

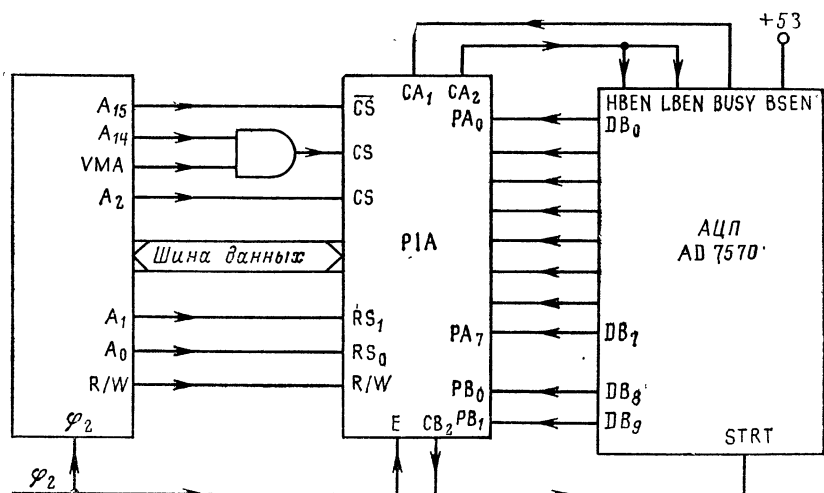


Рис. 8.60. Интерфейс между Motorola 6800 и цифро-аналоговым преобразователем AD7522 фирмы Analog Devices [на линию управления CB_2 подаются строб-сигналы как для младшего, так и для старшего байта. По линии управления CA_2 поступает сигнал LOAD DAC (загрузить преобразователь)]

В этом интерфейсе используются многие особенности PIA. Для стороны А линия CA_2 является последовательным выводом, снабженным фиксатором, с которого может выдаваться сигнал LDAC. Для стороны В линия CB_2 — короткий строб записи, по которому 10 бит данных помещается в буфер преобразователя (после того, как ЦП передал все 10 бит адаптеру). Разряды регистра управления стороны А имеют следующие значения:

- разряд 5 = 1, т. е. CA_2 — выход;
 - разряд 4 = 1, т. е. CA_2 — последовательный вывод с фиксацией, значение на котором равно значению разряда 3 регистра управления;
 - разряд 3 сначала равен нулю (запрещение работы ЦАП).
- Разряды регистра управления стороны В имеют следующие значения:
- разряд 5 = 1, т. е. CB_2 — выход;
 - разряд 4 = 0, т. е. CB_2 — строб записи;
 - разряд 3 = 1, поэтому длительность строб-сигнала записи равна одному периоду тактовых сигналов.

Выбор такой конфигурации PIA обусловлен следующими факторами:

1. Строб-сигнал записи может вырабатываться только стороной А адаптера. Поэтому ЦП пользуется этой стороной для второй операции загрузки таким образом, что строб-сигнал возникнет только после того, как все десять разрядов данных станут доступными.
2. Сигнал с высоким действующим значением может генерировать только линия вывода с фиксатором. Поэтому в данной конфигурации вырабатывается сигнал LDAC. Строб-сигналы имеют низкое действующее значение.
3. Так как от PIA поступает только незначительное число управляющих сигналов, то параллельная загрузка обоих байтов оказывается проще, чем загрузка по одному байту.
4. Стробы байтов (byte strobes) вырабатываются по фронту и для них требуется только короткий сигнал. Достаточно сигнала, управляемого входом «разрешение» PIA и длящегося один период тактовых сигналов.

Заметим, что при записи данных в регистр данных А строб-сигнал не вырабатывается. Сигнал «сброс» обнуляет все разряды управления и настраивает линию управления на ввод. Если использование сигнала «сброс» или изменение зна-

чений разрядов управления затруднительно, то может оказаться необходимым аппаратно управлять работой преобразователя.

Приведем текст программы.

```
*
* Настроить PIA для работы с ЦАП
*
      LDAA    #%00110000    Получить доступ к регистру направлений
*                               стороны A
      STAA    PIACRA
      CLR     PIACRB        Получить доступ к регистру направлений
*                               стороны B
      LDAB    #,$FF
      STAB    PIADRA        Все линии — на вывод
      STAB    PIADRB
      LDAA    #%00110100    Настроить сторону A на режим
*                               LDAC = 0
      STAA    PIACRA
      LDAA    #%00101100    Настроить сторону B на короткий строб
*                               записи
      STAA    PIACRB
*
* Послать данные преобразователю
*
      LDAA    LBYTE        Младшие восемь разрядов — на сторону A
      STAA    PIADRA
      LDAA    HBYTE        Старшие два разряда — на сторону
*                               B и строб-сигнал — на ЦАП
      STAA    PIADRB
*
* Загрузить ЦАП по сигналу
*   на CA2 (LDAC)
*
      LDAA    #%00111100    LDAC = 1
      STAA    PIACRA
      LDAA    #%00110100    LDAC = 0
      STAA    PIACRA
```

Программа вырабатывает сигнал LDAC с высоким действующим значением. Длительность сигнала значительно превышает необходимый минимум.

Пример 8. Аналого-цифровой преобразователь. Аналого-цифровой преобразователь, аналогичный преобразователю AD7570 фирмы Analog Devices, может подключаться к системе на основе процессора Motorola 6800 так, как показано на рис. 8.61. Преобразователь использует десять линий данных и три линии управления, одна из которых (ввод) сигнализирует о завершении преобразования, а две другие (вывод) инициируют преобразование и открывают тристабильные выходы. Линия «разрешение сигнала занято» (BUSY ENABLE) находится под постоянным высоким потенциалом, так что сигнал «занято» (BUSY) можно использовать в качестве сигнала «данные готовы» (сигнал «занято» принимает низкое значение, когда преобразование завершено). Сигнал «начать преобразование» и сигналы отпираания тристабильных выводов имеют высокое действующее значение; настройка PIA должна быть такой, чтобы обеспечить выдачу этих сигналов.

Разряды регистра управления стороны A имеют следующие значения:

разряд 5 = 1, поэтому CA2 — выход («разрешение» для старшего и младшего байтов);

разряд 4 = 1, поэтому CA2 — последовательный вывод с фиксацией, значение которого равно значению разряда 3 управляющего регистра (так как сигналы разрешения имеют высокое действующее значение);

разряд 0 сначала равен 0 (запрещение вывода);

разряд 1 = 1, поэтому переход от низкого уровня к высокому на линии BUSY установит разряд 7 в 1.

Разряды регистра управления стороны B PIA имеют следующие значения: разряд 5 = 1, поэтому CB2 — выход («начать преобразование»);

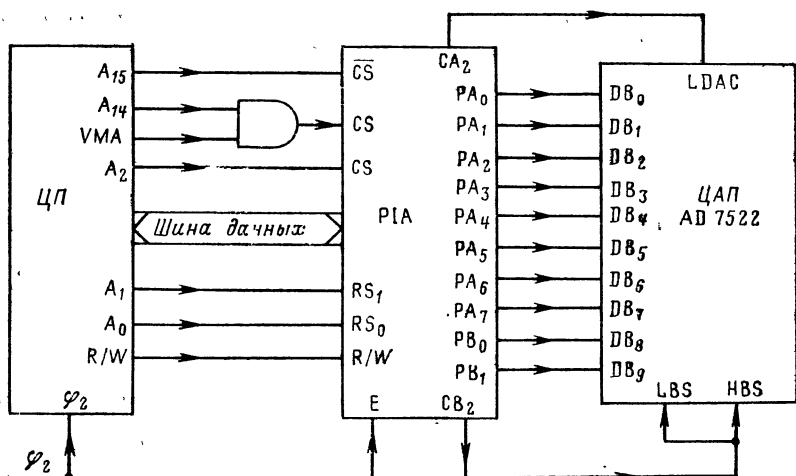


Рис. 8.61. Интерфейс между Motorola 6800 и аналого-цифровым преобразователем AD7570 фирмы Analog Devices [по линии управления CB₂ передается сигнал «начать преобразование» (START CONVERSION), по линии CA₁ — сигнал завершения преобразования. Линия управления CA₂ отпирает тристабильные входы преобразователя]

разряд 4 = 1, поэтому CB₂ — последовательный вывод с фиксацией, значение которого равно значению разряда 3 управляющего регистра (так как сигнал «начать преобразование» имеет высокое действующее значение); разряд 3 сначала равен 0 (запрещение преобразования). Приведем текст программы.

```

*
* Настроить PIA для работы с АЦП
*
LDAA    #%00110000    Получить доступ к регистрам направлений
*
STAA    PIACRA
STAA    PIACRB
CLR     PIADRA        Все линии — на ввод
CLR     PIADRB
LDAA    #%00110110    Настроить PIA
STAA    PIACRA
STAA    PIACRB
*
* Инициировать преобразование сигналом на линии STRT
*
LDAA    #%00111100    STRT = 1
STAA    PIACRB
LDAA    #%00110100    STRT = 0
STAA    PIACRB
*
* Ожидать окончания преобразования
*
CHBSY   LDAA    PIACRA    Проверить признак «занят»
        BPL     CHBSY     Ожидать, когда признак станет равен 1

```


* Отпереть вывод преобразователя, устанавливая разрешающие значения разрядов
 LDAА #%00111100 «Разрешение» = 1 — для получения данных от преобразователя
 * STAA PIACRA
 * Получить и сохранить данные
 LDAА PIADRA Считать младшие восемь разрядов данных
 STAA LBYTE Считать старшие два разряда данных
 LDAА PIADRB
 STAB HBYTE
 * Запереть вывод преобразователя
 LDAА #%00110100 Отсоединиться от преобразователя
 STAA PIACRA

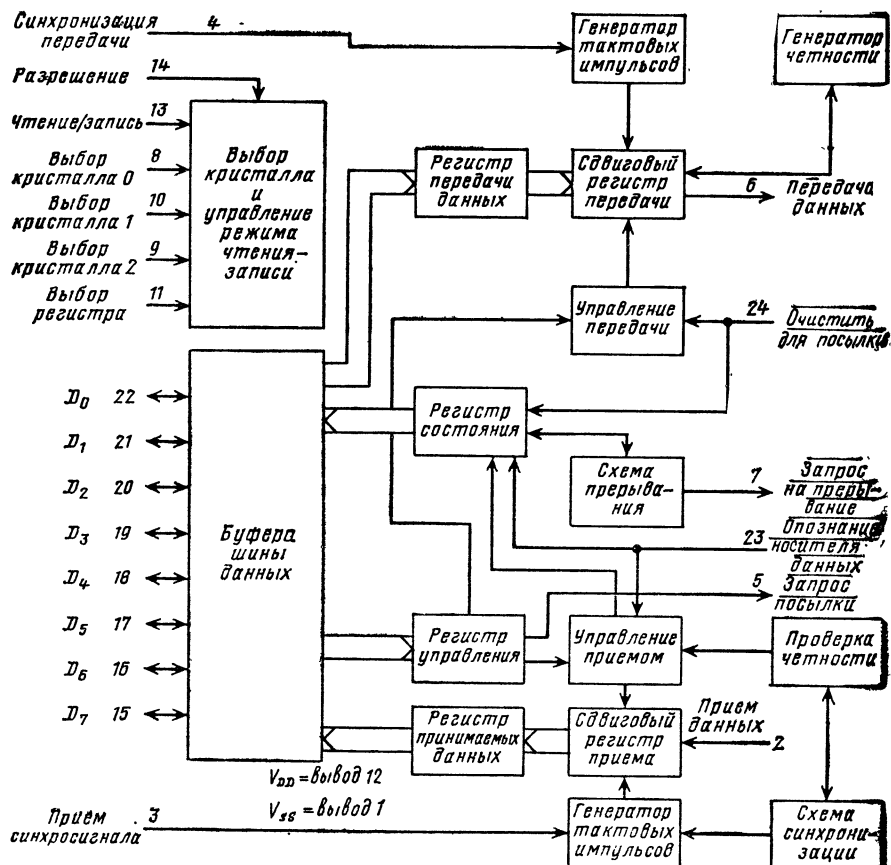


Рис. 8.62. Расширенная структурная схема асинхронного адаптера интерфейса связи Motorola MC6850

Таблица 8.12. Определение содержимого регистра ААИС

Номер линии шины данных	Адрес буфера			
	RS·R/W Регистр передачи данных (только запись)	RS·R/W Регистр приема данных (только чтение)	RS·R/W Регистр управ- ления (только запись)	RS·R/W Регистр состояния (только чтение)
0	Бит данных 0*	Бит данных 0	Выбор счетчика-дели- теля 1 (CR0)	Регистр приема данных (RDRF)
1	Бит данных 1	Бит данных 1	Выбор счетчика-дели- теля 2 (CR1)	Регистр передачи данных пуст (TDRE)
2	Бит данных 2	Бит данных 2	Выбор слова 1 (CR2)	Распознавание но- сителя данных (DCD)
3	Бит данных 3	Бит данных 3	Выбор слова 2 (CR3)	Очистить для по- сылки (CTS)
4	Бит данных 4	Бит данных 4	Выбор слова 3 (CR4)	Ошибка формата (FE)
5	Бит данных 5	Бит данных 5	Управление переда- чей 1 (CR5)	Потеря символа (OVRN)
6	Бит данных 6	Бит данных 6	Управление переда- чей 2 (CR6)	Ошибка четности (PE)
7	Бит данных 7***	Бит данных 7***	Разрешено прерыва- ние приема (CR7)	Запрос на преры- вание (IRQ)

* Ведущий бит=младший бит=бит 0; ** в режимах, использующих семь информацион-
ных разрядов и разряд четности, этот бит данных равен нулю; *** этот разряд не играет
роли для режимов, указанных во второй строке

Пример 9. Телетайп. В системах на основе МП Motorola 6800 интерфейс телетайпа можно выполнить как аппаратным, так и программным способами. Обычный аппаратный интерфейс — асинхронный адаптер интерфейса связи (рис. 8.62) — Asynchronous Communication Interface Adapter — УАПП с три-стабильными выводами, специально разработанный для применения совместно с процессором Motorola 6800 и низкоскоростным модемом 6860 той же фирмы. Асинхронный адаптер интерфейса связи (ААИС) занимает два адреса памяти и со-держит два регистра, доступных только для чтения («принять данные» и «состоя-ние»), и два регистра, доступных только для записи («передать данные» и «управ-ление»). В табл. 8.12 дано определение содержимого регистров, а в табл. 8.13 описываются разряды регистра управления (разряд 7 — бит разрешения преры-вания при приеме). На рис. 8.63 изображены типичные соединения ААИС с про-цессором.

Для иллюстрации приведем конфигурацию регистра управления ААИС при работе с телетайпом, в котором применяется 7-разрядный код с проверкой на «нечетно» и двумя стоповыми битами:
разряд 7 = 0 для запрета прерывания приема;

Таблица 8.13. Описание режимов, определяемых регистром управления ААИС

CR1	CR0	Функция
0	0	+1
0	1	+16
1	0	+64
1	1	Сброс (Master reset)

CR4	CR3	CR2	Функция
0	0	0	7 бит+бит контроля («четно») +2 стоп-бита
0	0	1	7 бит+бит контроля («нечетно») +2 стоп-бита
0	1	0	7 бит+бит контроля («четно») +1 стоп-бит
0	1	1	7 бит+бит контроля («нечетно») +1 стоп-бит
1	0	0	8 бит+2 стоп-бита
1	0	1	8 бит+1 стоп-бит
1	1	0	8 бит+бит контроля («четно») +1 стоп-бит
1	1	1	8 бит+бит контроля («нечетно») +1 стоп-бит

CR6	CR5	Функция
0	0	\overline{RTS} — низкий, прерывание передачи запрещено
0	1	\overline{RTS} — низкий, прерывание передачи разрешено
1	0	\overline{RTS} — высокий, прерывание передачи запрещено
1	1	\overline{RTS} — низкий, на выход передачи данных подается запирающий уровень напряжения. Прерывание передачи запрещено.

Примечание. Если разряд 7 регистра управления равен единице, то прерывание приема разрешено, если этот разряд равен нулю, то прерывание запрещено.

разряд 6 = 1, чтобы уровень \overline{RTS} был высоким (недействующим значением); согласно стандарту RS-232 это означает сигнал «запрос на посылку» (REQUEST TO SEND);

разряд 5 = 0 для запрета прерывания передачи;

разряд 4 = 0 для 7-разрядных символов;

разряд 3 = 0 для двух стоп-разрядов;

разряд 2 = 0 для дополнения числа единиц до нечетного;

разряды 1 и 0 равны единице для деления частоты тактовых импульсов на 16 (требуется внешний генератор с частотой 1760 ГГц).

Приведенная ниже программа сбрасывает ААИС, настраивает его, ожидает сигнала «регистр приема данных заполнен» (RDRE — RECEIVE DATA REGISTER FULL) и читает данные.

*

* Сбросить ААИС

*

LDAA # % 00000011

STAA ACIACR

Сбросить ААИС

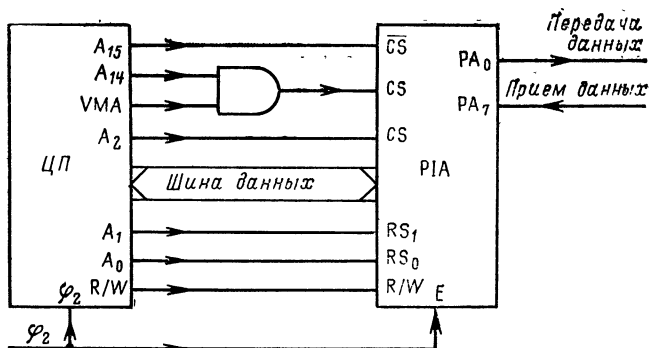


Рис. 8.64. Интерфейс между Motorola 6800 и последовательным УВВ (линия передачи соответствует разряду 0 стороны A PIA, линия приема — разряду 7 стороны A PIA)

равления, снабженная фиксатором, может управлять устройством считывания с перфоленты и перфоратором. В приведенной ниже программе предполагается, что последовательному вводу соответствует разряд 0, последовательному выводу — разряд 7, а подпрограммы DELAY и DLY2 обеспечивают временную задержку, равную длительности передачи бита и половине этой длительности соответственно, причем содержимое регистров и значения признаков сохраняются.

*

* Последовательный вывод на телетайп с одним стартовым битом,

* двумя стоп-битами и задержкой 9,1 мс между битами (линия

* вывода соответствует разряду 0).

	CLC		Перенос = стартовый бит = 0
	LDA	TDATA	Подготовить символ
	ROL		Стартовый бит — в позицию для передачи
	LDAB	# 11	Счетчик = 11 (число разрядов)
TRBIT	STAA	PIADRA	Передать 1 бит
	JSR	DELAY	Ожидать 9,1 мс между передачей битов
	RORA		Подготовить следующий бит к передаче
	SEC		Перенос = 1 (для генерации стоп-разрядов)
	DECB	TRBIT	
	BNE		

*

* Последовательный ввод с телетайпа с одним стартовым битом,

* двумя стоп-битами и задержкой 9,1 мс между битами (линия

* ввода соответствует разряду 7)

*

* Ожидать стартовый бит (0)

*

SRSTB	LDA	PIADRA	На последовательной линии ввода находится стартовый бит (0)?
-------	-----	--------	--

	BMI	SRSTB	Нет, продолжать проверку
--	-----	-------	--------------------------

*

* Преобразование данных в параллельный формат

*

	JSR	DLY2	Ожидать в течение половины времени передачи сигнала (центрирование)
--	-----	------	---

RCVBIT	LDA	#%10000000	Поместить бит отсчета в старший разряд
	JSR	DELAY	Ожидать 9,1 мс между передачей битов
	ROL	PIADRA	Бит данных — в разряд переноса
	RORA		Бит данных — в слово данных

	BCC	RCVBIT	Продолжать до тех пор, пока бит отсчета не пройдет через все слово
	STAA	RDATA	
*			
* Проверка	LDAA	#2	Число стоповых бит равно двум
STOPS	JSR	DELAY	Ждать 9,1 мс между передачей битов
	ROL	PIADRA	Является ли бит данных стоп-битом (единицей)
	BCC	FRERR	Нет, ошибка формата
	DECA		
	BNE	STOPS	

Процедура преобразования может выполнять проверку на четность следующим образом:

*			
* Преобразовать данные в параллельный формат и проверить на			
* четность (линия ввода соответствует разряду 7)			
*			
	JSR	DLY2	Ожидать в течение половины времени передачи сигнала (центрирование)
	LDAA	#%10000000	Поместить бит отсчета в старший разряд
RCVBIT	CLRB		Четность = 0
	JSR	DELAY	Ждать 9,1 мс между передачей битов
	EORB	PIADRA	ИСКЛЮЧАЮЩЕЕ ИЛИ для проверки на четность
	ROL	PIADRA	Бит данных — в разряд переноса
	RORA		Бит данных — в слово данных
	BCC	RCVBIT	Продолжать до тех пор, пока бит отсчета не пройдет через все слово
	TSTB		Проверить на четность
	BMI	PRERR	Ошибка, если число единиц нечетно
	STAA	RDATA	

Если используется проверка на «нечетно», то команду BMI PRERR следует заменить на команду BPL PRERR.

Чтобы подсчитать четность, нужно сдвигать вправо слово данных (точнее, его копию), прибавляя единицу к младшему разряду исходного слова данных каждый раз, когда в младшем разряде сдвигаемой копии окажется единица. При сложении младший разряд ведет себя так, как при выполнении команды ИСКЛЮЧАЮЩИЕ ИЛИ, поэтому признаки переноса не повлияют на окончательный результат.

*			
* Дополнение до «нечетно»			
*	CLRA		Четность = 0
	LDAB	TDATA	Выбрать данные из памяти
CALPAS	ABA		ИСКЛЮЧАЮЩИЕ ИЛИ для младшего разряда
	LSRB		Сдвинуть вправо копию данных
	BNE	CALPAR	Продолжать до тех пор, пока в копии не останутся одни нули

* Запомнить бит четности в старшем разряде данных

	RORA		Бит четности — в разряд переноса
	BCC	DONE	Число единиц уже четно?
	LDAB	TDATA	Нет, установить в 1 старший
	ORAB	#%10000000	разряд данных
	STAB	TDATA	

Для генерации бита, дополняющего число единиц до нечетного, следует заменить команду BCC на BCS,

8.7. ВЫВОДЫ

Организация ввода-вывода информации представляет собой весьма сложную задачу. Это связано с огромным разнообразием периферийных устройств, которые используются в микро-ЭВМ. Скорости работы, типы сигналов, структура управления различных УВВ сильно отличаются. Подсистема ввода-вывода должна преобразовывать вводимые данные и информацию о состоянии в форму, совместимую с форматом данных процессора, а также выводимые данные и управляющую информацию в форму, необходимую для работы УВВ.

Медленными устройствами, подобными переключателям или индикаторам, процессор может управлять и без точной синхронизации. Единственные трудности при асинхронной передаче данных связаны с переходными процессами, возникающими при изменении значений входных данных, а также с необходимостью фиксации данных при выводе в течение достаточного времени. Такими асинхронными УВВ, как клавишный пульт, мультиплексированные индикаторы, телетайпы и преобразователи, можно управлять по методу квити́рования. Центральный процессор должен определить готовность устройства выполнить передачу данных и послать устройству сигнал о завершении обмена. Процессор может управлять синхронными УВВ, согласуя работу своего генератора тактовых сигналов с генератором периферийного устройства. Для высокоскоростных передач можно использовать метод прямого доступа к памяти.

Для подсистемы ввода-вывода с большим числом портов требуется шинная структура, которая позволяет подсистеме ввода-вывода пользоваться шинами данных и адреса совместно с подсистемой доступа к памяти. При использовании изолированных линий ввода-вывода сигналы управления вводом-выводом обособлены; такие сигналы особенно полезны, когда порты ввода-вывода — ТТЛ-схемы. Метод ввода-вывода, адресуемого как память, требует резервирования некоторых адресов для ввода-вывода. Этот метод полезен, когда порты ввода-вывода реализованы в виде БИС, снабженных регистрами управления и состояния, а также фиксаторами и буферами. При встроенном вводе-выводе подсистема ввода-вывода является частью модулей памяти или процессора. С помощью этого метода можно строить одно- или двукристалльные микро-ЭВМ, имеющие узкую сферу применения, однако расширение системы становится при этом затруднительным.

Подсистемы ввода-вывода можно упростить, если использовать стандартные схемы ТТЛ и специальные устройства на БИС. Триггеры, мультивибраторы, счетчики, селекторы и сдвиговые регистры могут выполнять различные функции. Порты ввода-вывода средней степени интеграции можно заменить многие более элементарные компоненты. Универсальный асинхронный и синхронный приемопередатчики и параллельные интерфейсы могут выполнять разнообразные функции управления. Программируемые устройства обеспечивают более высокую гибкость при организации ввода-вывода, но требуют при этом дополнительного программного обеспечения.

Ввод-вывод в системах на основе микропроцессора Intel 8080 реализуется с помощью специальных команд ввода-вывода и сигналов управления. Базовым устройством интерфейса ввода-вывода в этом случае является порт 8212. Ввод-вывод в системах на основе Motorola 6800 адресуется как память. Базовым устройством интерфейса ввода-вывода для этих систем служит адаптер интерфейса периферийных устройств (PIA).

ГЛАВА ДЕВЯТАЯ

ПЕРЕРЫВАНИЯ В МИКРОПРОЦЕССОРНЫХ СИСТЕМАХ

Тема этой главы — организация ввода-вывода информации по сигналам прерывания. Глава начинается со сравнения системы ввода-вывода, управляемой прерываниями, с «регулярной» (программно-управляемой) системой ввода-вывода и с описания их особенностей.

Затем приводятся простые примеры использования прерываний и подробное описание аппарата прерываний в системах на основе микропроцессоров Intel 8080 и Motorola 6800. Завершает главу краткое изложение метода прямого доступа к памяти (ПДП).

9.1. ПРЕИМУЩЕСТВА И НЕДОСТАТКИ ПРЕРЫВАНИЙ

Одна из самых сложных проблем при проектировании подсистем ввода-вывода — согласование во времени работы ЦП и периферийного оборудования. Процессор должен определять, когда у периферийного устройства имеются новые данные для ввода или когда оно готово получить данные из ЦП. Для очень медленных УВВ, таких как переключатели или световые индикаторы, решение этой проблемы не вызывает затруднений, так как быстрой реакции ЦП не требуется. Центральный процессор может обмениваться информацией с такими УВВ в произвольные моменты времени.

По-другому обстоит дело с более быстрыми УВВ. Обычно их скорость не настолько велика, чтобы они могли работать в темпе процессора, но и не настолько мала, чтобы любой способ управления ими оказался приемлемым. В эту категорию попадают такие УВВ, как клавишные пульты, телетайпы, печатающие устройства, кассетные накопители, модемы, системы сбора данных (data acquisition systems) и др. Как было отмечено выше, обмен с такими УВВ может быть синхронным или асинхронным. При асинхронной передаче ЦП должен «быть уверен», что УВВ готово к обмену; при синхронной передаче ЦП инициирует процесс обмена и обеспечивает нужное согласование во времени.

Программно-организованное ожидание сигналов и обеспечение временных задержек приводит к потере процессором времени, увеличению размера и сложности программ. Пусть, например, ЦП ожидает начала передачи данных от УВВ, максимальная скорость которого составляет 10 символов/с. Тогда ЦП будет обнаруживать, что признак «данные готовы» установлен не чаще чем 10 раз в секунду. Зато программа проверки признака очень короткая:

СНК:	ЧИТАТЬ	DPORT
	AND	# MASK
	ПЕРЕХОД, ЕСЛИ НУЛЬ	СНК

Если каждая команда выполняется за 5 мкс, то ЦП проверит признак 6700 раз за 0,1 с. Ясно, что проверка, которая в 99,9% случаев будет безуспешной, малоэффективна.

Даже несколько подобных проверок вряд ли полностью загрузят ЦП. Например, ЦП может проверить каждый из десяти признаков 670 раз за 0,1 с. Кроме того, могут возникнуть и другие проблемы:

что произойдет, если одно УВВ подготовит данные, в то время как ЦП обслуживает другое УВВ?

как быть, если то УВВ, которое процессор проверяет первым, почти всегда активно?

Процесс проверки готовности каждого из УВВ называется *поллингом* (polling). Его можно сравнить (как бы ни была велика и сложна

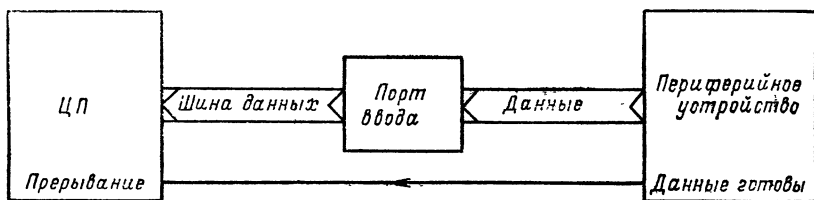


Рис. 9.1. Прерывание от устройства ввода (отдельный порт для сигнала «данные готовы» не нужен. В ответ на сигнал прерывания ЦП передает управление программе, читающей данные из порта ввода)

система) с работой телефонного коммутатора, при которой все линии последовательно опрашиваются, чтобы узнать, не поступил ли сигнал вызова.

При работе с синхронными УВВ необходимо точно соблюдать временные интервалы между пересылками информации. Центральный процессор может сам организовать задержку (см. § 8.1), загружая регистр и декрементируя его заданное число раз. Но при такой организации задержек процессор полностью загружен. Если потребовать, чтобы ЦП выполнял другую работу в этот период, то необходимо, чтобы либо программист определил требуемое время пересылки, либо процессор проверял показания таймера, чтобы узнать, закончился ли заданный интервал времени. Оба варианта приводят к усложнению программ.

Самое обычное решение, альтернативное применению программ полинга и программ реализации временных задержек, — использование прерываний. *Прерыванием*¹ называется подаваемый на вход процессора сигнал, который может непосредственно, аппаратным способом, изменить последовательность операций ЦП. Этот сигнал действует как зуммер, который заставляет процессор приостановить обычные действия и реагировать на сигнал. На рис. 9.1 показано использование прерывания в качестве сигнала «данные готовы». В данном случае нет необходимости в специальных командах проверки признака «готово», так как изменение значения признака непосредственно воздействует на ЦП. На рис. 9.2 запрос на прерывание поступает от таймера. Процессору не нужно самому вести отсчет времени, так как в момент завершения временного интервала возникнет прерывание.

Очевидно, что прерывания полезны при организации ввода-вывода, так как процессору при этом не нужно проверять признаки готовно-

¹Автор использует термин «прерывание» в следующих значениях: а) сигнал «запрос на прерывание», вырабатываемый внешним устройством и передаваемый либо непосредственно на вход процессора, либо на систему сравнения приоритетов, либо на другую аппаратуру, осуществляющую предварительную обработку запроса; б) признак «устройство запросило прерывание», зафиксированный в аппаратуре, т. е. «прерывание, ожидающее обслуживания»; в) сигнал «прерывание», подаваемый на специальный вход (или один из входов) процессора и фиксируемый внутри процессора; этот сигнал вызовет изменение обычной последовательности команд процессора, как только завершится текущий машинный цикл или команда; г) изменение обычной последовательности команд, т. е. реакция на сигнал «прерывание». (Прим. пер.)

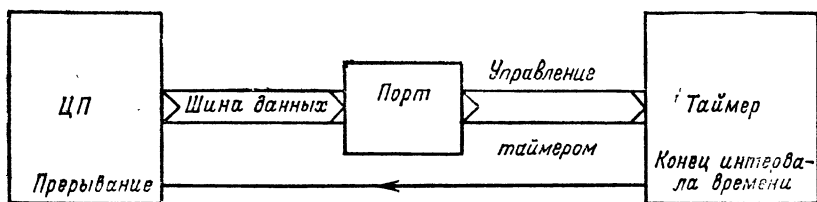


Рис. 9.2. Прерывание от таймера (процессор может определить длительность интервала времени и инициировать начало отсчета времени таймером. Реакция процессора на прерывание зависит от цели, с которой отсчитывается временной интервал)

сти и следить за длительностью временных интервалов. Аппарат прерываний используется также в следующих целях:

1. Сигнал тревоги. Такой сигнал может поступать от первичных преобразователей, переключателей, компараторов. Ситуации, в которых возникает сигнал тревоги, редки, но время реакции на такие сигналы должно быть очень мало. На рис. 9.3 приведен пример системы обеспечения безопасности.

2. Предупреждение о падении напряжения в сети электропитания. Такое прерывание дает системе возможность сохранить данные в блоке памяти, потребляющем мало энергии, или подключиться к резервному источнику питания.

Схема, обнаруживающая падение напряжения в сети электропитания (рис. 9.4), обычно представляет собой RC -цепь, которая реагирует на изменение напряжения достаточно быстро, так что процессор может выполнить еще много команд, прежде чем подача энергии полностью прекратится. Перебои в системе питания — редкие явления, реакция на которые должна быть быстрой. Обработка прерывания при отказе электропитания должна выполняться прежде всего, так как такой сбой приводит к полной неработоспособности системы.

3. Ручное управление с лицевой панели. С помощью прерывания можно разрешить ручное управление системой, что необходимо для профилактики, ремонта, тестирования и отладки системы.

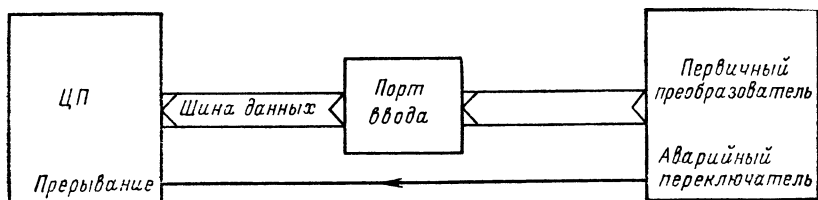


Рис. 9.3. Прерывание тревоги от системы обеспечения безопасности (система обеспечения безопасности — пример применения прерываний по сигналу «тревога». Чтобы убедиться в необходимости использования прерываний в данном случае, достаточно подсчитать число напрасных проверок в предположении, что аварийный переключатель срабатывает в среднем 1 раз за 4 ч, а время реакции не должно превышать долей секунды)

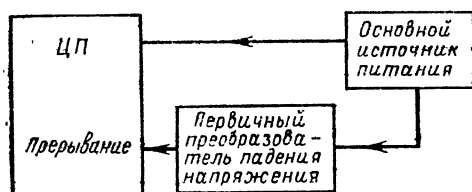


Рис. 9.4. Прерывание, сигнализирующее о падении напряжения источника питания

4. Средство отладки. Прерывания позволяют вносить исправления в программу, осуществлять ее трассировку и приостанавливать исполнение программы в контрольных точках.

5. Индикация аппаратных сбоев.

6. Индикация ошибок при передаче данных.

7. Координация работы аппаратуры в многопроцессорных системах.

8. Управление прямым доступом к памяти.

9. Управление работой операционной системы.

10. Измерение производительности системы.

11. Моделирование часов реального времени. Такие часы посылают сигналы прерывания через равные интервалы времени заданной длительности.

Чтобы оценить преимущества, которые дает система прерываний, следует учесть такую важную характеристику, как отношение времени реакции на прерывание к промежутку времени между событиями. Если время реакции должно быть намного меньше среднего промежутка времени между событиями, то процессор будет вынужден многократно проверять соответствующий признак.

Если, например, процессор должен реагировать на событие в течение 1 мс, а происходят подобные события 1 раз в минуту, то в среднем за 1 мин процессор осуществит 120 000 безуспешных проверок. Необходимость проверять несколько таких признаков может существенно снизить производительность процессора.

Недостатки прерываний

Основные недостатки прерываний связаны со случайным характером их возникновения, и хотя именно по этой причине аппарат прерываний оказывается столь полезным, случайный характер возникновения прерываний приводит к затруднениям при отладке и тестировании программ. Концепция прерываний находится в противоречии с современной тенденцией к созданию более простых и строго определенных программ — с тенденцией, отраженной в «структурном программировании» и методе проектирования программ «сверху вниз».

На практике оказывается, что программы, в которых предусмотрено управление с помощью сигналов прерывания, имеют гораздо менее четкую структуру, чем программы, в которых много явных команд передачи управления. В программах, управляемых по сигналам прерывания, передача управления может возникнуть в любой точке, однако

это не находит отражения в листинге программы. На рис. 9.5 показана структура программы и возможные пути передачи управления (пунктирные линии). Очевидно, что возможность возникновения ошибок программирования здесь очень велика.

Обычный способ обнаружить ошибки в программном обеспечении систем, управляемых прерываниями, — давать системе задачи, решение которых зависит от времени. Если ошибки возникают нерегулярно, то их источником обычно является система обработки прерываний, так как другие части программы при повторном исполнении дают один и тот же результат. Очевидно, что такой подход нельзя назвать научным. Отладка и тестирование любых (за исключением простейших) систем с прерываниями — сложный процесс, дающий ненадежные результаты. Только немногие системы разработки программ позволяют генерировать прерывания случайным образом и анализировать результаты тестирования.

Программы обработки прерываний часто бывает очень сложно написать, так как они должны давать правильные результаты независимо от того, когда возникает прерывание. Может оказаться необходимым, чтобы эти процедуры сохраняли содержимое регистров, значения признаков и адреса памяти, а перед возвратом управления восстанавливали эти значения. Чтобы удовлетворить всем возможным требованиям, потребуется, возможно, значительное число команд, которые, однако, будут выполняться только в редких случаях.

Для систем, использующих прерывания, требуется также дополнительное аппаратное обеспечение, особенно когда прерывания могут вызываться многими источниками. Обычные аппаратные средства включают в себя фиксаторы и триггеры, которые гарантируют правиль-

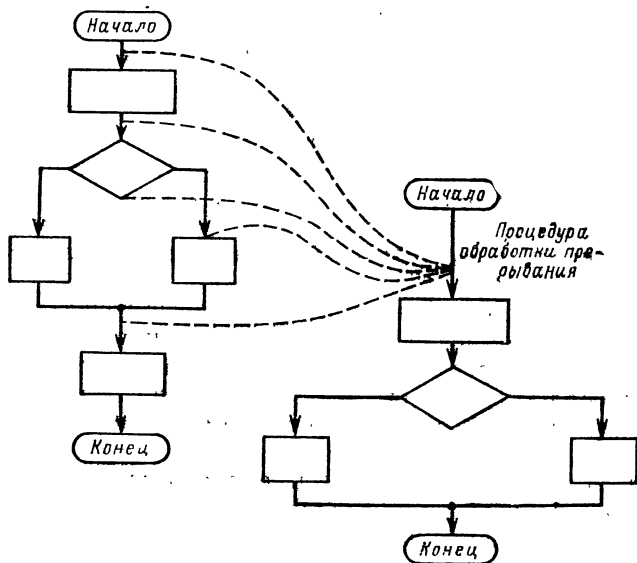


Рис. 9.5. Структура программ, управляемых по сигналам прерывания

ное опознание сигналов прерывания; логические схемы и шифраторы, с помощью которых комбинируются сигналы прерывания от различных источников; порты данных, с помощью которых идентифицируются различные источники прерываний. Количество аппаратуры возрастает с ростом числа источников прерываний, что является следствием необходимости быстрого определения источника.

Менее полезны прерывания при высоких скоростях ввода-вывода данных. Это объясняется тем, что для организации обработки прерываний необходим обмен данными через ЦП, нужна выборка и декодирование соответствующих команд. Кроме того, система прерываний требует дополнительных накладных расходов. Преимущества прерываний при высоких скоростях обмена данными становятся менее ощутимыми, polling — менее эффективным, а временные интервалы оказываются проще генерировать непосредственно с помощью генератора тактовых импульсов процессора. При скоростях обмена данными, превышающих 10 Кбит/с, возникает проблема совмещения функций процессора по организации ввода-вывода с выполнением другой работы. Метод прямого доступа к памяти (при котором программное управление обменом заменяется аппаратным и обеспечивается прямая связь между памятью и модулями ввода-вывода) позволяет резко увеличить пропускную способность ввода-вывода. Этот метод применяется при более высоких скоростях обмена. Системы, в которых применяется ПДП, содержат сложное оборудование. Эти системы кратко рассмотрены в конце главы.

9.2. ОСОБЕННОСТИ СИСТЕМ ПРЕРЫВАНИЙ

Основные проблемы

В любой системе прерываний решаются следующие основные проблемы:

1. В какие моменты проверяются входы запросов на прерывание и какими характеристиками должны обладать сигналы прерываний?
2. Каким образом процессор передает управление подпрограммам обработки прерываний?
3. Каким образом процессор сохраняет текущее состояние ЭВМ («статус машины») и восстанавливает его после завершения обработки прерывания?
4. Как процессор определяет источник прерывания?
5. Каким образом процессор отличает высокоприоритетное прерывание (такое, как прерывание «падение напряжения» или «тревога») от низкоприоритетных прерываний (как, например, прерывание от печатающего устройства, сообщающего, что оно готово к приему новых данных)?
6. Как отключить систему прерываний на время, когда выполняются программы, которые не должны прерываться?

Рассмотрим эти проблемы по порядку

Входы запросов на прерывание. Хотя различные системы прерываний значительно отличаются друг от друга, в большинстве таких систем

Рис. 9.6. Блок-схема цикла команды, включающая проверку сигнала запроса на прерывание (для выполнения команды может потребоваться несколько циклов. Центральный процессор проверяет вход запроса на прерывание только в конце последнего цикла)

процессор проверяет входы запросов на прерывание только в конце цикла команды, так как возобновление цикла команды с середины потребовало бы сохранения большого количества промежуточных результатов. В качестве стандартного метода проверки входов запросов на прерывания применяется метод, который представлен блок-схемой на рис. 9.6. Отметим, что при использовании данного метода необходимо фиксировать сигнал запроса на прерывание, так как полный цикл команды может быть очень продолжительным. Чтобы правильно распознать этот сигнал, необходимо синхронизировать его с генератором тактовых импульсов процессора. Для этого используется специальный триггер (рис. 9.7).

Число входов запросов на прерывание различно для разных типов процессоров. В табл. 9.1 приведено число входов для некоторых распространенных микропроцессоров. На каждый вход могут, разумеется, поступать сигналы запросов на прерывание от более чем одного источника, так как такие сигналы могут комбинироваться с помощью схемы ИЛИ (рис. 9.8). Если действующее значение сигнала прерывания низкое, то такой сигнал можно получить без использования схем ИЛИ, монтажно комбинируя выходы свободных коллекторов: прерывание будет иметь высокий уровень (т. е. недействующее значение) только тогда, когда все сигналы свободных коллекторов имеют высокий уровень.

Реакция на запросы на прерывание Методы, с помощью которых определяется реакция процессора на запросы на прерывание, также

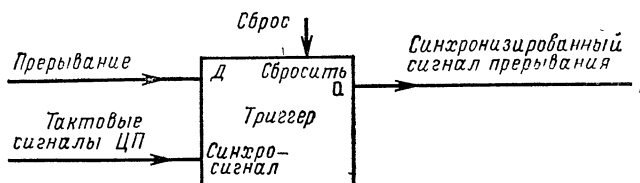
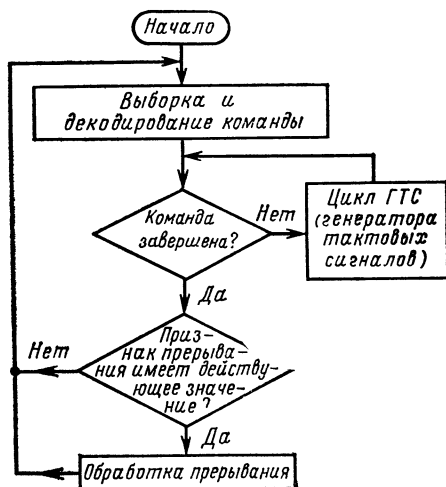


Рис. 9.7. Синхронизация сигнала запроса на прерывание с генератором тактовых импульсов процессора (использование D-триггера гарантирует, что сигнал запроса на прерывание появится на входе процессора в правильно выбранный момент цикла тактового сигнала)

Таблица 9.1. Число входов запросов на прерывание различных типов МП

Микропроцессор	Число входов запросов на прерывание	Микропроцессор	Число входов запросов на прерывание
Intel 4040	1	RCA CDP1802	1
Intel 8080	1	Signetics 2650	1
MOS Technology 6502	2	National PACE	6
Motorola 6800	2	Texas Instruments 9900	1
Rockwell PPS-8	3	Toshiba TLCS-12	8

очень разнообразны. Приведем наиболее распространены из них:

1. Выполнить команду ПЕРЕХОД К ПОДПРОГРАММЕ (CALL) или РЕСТАРТ (TRAP)¹ с заданным адресом. Этот способ используется в процессоре Intel 4040.

2. Выбрать из определенного регистра или определенной ячейки памяти новое значение счетчика команд. Такой способ используется в процессорах RCA CDP1802, National SC/MP и Motorola 6800.

3. Выполнить команду ПЕРЕХОД К ПОДПРОГРАММЕ с адресом, который определяется внешним устройством. Такой способ используется в Signetics 2650.

4. Используя выходной сигнал «подтверждение запроса на прерывание», предоставить шину данных для помещения на нее команды внешним устройством; как показано на рис. 9.9

Основным критерием выбора того или иного способа реакции на запрос на прерывание является соотношение между программными и аппаратными средствами. Применение первого и второго методов требует незначительного количества внешней аппаратуры, но не дает никаких средств для прямой идентификации источника прерывания. Третий и четвертый методы более гибкие и позволяют идентифицировать источник, но для их реализации требуется больше внешнего оборудования.

При применении первого и второго методов процессор может передать управление подпрограмме обработки прерывания и затем вернуть управление исходной программе так же, как он это делает в случае обычных подпрограмм. Любой из методов работы с подпрограммами, описанных в гл. 3, можно применить и здесь. Команда ПЕРЕХОД С

¹В технических описаниях команду TRAP называют ПОВТОРНЫЙ ВЫЗОВ (Прим. пер.)

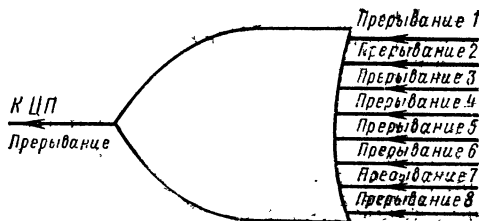


Рис. 9.8. Объединение входов прерываний с помощью схемы ИЛИ [сигнал, подаваемый на вход ЦП, принимает действующее значение (высокое), если хотя бы один из сигналов прерывания имеет высокий потенциал. Однако ЦП не может внутренним образом отличить различные источники прерываний друг от друга]

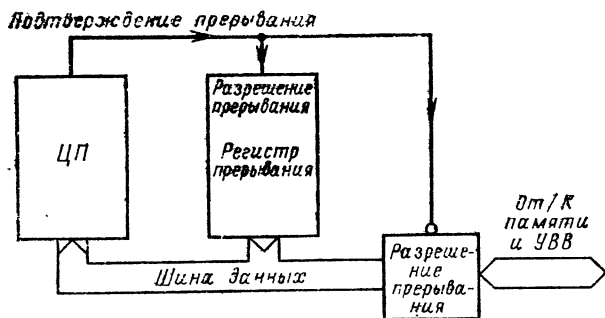


Рис. 9.9. Применение сигнала «подтверждение прерывания» (по сигналу «подтверждение прерывания» из внешнего регистра прерываний на шину данных помещается команда, а нормальный доступ к памяти подавляется)

СОХРАНЕНИЕМ АДРЕСА ВОЗВРАТА (JUMP AND MARK PLACE) передает управление подпрограмме обработки прерываний, находящейся в ОЗУ. Команда ПЕРЕХОД С ВОЗВРАТОМ (JUMP AND LINK) извлекает адрес подпрограммы обработки прерываний из регистра. Последняя команда — более быстрая, однако при ее применении резервируется один из регистров и его нельзя использовать обычным образом. Кроме того, без применения дополнительных команд невозможна реализация многоуровневых прерываний. Наиболее гибкой является команда CALL — ПЕРЕХОД К ПОДПРОГРАММЕ, помещающая адрес возврата в стек. Однако для выполнения этой команды необходимо либо внешнее ОЗУ, либо расположенный на том же кристалле стек; в обоих случаях требуется большое внимание при одновременном использовании ОЗУ (или встроенного стека) как для обычных вызовов подпрограмм, так и для обслуживания прерываний.

При применении третьего и четвертого методов загрузка адреса перехода или команды внешней аппаратурой может вызвать множество проблем, связанных с синхронизацией и управлением. Разумеется, внешнее оборудование не должно препятствовать выполнению обычного цикла памяти; вместе с тем модули памяти не должны мешать внешней аппаратуре, когда последняя получает управление шиной. Следует обратить внимание на то, что по сигналу «подтверждение запроса на прерывание» (см. рис. 9.9) адрес перехода или команда помещается внешним устройством на шину данных, а для обычных данных (из памяти) шина данных по этому сигналу оказывается недоступной. Очевидно, что в данном случае гораздо проще использовать адреса или команды длиной в одно слово, чем адреса или команды, состоящие из нескольких слов. Все или только некоторые разряды помещаемого на шину слова могут устанавливаться ТТЛ- или МОП-шифраторами. Однако, если пользоваться только адресами и командами длиной в одно слово, то число различных входных комбинаций будет весьма ограниченным; кроме того, адрес длиной в одно слово может противоречить принципу страничной организации памяти. Для устранения указанных недостатков следует использовать такое сочетание регистров и цепей задержки, с помощью которого можно помещать на шину данных

команды длиной в несколько слов. Требуемое для этого оборудование может входить в состав специальных контроллеров на БИС.

Сохранение и восстановление состояния процессора. Большинство процессоров автоматически сохраняет некоторую информацию о состоянии, предшествовавшем прерыванию, — это является составной частью стандартной реакции процессора на прерывание. Все процессоры сохраняют значение счетчика команд, но некоторые ЦП сохраняют и другую информацию. Например, Motorola 6800 сохраняет в стеке содержимое всех своих регистров. Этот способ, впрочем, удобен только тогда, когда регистры содержат полезную информацию; в противном случае напрасно расходуется память и время процессора. С помощью специальных команд, таких как ПРОГРАММНОЕ ПРЕРЫВАНИЕ (SOFTWARE INTERRUPT), РЕСТАРТ (TRAP) или ОЖИДАНИЕ ПРЕРЫВАНИЯ (WAIT FOR INTERRUPT), можно сохранить содержимое регистров программным способом; команды ВОЗВРАТ ИЗ ПРЕРЫВАНИЯ (RETURN FROM INTERRUPT, RETURN FROM TRAP) восстанавливают содержимое регистров в конце подпрограммы обработки прерывания.

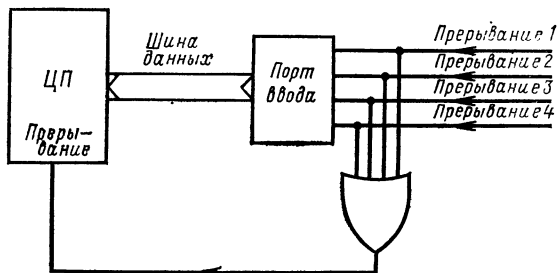
В большинстве процессоров требуется несколько команд, чтобы сохранить старое состояние процессора. Обычно применяется один из следующих трех методов:

1. Простое запоминание содержимого регистров и значений признаков в памяти. При этом методе подпрограмму обработки прерываний нельзя прервать, если нет другой доступной области памяти для хранения содержимого регистров при переходе к обслуживанию прерывания следующего уровня.
2. Запоминание содержимого регистров в стеке, расположенном в памяти. Этот метод прост, так как указатель стека содержит адрес, по которому хранится требуемая информация; допустимы также и многоуровневые прерывания. Основное неудобство состоит в том, что возможно переполнение стека. Адрес возврата также хранится в стеке; поэтому в случае необычного выхода из подпрограмм потребуется выполнение значительного числа операций со стеком.
3. Переход к другому набору регистров. В некоторых процессорах, таких как Zilog Z-80 и Signetics 2650, имеется двойной набор регистров. Подпрограмма обработки прерывания может просто использовать другой набор. Этот способ быстрее, чем оба описанных выше, но его применение означает, что некоторые регистры процессора не всегда доступны. Метод не допускает многоуровневых прерываний, так как имеются только два набора регистров. Процессор Texas Instruments 9900 в качестве поля регистров использует назначаемую область памяти; поэтому при прерывании достаточно изменить содержимое регистра-указателя.

Состояние ЭВМ, естественно, должно быть восстановлено до окончания процедуры обработки прерывания. При использовании стека необходимо восстановление информации в порядке, обратном тому, в котором информация сохранялась. При использовании других методов применяется восстановление в прямом порядке.

Число регистров, которые следует сохранить, зависит от набора ре-

Рис. 9.10. Система прерываний с полингом (чтобы определить источник прерывания, программа обработки прерывания должна опросить порт ввода и найти, какой разряд имеет действующее значение)



гистров, используемых основной программой и подпрограммой обработки прерывания. Если основная программа просто ожидает возникновения прерывания, то нет необходимости сохранять и восстанавливать какие-либо регистры. Программа обработки прерывания не обязана сохранять регистры, которыми она не пользуется. Для таких процессоров, как Motorola 6800, число регистров которых невелико, подпрограммы обработки прерываний должны сохранять и восстанавливать все регистры. Использование этой процедуры не требует значительных затрат времени и не вызывает трудностей при программировании. При составлении подпрограмм обработки прерываний для процессоров, подобных Intel 8080 или Fairchild F-8, имеющих много регистров, программисту следует тщательно выбирать регистры, которые следует сохранить. В противном случае время реакции на прерывание может оказаться слишком большим.

Определение источника прерывания. Если в системе имеется несколько возможных источников прерываний, то процессор должен идентифицировать источник. Процессор с несколькими входами запросов на прерывание может по-разному реагировать на прерывания, поступающие с различных входов, например, передавать управление различным регистрам или по различным адресам. Однако в том случае, когда число источников превышает число входов, определение источника прерывания становится затруднительным.

Существует два наиболее распространенных метода идентификации источника прерывания: полинг и векторное прерывание. Полинг подобен обычной проверке признаков «данные готовы» или «УВВ готово»: ЦП проверяет каждый из признаков прерываний, пока не обнаружит возбужденный (рис. 9.10). *Векторное прерывание* означает, что каждый источник прерываний передает данные (т. е. *вектор*), которые процессор использует для идентификации. Векторное прерывание выполняется быстрее и требует меньше программных средств, тогда как для полинга нужно меньше аппаратных средств.

Преимущество полинга в системах, использующих прерывания, перед полингом в системах без прерываний состоит в том, что при возникновении прерывания процессор знает, что по крайней мере один вход запроса на прерывание имеет действующее значение. Для реализации полинга в системах с прерываниями требуются лишь адресуемые триггеры — по одному на каждый признак прерывания.

Очевидно, что систему прерываний с полингом разумно использовать только тогда, когда число источников прерывания невелико. В про-

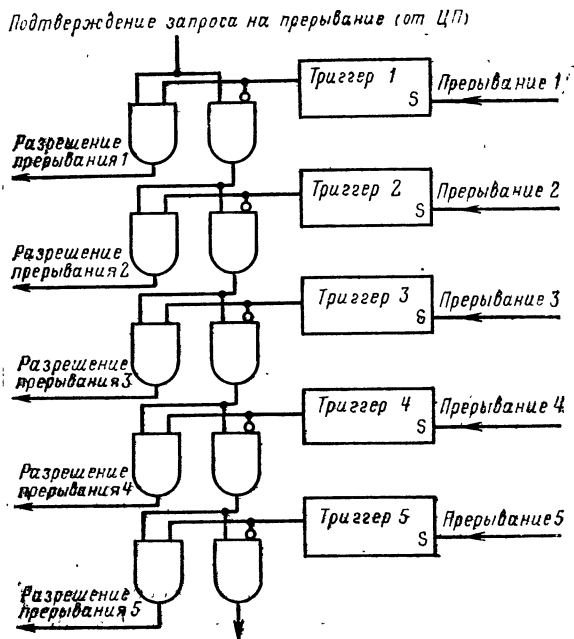


Рис. 9.11. Система прерываний, организованная по принципу «дейзи-цепочки» (по сигналам запроса на прерывание устанавливаются RS-триггеры. На рисунке не показаны цепи, формирующие сигнал запроса на прерывание, идущий к ЦП, а также цепи, сбрасывающие триггеры)

тивном случае время, которое процессор затрачивает на идентификацию источника, становится существенным. Искусное программное обеспечение, например такое, в котором при возникновении прерывания первыми проверяются наиболее вероятные источники прерываний или одновременно проверяются группы источников, — может несколько сократить среднее время идентификации источника прерывания. Циклическое изменение (сдвиг) порядка опроса источников усредняет время ожидания обслуживания для всех источников, а также предотвращает блокировку одного источника другим.

Для существенного усовершенствования метода полинга необходима дополнительная аппаратура. Один из распространенных методов — метод «дейзи-цепочки» (daisy chain), при котором сигнал подтверждения запроса на прерывание распространяется от одного источника к другому¹, пока не будет блокирован фактическим источником. На рис. 9.11 изображено применение метода «дейзи-цепочки» в системе прерываний с одним входом запроса на прерывание и пятью источниками прерываний. Сигналы прерываний подаются на вход процессора через схему ИЛИ. Сигнал «подтверждение запроса на прерывание» схемно комбинируется с признаком запроса на прерывание, поступаю-

¹Под источником здесь понимается, естественно, не само УВВ, а признак запроса на прерывание от данного источника. (Прим. пер.)

щим с соответствующего триггера, прежде чем он будет передан дальше. Если один из триггеров установлен, то выход этого триггера предотвратит распространение сигнала «подтверждение запроса на прерывание» дальше по цепочке. Схемы, изображенные на рис. 9.11 слева, вырабатывают сигналы разрешения прерывания, которые принимают действующее значение тогда и только тогда, когда сигнал «подтверждение запроса на прерывание» доходит до этого звена цепочки и установлен соответствующий триггер. С помощью сигналов разрешения прерывания можно идентифицировать источник либо непосредственно, либо с помощью кодирующего устройства.

При использовании метода «дейзи-цепочки» полинг не нужен; для реализации метода необходимы только незначительные дополнительные аппаратные средства. Добавление и исключение источников прерывания, включенных в «дейзи-цепочку», производится просто. Для сигналов подтверждения и разрешения запросов на прерывание могут понадобиться дополнительные порты. Недостаток метода «дейзи-цепочки» состоит в том, что программа не может изменить приоритеты источников прерываний, и поэтому источники, имеющие более высокий приоритет, могут заблокировать источники с более низким приоритетом. Процессор Fairchild F-8, входы запросов на прерывание которого соединены с каждым кристаллом памяти, имеет встроенную «дейзи-цепочку», для которой не нужны внешние схемы. На рис. 9.12 показана типичная конфигурация для трех источников прерываний. Признак \overline{ICB} (INTERRUPT CONTROL BIT) играет роль сигнала «подтверждение запроса на прерывание» для «дейзи-цепочки». Каждое ПЗУ (модуль хранения программ 3851—3851 Program Storage Unit или PSU) имеет линию PRIIN (ввод приоритета — PRIORITY INPUT) и линию PRI OUT (вывод приоритета — PRIORITY OUTPUT), с помощью которых

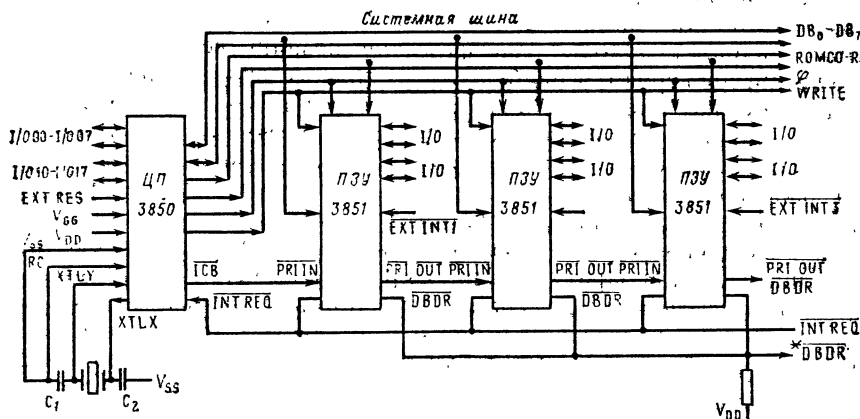


Рис. 9.12. Приоритетная система прерываний для процессора Fairchild F-8, организованная по принципу «дейзи-цепочки» (знаком * отмечен сигнал, который не используется, если на шине данных не применяются внешние буфера. В системе имеются три источника прерываний с низким действующим значением сигналов запроса на прерывание ($\overline{EXT INT1}$, $\overline{EXT INT2}$ и $\overline{EXT INT3}$), которые могут формировать сигнал запроса на прерывание ($\overline{INT REQ}$), идущий к ЦП F-8 (3850)

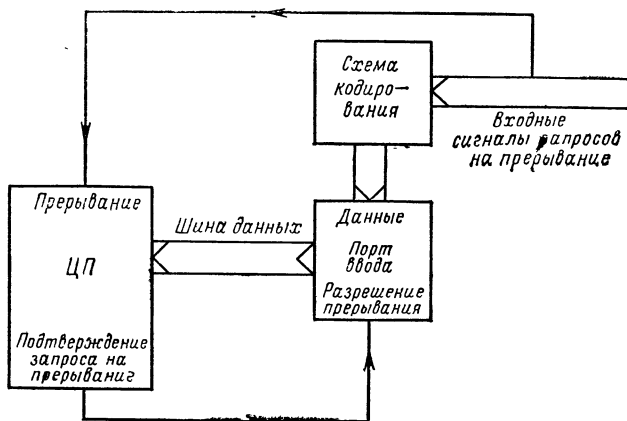


Рис. 9.13. Система прерываний по вектору [по сигналу «подтверждение запроса на прерывание» на шину данных помещается вектор, вырабатываемый схемой кодирования. Схема кодирования может состоять из ТТЛ или МОП-шифраторов, простых логических схем и ППЗУ (PROM)]

формируются связи в «дейзи-цепочке». Действующее значение сигнала запроса на прерывание на входе блокирует дальнейшее распространение сигнала «подтверждение запроса на прерывание» по «дейзи-цепочке» и передает управление по адресу, находящемуся в ПЗУ.

«Дейзи-цепочка» — разновидность векторного прерывания, так как каждый источник прерывания идентифицирует себя сам. В системах с явной реализацией векторного прерывания вырабатывается (с помощью шифраторов и сигналов управления) вектор, который помещается на шину данных. На рис. 9.13 приведена функциональная схема типичной векторной системы прерываний.

Число отличных друг от друга векторов зависит от сложности применяемой аппаратуры. Для реализации системы прерываний с большим числом векторов необходимы сложные схемы, такие как последовательности кодирующих устройств или ППЗУ (PROM) большого объема. В больших системах одновременно может использоваться и векторное прерывание и полинг. С помощью векторов производится разбиение источников прерывания на малые группы, а затем с помощью полинга идентифицируется конкретный источник из группы. Применение такого комбинированного подхода часто оказывается значительно дешевле, чем чисто векторный метод, при этом дополнительный расход времени невелик.

Термин «векторное прерывание» указывает на использование идентифицирующего кода для выполнения перехода на конкретную подпрограмму обработки прерывания¹. По первому способу процессор может

¹Автор употребляет термин «вектор» в нескольких значениях: а) информация, с помощью которой можно идентифицировать источник прерывания; б) способ позволяющий установить соответствие между кодом прерывания (т. е. вектором в первом смысле) и адресом точки входа в программу обработки прерывания; в) внутрипрограммная или внутрипроцессорная информация, обеспечивающая переход к подпрограмме. (Прим. пер.)

сам автоматически загрузить нужный код операции в регистр команд. По второму способу код операции должен вырабатываться внешней аппаратурой наряду с вектором. Имеется еще один способ: программное формирование адреса (по идентифицирующему коду) с последующим переходом по этому адресу. Команда перехода может формироваться с помощью косвенной адресации или адресации с использованием индексного регистра. Такой способ медленнее аппаратных способов; кроме того, в результате его применения часто получаются программы, работу которых трудно проследить. Для процессора с косвенной адресацией через регистр процедура формирования команды перехода такова: а) загрузить в регистр 1 идентифицирующий код и б) выполнить команду `JUMP@R1`, т. е. косвенный переход по адресу в регистре 1. Для индексного способа адресации процедура формирования команды перехода такова: а) загрузить регистр индекса идентифицирующим кодом и б) выполнить команду `JUMP 0, X`, т. е. переход по адресу, содержащемуся в регистре индекса. Некоторые процессоры должны преобразовывать код идентификации, чтобы сформировать адрес перехода, так как этот код может быть очень коротким. С помощью любой из указанных процедур легко получить по вектору начальный адрес подпрограммы обработки прерывания.

Приоритеты. Приоритетные методы обслуживания прерываний связаны с решением следующих основных вопросов:

1) Какой из нескольких одновременно присутствующих в системе запросов на прерывание процессор будет обслуживать первым?

2) Какими прерываниями может быть прервано выполнение процедур обработки других прерываний?

3) Как обрабатываются прерванные программы?

4) Как добиться того, чтобы прерывания, игнорируемые из-за их низкого приоритета, в конце концов были обслужены?

Если процессор имеет несколько входов запросов на прерывание (как, например, National RACE), то каждому входу можно приписать свой приоритет. На обслуживание будет принято то прерывание, которое в данный момент времени имеет самый высокий приоритет. На другие прерывания процессор не реагирует.

Если процессор имеет единственный вход запросов на прерывания, то для назначения приоритетов можно воспользоваться внешним шифратором приоритета (priority encoder). ТТЛ-шифраторы выдают вектор прерывания и блокируют одновременно возникшие прерывания более низкого приоритета. С помощью МОП-шифраторов или ППЗУ также можно обеспечить автоматический учет приоритетов на аппаратном уровне.

По приоритету прерывания можно установить, разрешено оно или нет¹. При наличии нескольких входов запросов на прерывание про-

¹Автор использует термин «приоритет» в следующих значениях: а) элемент набора $\{n_1, n_2, \dots, n_k\}$ целых чисел ($n_1 < n_2 < \dots < n_k$). Отношение порядка на элементах набора определяет приоритет. В этом смысле можно говорить, что устройству приписан приоритет n_1 («низший»), ..., n_k («высший»); б) информация, определяющая или позволяющая в принципе определить приоритет прерывания (согласно п. «а»): вектор прерывания, вход, на который подается сигнал

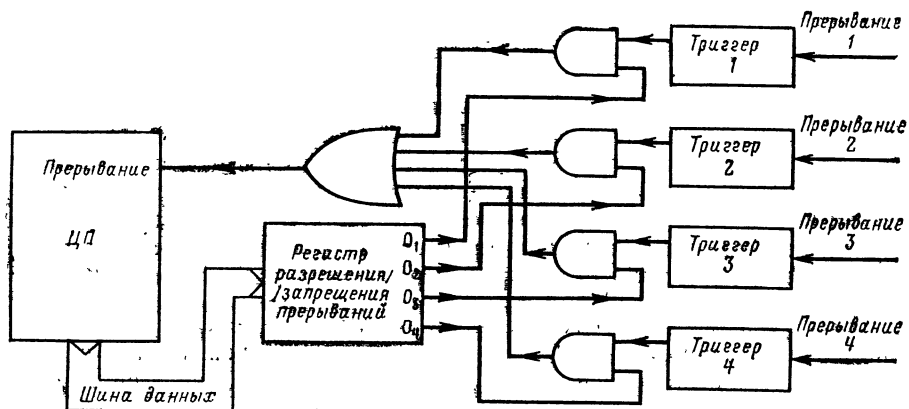


Рис. 9.14. Внешний регистр, используемый для разрешения и запрещения прерываний с определенным приоритетом (входной сигнал запроса на прерывание только тогда вызовет прерывание, когда установлен соответствующий разряд регистра. Содержимое регистра определяется при выполнении операции вывода ЦП, например: `LOAD # 1111B` `WRITE EREG;` разрешить все прерывания)

процессор может просто открывать или закрывать буфера, связанные с каждым из входов. Это наиболее гибкий метод, так как он позволяет разрешать и запрещать прерывания в отдельности для прерываний с различным приоритетом. Обычно разряды, соответствующие разрешенным прерываниям, сохраняются в регистре, поэтому программист может легко определить эти прерывания. Некоторые разряды могут устанавливаться автоматически внутренними схемами в момент поступления прерывания с определенным приоритетом.

Для процессоров с единственным входом запросов на прерывание требуется внешнее оборудование для разрешения или запрещения прерываний с различными приоритетами. На рис. 9.14 показан один из вариантов использования для этой цели внешнего регистра. В данном случае ЦП реагирует только на прерывания, для которых разряды регистра, соответствующие их приоритетам, установлены в единицу. При инициализации системы программа должна установить все разряды регистра в единицу. Содержимое этого регистра следует также сохранить в памяти, так как регистр доступен только для записи (чтобы иметь возможность восстановить его содержимое при завершении процедуры обработки прерывания).

Другой вариант принятия решения о разрешении или запрещении прерывания состоит в том, чтобы исключить все прерывания с приоритетом, не превосходящим приоритет прерывания, принятого на обработку. Исключение прерывания с приоритетом, равным приоритету прерывания, принятого на обработку, предотвращает приостановку

запроса на прерывание, и т. д. Приоритет в этом смысле можно часто просто отождествить со всей информацией, сопровождающей запрос на прерывание и поступающей на схемы системы приоритетов; в способ, позволяющий установить соответствие между приоритетом п. «б» и приоритетом п. «а». (Прим. пер.)

программы обработки прерывания при поступлении повторного запроса на прерывание от того же самого источника прерывания. Схема сравнения (компаратор) чисел (7485) может определить, является ли приоритет нового прерывания более высоким по сравнению с приоритетом обрабатываемого (текущего) прерывания. У внешнего регистра состояния должен также быть разряд «игнорировать приоритеты», чтобы можно было обойти процедуру сравнения и поставить на обслуживание прерывание самого низкого уровня (ведь если текущий приоритет равен нулю, то разрешены прерывания с приоритетами, большими, но не равными нулю). На рис. 9.15 изображена система приоритетного прерывания с компаратором.

Полинг и метод «дейзи-цепочки» автоматически обеспечивают распределение приоритетов. При полинге приоритет прерывания тем выше, чем раньше опрашивается вход запроса на прерывание. При использовании «дейзи-цепочки» более высоким приоритетом обладают первые звенья цепочки. С помощью сигналов разрешения прерывания можно разрешить (или запретить) все прерывания, возникающие за любым конкретным звеном цепочки, как показано на рис. 9.16.

В некоторых системах значительные трудности доставляют прерывания, которые игнорируются из-за своего низкого приоритета. Разумеется, сигналы прерывания должны запоминаться в триггерах запросов на прерывание. Однако время обслуживания прерывания с более высоким приоритетом может оказаться очень продолжительным, и некоторые прерывания могут так и остаться необслуженными. В таких случаях оказываются полезными следующие методы: автоматическое увеличение приоритета прерывания каждый раз, когда оно остается

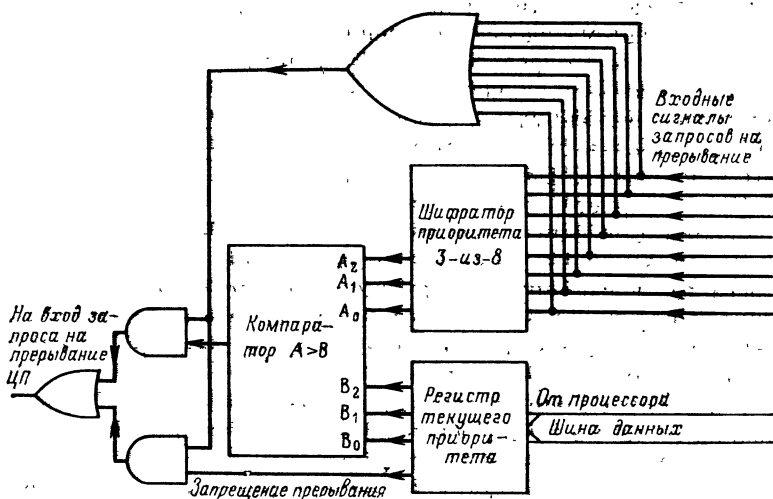


Рис. 9.15. Система приоритетного прерывания со схемой сравнения (компаратором) [установка одного из разрядов регистра отключает компаратор. Если этот разряд («запрещение работы компаратора») установлен, то прерывание работы процессора произойдет при поступлении сигнала с любого входа запроса на прерывание]

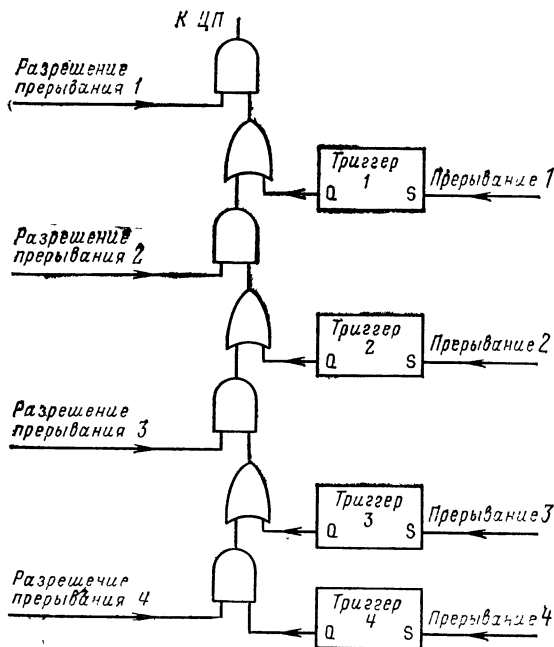


Рис. 9.16. Система приоритетного прерывания, организованная по принципу «цепочки» и снабженная дополнительными средствами разрешения и запрещения прерываний [каждый из сигналов разрешения влияет на статус (разрешено/запрещено) всех прерываний, поступающих по линиям, находящимся дальше от процессора]

необслуженным; отключение всей системы приоритетов или ее части в определенные моменты времени; просмотр низкоприоритетных прерываний перед принятием на обслуживание высокоприоритетного. К счастью, лишь немногие системы прерывания микропроцессоров являются

настолько сложными, что возникает необходимость в применении какого-нибудь из этих методов.

Разрешение и запрещение прерываний. В этой главе уже рассматривались некоторые вопросы, связанные с разрешением и запрещением прерываний. В определенных ситуациях почти все процессоры автоматически запрещают прерывания. К таким ситуациям относятся:

1. Сброс. Отключение системы прерываний по сигналу сброса позволяет программе загрузить внутренние или внешние регистры и установить значения переменных, которые могут потребоваться для обработки прерываний или их распознавания.

2. Ситуация, возникающая непосредственно после приема прерывания на обслуживание. Запрещение прерываний в этот момент позволяет выяснить источник прерывания, запомнить содержимое регистров и произвести действия, связанные с приоритетами, без помех со стороны новых сигналов прерывания. Запрещение прерываний также предотвращает приостановку выполнения программы обработки прерывания запросом на прерывание от того же самого источника. Следует обратить внимание на то, что прерывания запрещены до тех пор, пока они не будут явно разрешены, а также что необходимо возобновить разрешение работы системы прерываний до окончания обработки прерывания; это возобновление — часть действий по восстановлению состояния процессора, предшествовавшего прерыванию.

У некоторых микропроцессоров, в том числе у Motorola 6800 и Intel 8085, имеется *немаскируемое прерывание*, которое нельзя запретить никакой командой процессора. Такое прерывание возникает

при падении напряжения питания, так как обработка этого прерывания, очевидно, должна предшествовать любым другим действиям. Немаскируемое прерывание может быть запрещено внешней аппаратурой.

9.3. ПРОСТЫЕ СИСТЕМЫ ПРЕРЫВАНИЯ

В простейших системах, основанных на механизме прерываний, не выполняется никаких действий, кроме простого ожидания прерывания, указывающего на наличие входных данных. Такие данные могут поступать с клавиатуры, телетайпа, модема или преобразователя. Данные в таких системах принимаются так редко, что ЦП всегда успевает завершить обработку очередной порции данных до принятия новой. Для таких простых систем использование прерываний — скорее удобство, чем необходимость. Сигнал «данные готовы» от УВВ подается непосредственно на вход запроса на прерывание процессора; при этом не требуются ни дополнительный порт, ни программные средства для опроса порта.

На рис. 9.17 показана конфигурация аппаратуры для такого простейшего случая. Сигнал по линии «данные готовы» устанавливает D-триггер и вызывает прерывание. D-триггер фиксирует сигнал и преобразует длительный сигнал (продолжающийся несколько тактов генератора тактовых сигналов) в короткий импульс. Центральный процессор реагирует на прерывание путем выполнения процедуры обработки прерывания. Схема, которая открывает порт данных на время операции ввода, также сбрасывает триггер. Чтобы вызвать новое прерывание, потребуется новый сигнал по линии «данные готовы». Чтобы указать процессору нужную подпрограмму обработки прерывания, может потребоваться дополнительное оборудование.

Для подобных систем главная программа не содержит почти никаких других команд, кроме команды ОСТАНОВ (HALT) или ОЖИДАНИЕ ПРЕРЫВАНИЯ (WAIT FOR INTERRUPT). Если у процессора нет ни той, ни другой команды, то можно получить тот же результат

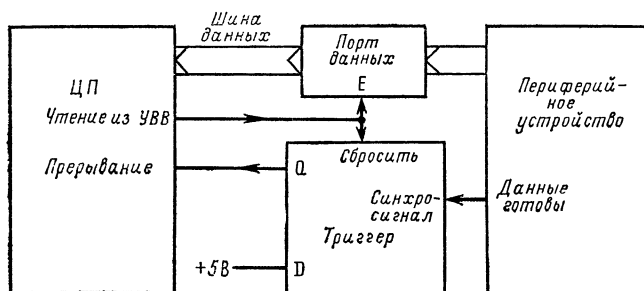


Рис. 9.17. Организация прерывания при наличии одного устройства ввода (по сигналу «данные готовы» устанавливается триггер и возникает прерывание ЦП. Сигнал «разрешение прерывания», подаваемый порту данных, сбрасывает также триггер запроса на прерывание)

с помощью команды безусловного перехода на эту команду:

HERE JUMP HERE

Вход в главную программу следует располагать по адресу, на кото-
рый передается управление по сигналу сброса, тогда система будет рабо-
тать правильно с момента включения питания. Некоторые дополни-
тельные команды могут потребоваться для подготовки передачи управ-
ления на подпрограмму обработки прерывания, в том числе команды на-
чальной установки указателя стека, загрузки регистра адресом под-

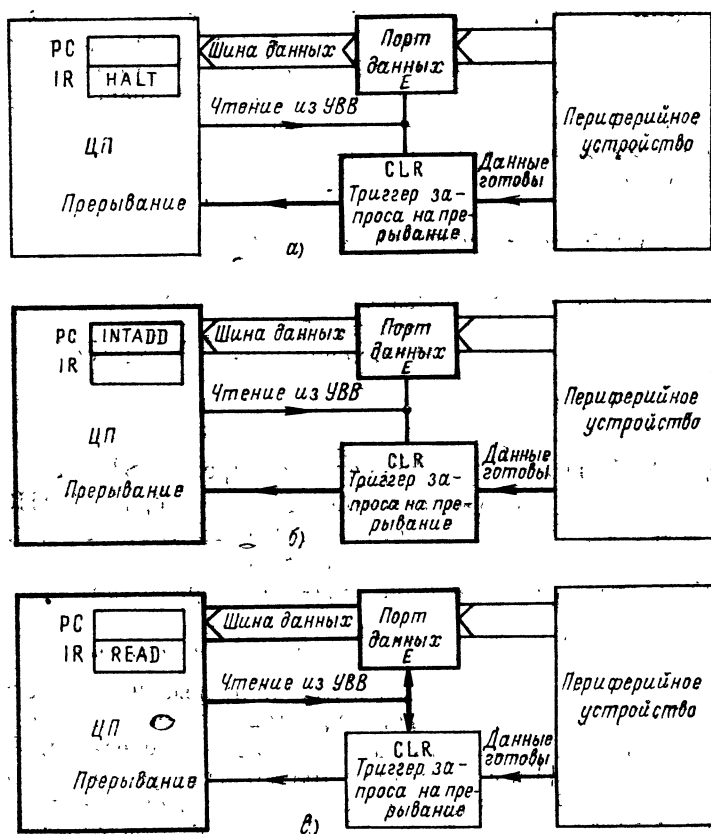


Рис. 9.18. Процесс обработки прерывания в простом случае:

а — УВВ¹ посылает сигнал «данные готовы» (а также «данные»); сигнал «данные готовы» фиксируется. Он вызывает прерывание процессора, выполнявшего команду HALT (ОСТАНОВ); б — процессор опознает прерывание и передает управление подпрограмме обработки прерывания, расположенной в памяти по адресу INTADD. Для принудительной загрузки адреса INTADD в счетчик команд могут потребоваться аппаратные средства; в — процессор читает данные из порта и сбрасывает триггер запроса на прерывание. Триггер запроса на прерывание сбрасывается при чтении данных. Однако разрешение прерываний не возобновляется еще некоторое время; поэтому вновь поступивший по линии «данные готовы» сигнал будет зафиксирован, но не принят на обработку

программы обработки прерывания, назначения регистров, начальной установки регистра приоритетов, и для других действий по управлению состоянием процессора. Главная программа должна также разрешить работу системы прерываний, так как «сброс» запрещает ее. В системе, показанной на рис. 9.17, не требуется, чтобы подпрограмма обработки прерывания возвращала управление главной программе. Фактически процедура обработки прерывания может заканчиваться переходом по тому же адресу, по которому передается управление при сбросе; в результате предшествовавшее прерыванию состояние будет восстановлено. Разумеется, не нужно сохранять ни значения флагов процессора, ни содержимое регистров и ячеек памяти. Нет необходимости и в программных или аппаратных средствах для определения источника прерывания, так как имеется только один такой источник. При сбросе должен сбрасываться и триггер запроса на прерывание, чтобы система начала работу в исходном состоянии.

Вся программа выглядит следующим образом:

ORG RSTADD

Установить счетчик адреса.

Настроить процессор на передачу управления процедуре обработки прерывания

ENABLE INTERRUPTS Разрешить прерывания

HALT Останов

ORG INTADD Установить счетчик адреса

:Выполнить программу

JUMP RSTADD Перейти по адресу RSTADD

С помощью специальной псевдокоманды ассемблера УСТАНОВИТЬ СЧЕТЧИК АДРЕСА (ORG) задается нужный адрес начала подпрограммы обработки прерывания. На рис. 9.18 показан процесс обработки прерывания, включающий в себя сброс триггера запроса на прерывание.

Прерывание при наличии одного устройства вывода

Несколько иной подход нужен в случае прерывания от устройства вывода, указывающего, что устройство готово принять данные; так как процессор должен подготовить данные перед их передачей. Рассмотрим простой случай, когда имеется только одно устройство вывода, которое вырабатывает сигнал прерывания, когда оно готово принять данные. Обмен информацией осуществляется достаточно редко, поэтому не нужны особые процедуры, гарантирующие наличие данных в момент их запроса устройством. И здесь система прерываний — удобство, а не необходимость: прерывание лишь обеспечивает непосредственную подачу сигнала «данные готовы» на ЦП.

На рис. 9.19 показана конфигурация оборудования для данного случая. Отличия от случая прерывания от устройства ввода (рис. 9.17) следующие: данные передаются в другом направлении и другой управляющий сигнал процессора активирует порт и сбрасывает триггер запроса на прерывание.

В системе, изображенной на рис. 9.19, главная программа начинается с адреса, на который передается управление при сбросе. Программа настраивает процессор на передачу управления процедуре обработки прерывания, готовит данные для вывода, разрешает прерывания и ожидает сигнала «УВВ готово». И здесь нет необходимости возвращать управление главной программе, сохранять состояние процессора и определять источник прерывания.

Программа на языке ассемблера имеет следующую структуру:

```

ORG      RSTADD      Установить счетчик адреса
.
.
.Настроить процессор на передачу управления процедуре обработки
.прерывания
.Подготовить данные для вывода
.
ENABLE   INTERRUPTS   Разрешить прерывания
HALT     Останов
ORG      INTADD        Установить счетчик адреса
.
.Послать данные устройству вывода
.
.
JUMP     RSTADD        Перейти по адресу RSTADD

```

На рис. 9.20 показана блок-схема данной программы. Отметим, что признак прерывания проверяется аппаратно, а не программно.

Другое возможное решение — разрешить прерывания сразу после настройки системы прерываний. В этом случае подпрограмма обработки прерывания должна определить, готовы ли данные. Программа может использовать два признака: DRDY (DATA READY — «данные готовы») и PRDY (PERIPHERAL READY — «УВВ готово»). Эти признаки принимают значение 1, когда готовы соответственно данные или

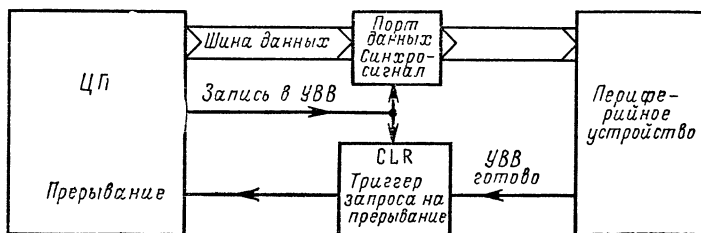


Рис. 9.19. Организация прерывания при наличии одного устройства вывода (по сигналу «УВВ готово» устанавливается триггер запроса на прерывание и возникает прерывание ЦП. Сигнал «запись в УВВ» синхронизирует запись информации в порт вывода и сбрасывает триггер запроса на прерывание)

Рис. 9.20. Блок-схема простой программы, обслуживающей прерывания от устройства вывода

УВВ. На рис. 9.21 изображены блок-схемы главной программы и подпрограммы обработки прерывания. Главная программа совпадает с описанной выше, за тем исключением, что разрешение прерываний предшествует подготовке данных. Подпрограмма обработки прерывания проверяет, подготовила ли главная программа данные. Если данные готовы, то подпрограмма посылает их устройству вывода и передает управление на начало главной программы; если данные не подготовлены, то подпрограмма «делает заметку» о том, что УВВ готово, устанавливая признак PRDY, и возвращает управление главной программе. Следует обратить внимание на то, что главная программа после подготовки данных должна проверить признак PRDY, так как возможно, что он уже установлен процедурой обработки прерывания.

Подпрограмма обработки прерывания должна сохранить содержимое всех тех регистров, которые нужны ей для проверки PRDY и установки PRDY. Это необходимо для возобновления работы главной программы. Следует обратить внимание и на то, что после приема прерывания на обработку все прерывания запрещены до тех пор, пока ЦП

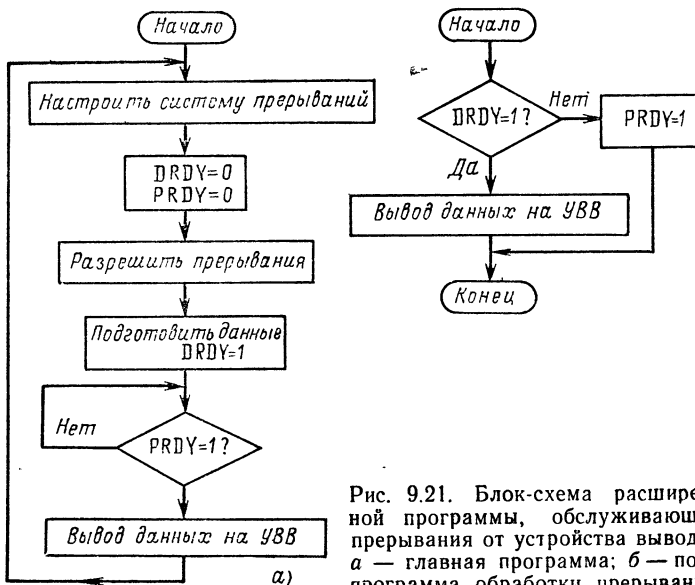
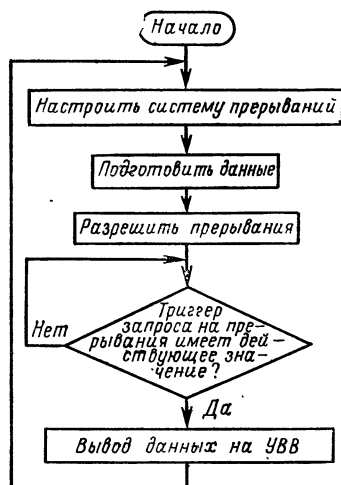


Рис. 9.21. Блок-схема расширенной программы, обслуживающей прерывания от устройства вывода: а — главная программа; б — подпрограмма обработки прерывания

не передаст данные периферийному устройству. Это происходит потому, что триггер запроса на прерывание останется установленным до тех пор, пока ЦП не сбросит его при записи данных в порт вывода.

Программа организована следующим образом:

```

ORG    RSTADD    Установить счетчик адреса
.
.
.Настроить систему прерываний
.
.
CLEAR      DRDY      Сбросить DRDY
CLEAR      PRDY      Сбросить PRDY
ENABLE     INTERRUPTS Разрешить прерывания
.
.
.Подготовить данные
.
.
LOAD       #1        Загрузить 1 в регистр
STORE      DRDY      Установить DRDY
LOAD       PRDY      Загрузить значение PRDY в регистр
SUBTRACT   #1        Вычесть единицу из содержимого
                     регистра
JUMP ON ZERO SEND    Если 0, то перейти по адресу SEND
HALT
ORG        INTADD    Установить счетчик адреса
.
.
. Сохранить содержимое регистров и значения признаков
LOAD       DRDY      Загрузить значение DRDY в регистр
SUBTRACT   #1        Вычесть единицу из содержимого
                     регистра
JUMP ON ZERO SEND    Если 0, то перейти по адресу SEND
LOAD       #1        Загрузить в регистр 1
STORE      PRDY      Установить PRDY
.
.
. Восстановить содержимое регистров и значения признаков
RETURN
ORG        SEND      Возврат управления главной программе
                     Установить счетчик адреса
                     Передать данные периферийному устройству
.
.
JUMP       RSTADD    Передать управление на начало главной
                     программы

```

В некоторых ЭВМ адреса подпрограмм обработки прерываний расположены один за другим таким образом, что для размещения между ними полных программ недостаточно места. В таком случае по этим адресам должны храниться команды передачи управления собственно подпрограммам обработки прерываний. Эти подпрограммы, как и команды передачи управления, можно разместить в нужных ячейках памяти с помощью псевдокоманды `ORG` ассемблера.

Хотя на рис. 9.17 и 9.19 триггер запроса на прерывание изображен отдельно от порта данных, эти два устройства часто объединяются в одном аппаратном модуле. Они объединены, например, в порте ввода-вывода Intel 8212 и в адаптере интерфейса периферийных устройств (PIA) фирмы Motorola, как это было описано в гл. 8. Частью этих устройств являются и схемы, с помощью которых сигнал сброса или схема выбора порта сбрасывают триггер запроса на прерывание.

Сочетание ввода и вывода в системах, использующих прерывания

Прерывания ввода и вывода можно объединить в единой системе ввода-вывода информации по сигналам прерывания. В такой системе по сигналу прерывания от устройства ввода начинается процесс подготовки к приему данных; сигнал прерывания от устройства вывода информирует ЦП, что периферийное устройство готово, после этого ЦП начнет передачу данных, как только будет закончена их подготовка. Простые системы подобного рода — низкоскоростное оконечное устройство связи, система сбора данных, преобразователь кодов, контроллер передачи информации «клавиатура — магнитная лента» или «клавиатура — гибкий диск», а также пульт управления.

На рис. 9.22 показана аппаратура, необходимая для организации такой системы прерываний. Порты ввода-вывода здесь такие же, как и на рис. 9.17 и 9.19. В каждом порте ввода фиксируются признаки прерываний, проверяя которые процессор может определить источник прерывания. Сначала рассмотрим систему прерываний с одним входом запросов на прерывание, без применения векторов. Использование более сложных систем прерываний будет описано далее.

Система работает следующим образом:

Шаг 1. Устройство ввода посылает сигнал «данные готовы» и данные. Сигнал «данные готовы» устанавливает триггер запроса на прерывание и вызывает прерывание работы процессора, который находится в состоянии останова.

Шаг 2. В ответ на запрос на прерывание ЦП начинает выполнять подпрограмму обработки прерывания, размещенную в памяти начиная с адреса `INTADD`.

Шаг 3. Проверяя триггеры прерываний, ЦП определяет источник прерывания и передает управление процедуре обработки прерывания от устройства ввода.

Шаг 4. Центральный процессор читает вводимые данные и сбрасывает триггер прерывания от устройства ввода.

Шаг 5. Центральный процессор готовит выводимые данные и ожидает прерывания от устройства вывода.

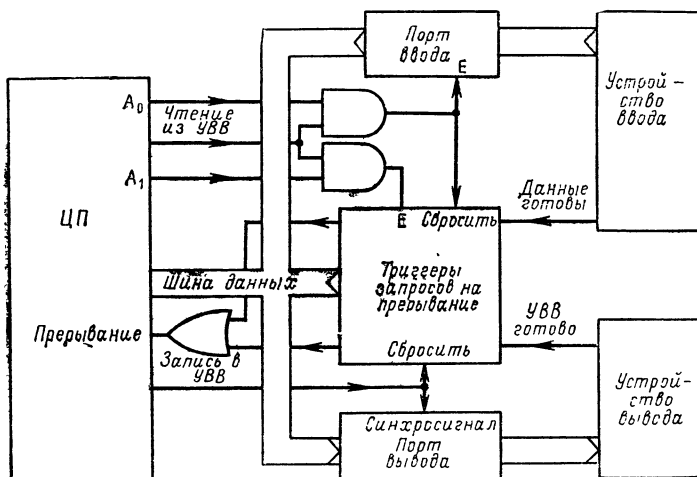


Рис. 9.22. Система управления по сигналам прерывания, в которой сочетаются ввод и вывод (с помощью линии адреса A_0 выбирается порт ввода, с помощью линии адреса A_1 — блок триггеров запросов на прерывание)

Шаг 6. Устройство вывода посылает сигнал «УВВ готово». По этому сигналу устанавливается триггер запроса на прерывание и прерывается работа процессора.

Шаг 7. В ответ на запрос на прерывание ЦП начинает выполнять программу обработки прерывания, размещенную начиная с адреса INTADD.

Шаг 8. Проверяя триггеры прерываний, ЦП определяет источник прерывания и передает управление процедуре обработки прерывания от устройства вывода.

Шаг 9. Центральный процессор посылает выводимые данные (при этом сбрасывается триггер прерывания от устройства вывода) и возвращается в состояние останова, в котором он находился перед первым шагом.

Если прерывание от устройства вывода возникает раньше, чем данные будут подготовлены, то система может просто «сделать заметку» о получении сигнала прерывания от устройства вывода (так же как и в предыдущем примере). При использовании описанной процедуры предполагается, что две операции ввода никогда не перекрываются, т. е. что процессор всегда завершает чтение очередной порции данных до получения следующей.

В программе, описывающей работу системы, изображенной на рис. 9.22, объединены приведенные выше процедуры ввода и вывода данных. Признаки DRDY и PRDY указывают на готовность данных и устройства вывода. В целом программа выглядит следующим образом:

```
ORG RSTADD  Установить счетчик адреса
```

```
      .      Настроить систему прерываний
```

* Сбросить признаки готовности данных и UBB:		
CLEAR	DRDY	Сбросить DRDY
CLEAR	PRDY	Сбросить PRDY
ENABLE	INTERRUPTS	Разрешить прерывания
HALT		Останов
ORG	INTADD	Установить счетчик адреса
.		
. Сохранить содержимое регистров и значения признаков		
* Определить источник прерывания		
READ	LATCHES	Прочитать значения триггеров прерываний из порта-фиксатора
AND	IFLAG	Маскированием выделить признак прерывания от устройства ввода
.		
JUMP ON NOT ZERO	RCV	Если не 0, то перейти на подпрограмму RCV («принять»)
NOT ZERO		
* Прерывание от устройства вывода		
* Послать данные, если они подготовлены		
LOAD	DRDY	Загрузить значение DRDY в регистр
SUBTRACT	#1	Вычесть 1 из содержимого регистра
JUMP ON ZERO	SEND	Если 0, то перейти на подпрограмму SEND («послать»)
* «Сделать заметку» о готовности устройства вывода		
LOAD	#1	Загрузить в регистр единицу
STORE	PRDY	Установить PRDY
.		
. Восстановить содержимое регистров и значения признаков		
.		
RETURN		Возвратить управление главной программе
* (маска IFLAG имеет 1 в разряде, соответствующем прерыванию		
* от устройства ввода, и 0 — в остальных разрядах)		
ORG	RCV	Установить счетчик адреса
ENABLE	INTERRUPTS	Разрешить прерывания
.		
. Подготовить данные для вывода		
.		
.		
LOAD	#1	Загрузить в регистр 1
STORE	DRDY	Установить DRDY
LOAD	PRDY	Загрузить значение PRDY в регистр
SUBTRACT	#1	Вычесть 1 из содержимого регистра
JUMP ON ZERO	SEND	Если 0, перейти на подпрограмму «послать» (SEND)
HALT		Останов
ORG	SEND	Установить счетчик адреса

. Передать данные периферийному устройству
JUMP RSTADD Передать управление на начало главной программы

При использовании такой процедуры программист должен точно определять, когда должны быть разрешены прерывания. Приведем некоторые сложные ситуации:

1. Первым возникает прерывание от устройства вывода. Процессор должен установить DRDY (т. е. «устройство вывода готово») и либо запретить прерывание от устройства вывода, либо послать на UBB символ-разделитель, который игнорируется устройством. Центральный процессор должен сбрасывать или запрещать прерывание от устройства вывода так, что прерывание от устройства ввода при этом не запрещается.

2. Прерывание от устройства вывода возникает до того, как данные будут подготовлены. Центральный процессор должен установить PRDY, но не возобновлять разрешение прерываний.

3. Перед прерыванием от устройства вывода возникает второе прерывание от устройства ввода. Центральный процессор должен либо запретить прерывание от устройства ввода (после того, как принял одно прерывание от устройства ввода), либо сохранить новые вводимые данные с целью их последующей обработки. Центральный процессор не может просто игнорировать прерывание от устройства ввода, не сбрасывая его.

При выполнении подпрограмм обработки прерываний всегда возникают «мертвые» промежутки времени, т. е. такие, когда система прерываний находится в состоянии «запрещено» и данные могут быть потеряны. Если прерывания возникают настолько часто, что наличие таких промежутков становится серьезной проблемой, то, возможно, систему ввода-вывода вообще не следует строить на основе аппарата прерываний. Системы, управляемые по сигналам прерывания, хорошо функционируют, во-первых, когда процессор может справиться с обработкой прерываний при максимально возможной в данной системе скорости обмена данными, и, во-вторых, когда между моментами поступления запросов на прерывания проходит большое количество машинных циклов.

Логика программы становится проще, если два источника прерываний различаются на аппаратном уровне. Применяя два отдельных входа запросов на прерывания или векторную систему прерываний, можно исключить аппаратные и программные средства, необходимые для определения источника прерывания. Полезной также оказывается и система приоритетов. Прерывание от устройства ввода получит меньший приоритет и будет запрещено после принятия на обработку предыдущего прерывания от устройства ввода. Вместе с тем прерывание от устройства вывода с более высоким приоритетом останется по-прежнему разрешенным. В этом случае система не примет на обработку новые прерывания от устройства ввода до тех пор, пока процессор не подготовит и не передаст данные устройству вывода.

Некоторые особенности таких простых систем, управляемых по сигналам прерывания, имеют большое значение:

1. Прежде чем разрешить прерывания, система должна инициализировать систему прерываний и установить начальные значения переменных параметров.

2. Источник прерывания может определяться программой обработки прерывания путем проверки признаков прерывания, хранящихся в порте ввода. Другой подход к определению источника прерывания заключается в использовании отдельных входов запросов на прерывание или прерываний по вектору. В последнем случае процессору могут указываться различные процедуры обработки для различных источников прерывания.

3. Подпрограмма обработки прерывания, если она возвращает управление главной программе, должна сохранять любые регистры и признаки, используемые в подпрограмме.

4. Подпрограмма обработки прерывания должна разрешить прерывания перед возвратом управления главной программе; подпрограмма может разрешить прерывания даже раньше, если поступление новых прерываний не помешает работе подпрограммы. С помощью системы приоритетов можно блокировать конкретные прерывания, которые могут помешать подпрограмме.

5. Триггер запроса на прерывание следует сбрасывать. Часто это выполняется аппаратно.

6. При разработке системы следует принимать во внимание «мертвые» промежутки времени, когда система прерываний отключена. Поэтому недостаточно заботиться только о фиксации сигналов прерываний: сигналы, возникающие слишком часто, также могут быть потеряны. Время реакции системы ограничивает скорость, с которой система успевает обрабатывать прерывания.

В системах, управляемых по сигналам прерывания, очень важна реентерабельность программ. Это свойство позволяет не только осуществлять многоуровневые прерывания, но и использовать в программах обработки прерываний подпрограммы, которые исполнялись в момент прерывания. К таким подпрограммам относятся подпрограммы преобразования кодов, различные действия с символами, подпрограммы проверки и исправления ошибок, подпрограммы управления вводом-выводом. Составление программ, управляемых по сигналам прерывания, — дело достаточно трудное, даже если не заботиться о реентерабельности программ.

Систему, имеющую только одно устройство ввода и одно устройство вывода, можно, конечно, расширить. Можно было бы добавить другие УВВ, сигнализаторы тревоги, таймеры, панели управления, схемы, предупреждающие о падении напряжения питания, кнопки «внешнее прерывание» (breakpoint switches) — все эти устройства могут быть источниками сигнала прерывания. Если нельзя допустить чрезмерной перегрузки процессора и программиста, то с ростом числа возможных источников прерывания потребуются и более сложные аппаратные средства. Любая из следующих задач может оказаться основной: определение источника прерывания, решение вопроса о том, какие прерывания следует разрешать в конкретные моменты времени, управление сигналами прерываний. Применение векторных прерываний мо-

жет существенно уменьшить процессорное время, сократить программное обеспечение и аппаратные средства, необходимые для идентификации источника. В настоящее время большинство фирм, производящих микропроцессоры, поставляет специальные контроллеры прерываний, которые используются в системах сбора данных, коммутации, управления технологическими процессами и производством, системах мониторинга и обеспечения безопасности и т. д. Операционные системы, предназначенные для работы в реальном масштабе времени, смогут значительно облегчить проектирование систем, связанных с обработкой прерываний¹.

9.4. СИСТЕМЫ ПРЕРЫВАНИЙ КОНКРЕТНЫХ МИКРОПРОЦЕССОРОВ

Микропроцессор Intel 8080

У микропроцессора Intel 8080 имеется единственный вход запросов на прерывание, который можно заблокировать (запретить прерывания) программно и который автоматически блокируется по сигналу «сброс» и в момент приема прерывания на обслуживание. Если запрос на прерывание возникает, когда работа системы прерываний разрешена, то процессор завершает выполнение текущей команды и затем выполняет цикл «подтверждение запроса на прерывание». Особенности этого цикла следующие:

1. Признак INTA (разряд 0 информации о состоянии) равен 1;
2. MEMR (разряд 7 информации о состоянии) равен 0;
3. DBIN имеет действующее значение, что указывает на цикл ввода.
4. Содержимое счетчика команд не изменяется.

Таким образом, ЦП производит выборку и декодирование команды, но выбирает команду не из памяти (так как MEMR = 0). Остальная информация, необходимая для реакции на запрос прерывания, обеспечивается программой или внешней аппаратурой. Следовательно, в общем случае для системы прерываний, построенной на базе процессора Intel 8080, требуется значительное количество внешней аппаратуры.

У микропроцессора 8080 имеется команда, специально предназначенная для работы в системах с прерываниями. Это команды RST (повторный пуск) — однобайтная команда (перехода к подпрограмме), которую внешняя аппаратура может поместить на шину данных во время цикла «подтверждение запроса на прерывание». По команде RST текущее значение счетчика команд помещается в стек и происходит передача управления по адресу $N_2N_1N_0000$, где $N_2N_1N_0$ — трехразрядный номер, являющийся частью команды. В двоичной форме команда RST выглядит следующим образом: $11N_2N_1N_0111$. Сгенерировать эту команду несложно, так как разряды, равные единице, легко получить, подключая линии данных к полюсу +5 В источника, а $N_2N_1N_0$ можно получить от шифратора «3-из-8».

Таблица 9.2. Адреса подпрограмм обработки прерываний при использовании команды RST

Команда	Адрес подпрограммы обработки (шестнадцатиричный)	Команда	Адрес подпрограммы обработки (шестнадцатиричный)
RST0	0000	RST4	0020
RST1	0008	RST5	0028
RST2	0010	RST6	0030
RST3	0018	RST7	0038

¹K. Burgett and E. F. O'Neill. An Integral Real-Time Executive for Microcomputers. Computer Design, vol. 16, № 7, July 1977, p. 77—82.


```

IN  PORT    ;Считать признаки прерываний
RAR        ;Проверить бит 0
JC  INT0
RAR        ;Проверить бит 1
JC  INT1
JMP INT2

```

Программы, начинающиеся с адресов INT0, INT1 и INT2, обрабатывают прерывания от определенных устройств. Обобщение приведенной программы на случай большего числа источников не вызывает затруднений, однако в результате возрастет время, которое требуется процессору для проверки всех признаков прерываний. Для подобных систем требуется очень немного внешнего оборудования.

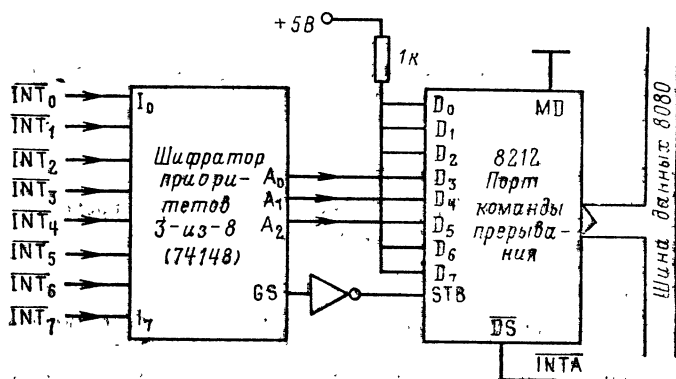


Рис. 9.24. Генерация команд RST с помощью ТТЛ-шифратора (генерируемые команды RST приведены в табл. 9.3)

ния, однако при этом не используются преимущества, которые могла бы дать векторная система прерываний микропроцессора Intel 8080. Такие системы работают удовлетворительно только тогда, когда имеется лишь один или два источника прерываний.

Несколько векторов прерываний. Если в системе имеется несколько различных векторов прерываний, то необходимо, чтобы порт команды прерывания помещал одну из возможных команд RST на шину данных. На рис. 9.24 показана схема функционирования такого порта. Сигнал INTA открывает буфера порта 8212 и разрешает этому устройству управлять шиной данных. Команда RST генерируется схемой: биты 2—4 задаются ТТЛ-шифратором 3-из-8 (74148). На входы шифратора (с низким действующим значением) обычно приходят сигналы от выходов INT (с низким действующим значением) портов ввода-вывода 8212. С этих же выходов должен подаваться процессору и сигнал прерывания. В табл. 9.3 приведены команды RST и соответствующие им приоритеты в порядке возрастания.

Таблица 9.3. Векторы прерываний, поступающие от шифраторов приоритетов

Приоритет	Команда	Приоритет	Команда
0	RST 7	4	RST 3
1	RST 6	5	RST 2
2	RST 5	6	RST 1
3	RST 4	7	RST 0

В типичной программе, обрабатывающей прерывания в системе, аналогичной той, которая изображена на рис. 9.24, имеются различные подпрограммы для обработки прерывания от каждого периферийного устройства. Например, программа, в которой используются три вектора прерываний с младшими приоритетами, может быть организована следующим образом:

```

;
; Передача управления главной программе при сбросе
;
ORG 0
JMP MAIN

;
; Адрес обработки прерывания с приоритетом 2
;
ORG 28H
JMP INT2 ; Передача управления подпрограмме обработки прерыва-
          ния с приоритетом 2
;
;
; Адрес обработки прерывания с приоритетом 1
ORG 30H
JMP INT1 ; Передача управления подпрограмме обработки прерыва-
          ния с приоритетом 1
;
;
; Адрес обработки прерывания с приоритетом 0
ORG 38H
JMP INT0 ; Передача управления подпрограмме обработки прерывания
          с приоритетом 0

```

Адреса обработки прерываний расположены один за другим таким образом, что для размещения между ними полных подпрограмм обработки прерываний недостаточно места.

Сохранение состояния, предшествовавшего прерыванию. По команде RST старое значение счетчика команд автоматически сохраняется в стеке для того, чтобы подпрограммы обработки прерывания могла восстановить это значение с помощью команды RETURN. Перед тем как разрешить работу системы прерываний, главная программа должна установить начальное значение указателя стека. Типичная последовательность команд имеет следующий вид:

```

;
; Передача управления главной программе по сбросу
;
ORG 0
JMP MAIN

;
; Настроить систему прерываний и разрешить ее работу
;
ORG MAIN
LXI SP, LASTM ; Расположить стек в конце поля памяти
EI             ; Разрешить прерывания

```

Подпрограммы обработки прерываний должны сохранять и восстанавливать все используемые ими регистры, признаки и ячейки памяти. Простейший прием состоит в том, чтобы сохранить старое состояние процессора в вершине стека при входе в процедуру обработки прерывания и восстановить это состояние, пользуясь вершиной стека, перед возвратом управления. Обычная последовательность команд при входе в подпрограмму обработки прерывания такова:

```

; Сохранить содержимое регистров и значения признаков
PUSH PSW
PUSH B
PUSH D
PUSH H

```


;Восстановить содержимое регистров и значения признаков

POP H
POP D
POP B
POP PSW

По команде PUSH (или POP) содержимое двух регистров сохраняется в стеке (восстанавливается из стека). Следует обратить особое внимание на пару регистров PSW (Processor Status Word — содержимое регистра признаков¹), состоящее из аккумулятора (старшие разряды) и признаков (младшие разряды). На рис. 9.25 показано, как расположена в стеке информация о старом состоянии процессора после выполнения последовательности команд входа в подпрограмму. Подпрограмма должна сохранить содержимое аккумулятора перед выполнением любой операции ввода (это нужно для полинга) и восстановить регистры перед выполнением команды RETURN.

Приоритетные системы прерывания. Для приоритетной системы прерывания необходимы: регистр, шифратор, компаратор и некоторые схемы (см. рис. 9.15). Если выходные сигналы шифратора имеют низкое действующее значение, то в регистр следует загружать дополнение приоритета до старшего, а сигнал прерывания получать с выхода «меньше чем» компаратора. В системе также необходима схема, отключающая компаратор. Эта схема требуется для приема на обслуживание прерываний самого низкого приоритета, так как в системе прерываний разрешены только прерывания с приоритетом, большим текущего.

Этот простой способ организации приоритетных систем прерывания хорошо согласуется с конфигурацией систем на основе МП Intel 8080, так как от ТТЛ-шифратора можно получать как команду RST, так и входные сигналы для компаратора.

Применение блока приоритетного прерывания Intel 8214 (Intel 8214 Priority Interrupt Control Unit), который изображен на рис. 9.26, позволяет значительно уменьшить число корпусов в микропроцессорной системе, так как в этом устройстве объединены шифратор приоритета, компаратор, регистр приоритета (priority register) и схемы управления. Отметим наиболее важные компоненты устройства 8214.

1. Четырехразрядный регистр текущего состояния (current status register), отпираемый сигналом $\overline{\text{ECS}}$ (enable current status). Процессор может рассматривать этот регистр как порт вывода. Разряд $\overline{\text{SGS}}$ — старший разряд регистра состояния (SGS — status group select — выбор состояния в группе). Если уровень $\overline{\text{SGS}}$ низкий, то работа компаратора разрешена. В табл. 9.4 приведены возмож-

Таблица 9.4. Содержимое регистра текущего состояния Intel 8214 и соответствующие значения приоритетов прерываний, принимаемых на обслуживание

Наименьший приоритет прерываний, принимаемых на обслуживание	Содержимое регистра текущего состояния Intel 8214				Наименьший приоритет прерываний, принимаемых на обслуживание	Содержание регистра текущего состояния Intel 8214			
	$\overline{\text{SGS}}$	$\overline{\text{B2}}$	$\overline{\text{B1}}$	$\overline{\text{B0}}$		$\overline{\text{SGS}}$	$\overline{\text{B2}}$	$\overline{\text{B1}}$	$\overline{\text{B0}}$
0	1	1	1	1	4	0	1	0	0
1	0	1	1	1	5	0	0	1	1
2	0	1	1	0	6	0	0	1	0
3	0	1	0	1	7	0	0	0	1

¹ По принятой в переводе терминологии понятие «слово состояния процессора» характеризует машинный цикл МП, а не содержимое регистра признаков FLAGS. (Прим. пер.)

Рис. 9.25. Запоминание состояния процессора Intel 8080 в стеке (PCH и PCL — соответственно восемь старших и восемь младших разрядов счетчика команд. Порядок операций таков: RST, PUSH PSW, PUSH B, PUSH D и PUSH H)

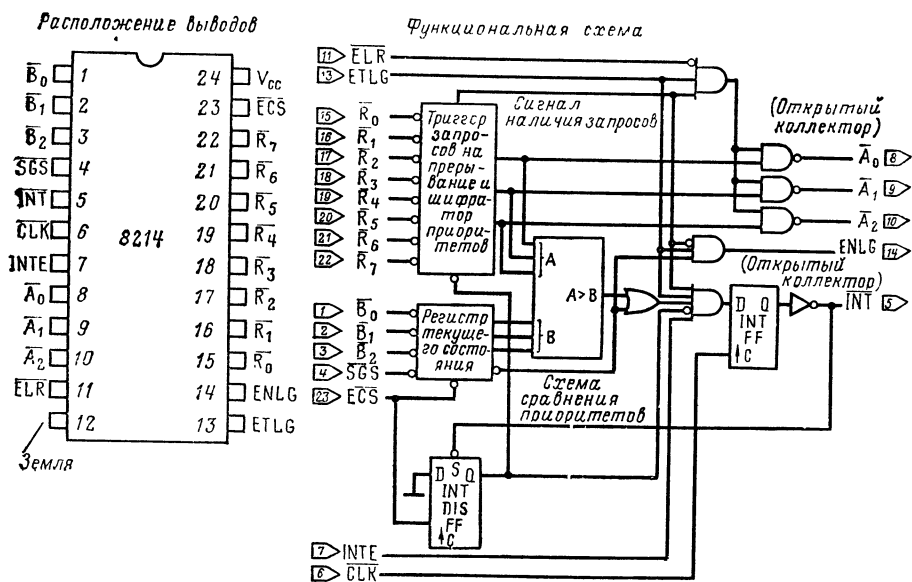
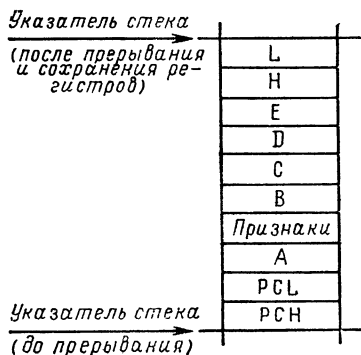
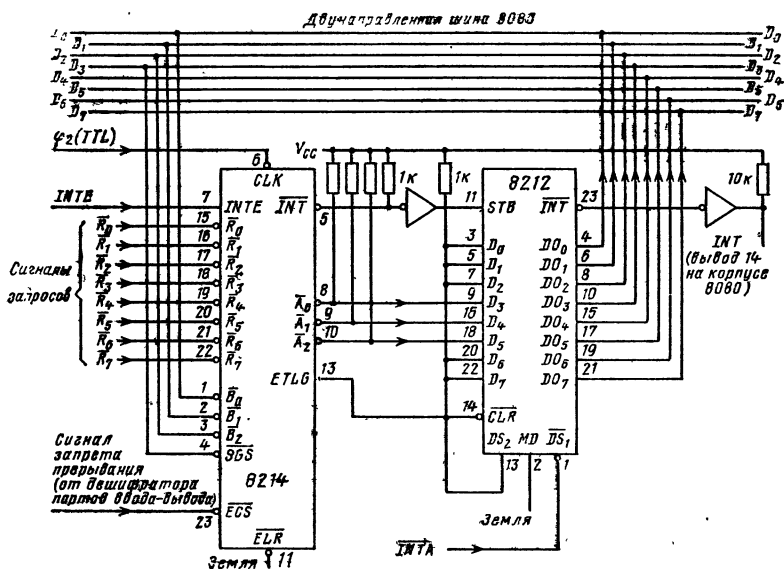


Рис. 9.26. Блок приоритетного прерывания Intel 8214: \bar{R}_0 — \bar{R}_7 (request levels) — сигналы запроса на прерывание восьми уровней (\bar{R}_7 имеет высший приоритет); \bar{B}_0 — \bar{B}_2 — код приоритета текущей программы; SGS (status group select) — выбор состояния в группе [подается от дешифратора состояний в многоуровневых ($R > 8$) системах прерываний]; \bar{ECS} (enable current status) — сигнал запрета прерывания; INTE (interrupt enable) — сигнал разрешения прерывания; CLK (clock) — сигнал синхронизации триггера запроса на прерывание; ELR (enable level read) — сигнал разрешения выдачи кода приоритета; ETLG (enable this level group) — сигнал разрешения работы данного блока приоритетного прерывания в многоблочной системе прерываний; \bar{A}_0 — \bar{A}_2 (request levels) код вектора прерывания; INT — сигнал запроса на прерывание (с низким действующим значением); ENLG (enable next level group) — сигнал разрешения работы следующего (в сторону убывания приоритетов) блока приоритетного прерывания в многоуровневой системе прерываний (\bar{A}_0 — \bar{A}_2 и INT — со свободного коллектора)



$D_7 \quad D_6 \quad D_5 \quad D_4 \quad D_3 \quad D_2 \quad D_1 \quad D_0$

Приоритет запроса	RST	1	1	A_2	A_1	A_0	1	1	1
Низший	0	7	1	1	1	1	1	1	1
	1	6	1	1	1	0	1	1	1
	2	5	1	1	1	0	1	1	1
	3	4	1	1	0	0	1	1	1
	4	3	1	1	0	1	1	1	1
	5	2	1	1	0	0	1	1	1
	6	1	1	1	0	1	1	1	1
Высший	7	0*	1	1	0	0	1	1	1

Рис. 9.27. Восьмиуровневая система приоритетного прерывания для процессора Intel 8080 на базе контроллера 8214. [Пояснение к таблице на рис. 9.27: *RST 0 загрузит счетчик команд адресом 0 и вызовет тем самым ту же процедуру, что и сигнал, поданный на вход «сброс» процессора 8080. Этим способом можно реинициализировать систему, в которой подпрограмма инициализации начинается с адреса 0 (предупреждение для системных программистов)]

ные состояния регистра и соответствующие им наименьшие приоритеты прерываний, принимаемых на обслуживание.

2. Регистр-фиксатор запросов на прерывание и шифратор приоритетов. Трехразрядный выход от этих устройств (с низким действующим значением) поступает на компаратор и на внешние линии для генерации команды RST.

3. Трехразрядный компаратор приоритетов.

На рис. 9.27 показано использование блока приоритетного прерывания 8214 в качестве контроллера восьмиуровневой приоритетной системы прерываний для микропроцессора Intel 8080. Разработчик должен учитывать следующие требования к системам, конструируемым на базе Intel 8214:

1. Перед тем как разрешить работу системы прерываний, главная программа должна загрузить в регистр приоритета слово, состоящее из восьми единиц (FF).

2. Программа должна сохранить «копию» текущего приоритета в ОЗУ, так как содержимое регистра текущего состояния 8214 недоступно ЦП для чтения.

3. Каждая подпрограмма обработки прерывания должна сохранить старый приоритет в стеке, а новый приоритет поместить в регистр состояния 8214, прежде чем выдать команду разрешения прерываний, а также восстановить старый приоритет перед возвращением управления главной программе.

4. Все значения приоритетов представляются в дополнительном коде.

5. Устройство 8214 не фиксирует запросы на прерывания. Обычно эти входные сигналы фиксирует порт 8212.

Типичная программа, обрабатывающая прерывания, при использовании Intel 8214 организована следующим образом:

```
;
; Процедура сброса
; ORG 0
; JMP MAIN ; Передать управление главной программе
;
; Точка входа для прерывания с приоритетом 6
;
; ORG 08H
; JMP INT6
; Точка входа для прерывания с приоритетом 5
;
; ORG 10H
; JMP INT5
;
; Точка входа для прерывания с приоритетом 4
; ORG 18H
; JMP INT4
;
; Точка входа для прерывания с приоритетом 3
;
; ORG 20H
; JMP INT3
;
; Точка входа для прерывания с приоритетом 2
;
; ORG 28H
; JMP INT2
;
; Точка входа для прерывания с приоритетом 1
;
; ORG 30H
; JMP INT1
;
; Точка входа для прерывания с приоритетом 0
;
; ORG 38H
; JMP INTO
;
; Главная программа
; ORG MAIN
;
; Установить начальное значение регистра состояния 8214
;
; MVI A, 00001111B
; OUT SPORT ; Приоритет = 0
; STA PRTY ; Сохранить «копию» приоритета
```

```

EI          ; Разрешить прерывания
            ; Главная программа
;
; Подпрограмма обработки прерывания с приоритетом 3
;
ORG INT3
PUSH PSW    ; Сохранить содержимое регистров
PUSH B
PUSH D
PUSH H
LDA PRTY
PUSH PSW    ; Сохранить старый приоритет
MVI A, 00000100B ; Новый приоритет = 3
OUT SPORT   ; Сохранить «копию» нового приоритета
STA PRTY
EI
; Подпрограмма обработки прерывания
;
POP PSW     ; Восстановить старый приоритет
STA PRTY
OUT SPORT
POP H       ; Восстановить регистры
POP D
POP B
POP PSW
RET

```

В этом примере показана только подпрограмма обработки прерывания, имеющего приоритет 3; другие подпрограммы построены аналогично. Следует обратить внимание на то, что каждая подпрограмма обработки прерывания должна загрузить значение приоритета в регистр состояния 8214, перед тем как разрешить прерывания, а также восстановить старое значение приоритета перед возвратом управления прерванной программе.

Системы прерываний с расширенными возможностями. Системы прерываний, основанные на использовании команды RST и контроллера 8214, могут с успехом применяться для обработки прерываний при небольшом числе входов запросов на прерывание и не требуют для своей работы сложной аппаратуры и программного обеспечения. Но в ситуациях, когда используется большее число входов запросов на прерывание и требуется большая гибкость при обработке прерываний, у этих систем обнаруживаются следующие недостатки:

1. Существует только восемь команд RST. Для систем, целиком основанных на векторном принципе и использующих более восьми входов запросов на прерывание, потребуется значительно больше аппаратуры и программных средств.

2. Точки входа в подпрограммы обработки прерываний всегда находятся на нулевой странице памяти. Во многих системах нулевая страница памяти используется для иных целей.

3. Точка входа в подпрограмму обработки прерывания, имеющего самый высокий приоритет, совпадает с адресом, на который передается управление при сбросе. Чтобы отличить вход «сброс» от сигнала прерывания, требуется дополнительное оборудование и программные средства.

4. Алгоритм, по которому прерываниям с различными векторами присваиваются приоритеты, фиксирован. Программа может только разрешать и запрещать прерывания, имеющие приоритет, равный или меньший данного конкретного значения. Программа не может (что было бы удобно) циклически «сдвигать» приоритеты, а также изменять приоритеты необслуженных прерываний.

5. Можно разрешать или запрещать только целые группы прерываний. Нет возможности разрешать или запрещать прерывания, имеющие данный и только данный приоритет.

Многие из этих недостатков организации прерываний в более сложных системах устраняются с помощью программируемого контроллера прерываний Intel 8259 (рис. 9.28). С помощью этого устройства можно разработать

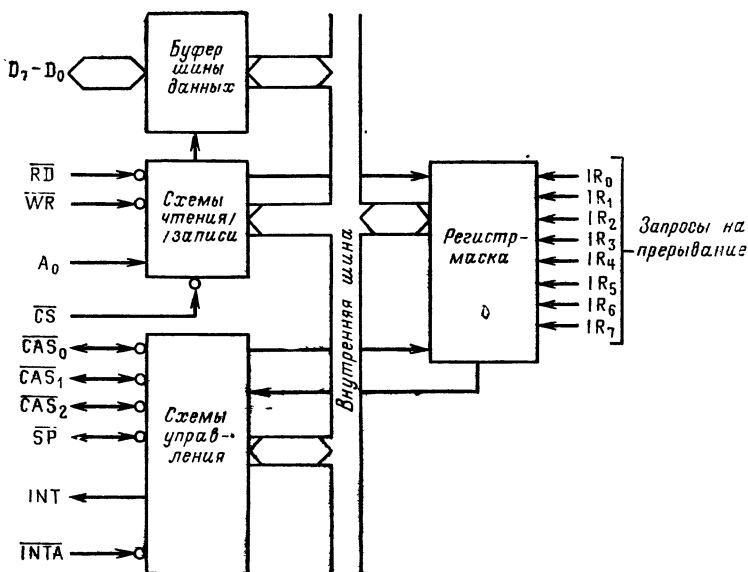


Рис. 9.28. Программируемый контроллер прерываний Intel 8259

сколь угодно сложные алгоритмы работы с приоритетами. Устройство вырабатывает трехбайтную команду CALL с программируемым базовым адресом, поэтому адрес подпрограммы обработки прерывания может быть размещен в памяти где угодно. Системный контроллер Intel 8228 обеспечивает три синхросигнала, необходимых для помещения команды CALL на шину дан-

Таблица 9.5. Программируемые режимы прерываний, возможные при использовании контроллера 8259

Режим	Действие
Полностью распределенные приоритеты	Линиям запросов на прерывания присписывается фиксированный приоритет: линии 0 — высший, линии 7 — низший
Автоматическое циклическое изменение приоритетов	Равные приоритеты. После того, как прерывание с данным приоритетом было обслужено, этому прерыванию присписывается низший приоритет (до тех пор, пока не возникнет новое прерывание)
Индивидуальное назначение приоритета	Система программно присписывает одному из прерываний низший приоритет. Приоритеты остальных прерываний получают путем последовательного отсчета от указанного приоритета
Просмотр	Система программно проверяет (через регистр состояния системы прерываний) состояние системы прерываний (закодированное так, как кодируются приоритеты)

ных. Каждое устройство 8259 обеспечивает восемь векторов, а система прерываний может быть скомбинирована из восьми таких устройств.

Возможные режимы работы с прерываниями при использовании контроллера Intel 8259 перечислены в табл. 9.5. Очевидно, что устройство 8259 дает возможность построить более гибкую и обладающую более широкими возможностями систему прерываний, чем устройство Intel 8214.

Обработка микропроцессором Intel 8080 типичных прерываний

Прерывания от внешнего переключателя. Такие прерывания используются для организации приостановки работы (breakpoints), выполнения программы в контрольных точках и перехода на управление с лицевой панели; они позволяют оператору ввести команды или данные. Используя эти прерывания, очень просто обеспечить такое средство отладки программ, как распечатка по запросу содержимого всех регистров с последующим возобновлением работы прерванной программы.

Приведем программу обработки прерывания (предполагается, что либо системный монитор, либо главная программа устанавливает начальное значение указателя стека и разрешает прерывания):

```

ORG INTADD
PUSH PSW ; Запомнить содержимое регистров в стеке
PUSH B
PUSH D
PUSH H
LXI H, 0 ; Использовать указатель стека
DAD SP ; как указатель данных
MVI B, 10 ; Суммарная длина всех регистров и PC = 10 байт
PRREG: MOV A, M
CALL PRINT ; Напечатать содержимое регистра
INX H
DCR B
JNZ PRREG
POP H ; Восстановить регистры
POP D
POP B
POP PSW
EI ; Разрешить прерывания
RET

```

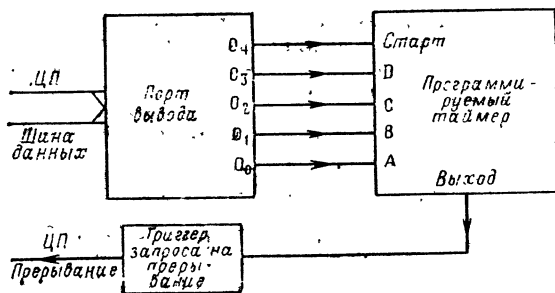
Эта программа распечатывает содержимое всех регистров общего назначения, аккумулятора, регистра признаков и счетчика команд. Необходимое преобразование кодов, форматизацию и передачу содержимого аккумулятора системному печатающему устройству должна выполнять подпрограмма PRINT.

Прерывания по таймеру. Прерывания по таймеру удобны для работы в реальном масштабе времени, а также для управления работой периферийных устройств. В системе, например, можно использовать программируемый таймер, в котором для задания оверной тактовой частоты применяется RC-цепь (рис. 9.29). На входы таймера подается сигнал START, по которому начинается отсчет интервала времени, и 4 бита данных, определяющих длину интервала.

Предположим, что:

- 1) когда на входы таймера подается полубайт FULL, то таймер вырабатывает сигнал длительностью 9,1 мс;
- 2) когда на входы таймера подается полубайт HALF, то таймер вырабатывает сигнал длительностью 4,55 мс;
- 3) маска START имеет единицу в разряде, соответствующем сигналу START таймера, и нули в остальных разрядах; так, для системы, изображенной на рис. 9.29, маска START имеет вид 00010000;
- 4) разряд 7 порта, обслуживающего телетайп, связан с линией последовательного ввода информации с телетайпа.

Рис. 9.29. Применение программируемого таймера (или генератора частоты передачи символов — baud rate generator). [С помощью входных сигналов А, В, С и D (D — старший разряд) определяется число каскадов таймера. Другими словами, увеличение на единицу значения двоичного четырехразрядного числа ABCD вдвое увеличивает длительность временного интервала]



Программа ввода с телетайпа, основанная на применении прерываний от таймера, работает следующим образом:

```

; Проверка стартового бита
SRSTB:      INTTY      ; Считать данные с последовательной
;                  ; линии
ANA  A        ; Получен ли стартовый бит (0)?
JM   SRSTB    ; Нет, ожидать

; Очистить порты для данных и сбросить счетчик разрядов
SUB  A        ; Данные = 0
STA  TTYD     ; Счетчик разрядов = 0
STA  BITCT

; Установить половинную длительность интервала
MVI  A, HALF ; Настроить таймер на половинную длительность интервала
ORI  START   ; Подать сигнал на вход START таймера
OUT  TIMER
XRI  START
OUT  TIMER

; Ожидать получения символа
EI

WAITC: LDA  BITCT ; Приняты ли 11 бит?
CPI  11
JNZ  WAITC ; Нет, ожидать

; Остальная часть программы:
; Обработка прерывания от таймера
ORG  TINTP
PUSH PSW
LDA  BITCT ; Счетчик разрядов = счетчик разрядов + 1
INR  A
STA  BITCT
CPI  10 ; Если счетчик разрядов > 9, то
JNC  STOPS ; проверить стоп-разряды
IN   TTY ; Считать данные с последовательной линии
RAL
LDA  TTYD ; Добавить разряд к принятому символу
RAR

```



```

STA 1TYD
JMP TDLY

```

; Проверка стоп-разрядов

```

STOPS:  IN  TTY      ; Считать данные с последовательной линии
        RAL
        JNC FRERR ; Ошибка формата, если стоп-разряд ≠ 1
        LDA BITCT; Проверить последний стоп-разряд
        CPI 11     ; Если последний, то не устанавливать таймер
        JZ  LAST

```

; Установить полную длительность интервала

```

TDLY:   MVI A, FULL ; Настроить таймер на полную длительность интервала
        ORI START   ; Подать сигнал на вход STAK1 таймера
        OUT TIMER
        EI
        XRI START
        OUT TIMER
LAST:   POP PSW      ; Восстановить содержимое аккумулятора и значения признаков
        RET

```

В промежутках между приемом символов от УВВ ЦП может выполнять другую работу, так как ему не нужно отсчитывать временные интервалы. Возможна такая модификация рассмотренной программы: программно организовать

работу с двумя буферами, т. е. дать возможность программе работать с одним из буферов, в то время как устройство ввода загружает данные в другой буфер.

В большинстве случаев изготовители микропроцессоров предлагают программируемые таймеры, специально разработанные для применения совместно с выпускаемыми ими микропроцессорами. Например, устройство Intel 8253 имеет три 16-разрядных счетчика с программируемыми режимами работы. Одноплатный компьютер Intel SBC 80/20 Single Board Computer использует два таких устройства. Они применяются для выполнения функций, указанных в табл 9 6. Таймер включен также в ОЗУ 8155 емкостью 2 К.

Прерывания от клавишного пульта. Строб-сигнал от клавиатуры может играть роль сигнала запроса на прерывание. Если с клавишного пульта можно вызывать запрос на прерывание, то процессору не нужно постоянно сканировать клавиатуру или проверять, зафиксирован ли

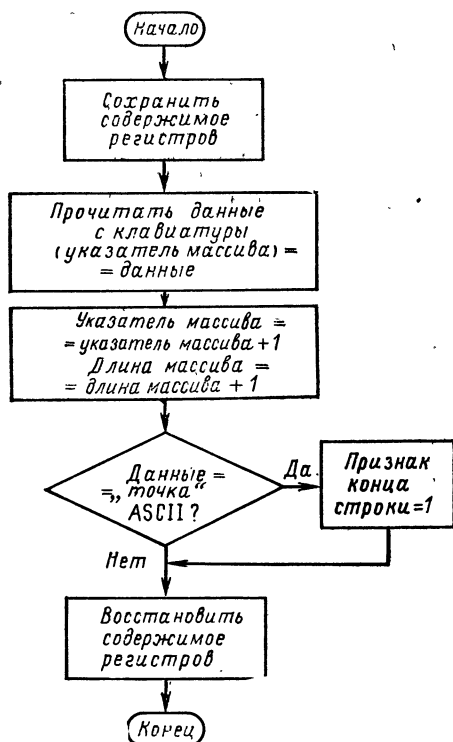


Рис. 9.30. Блок-схема программы обработки прерываний от клавишного пульта

**Т а б л и ц а 9.6. Функции, выполняемые программируемым таймером
одноплатного компьютера Intel SBC 80/20**

Функция	Реализация
Прерывания при достижении предельного значения счетчика	Когда содержимое счетчика временной установки становится равным нулю, генерируется сигнал запроса на прерывание. Эта функция таймера особенно полезна для генерации синхросигналов реального времени
Программируемого одновибратора	По фронту сигнала от внешнего триггера или по команде программы потенциал на выходе таймера становится низким; он снова становится высоким, когда содержимое счетчика временной установки становится равным нулю.
Генератора частоты	<i>N</i> -ичный счетчик. Потенциал на выходе таймера становится низким на время, равное одному периоду внешних тактовых сигналов, а промежуток времени между установлением двух последовательных низких значений потенциала выхода в <i>N</i> раз больше периода внешних тактовых сигналов
Генератора прямоугольных импульсов заданной частоты	Потенциал на выходе таймера остается высоким, пока не завершится первая половина отсчета временного интервала, и становится низким на все время, когда продолжается вторая половина отсчета
Счетчика событий	С помощью схемы выбора режима работы линии синхросигналов вход синхронизации переключается на прием сигналов от внешней системы. Центральный процессор может считать число происшедших событий или же счетчик генерирует сигнал прерывания, когда число событий в системе достигнет <i>N</i> .

**Адреса регистров таймера
(в шестнадцатиричной записи, в адресном пространстве ввода-вывода)**

Регистр	Адрес
Регистр управления	DF
Таймер 1	DC
Таймер 2	DD

Примечание. Загрузка счетчиков таймера выполняется как две последовательные операции вывода с одним и тем же адресом (адреса приведены в таблице).

Входные частоты	
Наименование	Значение
Опорная	1,0752 МГц $\pm 0,1$ % (номинальная длительность периода 0,930 мкс)
Частота событий	1,1 МГц (максимум)

Примечание Максимальная частота событий дана для функции счетчика событий.

Режим	Функция	Выходные частоты (интервалы времени)			
		Один таймер (счетчик)		Два таймера (счетчика)	
		Минимум	Максимум	Минимум	Максимум
0	Прерывания реального времени	1,86 мкс	60,948 мс	3,72 мкс	1,109 ч
1	Программируемого одновибратора	1,86 мкс	60,948 мс	3,72 мкс	1,109 ч
2	Генератора частоты	16,407 Гц	537,61 кГц	0,00025 Гц	268,81 кГц
3	Генератора прямоугольных импульсов заданной частоты	16,407 Гц	537,61 кГц	0,00025 Гц	268,81 кГц

строб. Например, следующая программа обработки прерываний сохраняет входные данные, поступающие от кодирующей клавиатуры, в буферном массиве, обновляет информацию «указатель массива» и «длина массива», а также проверяет, не является ли введенный символ признаком ASCII «конец строки». На рис. 9.30 изображена блок-схема программы. Указатель данных (KPTR) содержит адрес следующей незаполненной ячейки массива; в ячейке «длина массива» (KLENG) содержится число символов в массиве.

```

;
; Программа обработки прерываний от клавиатуры
;
ORG INTADD
;
; Сохранить содержимое регистров в стеке
;
PUSH PSW ; Сохранить содержимое аккумулятора и регистра призна-
; знаков
PUSH H ; Сохранить в стеке регистры H и L
;
; Сохранить полученные с клавиатуры данные в буферном массиве
;
LHLD KPTR ; Сформировать адрес следующего элемента массива
IN KBD ; Считать данные с клавиатуры
MOV M, A ; Сохранить данные в массиве
;
; Обновить указатель массива и счетчик числа элементов
;
INX H
SHLD KPTR ; Обновить указатель массива
;
LXI H, KLENG
INR M ; Обновить счетчик числа элементов
;
; Если получен символ «конец строки» кода ASCII, то
; установить признак «конец строки данных»
;
CPI ' ' ; Принят символ «конец строки»?

```

```

JNZ  ENDKY ; Нет, переход на конец процедуры
LXI  H, EOLF ; Да, установить признак «конец строки»
MVI  M, 1
; Восстановить регистры и вернуть управление
ENDKY: POP  H ; Восстановить содержимое H и L
      POP  PSW ; Восстановить содержимое аккумулятора и регистра признаков
EI ; Разрешить прерывания
RET

```

Прерывания от УАПП. Сигналы УАПП «буфер передатчика пуст» и «буфер приемника заполнен» также можно использовать в роли сигналов прерывания (рис. 9.31). В этом случае процессор не должен будет проверять эти признаки. В программе обработки прерывания можно просто передать данные и сбросить триггер запроса на прерывание.

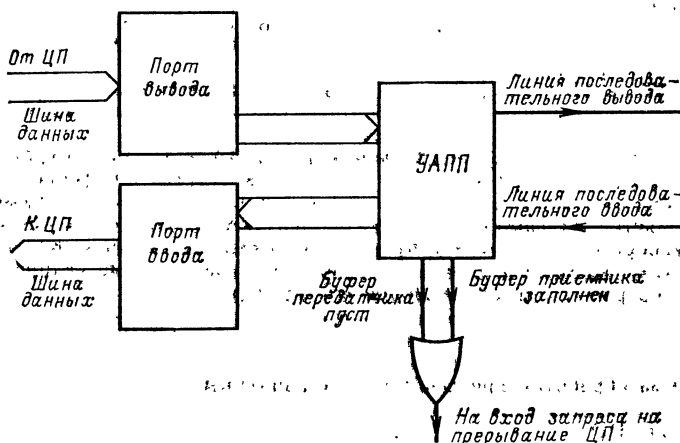


Рис. 9.31. Прерывания от УАПП (совместимый с процессором УАПП имеет в своем составе порты ввода-вывода и логические схемы для формирования сигналов запроса на прерывание)

В программе передачи данных можно воспользоваться буферным массивом точно так же, как это было сделано в программе ввода с клавиатуры. Программа посылает первый элемент массива (если он существует) и уменьшает содержимое счетчика «длина массива»; если массив исчерпан, то программа посылает «пустой символ синхронизации» (SYNCHRONOUS IDLE). Программа имеет следующий вид:

```

;
; Программа передачи, управляемая по сигналам прерывания
;
ORG  TRINT
;
; Сохранить содержимое регистров в стеке
;
PUSH PSW ; Сохранить содержимое аккумулятора и регистра признаков
PUSH  H  ; Сохранить содержимое регистров H и L
;
; Если длина массива равна 0, сформировать
; «пустой символ синхронизации» (SYN)
LXI  H, LENG

```

```

MOV    A, M    ; Считать из памяти значение длины массива
ANA    A        ; Длина массива = 0?
MVI    A, SYN;  Сформировать символ SYN
JZ     SENDC;   Если длина = 0, то послать символ SYN
; Считать данные из массива; обновить значение указателя и счетчика
; длины
;
DCR    M        ; Длина массива = длина массива - 1
LHLD   DPTR     ; Сформировать указатель данных
MOV    A, M      ; Считать данные из памяти
INX    H
SHLD   DPTR     ; Указатель данных = указатель данных + 1
;
; Послать данные УАПП
SENC:  OUT      TPORT ; Послать данные УАПП
;
; Восстановить содержимое регистров и вернуть управление
;
POP     H        ; Восстановить содержимое H и L
POP     PSW      ; Восстановить содержимое аккумулятора и регистра
; признаков
EI      ; Разрешить прерывания
RET

```

Программа обработки прерывания использует счетчик длины массива и указатель данных, чтобы получить информацию от главной программы (данные, которые следует послать) и передать информацию главной программе (тот самый символ из массива, который был послан УАПП).

Прерывания от преобразователей. Аналого-цифровой преобразователь с помощью сигнала запроса на прерывание указывает процессору, что преобразование завершено. Если это необходимо, то программа обработки прерывания может поместить данные в массив. Процессору не требуется проверять признак завершения преобразования и выполнять подпрограмму реализации временной задержки.

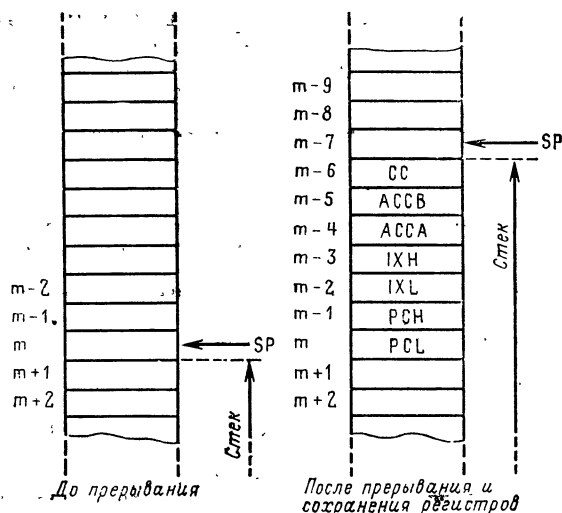
Микропроцессор Motorola 6800

Микропроцессор Motorola 6800 реагирует на прерывание тем, что помещает в счетчик команд новое значение, выбранное из определенной пары ячеек памяти. В табл. 9.7 приведены адреса ячеек памяти, которые используются для хранения адресов обработки различных видов прерываний. Микропроцессор автоматически сохраняет в стеке содержимое всех регистров (рис. 9.32), но у него нет специальных средств для определения источника поступивших на вход прерываний.

Таблица 9.7. Карта памяти векторов прерываний микропроцессора Motorola 6800

Вектор		Описание
Старшие разряды	Младшие разряды	
FFFF	FFFF	Сброс
FFFC	FFFD	Немаскируемое прерывание
FFFA	FFFB	Программное прерывание
FFF8	FFF9	Запрос на прерывание

Рис. 9.32. Запоминание состояния микропроцессора Motorola 6800 в стеке: SP — указатель стека; CC — признаки результата (этот байт называют также байтом состояния процессора); ACCB — аккумулятор В; ACCA — аккумулятор А; IXH — восемь старших разрядов индексного регистра; IXL — восемь младших разрядов индексного регистра; PCH — восемь старших разрядов счетчика команд; PCL — восемь младших разрядов счетчика команд



Микропроцессор Motorola 6800 не вырабатывает сигнала «подтверждение запроса на прерывание» и никак не идентифицирует цикл обработки запроса на прерывание (если не считать используемых в этом цикле адресов). Имеются три команды, которые специально предназначены для обработки прерываний: RTI (RETURN FROM INTERRUPT — ВОЗВРАТ ИЗ ПРЕРЫВАНИЯ), SWI (SOFTWARE INTERRUPT — ПРОГРАММНОЕ ПРЕРЫВАНИЕ) и WAI (WAIT FOR INTERRUPT — ОЖИДАНИЕ ПРЕРЫВАНИЯ). Команда RTI восстанавливает содержимое регистров, используя стек, и осуществляет действия, обратные действиям, совершаемым в момент прерывания. По команде SWI, как и в ответ на внешнее прерывание, в стеке сохраняется содержимое всех регистров; после этого в счетчик команд загружается содержимое пары ячеек с адресами FFFA и FFFB. По команде WAI содержимое всех регистров сохраняется в стеке и процессор переходит в состояние ожидания прерывания.

У микропроцессора Motorola 6800 фактически два входа запросов на прерывания с низким действующим значением: \overline{IRQ} — для обычных прерываний и \overline{NMI} — для немаскируемых прерываний. Эти входы могут обеспечить два вектора прерываний. Однако на практике использование немаскируемого прерывания вызывает затруднения, так как его нельзя запретить программно при настройке системы. Ограничимся рассмотрением входа \overline{IRQ} .

Разряд запрещения прерывания — это четвертый разряд регистра признаков. По команде SEI (SET INTERRUPT) прерывания запрещаются; а по команде CLI (CLEAR INTERRUPT) — разрешаются. В ответ на прерывание процессор автоматически сохраняет в стеке регистр признаков (и, следовательно, бит 4 — «прерывание»). По команде RTI в конце процедуры обработки прерывания восстанавливается старое содержимое регистра признаков и разрешение прерываний возобновляется, если процедура обработки не изменила данные, находящиеся в это время в стеке.

Прерывания от PIA. В большинстве случаев система прерываний микропроцессора Motorola 6800 получает на вход запросов на прерывание сигналы от адаптера интерфейса периферийных устройств (PIA). Каждое устройство PIA обладает двумя выходами прерываний (от свободных коллекторов с низкими действующими значениями): \overline{IRQA} и \overline{IRQB} (по одному на каждой из сторон PIA). Эти выходы от свободных коллекторов можно соединить по схеме монтажного ИЛИ для формирования входа \overline{IRQ} процессора.

В PIA включено большое число схем, необходимых для организации прерываний. Разряд 7 регистра управления является фиксатором запросов на прерывания.

вание для сигналов на линии управления 1. Центральный процессор может проверить этот разряд, чтобы определить, вызвано ли прерывание этим источником. Аналогично, разряд 6 регистра управления—фиксатор запросов на прерывание для сигналов на линии управления 2 (если эта линия работает как входная). PIA автоматически сбрасывает фиксаторы запросов на прерывание, когда процессор читает данные из соответствующего регистра данных.

Заметим, что при записи данных в регистр данных биты прерываний не сбрасываются; не сбрасываются они и при записи в регистр управления и при чтении из него. Поэтому программист должен вводить в процедуру обработки прерывания дополнительную команду, которая читает содержимое регистра данных (например, LOAD), только для того, чтобы сбросить биты прерываний.

Содержимое регистра управления PIA определяет способ работы с сигналами прерываний. Используются следующие разряды регистра:

разряд 0; если этот разряд равен 1, то разрешена подача на выход PIA сигнала запроса на прерывание, полученного с линии управления 1; если этот разряд равен 0, то подача сигнала запрещена;

разряд 1; если этот разряд равен 1, то фиксатор прерываний (разряд 7) будет устанавливаться по переходу «низкое значение—высокое значение» на линии управления 1; если этот разряд равен 0, то фиксатор будет устанавливаться по переходу «высокое значение—низкое значение» на этой линии.

Если линия управления 2 работает как ввод (управляющий разряд 5 равен нулю), то используются также следующие разряды:

разряд 3; если этот разряд равен 1, то разрешена подача на выход PIA сигнала запроса на прерывание, полученного с линии управления 2; если этот разряд равен нулю, то подача сигнала запрещена.

разряд 4; если этот разряд равен 1, то фиксатор прерываний (разряд 6) будет устанавливаться по переходу «низкое значение—высокое значение» на линии управления 2; если этот разряд равен нулю, то по переходу «высокое значение—низкое значение».

Таким образом, с помощью регистров управления PIA можно разрешать и запрещать прерывания от каждого источника прерывания в отдельности. Прерывания от каждого PIA система должна разрешать отдельно, так как по сигналу «сброс» запрещаются все прерывания.

В типичной процедуре настройки системы прерываний сначала разрешаются необходимые прерывания в PIA, а затем разрешается работа системы прерываний процессора. Предположим, что сигналом RESET сброшены все регистры управления. Тогда можно привести следующие примеры программ, настраивающих PIA:

- * Настроить PIA на ввод и на прерывание по переходу «высокий потенциал—низкий потенциал» на линиях CA1 и CB1

```
CLR PIADRA      Все линии — на вход
CLR PIADRB
LDAA  # % 00000101  Разрешить прерывания PIA
STAA PIACRA
STAA PIACRB
```

Разряды 2 регистров управления установлены в единицу, чтобы адресовать регистры данных; разряд 0 установлен в единицу, чтобы разрешить прерывания.

- * Настроить PIA на вывод и прерывания по переходу «низкий потенциал — высокий потенциал» на линиях CA1 и CB1

```
LDAA  # $ FF      Все линии — на выход
STAA PIADRA
STAA PIADRB
LDAA  # % 00000111  Разрешить прерывания PIA
STAA PIACRA
STAA PIACRB
```

Разряды 1 регистров управления установлены в единицу, чтобы переход от низкого потенциала к высокому на линиях CA1 и CB1 вызывал сигнал прерывания.

* Настроить PIA на ввод и прерывания по переходу «высокий потенциал — низкий потенциал» на любой из линий CA1 или CA2 и любой из линий CB1 или CB2

CLR PIADRA Все линии — на вход

CLR PIADRB

LDAA # %00001101 Разрешить прерывания PIA

STAA PIACRA

STAA PIACRB

Разряды 5 регистров управления установлены в нуль, чтобы линии управления CA2 и CB2 работали как входные; разряды 3 — в единицу, чтобы разрешить прерывания; так как разряд 4 также установлен в единицу, то переход от высокого потенциала к низкому на линии управления установит в единицу разряд 6 регистра управления соответствующей стороны и возбудит сигнал прерывания.

* Настроить PIA на вывод и прерывания по переходу «низкий

потенциал — высокий потенциал» на любой из линий CA1 или CA2 и любой из линий CB1 или CB2

LDAA # \$ FF Все линии — на выход

STAA PIADRA

STAA PIADRB

LDAA # %00011111 Разрешить прерывания PIA

STAA PIACRA

STAA PIACRB

Разряды 4 регистров управления установлены в единицу, чтобы переходом, вызывающим сигнал прерывания, стал переход от низкого потенциала к высокому на линии управления 2.

Полная программа (начало которой расположено по адресу, на который передается управление по сигналу сброса) может иметь следующую структуру:

ORG RSTADD

LD \$ LASTM Расположить стек в конце поля памяти

* Настроить все PIA

CLR Разрешить прерывания

По адресу, определяющему адреса перехода по сигналу сброса, будет храниться адрес RSTADD, т. е.

ORG \$ FFFE

FDB RSTADD

В процедуре обработки прерывания нет необходимости сохранять и восстанавливать содержимое регистров и значений признаков процессора. В ответ на прерывание все регистры автоматически сохраняются в стеке, а по команде RTL в конце процедуры восстанавливается исходное состояние, включая и восстановление значения признака прерывания.

Системы прерываний с полиномом. В простейших системах прерываний микропроцессора Motorola 6800 для определения источника прерывания применяется полинг. В таких системах в ячейках памяти, соответствующих вектору прерывания, должен храниться адрес начала процедуры полинга. Применение неполного декодирования адреса затрудняет расширение системы, так как при прерывании управление всегда передается по адресу, соответствующему вектору; обычно эти адреса относятся к ПЗУ. Подпрограммы полинга и обработки прерываний можно разместить в памяти где угодно.

Процедура полинга проверяет значения признаков прерывания, находящихся в регистрах управления PIA:

LDAA PIACR

BMI SERVE

Прочитать содержимое регистра PIA

Если седьмой разряд регистра PIA равен 1, то перейти к программе обработки прерывания

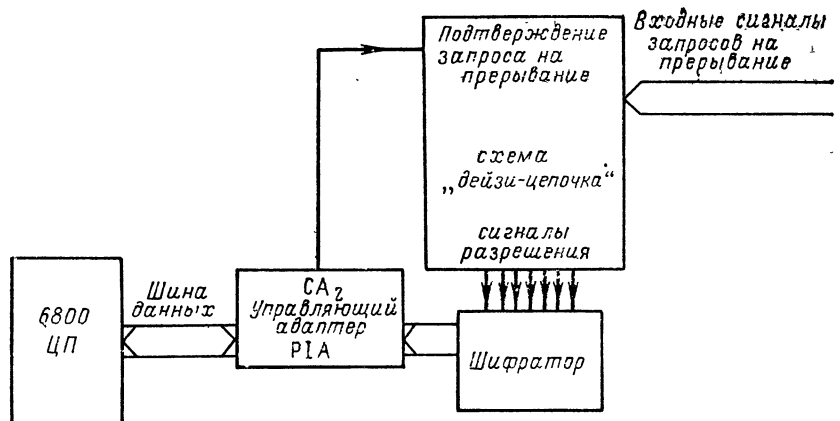


Рис. 9.33. Система прерывания процессора Motorola 6800, организованная по принципу «дейзи-цепочки» [сигнал «подтверждение запроса на прерывание» подается на «дейзи-цепочку» по линии CA₂ (см. рис. 9.11)]. Сигналы разрешения от «дейзи-цепочки» поступают на шифратор, выход которого доступен PIA (для чтения данных в регистр данных PIA)]

Применение именно седьмого разряда регистра PIA в качестве фиксатора сигнала прерывания удобно потому, что этот разряд можно проверять как знаковый. С помощью команд логического умножения (AND) и сдвига (SHIFT) ЦП может проверить значение шестого разряда.

К сожалению, проверка признаков прерываний PIA неудобна, так как адреса регистров управления различных PIA обычно не расположены в адресном пространстве регулярно, поэтому обычно в процедуре полинга для проверки каждого PIA применяется отдельная группа команд. Вследствие этого, когда число входов запросов на прерывание велико, программы полинга оказываются громоздкими, медленными и с трудом поддающимися расширению. Метод полинга, как правило, применим только для малых систем.

Векторные системы прерываний. Применение метода «дейзи-цепочки» (см. рис. 9.11) — один из способов избежать недостатков, характерных для полинга, в системах на базе микропроцессора Motorola 6800¹. Так как этот процессор не вырабатывает сигнал «подтверждение запроса на прерывание», то в программе необходимо создать его искусственно, специально для «дейзи-цепочки». Программа должна также обрабатывать выход от «дейзи-процесса» и использовать его в качестве вектора прерывания. На рис. 9.33 показана система, в которой дополнительный адаптер PIA применяется как порт вывода для «дейзи-цепочки» и как порт ввода для вектора-результата. Сигнал «подтверждение запроса на прерывание» представляет собой фиксируемый сигнал управления от PIA. В программе вектор прерывания используется для индексации таблицы переходов, которая содержит адреса подпрограмм обработки прерываний.

Сама программа выглядет следующим образом:

* Настроить PIA на управление «дейзи-цепочкой»

LDAA #00110100

STAA CPIACR

В результате этих команд выход CA₂ становится фиксатором со значением ноль.

¹См., например, J. D. Logan and P. S. Kreager. Using a Microprocessor: a Real-Life Application. Part 1 — Hardware. Computer Design, September 1975, p. 69—77.

- Процедура обработки прерывания
 ORG INTADD
 LDAA #%00111100 «Подтверждение запроса на прерывание» (INTA)=1
 для «дейзи-цепочки»
 STAA CPIACR
 LDAB CPIADR Получить вектор от «дейзи-цепочки»
 LDAA #%00110100 INTA = 0 для «дейзи-цепочки»
 STAA CPIACR
- Использовать вектор для индексации таблицы переходов
 STAB TPTR + 1 Вектор — младшие восемь разрядов значения сдвига
 для индексной адресации входа в программу обработки прерывания
- LDAB #TBLMS Подготовить старшие разряды адреса перехода
 STAB TPTR
 LDX TPTR
 JMP TBLLS, X Переход на обработку прерывания

TBLMS и TBLLS — соответственно восемь старших и восемь младших разрядов адреса начала таблицы переходов.

Векторные системы прерываний должны узнавать появляющийся на адресной шине фиксированный адрес, связанный с прерыванием, так как нет иного способа определить, является ли данный цикл циклом прерывания. Декодирующая схема, распознающая такой адрес, должна затем передать процессору один из векторов прерываний. Такая процедура предполагает помещение на шину данных (с соблюдением требований синхронизации) как старших, так и младших разрядов вектора. Все необходимые действия может выполнить контроллер приоритетных прерываний (Priority Interrupt Controller) Motorola 6828. Контроллер 6828: а) распознает вектор прерывания; б) изменяет восемь младших разрядов этого вектора в соответствии с тем прерыванием, которое имеет самый высокий приоритет (из возбужденных к этому моменту); в) подавляет все или некоторые прерывания (в соответствии с содержимым программно-управляемого регистра-маски). Каждое устройство 6828 обеспечивает обработку до восьми векторов прерывания; можно объединить восемь таких устройств.

Приоритетные системы. В контроллере 6828 имеются схемы, обеспечивающие работу с приоритетами. Такие схемы можно включить и в другие векторные системы. При использовании «дейзи-цепочки» приоритет каждого входа запросов на прерывание определяется его положением в цепочке.

Простые системы с полингом позволяют реализовать любой из следующих способов назначения приоритетов:

- 1) приоритеты определяются порядком, в котором процедура опрашивает признаки прерываний;
- 2) перед сбросом признака «прерывание» разрешаются или запрещаются прерывания от конкретных PIA. Однако реализация такой процедуры может потребовать много времени, так как каждый адаптер настраивается индивидуально;
- 3) используется внешний регистр текущего приоритета и компаратор; прерывания с приоритетом, меньшим текущего или равным ему, запрещаются,

Обработка микропроцессором Motorola 6800 типичных прерываний

Прерывания от внешнего переключателя. Такие прерывания используются для организации приостановки выполнения программы в контрольных точках и перехода на управление с лицевой панели. Если переключатель соединен с линией CA1 адаптера PIA, то программа обработки подобного прерывания может выглядеть следующим образом [здесь предполагается, что нормальное положение переключателя — «разомкнуто» (логическая единица)]:

```

* Настроить PIA
  LDA #00000101
  STAA PIACRA      Разрешить прерывания

* Программа, выполняемая в контрольной точке

  ORG INTADD
  LDA PIADRA      Сбросить признак «прерывания» PIA
  TSX              Указатель стека = базовый адрес
                  поля для автоматического сохранения
                  регистров

  LDAB #7          Число 7 (число печатаемых байтов) по-
                  местить в регистр B

* PRREG
  JSR LDA X
  INX PRINT        Напечатать содержимое регистра A
  DECB
  BNE PRREG
  RTI              Возврат в главную программу

```

Система прерываний микропроцессора Motorola 6800 превосходно приспособлена для этой простой задачи, так как по прерыванию содержимое регистров автоматически помещается в стек, а по команде RTI восстанавливается. Подпрограмма PRINT должна выполнить необходимое преобразование кодов, форматизацию и передачу данных (пересылку содержимого аккумулятора системному печатающему устройству). Следует обратить внимание на дополнительную команду LDA PIADRA, сбрасывающую признак прерывания.

Подпрограмма обработки подобного прерывания может разрешить и другие прерывания, отличные от прерывания рассматриваемого вида. Требуемая для этого последовательность команд следующая:

```

* Проверить, есть ли прерывание от переключателя
  LDA PIACRA
  BMI BRKPT

  Проверить, есть ли другие источники прерываний

* Запретить прерывания от переключателя
BRKPT ANDA #1111110
      STAA PIACRA      Запретить прерывания PIA
      CLI              Разрешить прочие прерывания

* Возобновить разрешение прерываний от переключателя
  LDA PIACRA
  GRAA #00000001
  STAA PIACRA      Разрешить прерывания PIA
  RTI

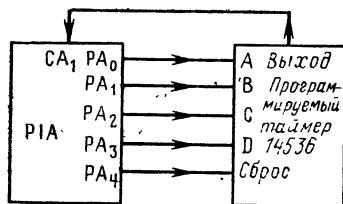
```

Середина программы обработки прерываний от переключателя останется такой же, как ранее.

Прерывания по таймеру. Обычным источником прерываний по таймеру служит программируемый таймер Motorola 14536—24-каскадный двоичный счетчик. С помощью 4-разрядного кода можно определить, какое число каскадов (из последних 16) будет использоваться. Устройство имеет также входы для установки и очистки счетчика, для подавления сигналов синхронизации и для пропуска первых восьми каскадов. Программируемый таймер Motorola 6840 — аналогичное устройство, разработанное специально для использования совместно с микропроцессором 6800.

Таймер 14536 выполняет разнообразные функции по отсчету времени. Эти функции определяются входными сигналами. На рис. 9.34 показана структурная схема, в которой таймер управляется PIA. На рисунке разряды 0—3 выхода

Рис. 9.34. Применение PIA для управления программируемым таймером 14536 (по другим линиям, данных и управления, процессор управляет другими входами таймера)



PIA определяют число каскадов, которые использует таймер, а разряд 4 управляет входом «сброс» таймера. Переход на выходной линии таймера (от низкого потенциала к высокому) идентифицирует конец временного интервала.

Приведем программу обработки прерывания:

```
*
* Настроить PIA
LDAA  #%00000110
STAA  PIACRA
```

Разряд 1 установлен в единицу, поэтому переход от низкого значения потенциала к высокому на линии CA1 вызовет прерывание.

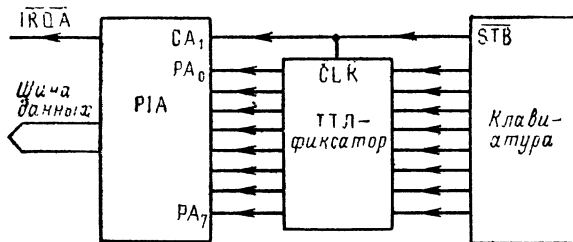
```
* Сбросить таймер и разрешить прерывания
LDAA  INTRVL
ORAA  #%00010000
STAA  PIADRA      Сбросить таймер
ANDA  #%00001111
STAA  PIADRA      Прекратить подачу сигнала «сброс»
LDAA  #%00000111
STAA  PIACRA      Разрешить прерывания от таймера
```

По адресу INTRVL находится шестнадцатиричная цифра, определяющая длительность интервала времени. Следует обратить внимание на то, что в подпрограмме обработки прерывания необходимо произвести чтение из порта таймера, чтобы сбросить признак прерывания.

Прерывания от клавишного пульта. Строб-сигнал (с низким действующим значением) от клавиатуры с кодированием может играть роль сигнала запроса на прерывание (рис. 9.35). Приведенная ниже подпрограмма обработки прерывания заполняет буферный массив, игнорируя начальные пробелы кода ASCII (шестнадцатиричный код — 20) до тех пор, пока не обнаружит символ ASCII «возврат каретки» (код 0D). Главная программа ждет завершения приема строки. Блок-схема программы приведена на рис. 9.36.

```
* Главная программа
*
* Начальная настройка PIA (на ввод)
```

Рис. 9.35. Прерывание от клавиатуры, снабженной шифратором (данные фиксируются по строб-сигналу (с низким действующим значением) от клавиатуры, подаваемому на синхровход CLK ТТЛ-фиксатора (напомним, что у PIA нет фиксатора входа). Одновременно строб вызывает прерывание процессора от PIA. При чтении входных данных признак «прерывание» PIA сбрасывается]



```

*
ORG RSTADD
CLR PIACRA
CLR PIADRA
LDAA #%%00000101
STAA PIACRA
*
* Установить указатель буфера строки и признаки в исходное
* состояние
CLR NONSP
CLR LENGTH
CLR EFLAG
LDX #START
STX BPTR
*
* Разрешить прерывания и ждать конца строки
*
CLI
WAITE LDAA EFLAG
BEQ WAITE
*
* Продолжение программы
*
* Итак, главная программа настроила PIA, установила в исход-
* ное состояние указатель, счетчик и признаки, а затем разре-
* шила прерывания.
*
* Процедура обработки прерывания
*
ORG INTADD
LDAA PIADRA
*
* Игнорировать начальные пробелы
LDAB NONSP
BNE NOCHK
*
CMPA #SPACE
BEQ ENDINT
INC NONSP
*
* Если обнаружен код «возврат каретки», то установить
* маркер «конец строки» и запретить прерывания
*
NOCHK CMPA #CR
BNE NOCR
INC EFLAG
PULB
*
ORAB #%%00010000
PSHB
*
* Запомнить данные в буфере строки
*
LDX BPTR
STAA X

```

Все линии — на ввод

Разрешить прерывания. PIA

Символы, отличные от пробела, еще не обнаружены

Длина строки = 0

«Возврат каретки» еще не обнаружен

Указатель=начало буфера

Разрешить прерывания

Признак конца строки = 0?

Да, ожидать

Итак, главная программа настроила PIA, установила в исходное состояние указатель, счетчик и признаки, а затем разрешила прерывания.

Процедура обработки прерывания

Считать данные с клавиатуры

Игнорировать начальные пробелы

Поступал ли хотя бы один символ, отличный от пробела?

Да, занести в массив без проверки на пробел

Нет, проверить, является ли символ пробелом

Да, игнорировать начальный пробел

Нет, установить признак NONSP

Если обнаружен код «возврат каретки», то установить

маркер «конец строки» и запретить прерывания

Данные = символ «возврат каретки»?

Нет, запомнить символ в буфере строки

Да, установить признак «конец строки»

Выбрать регистр признаков результата из стека

Запретить прерывания

Вернуть измененный регистр в стек

Запомнить данные в буфере строки

Выбрать из памяти указатель буфера

Записать данные в буфер строки

NOCHK

NOCR

INX		Указатель = указатель + 1
STX	BPTR	
INC	LENGTH	Длина строки = длина строки + 1
ENDINT	RTI	

Процедура обработки прерывания устанавливает бит «прерывания» (разряд 4 регистра признаков результата) в стеке, когда обнаруживает символ «возврат каретки». Это дает возможность программе переместить введенную строку из буфера до заполнения буфера новыми данными. В противном случае команда RTI автоматически восстановила бы старое значение бита прерывания и тем самым возобновила бы разрешение прерываний.

Прерывания от УАПП. Асинхронный адаптер интерфейса связи (ААИС) Motorola 6850 вырабатывает сигналы прерывания двух типов: от приемника и от передатчика. Прерывание от передатчика разрешено только тогда, когда разряд 6 регистра управления ААИС равен нулю, а разряд 5 — единице. В этом случае прерывание вызывает сигнал «регистр данных передатчика пуст» (TDRE — TRANSMITTER DATA REGISTER EMPTY). Признак «прерывание» сбрасывается при записи данных в регистр данных передатчика.

Прерывание от приемника разрешено только тогда, когда разряд 7 регистра управления ААИС равен единице. В этом случае прерывание вызывает сигнал «регистр данных приемника заполнен» (RDRF — RECEIVE DATA REGISTER FULL). Центральный процессор может определить источник прерывания, только проверив регистр состояния ААИС:

RDRF = разряд 0

TDRE = разряд 1

IRQ = разряд 7

При чтении данных сбрасывается признак прерывания от RDRF.

Для телетайпа (с 7 разрядными данными, разрядом проверки на нечетность и двумя стоп-разрядами) программа настройки ААИС на режим генерации сиг-

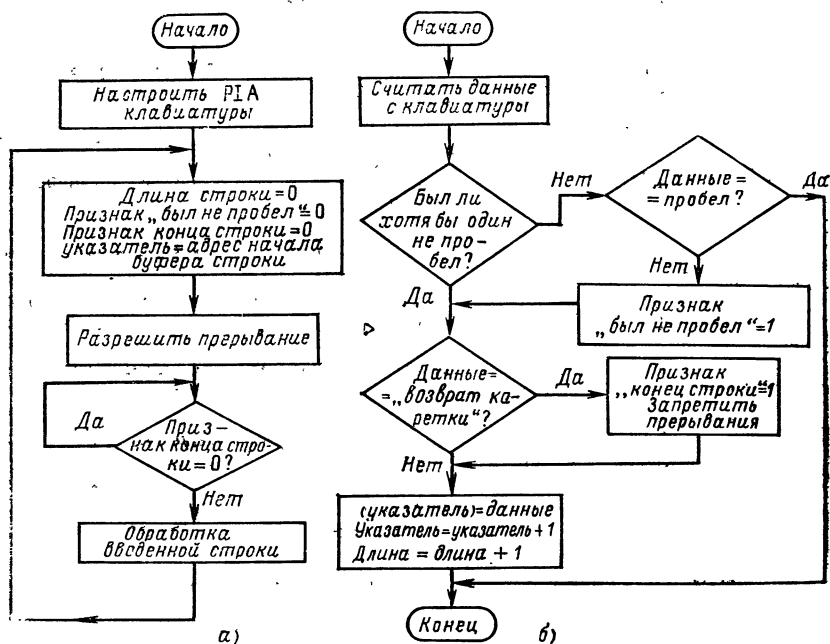


Рис 9.36. Блок-схема программы обработки прерывания от клавиатуры: а — главная программа; б — подпрограмма обработки прерывания

налов прерывания как от приемника, так и от передатчика имеет следующий вид:

```

*
* Сброс ААИС
*   LDAA  #%00000011
*   STAA  ACIACR      Сброс (очистка ААИС)
*
* Настроить ААИС
*   LDAA  #%10100100
*   STAA  ACIACR      Настроить ААИС

```

Асинхронный адаптер интерфейса связи очищается путем посылки специально подготовленных данных (master reset), так как ААИС не имеет входа «сброс».

Управляющие разряды принимают следующие значения:

разряд 7 = 1 — разрешение прерывания от приемника;
 разряд 6 = 0, разряд 5 = 1 — разрешение прерывания от передатчика;
 разряд 4 = 0, разряд 3 = 0, разряд 2 = 1 — установка формата данных;
 разряд 1 = 0, разряд 0 = 0 — коэффициент умножения оборотной частоты генератора тактовых импульсов равен 1.

Приведенная ниже программа обработки прерываний обслуживает прерывания, поступающие как от приемника, так и от передатчика. Алгоритм обработки следующий (рис. 9.37):

1. По содержимому регистра состояния определить источник прерывания (разряд 0 равен единице в случае прерывания от приемника; разряд 1 равен единице в случае прерывания от передатчика).

2. Если прерывание поступило от приемника, то поместить данные в буфер приемника. По адресу RPTR хранится адрес следующей пустой ячейки, по адресу RCTR — число слов в буфере.

3. Если прерывание поступило от передатчика, то проверить, имеются ли данные в буфере передатчика (указателями буфера будут служить TPTR и TCTR). Если данные есть, то послать слово данных ААИС; если данных нет, то передать символ-разделитель (шестнадцатиричный код FF).

4. Возвратить управление главной программе.

```

*
* Программа обработки прерываний от ААИС
*

```

ORG AINT

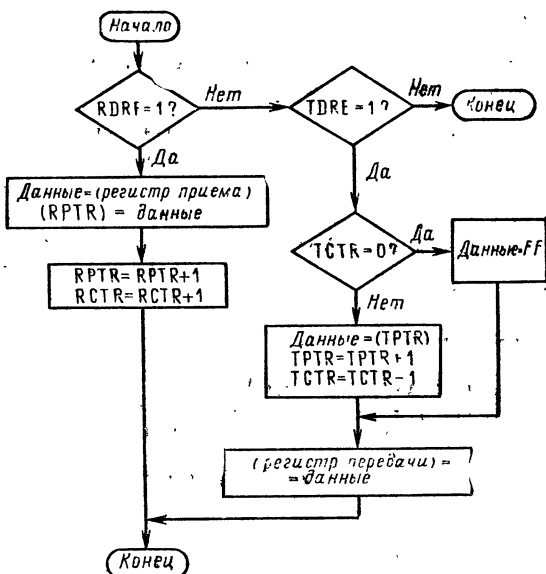


Рис. 9.37. Блок-схема программы обработки прерывания от ААИС

* По регистру "состояния" определить, приемником или передатчиком вызвано прерывание

LDAA ACIADR

Прочитать содержимое регистра состояния ААИС

* RORA

Признак «от приемника» равен 1?

BCS RINT

Да, прерывание от приемника

RORA

Признак «от передатчика» равен 1?

BCC DONE

Нет, обработка прерывания завершена

* Обработка прерывания от передатчика

* Определить, доступны ли данные для передачи

LDAA # \$ FF

Подготовить символ-разделитель

TST TCTR

Данные для передачи доступны?

BEQ TDATA

Нет, передать символ-разделитель

* Извлечь данные из буфера передатчика

LDX TPTR

Выбрать указатель буфера передатчика

LDAA X

Выбрать данные

INX

STX TPTR

Запомнить новое значение указателя

DEC TCTR

Установить новое значение счетчика буфера

* Передать данные

TDATA STAA ACIADR Передать данные ААИС

RTI

* Обработка прерывания от приемника

* Поместить данные в буфер приемника

RINT LDAA ACIADR

Получить данные от ААИС

LDX RPTR

Выбрать указатель буфера приемника

STAA X

Данные запомнить в буфере приемника

INX

STX RPTR

Запомнить новое значение указателя

INC RCTR

Установить новое значение счетчика буфера

* Возвратить управление главной программе

DONE RTI

Программа может также послать символ «уровень обрыва», установив в единичный разряд 6 регистра управления ААИС:

LDAA # % 11100100

STAA ACIADR

Уровень обрыва

LDAA # % 10100100

STAA ACIADR

Восстановить прерывание от передатчика

Не следует забывать, что содержимое регистра управления ААИС прочитать нельзя — он доступен только для записи.

Прерывания от преобразователей. Аналого-цифровой преобразователь AD7570, описанный в гл. 8, можно эксплуатировать в режиме прерывания. Для этого достаточно разрешить поступление сигнала с линии BUSY на вход запросов на прерывание. Переход от низкого потенциала к высокому на этой линии указывает, что процесс преобразования завершился. В этом случае нет необходимости вводить в программу обработки данных от преобразователя команды, реализующие временные задержки.

9.5. ПРЯМОЙ ДОСТУП К ПАМЯТИ

Прямой доступом к памяти (ПДП) называется такой метод ввода вывода, при котором вся процедура ввода-вывода полностью осуществляется специальными аппаратными средствами. Центральный процессор передает управление шинами контроллеру ПДП, который производит обмен данными непосредственно между памятью и подсистемой ввода-вывода. Обычно контроллер ПДП передает целый блок данных. Контроллер ПДП обеспечивает адресацию памяти и вырабатывает сигналы управления памятью, обновляет указатели адреса, подсчитывает число переданных слов и сигнализирует о завершении операции ввода-вывода.

Преимущество метода прямого доступа к памяти — скорость обмена данными. Обмен может происходить с любой скоростью, ограниченной только временем доступа к памяти. Процессор избавлен от необходимости выбирать из памяти и декодировать команды ввода-вывода, обновлять адресные указатели и счетчики, проверять, завершилась ли операция ввода-вывода. Эти функции выполняет контроллер, причем со скоростью, характерной для аппаратных средств.

Как всегда, когда аппаратные средства заменяют программные, возникает вопрос о выборе между высокой скоростью работы аппаратуры и такими преимуществами программных средств, как меньшая стоимость и большая гибкость системы. Обычно контроллеры ПДП — достаточно сложные устройства; даже простые контроллеры содержат 50—100 кристаллов (хотя число кристаллов можно сократить, применяя специальные БИС). Вообще, чем быстрее работают системы ПДП, тем больший объем оборудования они используют и тем выше их стоимость. По существу, контроллер ПДП представляет собой специализированный процессор, так как передача данных контроллером во многом аналогична операциям, осуществляемым обычным процессором.

Чтобы реализовать метод ПДП в системах на основе обычных микропроцессоров, необходим значительный объем аппаратуры. Можно использовать различные методы, но все они должны так или иначе решать следующие основные задачи:

- 1) подготовка процессора к выполнению операции ПДП;
- 2) управление шинами таким способом, который не помешает обычной работе процессора и не приведет к конфликтам из-за доступа к шинам;
- 3) определение длины и местонахождения передаваемых данных;
- 4) индикация завершения операции.

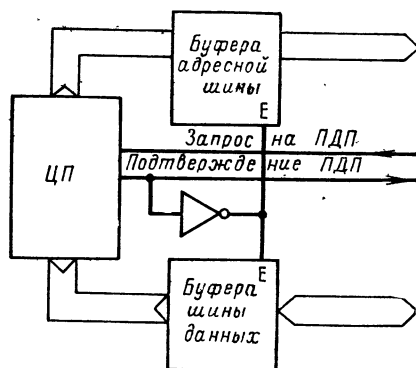
Как правило, процессор не должен делать почти ничего, кроме следующего: приостановить работу и сообщить, что ему не нужны шины. Контроллер ПДП делает все остальное сам.

Простой метод ПДП состоит в использовании тех циклов, в которых процессор не обменивается данными с памятью. В такие промежутки времени контроллер ПДП может пользоваться шинами, не информируя об этом процессор. Такой метод называется «захватом цикла» (*cycle stealing*). Основная проблема здесь — идентифицировать такие циклы и избежать временного перекрытия обмена ПДП с операциями про-

Рис. 9.38. Сигналы управления для простой реализации метода ПДП

цессора. Некоторые процессоры вырабатывают специальный сигнал (таков, например, сигнал VMA микропроцессора Motorola 6800), указывающий, используется ли в данном цикле память. У других процессоров для опознавания «внутренних» циклов требуется декодирование. Например, микропроцессор Intel 8080 никогда не обменивается памятью в состояниях T_4 и T_5 ; чтобы опознать эти состояния, требуется специальная схема. Применение метода захвата цикла не снижает скорости выполнения процессором операций, но может потребовать сложных схем синхронизации; метод позволяет осуществлять только случайные, нерегулярные передачи.

Более распространенный метод состоит в том, чтобы принудительно передать управление шинами контроллеру ПДП для выполнения операций обмена данными. «Запрос на ПДП» — это сигнал процессору; «подтверждение ПДП» — ответный сигнал ЦП. По сигналу «подтверждение ПДП» тристабильные шинные буфера запираются (процессор отсоединяется от шин) и начинает работать контроллер ПДП. На рис. 9.38 показаны необходимые сигналы управления. Широко распространенные микропроцессоры (такие как Intel 8080, Motorola 6800, Signetics 2650 и General Instrument CP-1600) имеют средства для ПДП именно этого типа.



Опознавание запроса на ПДП и синхронизация

Обработка сигнала «запрос на ПДП» аналогична обработке сигнала «запрос на прерывание». Другими словами, процессор проверяет наличие сигнала запроса в определенные моменты времени, когда выполняет обычные операции. Сигнал должен быть синхронизирован с сигналами тактового генератора. Микропроцессор может завершить выполнение всей текущей команды или ее части. Микропроцессор Intel 8080 при поступлении сигнала запроса на ПДП завершает текущий машинный цикл, но предоставляет шины для ПДП сразу же, когда они станут ненужными для его работы (тем не менее процессор выполняет действия, завершающие цикл, — такие как декодирование или даже выполнение команды, не использующей шины). У микропроцессора Motorola 6800 два режима управления. В первом режиме, называемом TSC (tristate control), процессор продолжает работу только до конца текущего тактового сигнала, а во втором режиме — HALT («останов») — до завершения выполнения команды. Сигнал «запрос на ПДП» должен быть сброшен во время последнего цикла команды, предшествующей ПДП. Контроллер ПДП должен обеспечивать правильную синхронизацию сигналов.

Для определения начального адреса поля данных и его длины (числа передач ПДП) используются различные методы. Один из простых методов состоит в том, что контроллер выбирает значения указанных параметров из регистров. Например, в процессоре RCA CDP1802 содержится одного из внутренних регистров используется как адрес при операции ПДП; содержимое регистра автоматически увеличивается на единицу в конце каждой пересылки. Описанный метод позволяет упростить внешнюю аппаратуру, но требует инициализации регистра и применения его только для этой цели. Чаще процессоры управляют ПДП через обычные каналы ввода-вывода. Например, процессор может послать контроллеру ПДП начальный адрес, число слов и сигнал активации. Контроллер ПДП должен иметь соответствующие порты и счетчики.

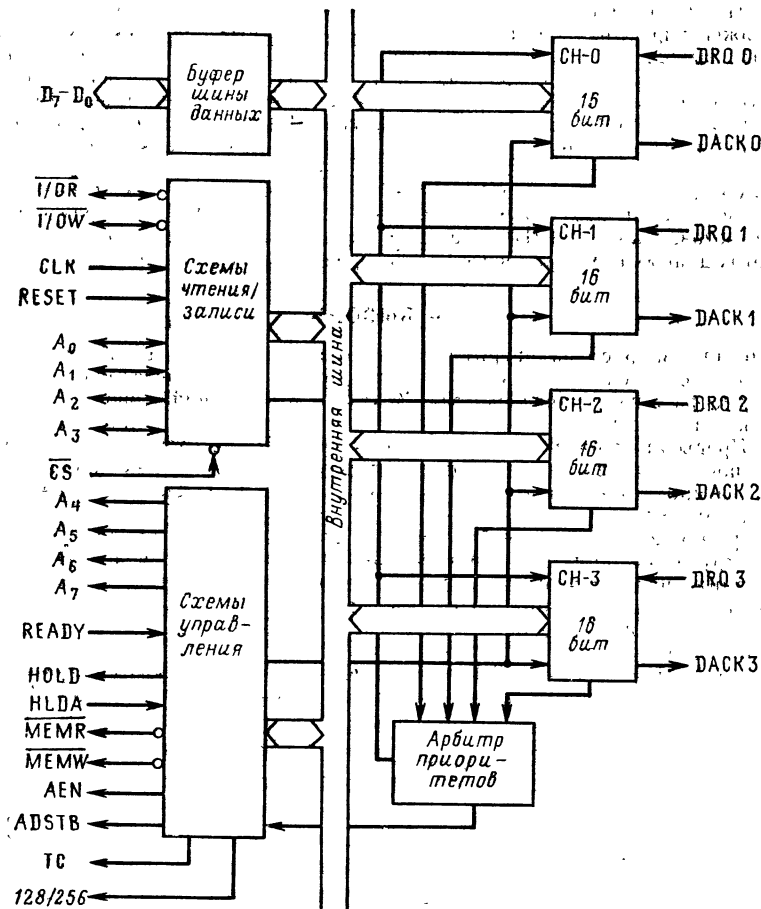


Рис. 9.39. Структурная схема программируемого контроллера ПДП Intel 8257

Как и сложные системы прерываний, система ПДП может иметь дополнительные возможности. Так, например, возможно наличие нескольких каналов ПДП, присоединяемых с использованием какой-либо схемы приоритетного управления; каналы могут прерывать работу друг друга. Возможно наличие средств разрешения/запрещения работы каналов (одного конкретного канала или нескольких) с помощью приоритетной системы. Подобные возможности обеспечиваются внешней аппаратурой или специальными контроллерами. Программируемый контроллер ПДП Intel 8257 (рис. 9.39) имеет четыре канала ПДП, приоритетную схему, схему подавления работы каналов, счетчики и другие логические схемы.

Программирование систем, основанных на методе ПДП

В целом программирование систем, основанных на применении метода ПДП, оказывается проще, чем программирование систем, управляемых по сигналам прерывания, так как моменты обмена данными полностью определяются аппаратурой. Память в системах, основанных на ПДП, работает подобно УВВ. Процессор должен определить, когда готов новый блок входных данных и когда можно передать выходные данные. Для организации обмена данными при использовании ПДП программист должен описать в программе действия, аналогичные тем, которые он задает для любой операции ввода-вывода. Но, в отличие от использования системы прерываний, использование ПДП не ухудшает структуру программы, так как никакая передача управления в программе не обусловлена внешними событиями. Однако для реализации систем, использующих ПДП, необходимо заботиться о точной синхронизации, от программиста требуется хорошее понимание процесса обмена данными при ПДП. Опыт программирования систем, основанных на ПДП, показывает, что в программах, с хорошей структурой поток данных играет не менее важную роль, чем поток команд.

9.6. ВЫВОДЫ

Сигналы запроса на прерывания — входные сигналы, вызывающие изменение обычной последовательности работы процессора и заставляющие процессор реагировать на внешние события. Механизм прерываний позволяет проверять признаки «данные готовы» и «УВВ готово» аппаратным, а не программным способом. Прерывания оказываются полезными также для организации временной приостановки выполнения программы в контрольных точках, при необходимости реагировать на сигналы тревоги и на сигналы падения напряжения питания, а также для работы в реальном масштабе времени. К основным недостаткам прерываний относится необходимость в дополнительной аппаратуре и ухудшение структуры программ. Система прерываний может определить источник прерывания либо путем оброса зафиксированных сигналов запроса на прерывание (полинг), либо по коду, идентифицирующему источник, полученному от устройства (векторное прерывание). Сложные системы прерываний могут иметь несколько входов запросов на прерывание, а также систему приоритетов. Системы прерываний на базе МП Intel 8080 используют внешнюю аппаратуру, которая по сигналу «подтверждение запроса на прерывание» помещает на шину данных команду RST или CALL. В системах прерываний МП Motorola 6800 используются специальные адреса памяти, по которым содержатся векторы; процедура обработки прерывания может отличать друг от друга различные источники с помощью полинга адаптеров PIA. Метод прямого доступа к памяти (ПДП) — самый быстрый метод ввода-вывода; для его реализации применяется специальный контроллер, осуществляющий передачу целого блока данных. Этот метод позволяет достигнуть высокой скорости за счет усложнения аппаратуры.

¹B. Shneiderman and P. Scheuermann. Structured Data Structures. Communications of the ACM, vol. 17, № 10, October 1974, p. 566—574.

СПИСОК ЛИТЕРАТУРЫ

К главе 2

- ALLISON, D. R., "A Design Philosophy for Microcomputer Architectures," *Computer*, Vol. 10, No. 2, February 1977, pp. 35-41.
- BURROUGHS CORPORATION, *Digital Computer Principles*, McGraw-Hill, New York, 1969.
- CHU, Y., *Computer Organization and Microprogramming*, Prentice-Hall, Englewood Cliffs, N.J., 1972.
- CUSHMAN, R. H., "The Intel 8080: First of the Second-Generation Microprocessors," *EDN*, Vol. 19, No. 9, May 5, 1974, pp. 30-36.
- ECKHOUSE, R. H., JR., *Minicomputer Systems*, Prentice-Hall, Englewood Cliffs, N.J., 1975.
- FINKEL, J., *Computer/Aided Experimentation*, Wiley-Interscience, New York, 1975.
- GEAR, C. W., *Computer Organization and Programming*, McGraw-Hill, New York, 1974.
- HUSSON, S. S., *Microprogramming: Principles and Practices*, Prentice-Hall, Englewood Cliffs, N.J., 1970.
- "Intel 8080 Microcomputer Systems User's Manual," Intel Corporation, Santa Clara, Ca., July 1975.
- KORN, G. A., *Minicomputers for Engineers and Scientists*, McGraw-Hill, New York, 1973.
- MAZUR, T., "Microprocessor Basics. Part 4: The Motorola 6800," *Electronic Design*, Vol. 24, No. 15, July 19, 1976, pp. 66-77.
- McKENZIE, K. and A. J. NICHOLS, "Build a Compact Microcomputer," *Electronic Design*, Vol. 24, No. 10, May 10, 1976, pp. 84-92.
- PEUTO, B. L. and L. J. SHUSTEK, "Current Issues in the Architecture of Microprocessors," *Computer*, Vol. 10, No. 2, February 1977, pp. 20-25.
- TORRERO, E. A., "Focus on Microprocessors," *Electronic Design*, Vol. 22, No. 18, September 1, 1974, pp. 52-69.

К главе 3

- BOND, J., "Designer's Guide to: Software for the Hardware Designer—Part I," *EDN*, Vol. 19, No. 11, June 5, 1974, pp. 40-44.
- BOND, J., "Designer's Guide to Software for the Hardware Designer—Part II," *EDN*, Vol. 19, No. 15, August 5, 1974, pp. 51-56.
- CUSHMAN, R. H., "The Intel 8080," *EDN*, Vol. 19, No. 9, May 5, 1974, pp. 30-36.
- CUSHMAN, R. H., "Microprocessor Instruction Sets," *EDN*, Vol. 20, No. 6, March 20, 1975, pp. 35-41.
- CUSHMAN, R. H., "A Very Complete Chip Set Joins the Great Microprocessor Race," *EDN*, Vol. 19, No. 22, November 20, 1974, pp. 87-94.
- ECKHOUSE, R. H., JR., *Minicomputer Systems*, Prentice-Hall, Englewood Cliffs, N.J., 1975.
- GEAR, C. W., *Computer Organization and Programming*, 2nd ed., McGraw-Hill, New York, 1974.

- "Intel 8080 Microcomputer Systems Manual," Intel Corporation, Santa Clara, Ca., 1975, p. 125.
- KORN, G. A., *Minicomputers for Engineers and Scientists*, McGraw-Hill, New York, 1973.
- LEVENTHAL, L. A., "Put Microprocessor Software to Work," *Electronic Design*, Vol. 24, No. 16, August 2, 1976, pp. 58-64.
- "M6800 Microprocessor Applications Manual," Motorola, Inc., Phoenix, Ariz., 1975.
- PEATMAN, J. B., *Microcomputer-Based Design*, McGraw-Hill, New York, 1977.
- WEITZMAN, C., *Minicomputer Systems*, Prentice-Hall, Englewood Cliffs, N.J., 1974.

К главе 4

- BARRON, D. W., *Assemblers and Loaders*, American Elsevier Inc., New York, 1972.
- BLAKESLEE, T. R., *Digital Design with Standard MSI and LSI*, Wiley, New York, 1975.
- BOND, J., "Designer's Guide to Software for the Hardware Designer, Part II," *EDN*, Vol. 20, No. 15, August 5, 1974, pp. 51-56.
- CAMPBELL-KELLY, M., *An Introduction to Macros*, American Elsevier Inc., New York, 1973.
- GEAR, C. W., *Computer Organization and Programming*, 2nd ed., McGraw-Hill, New York, 1974.
- GIBBONS, J., "When to Use Higher-Level Languages in Microcomputer-Based Systems," *Electronics*, Vol. 48, No. 16, August 7, 1975, pp. 107-111.
- Intel 8080 Assembly Language Programming Manual, Intel Corporation, Santa Clara, Ca., 1974.
- KNUTH, D. E., *The Art of Computer Programming: Fundamental Algorithms, Vol. 1*, Addison-Wesley, Reading, Ma., 1967.
- KNUTH, D. E., *The Art of Computer Programming: Seminumerical Algorithms, Vol. 2*, Addison-Wesley, Reading, Ma., 1969.
- Motorola 6800 Programming Manual, Motorola Semiconductor Products Inc., Phoenix, Az., 1974.
- WEITZMAN, C., *Minicomputer Systems*, Prentice-Hall, Englewood Cliffs, N.J., 1974.
- WYLAND, D. C., "Employ μP Software Tools Properly," *Electronic Design*, Vol. 23, No. 26, December 20, 1975, pp. 50-55.

К главе 5

- BLAKESLEE, T. R., *Digital Design With Standard MSI and LSI*, Wiley, New York, 1975.
- CUSHMAN, R. H., "Beware of the Errors that Can Creep Into μP Benchmark Programs," *EDN*, Vol. 20, No. 12, June 20, 1975, p. 105.
- CUSHMAN, R. H., "Exposing the Black Art of Microprocessor Benchmarking," *EDN*, Vol. 20, No. 8, April 20, 1975, p. 41.
- CUSHMAN, R. H., "The Intel 8080: First of the Second Generation Microprocessors," *EDN*, Vol. 19, No. 9, May 5, 1974, p. 30.

- CUSHMAN, R. H., "Microprocessor Benchmarks," *EDN*, Vol. 20, No. 10, May 20, 1975, p. 43.
- ECKHOUSE, R. H., Jr., *Minicomputer Systems*, Prentice-Hall, Englewood Cliffs, N.J., 1975.
- Intel 8080 Assembly Language Programming Manual, Intel Corporation, Santa Clara, Ca., 1974.
- Intel 8080 Microcomputer Systems User's Manual, Intel Corporation, Santa Clara, Ca., July 1975.
- LEVENTHAL, L. A., "Take Advantage of 8080 and 6800 Data-Manipulation Capabilities," *Electronic Design*, Vol. 25, No. 8, April 12, 1977, pp. 90-97.
- LEVENTHAL, L. A., "Cut Your Processor's Computation Time," *Electronic Design*, Vol. 25, No. 17, August 16, 1977, pp. 82-89.
- LEVENTHAL, L. A., *8080A/8085 Assembly Language Programming*, Osborne and Associates, Berkeley, Ca., 1978.
- LEWANDOWSKI, R., "Preparation: The Key to Success with Microprocessors" in *Microprocessors*, McGraw-Hill, New York, 1975, p. 74.
- Motorola 6800 Microprocessor Applications Manual, Motorola Semiconductor Products Inc., Phoenix, Az., 1975.
- Motorola 6800 Programming Manual, Motorola Semiconductor Products Inc., Phoenix, Az., 1975.
- SHIMA M., and F. FAGGIN, "In Switching to N-MOS Microprocessor Gets a Two Microsecond Cycle Time," *Electronics*, Vol. 47, No. 8, April 18, 1974, pp. 95-100.
- TITUS, J., "How to Design a Microprocessor-Based Controller System," *EDN*, Vol. 19, No. 16, August 20, 1974, pp. 49-56.
- WELLER, W. J., et al., *Practical Microcomputer Programming: the Intel 8080*, Northern Technology Books, Evanston, Ill., 1976.

KRITIK 6

- ABRAHAM, P., "Structured Programming Considered Harmful," *SIGPLAN Notices*, Vol. 10, No. 4, April 1975, pp. 13-24.
- CASILLI, G., and W. KIRN, "Microcomputer Software Development," *Digital Design*, Vol. 5, No. 7, July 1975, pp. 50-52.
- CHAPIN, N., *Flowcharts*, Auerbach Publishers Inc., Princeton, N.J., 1971.
- DENNING, P. J., "Is Structured Programming Any Longer the Right Term?," *SIGPLAN Notices*, Vol. 9, No. 11, November 1974, pp. 4-6.
- DOLLHOFF, T., "μP Software. How to Optimize Timing and Memory Usage," *Digital Design*, Vol. 6, No. 11, November 1976, pp. 56-69.
- HEITZEL, W., *Program Test Methods*, Prentice-Hall, Englewood Cliffs, N.J., 1973.
- HUGHES, J. K., and J. I. MIGHTOM, *A Structured Approach to Programming*, Prentice-Hall, Englewood, Cliffs, N.J., 1977.
- KARPINSKI, R. H., "An Unstructured View of Structured Programming," *SIGPLAN Notices*, Vol. 9, No. 3, March 1974, pp. 112-119.

- KNUTH, D. E., "Structured Programming with GO TO Statements," *Computing Surveys*, Vol. 6, No. 4, December 1974, pp. 261-301.
- KRUMMEL, L., and G. SCHULTZ, "Advances in Microcomputer Development Systems," *Computer*, Vol. 10, No. 2, February 1977, pp. 13-19.
- LEVENTHAL, L. A., "Can Structured Programming Help the Bench Programmer?" 1977 IEEE Workshop on Bench Programming of Microprocessors, Philadelphia, Pa., pp. 1-5.
- MCCRACKEN, D. D., "Revolution in Programming: An Overview," *Datamation*, Vol. 19, No. 12, December 1973, pp. 50-52.
- MCGOWAN, C. L., and J. R. KELLY, *Top-Down Structured Programming Techniques*, Petrocelli/Charter, New York, 1975.
- MARTINEZ, R., "A Look at Trends in Microprocessor/Microcomputer Software Systems," *Computer Design*, Vol. 14, No. 6, June 1975, pp. 51-57.
- PARNAS, D. L., "On the Criteria to be Used in Decomposing Systems into Modules," *Communications of the ACM*, Vol. 15, No. 12, December 1972, pp. 1053-1058.
- PARNAS, D. L., "The Influence of Software Structure on Reliability," *SIGPLAN Notices*, Vol. 10, No. 6, June 1975, pp. 358-362.
- REIFER, D. J., "Automated Aids for Reliable Software," *SIGPLAN Notices*, Vol. 10, No. 6, June 1975, pp. 131-142.
- SHNEIDERMAN, B., and P. SCHEUERMANN, "Structured Data Structures," *Communications of the ACM*, Vol. 17, No. 10, October 1974, pp. 566-574.
- SRINI, V. P., "Fault Diagnosis of Microprocessor Systems," *Computer*, Vol. 10, No. 1, January 1977, pp. 60-65.
- YOURDON, E., *Techniques of Program Structure and Design*, Prentice-Hall, Englewood Cliffs, N.J., 1976.

К главе 7

- BLAKESLEE, T. R., *Digital Design with Standard MSI and LSI*, Wiley, New York, 1975.
- Data Manual*, Signetics Co., Menlo Park, Ca., 1976.
- DAVIS, S., "Selection and Application of Semiconductor Memories," *Computer Design*, Vol. 13, No. 1, January 1974, pp. 65-77.
- FARNBACH, W. A., "Bring up Your μ P Bit-by-Bit," *Electronic Design*, Vol. 24, No. 15, July 19, 1976, pp. 80-85.
- FRANKENBERG, R. J., *Designer's Guide to Semiconductor Memories*, Cahners, Boston, Ma., 1975.
- GREENE, R., and D. HOUSE, "Designing with Intel PROMs and ROMs," *Intel Application Note AP-6*, Intel Corporation, Santa Clara, Ca., 1975.
- Intel 8080 Microcomputer Systems User's Manual*, Intel Corporation, Santa Clara, Ca., 1975.
- LEVINE, L., and W. MYERS, "Timing: a Crucial Factor in LSI-MOS Main Memory Design," *Electronics*, Vol. 48, No. 14, July 10, 1975, pp. 107-111.
- LUECKE, G., et al., *Semiconductor Memory Design and Application*, McGraw-Hill, New York, 1973.

Motorola 6800 Microprocessor Applications Manual, Motorola Semiconductor Products Inc., Phoenix, Ariz., 1975.

RAPHAEL, H., "How to Expand a Microcomputer's Memory," *Electronics*, Vol. 49, No. 26, December 23, 1976, pp. 67-69.

Semiconductor Memory Data Book, Texas Instruments Inc., Dallas, Texas, 1975.

SPRINGER, J., "Designers' Guide to Semiconductor Memory Systems," *EDN*, Vol. 19, No. 16, September 5, 1974, pp. 49-56.

The TTL Data Book, Texas Instruments Inc., Dallas, Texas, 1973.

THOMAS, A. T., "Design Techniques for Microprocessor Memory Systems," *Computer Design*, Vol. 14, No. 8, August 1975, pp. 73-78.

К главе 8

ALDRIDGE, D., "Analog-to-Digital Conversion Techniques with the M6800 Microprocessor System," Motorola Application Note AN-757, Motorola Semiconductor Products Inc., Phoenix, Ariz., 1975.

Analog-Digital Conversion Handbook, Analog Devices, Inc., Norwood, Ma., 1972.

BACKLER, J., "A Display Casebook," *Digital Design*, Vol. 6, No. 2, February 1976, pp. 44-48.

BAUNACH, S. C., "An Example of an M6800-based GPIB Interface," *EDN*, Vol. 22, No. 17, September 20, 1977, pp. 125-128.

BLAKESLEE, T. R., *Digital Design with Standard MSI and LSI*, Wiley, New York, 1975.

BURSKY, D., "Focus on Data Converters," *Electronic Design*, Vol. 24, No. 19, September 13, 1976, pp. 68-79.

CONWAY, J., "What You Should Know About the 488 and 583 Interface Standards," *EDN*, Vol. 22, No. 15, August 5, 1976, pp. 49-54.

Data Manual, Signetics Corp., Mountain View, Ca., 1976.

EIA Standard RS-232-C: Interface Between Data Terminal Equipment and Data Communications Equipment Employing Serial Binary Data Interchange, Electronic Industries Association, Washington, D.C., 1969.

ETCHEVERRY, F. W., "Binary Serial Interfaces—Making the Digital Connection," *EDN*, Vol. 22, No. 8, April 20, 1976, pp. 40-43.

FULLAGAR, D., et. al., "Interfacing Data Converters and Microprocessors," *Electronics*, Vol. 49, No. 25, December 9, 1976, pp. 81-89.

Guide to Standard MOS Products, American Microsystems Inc., Santa Clara, Ca., 1975.

HILBURN, J. L., and P. N. JULICH, *Microcomputers / Microprocessors: Hardware, Software, and Applications*, Prentice-Hall, Englewood Cliffs, N.J., 1976.

HOLDERBY, W. S., "Designing a Microprocessor-Based Terminal for Factory Data Collection," *Computer Design*, Vol. 16, No. 3, March 1977, pp. 81-88.

HNATEK, E. R., *A User's Handbook of D/A and A/D Converters*, Wiley, New York, 1975.

Intel 8080 Microcomputer Systems User's Manual, Intel Corporation, Santa Clara, Ca., 1975.

- KUZDRALL, J. A., "Memory, Peripherals Share Microprocessor Address Range," *Electronics*, Vol. 48, No. 24, November 27, 1975, pp. 105-107.
- LAENGRICH, N., "IEEE Std-488/1975—General Purpose Means of Providing Measurement and Stimulus Instrument Communication," *WESCON 1976*, Session 12, Paper 1.
- LARSEN, D. G. et. al., "INWAS: Interfacing with Asynchronous Serial Mode," *IEEE Transactions on Industrial Electronics and Control Instrumentation*, Vol. IECI-24, No. 1, February 1977, pp. 2-12.
- LOGAN, J. D. and P. S. KREAGER, "Using a Microprocessor: A Real-Life Application. Part 1: Hardware," *Computer Design*, Vol 14, No. 9, September 1975, pp. 69-77.
- M6800 Applications Manual*, Motorola Semiconductor Products, Inc., Phoenix, Ariz., 1975.
- M6800 Microcomputer System Design Data*, Motorola Semiconductor Products Inc., Phoenix, Ariz., 1976.
- PEATMAN, J. B., *Microcomputer-Based Design*, McGraw-Hill, New York, 1977.
- PICKLES, G., "Who's Afraid of RS-232?" *Kilobaud*, Vol. 1, No. 5, May 1977, pp. 50-54.
- PSHAENICH, A., "Interface Considerations for Numeric Display Systems," *Motorola Application Note AN-741*, Motorola Semiconductor Products, Inc., Phoenix, Ariz., 1975.
- RONY, P. R. and D. G. LARSEN, *The Bugbook I*, E and L Instruments Inc., Derby, Conn., 1974.
- RONY, P. R. and D. G. LARSEN, *The Bugbook II*, E and L Instruments Inc., Derby, Conn., 1974.
- RONY, P. R. et. al., *The Bugbook III*, E and L Instruments Inc., Derby, Conn., 1975.
- RUNYON, S., "Focus on Keyboards," *Electronic Design*, Vol. 20, No. 23, November 9, 1972, pp. 54-64.
- The TTL Data Book*, Texas Instruments Inc., Dallas, Texas, 1973.
- WAKERLY, J. F. "Microprocessor Input/Output Architecture," *Computer*, Vol. 10, No. 2, February 1977, pp. 26-33.
- WESTBURG, A., JR., "Displays: Today and Tomorrow," *Digital Design*, Vol. 6, No. 4, April 1976, pp. 64-72.

К главе 9

- BALDRIDGE, R. L., "Interrupts Add Power, Complexity to μ C-System Design," *EDN*, Vol. 22, No. 15, Aug. 5, 1977, pp. 67-73.
- BURGETT, K. and E. F. O'NEIL, "An Integral Real-Time Executive for Microcomputers," *Computer Design*, Vol. 16, No. 7, July 1977, pp. 77-82.
- CUSHMAN, R. H., "Let the Interrupts Do Their Work," *EDN*, Vol. 21, No. 11, June 5, 1976, pp. 92-97.
- CUSHMAN, R. H., "Single-Chip Microprocessors Move into the 16-Bit Arena," *EDN*, Vol. 20 No. 4, February 20, 1975, pp. 24-30.
- ECKHOUSE, R. H., Jr., *Minicomputer Systems*, Prentice-Hall, Englewood Cliffs, N.J., 1975.
- F8 Microprocessor User's Manual*, Fairchild Semiconductor, Menlo Park, Ca., 1975.

- FALK, H., "Linking Microprocessors to the Real World," *IEEE Spectrum*, Vol. 11, No. 9, September 1974, pp. 59-67.
- FISHER, E., "Speed Microprocessor Responses," *Electronic Design*, Vol. 23, No. 23, November 8, 1975, pp. 78-83.
- GRANDBOIS, G., "Log Data Under μ P Control," *Electronic Design*, Vol. 24, No. 10, May 10, 1976, pp. 94-101.
- Intel 8080 Microcomputer Systems User's Manual*, Intel Corporation, Santa Clara, Ca., 1975.
- KHANNA, V., and T. DALY, "Making the Most of Your Micro," *Digital Design*, Vol. 5, No. 7, July 1975, pp. 36-49.
- LANE, A., "Microprocessor System Design," *Digital Design*, Vol. 5, No. 8, August 1975, pp. 62-70.
- LEWIN, M. H., "Integrated Microprocessors," *IEEE Transactions on Circuits and Systems*, Vol. CAS-22, No. 7, July 1975, pp. 577-585.
- LOGAN, J. D., and P. S. KREAGER, "Using a Microprocessor: A Real-Life Application. Part I—Hardware," *Computer Design*, Vol. 14, No. 9, September 1975, pp. 69-77.
- MOORE, A., and M. EIDSON, *Printer Control: A Minor Task for a Fast Microprocessor*, Motorola Semiconductor Products, Phoenix, Ariz., 1975.
- OSBORNE, A., *An Introduction to Microcomputers, Vol. I, Basic Concepts*, Adam Osborne and Associates, Berkeley, Ca., 1975.
- OSBORNE, A., *An Introduction to Microcomputers, Vol. II, Some Real Products*, Adam Osborne and Associates, Berkeley, Ca., 1977.
- RIPPS, D. L., "Multitasking Operating Systems Simplify μ C Software Development," *EDN*, Vol. 22, No. 2, January 20, 1977, pp. 67-69.
- The TTL Data Book*, Texas Instruments, Inc., Dallas, Texas, 1973.
- VACROUX, A. G., "Explore Microcomputer I/O Capabilities," *Electronic Design*, Vol. 23, No. 10, May 10, 1975, pp. 114-119.
- WEISBECKER, J., "A Simplified Microcomputer Architecture," *Computer*, Vol. 7, No. 3, March 1974, pp. 41-47.
- WEITZMAN, C., *Minicomputer Systems*, Prentice-Hall, Englewood Cliffs, N.J., 1974.
- WELLER, W. J. et al., *Practical Microcomputer Programming. The Intel 8080*, Northern Technology Books, Evanston, Ill., 1976.
- WYLAND, D. C., "Increase Microcomputer Efficiency," *Electronic Design*, Vol. 23, No. 23, November 8, 1975, pp. 70-75.

ОГЛАВЛЕНИЕ

Предисловие редактора перевода	3
Предисловие	5
Глава первая. Введение в микропроцессоры.	9
1.1. Вычислительная техника и микропроцессоры	9
1.2. Большие и малые ЭВМ	11
1.3. Сравнение типичных ЭВМ	13
1.4. История развития микропроцессоров	24
1.5. Полупроводниковые технологии	27
1.6. Конструктивные характеристики интегральных микро- схем	29
1.7. Полупроводниковая память	32
1.8. Области применения МП	33
Глава вторая. Архитектура микропроцессоров	39
2.1. Общая архитектура ЭВМ	40
2.2. Регистры	47
2.3. Арифметическое устройство	52
2.4. Реализация команд	54
2.5. Стек	56
2.6. Особенности архитектуры микропроцессоров	59
2.7. Примеры архитектуры микропроцессоров	61
2.8. Выводы	61
Глава третья. Системы команд микропроцессоров	65
3.1. Форматы команд	65
3.2. Методы адресации	69
3.3. Типы команд	80
3.4. Системы команд микропроцессоров	95
3.5. Примеры систем команд микропроцессоров	95
Глава четвертая. Микропроцессорные ассемблеры	109
4.1. Сравнение языков различного уровня	110
4.2. Характеристики ассемблеров	117
4.3. Особенности микропроцессорных ассемблеров	133
4.4. Ассемблеры микропроцессоров Intel 8080 и Motorola 6800	134
4.5. Выводы	142
Глава пятая. Программирование на языке ассемблера	142
5.1. Простые программы	144
5.2. Организация циклов и обработка массивов	157
5.3. Вычислительные программы	176
5.4. Обработка символической информации	190
5.5. Подпрограммы	193
5.6. Выводы	203

Глава шестая. Разработка программного обеспечения для микропроцессора	209
6.1. Задачи разработки программного обеспечения	209
6.2. Постановка задачи	214
6.3. Конструирование программы	218
6.4. Кодирование	228
6.5. Отладка	232
6.6. Тестирование	239
6.7. Документирование	242
6.8. Перепроектирование	245
6.9. Системы разработки ПО	247
6.10. Выводы	256
Глава седьмая. Модули памяти микропроцессоров	257
7.1. Общие характеристики интерфейса памяти	257
7.2. Простые модули памяти	261
7.3. Структуры шин	276
7.4. Проектирование модулей памяти с тремя состояниями	287
7.5. Модули памяти для конкретных микропроцессоров	290
7.6. Выводы	305
Глава восьмая. Организация ввода-вывода в микропроцессорных системах	305
8.1. Основные проблемы, возникающие при организации ввода-вывода	305
8.2. Простые подсистемы ввода-вывода	312
8.3. Более сложные подсистемы ввода-вывода	317
8.4. Аппаратные средства, применяемые при вводе-выводе	324
8.5. Устройства ввода-вывода	340
8.6. Разработка подсистем ввода-вывода для конкретных микропроцессоров	362
8.7. Выводы	394
Глава девятая. Прерывания в микропроцессорных системах	394
9.1. Преимущества и недостатки прерываний	395
9.2. Особенности систем прерываний	400
9.3. Простые системы прерываний	413
9.4. Системы прерываний конкретных микропроцессоров	424
9.5. Прямой доступ к памяти	452
9.6. Выводы	455
Список литературы	456

2р.70к.